Term Project for Computational Physics, Spring 1997

Synthesizer for Wave-Files

by Richard Metzler

Abstract

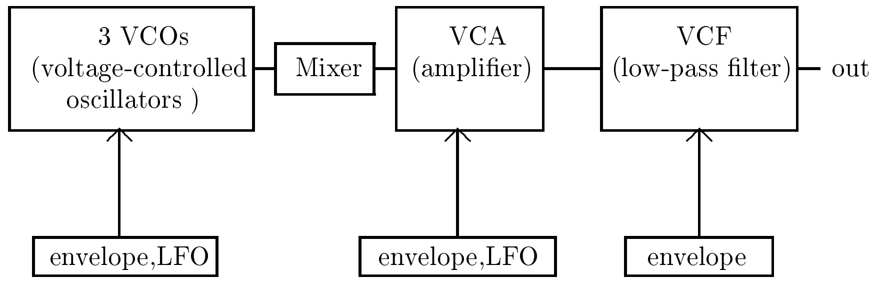
The program presented here loads RIFF-Wave-files, subjects them to a standard synthesizer treatment including manipulation of frequency, amplitude and harmonic structure, and saves the modified files as wave-files again.

Introduction

The history of musical synthesizers essentially began in 1968, when Robert Moog developed his modular synthesizer system. It consisted of an arbitrary number of modules, each responsible for either creating a wave (choosing from the standard waveforms like sine, square and sawtooth), creating an envelope or manipulating a signal, e.g. modulating the amplitude of one input (the wave) with that of another (the envelope), and a keyboard. The modules were connected by patch cords.

This modular system allowed the creation of spectacular, unheard-of sounds. It was also expensive, bulky, unreliable, and difficult to program, such that it was used only by few artists, among them Walter Carlos, Keith Emerson of Emerson, Lake and Palmer and Klaus Schulze.

Bob Moog decided to construct a reduced, easy-to-use, portable version of the modular system, called the Minimoog. It was a huge success, becoming the workhorse for the keyboardists of countless rock bands (among them Yes, Rush, Eloy, Rainbow etc.) and setting the structural standards for the majority of synthesizers that were developed since. Its structure is roughly the following:



LFO stands for low frequency oscillator, typically a sine wave applied for vibrato or tremolo.

The technology used in Moog synthesizers was purely analog, as was to be expected in the early 70s. In the beginning of the 80s, digital synthesizers and digital-analog hybrids hit the market. With the exception of FM-based synthesizers (Yamaha DX7 and related machines) and occasional additive-synthesis generators, all of these followed the structure sketched above, however replacing the simple oscillators with other sources like sampled bits of natural instruments and the analog filters with digital ones.

How to use this program

The program `synth` presented here is a non-realtime synthesizer that loads samples in the form of RIFF-wave-files (.wav, the standard for Windows-applications), modulates their frequency, amplitude and harmonic structure and saves them again as RIFF-files. The parameters for envelopes and LFOs can be entered through a graphical user interface.

It takes a maximum of two command line parameters, namely the name of the file to be processed (default: in.wav) and the name of the output file (default: out.wav).

Main menu

The main menu gives information about the input file and offers five options: - recalculate

uses the parameters entered so far to load, process and save the input file

- frequency page

branches to the frequency menu, where the parameters for the frequency

envelope and LFO can be modified and the envelope is visualized
- amplitude page
  very much like frequency page, but for the amplitude parameters
- filter page
  like the two options before, but instead of an LFO the resonance parameter of the filter can be set
- exit
  does what it says

Submenus

To alter a parameter, click on the corresponding button and click on "increase" or "decrease" or press "+" or "-", respectively. The new value will be displayed, and the envelope will be redrawn. All envelope values run from 0 to 1. The times correspond to the position in the sample relative to its total length.

Frequency The sum of envelope value and LFO value at any given time determines how much the sample will be stretched (tuned down) or compressed (sped up). A value of 0 corresponds to a stretch factor of 2 or a shift by one octave down, while a value of 1 means a compression of 2 or one octave up. 0.5 means an unmodified frequency.

