Small Scale Voice Recognition

Bryan Armstrong

University Undergraduate Research Fellow, 1994-95

Texas A&M University

Department of Computer Science

APPROVED

Undergraduate Advisor Walter C. Daugherity

Exec. Dir., Honors Program [illegible]

ABSTRACT

Small Scale Voice Recognition

Bryan M. Armstrong

Undergraduate Fellows Program

Advisor: Dr. Walter Daugherity

This report describes a small-scale, isolated word voice recognition module. I explain my approach to the problem of voice recognition and provide performance figures and sample runs to give a picture of how this module operates. In general the module is successful in matching isolated words. However, my project goal of speaker independence was not met.

Voice recognition is not a new idea. Successful voice recognition was achieved in the 1950's using analog speech storage and comparison methods. However, most research focuses on recognition of large vocabularies and continuous speech. For my recognition module, I use a simple matching algorithm that matches waveform data processed by a Fast Fourier Transform to previously stored patterns to recognize human speech.

TABLE OF CONTENTS

APPENDICES

LIST OF ILLUSTRATIONS

WHY VOICE RECOGNITION?

I chose to research voice recognition because of the wide variety of topics yet to be covered, and my personal interest in the subject. Also, there is no small-scale voice recognition package currently available. I wanted to build a small voice recognition module that could be added to any existing computer application to add simple voice control.

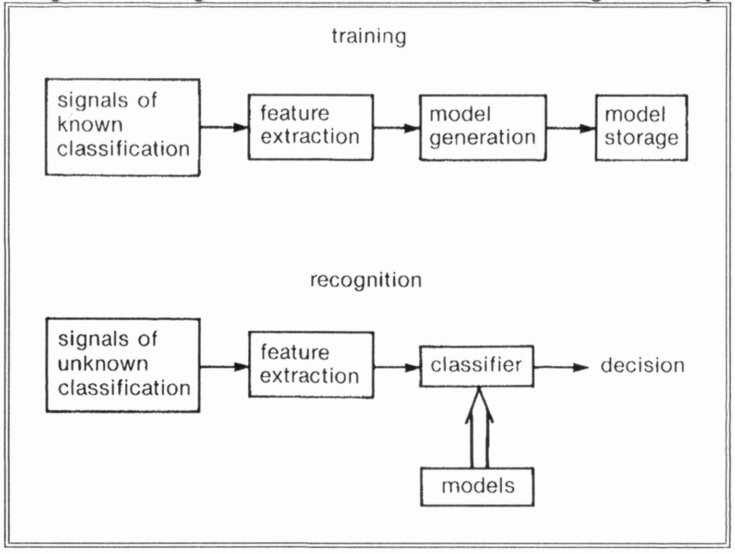
Background

Voice recognition is an idea which has been around since the 1950's when only analog equipment was available. As voice recognition advanced, digital processing of speech became more convenient and efficient to use than analog. Digital Signal Processors made the digital switch over easy by increasing the speed of feature extraction, usually the most time consuming pieces of the voice recognition process.

The Voice Recognition Process

The voice recognition process is broken down into two major steps: feature extraction and model pattern matching (Cox 211). (*See Figure 1.*)

Figure 1:
Training and Recognition Phases of a Voice Recognition System



Feature extraction is the process of formulating unique features from an input speech wave. The most commonly used feature is the frequency domain plot of a wave. Figure 2 shows the phrase "Noon is the sleepy time of day" plotted in the frequency domain (Yannakoudakis 70). Notice the two frequency bands between 1000 and 2500Hz. These bands carry the majority of the unique features of a word. Figure 3 is an example of a speech spectrogram generated from data produced by the small scale recognizer. The lighter areas denote regions of higher voice energy.

Figure 2:
Speech Spectrum of a Sample Phrase

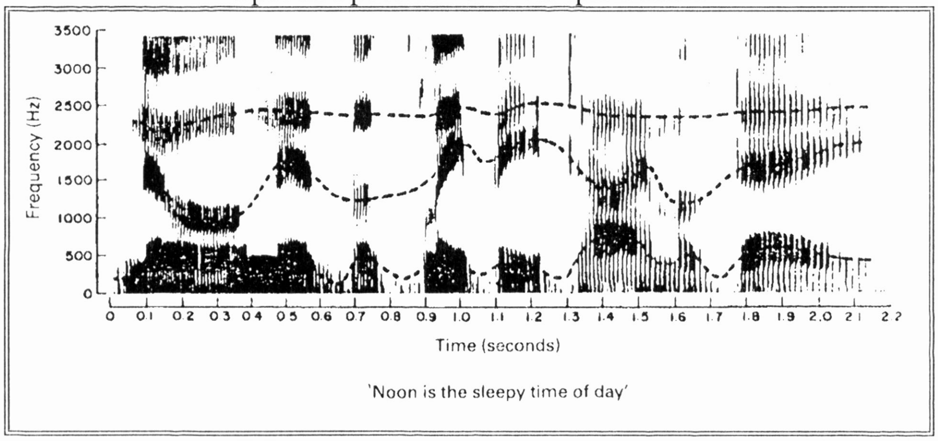
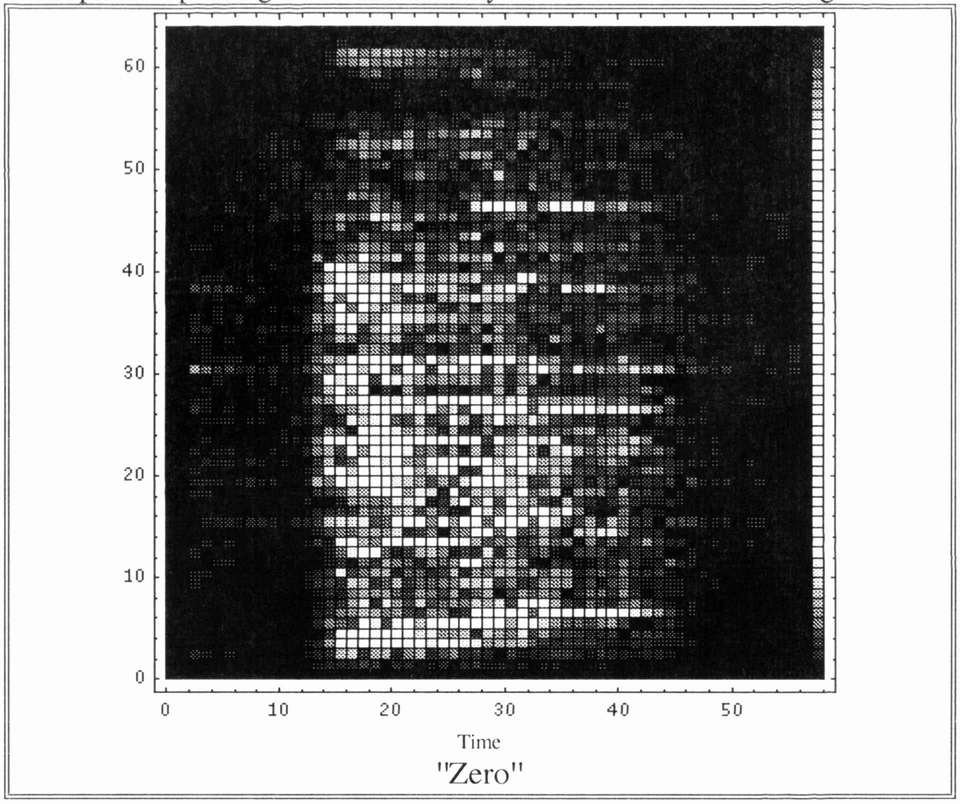


Figure 3:
Speech Spectrogram Generated by Small Scale Voice Recognizer



While pattern matching may sound simple, most of the research in voice recognition focuses on the pattern matching process. A voice recognition system uses pattern matching to compare an input signal with stored speech templates to decide on an appropriate output.

Voice recognition does not work without help from the user. One must train the system to recognize his or her speech. This training session may take a few minutes, or a few hours depending on the pattern matching method and number of words the system needs to recognize. The training process follows the recognition process through feature extraction and then it stores the results for later access. (*See Figure 1.*)

APPROACH TO THE PROBLEM

Simplicity

I approached the problem with simplicity in mind. There are many ways of matching voice patterns to stored waveforms: Dynamic Time Warping, Hidden Markov Models, and Chain Code Matching are a few. (Cater 122-138) I chose a template searching algorithm because my database is small, and the algorithm is simple. Each stored pattern is in an array type structure which is searched to find the closest match to the spoken pattern. For a small database an O(n) search is not unreasonable. Most large vocabulary recognizers use some kind of index, cross reference, or tree search to minimize search time.

Digital Sampling

I chose to use a sampling rate of 11111Hz to record the voice samples used in the training and recognition processes. The Nyquist theorem states that an analog signal must be sampled at a rate twice the frequency of the signal to successfully reproduce the signal digitally. Since human speech is located under the 4000Hz frequency I chose to use the 11111Hz sampling rate because it would correctly represent frequencies to 5555Hz.

I also chose to use an 8-bit linear format for the sample. This allows 256 possible values which has a dynamic range of about 48dB. This is sufficient for voice recognition.

Speed

The recognizer should be able to complete at least one word recognition per second. This is only possible on a machine with a floating point unit or digital signal processor. Otherwise, the Fast Fourier Transform will slow down the process completely. Originally I was to use a NeXT machine with a Motorola 56000 DSP to complete my project, but I chose not to use a DSP because of the time involved in learning the instruction set and the programming involved to take input from the CPU, run a FFT on the data, and return the results to the calling program.

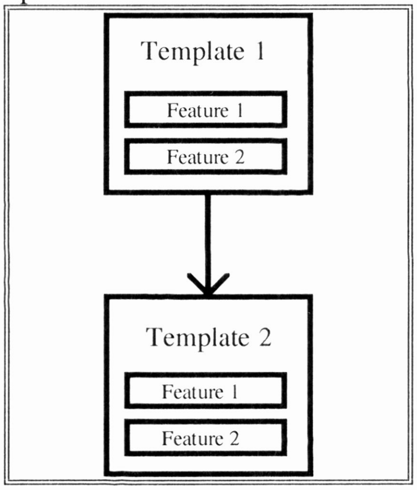
Short Research Time

Since I had such a short time period to complete the research and coding, I chose to use a template searching algorithm. The other methods would have taken far more time to research and the final coding would have been more complicated and harder to debug.

Feature Sets

Each word pattern (template) contains feature sets which represent the word. (*See Figure 4.*) In this case only one feature set was used.

Figure 4:
Template and Feature Set Relationship



The feature set I use is Percentage of Total Energy. The speech sample is broken down into 16 (or any power of two) ranges of frequencies, and a percentage of the total energy is found for each range. The feature set is then built from the minimum and maximum percentages in each frequency range. To get these minimum and maximum values the user must provide at least two, and preferably three samples for each word in the database.

Matching Algorithm

After building a template for each word needing to be recognized, a template matching algorithm is used. Each feature set is compared to a stored feature set giving a number representing how strongly they match. The template with the strongest match will be chosen as the correct word.

To generate each template, a spectrogram is first created. (*See Figure 3.*) To create this spectrogram the speech data is first read in 128 byte segments and fed into a FFT. The lower 64 points of the transform are then fed into a spectrum shaping function that de-emphasizes the extreme high and low portions of the spectrum and linearly shapes it to the center frequency. The shaped spectrum is then written to a file. This process is continued to the end of the sample, thus completing the generation of the spectrogram. The spectrogram is then used to calculate the feature set which is stored in a template.

RESTRICTIONS

No DSP Use

A DSP would greatly increase the speed of the FFT. While researching this project, I came across FFT code for the Motorola 56000 DSP. I procured a DSP56000 manual set and attempted to learn the instruction set and how to program the DSP. I abandoned the DSP avenue when I realized I did not have time to delve deeply into the subject. I found that using a machine with a floating point unit produced acceptable speed.

No Real Time Input

Real time input should not be difficult to add to the recognition module. Assuming a Sun Microsystems machine and SunOS operating system, the additional code would necessitate the following procedure:

1. Use ioctl() call to set up sampling rate, encoding type, number of channels and precision.
2. Open /dev/audio as a stream.
3. Read from the data stream using read() until a speech sample is captured.
4. Close /dev/audio.

The audio man page offers more information on the audio interface.

