ImagePilot 2.0, A Drawing Interpretation Tool for the Sight-impaired

By
Farzad M. Valad

Thesis submitted to the Faculty of Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Master of Science
In
Computer Engineering

Dr. Lynn Abbott, Chairman
Dr. Morton Nadler
Dr. Daniel Schmoldt
Dr. Richard Conners

February 18, 1999
Blacksburg, Virginia

Keywords: ImagePilot, Drawing, Sight-impaired, Blind

IMAGEPILOT 2.0, A DRAWING INTERPRETATION TOOL FOR THE SIGHT-IMPAIRED

Farzad M. Valad

Abstract

This thesis describes the design and implementation of an innovative drawing interpretation tool for the sight-impaired. The move towards Graphical User Interfaces in today's computer era presents many challenges to those with impaired vision. Although there are many tools to aid this group in reading and writing text-based electronic documents, few software packages are available to help the sight-impaired interpret electronic images. This new tool, known as ImagePilot 2.0, processes electronic image files that contain line drawings and produces audio feedback to guide the user through the drawing. ImagePilot 2.0 receives input through a pointing device, thereby providing the user with a means of examining a drawing interactively, and serving as an aid for recognizing the outlines of familiar shapes and objects.

In this study, a "line drawing" is a simple image that contains line segments and curves, without any shading or colors. It can be represented as a 2-dimensional array of picture elements (pixels), in which each pixel is either black or white. The drawing is represented by a pattern of black (foreground) pixels against a white background. Typically, groups of connected foreground pixels represent segments or curves that are associated with a single object. ImagePilot 2.0 makes it possible for a sight-impaired user to interpret such an image.

Line drawings can be stored in many different electronic formats. ImagePilot 2.0 supports valid Graphics Interchange Format (GIF) files. If the foreground image regions in the file are wider than one pixel in width, a Zhang-Suen thinning algorithm is applied to thin the drawing. The tool identifies the separate regions in the drawing and decides on the best starting point for each region. Once a starting point is chosen, the drawing is processed using a modified chain-coding algorithm.

The audio feedback consists of two types of audio cues, verbal and tone, along with stereo playback. Verbal feedback guides the user with a set of verbal cues played through the speakers. The tone feedback uses three tones representing above, level, and below the horizontal. The left and right speakers provide left and right directional information at each level.

Speed is a critical factor in image analysis and interpretation applications. While the tool is receiving and processing input, it must also respond to the user within an acceptable amount of time. ImagePilot 2.0 uses multi-threading and multi-tasking techniques to achieve higher performance speeds. After design and implementation, two groups of people tested the tool. The tool demonstrated the ability to help the user find and trace the segments in the test drawings with high efficiency and acceptable response time.

This tool is written in pure Java, and complies with Sun's Java API specification 1.1.7 released in October 1998. The tool functions on systems with multimedia capabilities that have a Java Virtual Machine (JVM) and Java Media Framework (JMF) installed.

ACKNOWLEDGMENTS

I would like to thank Dr. Lynn Abbott, my committee chair, for his advice and guidance, and for giving me the opportunity to work on this project. I also thank my committee members, Dr. Morton Nadler, Dr. Richard Conners, and Dr. Daniel Schmoldt for their teachings throughout my student years and their support and guidance for this project. I would also like to thank Peter Vazquez for his help and support with the undergraduate study that led to this thesis. In addition, I thank the volunteers who helped test the tool.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. Introduction

1.1 Overview

The move towards graphical user interfaces is widely regarded as an advance in human-computer interaction. However, the abandonment of old-fashioned text-based interfaces presents new challenges to those computer users who are sight-impaired [4]. Much work has been done to aid such users with the advanced computers of today. While most products aid the sight-impaired in text-related issues, such as reading and writing, not much has been done to help them interpret electronic images.

This thesis introduces a tool known as ImagePilot 2.0 to help sight-impaired users examine and recognize electronic images. ImagePilot 2.0 is the first complete image interpretation tool developed for the sight-impaired. The goal of the tool is to allow a sight-impaired user to trace and interpret a digital image using a tablet and stylus, something that was not possible before. Although some verbal feedback is used, speech synthesis techniques are not yet mature enough to provide a desired variety of verbal responses. Therefore, audio tones are used in many cases to provide directional information. Despite the fact that speech synthesis and image analysis are ongoing research areas, ImagePilot 2.0 defines and demonstrates techniques for presenting electronic images to the sight-impaired.

In this thesis, a “line drawing” refers to a simple image that contains line segments and curves, without any shading or colors. The content of the drawing is contained in the pattern of black (foreground) pixels against a white background. Typically, groups of connected foreground pixels represent segments or curves that are associated with a single object as shown in Figure 1.

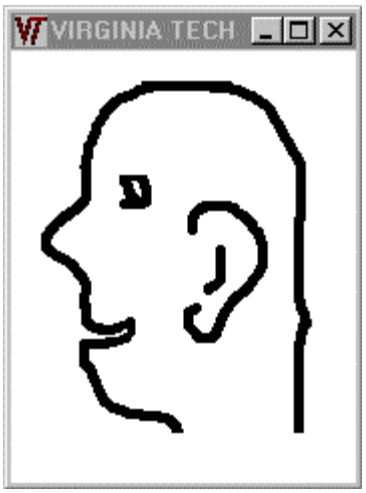

Figure 1: A line drawing that consists of a human profile as an example of a simple but interesting test image. Some sight-impaired individuals can recognize the contents of pencil drawings such as this through tactile sensing. ImagePilot 2.0 makes it possible to interpret drawings such as this without the aid of tactile cues.

A sight-impaired person develops and uses senses other than vision for interacting with the world around them. In a study of perception by Kennedy [1], sight-impaired individuals demonstrated the ability to trace and recognize the contents of a pencil drawing by touch, feeling the impression left on the paper by the pencil. Kennedy also showed that sight-impaired people rely on their imagination and sense of touch to render pictures of familiar objects.

The goal of ImagePilot 2.0 is to facilitate the same kind of recognition for images that are stored in digital form. The user uses a tablet with stylus to trace the foreground pixels of a digital image that has been processed by the program. While the user traces the foreground pixels, the program provides audio cues as to the position of the next pixel. Therefore, the program guides the user through the drawing. The program reads the location of the stylus and provides acoustic feedback based on the location of the next pixel. The acoustic feedback is a set of stereo tones assigned to each of the next possible directions. The user can choose a set of verbal cues to serve as the acoustic feedback instead.

The physical interaction of the user with the tablet provides a sense of the viewing area. If a tablet and stylus are unavailable, the user can use a computer mouse. Using either pointing device, approaching a region and tracing it are distinguished by two different cues. By default, verbal tones are used to guide the user to the beginning of a region and audio tones are used for tracing that region. The program is also capable of operating in verbal mode only.

1.2 Problem Statement

There are many tools available to the sight-impaired user for interacting with images. When it comes to tracing images, many tools are designed to complement the sense of touch. Some tools are designed to provide audio feedback, but in most cases the audio feedback is very general and does not provide the user with detailed information about specific parts of an image. Some tools even use sonar like audio feedback to help the user move about in real life. The goal of this research is to develop a working prototype of a drawing interpretation tool for the sight-impaired users. This tool will use audio feedback to guide the sight-impaired user in tracing meaningful portions of an image.

1.3 Research Contributions

This research has resulted in a completely new aid for the sight-impaired. Two preliminary versions of ImagePilot 2.0, known as TraceCad [2] and ImagePilot 1.0 [3], have led to the first complete prototype of its kind. The main contributions of this work are as follows:

- ImagePilot 2.0 is a novel tool that can aid sight-impaired users in the interpretation of digital line drawings.
- A novel technique for providing audio feedback was developed.
- Practical considerations required more efficient and faster image interpretation techniques than in the two previous versions.
- Reducing system requirements needed more efficient and better data structures then the two previous versions.

- Several well-known techniques, such as chain coding and the Zhang-Suen thinning algorithm, were modified and ported to Java for use in ImagePilot 2.0.
- Unlike the two previous versions, ImagePilot 2.0 is available as a complete easy-to-install software package.

1.4 Thesis Organization

This thesis is divided into seven chapters and two appendices. Chapter 1 has presented an introduction, describing a general overview and new ideas used in this research. Chapter 2 provides background and a discussion of previous research. Chapter 3 presents the overall design, the data structures used for storage and retrieval of information, the data processing techniques, and the unique audio feedback format used in this research. Chapter 4 describes the runtime operation and results of this research. Chapter 5 presents the results of testing the tool with human subjects. Chapter 6 presents concluding remarks. The Bibliography lists all the references used for conducting this research and writing this thesis.

In addition to the above, the thesis contains two appendices. Appendix A describes system requirements and installation procedures. Appendix B provides the digital “Readme” file and all the Java source code for the program.

2. Background

2.1 Motivation and Early Work

Although most of us draw what we see, is it possible for a sight-impaired person to draw or have any interest in drawing? Most people have assumed that sight-impaired people have little interest or talent in drawing, or in interpreting drawings through touch or sound. However, Kennedy has shown that sight-impaired people use many of the same methods as sighted individuals to sketch their surroundings [1]. In addition,

> "… we have learned that blind and sighted people share a form of pictorial shorthand. That is, they adopt many of the same devices in sketching their surroundings: for example, both groups use lines to represent the edges of surfaces. Both employ foreshortened shapes and converging lines to convey depth. Both typically portray scenes from a single vantage point. Both render extended or irregular lines to connote motion. And both use shapes that are symbolic, though not always visually correct, such as a heart or a star, to relay abstract messages. [1]"

Inspired by these findings, the author began his quest to develop a software package that sight-impaired computer users could use to interpret electronic images. ImagePilot 2.0 is the third generation of drawing interpretation software for the sight-impaired. The first version to guide the user through a drawing with audio feedback was aptly named TraceCad [2]. A description of it placed third in a student paper contest hosted by the Virginia Mountain Section of the IEEE. The next generation was named ImagePilot 1.0 [3]. The same 2-person team was responsible for both TraceCad and ImagePilot 1.0. Several years prior to TraceCad, one attempt was made at Virginia Tech to develop a drawing interpretation tool. Some progress was made in a Mac environment, but no presentable results are available [13].

TraceCad used a preprocessed array that contained two-dimensional distance and type information about image regions that can be reached from any background pixel. This

scheme enabled a user to find a region using a hill-climbing method. The audio feedback became louder as the user moved closer to a region. The flaw in this scheme was that while it helped the user to find different regions, there was no guidance for the users to trace any particular region after it was found. In theory, the user could make mouse movements such that they continue to stay on the region, or work their way back to that region. However, in practice, that proved to be a very difficult and challenging method for the user.

ImagePilot 1.0, on the other hand, used chain-coding techniques to embed directional information into the foreground pixels. The advantage of this method was that after a user reached a region, he or she received directional guidance to trace that region. This scheme proved more effective than the scheme used in TraceCad. However, it still lacked many fundamental features. For example, an input image had to be formatted manually prior to use by the program. In addition, the users did not have any way of learning the different audio feedback responses in advance. The most important feature ImagePilot 1.0 lacked was the ability to guide the user back to the last tracing point, the point where the user's pointing device drifted off a region.

2.2 Other Computer Aids

This research is not the only work done to develop new technology helping the sight-impaired users interact with the world. Other programs such as TDraw, Giving Blind People Access to Graphics, and the vOICe, are examples of innovative methods developed to aid sight-impaired users.

TDraw is a computer-based tactile drawing tool for sight-impaired users, allowing them to draw pictures [14]. The system is created from swell-paper and a special pen. Swell-paper is placed on a digitizer tablet, and the user presses on the paper while drawing. This provides the user with tactile information as the digital drawing is being created. Input is made via a speech-recognition program to simulate keyboard input. The main program controlling the system is called TDraw (for Tactile Draw). Each object and its given name are immediately catalogued by the program. As the user draws the

computer records the object and it is labeled with a name when the user speaks it. In a study conducted by Kurze [14] using TDraw, the volunteer subjects were able to demonstrate their ability to draw 3D objects as shown in Figure 2 and 3. The truck toy in Figure 2 was a 3D-drawing model for the sight-impaired users. Figure 3 shows two sketches of the truck toy from two different users. The loading platform of the truck is clearly distinguishable from the driver's cab.

Figure 2: A three dimensional object used to better understand the mental model sight-impaired users create of the objects around them.

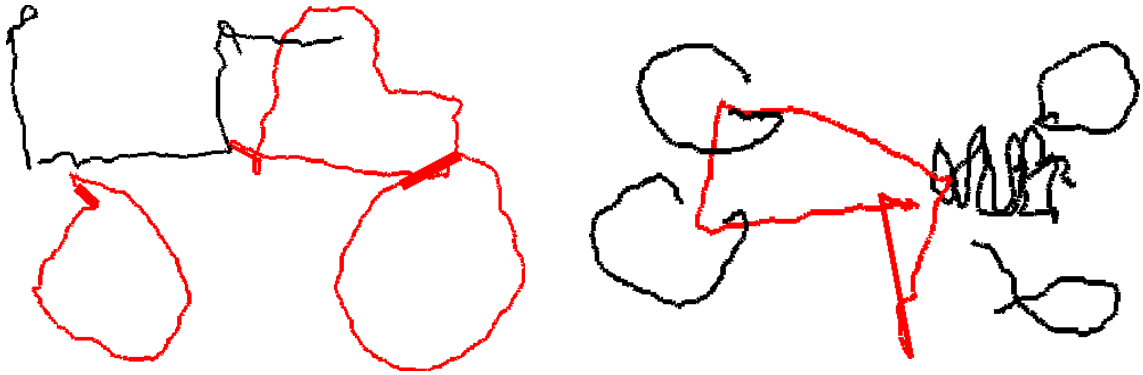

Figure 3: Two sketches of the truck toy shown above, drawn by different sight-impaired individuals. Open and closed parts have been drawn differently. The loading platform of the truck is clearly distinguishable from the driver's cab.

Although TDraw is an aid for drawing by sight-impaired users, it is not designed to help them interpret images. The TDraw study demonstrated that sight-impaired "…people have a spatial concept (a mental model) of the real world which is nearly the same as that of sighted people [14]."

Another innovative approach is called "Giving Blind People Access to Graphics. [16]" This study presented a two-phase approach in giving sight-impaired users access to graphical information. In phase one, a sighted person (the moderator) uses a pointing device to trace the image regions manually. The moderator also annotates each region with a description and size (scale) information. The moderator takes into account the idea presented by the graphic and the available interaction facilities to the sight-impaired person. "… In phase two, the sight-impaired person is enabled to interact, explore, and experience the graphics, much in the same way a sighted viewer would…[16]" Figure 4 represents the graphical model of the two-phase approach.

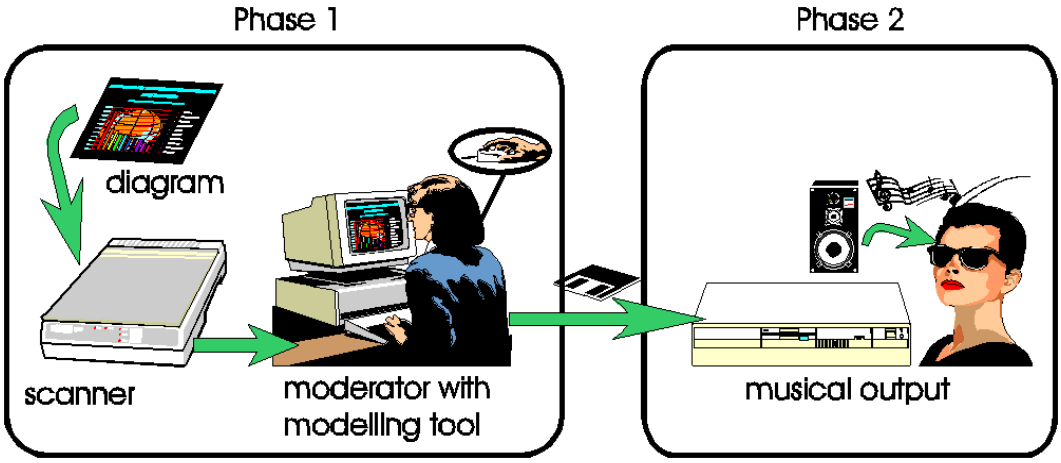

Figure 4: The graphical model of the two-phase approach. A moderator (sighted user) produces an augmented model for several sight-impaired users to use independently.

A scenario that draws on this approach is called wing-based description. A non-visual description of the scene is created using the concept of wings. Then the user can ask the system questions about the content, background, and foreground of the scene. They also can ask questions about the positions of certain objects or about objects in certain positions. Figure 5 presents an example image.

Figure 5: A photograph of a scene to be described using the wing-based scene description.

The moderator can use a pointing device to isolate different wings in the scene. Figure 6 shows the results of the modeling process done by the moderator. Each wing has a name assigned to briefly explain the contents of that wing. This step can be performed recursively to generate a hierarchical wing-based scene with finer-grained descriptions. For example, the family wing can be split into four wings for the people and four wings for the bicycles.

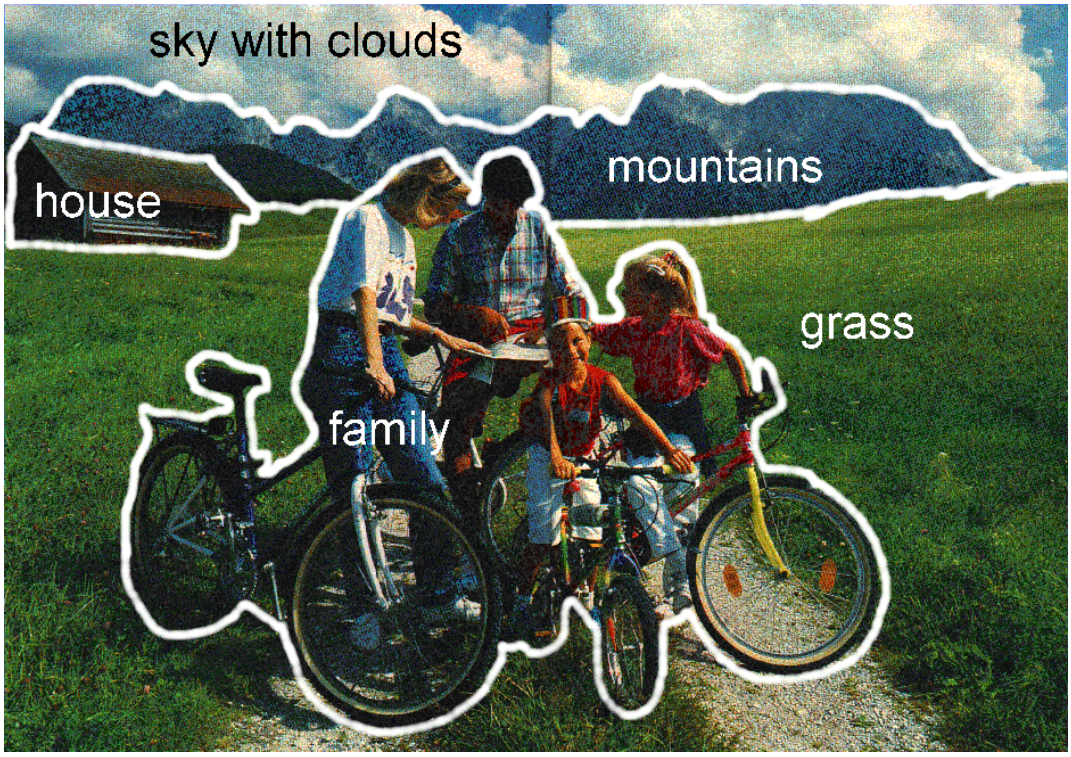

Figure 6: This image shows the results of completing the modelling phase by the moderator. Each wing can further be subdivided for finer-grained descriptions.

The next step in the modeling process is to assign a scale to each wing. The primary result of this step is to create a three-dimensional model that consists of two-dimensional wings. Figure 7 shows the result of adding scale to the wings.

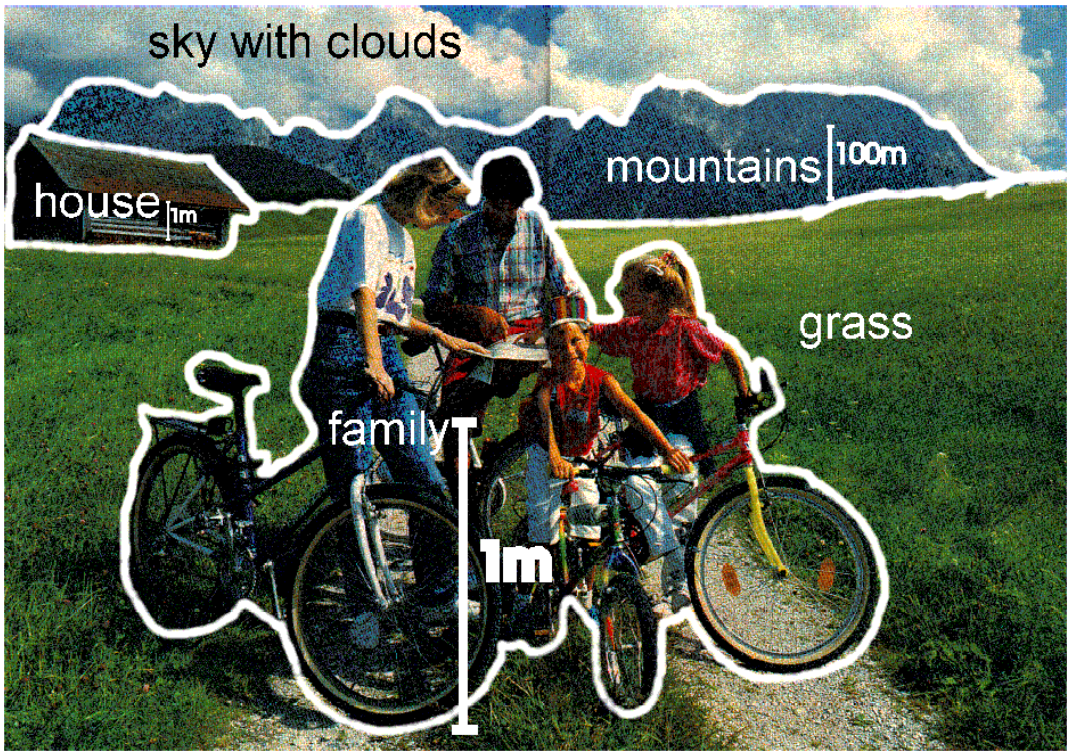

Figure 7: The final scene with split wings and different scales after phase one is completed. The scale adds distance information to form a three-dimensional model from the two-dimensional wings.