Amplitude

Here, the sum of envelope and LFO give a factor by which the amplitude is multiplied. A value of 1 means unmodified volume. Amplifications that are greater than 1 (by means of LFO settings) can lead to digital clipping — not a desirable effect.

Filter

The filter implemented here is a resonant low-pass filter with -12dB/octave attenuation. The envelope sets the cutoff frequency to a value between 0 and the Nyquist frequency (sampling frequency/2). The resonance parameter determines how much the cutoff frequency is emphasized. High resonance values can again lead to clipping, which can be countered by lowering the amplitude.

The algorithms

Recalculate

This function is at the heart of the program. It is divided into three steps corresponding to the three submenus. In each step the file is loaded in small chunks (2048 samples per channel), modified and saved to a temporary or final file. This may seem unnecessarily complicated and inefficient; however, there are reasons for this method:
- the frequency modulation changes the length of the sample; that means that there would have to be one original array the size of the sample and at least one, probably two that are more than twice as long (due to sample stretching). Many soundfiles include 10^6 or more samples, each corresponding to 4 bytes. The memory used up would be appreciable even for well-equipped machines.
- the modular character of the program is emphasized. An arbitrary number of new algorithms can be inserted with little effort and no additional use of RAM.

Frequency: The input wave is essentially resampled in steps whose width is determined by the value of the function timestretch at the given time. When the array that contains the y-values of the resampled function is handed to AFwriteData, they are interpreted as equidistant samples, resulting in an effective stretch or compression of the data. The timestrech function is an exponential function scaled such that it maps 0 onto 0.5, 0.5 onto 1 and 1 onto 2:

$$timestretch(t) = 0.5 \cdot 2^{env(t)+LFO(t)/3}$$

For resampling, the program interpolates linearily between neighboring sample points.

Amplitude:
This one is straightforward: the input wave is loaded, multiplied element by element with the corresponding value of the amplitude envelope plus amplitude LFO, and saved again.

Filter:
The input file is loaded, again in blocks of 2048 samples/channel. This corresponds to roughly 0.1 seconds at a sample rate of 22.05 kHz. It is assumed that the signal contains frequencies of 20 Hz and more, such that all frequencies are represented in each block. The block is padded with an equal number of zeros to avoid wrapping effects and subjected to a fast fourier transformation.

To achieve a higher time resolution than the mentioned tenth of a second, each block of samples is multiplied with the respective time-dependent response function at eight equidistant times, then transformed back, and only the corresponding eighth of the reformed wave is kept.

The implemented filter function corresponds to the frequency response of a damped driven oscillator:

$$res(f,t) = f_c(t)^2 / \sqrt{(f^2 - f_c(t)^2)^2 + r \cdot f^2}$$

, where f_c is the cutoff frequency set by the evelope. r is calculated from the resonance parameter by

$$r = 2 \cdot f_c^2 \cdot (1 - (1 - \Delta) \cdot resonance)$$

where Δ is a small number that prevents a resonance catastrophe, enforcing a small, but non-zero damping. This choice of r guarantees that the filter acts as a regular low-pass filter if the resonance is set to 0.
It was also necessary to add a small quantity ϵ to both the denominator and the numerator to avoid a numerical divergence when both f and f_c drop to 0.

Envelopes:
Each envelope has eight adjustable parameters, corresponding to four points in the time-amplitude plane between which the envelope function interpolates linearily. In addition, there is the non-adjustable parameter lvl0, which determines where the envelope starts (can be overridden by lvl1 if time1 is set to 0). As mentioned, the time is relative to the length of the sample.
LFO:
The LFO provides four waveforms (sine, square, upward sawtooth, and triangle) that can be adjusted in frequency from 0 to 20 Hz. There is also an offset parameter which determines at what point in the sample the LFO reaches its full effect. It will fade in linearily from $t = offset - 0.1$.