One Feature Set Used

Unfortunately I had time to implement only one feature set: Percentage of Total Energy. Other possible feature sets include speech length, phonemes and number of syllables. These could be easily incorporated into the current template structure, and the matching algorithm would require minimal change.

SAMPLE RUN

What follows is a sample run of the recognizer on a single sample stored in the file *0bryan.voc*.

```
sparc70~: start

Program start...
Checking start.ini
Trying to open start.ini
Loading patterns:
Opening file: 0.ftr.  Pattern added: 0.ftr
Opening file: 1.ftr.  Pattern added: 1.ftr
Opening file: 2.ftr.  Pattern added: 2.ftr
Opening file: 3.ftr.  Pattern added: 3.ftr
Opening file: 4.ftr.  Pattern added: 4.ftr
Opening file: 5.ftr.  Pattern added: 5.ftr
Opening file: 6.ftr.  Pattern added: 6.ftr
Opening file: 7.ftr.  Pattern added: 7.ftr
Opening file: 8.ftr.  Pattern added: 8.ftr
Opening file: 9.ftr.  Pattern added: 9.ftr


Voice Recognition Menu
1. Use Recognizer
2. Train Recognizer
3. Load File to Recognize
4. Calibrate Recognizer
5.
Q. Quit
3

Recognize from a file

File to load: 0bryan.voc

Bytes done: 256 512 768 1024 1280 1536 1792 2048 2304 2560 2816 3072 3328 3584
3840 4096 4352 4608 4864 5120 5376 5632 5888 6144 6400 6656 6912 7168 7424
7680 7936 8192 8448 8704 8960 9216 9472 9728 9984 10240 10496 10752 11008
11264 11520 11776 12032 12288 12544 12800 13056 13312 13568 13824 14080
14336 14592 14848 15104 15360 15616 15872 16128 16384 16640 16896 17152
The probability this matches 0 is: 0.7781
The probability this matches 1 is: 0.5555
The probability this matches 2 is: 0.7030
The probability this matches 3 is: 0.4510
The probability this matches 4 is: 0.4434
The probability this matches 5 is: 0.4359
The probability this matches 6 is: 0.2007
The probability this matches 7 is: 0.5887
The probability this matches 8 is: 0.4108
The probability this matches 9 is: 0.5275
The number chosen was: 0.7781
The best choice was: 0

Voice Recognition Menu
1. Use Recognizer
2. Train Recognizer
3. Load File to Recognize
4. Calibrate Recognizer
5.
Q. Quit
```

PERFORMANCE

A Word About Floating Point Units

During most of the development of the recognition software I used an Intel 386DX 25Mhz machine without a floating point unit. The absence of a floating point unit was more than a little noticeable. The spectrum building took almost 45 times longer than running on an Intel 486DX2/50Mhz machine with a floating point unit. True, the Intel 486 I was using has twice the clock rate and an internal data and instruction cache, but without a FPU the performance would have been as dismal as the Intel 386.

Performance Results for the Digits

All performance results are from a Sun 4/50 running SunOS 4.1.3. Each word recognition was completed in under one second.

First Attempt

For my first testing I used a set of training samples created from my own voice. The digits zero through nine were created with three samples of each digit. The first test was not reassuring. The three recognition tests on my voice did not produce accurate results. The test results follow.

Male Speaker 1 (Speaker that trained the recognizer)

Set 1: 60%	0,1,2,3,4,9 recognized correctly
Set 2: 40%	1,2,4,9 recognized correctly
Set 3: 60%	0,1,2,4,7,9 recognized correctly

Male Speaker 2

Set 1: 40%	1,2,4,7 recognized correctly

Male Speaker 3

Set 1: 50%	0,1,2,4,9 recognized correctly

Female Speaker 1
Set 1: 40%	4,5,8,9 recognized correctly
Set 2: 30%	5,7,9 recognized correctly

Second Attempt

After the disappointing results from the first test, I looked at the training samples and compared them to the test samples. The training samples were of a much lower amplitude. I then trained the recognizer with a new set of digits spoken more loudly, and the results were much improved.

In both the previous and current test, the number four was consistently recognized. However, the number nine which was frequently recognized in the first test, was only recognized by one speaker other than the training speaker in the second test.

Male Speaker 1 (Speaker that trained the recognizer)
Set 1: 90%	0,1,2,3,4,5,6,8,9 recognized correctly
Set 2: 80%	1,2,4,5,6,7,8,9 recognized correctly

Male Speaker 2
Set 1: 40%	1,2,4,5 recognized correctly
Set 2: 30%	2,4,7 recognized correctly

Male Speaker 3
Set 1: 60%	1,2,4,5,6,7 recognized correctly

Male Speaker 4 (slight Oriental accent)
Set 1: 50%	2,4,5,7,8 recognized correctly
Set 2: 50%	2,4,5,7,8 recognized correctly

Female Speaker 1
Set 1: 50%	2,4,5,8,9 recognized correctly
Set 2: 30%	4,5,9 recognized correctly

Female Speaker 2 (slight Russian accent)
Set 1: 40%	2,4,5,8 recognized correctly
Set 2: 20%	4,5 recognized correctly

CONCLUSION

From the results presented in the previous section two conclusions are made.

Firstly, using percentage of total energy as a sole feature for voice recognition is very limited. This feature can be used to accurately recognize the trainer's voice, but speaker independent recognition is not possible. More feature sets should be used to allow more accurate speaker dependent and independent recognition.

Secondly, and most importantly, small scale voice recognition is definitely possible. The simple template matching algorithm has little overhead with a small word database. With a word database of ten, this module can complete one recognition in under one second on a machine with a floating point unit. The executable occupies less than 130000 bytes of storage space, and the data files less than 3000 bytes.

FUTURE RESEARCH

Adding More Feature Sets

Since the recognizer only matches voice energy percentages, it is very limited for speaker independent applications. A fuller set of patterns to match could include timing, phonemes, and envelope contours. (Cater 122-38) One could possibly break down each word into phonemes and keep a phoneme database to search. This would reduce the disk storage requirements, since each word would have a specific phoneme construction which takes less space to store than a complete spectral analysis. However, breaking down a speech sample into phonemes is a difficult and complex process which would greatly increase the load on the CPU, thereby slowing down the recognition process.

Using a Digital Signal Processor

The use of a DSP would probably increase the speed at least ten fold over a machine with only a floating point unit. The Motorola DSP56000 could easily complete a 128 point FFT in real time on 8000Hz sampled digital data. This would allow almost instantaneous recognition of isolated words.

HOW TO RUN THE RECOGNIZER

If No Executable Exists

If the executable does not exist, the user will have to compile it before attempting to use the recognizer.

The program set consists of a *Makefile* and all the *.c* and *.h* files necessary for the program to function. On a Unix machine simply type *make* and the program should build correctly. The executable is named *start*.

Running

First, the user should record three or more samples of each word for the recognizer to learn. These word samples should be in a linear 8-bit PCM format. Sound tools are standard with most workstations to allow recording and playback of sound.

Next, run the *start* executable.

Then, select menu option "2. Train Recognizer" from the main menu. A training menu will then appear. Select "1. Train from file" and enter the filename of the first word to train. The program will then prompt for a word name with which to create the template. Continue the training from file process until all words are trained.

Finally, the recognition can be run. Select "3. Recognize from file" on the main menu. Type in the file name at the prompt. You will see the byte counter at the bottom of the screen count through the voice sample. After completion, the recognizer will display a list of valid words and the word in the database with the highest probability of matching the file.

REFERENCES

Cater, John P. *Electronically Hearing: Computer Speech Recognition*. Indianapolis: Howard W. Sams & Co., 1984.

Cox, S.J. "Hidden Markov Models for Automatic Speech Recognition: Theory and Application." *Speech and Language Processing*. Eds. C. Whedden and R. Linggard. London: Chapman and Hall, 1990. 209-30.

Yannakoudakis, E. J. and P. J. Hutton. *Speech Synthesis and Recognition Systems*. New York: Ellis Horwood, 1987.

APPENDIX A

Source Code

start.c

```
/*
start.c

Mainline of voice recognition.
*/

#include "start.h"

struct pattern_list pattlist;

/*
These are default values used by the get_input() function
and others to detect voice input.  These values will be replaced
by those in the start.ini file if it exists.
*/
int MaxCount = 16;
int MedianOfSampRange = 127;
int NoiseLevel = 2;
int SizeOfSample = 1; /* in bytes */

/*
Offset value used in seeking past any headers contained in voice
files.
*/
int Offset = 33;


#ifdef _PC_
#include "graph.h"
#endif

#include "filerec.h"
#include "train.h"
#include "use.h"
#include "calibrat.h"
#include "init.h"
#include "extract.h"
#include "recogniz.h"

void clear_screen();
void print_menu();

void clear_screen()
{
  int i;
  for(i=0;i<25;i++) printf("\n");
}
```

```
void print_menu()
{
  #ifdef _PC_
  #ifdef DEBUG
  printf("\nCoreleft: %u\n",coreleft());
  #endif
  #endif
  printf("\nVoice Recognition Menu\n");
  printf("1. Use Recognizer\n");
  printf("2. Train Recognizer\n");
  printf("3. Load File to Recognize\n");
  printf("4. Calibrate Recognizer\n");
  printf("5. \n");
  printf("Q. Quit\n");
}

main(argc)
int argc;
{
  int done = 0;
  char ch;

  printf("\nProgram start...\n");

  if (argc != 2)
    init_stuff();
  else
  {
    /* Build database from existing files 0a..0c-9a..9c */
    build_stuff();
    exit(0);
  }

  /* clear_screen(); */
  while(!done)
  {
    print_menu();
    ch = getchar();
    if (islower(ch))
    ch = toupper(ch);
    switch(ch)
    {
      case '1': use_recognizer();
                break;
      case '2': train_recognizer();
                destroy_pattlist();
                init_stuff();
                break;
      case '3': printf("\nRecognize from a file\n");
                file_recognize();
                break;
      case '4': calibrate_recognizer();
                break;
      case 'Q': done = 1;
```

```
                break;
        }
    }

    destroy_pattlist();
    return(1);
}
```

init.c

```
/*
init.c

This will check for a config file and load it if
it exists.  It will also create the pattern linked list
necessary for the rest of the program.
*/

#include "start.h"
#include "init.h"

/*
Destroy the pattern link list.
*/
int destroy_pattlist()
{
  struct pattern *temp;

  extern struct pattern_list pattlist;

  while (pattlist.head != NULL)
  {
    temp = pattlist.head;
    pattlist.head = pattlist.head -> next;
    pattlist.number--;
    free(temp);
  }

  return(1);
}

/*
Add a pattern to the pattern link list.
*/
int add_pattern(newpat)
struct pattern *newpat;
{
  extern struct pattern_list pattlist;

  /* Insert new pattern at end of list */
  if (pattlist.head == NULL)
  {
    pattlist.head = newpat;
    pattlist.tail = newpat;
    pattlist.number++;
  }
  else
  {
    pattlist.tail -> next = newpat;
    pattlist.tail = newpat;
    pattlist.number++;
  }

  return(1);
}

/* Build database from existing files 0a-9c */
int build_stuff()
{
  int i,j;
  char str[80];
  struct pattern *patt;
  struct spectrogram *spec;
  char names[][2] = {"0","1","2","3","4","5","6","7","8","9"};
  char ext[][2] = {"a","b","c"};

  for (i=0;i<10;i++)
  {
    for (j=0;j<3;j++)
    {
      strcpy(str,names[i]);
      strcat(str,ext[j]);
      strcat(str,".voc");
      spec = build_spectrogram(str);
      if (spec != NULL)
      {
        patt = extract(spec);
        store_pattern(patt,names[i]);

        free(patt);
        free(spec->array);
        free(spec);
      }
      else
      {
        printf("Error handling %s.\n",names[i]);
      }
    }
  }
  return(1);
}


int init_stuff()
{
  FILE *infile;
  struct pattern *newpat;
  char str[80];

  extern struct pattern_list pattlist;
  extern int MaxCount;
  extern int MedianOfSampRange;
  extern int NoiseLevel;

  pattlist.head = NULL;
  pattlist.tail = NULL;
  pattlist.number = 0;

  printf("Checking start.ini\n");
  infile = fopen("start.ini","r");
  printf("Trying to open start.ini\n");
  while ((infile != NULL) && !feof(infile))
  {
    printf("start.ini found!\n");
    fscanf(infile,"%d",MaxCount);
    fscanf(infile,"%d",NoiseLevel);
    fscanf(infile,"%d",MedianOfSampRange);
  }
  fclose(infile);

  printf("Loading patterns:\n");
  infile = fopen("patterns","r");
  while ((infile != NULL) && !feof(infile))
  {
    fscanf(infile,"%s",str);
    newpat = get_pattern(str);
    add_pattern(newpat);
    printf("Pattern added: %s\n",str);
  }
  fclose(infile);
  printf("\n");

  return(1);
}
```