As shown in the figures above, the wing model is used to generate an abstract representation of a scene for interactive exploration by sight-impaired users. Sight-impaired users can now explore the scene interactively in ways appropriate to their needs. Although this scheme uses audio feedback to convey information about the image to the users, it still requires the modeling step by a sighted moderator.

Another innovative process is employed by a system called “the vOICe [15]” developed by Peter Meijer. The main part of the system is a video camera that captures pictures that are converted into a digitized image. The size of each digitized image is 64 by 64 pixels. Each image is converted into sounds using a computer, following two simple rules. The location of the pixel determines the tone, and the brightness determines the volume of it. Therefore, pixels situated “high” in the image are converted into high tones and the “low” ones are converted into low tones. Secondly, the brighter the pixel, the

louder the sound. For example, a bright pixel near the top of the image is converted into a high-pitched and loud sound.

Meijer's device doesn't play the entire image at once. Instead, the device plays a column at a time from left to right. "A bright, diagonal line stretching upward to the right produces a loud ooiieep sound and another stretching downward to the right makes the opposite sound – eeiioop. [15]" A complete left to right scan of a single image takes about a second. If the image changes, so will the next pattern of sound generated for the next image.

"The vOICe" offers a new auditory visualization approach as a solution to the orientation and mobility problem for the sight-impaired, an auditory display device that converts arbitrary images into sound as shown in Figure 8.

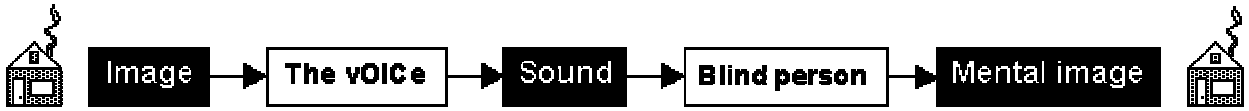

Figure 8: This shows the sensory substitution model by hearing sonified pictures.

Meijer performed spectral analysis to prove that "the vOICe" system preserved enough information from the original image to help a sight-impaired user. Figure 9 shows an input image to the system.

Figure 9: This visual reconstruction proves that much of the original image is preserved in sound generated by the vOICe. [15]"

Using spectrographic analysis, the resulting sound from the vOICe is mapped back to an image as shown in Figure 10.

Figure 10: A photograph of a parked car that will be mapped into sound, and then mapping the resulting sound back into an image using spectrographic analysis.

The converted image shows that the vOICe system preserved enough of the image information to be recognizable. Otherwise, it would have been impossible to produce a recognizable spectrogram. Although Meijer's tool is an innovative method to help sight-impaired users interact with the world around them, the user can only listen to the image without any interaction.

2.3 Summary

TDraw, Giving Blind People Access to Graphics, and the vOICe, provide innovative solutions to overcome the challenges sight-impaired users are faced with everyday. Neither solution provides a means for the user to interpret images. Image Pilot 2.0 provides a unique solution to help sight-impaired users recognize digital images that is different from other techniques currently available.

3. Design

3.1 Overview

ImagePilot 2.0 attempts to provide a solution to a complex problem. It takes advantage of new technologies available to accomplish this task. It is written in pure Java, making it possible in the future for the program to run under any environment. ImagePilot 2.0 is developed in modules and stages using the object oriented programming characteristics of Java. Figure 11 shows the high level design and module interaction of the program.

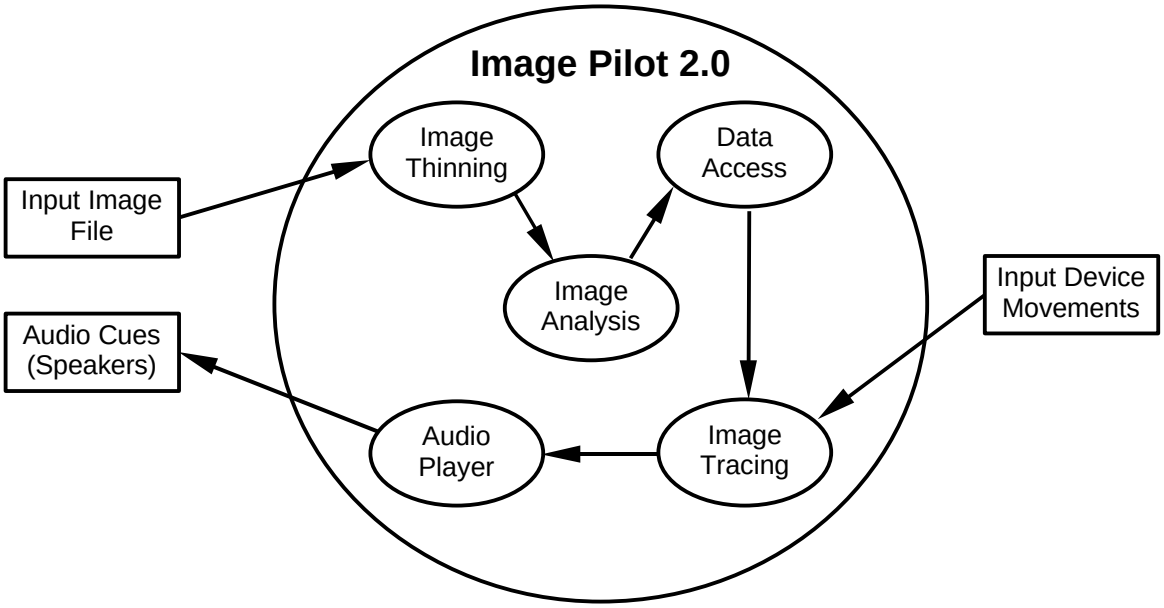

Figure 11: ImagePilot 2.0 object diagram presenting the different modules of processing and the input and output of the program.

Image files must be in GIF format, with the image containing only black and white values (no shades of gray). ImagePilot 2.0 does not impose any size limits on the image. Every time an input image file is opened it is analyzed and processed. First the image is thinned in the Image Thinning module, then it is analyzed and the different parts are labeled by the Image Analysis module. After the image is analyzed, each pixel is coded with tracing information and stored in the Data Access module. At this point, the system is ready to receive user input. The Image Tracing module intercepts the input device movements and retrieves the correct audio cue index from the Data Access module. The audio cue index is passed on to the Audio Player module, which in

turn plays the correct audio tone through the speakers. The next sections describe in more detail how each of the modules work and interact with the user to provide an efficient, fast, and reliable tool for the sight-impaired.

3.2 Data Structures

The goal of the program is to guide a user in tracing meaningful portions of an image. Before ImagePilot 2.0 is able to provide guidance, it must identify the basic elements that compose the image. Every image is assumed to contain one or more connected foreground regions. After thinning, these regions may consist of portions that are straight, curved, closed, and/or open. The first step in understanding the image is to determine the set of constituent parts called segments, that make up each region.

Data structures are the building blocks of any software application. Good sets of data structures greatly effect the efficiency and speed of an application. Determining the storage and processing needs of an application are the key to developing the data structure set. In order to achieve the task some fundamental data structures must be defined. The following list describes the definitions and data structure set needed in this work to identify and store fundamental building blocks of an image.

- Pixel – A single element of an input image that contains a value of 0 or 1.
- Image – A rectangular set of pixels.
- 4-neighbors – Two pixels that are positioned vertically or horizontally with respect to each other.
- 8-neighbors – Two pixels that are positioned vertically, horizontally or diagonally with respect to each other.
- Disjoint neighbors – Two neighboring pixels that are not 4-neighbors of each other.
- Intersection – A foreground pixel that has three or more 8-neighbor foreground pixels, such that each neighbor is disjoint with respect to the other neighbors.
- Endpoint – A foreground pixel that has only one foreground 8-neighbor.
- Digital curve – A set of 8-connected foreground pixels.

- Starting pixel – A pixel that marks the beginning of a digital curve. This pixel can be either an Intersection or Endpoint.
- Ending pixel – A pixel that marks the end of a digital curve. This pixel can be either an Intersection or Endpoint.
- Segment – A digital curve that has one starting and one ending pixel, with zero or more regular pixels connecting the start and end. No pixels in between the starting and ending pixels can be of type intersection or endpoint. In addition, the minimum length of segment by definition is two pixels.
- Closed segment – A segment for which one pixel serves as both the starting pixel and the ending pixel. If a digital curve doesn't contain any endpoints or intersections, then ImagePilot 2.0 automatically splits the curve from the first pixel of the curve it encounters, which is the top left pixel. By definition, a circle, a square, or any other closed shapes by themselves are considered closed, and ImagePilot 2.0 automatically designates the top left pixel as both the starting and ending pixel for those shapes.
- Open segment – A segment that is not closed.
- Region – A region consists of one or more connected segments. The segments may only connect to each other at their starting or ending pixel or both.
- Segment table – A table that lists all the segments contained in a region without any duplicate entries. The entries are not stored in any particular order.

Figure 12 illustrates these definitions using an image that is composed of three separate regions. Each region is composed of segments that have endpoints and/or intersections. Some segments are closed and some are open.

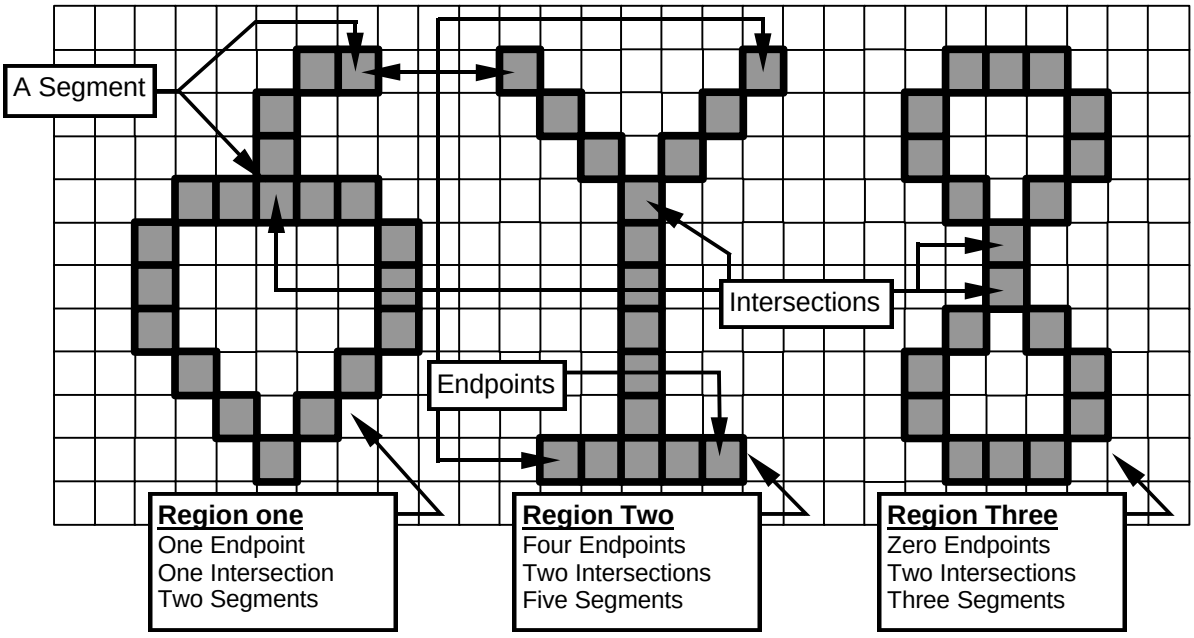

Figure 12: An example of an electronic image illustrating the data structures used in ImagePilot. Region one and three contain closed curves that have overlapping start and end points. In such situations, ImagePilot 2.0 stores a closed curve as a segment that has the same coordinates for the starting and ending point.

The task for ImagePilot is to identify the regions in an image, decompose each region into its constituent segments, and aid the user in tracing these regions.

3.3 Data Processing

3.3.1 Overview

The most time-intensive and important part of ImagePilot 2.0 is data processing. Figure 13 presents the different stages of processing and preparing an input image.

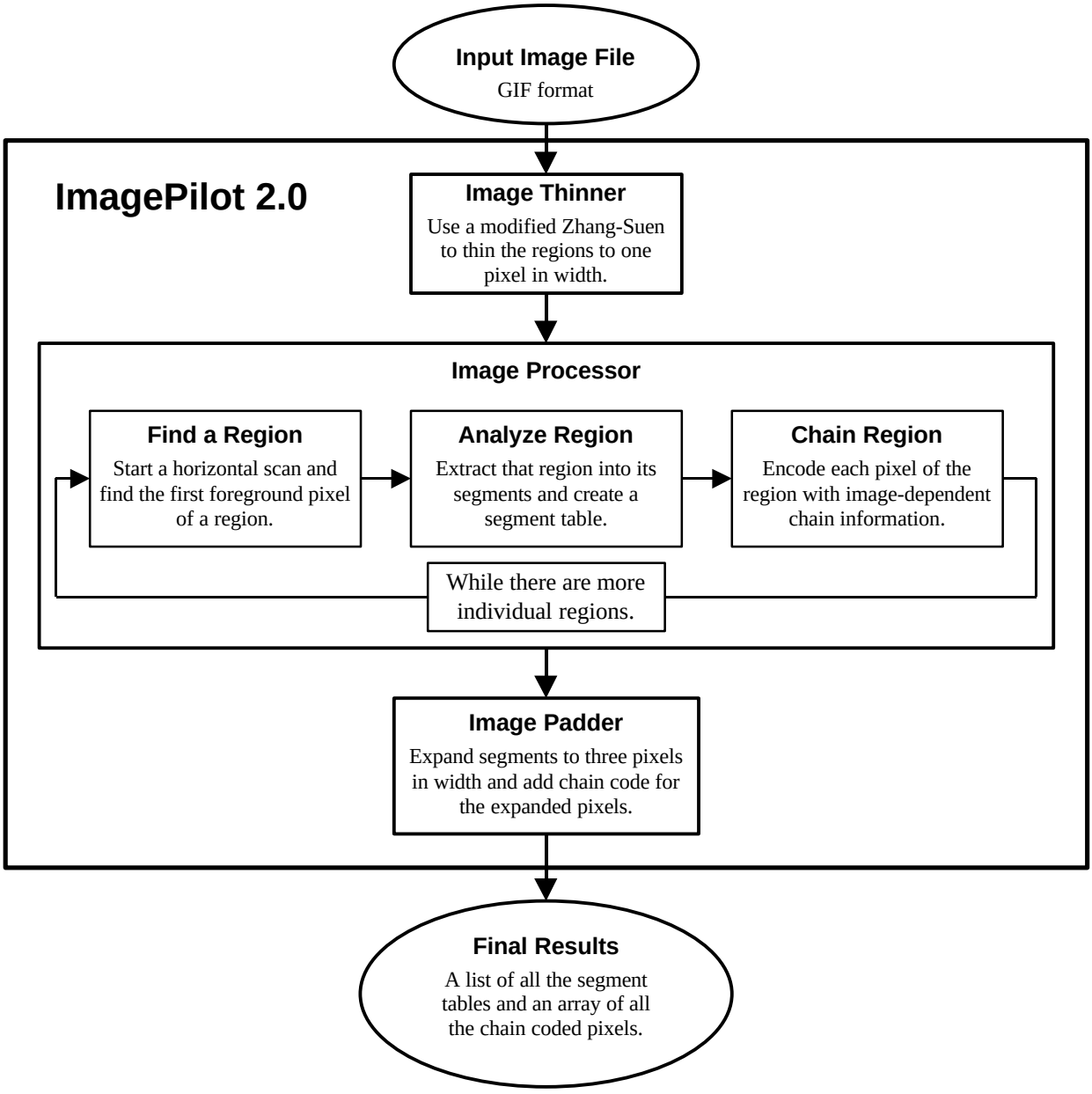

Figure 13: A high level flow chart of the data processing stages needed to process an electronic image. The final result is a list of all the segment tables and an array of all the chain coded pixels.

Regardless if an image has been processed by ImagePilot 2.0 before, every time an image is loaded it is processed to ensure capturing all new changes since the last time it was opened. At this stage, image-dependent audio cues for the user are determined

and stored. The regions in the image are processed and decomposed into segment tables. ImagePilot 2.0 uses multi-threading techniques to expedite the processing.

ImagePilot 2.0 is designed to process GIF files of any size that represent line drawings. Image analysis is performed in three independent stages: region thinning, region coding, and padding. ImagePilot 2.0 takes advantage of multi-threading techniques to thin, code, and pad each region, which allows faster processing of the image than a sequential method would.

The first stage applies the Zhang-Suen thinning algorithm to the input image [8]. In the second stage each region is identified, decomposed into segments, and stored into a segment table. Then, a chain code representation is created for each segment of a region. The second stage continues until all the separate regions are identified and processed.

Thinning the image helps produce a skeleton for faster and easier analysis. A side effect of the thinning process is that the image becomes difficult to trace. Therefore, the final stage pads the segments from one pixel width to three pixel width for easier tracing. The padded pixels are not updated in the segment table, they are only stored in an array that will be overlayed over the thinned image as the final step prior to allowing the user to trace.

3.3.2 Thinning

ImagePilot 2.0 is designed to process one-pixel width line drawings. Since a given line drawing may not satisfy this requirement, a thinning algorithm is required to ensure data integrity prior to processing. A skeleton of the original image is much easier to analyze.

There are many known thinning algorithms in use today. For this application, the efficiency of the thinning algorithm is not as important as its accuracy. Among the many known thinning algorithms (for example, Rutovitz [5], Deutsch [5], Hilditch [5], Arcelli [5], Steffanelli and Rosenfeld [5,12], Rosenfeld [5], Davies and Plummer [5,6], Zhang and Suen 1984 [5,7]), the Zhang-Suen thinning algorithm [5,7] provided adequate performance and accuracy for this work. ImagePilot 2.0 adopted and ported a previously implemented version of the Zhang-Suen thinning algorithm [8]. Figure 14 shows a simple input file that is used to demonstrate how ImagePilot 2.0 processes an input image into traceable data.

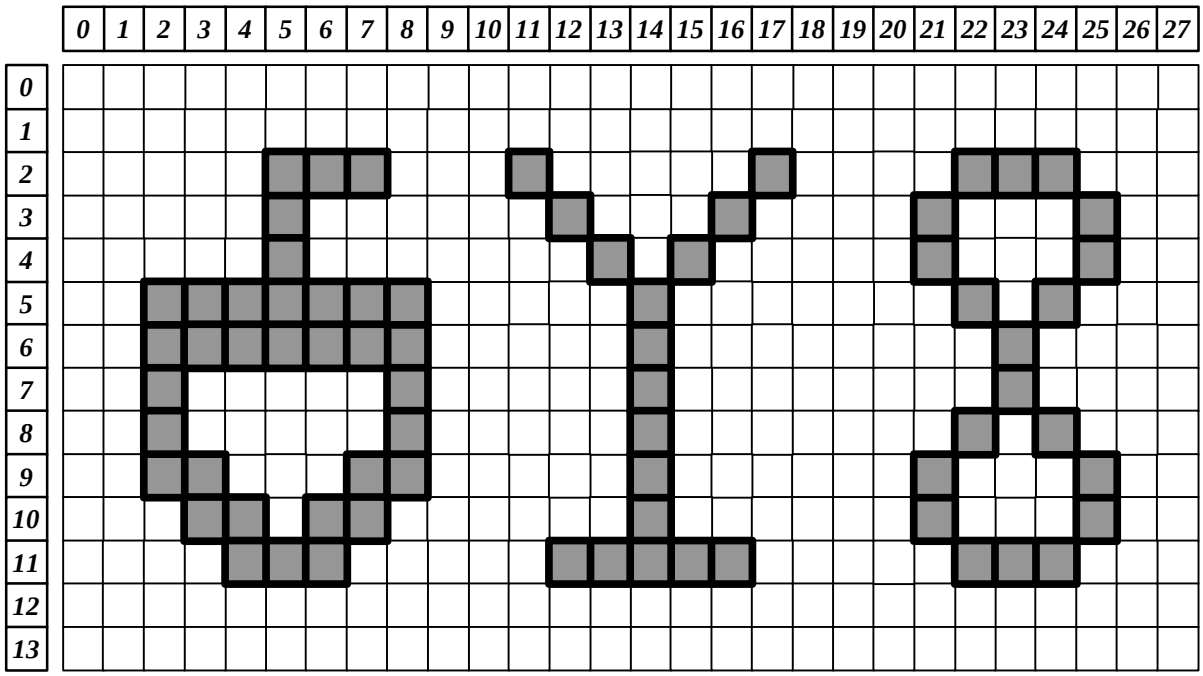

Figure 14: A simple input image used to demonstrate how ImagePilot 2.0 produces traceable data.

Figure 15 illustrates the results of applying the Zhang-Suen thinning algorithm to the raw input image that is converted into one-pixel width regions for processing.

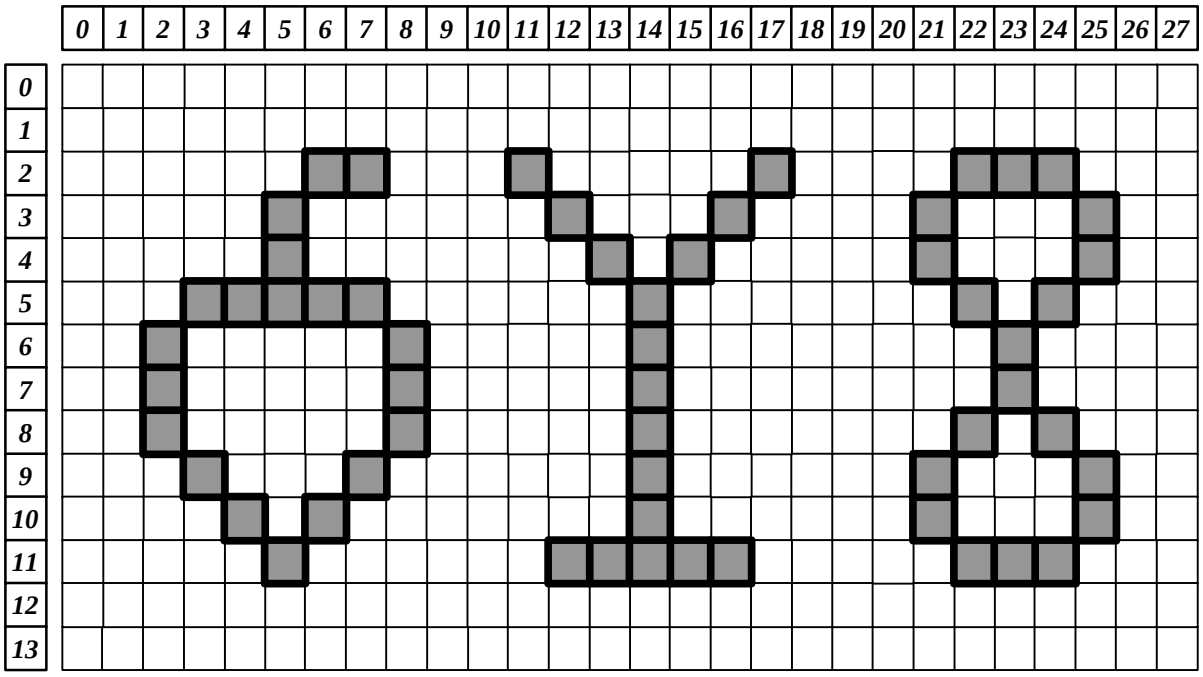

Figure 15: Shows the result of applying the Zhang-Suen thinning algorithm to the sample image in Figure 14.