The Submenus

Each submeny has the same structure: there is a window that displays the corresponding envelope, writes the current values of the parameters and has buttons for each parameter, an increase and a decrease button, and a leave button. Every time one of the parameter buttons is pressed, the value and the number of the parameter are written to temporary variables. If decrease or increase are activated, a subroutine (*renew(...)) increases the chosen parameter and calls *write(), which prints the current values to the screen and calls dravenv(...) to update the envelope.

Possible Expansions of the Program

The possibilities of data manipulation in the framework of this program are limited only by the imagination and spare time of the programmer. It would be easy to include processing steps that add effects like echo, chorus or distortion (by applying a nonlinear function to the amplitude of each sample). As the program is not bound by the contraints of causality, even true backwards effects are possible, like reverse reverb.

The filter can easily be changed to any conceivable response function, for example high- or bandpass filters, filters with time-dependent resonance, formant filters or any arbitrarily bizzare function. By adding only a few lines, it can also be used as a powerful analysis tool that puts out the time-dependent harmonic structure of the signal (before or after the application of the filter).

By replacing the AFsp functions with appropriate other input/output-routines, the synthesizer can be applied to any kind of sample or time series, which is particularly interesting in connection with the filters that could be used.

The creation of waves is not necessarily confined to using samples. For example, it would be possible to implement frequency modulation or additive synthesis modules or even try physical modeling (solving differential equations for interacting nonlinear oscillators to get complex waveforms).

To increase user-friendliness it would be possible to allow the user to save and load sets of parameters, and to add, append and cut samples. Also, if the program is installed on a computer with sound output facilities, a function that plays the created sample would make things easier.

Appendix A: Required Libraries & Installation

The program synth resides in **usr/people/metzler/synth/**. To compile properly, the following additional libraries are required:
-**AFsp** is a library to read, write and extract information from sound files, available for example at
www.unix.digital.com/demos/multimedia/html/afsp.html
-**Xgraphics** is a graphical user interface written by Martin Lueders, a physics grad student at the Universitaet Wuerzburg, Germany. It can be found at ftp.physik.uni-wuerzburg.de
-**fftpack** contains routines for fast fourier transformation. Any library that does this will do, if the functions srffti_ (initialization), srfftf_ (forward transformation) and srfftb_ (back transformation) are replaced with corresponding functions. **Note:** The version of AFsp currently installed on my account had to be modified to be compatible with fftpack. Before its installation, the Makefiles had to be changed to include the compilation option -n32.
Also note that the program is not confined to the RIFF-wave-format. It should automatically read most conventional formats, and the output format can be set by changing the format parameter in the last AFopenWrite-command. For details see the manual.

Used Subroutines

AFsp: The manual entries for the routines AFopenRead, AFopenWrite, AFreadData, AFwriteData and AFclose that were used in synth can be found in /usr2/people/metzler/synth/AFsp-V2R2/man/libtsp.
Xgraphics: The manual for Xgraphics is only available in German (to my knowledge, at least). Here is an overview over the used functions:
void InitX() initializes Xgraphics
void ExitX() closes all windows, clears memory
Window CreateWindow(int width, int height, char* name) creates a window of given width and heigth
void ShowWindow(Window win) actually displays the named window
void HideWindow(Window win) removes the specifid window from the screen, but not from memory.
World CreateWorld(Window win, int px, int py, int pwidth, int pheight, float wx1, float wy1, float wx2, float wy2, int scalable, int gravity) creates a world (an area) in the window win. The world has one corner at (px,py), has dimensions (pwidth, pheight), and its opposite corners have the effective (for manipulations within the window) coordinates (wx1,wy1) and (wx2,wy2).
void InitButtons(Window win, const char* buttonstring, int width) generates a