filerec.c

```
/*

filerec.c

Recognize saved file.
*/

#include "start.h"
#include "filerec.h"

int file_recognize()
{
  struct spectrogram *spec = NULL;
  struct pattern *patt = NULL;
  struct pattern *bestchoice = NULL;
  char filename[40];
#ifdef _PC_
  char ch;
  int done = 0;
  int first = 0;
  int second = 0;
#endif

  printf("\nFile to load: ");
  scanf("%s",filename);

  spec = (struct spectrogram *)build_spectrogram(filename);

#ifdef _PC_
  if (spec != NULL)
  {
    if (intograph())
    {
      while (!done)
      {
        graph(spec,spec -> size,first,second);

        kbhit();
        if ('Q' == toupper(ch)) done = 1;
        if ('C' == toupper(ch))
        {
          gotoxy (40,12);
          printf("Enter new values: ");
          scanf("%d %d",&first,&second);
        }
      }

      outofgraph();
    }
  }
  else
  {
    printf("\nSpectrogram Build Failed\n");
    return(0);
  }
#endif

  if (spec != NULL)
  {
    patt = extract(spec);
    bestchoice = recognize(patt);

    printf("The best choice was: %s\n",bestchoice->name);

    free(patt);
    free(spec -> array);
    free(spec);
    return(1);
  }
  else
  {
    printf("\nFailed");
    return(0);
  }
}
```

extract.c

```
/*
extract.c

Extract all of the features from the spectrogram.
*/

#include "start.h"
#include "extract.h"

struct pattern *percents(spec,patt)
struct spectrogram *spec;
struct pattern *patt;
{
  unsigned int count;
  unsigned long int sum;
  unsigned int pos;
  int halfnp = NUMPOINTS/2;

  /* Initialize the percent freqs */
  for (count = 0; count < NUMBANDS*2; count++)
  {
    patt -> percent_freqs[count] = 0;
  }

  /* Calculate sums for all freqs and total energy. */
  sum = 0;
  for (count = 0; count < spec -> size; count++)
  {
    pos = 2*((count % halfnp)/(halfnp/NUMBANDS));
    sum = sum + spec -> array[count];
    patt -> percent_freqs[pos+1] += spec -> array[count];
  }

  /* Put average energy values in. */
  for (count = 0;count < NUMBANDS*2; count ++)
  {
    patt -> percent_freqs[count] /= sum/100;
  }

  return(patt);
}


int printstuff(patt)
struct pattern *patt;
{
  int count;
  for (count=0;count<NUMBANDS;count+=2)
  {
    printf("%d: %8.3f, ",count/2,patt->percent_freqs[count]);
    printf("%8.3f     ",patt->percent_freqs[count+1]);
```

```
      printf("%d: ",(count+NUMBANDS)/2);
      printf("%8.3f,",patt->percent_freqs[count+NUMBANDS]);
      printf(" %8.3f\n",patt->percent_freqs[(count+NUMBANDS)+1]);
    }
    return(1);
  }


struct pattern *extract(spec)
struct spectrogram *spec;
{
  struct pattern *patt;

  patt = (struct pattern *)malloc(sizeof(struct pattern));
  if (patt != NULL)
  {
    patt = percents(spec,patt);
    /*
    Extract more stuff here.
    */
    #ifdef DEBUG
    printstuff(patt);
    #endif

    return(patt);
  }
  else
  {
    return(NULL);
  }
}
```

buildspc.c

```
/*
buildspc.c

Build the spectrogram.
*/

#include "start.h"
#include "buildspc.h"

struct spectrogram *build_spectrogram(filename)
char *filename;
{
  FILE *infile;
  FILE *outfile;
  #ifdef DUMPSPECTROGRAM
  FILE *dumpfile; /* For dumping the spectrogram */
  #endif
  double *x;  /* Stores the real part of the fft */
  double *y;  /* Stores the imaginary part of the fft */
  int    *store; /* Stores the integer values of the fft */
  struct spectrogram *spec;
  extern int MedianOfSampRange;
  extern int SizeOfSample;
  extern int Offset;

  int count;
  int done;
  long length;
  float tempnum;
  unsigned int tempint1;
  unsigned int tempint2;
  int halfnp = NUMPOINTS/2;

  /* Malloc my working space */
  x = (double *)malloc(sizeof(double)*NUMPOINTS);
  y = (double *)malloc(sizeof(double)*NUMPOINTS);
  store = (int *)malloc(sizeof(int)*NUMPOINTS);

  if ( (x == NULL) || (y == NULL) || (store == NULL) )
  {
    printf("\nCould not malloc x,y, or store in buildspec.c\n");
    exit(1);
  }

  infile = fopen(filename,"rb");
  outfile = fopen("temp","wb");

  if (infile == NULL)
  {
    printf("\nFILENAME <%s> not found\n",filename);
    fclose(infile);
    fclose(outfile);
    free(x);
    free(y);
    free(store);
    return(NULL);
  }

  if (outfile == NULL)
  {
    printf("\nCould not open temp file\n");
    exit(1);
  }

  fseek(infile,Offset,0);

  /* Change file data into spectrogram. */
  printf("\nBytes done:");
  length = 0;
  done = 0;
  while(!done)
  {
    /* Read in NUMPOINTS worth of data */
    for (count = 0;count < NUMPOINTS;count++)
    {
      tempint1 = fgetc(infile);

      /* If the sample is other than 1 byte */
      if (SizeOfSample > 1)
      {
        tempint2 = fgetc(infile);
        /* Shift left 8 bits */
        tempint2 <<= 8;
        tempint1 = tempint1 | tempint2;
      }

      tempnum = (double)tempint1;
      if (!feof(infile))
      {
        x[count] = tempnum - MedianOfSampRange;
        y[count] = 0.0;
      }
      else
      {
        /* Fill in the value for no data for the slack space in file */
        done = 1;
        x[count] = 0.0;
        y[count] = 0.0;
      }
    }
    fft(x,y,NUMPOINTS);

    /* Make power spectrum and shape it */
    for (count = 0;count < halfnp;count++)
    {
```

```
    x[count] = (double)(pow(pow(x[count],2)+pow(y[count],2),0.5));

    if (count < (halfnp/2))
    {
      store[count] = (int)(x[count] * 1.0/(halfnp/2)*count);
    }
    else
    {
      store[count] = (int)(x[count] * 1.0/(halfnp/2)*(halfnp-count));
    }
  }

  if (!done)
  {
    /* Write the sucker to a file */
    /* ONLY HALF OF THE ARRAY IS WRITTEN */
    fwrite(store,sizeof(int)*halfnp,1,outfile);
    length += halfnp;
    printf(" %u",length*sizeof(int));
    fflush(stdout);
  }
}
printf("\n");

/* Clean up! */
free(x);
free(y);
free(store);

fclose(infile);
fflush(outfile);
fclose(outfile);

outfile = fopen("temp","rb");
rewind(outfile);

/* Get space for spectrum */
store = (int *)malloc(sizeof(int)*length);

if (store == NULL)
{
  printf("\nCould not malloc %s\n",sizeof(int)*length);
  exit(1);
}

fread(store,sizeof(int),length,outfile);

fclose(outfile);
unlink("temp");

#ifdef DUMPSPECTROGRAM
dumpfile = fopen("spectro.out","w");
if (dumpfile != NULL)
{
      for (count = 0;count < length;count++)
      {
        /*fprintf(dumpfile,"%d ",count % halfnp);*/
        /*fprintf(dumpfile,"%d ",count / halfnp);*/
        fprintf(dumpfile,"%u\n",store[count]);
      }
      fclose(dumpfile);
    }
    else
    {
      printf("Could not open spectro.out\n");
    }
    #endif

    spec = (struct spectrogram *)malloc(sizeof(struct spectrogram));
    if (spec != NULL)
    {
      spec -> size = length;
      spec -> points = NUMPOINTS;
      spec -> array = store;
      return(spec);
    }
    else
    {
      return(NULL);
    }
  }
```

pattfile.c

```
/*
pattfile.c

Handles all pattern file reading and writing.
*/

#include "start.h"
#include "pattfile.h"

int store_pattern(patt,samplename)
struct pattern *patt;
char *samplename;
{
  char samplefilename[40];
  struct pattern *pattfile;
  FILE *outfile;
  int count;

  strcpy(samplefilename,samplename);
  strcat(samplefilename,".ftr");
  if (!(outfile = fopen(samplefilename,"r")))
  {
    /* If this is a new pattern do this. */

    outfile = fopen(samplefilename,"w");
    fprintf(outfile,"%s\n",samplename);

    /* Store word pattern data in .ftr file */
    for (count = 0; count < NUMBANDS*2;count+=2)
    {
      fprintf(outfile,"%10.5f ",patt -> percent_freqs[count+1]);
      fprintf(outfile,"%10.5f\n",patt -> percent_freqs[count+1]);
    }
    fclose(outfile);

    /* Update the patterns file */
    outfile = fopen("patterns","a");
    fprintf(outfile,"%s\n",samplefilename);
    fclose(outfile);
  }
  else
  {
    /* If pattern already exists we need to update it here */

    /* Get the pattern already on disk */
    pattfile = get_pattern(samplefilename);
    if (pattfile == NULL) return(0);

    outfile = fopen(samplefilename,"w");
    fprintf(outfile,"%s\n",samplename);
    /* Using the pattern in memory and the one from disk
```

```
       write a new pattern file.
    */
    for (count = 0; count < NUMBANDS*2;count+=2)
    {
      if (patt -> percent_freqs[count+1] <
          pattfile -> percent_freqs[count])
      {
        pattfile->percent_freqs[count] = patt->percent_freqs[count+1];
      }
      else
      {
        if (patt -> percent_freqs[count+1] >
                 pattfile -> percent_freqs[count+1])
        {
          pattfile->percent_freqs[count+1] = patt->percent_freqs[count+1];
        }
      }
    }

    /* Store word pattern data in .ftr file */
    for (count = 0; count < NUMBANDS*2;count+=2)
    {
      fprintf(outfile,"%10.5f ",pattfile -> percent_freqs[count]);
      fprintf(outfile,"%10.5f\n",pattfile -> percent_freqs[count+1]);
    }
    free(pattfile);
    fflush(outfile);
    fclose(outfile);
  }
  return(1);
}

struct pattern *get_pattern(filename)
char *filename;
{
  FILE *patternfile;
  struct pattern *newpat;
  char tempstr[80];
  float tempnum1;
  float tempnum2;
  int count;

  printf("Opening file: %s.  ",filename);
  patternfile = fopen(filename,"r");
  if (patternfile != NULL)
  {
    newpat = (struct pattern *)malloc(sizeof(struct pattern));
    if (newpat == NULL)
    {
      printf("\nCould not allocate memory to add pattern.\n");
      exit(1);
    }

    fscanf(patternfile,"%s",tempstr);
```

```
      strcpy(newpat -> name,tempstr);

      for (count = 0; count < NUMBANDS*2;count+=2)
      {
        fscanf(patternfile,"%f %f",&tempnum1,&tempnum2);
        newpat -> percent_freqs[count] = tempnum1;
        newpat -> percent_freqs[count+1] = tempnum2;
      }

      newpat -> next = NULL;
      fclose(patternfile);
      return(newpat);
    }
    else
    {
      printf("\n%s pattern file not found!!",filename);
      fclose(patternfile);
      return(NULL);
    }

}
```