3.3.3 Analysis

3.3.3.1 Segmentation

A convention must be selected to assist in encoding directional information. ImagePilot 2.0 uses a combination of 8-connectedness and 4-connectedness schemes to process images. 8-connectedness is used to determine disjoint foreground pixels and store chain-coding data. 4-connectedness is used to determine disjoint foreground neighbors around any given foreground pixel. The adopted convention for 8-connectedness is shown in Figure 16, assigning a unique number to each of the eight directions.

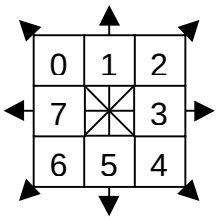

Figure 16: Shows the numbering assignment for chain coding segments. For example a north movement is represented with the number 1 and a south-west movement with the number 6.

The simple input image (Figure 15) used for this demonstration consists of three regions, two of which have closed segments. Processing a line drawing involves finding all the regions in the drawing and encoding traceable information in each pixel. A horizontal scan starting from the top left corner of the image is used to find all the regions. This scan continues until the first foreground pixel of a region is encountered; then the labeling phase starts. The labeling phase stores the direction codes of all the foreground neighbors for each pixel using the directional coding assignment of Figure 16. For the image of Figure 15, the result is shown in Figure 17.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
0																												
1																												
2							63	7				4						6					63	73	74			
3						52							40				62					52				50		
4						51								40		62						41				61		
5				63	73	731	73	74							520								40		62			
6			52						50						51									520				
7			51						51						51									641				
8			41						61						51								62		40			
9				40				62							51							52				50		
10					40		62								51							41				61		
11						20							3	73	731	73	7						30	73	72			
12																												
13																												

Figure 17: This figure shows the stored neighbor direction codes of each pixel. For example "53" represents neighbors in two different directions, 5 and 3. "731" represents neighbors in three different directions, 7, 3, and 1.

Concurrent with loading the thinned image, each region is decomposed into segments and recorded into a segment table as illustrated in Tables 1, 2, and 3. The horizontal and vertical labeling in Figure 17 are used to determine the exact location of a pixel. Each pair (x,y) represents x pixels in the horizontal direction and y pixels in the vertical direction from the top-left pixel in the image. The top-left corner of the image is at coordinate (0,0).

Table 1: This table lists the segments in region one, which is composed of two segments. The pairs represent the location of each pixel in the image. The horizontal and vertical labels on the edge of Figure 17 represent the coordinate system. For example, pixel (7,2) represents a pixel located 7 pixels horizontally and 2 pixels vertically from the top left corner of the image.

	Start Pixel	**End Pixel**	**Start Type**	**End Type**	**Start Direction**	**End Direction**	**Segment Length**
Segment 1	(7,2)	(5,5)	EndPoint	Intersection	7	1	5
Segment 2	(5,5)	(5,5)	Intersection	Intersection	3	7	16

Table 2: This table lists the segments in region two, which is composed of five segments.

	Start Pixel	**End Pixel**	**Start Type**	**End Type**	**Start Direction**	**End Direction**	**Segment Length**
Segment 1	(11,2)	(14,5)	EndPoint	Intersection	4	0	4
Segment 2	(14,5)	(17,2)	Intersection	EndPoint	2	6	4
Segment 3	(14,5)	(14,11)	Intersection	Intersection	5	1	7
Segment 4	(14,11)	(16,11)	Intersection	EndPoint	3	7	3
Segment 5	(14,11)	(12,11)	Intersection	EndPoint	7	3	3

Table 3: This table lists the segments in region three, which is composed of three segments.

	Start Pixel	**End Pixel**	**Start Type**	**End Type**	**Start Direction**	**End Direction**	**Segment Length**
Segment 1	(23,6)	(23,6)	Intersection	Intersection	0	2	10
Segment 2	(23,6)	(23,7)	Intersection	Intersection	5	1	2
Segment 3	(23,7)	(23,7)	Intersection	Intersection	4	6	10

Prior to chain coding, a starting point is selected for each region in the drawing. The starting point is the pixel where the user is guided to first. ImagePilot 2.0 prefers an endpoint of the longest segment in each region as the starting point. Segments that have at least one endpoint are preferred over segments without endpoints. If there are no segments with an endpoint then an intersection is selected as a starting point. In addition, in case there are several segments of similar ending types and same length, the first segment that was processed is selected as the starting segment. The final scenario is a case where there are no endpoints or intersections. In this case, the first pixel encountered is selected as the starting point. The pseudocode below shows the selection process for the starting segment. As new segments are found, the algorithm is invoked to determine if the new segment has a better starting point.

```
// The segment that has the lastest start point is labeled the startSegment, and the new segment that can possibly have
// a better start point is labeled currentSegment. If no start point is selected yet, then the value of startSegment is
// null. If a start point is selected then the new segment is compared to the current startSegment to determine if the
// new segment has a better starting point.

if ( startSegment != null ) {
    // If the currect start segment has an end point
    if ( startSegment.hasEndPoint() ) {
        // If the current segment has an endpoint and the length of the current
        // segment is longer than the current start segment
        if ( currentSegment.hasEndPoint() && currentSegment.length > startSegment.length ) {
            // Then make the current segment the new start segment
            startSegment = currentSegment ;
        }
    }
    // if the current start segment doesn't have an endpoint
    else {
        // if the current segment has an endpoint
        if ( currentSegment.hasEndPoint() ) {
            // then make the current segment the new start segment
            startSegment = currentSegment ;
        }
        // if the length of the current segment is longer than the current start segment
        else if ( currentSegment.length > startSegment.length ) {
            // then make the current segment the new start segment
            startSegment = currentSegment ;
        }
    }
}
// if no start segment is selected yet
else {
    // then make the current segment the new start segment
    startSegment = currentSegment ;
}
```

As mentioned before, in a region, segments with an endpoint are considered first. If there aren't any segments that have at least one endpoint then segments with intersections are considered for a starting point. For the sample input image, the starting point for region one is the pixel (7,2), for region two is pixel (11,2), and for region three is pixel (23,6).

Both regions two and three show special cases where there is more than one possible starting point. In region two there are two possibilities, pixel (11,2) or pixel (17,2). In region three there are also two possibilities, pixel (23,6) or pixel (23,7). For region two, pixel (11,2) and for region three, pixel (23,6) is selected since they are the starting pixels of the first processed segments.

Region two also shows another special case where the starting point is not necessarily part of the longest segment. The pixels (11,2) and (17,2) are preferred over the pixel (14,5) because the first two are endpoints and pixel (14,5) is an intersection. Therefore, this is a case where the starting point is not necessarily a pixel from the longest segment. In contrast, region three has no segments with endpoints, therefore intersections are considered for a starting point.

3.3.3.2 Chain Coding

Once the starting points are determined then the chain-coding phase begins. The purpose of the chain-coding phase is defining a traceable path for each segment. When regions were detected and analyzed, the direction codes of each pixel's neighbors were stored with that pixel. In the chain-coding phase, the extra direction codes are removed and the index of the next pixel is left. Once all the pixels in the segments only have the index of their next neighbor, then those segments are chain coded and ready to be traced. The pseudocode below shows how the extra indices are removed from each pixel.

```
// Save the traveling direction for going to the next pixel
lastDirection = currentSegment.startPixel.value ;
do {
    // Determine the neighbor index that should be removed
    removeIndex = ( ( lastDirection + 4 ) % 8 ) ;

    // Remove the index from the stored indices in the current pixel
    currentSegment.currentPixel.remove ( removeIndex ) ;

    // Save the traveling direction for going to the next pixel
    lastDirection = currentSegment.currentPixel.value ;

    // Move to the next pixel
    currentSegment.currentPixel.movetoNextPixel () ;

// If the end of the segment is reached stop
} while ( currentSegment.currentPixel != currentSegment.endPixel ) ;
```

Applying the chain coding algorithm to the image in Figure 17 produces a tracable image as shown in Figure 18.

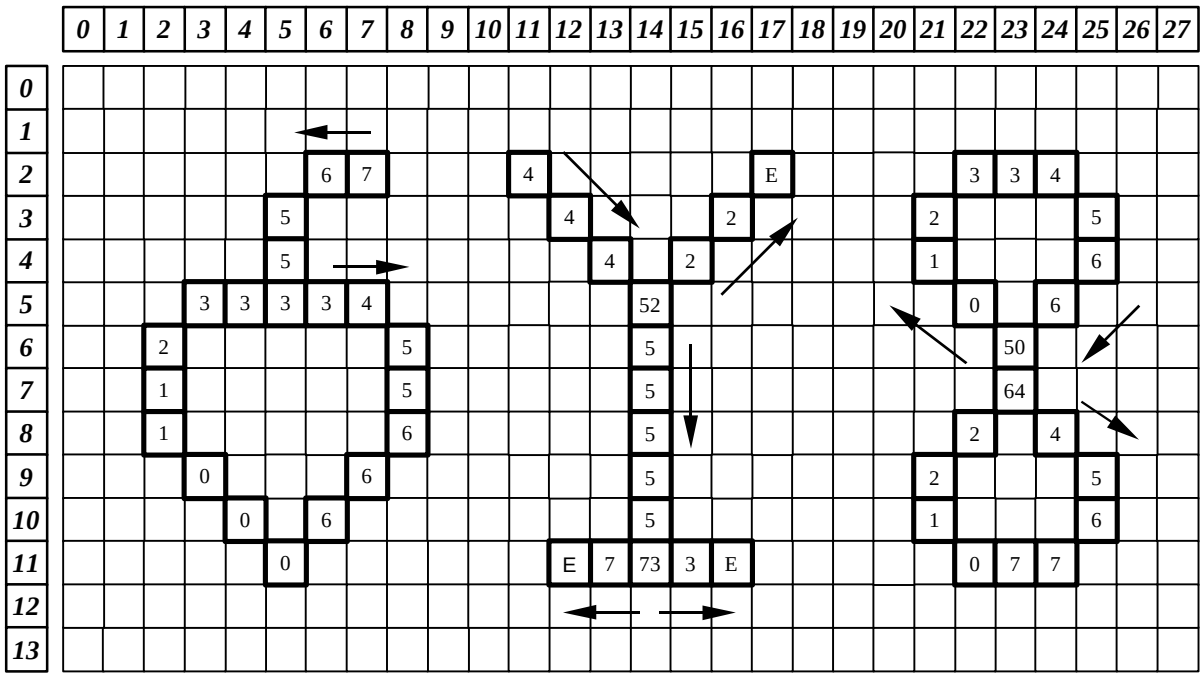

Figure 18: A chain-coded image. Each pixel contains the direction of the next pixel, hence the traceable data.

3.3.4 Padding

Regions that are one pixel wide are very hard to trace even with steady hands and eyesight. After the chain-coding algorithm is applied, the regions are thickened to three pixels wide using a padding algorithm.

The padding algorithm expands each region of the drawing independently and stores the information into an overlay array. Once all the regions are padded, the overlay is superimposed over the original image to create the new padded image. The padding process is very similar to a dilating operation, except that the added pixels must form a traceable path. Therefore, the common dilating algorithms cannot be used for the padding process.

The simplest and most intuitive approach is to copy the value of a pixel in the up and down or left and right neighbors based on where its next neighbor lies. There are two problems with this approach. First, it would be possible to store values over the actual pixels of the image. Second, gaps can form in the pixels that are added around turns. In both cases, the chain code is broken, which results in non-continuous audio feedback. The padding algorithm must consider the special cases where the segment changes direction. Those are the only cases in which simply copying the value of a pixel into its neighbors can result in noncontiguous chain-code. The pseudocode used to expand a segment properly in ImagePilot 2.0 is shown below.

```
// If the current pixel has a neighbor in the 5 positon or has a neighbor in the 1 position
if ( currentPixel.has5Neighbor () || currentPixel.has1Neighbor () ) {
      // If the top or bottom neighbor is occupied, then copy values to left and right neighbors
      copyMode = LEFTandRIGHT ;
}
else {
      // If the left or right neighbor is occupied, then copy values to top and bottom neighbors
      copyMode = UPandDOWN ;
}

// Store the value of the current pixel into the proper neighbors according to the current mode
currentPixel.copyValue ( copyMode ) ;

do {
      // Check if the next pixel can use the curren mode
      occupiedPixel = currentPixel.nextPixel.areNeighborsEmpty ( copyMode ) ;

      // Store the current pixel value
      previousValue = currentPixel.value ;

      // Advance the current pixel pointer to the next pixel
      currentPixel = currentPixel.moveToNext () ;

      // If the next pixel to write in are not occupied
      if ( occupiedPixel == NONE) {
            // If the pixel can use the current mode, then copy its values into the proper neighbors
            currentPixel.copyValue ( copyMode ) ;
      }
      else  {
            // If the pixel can not use the current mode, switch mode and fill proper neighbors.
            currentPixel.copyValue ( ( ( occupiedPixel + 4 ) % 8 ), previousValue ) ;

            // Switch copy mode
            copyMode = !copyMode ;

            // Store the value of the current pixel into the proper neighbors according to the current mode
            currentPixel.copyValue ( copyMode );
      }
// Continue the process until the end is reached
} while ( currentPixel != segment.endPixel ) ;
```

There several ways a segment can change direction. Figure 19 shows the worst special case where a segment starts horizontal then becomes vertical.

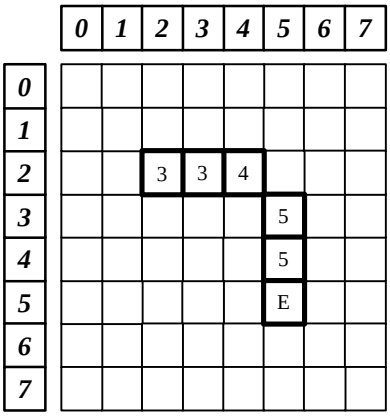

Figure 19: This figure represents a special case where the padding algorithm cannot simply copy the value of each pixel. The padding algorithm must consider the change in direction.

The padding algorithm is designed to handle all different orientations of this special case. Figure 20 shows the systematic padding of the segment shown in Figure 19. As shown in Figure 20 the result of padding a segment is the addition of continuous chain codes around the segment. This type of padding helps ensure continuous audio feedback to the user during tracing.

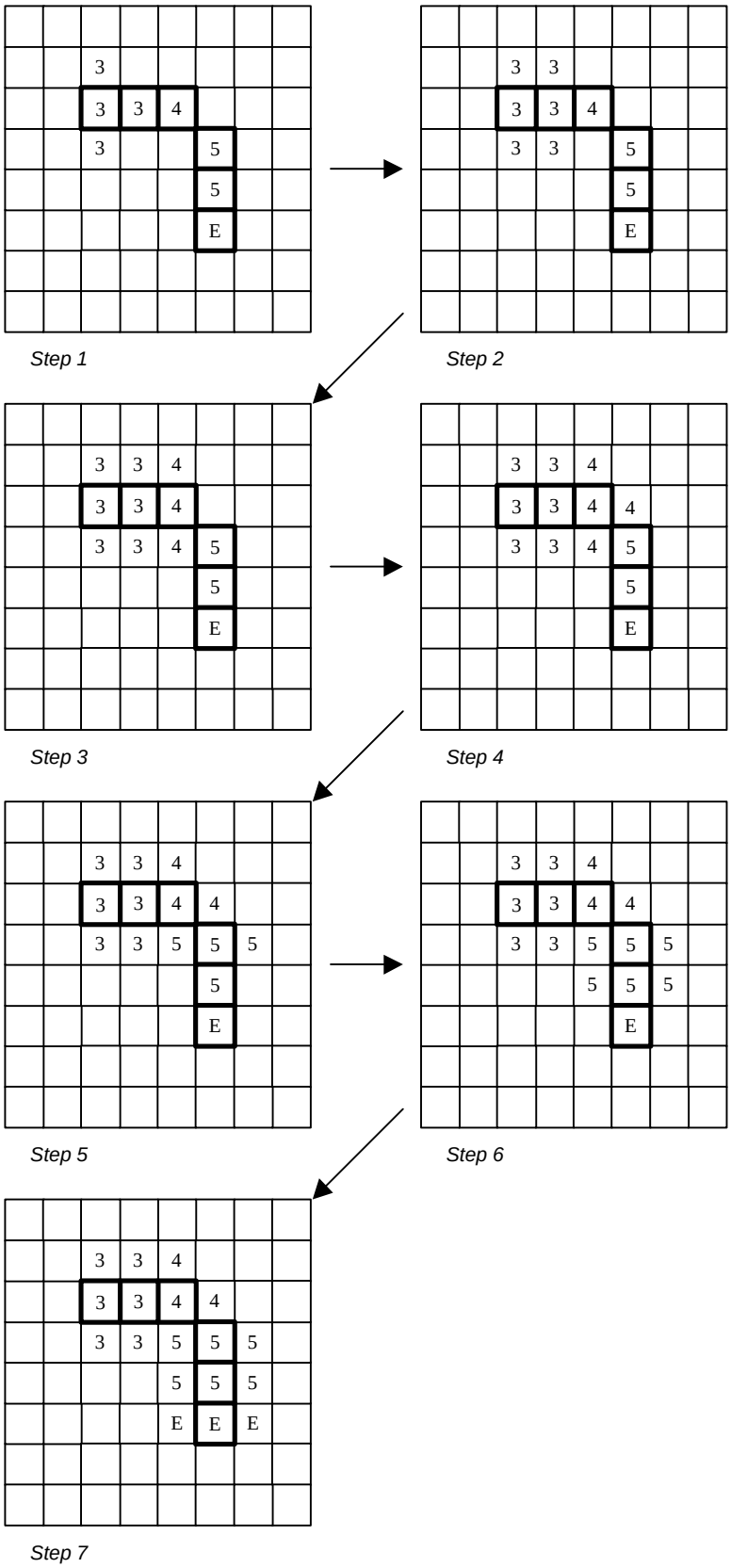

Figure 20: Shows the systematic process of padding a special case. The result is two continuous chain-codes around the original segment. Each chain code is a tracing path by itself.

The padding of a single region starts by padding each individual segment. Figure 21 shows the additional pixels produced for the first segment of each region in the simple input image of Figure 15.

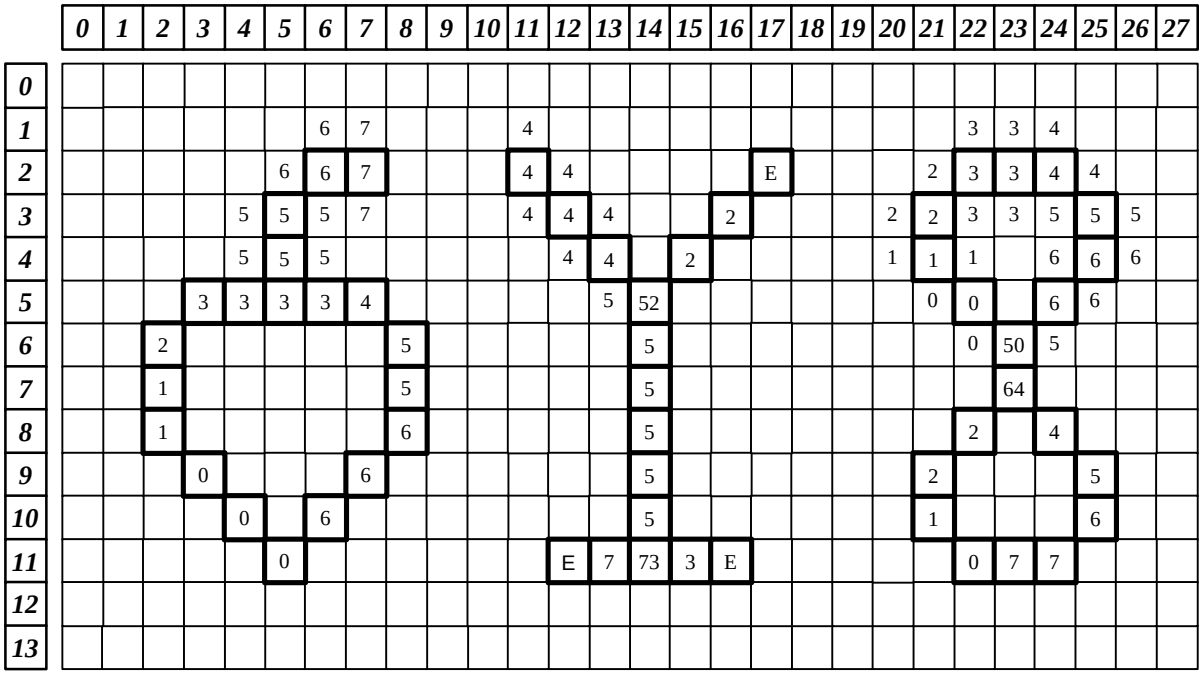

Figure 21: The first segment of each region is padded. Since each region is not connected to other regions, the padding algorithm uses a separate thread to pad all regions simultaneously.

Each region is guaranteed to be disjoint from others, therefore all regions are padded using concurent threads. Figure 22 shows the result of the padding algorithm for all segments in each region.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
0																												
1							6	7				4						E					3	3	4			
2						6	6	7				4	4				2	E				2	3	3	4	4		
3					5	5	5	7				4	4	4		2	2	E			2	2	3	3	5	5	5	
4				3	3	5	3	4					4	4		2	2				1	1	1		6	6	6	
5			2	3	3	3	3	4	4					5	52	2						0	0		6	6		
6		2	2	3	3		3	5	5	5				5	5	5							0	50	5			
7		1	1	1				5	5	5				5	5	5							1	64	4			
8		1	1	1				6	6	6				5	5	5						2	2		4	4		
9			0	0	0		6	6	6					5	5	5					2	2	2		5	5	5	
10				0	0	0	6	6					E	7	5	3	E				1	1	1	7	7	6	6	
11					0	0	6						E	7	73	3	E					0	0	7	7	6		
12						0							E	7		3	E						0	7	7			
13																												

Figure 22: The padding algorithm completed padding all the segments in each region. There are a few gaps in the pad, squares (5,6), (14,4), (14,12), (23,4), and (23,8) that must be filled for continuous guidance.