recogniz.c

```
/*
recogniz.c

THE RECOGNIZER!!!! Ta-da.
*/

#include "start.h"
#include "recogniz.h"


float compare(patt,temp)
struct pattern *patt;
struct pattern *temp;
{
  struct pattern storage;
  float average;
  float point1;
  float point2;
  float tempnum;
  int count;


  /* Check percent freqs. */
  for (count = 0; count < NUMBANDS*2;count +=2)
  {
    if ((patt -> percent_freqs[count+1] <=
         temp -> percent_freqs[count+1]) &&
        (patt -> percent_freqs[count+1] >=
         temp -> percent_freqs[count]))
    {
      storage.percent_freqs[count+1] = 1.0;
      point1 = patt->percent_freqs[count+1];
      #ifdef DEBUG
      printf("Middle: patt: %8.3f ",point1);
      printf("%4.3f\n",storage.percent_freqs[count+1]);
      #endif
    }
    else
    {
      if (patt -> percent_freqs[count+1] >
          temp -> percent_freqs[count+1])
      {
        point1 = patt -> percent_freqs[count+1];
        point2 = temp -> percent_freqs[count+1];
        tempnum = 1-((point1 - point2)*(FREQSLOPE));
        if (tempnum < 0.0) tempnum = 0.0;
        #ifdef DEBUG
        printf("Greater: patt: %8.3f temp: %8.3f ",point1,point2);
        printf("%4.3f\n",tempnum);
        #endif
      }
      else
      {
        point1 = patt -> percent_freqs[count+1];
        point2 = temp -> percent_freqs[count];
        tempnum = 1-((point2 - point1)*(FREQSLOPE));
        if (tempnum < 0.0) tempnum = 0.0;
        #ifdef DEBUG
        printf("Lesser: patt: %8.3f temp: %8.3f ",point1,point2);
        printf("%4.3f\n",tempnum);
        #endif
      }
      storage.percent_freqs[count+1] = tempnum;
    }
  }

  /* Weight each category. */
  average = 0.0;
  for (count = 0; count < NUMBANDS*2;count += 2)
  {
    average += storage.percent_freqs[count+1];
  }
  average /= NUMBANDS;
  printf("The probability this matches %s ",temp->name);
  printf("is: %5.4f\n",average);

  return(average);
}

struct pattern *recognize(patt)
struct pattern *patt;
{
  extern struct pattern_list pattlist;

  struct pattern *temp = pattlist.head;
  struct pattern *bestchoice = NULL;
  float previousnum = 0.0;
  float currentnum = 0.0;

  /* Compare patt to list of patterns and return a result. */
  while (temp != NULL)
  {
    /* Check using all features. */
    currentnum = compare(patt,temp);
    if (currentnum > previousnum)
    {
      previousnum = currentnum;
      bestchoice = temp;
    }
    temp = temp -> next;
  }

  printf("The number chosen was: %5.4f\n",previousnum);

  return(bestchoice);
```

```
}
```

train.c

```
/*
train.c

Train the recognizer.
*/

#include "start.h"
#include "train.h"

void train_menu()
{
  #ifdef _PC_
  #ifdef DEBUG
  printf("\nCorefree: %u",coreleft());
  #endif
  #endif
  printf("\nTraining Menu\n");
  printf("1. Train from stored files\n");
  printf("2. Train in real time\n");
  printf("Q. Quit to main\n");
}

void train_recognizer()
{
  int done = 0;
  int ch;

  /* Memory leak is here somewhere (it may be fixed now) */
  while (!done)
  {
    train_menu();
    ch = getchar();
    if (islower(ch))
    ch = toupper(ch);
    switch(ch)
    {
      case '1': printf("\nTrain from a file\n");
                train_files();
                break;
      case '2': train_realtime();
                break;
      case 'Q': done = 1;
                break;
    }
  }

}

/*
Train from a pre-recorded voice file.
*/
```

```
int train_files()
{
  struct spectrogram *spec = NULL;
  struct pattern *patt;
  char filename[40];
  char samplename[40];


  printf("\nFilename: ");
  scanf("%s",filename);
  printf("\nName of sample: ");
  scanf("%s",samplename);

  spec = build_spectrogram(filename);

  if (spec != NULL)
  {
    /* Extract the features from the spectrogram */
    patt = extract(spec);

    /* Store features in *.ftr */
    store_pattern(patt,samplename);

    /* Clean up */
    free(patt);
    free(spec -> array);
    free(spec);
  }
  else
  {
    printf("\nFailed\n");
    return(0);
  }

  return(1);
}

/*
Train from real time user voice input.  NOT IMPLEMENTED.
*/
int train_realtime()
{
  struct pattern *patt;
  struct spectrogram *spec;
  char samplename[40];

  printf("Waiting for Voice Input\n");
  printf("Hit a key to escape\n");
  get_input();

  printf("\nName of captured sample: ");
  scanf("%s",samplename);

  /*
  NOT IMPLEMENTED.

  Build feature list.
  spec = build_spectrogram("");

  Extract features.
  patt = extract(spec);

  Store features in *.ftr
  store_pattern(patt,samplename);

  free(spec -> array);
  free(spec);
  free(patt);
  */

  return(1);
}
```

graph.c

```
/*
graph.c

Graphing section.
*/

#include "start.h"
#include "graph.h"

#ifdef _PC_

int intograph()
{
  int gdriver = VGA;
  int gmode = VGAHI;
  int errorcode;

  initgraph(&gdriver,&gmode,"/tc/bgi");

  errorcode = graphresult();

  if (errorcode != grOk)
  {
    printf("Error: %s",grapherrormsg(errorcode));
    return(0);
  }

  return(1);
}

void outofgraph()
{
  closegraph();
}

int graph(spec,length,first,second)
struct spectrogram *spec;
unsigned int length;
int first;
int second;
{
  int count;
  int count2;
  int temp;
  int halfnp = NUMPOINTS/2;
  unsigned long int sum = 0;

  /* Find the average value */
  for (count = 0;count < length; count++)
  {
    sum = sum + spec -> array[count];
  }
  sum = (unsigned long int)sum/length;
  if (first == 0)
  {
    first = sum * 5;
    second = first * 1.2;
  }

  gotoxy(1,22);
  printf("%d %d",first,second);

  gotoxy(40,14);
  printf("Average: %d",sum);

  /* Using a halfnp by length/halfnp two dim array */
  for (count2 = 0;count2 < length/halfnp; count2++)
  {
    for (count = 0; count < halfnp;count++)
    {
      gotoxy(1,25);
      printf("%14.4f is the num",spec -> array[count+count2*halfnp]);
      temp = spec -> array[count+count2*halfnp];
      if (temp > first)
        putpixel((count2*2) % 640,halfnp*2-(count*2),WHITE);
      if (temp > second)
        putpixel((count2*2) % 640,halfnp*2-(count*2),LIGHTRED);
      gotoxy(1,24);
      printf("%u: on count = %u",count2*halfnp,count);
    }
    if (kbhit()) return(1);
  }
  return(1);
}

#endif
```

getinput.c

```
/*
getinput.c

Get the input from /dev/dsp and put it somewhere for the
rest of the program to find.

Remember to do something about the format of the sound sample!!
u-law needs to be converted to int if used on a Sun.
*/

#include "start.h"
#include "getinput.h"

/*
Grab a single sample.
*/
unsigned int get_single_sample(audioin)
FILE *audioin;
{
  unsigned int tempnum1;
  unsigned int tempnum2;
  extern int SizeOfSample;

  tempnum1 = fgetc(audioin);

  if (SizeOfSample > 1)
  {
    tempnum2 = fgetc(audioin);
    tempnum2 <<= 8;
    tempnum1 = tempnum1 | tempnum2;
  }

  return(tempnum1);
}

/*
Read a voice block in and write to a file of ints. NOT IMPLEMENTED.
*/
int read_block(audioin)
FILE *audioin;
{
  return(1);
}

/*
Chop the excess dead space off each end of sample. NOT IMPLEMENTED.
*/
int chop_block()
{
  return(1);
}
```

```
/*
Get input from /dev/dsp.  May not be available.  Might have to
use /dev/audio.  Have to filter .au header and return byte
values. ??????????????
*/
int get_input()
{
  int count = 0;
  unsigned int samp;
  extern int MaxCount;
  extern int MedianOfSampRange;
  extern int NoiseLevel;
  FILE *audioin;

  audioin = fopen("/dev/audio","r");
  if (audioin != NULL)
  {
    while((count < MaxCount) /*&& !kbhit()*/)
    {
      /*
      Open a stream from /dev/dsp and check until a certain
      level is reached MaxCount times.  Then capture the puppy.
      */
      samp = get_single_sample(audioin);
        if (abs(samp - MedianOfSampRange) > NoiseLevel) count++;

        if (count > MaxCount)
        {
          /* Read in sample for a while. */
         read_block(audioin);
         fclose(audioin);
         /* Find the end of the sample and cut off excess. */
         chop_block();
         return(1);
      }
      else
      {
        printf("\nCould not open /dev/audio.  Exiting.\n");
        exit(1);
      }
    }
    return(0);
  }
}
```

use.c

```
/*
use.c


Use the recognizer.
(as in real time)
*/

#include "start.h"
#include "use.h"

void use_recognizer()
{
  struct pattern *patt;
  struct pattern *bestchoice;
  struct spectrogram *spec;

  printf("Waiting for Voice Input\n");
  printf("Hit a key to escape\n");
  get_input();

  /*
  NOT IMPLEMENTED.

  Build the spectrogram.
  spec = build_spectrogram("");

  Build the feature list
  patt = extract(spec);

  Send feature list to recognizer
  bestchoice = recognize(patt);

  Output result

  free(spec -> array);
  free(spec);
  free(patt);
  */
}
```

calibrat.c

```
/*
calibrat.c

Calibrate the recognizer.  NOT IMPLEMENTED.
*/

#include "start.h"
#include "calibrat.h"

int calibrate_recognizer()
{
  printf("Calibrate Menu\n");
  printf("NOT IMPLEMENTED.\n");
  return(0);
}
```

APPENDIX B

Source code for FFT from Dave Edelblute

bryfft.c

```
/*
** by: Dave Edelblute, edelblut@cod.nosc.mil, 05 Jan 1993
** Modified: R. Mayer to work with my benchmark routines.
*/

#include "bryfft.h"

#include "tables.h"

#ifndef PI
#define PI  3.14159265358979323846
#endif

double l_cos(double x)
{
  int num;

  num = (int) x/PI*180;
  return(costable[num]);
}

double l_sin(double x)
{
  int num;

  num = (int) x/PI*180;
  return(sintable[num]);
}

/*
       A Duhamel-Hollman split-radix dif fft
       Ref: Electronics Letters, Jan. 5, 1984
       Complex input and output data in arrays x and y
       Length is n.
*/

int fft( x, y, np )
double x[2];
double y[2];
int np ;
{
       double *px,*py;
       int i,j,k,m,n,i0,i1,i2,i3,is,id,n1,n2,n4 ;
       double  a,e,a3,cc1,ss1,cc3,ss3,r1,r2,s1,s2,s3,xt ;
       px = x - 1;
       py = y - 1;
       i = 2;
       m = 1;
```

```
        while (i < np) {
                i = i+i;
                m = m+1;
        };
        n = i;
        if (n != np) {
                for (i = np+1; i <= n; i++)  {
                        *(px + i) = 0.0;
                        *(py + i) = 0.0;
                };
                printf("\nuse %d point fft",n);
        }

        n2 = n+n;
        for (k = 1;  k <= m-1; k++ ) {
                n2 = n2 / 2;
                n4 = n2 / 4;
                e = 2.0 * PI / n2;
                a = 0.0;
                for (j = 1; j<= n4 ; j++) {
                        a3 = 3.0*a;
       cc1 = l_cos(a);
       ss1 = l_sin(a);
       cc3 = l_cos(a3);
       ss3 = l_sin(a3);
                        a = j*e;
                        is = j;
                        id = 2*n2;
                        while ( is < n ) {
                                for (i0 = is; i0 <= n-1; i0 = i0 + id) {
                                        i1 = i0 + n4;
                                        i2 = i1 + n4;
                                        i3 = i2 + n4;
                                        r1 = *(px+i0) - *(px+i2);
                                        *(px+i0) = *(px+i0) + *(px+i2);
                                        r2 = *(px+i1) - *(px+i3);
                                        *(px+i1) = *(px+i1) + *(px+i3);
                                        s1 = *(py+i0) - *(py+i2);
                                        *(py+i0) = *(py+i0) + *(py+i2);
                                        s2 = *(py+i1) - *(py+i3);
                                        *(py+i1) = *(py+i1) + *(py+i3);
                                        s3 = r1 - s2;
                                        r1 = r1 + s2;
                                        s2 = r2 - s1;
                                        r2 = r2 + s1;
                                        *(px+i2) = r1*cc1 - s2*ss1;
                                        *(py+i2) = -s2*cc1 - r1*ss1;
                                        *(px+i3) = s3*cc3 + r2*ss3;
                                        *(py+i3) = r2*cc3 - s3*ss3;
                                }
                                is = 2*id - n2 + j;
                                id = 4*id;
                        }
                }
        }

/*
--------------------Last stage, length=2 butterfly--------------------
*/
        is = 1;
        id = 4;
        while ( is < n) {
                for (i0 = is; i0 <= n; i0 = i0 + id) {
                        i1 = i0 + 1;
                        r1 = *(px+i0);
                        *(px+i0) = r1 + *(px+i1);
                        *(px+i1) = r1 - *(px+i1);
                        r1 = *(py+i0);
                        *(py+i0) = r1 + *(py+i1);
                        *(py+i1) = r1 - *(py+i1);
                }
                is = 2*id - 1;
                id = 4 * id;
        }

/*
-------------------------Bit reverse counter
*/
        j = 1;
        n1 = n - 1;
        for (i = 1; i <= n1; i++) {
                if (i < j) {
                        xt = *(px+j);
                        *(px+j) = *(px+i);
                        *(px+i) = xt;
                        xt = *(py+j);
                        *(py+j) = *(py+i);
                        *(py+i) = xt;
                }
                k = n / 2;
                while (k < j) {
                        j = j - k;
                        k = k / 2;
                }
                j = j + k;
        }

        /*
  for (i = 1; i<=16; i++) printf("%d  %g   %gn",i,*(px+i),(py+i));
*/

        return(n);

}
```

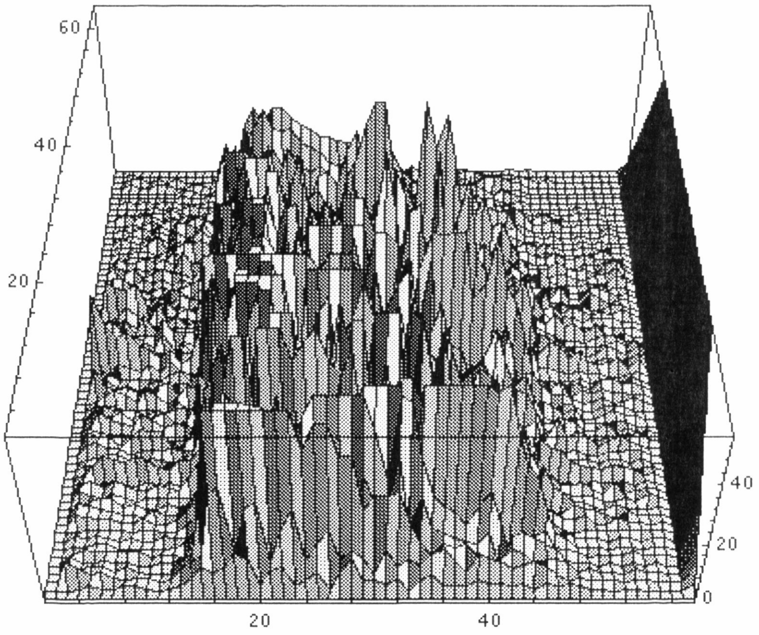
tables.h

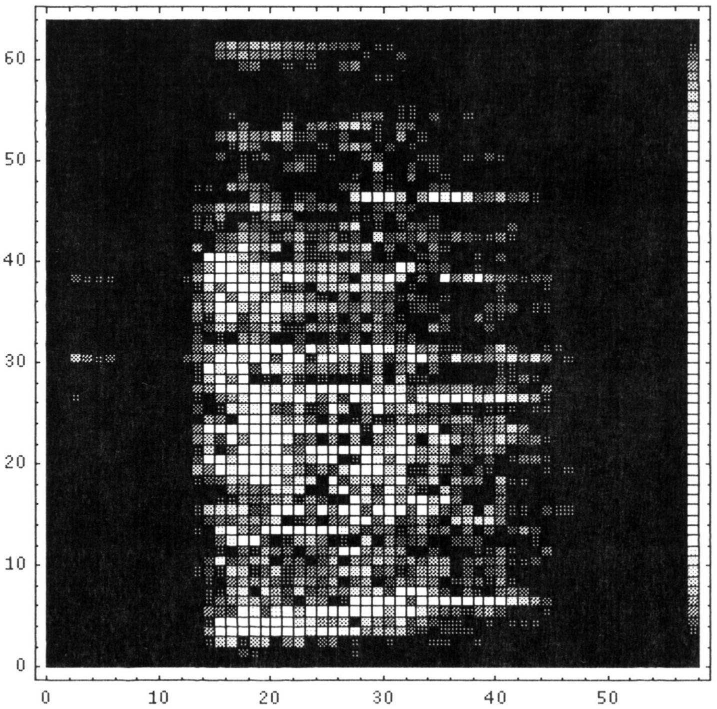
```
/*
tables.h
*/
double costable[] = {
1.000000000000000,0.999847695156391,0.999390827019097,0.998629534754576,
0.997564050259827,0.996194698091750,0.994521895368280,0.992546151641332,
0.990268068741583,0.987688340595153,0.984807753012227,0.981627183447687,
0.978147600733833,0.974370064785268,0.970295726276034,0.965925826289111,
0.961261695938368,0.956304755963091,0.951056516295215,0.945518575599386,
0.939692620785984,0.933580426497285,0.927183854566879,0.920504853452540,
0.913545457642709,0.906307787036767,0.898794046299294,0.891006524188504,
0.882947592859073,0.874619707139552,0.866025403784605,0.857167300702290,
0.848048096156614,0.838670567945624,0.829037572555253,0.819152044289215,
0.809016994375182,0.798635510047540,0.788010753606982,0.777145961457244,
0.766044443119264,0.754709580223071,0.743144825477706,0.731353701619496,
0.719339800338991,0.707106781186901,0.694658370459365,0.681998360062880,
0.669130606359255,0.656059028990918,0.642787609686965,0.629320391050278,
0.615661475326114,0.601815023152519,0.587785252292959,0.573576436351547,
0.559192903471263,0.544639035015558,0.529919264233751,0.515038074910616,
0.500000000000577,0.484809620246930,0.469471562786499,0.453990499740170,
0.438371146789716,0.422618261741354,0.406736643076470,0.390731128489959,
0.374606593416613,0.358367949546016,0.342020143326400,0.325568154457902,
0.309016994375708,0.292371704723512,0.275637355817790,0.258819045103326,
0.241921895600487,0.224951054344699,0.207911690818607,0.190808995377406,
0.173648177667806,0.156434465041120,0.139173100960968,0.121869343406063,
0.104528463268582,0.087155742748599,0.069756473745078,0.052335956243909,
0.034899496703478,0.017452406438272,0.000000000001000,-0.017452406436273,
-0.034899496701480,-0.052335956241912,-0.069756473743083,-0.087155742746607,
-0.104528463266593,-0.121869343404078,-0.139173100958987,-0.156434465039145,
-0.173648177665836,-0.190808995375443,-0.207911690816651,-0.224951054342750,
-0.241921895598547,-0.258819045101394,-0.275637355815867,-0.292371704721600,
-0.309016994373806,-0.325568154456012,-0.342020143324520,-0.358367949544149,
-0.374606593414758,-0.390731128488118,-0.406736643074643,-0.422618261739541,
-0.438371146787919,-0.453990499738389,-0.469471562784733,-0.484809620245181,
-0.499999999998845,-0.515038074908902,-0.529919264232055,-0.544639035013881,
-0.559192903469605,-0.573576436349908,-0.587785252291341,-0.601815023150921,
-0.615661475324538,-0.629320391048724,-0.642787609685433,-0.656059028989409,
-0.669130606357768,-0.681998360061418,-0.694658370457926,-0.707106781185487,
-0.719339800337601,-0.731353701618133,-0.743144825476368,-0.754709580221759,
-0.766044443117978,-0.777145961455985,-0.788010753605751,-0.798635510046337,
-0.809016994374007,-0.819152044288068,-0.829037572554134,-0.838670567944535,
-0.848048096155555,-0.857167300701260,-0.866025403783605,-0.874619707138583,
-0.882947592858134,-0.891006524187596,-0.898794046298417,-0.906307787035922,
-0.913545457641896,-0.920504853451759,-0.927183854566130,-0.933580426496569,
-0.939692620785300,-0.945518575598734,-0.951056516294597,-0.956304755962506,
-0.961261695937817,-0.965925826288594,-0.970295726275550,-0.974370064784818,
-0.978147600733418,-0.981627183447306,-0.984807753011880,-0.987688340594841,
-0.990268068741304,-0.992546151641088,-0.994521895368071,-0.996194698091576,
-0.997564050259688,-0.998629534754471,-0.999390827019027,-0.999847695156357,
-1.000000000000000,-0.999847695156426,-0.999390827019166,-0.998629534754680,
-0.997564050259967,-0.996194698091925,-0.994521895368489,-0.992546151641575,
-0.990268068741861,-0.987688340595466,-0.984807753012575,-0.981627183448069,
-0.978147600734249,-0.974370064785718,-0.970295726276518,-0.965925826289629,
-0.961261695938919,-0.956304755963675,-0.951056516295833,-0.945518575600037,
-0.939692620786668,-0.933580426498002,-0.927183854567628,-0.920504853453322,
-0.913545457643523,-0.906307787037613,-0.898794046300170,-0.891006524189412,
-0.882947592860012,-0.874619707140522,-0.866025403785605,-0.857167300703320,
-0.848048096157674,-0.838670567946713,-0.829037572556371,-0.819152044290362,
-0.809016994376358,-0.798635510048744,-0.788010753608213,-0.777145961458502,
-0.766044443120549,-0.754709580224383,-0.743144825479045,-0.731353701620860,
-0.719339800340380,-0.707106781188315,-0.694658370460803,-0.681998360064343,
-0.669130606360741,-0.656059028992427,-0.642787609688497,-0.629320391051832,
-0.615661475327689,-0.601815023154116,-0.587785252294576,-0.573576436353185,
-0.559192903472921,-0.544639035017236,-0.529919264235447,-0.515038074912330,
-0.500000000002309,-0.484809620248679,-0.469471562788265,-0.453990499741953,
-0.438371146791514,-0.422618261743166,-0.406736643078297,-0.390731128491800,
-0.374606593418467,-0.358367949547883,-0.342020143328279,-0.325568154459794,
-0.309016994377610,-0.292371704725425,-0.275637355819712,-0.258819045105257,
-0.241921895602427,-0.224951054346648,-0.207911690820563,-0.190808995379369,
-0.173648177669775,-0.156434465043095,-0.139173100962948,-0.121869343408047,
-0.104528463270571,-0.087155742750591,-0.069756473747073,-0.052335956245906,
-0.034899496705477,-0.017452406440272,-0.000000000003000,0.017452406434273,
0.034899496699481,0.052335956239915,0.069756473741089,0.087155742744614,
0.104528463264604,0.121869343402093,0.139173100957006,0.156434465037169,
0.173648177663867,0.190808995373481,0.207911690814694,0.224951054340801,
0.241921895596606,0.258819045099462,0.275637355813945,0.292371704719687,
0.309016994371905,0.325568154454120,0.342020143322641,0.358367949542282,
0.374606593412904,0.390731128486277,0.406736643072816,0.422618261737729,
0.438371146786121,0.453990499736607,0.469471562782968,0.484809620243431,
0.499999999997113,0.515038074907188,0.529919264230360,0.544639035012204,
0.559192903467947,0.573576436348270,0.587785252289722,0.601815023149324,
0.615661475322962,0.629320391047170,0.642787609683901,0.656059028987899,
0.669130606356282,0.681998360059955,0.694658370456488,0.707106781184073,
0.719339800336213,0.731353701616768,0.743144825475030,0.754709580220447,
0.766044443116693,0.777145961454726,0.788010753604519,0.798635510045133,
0.809016994372831,0.819152044286921,0.829037572553016,0.838670567943445,
0.848048096154495,0.857167300700230,0.866025403782606,0.874619707137613,
0.882947592857195,0.891006524186688,0.898794046297540,0.906307787035077,
0.913545457641083,0.920504853450978,0.927183854565381,0.933580426495852,
0.939692620784616,0.945518575598083,0.951056516293979,0.956304755961921,
0.961261695937266,0.965925826288076,0.970295726275066,0.974370064784368,
0.978147600733002,0.981627183446924,0.984807753011533,0.987688340594528,
0.990268068741026,0.992546151640844,0.994521895367862,0.996194698091402,