There are a few gaps in the pad, squares (14,4), (14,12), (23,4), and (23,8) that must be filled for continuous guidance. Gaps can create interrupted audio feedback to the user, so each gap is analyzed and then filled by the algorithm to provide users with uninterrupted guidance. The neighbors of a gap are analyzed and the value that is repeated the most is used to fill the gap. Figure 23 shows the complete overlay superimposed over the original image to form traceable data.

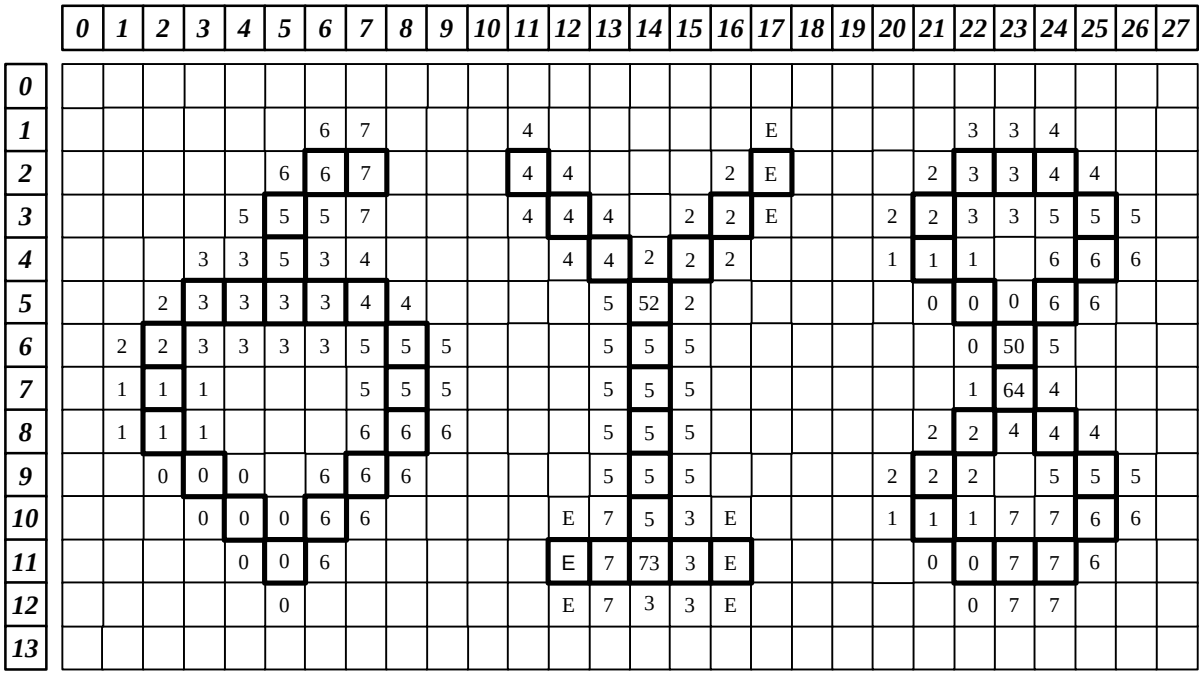

Figure 23: All the gaps are filled and the image is now ready for tracing. The net result of the padding phase is expanding the image by one pixel around its perimeter, inside and outside.

The net result of the padding phase is expanding the image by one pixel around its perimeter, inside and outside. Once the padding phase is complete, the image is ready for tracing.

3.4 Audio Format

Selecting a good set of audio cues is one of the most important parts of this research. The set of tones should minimize confusion for the user, and must be relatively easy to generate. There are no algorithms or equations to define a good set of audio cues. It depends on the application and the needs of the users involved. The main characteristics of a good set of audio tones for this research were:

- Easily understandable.
- Relatively easy and fast to generate.
- Minimize playback time.
- Indicate clearly one of the 8 directions to the user.

A simple approach would have been to use one tone per direction. However, this design would have required eight different tones, which can be very confusing to an untrained ear. On the other hand, eight verbal feedbacks would not be as confusing. The drawback with verbal feedback is the time it requires to clearly pronounce a certain direction. In either case, care must be taken to select sounds that are not overly irritating to the user when played repeatedly.

It is important to understand the audience by which this program will be used. The assumption is that the majority of the users are sight-impaired users with normal hearing. Given this assumption, left and right speakers can be used to convey directional information in addition to the tones themselves. For example, a tone can be assigned for moving up (vertically in the image). Therefore, a tone played with equal volume from both speakers indicates a vertical movement, or the direction up right can use the same tone only played out of the right speaker. Consequently, the direction up-left can also use the same tone played in the left speaker.

Using the left and right speakers in this fashion, allows the program to use only three different tones to represent the eight directions. Since high frequency tones can be irritating to hear, an upper bound was established by testing different tones and just selecting a suitable upper bound. The other two (lower) frequencies were selected such that the three tones are easily distinguishable from each other. The three tones selected for this program are a 660 Hz tone to represent up, a 440 Hz tone for the level plane, and a 220 Hz tone for down. Table 4 shows the actual audio assignment for each of the eight possible directions a user can travel from a pixel.

Table 4: This table presents the audio assignment scheme used to provide feedback to the user. For example, to tell the user to move in the up-left direction a 660 Hz tone is played only in the left speaker.

Direction	Tone Generated
Up-left	A 660 Hz tone in left speaker
Up	A 660 Hz tone in both speakers
Up-right	A 660 Hz tone in right speaker
Right	A 440 Hz tone in right speaker
Down-right	A 220 Hz tone in right speaker
Down	A 220 Hz tone in both speakers
Down-left	A 220 Hz tone in left speaker
Left	A 440 Hz tone in left speaker

For users who prefer verbal feedback, possibly those with partially impaired hearing, verbal feedback can be selected using a simple command to ImagePilot 2.0. As mentioned before, the drawback with verbal tones is the playback time required for delivering clear and concise audio cues. Consequently, the only effect to the user is the decrease in speed at which they will receive audio cues while tracing the image.

4. Run Time Operation

Depending on the configuration of the system a sight-impaired user might need the assistance of a sighted user to start the program. Some systems come equipped with voice recognition capabilities that allows a sight-impaired user to perform basic tasks without any assistance, such as running applications and opening files. Whether a sighted user is needed or not, the program requires three parameters: the name of the executable file for the Java virtual machine, the name of the program, and finally the input image file.

The program is designed to be invoked completely from a keyboard. The syntax to run the program with a Java runtime environment is “jre ImagePilot <input file name>”. The executable file is named jre.exe and the “<input file name>” parameter is the file to be opened by ImagePilot 2.0 and traced by the user. More details are given in Appendix A.

ImagePilot 2.0 was designed and developed for use with digitizing tablets with a pen pointing device to trace the electronic images. However, any input device that emulates a mouse can be used for tracing. In addition to using a pointing device for tracing an image, the user can use the keyboard to invoke different commands.

After the image has been completely processed, the user can begin tracing the image using the pointing device. ImagePilot 2.0 is designed to guide the user to the beginning of the first segment with verbal cues. Forcing a user to trace a region completely before going to the next region is equivalent to asking a user find every pixel in the region. This is a very difficult task even for a sighted user. For this reason, the user can use a simple keyboard stroke to move to the next region at anytime.

The general design is that ImagePilot 2.0 will guide the user to the beginning of the first region. Once the beginning is reached, the audio system switches to tone cues to trace the segments. The two basic guiding concepts designed into ImagePilot are to get the sight-impaired user to the foreground pixels, then keep the user on the foreground pixels. The user receives tone cues to trace segments. In addition, they are verbaly informed when an intersection or endpoint is reached. Due to the limited available audio technoloy, the verbal cues are short and minimized to achieve better runtime performance. Besides having to start from a specific segment, there are no other restrictions on tracing the segments of a particular region. If the user reaches an intersection, they are not forced to trace any particular branch. Once the user decides they are done with the first region, they can move to the next region.

Although ImagePilot 2.0 attempts to guide a sight-impaired user through an image without the need for instructions from a sighted user, it also provides convenient options for the users. The user can direct the program to move to the next region, restart tracing the current region, restart tracing the image, toggle the audio feedback between verbal and tone, and also invoke the audio familiarization feature. Table 5 shows the keyboard assignments for the different built-in features.

Table 5: This table shows the keyboard assignments for the built-in features of ImagePilot 2.0. The user can invoke any of these actions by simply pressing the designated key on the keyboard.

Keyboard key	Action taken by ImagePilot 2.0
N	Move and start tracing the next region
R	Restart tracing the image from beginning
T	Restart tracing the current region
P	Play the audio test for familiarization
V	Toggle between verbal and audio feedback

ImagePilot 2.0 has some features that provide automatic assistance to users. One of these features is the Auto-Guidance system. This system is automatically invoked when the user drifts away from the most recent region visited. Once activated, the system guides the user to the last known point on the most recently visited region. The other feature, Visual-Tracing, displays the path the user traveled while attempting to trace the image. This feature is turned on and off by consecutive pen or mouse clicks. As illustrated in Figure 24, this data can be useful to researchers in evaluating ImagePilot 2.0, or for studying the capabilities of sight-impaired users.

Figure 24: This shows the resulting trace by a user with the Visual-Trace feature active. The red lines show the path the user traveled to trace certain regions of this image.

The user can leave ImagePilot 2.0 in several ways. On Windows based systems, if the keys Alt and f4 are pressed simultaneously the active application is terminated. For this method to work ImagePilot 2.0 must be the active application. Note that the last application invoked is the active application. If the system is equipped with voice recognition software, then the user can also terminate ImagePilot 2.0 verbally by the command they assigned to terminating active applications.

5. Testing

ImagePilot 2.0 requires two modules to run properly on any system, Java Runtime Environment (JRE) 1.1.7 or higher and Java Media Framework (JMF) 1.0.2 or higher class files. Since Java is a relatively new programming language, many of its components are not yet ported to different operating systems. ImagePilot 2.0 was tested only on Microsoft Windows based systems, because at this time the audio package, JMF, is only available for Windows operating systems.

The first testing environment used a 90 MHz Intel Pentium based system running Windows NT operating system. The second testing environment used a 300 MHz Intel Pentium II based system running Windows 98. All test users used a SummaSketch III tablet with a two-button pointing pen as shown in Figure 25.

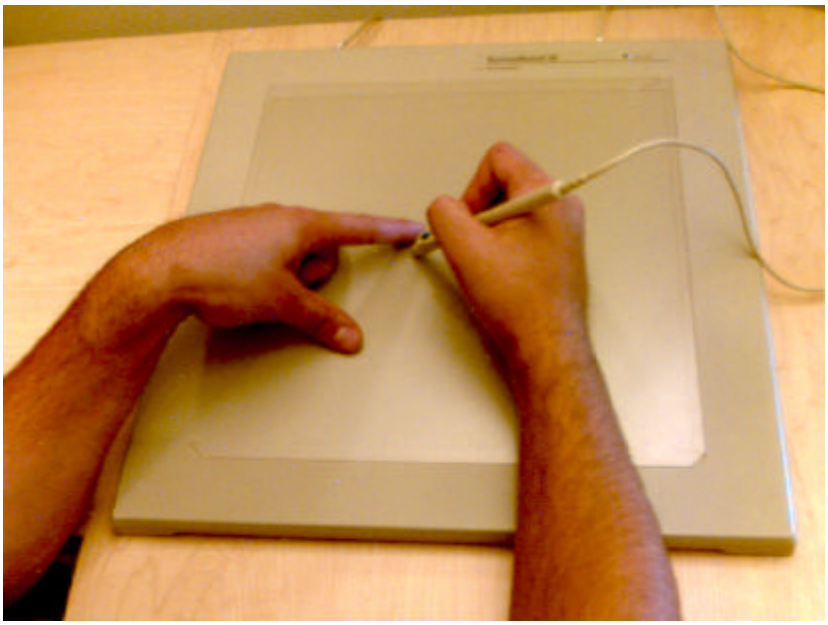

Figure 25: A SummaSketch III tablet used for receiving input from user. Using a digitizing tablet allowed users to trace image with higher accuracy than a mouse.

Although ImagePilot 2.0 is capable of receiving user input from other pointing devices such as a mouse, a digitizing tablet with a pen-pointing device is the recommended

option. Using a tablet maximizes ImagePilot's ability to guide a user and allows users to trace digital images with higher efficiency.

Testing was performed in two phases using two different groups of people. Table 6 shows some details about the subjects.

Table 6: This table shows some details about the individuals that tested ImagePilot 2.0. All subjects in test group 1 had normal eyesight and were blind folded during test. All subjects in test group 2 were sight-impaired.

	Test Group 1			**Test Group 2**		
Test User	S1	S2	S3	S4	S5	S6
Age	31	24	28	21	25	24
Eye sight	Normal	Normal	Normal	None	None	Outlines
Blind since	N/A	N/A	N/A	1	1	14
Sex	Male	Male	Male	Female	Female	Male
Occupation	Student	Student	Architect	Student	Student	Student

Test group 1 consisted of three people, all of them with normal (or near-normal) eyesight. During the initial prototype-testing phase, these users were blindfolded and had no prior knowledge about the test set. As the first testing group, it was important to be able to talk and visually explain how the system works and get feedback after the user was done with an image.

The image set consisted of five different shapes: a straight line, a square, a star, a circle, and finally a house, as shown in Figure 26.

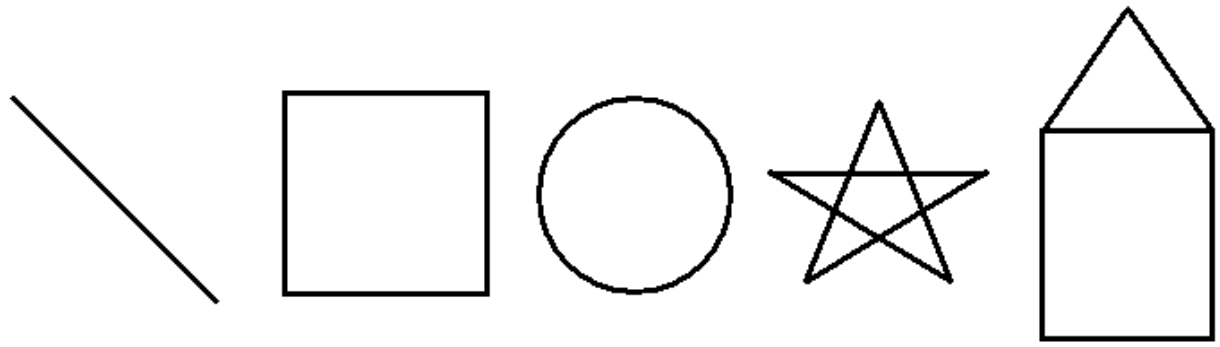

Figure 26: The five test images administered to each of the test subjects. The most difficult image to identify among the five test cases was the star.

The straight-line test image demonstrates the simplest case, where the program guides the user to a starting point and then in one direction along the line. The main purpose of the line, square, and circle test images was to determine how effectively a user can use a tool to recognize some common shapes. Another goal of the circle test case was to determine how effectively the program can guide the user around a constantly turning curve. The house test case was used to determine how important the role of mental image is, which is revealed later in this section. The star test case is selected to test the effectiveness of the tool in dealing with images that have many branches.

The users in test group 1 recognized all the objects displayed in Figure 26 except for the star. The star has many intersections with multiple branches, which makes it difficult for user to remember the different segments of the region. One user suggested that it would have helped if the system also told the users if a segment had been traced previously. The initial prototype-testing phase ended with moderate success. Table 7 shows the individual performance of the test users for the initial prototype-testing phase.

Table 7: This table shows the individual results of each test user in Test Group 1. All users were blindfolded during the testing phase and had no prior knowledge about the test image set.

	Line	Square	Circle	Star	House
S1	Recognized	Recognized	Recognized	Not Recognized	Recognized
S2	Recognized	Recognized	Not Recognized	Not Recognized	Recognized
S3	Recognized	Recognized	Recognized	Not Recognized	Recognized

The second prototype-testing phase included only users that were visually impaired. After hearing an explanation of how the system works, each user had a chance to practice using ImagePilot 2.0 with the practice line drawing shown in Figure 27. They were also given a raised line drawing on a sheet of paper of the same drawing. The purpose of this practice image was to provide all the possible audio clues during their practice session.

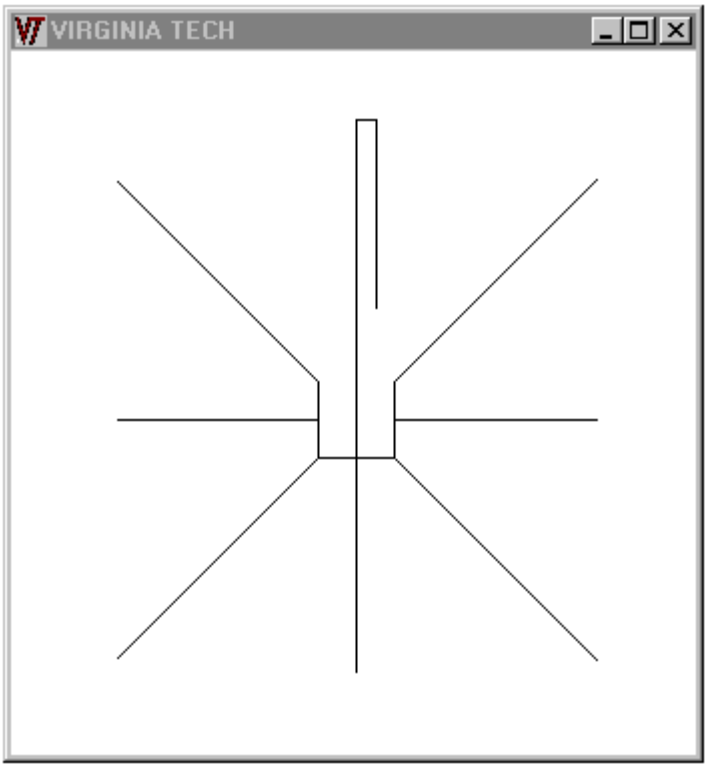

Figure 27: A sample line drawing used to allow test users to acclimate themselves with ImagePilot 2.0 before the actual prototype-testing phase.

After each user had a chance to practice with ImagePilot 2.0 and felt ready, they were given the test image set shown in Figure 26 in random order without advance information concerning the image content. Table 8 gives more details on the individual performance of each test user in Test Group 2.

Table 8: This table shows the individual results of each test user in Test Group 2.

	Line	Square	Circle	Star	House
S4	Recognized	Recognized	Not Recognized	Not Recognized	Recognized
S5	Recognized	Not Recognized	Recognized	Not Recognized	Not Recognized
S6	Recognized	Recognized	Recognized	Not Recognized	Recognized

As might be expected, the performance of Test Group 2 was not as good as Test Group 1. It seems that the blindfolded sighted users were able to create a better visual model of the test images. Their greater ability maybe attributed to their experience of seeing and remembering shapes and objects throughout their lifetimes.

A second experiment was carried out with Test Group 2 using the same test image set. This time they knew in advance what different shapes would be present in the test set, but the images were presented in a different order. Table 9 gives the result of this experiment.

Table 9: This table shows the individual results of each test user in Test Group 2 for the second test.

	Line	Square	Circle	Star	House
S4	Recognized	Recognized	Recognized	Recognized	Not Recognized
S5	Recognized	Recognized	Recognized	Recognized	Recognized
S6	Recognized	Recognized	Recognized	Recognized	Recognized

The results are much better than the first time. This time the test users knew the shapes in the set and were able to use the audio feedback from ImagePilot 2.0 to recognize them with higher accuracy. The results of the second test are improved partially based on the users' prior knowledge about the test images. The second test

suggests that giving users general knowledge concerning the image content can help them recognize drawings with higher accuracy.

Users in Test Group 2 also added a few suggestions for improving the quality of ImagePilot 2.0 that are listed below:

- Allow the user to select the starting point.
- Give the users some general information about the image prior to testing, such as number of endpoints, intersections, and segments.
- Do not repeat directional information as long as the user is tracing in the correct direction.
- Tell the user when a previously traced segment is encountered again.

Given time and resources these comments can be incorporated into the next version of ImagePilot without much difficulty. The users overall impression of ImagePilot 2.0 was positive and they would like to see newer and better releases of the system.

6. Conclusions

This thesis has presented a novel software tool that can aid sight-impaired users in the interpretation of digital line drawings. The system processes digital images stored in GIF format, and then provides audio and verbal cues to a user who interactively explores the images. The goal of this research was the development of a working prototype drawing interpretation tool. This thesis describes the design and test results for a prototype that performed well.

In spite of the successes of ImagePilot 2.0, there is still much room for improvement. For example, pattern recognition techniques could be incorporated to identify simple common shapes such as circles and squares. Existing techniques such as thinning and padding could be used to provide more flexibility and accuracy. The tracing system could be improved to provide more guidance for recognizing images with many regions and segments.

The ultimate goal of developing a verbal interpretation tool for sight-impaired users is a complex and challenging process. Although “tactile images” still remain the prevailing means in conveying the world around us to sight-impaired users, ImagePilot 2.0 is a convenient alternative. ImagePilot 2.0 is a working prototype that represents a large step in this direction. It provides a framework for better tools to come, tools with more recognition capabilities, faster processing, sophisticated audio feedback, and more user-friendly features.

APPENDIX A. INSTALLATION & REQUIREMENTS

A.1 Overview

ImagePilot 2.0 is written in Pure Java using Java API specification 1.1.7 and Java Media Framework API specification 1.0.2 from Sun Microsystems, Inc. Theoretically, it is capable of running on any platform that has Java support installed. ImagePilot 2.0 does not have any additional memory and hardware requirements over those needed to install and run a Java environment.

ImagePilot 2.0 was developed on an Intel based system with Windows NT 4.0 as the operating system. The program was developed using Sun's Java Development Kit 1.1.7 and Sun's Java Media Framework 1.0.2 packages. There are three steps in properly installing Java support for ImagePilot 2.0:

1) Installing a Java Runtime Environment (JRE) version 1.1.7 or higher.
2) Installing Java Media Framework (JMF) version 1.0.2 or higher.

All the files necessary to run ImagePilot 2.0 are packaged into a single compressed file, Ipilot2.zip, which is obtainable from http://www.ee.vt.edu/~fvalad.

A.2 Installing JDK or JRE

The user must have the Java library classes in order to run ImagePilot 2.0. The user can obtain these files either by installing a JDK or JRE. A JDK is used primarily for Java development and/or debugging. It also allows users to run Java applications. On the other hand, a JRE is all the essential class files that are needed to run Java applications. The JDK or JRE packages can be obtained from the Internet at http://java.sun.com/products/jdk/. After successfully installing and configuring a JDK or JRE, the user can run ImagePilot 2.0. The command to invoke ImagePilot 2.0 using a JDK is "Java ImagePilot <GIF filename>", and using a JRE is "Jre ImagePilot <GIF

filename>". The original packaging of ImagePilot 2.0 contains a file named "run.bat", which ensures proper pathname setup as well as automated program invocation.

A.3 Installing JMF

The JMF can be obtained from the Internet at http://java.sun.com/products/java-media/. After successfully downloading and installing the JMF package, the CLASSPATH variable must be set to include the path to the JMF class files. The CLASSPATH variable provides the possible locations that a Java Virtual Machine must search for library class files. Once properly set, the JVM will load the necessary class files to support ImagePilot 2.0 at run time.

APPENDIX B. README AND JAVA SOURCE CODE

This section contains the Readme file and all the source code developed for ImagePilot 2.0. The source code files are presented alphabetically by their file names. The main method is in the file named ImagePilot.java, which begins the execution of the program. The Readme file contains basic information such as system requirements, installation, and new features. This file is provided as an electronic reference that describes how to install and use ImagePilot 2.0. The description and importance of the main files are listed below.

- ImagePilot.java – Contains the code that begins execution when the program is invoked. The three main modules ImageProcessor, AudioPlayer, and ImageTracer are all invoked and used from this file.
- ImageProcessor.java – Contains the code to process the input image file. This module has three independent sections that thins, analyzes, and stores data. This module uses the code in BWFilter.java, BWINVFilter.java, and ImageThinner.java to prepare and thin the image. The ImageProcessor module also makes use of the code in ImageThread.java, and ChainThread.java to analyze the prepared image. Finally, it makes use of the code in Pixel.java, Segment.java, and SegTable.java to store the analyzed data.
- AudioPlayer.java – Contains the code to create all the necessary audio feedback objects for use by ImageTracer module. The AudioPlayer module also contains the code for playing each feedback.
- ImageTracer.java – Contains the code that intercepts pointing device movements and provides audio feedback based on the movements.

B.1 Readme

```
README.TXT for ImagePilot 2.0
=============================

Thank you for choosing to evaluate ImagePilot 2.0.

TABLE OF CONTENTS
-----------------
    INTRODUCTION
    MINIMUM REQUIREMENTS
    INSTALLATION
    WHAT'S NEW
    RUNTIME

INTRODUCTION:
-------------
ImagePilot 2.0 is the first complete interpretation tool developed for the
sight-impaired.  The goal of the tool is to allow a sight-impaired user to
trace and interpret a digital image, something that was not possible before.
Although some verbal feedback is used, speech synthesis techniques are not
yet mature enough to allow a variety of verbal responses. ImagePilot 2.0
defines and demonstrates techniques for presenting electronic images to the
sight-impaired.

To learn more, install ImagePilot 2.0 by referring to the instructions below.

MINIMUM REQUIREMENTS:
---------------------
   o Pentium 90 MHz with minimum 32MB of RAM (64MB is recommended)
   o 8MB of free hard drive space
   o Sound card with pre-installed sound drivers
   o Stereo Headphones (Recommended) or stereo speakers
   o Digitizing Tablet with a pointing pen (Recommended) or mouse
   o ImagePilot 2.0 is written in Pure Java, theoretically allowing
     the program to run under any environment.


INSTALLATION:
-------------
    To install ImagePilot 2.0 from a ZIP file:
        1) Download ImagePilot 2.0 from http://www.ee.vt.edu/~fvalad
        2) Unzip the file using the existing folder names.

    Important Notes:
    ----------------
    + You do not need to have any Java Environment pre-installed.  The zip
      File contains all the necessary files, including Java Multimedia
      support.

    + Your path must include or only have ..\IPilot2\lib and ..\IPilot2\bin
      in the path.

WHAT'S NEW:
-----------
```

- Faster and advance feedback system.
  Audio feedback now includes verbal tones for all direction in 8-connected scheme. The user may toggle between tone and verbal by pressing the v key on the keyboard. The audio module is redesigned to complete playing a verbal tone without interruption. Also the system is upgraded to use system events to detect playback completion, player creation, and player caching.

- Last known position guiding system.
  If the program detects that the user is confused and lost, then the self guiding system is invoked. The system will guide the user to he last known pixel that was traced. The system start after user moves to 25 non-foreground pixels.

- Thinning support.
  ImagePilot 2.0 uses a ported Zhang-Suen thinning algorithm to thin thick images.

- Self contained install package.
  The zip file contains all the files necessary to run ImagePilot 2.0.

- Faster processing and lower memory requirement.
  ImagePilot 2.0 uses multi-threading techniques to process an image, providing maximum efficiency possible by the algorithm.

- Continuous padding algorithm.
  ImagePilot 2.0 uses a new padding algorithm the dilates the image by one pixel on each side without any gaps in the pixels.

- Audio familiarization system.
  ImagePilot 2.0 includes a sub system that plays every verbal feedback and its corresponding tone feedback to help to user learn the meaning of the different tones.

- Trace Tracking system
  You can now see how the user has attempted to trace an image. The system is started and stopped with a left mouse click, resulting in a red line denoting the trace path.

- Self Adjusting GUI.
  ImagePilot 2.0 now detects the users resolution and adjusts its GUI and the image to center according to the new settings.

RUNTIME:

The install zip file contains a batch file, run.bat, to run the default sample image provided. The command line syntax for ImagePilot 2.0 is

```
jre -cp .;lib/jmf.jar ImagePilot <image filename>
```

where <image filename> is replaced with the name of the acutal image file to be traced.

The guiding system works in two stages, the first stage guides you to the begining of the first region using verbal tones. Once that point is reached the tone system is activated to guide the user in tracing the region.

If at any point the user travels 25 pixels with out crossing the region then the verbal system is reactivated to guide the user to the last known position on that region.

The user may use the built-in keyboard commands to start tracing different regions in the image. The built-in keyboard commands are described in the following table.

Keyboard	Action taken by ImagePilot 2.0
N	Move and start tracing the next region
R	Restart tracing the image from beginning
T	Restart tracing the current region
P	Play the audio test for familiarization
V	Toggle between verbal and audio feedback

B.2 AudioPlayer.java

```
/*
 * Copyright 1997-1998 by Farzad Valad,
 * POB 8239 Longmont CO 80501
 * All rights reserved.
 */

import java.io.*;
import java.awt.*;
import java.net.*;
import javax.media.*;
import java.awt.event.*;

/**
 * This class is responsible for creating, caching, preparing all the wave
 * files for playback.  It also performs a complete audio system test when
 * requested.
 */

class AudioPlayer implements ControllerListener {
    transient ActionListener actionListener;

    // Array of all the possible wave files
    private Player[] player = null;

    // Marks the beginning of a wave file
    private Time begin = null;

    // Holds the last mouse coordinate that a wave file was played for
    private Pixel lastPt = null;

    // Holds the index of the current playing wave file
    private int activePlyr = -1;

    // Flag to determine if a wave file is playing
    private boolean done = true;

    // Flag representing the initialization state of all wave files
    private boolean init = false;

    // Flag to determine if the audio player is ready
    private boolean ready = false;

    // Variable of playing back all the current wave files
    private int selfTestIndex = 0;
    private boolean selfTest = false;
    private int[] selfTestOrder = {8,0,9,1,10,2,11,3,12,4,13,5,14,6,15,7};

    private int construct = 0;
    public AudioPlayer() {
      // Initialize the default values and construct objects
      begin = new Time(0);
      lastPt = new Pixel(-1,-1);
      player = new Player[Const.MAXPLYRS];

      URL mediaURL = null;
      try {
         mediaURL = new URL(Const.PATH + "audio\\" + construct + ".wav");

         // Fetch the wave file
         this.player[construct] = Manager.createPlayer(mediaURL);

         // Create the player and start caching the file
         this.player[construct].realize();
         this.player[construct].addControllerListener(this);
         construct++;
      } catch (MalformedURLException e) {
         Const.fatalError("Invalid media file URL!");
```

```
  } catch(NoPlayerException e) {
     Const.fatalError("Unable to find a player for "+mediaURL);
  } catch(NullPointerException e) {
     Const.fatalError("Could not create player for "+mediaURL);
  }catch(java.io.IOException e) {
     Const.fatalError("IO Exception encountered opening " + mediaURL);
  }
}

public synchronized void addActionListener(ActionListener l) {
  actionListener = AWTEventMulticaster.add(actionListener, l);
}

public synchronized void removeActionListener(ActionListener l) {
  actionListener = AWTEventMulticaster.remove(actionListener, l);
}

protected void reportDone() {
    if (actionListener != null) {
        actionListener.actionPerformed(new ActionEvent(lastPt,0,"AudioPlayer"));
    }
}

public void stop() {
  // if a wave file is playing
  if(init && done && activePlyr>=0) {

     // Stop the playing wave file
     player[activePlyr].stop();

     // Reset the last active wave file
     player[activePlyr].setMediaTime(begin);

     // Make player ready for the next request
     activePlyr = -1;
     reportDone();
     ready = true;
  }
}

public void playAudio(int audio) {
  // Play the requested wave file
  player[audio].start();

  // Set flags so that wave file finishes playing
  activePlyr = -1;
  ready = false;
  done = false;
}

public void playerStart(int newPlyr, Pixel pt) {
  // If the player is ready and the new file is a different
  // request than the current active wave file
  if(init && done && newPlyr!=activePlyr) {
     // Stop any wave file that might be playing
     if(activePlyr!=-1) {
        player[activePlyr].stop();
        player[activePlyr].setMediaTime(begin);
     }

     // Start playing the file and set the Flags
     ready = false;
     player[newPlyr].start();
     activePlyr = newPlyr;
     lastPt.move(pt.x,pt.y);

     // If the request file is a verbal feedback set the
     // flag "done" so the the entire file is played
     if(activePlyr>7) {
        done = false;
     }
```

```
  }
}

public boolean isReady() {
  return ready;
}

public void toggleSelfTest() {
  selfTest = !selfTest;
  if(selfTest && ready)
     playAudio(selfTestOrder[selfTestIndex++]);
  else
     selfTestIndex = 0;
}

public synchronized void controllerUpdate(ControllerEvent e) {
  // Get the reference to the active player
  Player p = (Player)e.getSource();

  // If the wave file has completed reset the file and flags
  if (e instanceof EndOfMediaEvent) {
     p.setMediaTime(begin);
     activePlyr = -1;
     done = true;
     ready = true;

     // If in a testing mode play the next file
     if(selfTest) {
        if(selfTestIndex<selfTestOrder.length)
           playAudio(selfTestOrder[selfTestIndex++]);
        else {
           selfTest = false;
           selfTestIndex = 0;
        }
     }
     else {
        reportDone();
     }
  }

  // When all players for the wave files are created
  else if(e instanceof RealizeCompleteEvent) {
     System.out.print("");
     p.setMediaTime(begin);
     p.prefetch();
  }

  // When all players for the wave files are cached
  else if(e instanceof PrefetchCompleteEvent) {
     if(!init) {
        if(p.equals(player[player.length-1])) {
           System.out.println("Done constructing " + player.length + " players");
           init = true;
           ready = true;
        }
        else {
           //System.out.println("Wave file number " + construct + "Prefetched.");
           URL mediaURL = null;
           try {
              mediaURL = new URL(Const.PATH + "audio\\" + construct + ".wav");

              // Fetch the wave file
              this.player[construct] = Manager.createPlayer(mediaURL);

              // Create the player and start caching the file
              this.player[construct].realize();
              this.player[construct].addControllerListener(this);
              construct++;
           } catch (MalformedURLException mue) {
              Const.fatalError("Invalid media file URL!");
           } catch(NoPlayerException npe) {
```

```
                Const.fatalError("Unable to find a player for "+mediaURL);
             } catch(NullPointerException npe) {
                Const.fatalError("Could not create player for "+mediaURL);
             }catch(java.io.IOException ioe) {
                Const.fatalError("IO Exception encountered opening " + mediaURL);
             }
          }
       }
    }
 }
```

B.3 BWFilter.java

```
/*
 * Copyright 1997-1998 by Farzad Valad,
 * POB 8239 Longmont CO 80501
 * All rights reserved.
 */

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

/**
 * This class is a filter to convert a color file into a black and white file
 */

class BWFilter extends RGBImageFilter {
    public BWFilter() {
      canFilterIndexColorModel = false;
    }

    public int filterRGB(int x, int y, int rgb) {
      int red   = (rgb & 0x00ff0000) >> 16;
      int green = (rgb & 0x0000ff00) >> 8;
      int blue  = (rgb & 0x000000ff);

      // If pixel is a color pixel
      if (!(blue == 0xff  &&  green == 0xff  &&  red >= 0xff))
         // make it black
         return 0xff000000;
      else
         // otherwise return itself
         return rgb;
    }
}
```

B.4 BWInvFilter.java

```
/*
 * Copyright 1997-1998 by Farzad Valad,
 * POB 8239 Longmont CO 80501
 * All rights reserved.
 */

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

/**
 * This class is inverts a black and white image file, where white
 * pixel turn black, and black pixels turn white.  Inverting the
 * image file helps reduce processing logic, because the pixelgrabber
 * method assigns a 0xff000000 to white pixels.
 */

class BWInvFilter extends RGBImageFilter {
    public BWInvFilter() {
      canFilterIndexColorModel = false;
    }

    public int filterRGB(int x, int y, int rgb) {
      int red   = (rgb & 0x00ff0000) >> 16;
      int green = (rgb & 0x0000ff00) >> 8;
      int blue  = (rgb & 0x000000ff);

      // If the pixel is not a black pixel
      if (!(blue == 0xff  &&  green == 0xff  &&  red >= 0xff))
         // return black
         return 0xffffffff;
      else
         // otherwise return white
         return 0xff000000;
    }
}
```

B.5 ChainThread.java

```
/*
 * Copyright 1997-1998 by Farzad Valad,
 * POB 8239 Longmont CO 80501
 * All rights reserved.
 */

class ChainThread implements Runnable
{
    // The current component
    private int comp;

    // List of all active threads
    private Thread[] t;

    // Maximum number of concurrent threads
    private int maxThreads;

    // The entering direction into a pixel
    private byte[] remD;

    // The coordinate fot the current pixel
    private Pixel[] currPt;

    // Debuging vairables for printing the chain code
    private byte[] printD;
    private Pixel[] printPt;

    // Referece to a segment table for the current component
    private SegTable segT;

    // Variable to track current active threads
    public static int threadCount = 0;

    // A reference to ImageProcessor to access the data array
    private static ImageProcessor imgPr;
    private ThreadGroup tGrp = new ThreadGroup("Threads");

    //Pixel Scan Standard:          0 1 2
    //top left corner is 0 (clockwise) 7 X 3
    //dxdyTable[][]={0,1,2,3,4,5,6,7}  6 5 4
    private  static  final  byte  dxdyTable[][]={{-1,-1},{0,-1},{1,-1},{1,0},{1,1},{0,1},{-1,1},{-
1,0}};

    // Constructor for creating and initializing data objects
    public ChainThread(ImageProcessor imgPr, int maxThreads) {
      this.imgPr = imgPr;
      this.maxThreads = maxThreads;

      this.remD = new byte[maxThreads];
      this.currPt = new Pixel[maxThreads];

      this.printPt = new Pixel[maxThreads];
      this.printD = new byte[maxThreads];

      this.t = new Thread[maxThreads];
    }


    public void setSegTable(SegTable segT) {
      this.segT = segT;
    }
    // Start a thread if possible from startPt in the direction of d
    // The startPt belongs to the component comp
    public boolean startThread(Pixel startPt, byte d, int comp) {
      int i;
      this.comp = comp;
```

```
  // Find an open slot in the thread array
  synchronized(this) {
     for(i=0; i<maxThreads && t[i]!=null; i++);
  }

  if(i==maxThreads)
     return false;
  else {
     if(currPt[i]==null)
        this.currPt[i] = new Pixel(startPt);
     else
        this.currPt[i].move(startPt);

     // Since each point has the indices of the neighbors
     // remember which one needs to be removed
     this.remD[i] = d;

     //for printing purpose only
     if(printPt[i]==null)
        this.printPt[i] = new Pixel(startPt);
     else
        this.printPt[i].move(startPt);

     //for printing purpose only
     this.printD[i] = d;


     this.t[i] = new Thread(tGrp, this, Integer.toString(i));
     t[i].start();

     synchronized(this) {
        threadCount++;
     }

     return true;
  }
}

public void run() {
  int stop;
  // Get the id of the current thread
  int currT = Integer.parseInt(Thread.currentThread().getName());

  do {
     // Advance the thread to the next point
     stop=moveNext(currT);
     if(stop!=0) {
        spawnThreads(currT, stop);
        break;
     }
  }while(setChainNum(currT));

  synchronized(this) {
     // Is a t[currT]==null; required?
     threadCount--;
  }
}

// For debugging purpose only, it is kept for reference
// Prints a textual representation of the chain code for a thread
public synchronized void printChainCode(int currT) {
  int x,y,newD;
  boolean stop = false;

  System.out.print(printPt[currT] + " ");

  do {
     newD = imgPr.getPixel(printPt[currT]);

     if(newD>7 || newD==Const.ENDP) {
        if(stop)
```

```
            break;
         else {
            System.out.print(newD + " << ");
            newD = printD[currT];
         }
      }

      System.out.print(newD + " ");

      x = printPt[currT].x + dxdyTable[newD][0];
      y = printPt[currT].y + dxdyTable[newD][1];

      printPt[currT].move(x,y);
      stop = true;

   }while(newD<=7);

   System.out.println(newD);
}

// Stores the proper chain number and decides whether the
// current thread needs to terminate.  In that case it will
// return a false and the thread is terminated in the run method.
public boolean setChainNum(int currT) {
   int pixVal = imgPr.getPixel(currPt[currT]);

   // If the current pixel is the start or end of a line segment
   // then mark that pixel and terminate the thread (return false).
   if(pixVal<=7 || pixVal==Const.ENDP) {
      imgPr.setPixel(currPt[currT],Const.ENDP+comp*100000);
      return false;
   }
   // If the current pixel is not an endpoint nor an intersection
   // then remove the unwanted neighbor info depending on remD array
   // and store the value with the component info.
   else if(pixVal<=75)  {
      if(pixVal%10 == remD[currT]) {
         imgPr.setPixel(currPt[currT],(pixVal/10)+comp*100000);
      }
      else {
         imgPr.setPixel(currPt[currT],(pixVal%10)+comp*100000);
      }
      return true;
   }
   // If the current pixel is an intersection, then remove the
   // entering direction and start a thread in all the other directions.
   else if(pixVal<=8888)
   {
      int nextDig=1;
      int tmp=pixVal;

      pixVal=0;

      // remove the entering index from the intersection value
      while(tmp!=0) {
         // strip tmp which is the original value
         // if the smallest digit of tmp is not the
         // direction to be removed then store it in pixVal
         if(tmp%10!=8 && tmp%10!=remD[currT]) {
            pixVal += (tmp%10)*nextDig;
            nextDig *= 10;
         }
         tmp /= 10;
      }

      // Store the new intersection index
      imgPr.setPixel(currPt[currT], pixVal+10000+comp*100000);

      // Start a chain thread in all the other directions of the intersection
      spawnThreads(currT, pixVal);
      return false;
```

```
      }
      else
         return false;
   }

   public synchronized void spawnThreads(int currT, int pixVal) {
      int tmp = pixVal;

      // Strip each individual index from the pixVal and start a thread
      while(tmp!=0) {
         if((tmp%10)!=8 && !segT.isMapped(currPt[currT],(byte)(tmp%10)))   {
            while(!startThread(currPt[currT],(byte)(tmp%10),comp))
               t[currT].yield();
         }
         tmp /= 10;
      }
   }

   public int moveNext(int currT) {
      int newD = imgPr.getPixel(currPt[currT]);

      // if the current point is not an endpoint
      if(newD!=Const.ENDP && newD>7 && remD[currT]!=-1)
         newD = remD[currT];
      // if the point is a starting endpoint.
      else if(remD[currT]==-1)
         imgPr.setPixel(currPt[currT], newD+comp*100000);

      // Determine the x and y of the new point to move to
      int x = currPt[currT].x + dxdyTable[newD][0];
      int y = currPt[currT].y + dxdyTable[newD][1];

      currPt[currT].move(x,y);

      // Translate and store the removing direction
      remD[currT] = (byte)((newD+4)%8);
      return 0;
   }
}
```

B.6 Const.java

```
/*
 * Copyright 1997-1998 by Farzad Valad,
 * POB 8239 Longmont CO 80501
 * All rights reserved.
 */

import java.io.*;
import java.awt.*;

class Const
{
    public static final String PATH       = getPath();
    public static final Dimension WINRES  = Toolkit.getDefaultToolkit().getScreenSize();

    public static final int ENDP       = 1111;  //Marks and endpoint of a segement
    public static final int INTFLAG    = -4; //Marks a pixel as an intersection
    public static final int ENDFLAG    = -3; //Marks a pixel as an endpoint

    public static int IMGOFFSETX       = 0;  //X offset of the displaying image
    public static int IMGOFFSETY       = 0;  //Y offset of the displaying image

    public final static int FRAMEOFFSETX  = 20; //X offset of the displaying frame
    public final static int FRAMEOFFSETY  = 40; //Y offset of the displaying frame

    public final static int MAXPLYRS   = 21; //Maximum number of wave files used

    public final static int UPLEFT        =  8; //--|
    public final static int UP         =  9; // |
    public final static int UPRIGHT       = 10; // |
    public final static int RIGHT      = 11; // |
    public final static int DOWNRIGHT     = 12; // |
    public final static int DOWN       = 13; // |-- Directional index assigment
    public final static int DOWNLEFT   = 14; // |--   for use in Audio Player.
    public final static int LEFT       = 15; // |-- They also corespond to the
                                       // |-- wave file name stored on HD.
    public final static int BEGIN      = 16; // |
    public final static int INTERSECT     = 17; // |
    public final static int ENDPOINT   = 18; // |
    public final static int I2BRANCH   = 19; // |
    public final static int I3BRANCH   = 20; //--|

    public static PrintWriter debug       = null;  // Debugging flag
    public static Image icon           = null;

    //Pixel Scan Standard:          0 1 2
    //top left corner is 0 (clockwise) 7 X 3
    //dxdyTable[][]={0,1,2,3,4,5,6,7}  6 5 4
    public  static  final  int  dxdyTable[][]={{-1,-1},{0,-1},{1,-1},{1,0},{1,1},{0,1},{-1,1},{-
1,0}};

    // Method to print out the data array "arr", into the text file named "filename"
    // If comp flag is set print the component info of each pixel, other wise
    // if val flag is set print the chain-code values stored in each pixel.
    // else just just print a 1 when the pixel isn't a background pixel.
    public static void print(int[] arr, int iw, int ih, String filename, boolean val, boolean
comp) {
      try {
         BufferedWriter bw = new BufferedWriter(new FileWriter(filename));
         int num;
         int index;
         for(int i=0; i<ih; i++) {
            for(int j=0; j<iw; j++) {
               index = i*iw+j;
               if(arr[index]==0xff000000)
                  bw.write(" ");
               else if(val) {
                  num = arr[index]%10000;
```

```
                bw.write(((num>9)?"9":""+num));
             }
             else if(comp) {
                num = arr[index]/100000;
                bw.write(((num>9)?"9":""+num));
             }
             else {
                num = arr[index];
                bw.write("1");
             }
          }
          bw.newLine();
       }
       bw.flush();
     } catch (FileNotFoundException fnfe) {
     } catch (IOException ioe) {}
   }

   public static String getPath() {
     // create any file in current directory
     File currentDir = new File("x");
     // get the full path
     currentDir = new File(currentDir.getAbsolutePath());
     // and lose the fake filename.
     currentDir = new File(currentDir.getParent());

     return (new String("file:\\\\\\" + currentDir.toString() + "\\"));
   }

   public static void fatalError(String s) {
     System.out.println(s);
     System.exit(0);
   }
}
```