0.997564050259548,0.998629534754366,0.999390827018957,0.999847695156322};

double sintable [] = {
0.000000000000000,0.017452406437272,0.034899496702479,0.052335956242911,
0.069756473744081,0.087155742747603,0.104528463267587,0.121869343405070,
0.139173100959977,0.156434465040132,0.173648177666821,0.190808995376425,
0.207911690817629,0.224951054343724,0.241921895599517,0.258819045102360,
0.275637355816828,0.292371704722556,0.309016994374757,0.325568154456957,
0.342020143325460,0.358367949545082,0.374606593415685,0.390731128489039,
0.406736643075557,0.422618261740448,0.438371146788818,0.453990499739279,
0.469471562785616,0.484809620246055,0.499999999999711,0.515038074909759,
0.529919264232903,0.544639035014720,0.559192903470434,0.573576436350728,
0.587785252292150,0.601815023151720,0.615661475325326,0.629320391049501,
0.642787609686199,0.656059028990163,0.669130606358511,0.681998360062149,
0.694658370458646,0.707106781186194,0.719339800338296,0.731353701618814,
0.743144825477037,0.754709580222415,0.766044443118621,0.777145961456614,
0.788010753606366,0.798635510046938,0.809016994374595,0.819152044288641,
0.829037572554694,0.838670567945079,0.848048096156085,0.857167300701775,
0.866025403784105,0.874619707139067,0.882947592858604,0.891006524188050,
0.898794046298855,0.906307787036345,0.913545457642303,0.920504853452149,
0.927183854566504,0.933580426496927,0.939692620785642,0.945518575599060,
0.951056516294906,0.956304755962798,0.961261695938092,0.965925826288853,
0.970295726275792,0.974370064785043,0.978147600733625,0.981627183447496,
0.984807753012054,0.987688340594997,0.990268068741443,0.992546151641210,
0.994521895368176,0.996194698091663,0.997564050259758,0.998629534754523,
0.999390827019062,0.999847695156374,1.000000000000000,0.999847695156409,
0.999390827019131,0.998629534754628,0.997564050259897,0.996194698091838,
0.994521895368385,0.992546151641453,0.990268068741722,0.987688340595310,
0.984807753012401,0.981627183447878,0.978147600734041,0.974370064785493,
0.970295726276276,0.965925826289370,0.961261695938643,0.956304755963383,
0.951056516295524,0.945518575599711,0.939692620786326,0.933580426497644,
0.927183854567253,0.920504853452931,0.913545457643116,0.906307787037190,
0.898794046299732,0.891006524188958,0.882947592859542,0.874619707140037,
0.866025403785105,0.857167300702805,0.848048096157144,0.838670567946168,
0.829037572555812,0.819152044289788,0.809016994375770,0.798635510048142,
0.788010753607597,0.777145961457873,0.766044443119906,0.754709580223727,
0.743144825478376,0.731353701620178,0.719339800339685,0.707106781187608,
0.694658370460084,0.681998360063612,0.669130606359998,0.656059028991673,
0.642787609687731,0.629320391051055,0.615661475326901,0.601815023153317,
0.587785252293767,0.573576436352366,0.559192903472092,0.544639035016397,
0.529919264234600,0.515038074911473,0.500000000001443,0.484809620247804,
0.469471562787382,0.453990499741062,0.438371146790615,0.422618261742260,
0.406736643077383,0.390731128490879,0.374606593417539,0.358367949546950,
0.342020143327339,0.325568154458848,0.309016994376659,0.292371704724468,
0.275637355818751,0.258819045104291,0.241921895601457,0.224951054345673,
0.207911690819585,0.190808995378388,0.173648177668790,0.156434465042107,
0.139173100961958,0.121869343407055,0.104528463269576,0.087155742749595,
0.069756473746076,0.052335956244908,0.034899496704477,0.017452406439272,
0.000000000002000,-0.017452406435273,-0.034899496700480,-0.052335956240913,
-0.069756473742086,-0.087155742745611,-0.104528463265598,-0.121869343403085,
-0.139173100957997,-0.156434465038157,-0.173648177664851,-0.190808995374462,
-0.207911690815673,-0.224951054341776,-0.241921895597576,-0.258819045100428,
-0.275637355814906,-0.292371704720643,-0.309016994372855,-0.325568154455066,
-0.342020143323581,-0.358367949543215,-0.374606593413831,-0.390731128487198,
-0.406736643073730,-0.422618261738635,-0.438371146787020,-0.453990499737498,
-0.469471562783850,-0.484809620244306,-0.499999999997980,-0.515038074908045,
-0.529919264231208,-0.544639035013042,-0.559192903468776,-0.573576436349089,
-0.587785252290531,-0.601815023150123,-0.615661475323750,-0.629320391047947,
-0.642787609684667,-0.656059028988654,-0.669130606357025,-0.681998360060686,
-0.694658370457207,-0.707106781184780,-0.719339800336907,-0.731353701617450,
-0.743144825475699,-0.754709580221103,-0.766044443117335,-0.777145961455356,
-0.788010753605135,-0.798635510045735,-0.809016994373419,-0.819152044287494,
-0.829037572553575,-0.838670567943990,-0.848048096155025,-0.857167300700745,
-0.866025403783105,-0.874619707138098,-0.882947592857665,-0.891006524187142,
-0.898794046297979,-0.906307787035500,-0.913545457641489,-0.920504853451368,
-0.927183854565755,-0.933580426496210,-0.939692620784958,-0.945518575598409,
-0.951056516294288,-0.956304755962214,-0.961261695937541,-0.965925826288335,
-0.970295726275308,-0.974370064784593,-0.978147600733210,-0.981627183447115,
-0.984807753011707,-0.987688340594684,-0.990268068741165,-0.992546151640966,
-0.994521895367967,-0.996194698091489,-0.997564050259618,-0.998629534754419,
-0.999390827018992,-0.999847695156339,-1.000000000000000,-0.999847695156444,
-0.999390827019201,-0.998629534754733,-0.997564050260037,-0.996194698092012,
-0.994521895368594,-0.992546151641697,-0.990268068742000,-0.987688340595623,
-0.984807753012748,-0.981627183448260,-0.978147600734457,-0.974370064785943,
-0.970295726276760,-0.965925826289888,-0.961261695939195,-0.956304755963968,
-0.951056516296142,-0.945518575600362,-0.939692620787010,-0.933580426498360,
-0.927183854568003,-0.920504853453712,-0.913545457643929,-0.906307787038035,
-0.898794046300609,-0.891006524189866,-0.882947592860481,-0.874619707141007,
-0.866025403786105,-0.857167300703835,-0.848048096158204,-0.838670567947258,
-0.829037572556930,-0.819152044290935,-0.809016994376946,-0.798635510049346,
-0.788010753608829,-0.777145961459131,-0.766044443121192,-0.754709580225039,
-0.743144825479714,-0.731353701621542,-0.719339800341075,-0.707106781189022,
-0.694658370461523,-0.681998360065075,-0.669130606361484,-0.656059028993182,
-0.642787609689263,-0.629320391052609,-0.615661475328477,-0.601815023154914,
-0.587785252295386,-0.573576436354004,-0.559192903473749,-0.544639035018074,
-0.529919264236296,-0.515038074913187,-0.500000000003175,-0.484809620249554,
-0.469471562789148,-0.453990499742843,-0.438371146792413,-0.422618261744073,
-0.406736643079210,-0.390731128492720,-0.374606593419394,-0.358367949548817,
-0.342020143329218,-0.325568154460739,-0.309016994378561,-0.292371704726381,
-0.275637355820673,-0.258819045106223,-0.241921895603398,-0.224951054347621,
-0.207911690821542,-0.190808995380351,-0.173648177670760,-0.156434465044082,
-0.139173100963938,-0.121869343409040,-0.104528463271565,-0.087155742751588,
-0.069756473748071,-0.052335956246905,-0.034899496706476,-0.017452406441272};
```

APPENDIX C

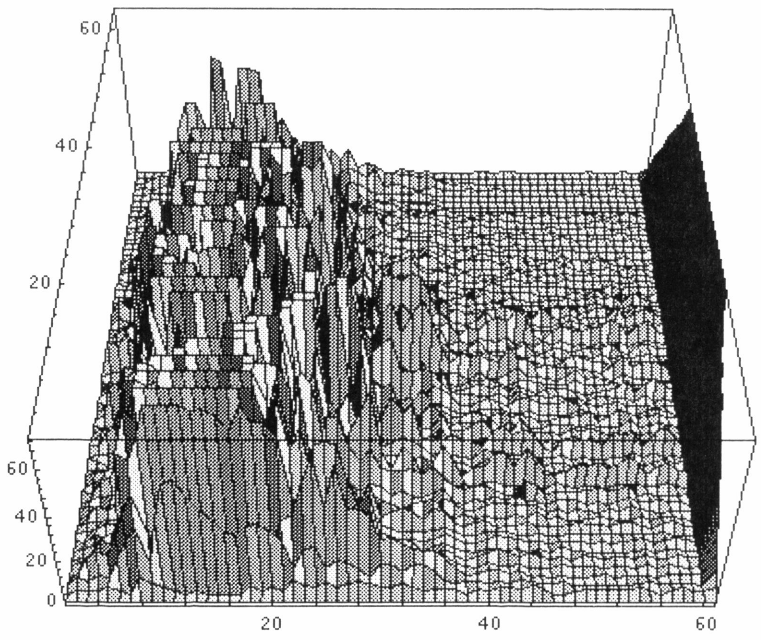
Speech Waveforms

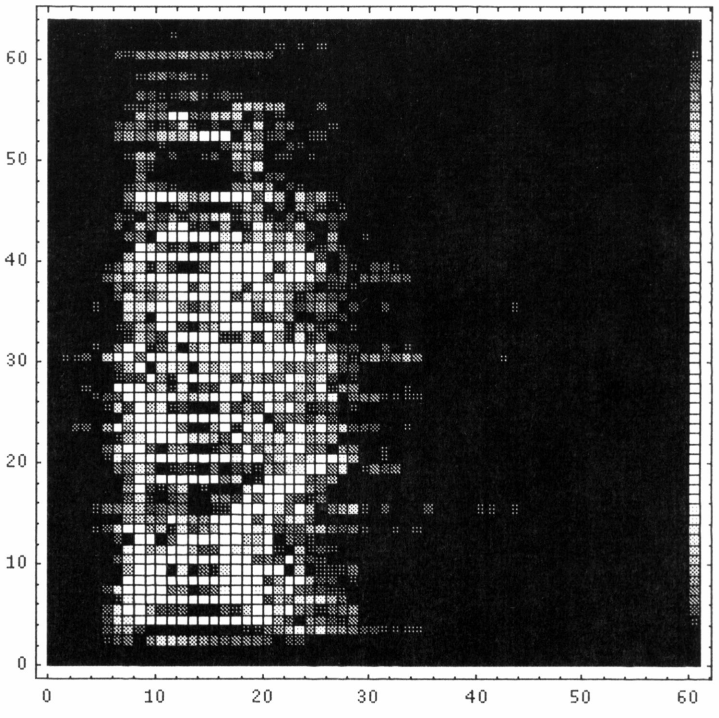
Zero



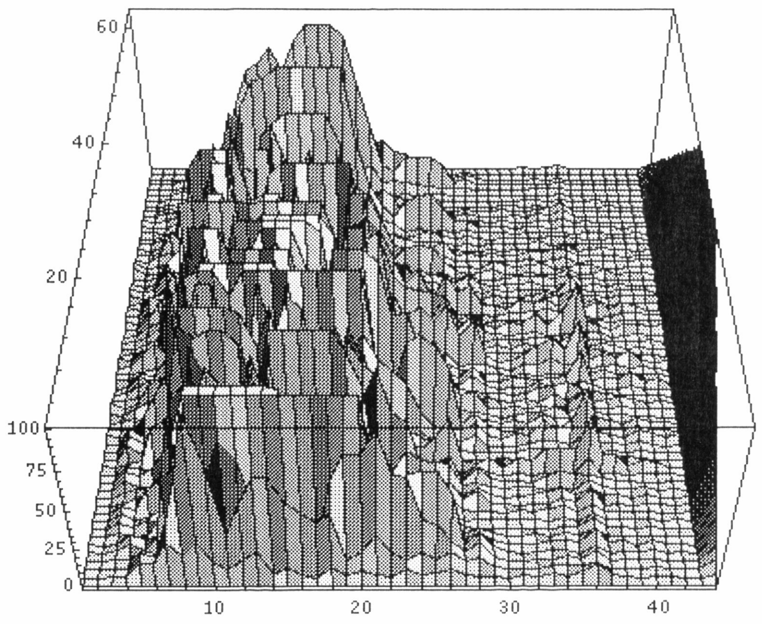


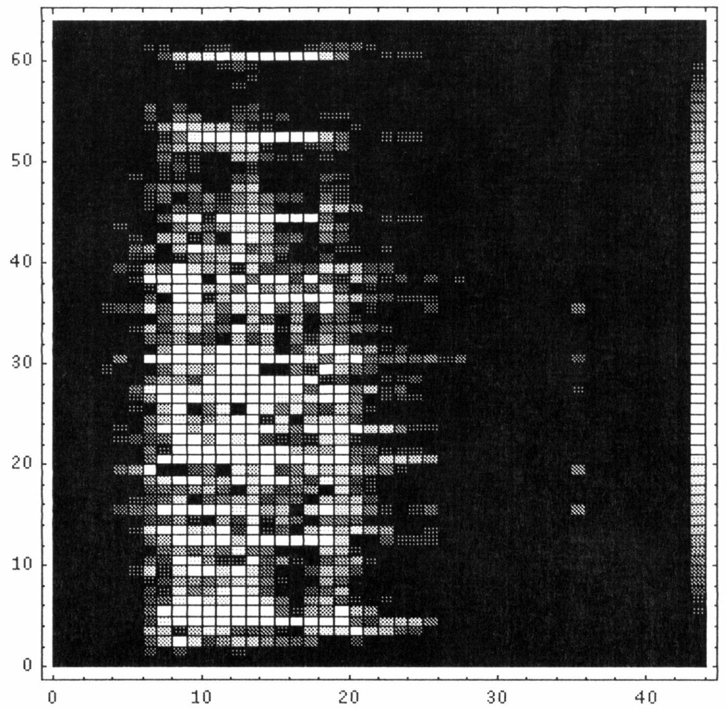
One



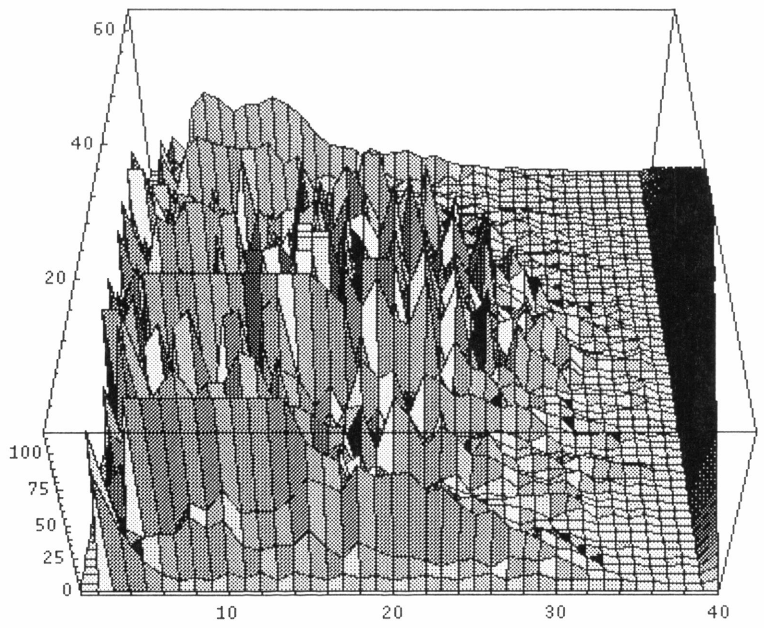


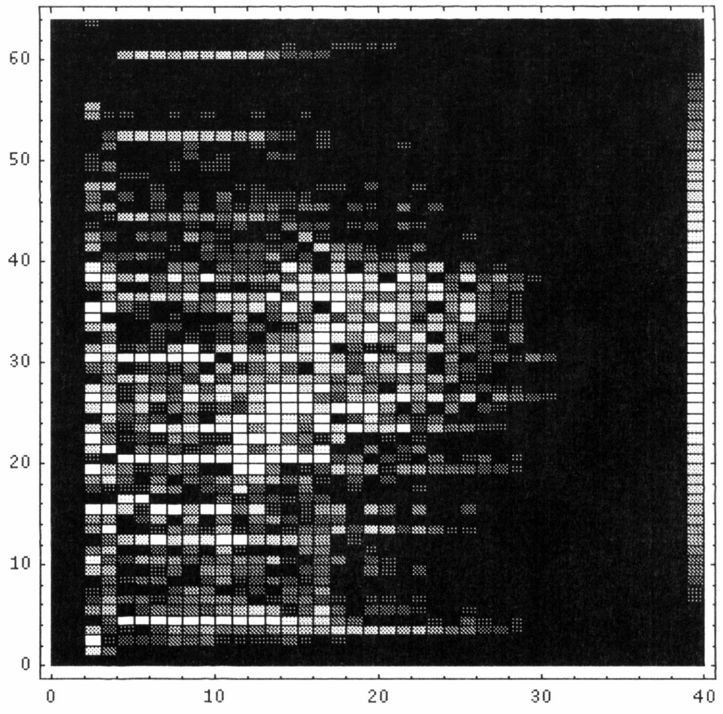
Two



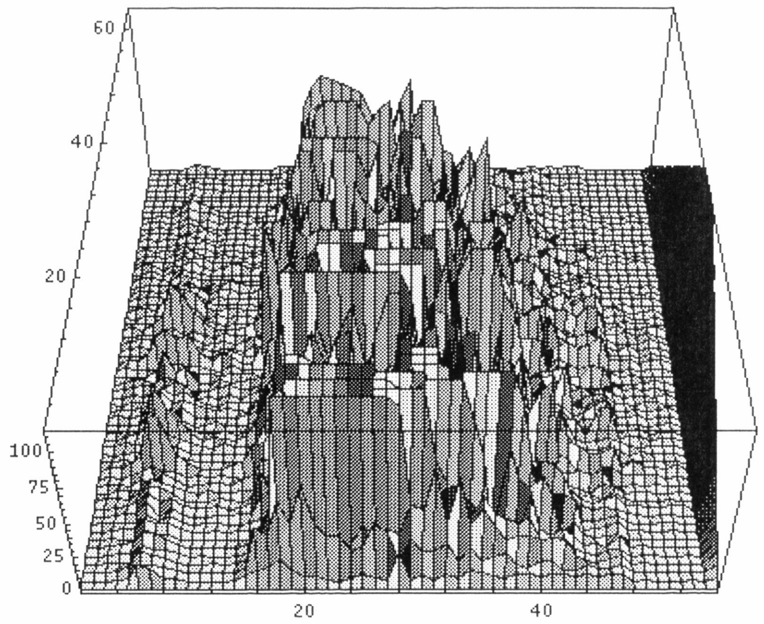


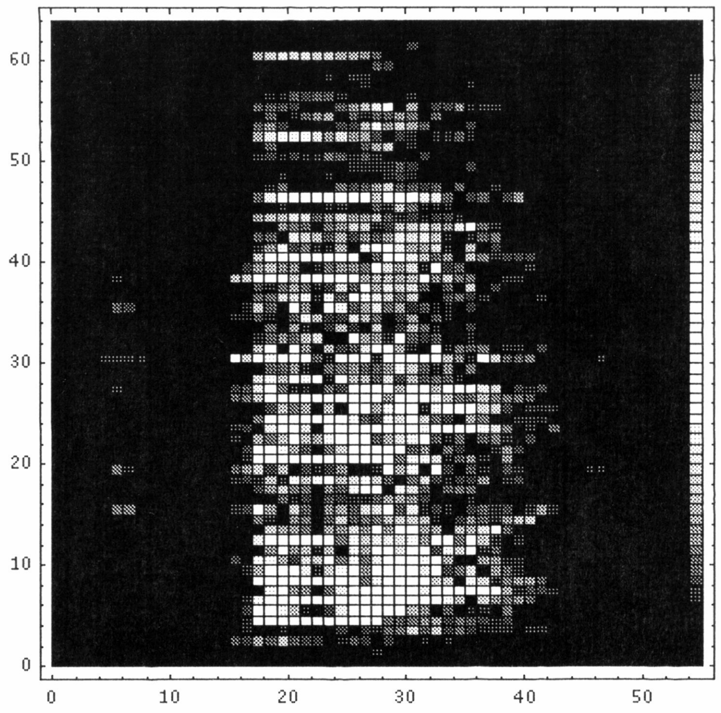
Three



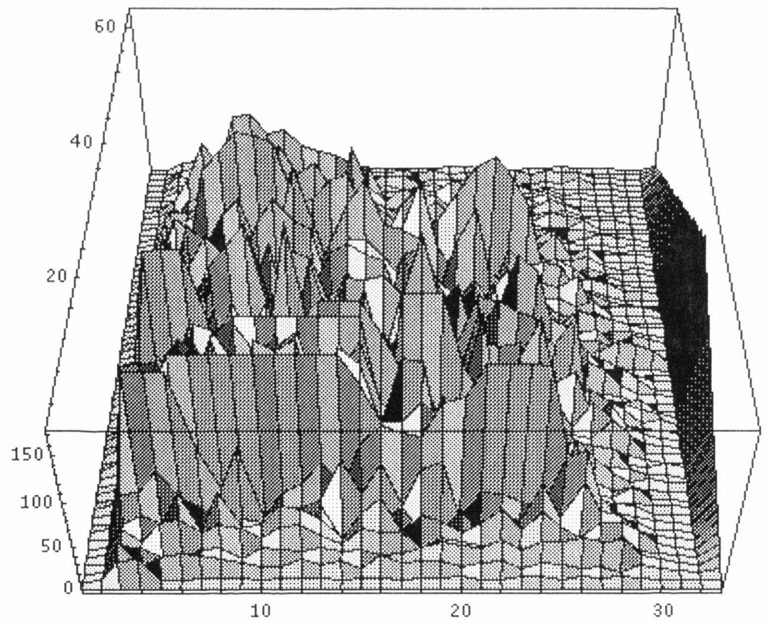


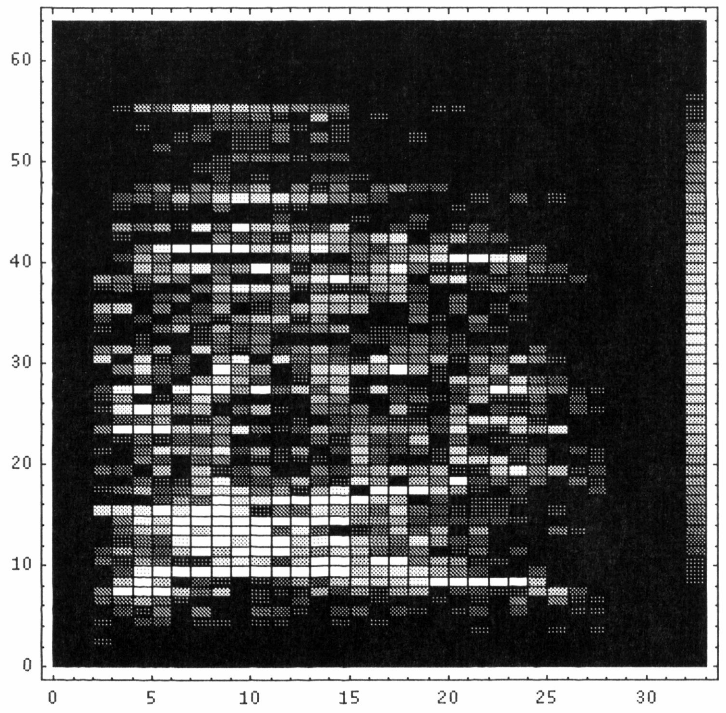