B.7 ImagePad.java

```
/*
 * Copyright 1997-1998 by Farzad Valad,
 * POB 8239 Longmont CO 80501
 * All rights reserved.
 */

import java.io.*;
import java.util.*;

public class ImagePad {

    // The total number of all pixels
    private int size;
    // Width of the image array
    private int width;
    // Height of the image array
    private int height;
    // Reference to the image array with valid chain codes
    private int img[];
    // The pad overlay array
    private int pad[];
    // List of all endpoints in the image
    private int elist[];
    // List of all intersections in the image
    private int ilist[];

    // Index of the current component
    private int comp;
    // Value of the current pixel
    private int pixVal;
    // Flag to determine if the Padder is in Vertical or Horizontal Mode
    private boolean vMode;

    public ImagePad(Vector data, int w, int h) {
      // Extract the image array
      this.img = (int[])data.elementAt(0);
      // Extract list of all endpoints of the segments in the image
      // Every two consequative pair represent the start and end of a segment
      this.elist = (int[])data.elementAt(1);
      // Extract and list of all intersections in the image
      this.ilist = (int[])data.elementAt(2);

      // Store dimentional information
      this.size = w*h;
      this.width = w;
      this.height = h;

      // Create the pad overlay array
      this.pad = new int[width*height];

      // Reset all the pixels to the valid white RGB number
      for(int i=0; i<size; i++)
         pad[i]=0xff000000;
    }

    public void run() {
      // Pad all the segments individually and store the pad information into the overlay
      createPad(elist);
      // Combine the original image with the its pad into one array
      overlay(pad,img);
      // Check and fix the intersections for any gaps around them
      chkIntersects();
    }

    private void chkIntersects() {
      int orgpnt=0;
      int index=0;
```

```
  // Loop around every intersection in the image
  for(int i=0; i<ilist.length; i++) {
     orgpnt = ilist[i];
     for(int j=1; j<8; j+=2) {
        index = orgpnt + Const.dxdyTable[j][0] + Const.dxdyTable[j][1]*width;

        // If there is a gap around the intersection
        if(img[index]==0xff000000)
           // Attempt to fix it
           fixPad(index);
     }
  }
}

// This method fills the gap around an intersection with
// a number that represent the majority of the repeated
// chain code around the gap
private void fixPad(int index) {
  int comp = -1;
  int pixVal = -1;
  int latest = -1;
  int[] hits = new int[8];

  // Survey and store the number of times a certain chain code
  // value is repeated around the gap
  for(int i=0; i<8; i++) {
     pixVal = img[index + Const.dxdyTable[i][0] + Const.dxdyTable[i][1]*width]%10000;
     if(0<=pixVal && pixVal<=7)
        hits[pixVal]++;
  }

  // Find the chain code that was repeated the most
  for(int i=0; i<8; i++) {
     pixVal = img[index + Const.dxdyTable[i][0] + Const.dxdyTable[i][1]*width]%10000;
     if(pixVal==i)
        if(latest==-1 || hits[latest]<hits[pixVal]) {
           latest = i;
           comp = img[index + Const.dxdyTable[i][0] + Const.dxdyTable[i][1]*width]/100000;
        }
  }

  // Fill the gap with selected chain code
  img[index] = latest + comp*100000;
}

private void createPad(int elist[]) {
  int st;
  int tmp1, tmp2;
  // padPos1 marks the current position of the pad on one side
  // padPos2 marks the current position of the pad on the other side
  int padPos1, padPos2;
  int count = 1;

  // Every two point in the list represent the start and end of a segment
  // that is why the index is increment by 2 every iteration.
  for(int i=0; i<elist.length; i+=2) {
     // mark the start of a segment
     st = elist[i];
     // extract the value of the chain code at that pixel
     pixVal = img[st]%10;
     // extract the component the segment belongs to
     comp = (img[st]/100000)*100000;

     // Determine the start mode of the padding processs
     // if there isn't a neighbor pixel on top and below the
     // current pixel then start in Vertical mode
     if(pixVal!=1 && pixVal!=5) {
        padPos1 = st - width;
        padPos2 = st + width;
```

```
   pad[padPos1]=pad[padPos2]=img[st];
   vMode = true;
}
// Otherwise start in Horizontal mode
else {
   padPos1 = st - 1;
   padPos2 = st + 1;

   pad[padPos1]=pad[padPos2]=img[st];
   vMode = false;
}

int tcount = 0;
int tmpPos = -1;

// while the end of the current segment is not reached
while(st!=elist[i+1]) {

   pixVal = img[st]%10;

   // Peak at the next pixels to move the pad to
   tmp1 = padPos1 + Const.dxdyTable[pixVal][0] + Const.dxdyTable[pixVal][1]*width;
   tmp2 = padPos2 + Const.dxdyTable[pixVal][0] + Const.dxdyTable[pixVal][1]*width;

   // if the segment is occupying the next pixel for side 1
   if(img[tmp1]!=0xff000000) {
      // move side 2 forward
      padPos2 = tmp2;
      pad[tmp2]=img[st];

      // Switch the mode from Vertical to Horizontal, or Horizontal to Vertical
      while(true) {
         // Switch the mode from V to H, or H to V
         tmpPos = switchMode(padPos2,padPos1);
         // Toggle the Vertical mode flag
         vMode = !vMode;

         // This case happens when the segment has steps
         // The pad positions must be adjusted to avoid
         // overwriting the component itself.  This
         // process continues until both sides of the pad
         // can move forward without overwriting the comp.
         if (tmpPos<0) {
            padPos2 = -1*tmpPos;
            pad[padPos1]=pixVal+comp;

            tmpPos = switchMode(padPos1,padPos2);
            vMode = !vMode;

            // If still the component will be over written
            if(tmpPos<0) {
               padPos1 = -1*tmpPos;
               pad[padPos2]=pixVal+comp;
               continue;
            }
            else {
               padPos1 = tmpPos;
               break;
            }
         }
         else {
            padPos2 = tmpPos;
            //vMode = !vMode;
            break;
         }
      }

      // After both pad positions are adjusted and obstacle free
      // determine the actual component pixel they surround.
      st = (padPos2 + padPos1)/2;
      pad[padPos1]=pad[padPos2]=img[st];
```

```
        }
        // if the segment is occupying the next pixel for side 2
        else if(img[tmp2]!=0xff000000) {
           // move side 1 forward
           padPos1 = tmp1;
           pad[tmp1]=img[st];

           // Switch the mode from Vertical to Horizontal, or Horizontal to Vertical
           while(true) {
              // Switch the mode from V to H, or H to V
              tmpPos = switchMode(padPos1,padPos2);
              // Toggle the Vertical mode flag
              vMode = !vMode;

              // This case happens when the segment has steps
              // The pad positions must be adjusted to avoid
              // overwriting the component itself.  This
              // process continues until both sides of the pad
              // can move forward without overwriting the comp.
              if(tmpPos<0) {
                 padPos1 = -1*tmpPos;
                 pad[padPos2]=pixVal+comp;

                 tmpPos = switchMode(padPos2,padPos1);
                 vMode = !vMode;
                 // If still the component will be over written
                 if(tmpPos<0) {
                    padPos2 = -1*tmpPos;
                    pad[padPos1]=pixVal+comp;
                    continue;
                 }
                 else {
                    padPos2 = tmpPos;
                    break;
                 }
              }
              else {
                 padPos1 = tmpPos;
                 break;
              }
           }

           // After both pad positions are adjusted and obstacle free
           // determine the actual component pixel they surround.
           st = (padPos2 + padPos1)/2;
           pad[padPos1]=pad[padPos2]=img[st];
        }
        // Otherwise both pad positions can move forward, advance them.
        else {
           padPos1 = tmp1;
           padPos2 = tmp2;

           st += Const.dxdyTable[pixVal][0] + Const.dxdyTable[pixVal][1]*width;
           pad[padPos1]=pad[padPos2]=img[st];
        }
     }
  }
}

private int switchMode(int padPos, int ref) {
  padPos += Const.dxdyTable[pixVal][0] + Const.dxdyTable[pixVal][1]*width;

  int pPx = padPos%width;
  int pPy = padPos/width;

  int refx = ref%width;
  int refy = ref/width;

  int dir = 0;
  int val = 0;
```

```
  if(vMode) {
     if(refy<pPy)
        val = 1;
     else
        val = 5;
  }
  else {
     if(refx<pPx)
        val = 7;
     else
        val = 3;
  }

  if(refy<pPy) {
     if(!vMode && refx>pPx)
        dir = 1;
     else
        dir = -1;
  }
  else {
     if(!vMode && refx<pPx)
        dir = -1;
     else
        dir = 1;
  }

  pad[padPos]=val+comp;

  int tmpPos = padPos;

  if(vMode)
     while(Math.abs(padPos-ref)!=2) {
        tmpPos += dir*width;

        if(img[tmpPos]!=0xff000000) {
           ref += -1*dir*width;
           pixVal = img[(padPos+ref)/2]%10;
           return -1*padPos;
        }

        padPos += dir*width;
        pad[padPos]=val+comp;
     }
  else
     while(Math.abs(padPos-ref)!=2*width) {
        tmpPos += dir*1;

        if(img[tmpPos]!=0xff000000) {
           ref += -1*dir*1;
           pixVal = img[(padPos+ref)/2]%10;
           return -1*padPos;
        }

        padPos += dir*1;
        pad[padPos]=val+comp;
     }
  return padPos;
}

private void overlay(int pad[], int img[]) {
  for(int i=0; i<size; i++) {
     if(img[i]==0xff000000)
        img[i]=pad[i];
  }
}

private void printPad(String filename) {
  try {
     int val;

     BufferedWriter bwp = new BufferedWriter(new FileWriter(filename));
```

```
            for(int i=0; i<height; i++) {
               for(int j=0; j<width; j++) {
                  val = pad[i*width+j];
                  if(val==0xff000000)
                     bwp.write(' ');
                  else if((val%10000)>7 || val==Const.ENDP)
                     bwp.write('9');
                  else
                     bwp.write("" + val%10);
               }
               bwp.newLine();
            }
            bwp.flush();
         } catch (FileNotFoundException fnfe) {
         } catch (IOException ioe) {}
      }

      private void print(String filename) {
         try {
            int val;

            BufferedWriter bwp = new BufferedWriter(new FileWriter(filename));
            BufferedWriter bwc = new BufferedWriter(new FileWriter("c" + filename));

            for(int i=0; i<height; i++) {
               for(int j=0; j<width; j++) {
                  val = img[i*width+j];
                  if(val==0xff000000) {
                     bwp.write(' ');
                     bwc.write(' ');
                  }
                  else if((val%10000)>7 || val==Const.ENDP) {
                     bwp.write('9');
                     bwc.write('9');
                  }
                  else {
                     bwp.write("" + val%10);
                     bwc.write("" + val/100000);
                  }
               }
               bwp.newLine();
               bwc.newLine();
            }
            bwp.flush();
            bwc.flush();
         } catch (FileNotFoundException fnfe) {
         } catch (IOException ioe) {}
      }

      private void printCoord(String str, int index) {
         System.out.print(str + "(" + index%64 + "," + index/64 + ") ");
      }
   }
```

B.8 ImagePilot.java

```
/*
 * Copyright 1997-1998 by Farzad Valad,
 * POB 8239 Longmont CO 80501
 * All rights reserved.
 */

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

/**
 * This class is the main object of the program.  This is the point where
 * everything starts.  This class is responsible for starting
 * the image processing, and intercepting mouse movements and keyboard input.
 */
class ImagePilot extends Frame implements ActionListener{
    private int iw;                    // The width of the image
    private int ih;                    // The height of the image
    private int length;
    private Image img;                 // The input image
    private int pixels[];              // The data array, contains all the audio info

    private String path = null;           // The install path

    private Image offImg = null;       // Off-screen image for double buffering
    private Graphics offG = null;         // Off-screen graphics for double buffering
    private Pixel lastMousePt = null;     // Last known mouse movement point

    private boolean showTrace = false;    // Flag to determine to show trace or not

    private Point clickPoint = null;   // The current point the user has clicked
    private ImageProcessor imgPr = null;  // Reference to the Image Processor module

    private static Vector compList;       // A list of all the independent components
    private static Pixel[] startPt = null;// A list of all the start points

    private AudioPlayer player = null;    // The audio player
    private ImageTracer imgtrcr = null;   // The Image Trace Guider

    public ImagePilot(String imageFile) {
      // Set the name of the main Frame
      super("ImagePilot [" + imageFile + "]");
      lastMousePt = new Pixel();

      // Get and open the image file
      getImage(imageFile);

      // Maximize the main frame to the current resolution
      setSize(Const.WINRES.width,Const.WINRES.height);

      // Set the properties of the main frame
      setCursor(CROSSHAIR_CURSOR);
      setIconImage(Const.icon);

      // Start the audio module and load in wave files
      player = new AudioPlayer();

      // Create a link list for saving all the segment tables.
      this.compList = new Vector(5,2);

      // if the debug flag is set to true, save debug info
      if(Const.debug!=null) {
         Const.print(pixels,iw,ih,"input.txt",false,false);
         Const.debug.println("INPUT.TXT:     contains the original image");
      }
```

```
  // Initialize and start the thinning module
  ImageThiner imgThin = new ImageThiner(pixels, iw, ih);
  imgThin.run();

  // if the debug flag is set to true, save debug info
  if(Const.debug!=null) {
     Const.print(pixels,iw,ih,"thined.txt",false,false);
     Const.debug.println("THINNED.TXT:   contians the thined image");
  }

  // Initialize and start the image processing module
  imgPr = new ImageProcessor(pixels, iw, ih, compList);
  imgPr.addActionListener(this);
}

private void addListeners() {
  // Add listener so that the main frame will close properly
  WindowListener wl = new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    };
  this.addWindowListener(wl);

  // Add mouse listener to intercept the mouse movements
  MouseListener ml = new MouseAdapter()
  {
     public void mouseExited(MouseEvent e) {
        imgtrcr.stop();
     }
     public void mouseEntered(MouseEvent e) {
        imgtrcr.trackPt(e.getX(),e.getY());
     }
     public void mousePressed(MouseEvent e) {
        showTrace = !showTrace;
        lastMousePt.move(e.getX(),e.getY());
     }
  };
  this.addMouseListener(ml);

  MouseMotionListener mml = new MouseMotionAdapter() {
     public void mouseMoved(MouseEvent e) {
        Point pt = e.getPoint();

        if(showTrace) {
           offG.drawLine(lastMousePt.x, lastMousePt.y, pt.x, pt.y);
           lastMousePt.setLocation(pt);
           repaint();
        }
        imgtrcr.trackPt(pt.x, pt.y);
     }
  };
  this.addMouseMotionListener(mml);

  // Add keyboard listener to intercept keyboard presses
  KeyListener kl = new KeyAdapter() {
     public void keyPressed(KeyEvent e) {
        imgtrcr.keyPressed(e);
     }
  };
  this.addKeyListener(kl);
}

// After processing is complete create a list
// of the start points for each of the different
// regions for tracing.
public static void createStartList() {
  // Determine the number of distinct regions
  int vecSize = compList.size();
```

```
  // Create a list to store the start points
  startPt = new Pixel[vecSize];

  // store the start points into the array
  for(int i=0; i<vecSize; i++) {
     startPt[i] = new Pixel(((SegTable)compList.elementAt(i)).bestStart());
  }
}

// Method to get and open the image file
private void getImage(String fileName) {
  // Attempt to open the file from the hard drive
  try {
     img = Toolkit.getDefaultToolkit().getImage(fileName);
     // Attach a media tracker to determine if the file loaded
     MediaTracker t = new MediaTracker(this);
     t.addImage(img, 0);
     t.waitForID(0);         // Waits until the image is loaded
  }
  catch(InterruptedException ie){}

  // get the height and width of the image file
  iw = img.getWidth(null);
  ih = img.getHeight(null);

  // Detemine the image display offset and store them
  Const.IMGOFFSETX = (Const.WINRES.width-iw)/2;
  Const.IMGOFFSETY = (Const.WINRES.height-ih)/2;

  length = iw*ih;
  // create the traceable data array
  pixels = new int[iw*ih];

  try {
     ImageProducer producer = img.getSource();

     // Running a Black & White filter that inverts the values.  All blacks
     // become white and all whites become black
        producer = new FilteredImageSource(producer, new BWInvFilter());
      //filteredImage = Toolkit.getDefaultToolkit().createImage(producer);

     // Load the pixels array with the pixels in the image.
     PixelGrabber pg = new PixelGrabber(producer,0,0,iw,ih,pixels,0,iw);
     pg.grabPixels();
  } catch(InterruptedException ie){};
}

public void paint(Graphics g) {
  update(g);
}

public void update(Graphics g) {
  if(offG==null) {
     offImg = createImage(Const.WINRES.width,Const.WINRES.height);
     offG = offImg.getGraphics();
     offG.setColor(getBackground());
     offG.fillRect(0, 0, Const.WINRES.width, Const.WINRES.height);

     offG.setColor(Color.black);
     offG.drawImage(img,Const.IMGOFFSETX,Const.IMGOFFSETY,this);

     offG.setColor(Color.red);
  }
  g.drawImage(offImg,0,0,this);
}

// Intercept action performed event from Image tracer to
// determine when processing is done.
public void actionPerformed(ActionEvent e) {
  createStartList();
  if(Const.debug!=null) {
```

```
         System.out.println();
         Const.debug.close();
      }
      imgtrcr = new ImageTracer(pixels,compList,startPt,iw,ih,player);
      addListeners();
      // Display the main frame
      setVisible(true);
   }

   // Main method of the program that is called first
   public static void main(String args[]) {
      if(args.length==0 || args.length>2) {
         System.out.println();
         System.out.println("Usage: Java ImagePilot <gif> <trace>");
         System.out.println("-<gif>:      The input file must be a valid gif image file name.");
         System.out.println("-<trace>:     type true to get a txt file output at each phase of
processing.");
         System.out.println();
         System.exit(0);
      }
      else if(args.length==2) {
         FileOutputStream fdout = new FileOutputStream(FileDescriptor.out);
            BufferedOutputStream bos = new BufferedOutputStream(fdout, 1024);
         Const.debug = new PrintWriter(bos, false);
      }
      Logo logo = new Logo();

      ImagePilot window = new ImagePilot(args[0]);
      logo.dispose();
   }
}
```