Four



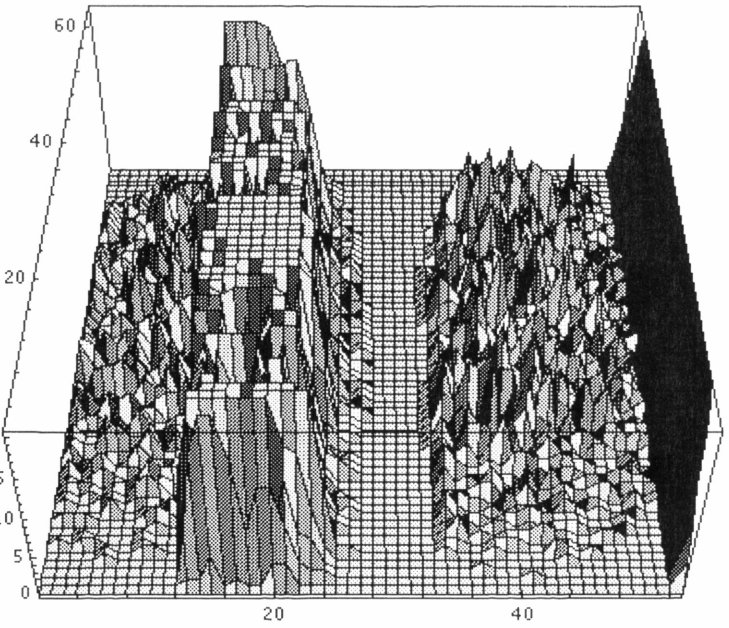


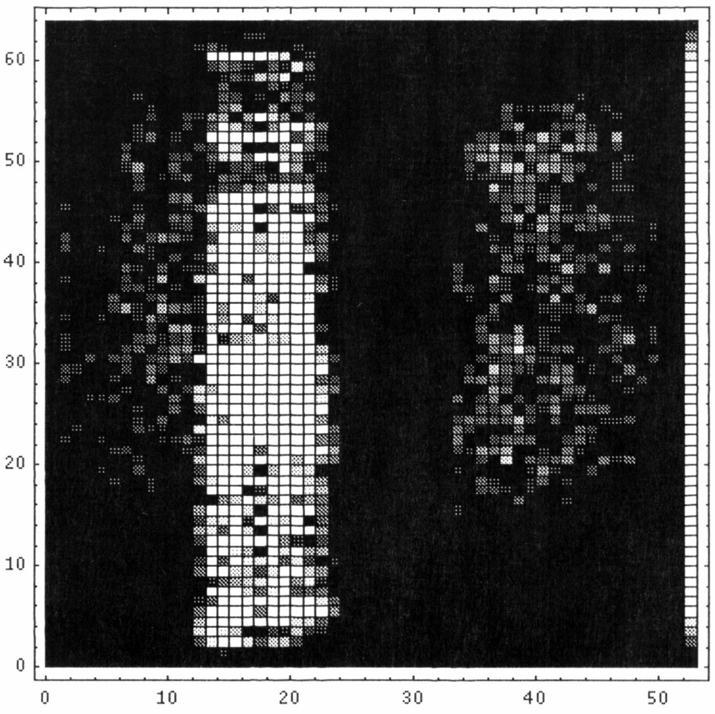
Five



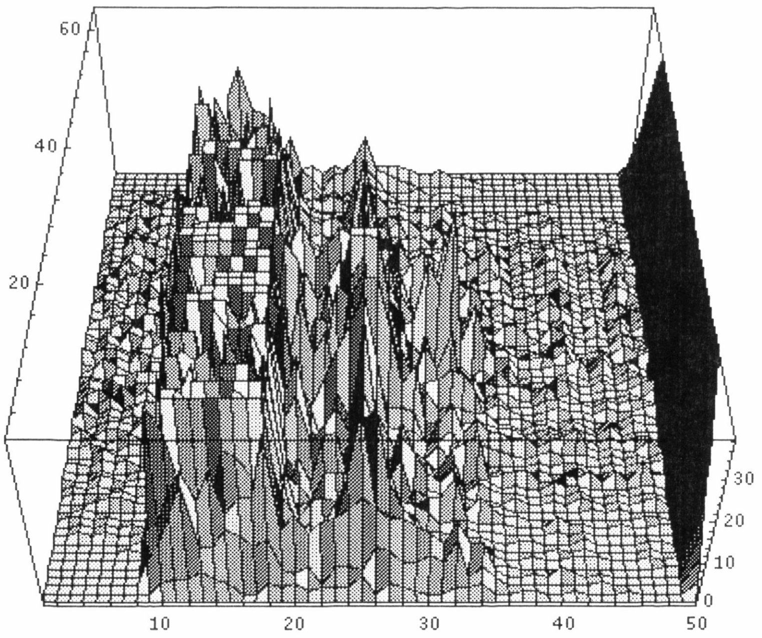


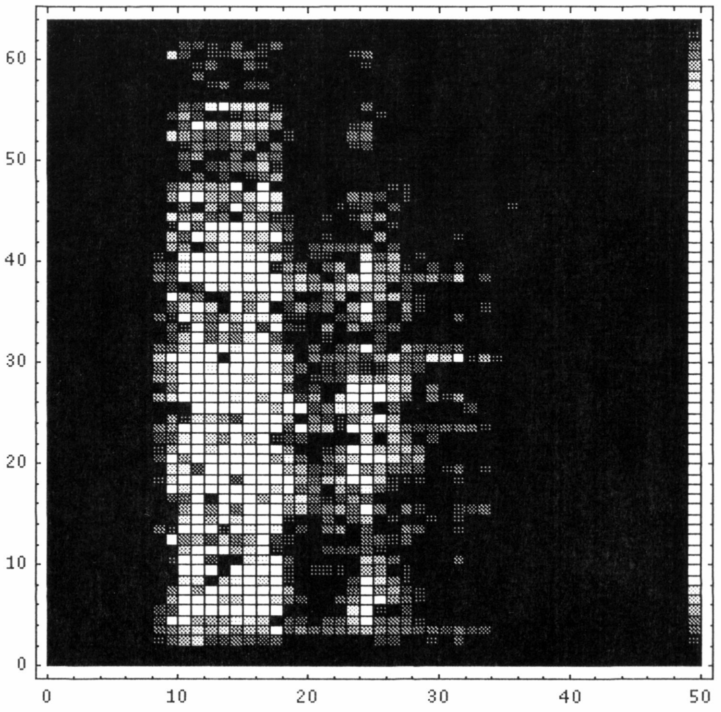
Six



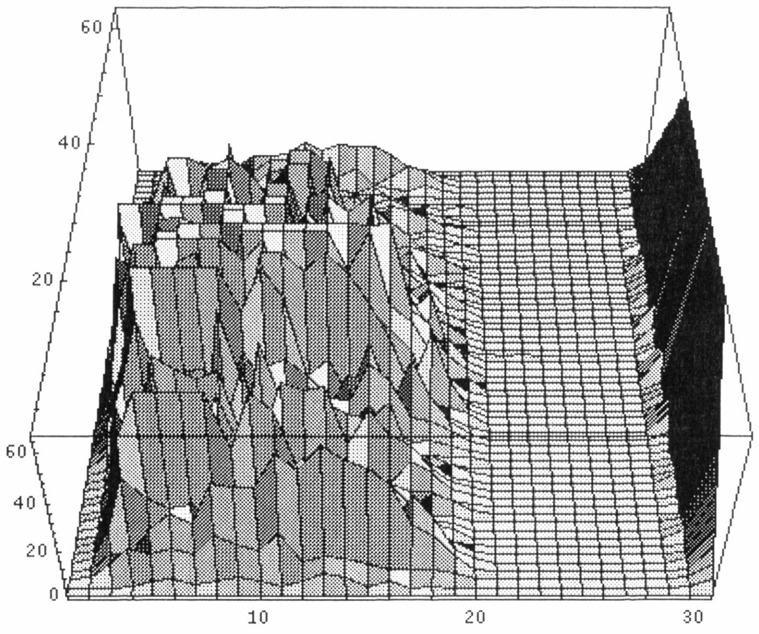


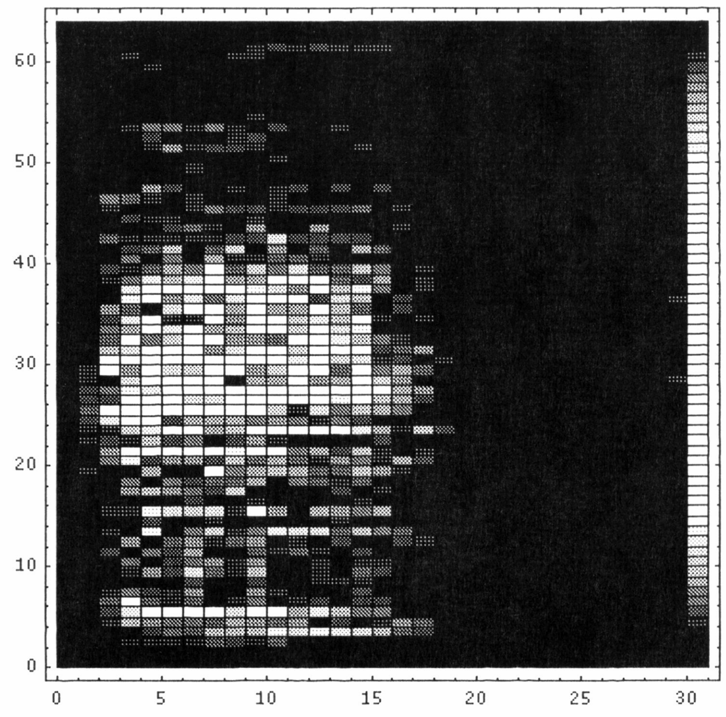
Seven



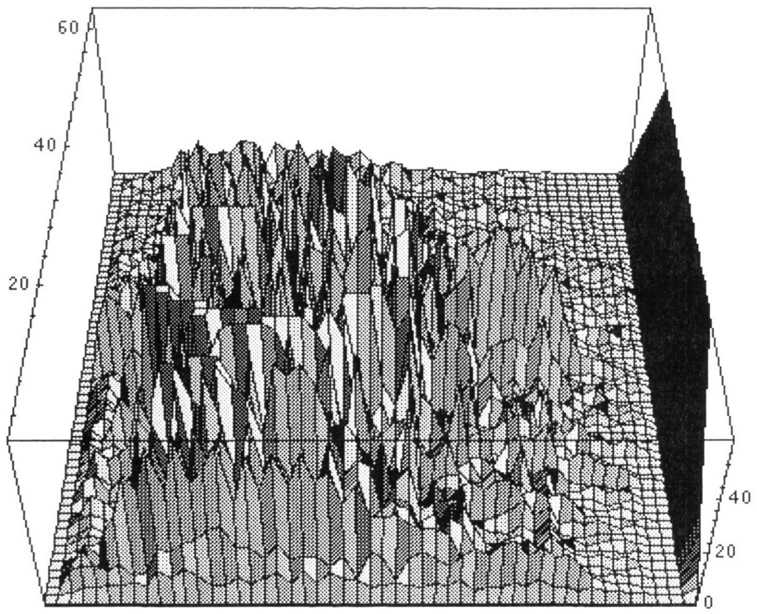


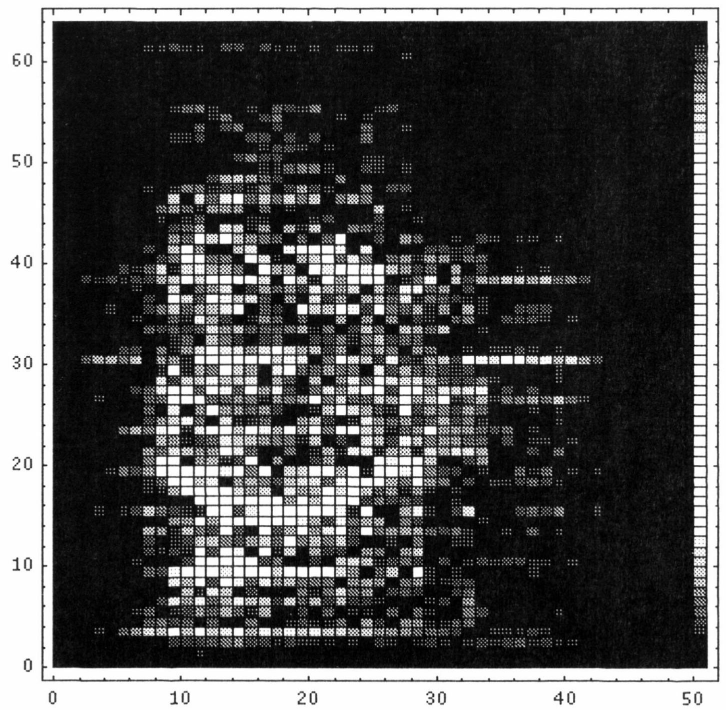
Eight





Nine


60
40
20
10
20
30
40
50
0
20
40


60
50
40
30
20
10
0
0
10
20
30
40
50