B.9 ImageProcessor.java

```
/*
 * Copyright 1997-1998 by Farzad Valad,
 * POB 8239 Longmont CO 80501
 * All rights reserved.
 */

import java.awt.*;
import java.util.*;
import java.awt.event.*;

/**
 * This class handles processing a pixel array that has all the black pixels marked as
 * -1 (0xFFFFFFFF) and all the white pixels marked as 0 (0xFF000000).It will then analyze
 * the entire image and find segements that start and end with and endPoint (-3) or
 * intersection (-4).  Along the way creating a segment table (SegTable).
 */
public class ImageProcessor implements Runnable {
    transient ActionListener actionListener;

    private Thread t;

    private int iw;           // Width of the data array
    private int ih;           // Height of the data array
    private int[] pixels;
    private Vector segList;   // A refrence to the component list in ImagePilot

    public SegTable segT;     // The list of segments for the current component

    //Pixel Scan Standard:          0 1 2
    //top left corner is 0 (clockwise) 7 X 3
    //dxdyTable[][]={0,1,2,3,4,5,6,7}  6 5 4
    private int dxdyTable[][]={{-1,-1},{0,-1},{1,-1},{1,0},{1,1},{0,1},{-1,1},{-1,0}};

    // Initialize and store passed values
    public ImageProcessor(int[] pixels, int width, int height, Vector segList) {
      this.iw = width;
      this.ih = height;
      this.pixels = pixels;   // The data array

      this.segT = null;
      this.segList = segList;

      t = new Thread(this);
      t.start();
    }

    public void run() {
      // Start horizontal scan
      hScanImage();

      // if the debug flag is set to true, save debug info
      if(Const.debug!=null) {
         Const.print(pixels,iw,ih,"imgPrComp.txt",false,true);
         Const.print(pixels,iw,ih,"imgPrVal.txt",true,false);
         Const.debug.println("IMGPRVAL.TXT:  contains the pixel values after chaining");
         Const.debug.println("IMGPRCOMP.TXT:   contains   the   pixel   component   values   after
chaining");
      }

      // Dialate the image to three pixel width segments
      padImage();

      // if the debug flag is set to true, save debug info
      if(Const.debug!=null) {
         Const.print(pixels,iw,ih,"padComp.txt",false,true);
         Const.print(pixels,iw,ih,"padVal.txt",true,false);
         Const.debug.println("PADVAL.TXT:    contains the pixel values after padding");
```

```
      Const.debug.println("PADCOMP.TXT:   contains the pixel component values after padding");
    }

    // Tell ImagePilot that processing is complete
    reportDone();
  }

  public synchronized void addActionListener(ActionListener l) {
    actionListener = AWTEventMulticaster.add(actionListener, l);
  }

  public synchronized void removeActionListener(ActionListener l) {
    actionListener = AWTEventMulticaster.remove(actionListener, l);
  }

  protected void reportDone() {
      if (actionListener != null) {
          actionListener.actionPerformed(new ActionEvent(this,0,"ImageProcessor"));
      }
  }


  private void padImage() {
    // Create a list of the data to be passed
    Vector data = new Vector(3,1);

    int[] elist = createPadList();
    int[] ilist = getIntList();

    // Store a reference of the data to be passed
    data.addElement(pixels);
    data.addElement(elist);
    data.addElement(ilist);

    // Start dialating the image to three pixels
    ImagePad ip = new ImagePad(data, iw, ih);
    ip.run();
  }

  // This method create the list of all the end points of
  // the region segments to be padded then returns that
  // array.
  private int[] createPadList() {
    int[] padList = null;
    int[] tmpList = null;
    int[] tmpOldList = null;

    SegTable tmpSegT = null;
    int count = segList.size();

    for(int i=0; i<count; i++) {
       tmpSegT = (SegTable) segList.elementAt(i);
       tmpList = tmpSegT.getPadEnds(iw);
       tmpOldList = padList;

       if(padList==null) {
          padList = tmpList;
          continue;
       }
       else {
          padList = new int[tmpList.length + tmpOldList.length];
       }

       System.arraycopy(tmpOldList,0,padList,0,tmpOldList.length);
       System.arraycopy(tmpList,0,padList,tmpOldList.length,tmpList.length);
    }
    return padList;
  }

  // This method reports a list of all the intersection
  // in the image for all the regions.
```

```
private int[] getIntList() {
  int[] tmplist = null;
  SegTable tmpSegT = null;
  int count = segList.size();

  int iCount = 0;
  Vector listVec = new Vector(5,5);

  for(int i=0; i<count; i++) {
     tmpSegT = (SegTable) segList.elementAt(i);
     tmplist = tmpSegT.getIntList(iw);
     if(tmplist==null)
        continue;
     else {
        iCount += tmplist.length;
        listVec.addElement(tmplist);
     }
  }

  count =0;
  int []ilist = new int[iCount];

  for(Enumeration e = listVec.elements(); e.hasMoreElements();) {
     tmplist = (int[])e.nextElement();
     System.arraycopy(tmplist,0,ilist,count,tmplist.length);
     count += tmplist.length;
  }
  return ilist;
}

// This function does a horizontal scan of all the pixel in the passed pixel array.
// Upon finding a black pixel, it extracts that entire component using an image thread
// then chains the component using a chain thread.  The process of extracting components
// and chaing them is sequential, although ImageThread and ChainThread are multi threaded
// classes.
private void hScanImage() {
  int count = 0;
  int endCount = 0;
  int intersectCount = 0;
  ChainThread chnThread = new ChainThread(this, 25);
  ImageThread imgThread = new ImageThread(this, 25);

  for(int i=0; i<ih*iw; i++) {
     if(pixels[i]==0xffffffff) {
        this.segT = new SegTable(this);

        while(!imgThread.startThread(new Pixel(i%iw,i/iw),(byte)-1,null,(byte)-1))
           t.yield();

        while(imgThread.threadCount>0)
           t.yield();

        segList.addElement((Object)segT);

        chnThread.setSegTable(segT);
        while(!chnThread.startThread(segT.bestStart(),(byte)-1,count++))
           t.yield();

        while(chnThread.threadCount>0)
           t.yield();

        endCount += segT.getEndPoints();
        intersectCount += segT.getIntersects();
        //System.out.println(segT);
     }
  }

  System.out.println("Ints: " + intersectCount + " Ends: " + endCount);
  System.out.println("Horizontal Scan Done.");
}
```

```
public int getPixel(Pixel pt) {
  return (pixels[pt.y*iw+pt.x]%100000);
}

public synchronized boolean setPixel(Pixel pt, int newVal) {
  try   {
     if(pixels[pt.y*iw+pt.x]!=newVal) {
        pixels[pt.y*iw+pt.x]=newVal;
        return true;
     }
     else
        return false;

  }catch(ArrayIndexOutOfBoundsException aob){return false;};
}

// This function returns the type of a particular pixel.  If going around a
// pixel, it has a 01 or 10 sequence of white and blacks, then it is an endpoint.
// If it has a 1010 or 0101 sequence then it is a black pixel (mid segment), and
// finally if it has a 10101 or 1010101 sequence then it is an intersection.
public byte getType(Pixel pt) {
  int x, y;
  int state=0, count=0, start=0, next=0;

  while(count<8 && start<8) {
     x = pt.x + dxdyTable[(count+start)%8][0];
     y = pt.y + dxdyTable[(count+start)%8][1];

     if ( x>=0 && x<=(iw-1) && y>=0 && y<=(ih-1)) {
        try   {
           if(pixels[y*iw+x]!=0xff000000) {
              //Make sure that the first pixel to check is a white pixel
              if(count==0) {
                 start++;
                 continue;
              }

              if(state==0 || state==2 || state==4)
                 state++;
           }
           else {
              if(state==1 || state==3)
                 state++;
           }
           count++;

        } catch(ArrayIndexOutOfBoundsException aob){continue;};
     }
     else {
        //If the checking pixel is out side the array, then
        //pretend it is white.
        if(state==1 || state==3)
           state++;
        count++;
     }
  }

  return ((state>=5)?(byte)-4:((state<3)?(byte)-3:(byte)-2));
}

// Returns how many seperate black neighbors are around a pixel
// and if array is not null it will load it with the neighbors
// The main logic is identical to getType (above).
public byte getNeighbors(Pixel pt, char array[]) {
  int x, y;
  int state=0, count=0, start=0;

  while(count<8 && start<8) {
     x = pt.x + dxdyTable[(count+start)%8][0];
     y = pt.y + dxdyTable[(count+start)%8][1];
```

```
         if (x>=0 && x<=(iw-1) && y>=0 && y<=(ih-1)) {
            try   {
               if(array!=null && (start+count)<8)
                  //Load a 1 in the array for black pixels and 0 for white pixels. 2 should never
be loaded.
                  array[count+start]                                                              =
((pixels[y*iw+x]==0xffffffff)?'1':((pixels[y*iw+x]==0xff000000)?'0':'2'));

               if(pixels[y*iw+x]!=0xff000000) {
                  //Make sure that the first pixel to check is a white pixel
                  if(count==0) {
                     start++;
                     continue;
                  }

                  if(state==0 || state==2 || state==4)
                     state++;
               }
               else {
                  if(state==1 || state==3)
                     state++;
               }
               count++;

            } catch(ArrayIndexOutOfBoundsException aob){continue;};
         }
         else {
            //If the checking pixel is out side the array, then
            //pretend it is white.
            if(state==1 || state==3)
               state++;
            count++;
         }
      }

      return ((state>=5)?(byte)-4:((state<3)?(byte)-3:(byte)-2));
   }

   //A pixel is an intersection if there are more than two segements of seperate black neighbors
   public boolean isIntersect(Pixel pt) {
      return (getType(pt)==-4);
   }

   //A pixel is an endpoint if there is only one segement of black neighbors
   public boolean isEnd(Pixel pt) {
      return (getType(pt)==-3);
   }

   // Method that increments the intersection count in the segment table
   public void incIntCount(Pixel pt) {
      segT.incIntersect(pt);
   }

   // Method that increments the endpoint count in the segment table
   public void incEndCount(Pixel pt) {
      segT.incEndPoints(pt);
   }
}
```

B.10 ImageThinner.java

```
import java.io.*;

public class ImageThiner {
    private int height;
    private int width;
    private int image[];

    private int y[];

    public ImageThiner(int[] img, int iw, int ih) {
      height = ih;
      width = iw;
      y = new int[height*width];
      image = img;
      convertImage(true);
    }

    private void convertImage(boolean flag) {
      if(flag) {
         for(int i=0; i<height; i++)
            for(int j=0; j<width; j++)
               if(image[i*width+j] == -1)
                  image[i*width+j] = 1;
               else
                  image[i*width+j] = 0;
      }
      else {
         for(int i=0; i<height; i++)
            for(int j=0; j<width; j++)
               if(image[i*width+j] == 1)
                  image[i*width+j] = -1;
               else
                  image[i*width+j] = 0xff000000;
      }
    }

    private void updateImage(int a[],int b[])
    {
      for(int i=0; i<height; i++)
         for(int j=0; j<width; j++)
            b[i*width+j] -= a[i*width+j];
    }

    private int analyzeNeighbors(int i,int j,int a[],int b[])
    {
      for (int n=0; n<8; n++) a[n] = 0;

      if (i-1 >= 0) {
         a[0] = image[(i-1)*width + j];
         if (j+1 < width) a[1] = image[(i-1)*width + j+1];
         if (j-1 >= 0) a[7] = image[(i-1)*width + j-1];
      }

      if (i+1 < height) {
         a[4] = image[(i+1)*width + j];
         if (j+1 < width) a[3] = image[(i+1)*width + j+1];
         if (j-1 >= 0) a[5] = image[(i+1)*width + j-1];
      }

      if (j+1 < width) a[2] = image[i*width + j+1];
      if (j-1 >= 0) a[6] = image[i*width + j-1];

      int m = 0;
      b[0] = 0;
      for (int n=0; n<7; n++) {
         if ((a[n]==0) && (a[(n+1)%8]==1)) m++;
```

```
      b[0] = b[0] + a[n];
   }
   if ((a[7] == 0) && (a[0] == 1)) m++;
   b[0] = b[0] + a[7];
   return m;
}

public void run() {
   int i,j,ar,p1,p2;
   int a[] = new int[8];
   int br[] = new int[1];
   int count=0;
   boolean cont = true;

   do {
      count++;
      cont = false;

      for (i=0; i<height; i++)
        for (j=0; j<width; j++) {
         if (image[i*width+j] == 0) {
            y[i*width+j] = 0;
            continue;
         }
         ar = analyzeNeighbors(i, j, a, br);
         p1 = a[0]*a[2]*a[4];
         p2 = a[2]*a[4]*a[6];
         if ( (ar == 1) && ((br[0]>=2) && (br[0]<=6)) &&
            (p1 == 0) && (p2 == 0) )  {
               y[i*width+j] = 1;
               cont = true;
         }
         else y[i*width+j] = 0;
      }
      updateImage(y, image);

      for (i=0; i<height; i++)
        for (j=0; j<width; j++) {
         if (image[i*width+j] == 0) {
            y[i*width+j] = 0;
            continue;
         }
         ar = analyzeNeighbors(i, j, a, br);
         p1 = a[0]*a[2]*a[6];
         p2 = a[0]*a[4]*a[6];
         if ( (ar == 1) && ((br[0]>=2) && (br[0]<=6)) &&
            (p1 == 0) && (p2 == 0) )  {
               y[i*width+j] = 1;
               cont = true;
         }
         else y[i*width+j] = 0;
      }
      updateImage(y, image);
   }while(cont);

   do {
      count++;
      cont = false;

      for (i=0; i<height; i++)
        for (j=0; j<width; j++) {
         if (image[i*width+j]==0) {
            continue;
         }
         ar = analyzeNeighbors(i, j, a, br);
         p1 = a[0]*a[2] + a[2]*a[4] + a[4]*a[6] + a[6]*a[0];
         p2 = a[2]*a[3]*a[4];

         if( (ar==2 && br[0]<=3 && p1==1) || (p2==1) ){
               image[i*width+j] = 0;
               cont = true;
```

```
            }
            else image[i*width+j] = 1;
         }
      }while(cont);

      System.out.println("A total of " + count + " iterations ran");
      convertImage(false);

   }
}
```

B.11 ImageThread.java

```
/*
 * Copyright 1997-1998 by Farzad Valad,
 * POB 8239 Longmont CO 80501
 * All rights reserved.
 */

import java.awt.*;

class ImageThread implements Runnable
{
    public static ImageProcessor imgPr;   // A reference to the image processor
    public static int threadCount = 0;    // Number of threads running

    public static int segCount = 0;       // The index of the current segment

    private Thread[] t;
    private int maxThreads;      // Maximum number of threads allowed
    private Segment[] segList;   // Reference to the master seg table list
    private Pixel[] fromPt;      // Used to trace a single segment

    private ThreadGroup tGrp = new ThreadGroup("Threads");

    //Pixel Scan Standard:          0 1 2
    //top left corner is 0 (clockwise) 7 X 3
    //dxdyTable[][]={0,1,2,3,4,5,6,7}  6 5 4
    private  static  final  byte  dxdyTable[][]={{-1,-1},{0,-1},{1,-1},{1,0},{1,1},{0,1},{-1,1},{-
1,0}};

    public ImageThread(ImageProcessor imgPr, int maxThreads) {
      this.imgPr = imgPr;
      this.maxThreads = maxThreads;
      this.segList = new Segment[maxThreads];
      this.fromPt = new Pixel[maxThreads];
      this.t = new Thread[maxThreads];
    }

    public boolean isStarted() {
      if(t[0]!=null)
         return t[0].isAlive();
      else
         return false;
    }

    public boolean isAlive() {
      return (tGrp.activeCount()!=0);
    }

    public boolean startThread(Pixel start, byte type, Pixel end, byte d) {
      int i;
      // Find a open slot in the thread array t.
      synchronized(this) {
         for(i=0; i<maxThreads && t[i]!=null; i++);
      }

      // if table is full return.
      if(i==maxThreads)
         return false;
      // else initialize data structures.
      else {
         this.segList[i] = new Segment();

         this.segList[i].start.move(start);
         this.segList[i].startType = type;
         this.segList[i].startD = d;

         if(type!=-1)
            this.segList[i].end.move(end);
```

```
      else
         this.segList[i].end.move(start);

      this.segList[i].endType = 0;
      this.segList[i].endD = (byte)((d+4)%8);

      if(this.fromPt[i]==null)
         this.fromPt[i] = new Pixel(start);
      else
         this.fromPt[i].move(start);

      this.t[i] = new Thread(tGrp, this, Integer.toString(i));
      t[i].start();

      synchronized(this) {
         threadCount++;
      }

      return true;
   }
}

public void run() {
   // Get the index of the thread that invoked this method
   int currT = Integer.parseInt(Thread.currentThread().getName());
   // Create a data array for searching around a pixel
   char[] blackPt = new char[8];

   do {
      // Locate the neighbors around a pixel and store the type of that pixel
      segList[currT].endType = imgPr.getNeighbors(segList[currT].end, blackPt);
   // Determine what to do next
   }while(analyzePixel(currT, blackPt));

   // Report your segment to the segment table and wait until
   // segment table has loaded the segment you found.
   while(!imgPr.segT.loadSeg(segList[currT]))
      t[currT].yield();

   t[currT] = null;
   segList[currT] = null;

   synchronized(this) {
      threadCount--;
   }
}

// This method determines what to do next.
private boolean analyzePixel(int currT, char[] blackPt) {
   int i;
   int pixelVal = filterArray(blackPt);

   // If the current point in question is an intersection
   if(segList[currT].endType==-4) {
      //System.out.println("Intersection at " + segList[currT].end);

      // Increment the intersection counter.
      imgPr.incIntCount(segList[currT].end);

      // If no other thread has visited this intersection
      // spawn thread in the diffirent directions from this intersection
      // .setPixel() returns true or false depending on if the intersection
      // hasn't or has been visited.
      if(imgPr.setPixel(segList[currT].end,pixelVal)) {
         spawnThreads(currT, blackPt);
      }
      return false;
   }
   // else if the current point is an endpoint
   else if(segList[currT].endType==-3) {
      //System.out.println("End Point at " + segList[currT].end);
```

```
         // increment the endpoint counter
         imgPr.incEndCount(segList[currT].end);
         // store the endpoint information into the main
         // array inside the image processor.
         imgPr.setPixel(segList[currT].end,pixelVal);

         // If the endpoint is the first pixel encountered
         if(segList[currT].start.equals(segList[currT].end))   {
            segList[currT].startType = -3;
            return !deadEnd(currT, blackPt);
         }
         else
            return false;
      }

      // else if the current pixel is a mid pixel
      else if(segList[currT].endType==-2) {
         segList[currT].pixelCount++;
         imgPr.setPixel(segList[currT].end,pixelVal);

         if(segList[currT].start.equals(segList[currT].end)) {
            spawnThreads(currT, blackPt);
            return false;
         }
         else {
            return !deadEnd(currT, blackPt);
         }
      }

      // No man's land! The code should never enter this else part.
      else {
         System.out.println(currT + " is confused. ");
         return false;
      }
   }

   private boolean deadEnd(int currT, char[] blackPt) {
      byte i;
      fromPt[currT].move(segList[currT].end.x, segList[currT].end.y);
      //filterArray(blackPt);

      for(i=0; i<8; i++)
         if(blackPt[i]=='1')
            break;
      if(i==8)
         return true;

      if(segList[currT].startD==-1)
         segList[currT].startD=i;
      segList[currT].endD=(byte)((i+4)%8);

   segList[currT].end.move(segList[currT].end.x+dxdyTable[i][0],segList[currT].end.y+dxdyTable[i
][1]);
      return false;
   }

   // '0' means that pixel is empty.
   // '1' means that pixel has been analyzed yet
   // '2' means that pixel has been analyzed and it is
   // either an intersection (-4) or endpoint (-3) or just
   // a black pixel (-2).
   private int filterArray(char[] n) {
      int pixelVal=0, nextDig=1;

      if(n[1]!='0')
         n[0] = n[2] = '0';

      if(n[3]!='0')
         n[2] = n[4] = '0';
```

```
      if(n[5]!='0')
         n[4] = n[6] = '0';

      if(n[7]!='0')
         n[0] = n[6] = '0';

      for(int i=0; i<8; i++) {
         if(n[i]!='0') {
            pixelVal += i*nextDig;
            nextDig *= 10;
         }
      }

      if(n[1]=='2')
         n[1]='0';

      if(n[3]=='2')
         n[3]='0';

      if(n[5]=='2')
         n[5]='0';

      if(n[7]=='2')
         n[7]='0';

      return pixelVal;
   }

   //n is the array of neighbors.  In order to spawn    0 1 2
   //a thread in the diagnol directionboth the       7 X 3
   //flush neighbors of that diagnol pixel must be blank.  6 5 4
   private void spawnThreads(int currT, char[] n) {
      for(byte i=0; i<8; i++) {
         if(n[i] == '1')
            startThread(segList[currT].end,              segList[currT].endType,              new
Pixel(segList[currT].end.x+dxdyTable[i][0],segList[currT].end.y+dxdyTable[i][1]), i);
      }
   }
}
```

B.12 ImageTracer.java

```
/*
 * Copyright 1997-1998 by Farzad Valad,
 * POB 8239 Longmont CO 80501
 * All rights reserved.
 */

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

class ImageTracer implements ActionListener{
    private int iw = 0;             // width of the data array
    private int ih = 0;             // height of the data array

    private int currComp = 0;       // The current component being traced
    private int lostCount = 0;

    private Pixel lastMPt = null;   // Last known mouse coordinate audio was played for
    private int currX = 0;          // The current X coordinate of the mouse
    private int currY = 0;          // The current Y coordinate of the mouse

    private int[] pixels;
    private Pixel currPt = null;    // Current point that audio is played for
    private Pixel[] startPt = null;    // The starting point of the current segment
    private Vector compList = null;    // A list of all the independent components

    private boolean verbal = false;    // Flag to determine the audio mode, tone or verbal
    private boolean tracing = false;// Flag to know when the audio players are loaded
    private AudioPlayer player = null; // Reference to the audio player module

    public ImageTracer(int[] pixels, Vector compList, Pixel[] startPt, int width, int height,
AudioPlayer ap) {
      this.compList = compList;
      this.startPt = startPt;

      this.iw = width;
      this.ih = height;
      this.pixels = pixels;

      this.player = ap;
      player.addActionListener(this);

      this.lastMPt = new Pixel(-1,-1);
      this.currPt = new Pixel(startPt[currComp]);
    }

    public void actionPerformed(ActionEvent e) {
      // if the last known point is not the same as the current point
      if(!lastMPt.equals((Pixel)e.getSource()))
         trackPt(lastMPt.x, lastMPt.y);
    }

    public void stop() {
      player.stop();
    }

    // This method moves to the next region.
    public void nextComp() {
      if(currComp<compList.size()-1) {
         currComp++;
         tracing = false;
      }
      else {
         currComp = 0;
         imageDone();
```

```
      }
      currPt.move(startPt[currComp]);
    }

    // Sets the number of the current region
    // to be traced next.
    public void setComp(int num) {
      currComp = num;
      currPt.move(startPt[currComp]);
    }

    private void imageDone() {
      System.out.println("Tada!");
      for(int i=0; i<compList.size(); i++)
         System.out.println("Component " + i + ": " + ((SegTable)compList.elementAt(i)).score());
    }

    // This method is responsible for determining what audio file to be played by
    // the player.
    public void trackPt(int x, int y) {
      // Store the coordinates of the current
      // point in the array.
      currX = x-Const.IMGOFFSETX;
      currY = y-Const.IMGOFFSETY;
      // Store the last mouse point received
      lastMPt.move(x,y);

      // if system is in tracing mode and the user is not lost
      if(tracing && lostCount<25) {
         // if the current point is within the image and the black
         // pixle belongs to the current black region
         if((currY>=0          &&          currY<ih          &&          currX>=0          &&
currX<iw)&&(currComp==getComp((currY)*iw+(currX)))) {
            currPt.move(currX,currY);
            lostCount = 0;
            int type = getType((currY)*iw+(currX));

            // if the current point type is an endpoint or intersection
            if(type>7) {
               if(!((SegTable)compList.elementAt(currComp)).markOff(currX,currY))
                  nextComp();
               if(type==Const.ENDP)
                  player.playerStart(Const.ENDPOINT,lastMPt);
               else if(type>100)
                  player.playerStart(Const.I3BRANCH,lastMPt);
               else
                  player.playerStart(Const.I2BRANCH,lastMPt);
            }
            // otherwise play the corresponding chain-code
            // wave file to the user.
            else {
               player.playerStart((verbal?type+8:type),lastMPt);
            }
         }
         else {
            // the user is lost and keep track of the count.
            lostCount++;
            player.stop();
         }
      }
      else {
         // The user hasn't found the starting point yet,
         // guide them in that direction.
         findPixel(currPt);
      }
    }

    // Given a point destPt, this method guides the user to it
    private void findPixel(Pixel destPt) {
      if(Math.abs(currY-destPt.y)<=1          &&          Math.abs(currX-destPt.x)<=1          &&
getType((currY)*iw+(currX))>=0) {
```

```
      int type = getType((currY)*iw+(currX));
      player.playerStart((verbal?type+8:type),lastMPt);
      ((SegTable)compList.elementAt(currComp)).markOff(destPt.x,destPt.y);
      lostCount = 0;
      tracing = true;
   }
   else
      tellDirection(destPt);
}

// Play the corresponding audio file to tell the
// user which direction they need to travel
// relative to their current position.
private void tellDirection(Pixel destPt) {
   if(Math.abs(currY-destPt.y)<=1)  {

      if(currX<destPt.x)
         player.playerStart(Const.RIGHT,lastMPt);
      else if(currX>destPt.x)
         player.playerStart(Const.LEFT,lastMPt);
      else if(currX==destPt.x) {

         if(currY<destPt.y)
            player.playerStart(Const.DOWN,lastMPt);
         else if(currY>destPt.y)
            player.playerStart(Const.UP,lastMPt);
         else{
            int type = getType((currY)*iw+(currX));
            player.playerStart((verbal?type+8:type),lastMPt);
            ((SegTable)compList.elementAt(currComp)).markOff(currX,currY);
            lostCount = 0;
            tracing = true;
         }
      }
   }
   else if(currY<destPt.y) {
      if(Math.abs(currX-destPt.x)<=1)
         player.playerStart(Const.DOWN,lastMPt);
      else if(currX>destPt.x)
         player.playerStart(Const.DOWNLEFT,lastMPt);
      else if(currX<destPt.x)
         player.playerStart(Const.DOWNRIGHT,lastMPt);
   }
   else if(currY>destPt.y) {
      if(Math.abs(currX-destPt.x)<=1)
         player.playerStart(Const.UP,lastMPt);
      else if(currX>destPt.x)
         player.playerStart(Const.UPLEFT,lastMPt);
      else if(currX<destPt.x)
         player.playerStart(Const.UPRIGHT,lastMPt);
   }
}

public int getType(int index) {
   return ((pixels[index]==Const.ENDP)?Const.ENDP:(pixels[index]%10000));
}

public int getComp(int index) {
   return (pixels[index]/100000);
}

public void keyPressed(KeyEvent e) {

   char c = e.getKeyChar();

   // Move to the next component
   if(c=='n' || c=='N') {
      nextComp();
   }

   // Reset the whole image
```

```
      else if(c=='r' || c=='R') {
         SegTable tmpTable;
         setComp(0);
         tracing = false;

         for(int i=0; i<compList.size(); i++) {
            tmpTable = ((SegTable)compList.elementAt(i));

            tmpTable.resetCount();
            tmpTable.resetFlags();
         }
      }

      // Reset the current component
      else if(c=='t' || c=='T') {
         tracing = false;

         SegTable tmpTable = ((SegTable)compList.elementAt(currComp));

         tmpTable.resetCount();
         tmpTable.resetFlags();
      }
      // Done with the image
      else if(c=='d' || c=='D') {
         imageDone();
      }
      // If in self test mode stop,
      // otherwise starting doing the self test routine.
      else if(c=='p' || c=='P') {
         player.toggleSelfTest();
      }
      // switch between verbal and non-verbal feedback.
      else if(c=='v' || c=='V') {
         verbal = !verbal;
      }
   }
}
```

B.13 Logo.java

```
/*
 * Copyright 1997-1998 by Farzad Valad,
 * POB 8239 Longmont CO 80501
 * All rights reserved.
 */

import java.awt.*;

/*
 * This class displays the initial splash screen
 * and the logo and version information while
 * the program is processing the image file.
 */

class Logo extends Frame {
    private int width;
    private int height;
    private Image logo = null;

    public Logo() {
      super("VIRGINIA TECH");
      getImage("res\\logo.bin");
      // Center the logo frame to be in the center of the current resoloution
      setBounds((Const.WINRES.width-width)/2,(Const.WINRES.height-height)/2,width,height);
      setCursor(WAIT_CURSOR);
      setVisible(true);
      repaint();
    }

    private void getImage(String fileName) {
      try {
         logo = Toolkit.getDefaultToolkit().getImage(fileName);
         Const.icon = Toolkit.getDefaultToolkit().getImage("res\\vt.bin");

         MediaTracker t = new MediaTracker(this);
         t.addImage(logo, 0);
         t.addImage(Const.icon, 0);
         t.waitForID(0);         // Waits until the images are loaded
         setIconImage(Const.icon);
      }
      catch(InterruptedException ie){}

      width = logo.getWidth(null) + Const.FRAMEOFFSETX;
      height = logo.getHeight(null) + Const.FRAMEOFFSETY;
    }

    public void paint(Graphics g) {
      Update(g);
    }

    public void Update(Graphics g) {
      g.drawImage(logo,9,29,this);
    }
}
```

B.14 Pixel.java

```
/*
 * Copyright 1997-1998 by Farzad Valad,
 * POB 8239 Longmont CO 80501
 * All rights reserved.
 */

import java.awt.*;

/*
 * This class represents a data structure for
 * representing a point in the x-y coordinate
 * system, with all the needed methods to perform
 * certain related tasks.
 */

class Pixel extends Point
{
    public Pixel() {
      x = -1;
      y = -1;
    }

    public Pixel(Pixel pt) {
      x = pt.x;
      y = pt.y;
    }

    public Pixel(int orgX, int orgY) {
      x = orgX;
      y = orgY;
    }

    public void translate(int direction) {
      x += Const.dxdyTable[direction][0];
      y += Const.dxdyTable[direction][1];
    }

    public int getIndex(int width) {
      return (y*width+x);
    }

    public void move(Pixel pt) {
      x = pt.x;
      y = pt.y;
    }

    public boolean isEmpty() {
      return (x==-1 && y==-1);
    }

    public void clear() {
      x = -1;
      y = -1;
    }

    public boolean isGreater(Pixel compTo) {
      return ((x==compTo.x)?(y>compTo.y):(x>compTo.x));
    }

    public String toString() {
      return ("(" + Integer.toString(x) + "," + Integer.toString(y) + ")");
    }
}
```

B.15 Segment.java

```
/*
 * Copyright 1997-1998 by Farzad Valad,
 * POB 8239 Longmont CO 80501
 * All rights reserved.
 */

/**
 * This class represent the data structure that stores info for
 * a single segment.  A segment is a set of pixels that has two
 * ending points.  They can be endpoints or intersections.
 */

class Segment {
    public boolean startFlg, endFlg;// Flags for processing
    public Pixel start, end;        // The coordiante of the end and start
    public byte startType, endType;    // The type of the end and start
    public byte startD, endD;       // The start and end travel direction
    public int pixelCount;          // Length of the segment
    public int dupSeg;              // Duplicate index in the segment table

    public Segment() {
      dupSeg = -1;
      startFlg = false;
      endFlg = true;    //After image processing, it will be the same as startFlg

      start = new Pixel();
      end = new Pixel();

      startType = endType = 0;
      startD = endD = -1;

      pixelCount = 0;
    }

    public Segment(Segment org) {
      dupSeg = -1;
      startFlg = false;
      endFlg = true;    //After image processing, it will be the same as startFlg

      startD = org.startD;
      startType = org.startType;
      start = new Pixel(org.start);

      endD = org.endD;
      endType = org.endType;
      end = new Pixel(org.end);

      pixelCount = org.pixelCount;
    }

    // This method swaps the information of the
    // start and end of the semgent.
    public void swapEnds() {
      byte tmpByte;
      Pixel tmpPixel = new Pixel();

      tmpByte = startType;
      startType = endType;
      endType = tmpByte;

      tmpByte = startD;
      startD = endD;
      endD = tmpByte;

      tmpPixel.move(start);
      start.move(end);
      end.move(tmpPixel);
```

```
    }

    public String toString() {
      return (start + "[" + startD + "]-->" + end + "[" + endD + "] " + startType + " " +
              endType + " " + pixelCount + " " + startFlg + " " + endFlg + " " + dupSeg);
    }

    public void toggleStartFlag() {
      startFlg = !startFlg;
    }

    public boolean isEndType(int type) {
      return (endType==type);
    }

    public boolean isStartType(int type) {
      return (startType==type);
    }

    public Pixel getStart() {
      return start;
    }

    public Pixel getEnd() {
      return end;
    }
}
```

B.16 SegTable.java

```
/*
 * Copyright 1997-1998 by Farzad Valad,
 * POB 8239 Longmont CO 80501
 * All rights reserved.
 */

import java.util.*;

/**
 * This class is a table of segments.  It contains all the
 * individual segments that make up a single region in the image.
 */

class SegTable {
    public Segment bestSeg;   // Stores the best starting point for the region

    private Vector segT;   // The list of all the segments
    private Segment piece;    // A reference to segment fragment
    private Vector ilist;     // The list of all intersections
    private Vector elist;     // The list of all endpoints
    private int currCount = 0;// The number of different segments
    private ImageProcessor imgPr;
    private int intCount=0, endCount=0;

    public SegTable(ImageProcessor imgPr) {
      this.piece = null;
      this.imgPr = imgPr;
      this.bestSeg = null;
      this.segT = new Vector(50,50);
      this.ilist = new Vector(5,5);
      this.elist = new Vector(5,5);
    }

    public String toString() {
      for(int i=0; i<segT.size(); i++)
         System.out.println(i + " " + (Segment)segT.elementAt(i));

      return ("Best Start: " + bestSeg.start + "[" + bestSeg.startD + "] Length: " +
bestSeg.pixelCount);
    }

    // Return a list of all the start and stop points for padding
    public int[] getPadEnds(int width) {
      Pixel tmpPnt = new Pixel();
      Segment tmpSeg = null;
      int count = segT.size();
      int[] ends = new int[2*count];

      for(int i=0,j=0; i<count; i++) {
         j = 2*i;
         tmpSeg = (Segment)segT.elementAt(i);

         tmpPnt.move(tmpSeg.getStart());
         // if the point is an intersection
         // move one pixel back
         if(tmpSeg.isStartType(Const.INTFLAG)) {
            // with the passing parameter .startD
            // this method will return on point
            // prior to this point.
            tmpPnt.translate(tmpSeg.startD);
         }
         ends[j] = tmpPnt.getIndex(width);

         tmpPnt.move(tmpSeg.getEnd());
         // if the point is an intersection
         // move one pixel back
         if(tmpSeg.isEndType(Const.INTFLAG)) {
```

```
         // with the passing parameter .endD
         // this method will return on point
         // prior to this point.
         tmpPnt.translate(tmpSeg.endD);
      }
      ends[j+1] = tmpPnt.getIndex(width);
   }
   return ends;
}

// This method returns the list of all the intersections.
public int[] getIntList(int width) {
   int i=0;
   int size = ilist.size();

   if(size==0)
      return null;
   else {
      int[] list = new int[size];
      for(Enumeration e = ilist.elements(); e.hasMoreElements();i++) {
         list[i] = ((Pixel)e.nextElement()).getIndex(width);
      }
      return list;
   }
}

public Pixel bestStart() {
   return bestSeg.start;
}


public boolean isMapped(Pixel pt, byte d) {
   Segment tmpSeg;
   int index = findSeg(pt,d);

   if(index==-1)
      return true;

   tmpSeg = ((Segment)segT.elementAt(index));


   if(!tmpSeg.startFlg) {
      tmpSeg.startFlg = true;
      if(tmpSeg.dupSeg!=-1) {
         segT.removeElementAt(tmpSeg.dupSeg);
         updateDupIndexOnRemove(tmpSeg.dupSeg);
         tmpSeg.dupSeg=-1;
      }
      return false;
   }
   else
      return true;
}

public void resetCount() {
   currCount = 0;
}

public void resetFlags() {
   Segment tmpSeg;
   for(int i=0; i<segT.size(); i++) {
      tmpSeg = ((Segment)segT.elementAt(i));
      tmpSeg.startFlg = tmpSeg.endFlg = true;
   }
}

public int score() {
   return currCount;
}

private int findSeg(Pixel pt, byte d) {
```

```
   Segment tmpSeg;
   for(int i=0; i<segT.size(); i++) {
      tmpSeg = ((Segment)segT.elementAt(i));
      if(tmpSeg.start.equals(pt) && tmpSeg.startD==d)
         return i;
   }
   return -1;
}

public boolean markOff(int x, int y) {
   Segment tmpSeg;
   Pixel pt = new Pixel(x,y);

   for(int i=0; i<segT.size(); i++){
      tmpSeg = ((Segment)segT.elementAt(i));

      if(tmpSeg.start.equals(pt)&& tmpSeg.startFlg) {
         tmpSeg.startFlg = false;
         currCount++;
      }

      if(tmpSeg.end.equals(pt) && tmpSeg.endFlg) {
         tmpSeg.endFlg = false;
         currCount++;
      }
   }

   if(currCount==2*segT.size())
      return false;
   else
      return true;
}

public synchronized boolean loadSeg(Segment newSeg) {
   if(newSeg.startType==-1)
      return true;

   else if(newSeg.startType==-2 || newSeg.endType==-2)
      return mergePiece(newSeg);

   else {
      insert(newSeg);
      return true;
   }
}

private synchronized void insert(Segment newSeg) {
   int tmp;

   if(newSeg.endType==-4 && newSeg.startType==-4 && !newSeg.start.equals(newSeg.end)) {
      Segment dup = new Segment(newSeg);
      dup.swapEnds();

      if(newSeg.start.isGreater(newSeg.end)) {
         newSeg.dupSeg = insertInOrder(0, dup);
         updateDupIndexOnInsert(newSeg.dupSeg);

         tmp = insertInOrder(newSeg.dupSeg+1,newSeg);
         updateDupIndexOnInsert(tmp);
         dup.dupSeg = tmp;
      }
      else {
         dup.dupSeg = insertInOrder(0,newSeg);
         updateDupIndexOnInsert(dup.dupSeg);

         tmp = insertInOrder(dup.dupSeg+1, dup);
         updateDupIndexOnInsert(tmp);
         newSeg.dupSeg = tmp;
      }
   }
   else {
```

```
         //It will swap lines with two endpoints automatically.
         if(newSeg.startType!=-4)
            newSeg.swapEnds();

         updateDupIndexOnInsert(insertInOrder(0,newSeg));
      }

      updateStart(newSeg);
   }

   private int insertInOrder(int startI, Segment newSeg) {
      int i;

      for(i=startI; i<segT.size(); i++) {
         if(((Segment)segT.elementAt(i)).start.isGreater(newSeg.start)) {
            segT.insertElementAt((Object)newSeg,i);
            return i;
         }
      }

      if(i==segT.size())
         segT.addElement(newSeg);

      return i;
   }

   private void updateDupIndexOnInsert(int index) {
      for(int i=0; i<segT.size(); i++) {
         if((((Segment)segT.elementAt(i)).dupSeg)>=index)
            ((Segment)segT.elementAt(i)).dupSeg++;
      }
   }

   private void updateDupIndexOnRemove(int index) {
      for(int i=0; i<segT.size(); i++) {
         if((((Segment)segT.elementAt(i)).dupSeg)>=index)
            ((Segment)segT.elementAt(i)).dupSeg--;
      }
   }

   private void updateStart(Segment newSeg) {
      // No one segemnt has been decided as the best yet, or ...
      if( (bestSeg==null)
         // Segments with at least one endpoints superseed segments with intersections only, or
...
         // Since all segments in the table must have their intersection as the start point, then
         // all we need to check in the endType and it will tell us if it is a -4 -4, -4 -3, -3 -
3 seg.
         || (bestSeg.startType==-4 && (newSeg.startType==-3 || newSeg.endType==-3))
            // Let the largest segment win, that is a middle segment!
            ||     (bestSeg.pixelCount<newSeg.pixelCount     &&     !(bestSeg.startType==-3     &&
newSeg.startType==-4 && newSeg.endType==-4)) ) {
         if(newSeg.endType!=-4)
            newSeg.swapEnds();

         if(bestSeg!=null && bestSeg.startType!=-4)
            bestSeg.swapEnds();

         bestSeg = newSeg;
      }
   }

   private void removeDirection(Pixel pt, byte d) {
      int pixVal=imgPr.getPixel(pt), tmpVal=0, nextDig=1;

      while(pixVal!=0) {
         if(pixVal%10==d) {
            tmpVal += 8*nextDig;
            nextDig *= 10;
         }
         else {
```

```
            tmpVal += (pixVal%10)*nextDig;
            nextDig *= 10;
         }
         pixVal /= 10;
      }

      imgPr.setPixel(pt,tmpVal);
   }

   private boolean mergePiece(Segment newSeg) {
      if(piece==null) {
         piece = newSeg;
         return true;
      }
      else {
         if(newSeg.start.equals(piece.start) && newSeg.end.equals(piece.end)) {
            if(piece.endType==-2) {
               piece.end.move(newSeg.start);
               piece.endD = newSeg.startD;
               piece.endType = newSeg.startType;
            }
            else {
               piece.start.move(newSeg.end);
               piece.startD = newSeg.endD;
               piece.startType = newSeg.endType;
            }

            piece.pixelCount += newSeg.pixelCount - 1;

            removeDirection(piece.start,piece.endD);

            insert(piece);

            piece = null;
            return true;
         }
         else if(newSeg.start.equals(piece.end) && newSeg.end.equals(piece.start)) {
            if(piece.endType==-2) {
               piece.end.move(newSeg.end);
               piece.endD = newSeg.endD;
               piece.endType = newSeg.endType;
            }
            else {
               piece.start.move(newSeg.start);
               piece.startD = newSeg.startD;
               piece.startType = newSeg.startType;
            }

            piece.pixelCount += newSeg.pixelCount - 1;

            removeDirection(piece.start,piece.endD);

            insert(piece);

            piece = null;
            return true;
         }
         else if(newSeg.start.equals(piece.start)) {
            if(piece.endType!=-2 && newSeg.endType!=-2) {
               piece.start.move(newSeg.end);
               piece.startD = newSeg.endD;
               piece.startType = newSeg.endType;

               piece.pixelCount += newSeg.pixelCount + 1;
               insert(piece);

               piece = null;
               return true;
            }
            else  if(  (piece.endType!=-2  &&  newSeg.endType==-2)  ||  (piece.endType==-2  &&
newSeg.endType!=-2) ) {
```

```
                piece.start.move(newSeg.end);
                piece.startType = newSeg.endType;
                piece.startD=newSeg.endD;
                piece.pixelCount += newSeg.pixelCount + 1;
                return true;
             }
             else
                return false;
          }
          else if(newSeg.end.equals(piece.end)) {
             if(piece.startType!=-2 && newSeg.startType!=-2) {
                piece.end.move(newSeg.start);
                piece.endD = newSeg.startD;
                piece.endType = newSeg.startType;

                piece.pixelCount += newSeg.pixelCount - 1;
                insert(piece);

                piece = null;
                return true;
             }
             else  if(  (piece.startType!=-2  &&  newSeg.startType==-2)  ||  (piece.startType==-2  &&
newSeg.startType!=-2) ){
                piece.end.move(newSeg.start);
                piece.endType = newSeg.startType;
                piece.endD=newSeg.startD;
                piece.pixelCount += newSeg.pixelCount + 1;
                return true;
             }
             else
                return false;
          }
          else
             return false;
       }
    }

    public void incIntersect(Pixel pt) {
       intCount++;
       ilist.addElement(new Pixel(pt));
    }

    public void incEndPoints(Pixel pt) {
       endCount++;
       elist.addElement(new Pixel(pt));
    }

    public int getIntersects() {
       return intCount;
    }

    public int getEndPoints() {
       return endCount;
    }
}
```

BIBLIOGRAPHY

[1] J. M. Kennedy, “How the Blind Draw”, *Scientific American*, 76-81, January 1997.

[2] F. M. Valad and P. Vazquez, “TraceCad, Human-Computer Interactive Software”, Course XXXX, Project Report, Bradley Department of Electrical and Computer Engineering, 1997.

[3] F. M. Valad and P. Vazquez, “ImagePilot 1.0, Drawing Tool for the Visually Impaired”, Course XXXX, Project Report, Bradley Department of Electrical and Computer Engineering, 1998.

[4] G. C. Vanderheiden, Nonvisual Alternative Display Techniques for Output from Graphics-Based Computers, *J. Visual Impairment and Blindness*, 83(8): 383-390, 1989.

[5] L. Lam, S.W. Lee and C.Y.Suen, “Thinning methodologies - A comprehensive survey”, *IEEE Transactions on Pattern Analysis and Machine Intelligance*, 14(9): 869-885, 1992.

[6] E. R. Davies, and A. P. N. Plummer, “Thinning algorithms: A critique and a new methodology”, *Pattern Recognition,* 14(3): 53-63, 1981.

[7] T. Y. Zhang and C.Y. Suen, “A fast parallel algorithm for thinning digital patterns”. *Communications of the* ACM, 27(3): 236-239, 1984.

[8] J. R. Parker, *Practical Computer Vision Using C*, John Wiley & Sons Inc. 77-94, 1993.

[9] J. C. Russ, *The Image Processing Handbook (second edition),* New York: CRC Press, 1995.

[10] D. H. Ballard and C. M. Brown, *Computer Vision*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1982.

[11] J. D. Foley, A. Dam, S. K. Feiner, and J. F. Hughes, *Computer Graphics: Principles and Practice (second edition),* New York: Addison-Wesley, 1990.

[12] R. Stefanelli, and A. Rosenfeld, “Some parallel thinning algorithms for digital pictures”, *Journal of the Association for Computing Machinery* 18: 255-264, 1971.

[13] M. Nadler, personal communication, January 1999.

[14] M. Kurze, “TDraw: A Computer-Based Tactile Drawing Tool for Blind People”, http:// http://www.cs.rpi.edu/pub/assets96/papers/ascii/Kurze.txt, University of Berlin, Germany,

[15] P. B. L. Meijer, “An Experimental System for Auditory Image Representations,'' IEEE Transactions on Biomedical Engineering, 39(2), pp. 112-121, Feb. 1992.

[16] M. Kurze, “Giving Blind People Access to Graphics (Example: Business Graphics)”, http:// www.inf.fu-berlin.de/~kurze/publications/se_95/swerg95.htm, University of Berlin, Germany, 1995.

VITA

Farzad Valad was born in January 1971, and raised in Baltimore, Maryland. He moved to Iran in 1984 and graduated from Shafa High School in 1990. He then moved back to United States, and in 1997 graduated top of his class from VPI with a Bachelor of Science degree in Computer Engineering. During his undergraduate years, he was accepted into the 5 year Bachelors/Masters program at VPI.

He continued his higher education career at Virginia Tech as graduate student and completed all his class work by August 1998. He then accepted a job offer from IBM as a Software Engineer working on AS/400 systems and continued to work on his thesis remotely from Boulder, Colorado. Successfully defending his thesis in February 1999, he continues to work for IBM. He is very eager to continue his higher education career and work on his Ph.D. when the time comes.