row of buttons on the right side of win. Buttonstring contains three informations for each element, divided by commas:
b for a button or t for text;
label of button or text;
string returned upon pressing button.
int GetEvent(XEvent* event, int waitflag) hands next event in event-cue to varable event. Possible event types are: Expose: a previously covered window was exposed. Possibly the contents of the window have to be renewed. KeyPress: a key was pressed (or a Button on the screen was clicked on). ButtonPress: a mouse button was pressed.
char ExtractEvent(Xevent event) reads the ASCII code of the key that was pressed.
void ClearArea(Window win, int x, int y, int width, int height) clears the specified area.
void WDrawLine(World world, float x1, float y1, float x2, float y2, int c) draws line of color c between coordinates (x1,y1) and (x2,y2).
WDrawString(World world, float x, float y, const char *text, int c) draws string text, beginning at specified point.

Appendix B: Wavemaker

The program wavemaker, which resides in the subdirectory /synth/wavemaker, is an auxiliary program to generate raw material for treatment by synth. The frequency and duration of the desired wave can be set in the declarations, and an arbitrary (but desirably periodic) function can be implemented in the function wave.

Source Code of Wavemaker

```
/*   wavemaker - auxiliary program for synth                             */
/*   delivers arbitrary waveform with desired length & frequency         */
/*   as raw material for manipulation                                    */
/*   sorry, no time to program a nice interface here...                  */

#include <math.h>
#include <string.h>
#include "/usr2/people/metzler/synth/AFsp-V2R2/include/libtsp.h"

float buffer[3000];
float time = 3.0;         /* length of sample in seconds                  */
float frequency = 220.00; /* basic frequency of wave                      */
float pi = 3.1415927;
float amp = 5000;         /* volume; total output should not exceed 16000 */
float samplerate =16000.0;/* common rates: 16 kHz, 22.05 kHz, 44.1 kHz    */

float wave(float phase);

int nsteps;
int i,j;

AFILE *outptr;char
outfile[60]= "wave.wav";

void main()
{
nsteps = (int)(1.0/(frequency*samplerate));

for(i=0; i<nsteps;i++)
```

```
{buffer[i] = amp*wave(2*pi*(float)(i)/(float)(nsteps));}

outptr=AFopenWrite(outfile,261,1,samplerate,NULL);

for(j=0; j<(int)(time*frequency);j++)
{AFwriteData(outptr,buffer,nsteps);}

AFclose(outptr);
}

/* insert any periodic (or nonperiodic) function here */

float wave(float phase)
{return (sin(phase)+0.6*sin(1.5*phase)+ 0.4*sin(2.0*phase)+0.3*sin(3*phase));}
```

Appendic C: Source Code of Synth

```
/*^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^*/
/*     synth - the synthesizer treatment for YOUR wavefiles!       */
/*           term project for Computational Physics                */
/*                       spring 1997                               */
/*                   by Richard Metzler                            */
/*_________________________________________________________________*/

/*-----------------------------------------------------------------*/
/*                   include-files                                 */
/*-----------------------------------------------------------------*/


#include <stdio.h>
#include <math.h>
#include <string.h>
#include "Xgraphics.h"
#include "/usr2/people/metzler/synth/AFsp-V2R2/include/libtsp.h"



/*-----------------------------------------------------------------*/
/*                    declarations                                 */
/*-----------------------------------------------------------------*/


/* GUI windows and worlds */

Window mainwindow, freqwindow, ampwindow,filtwindow;
World  mainworld, mainwrite;

World fvalues, feg, avalues, aeg, ivalues, ieg;

XEvent mainmenu;

/* functions for envelopes etc. */
```

```
float env(float lvl0, float lvl1,float lvl2, float lvl3, float lvl4,
               float time1, float time2, float time3, float time4,
               float time);
float lfo(int form, float offset,float pos,float amp, float freq, float time);


/* subroutines for editing parameters */

void mainrenew();

void freqpage();
void frenew(char menu, float ftmpfloat);
void freqwrite();

void amppage();
void arenew(char amenu, float atmpfloat);
void awrite();

void filtpage();
void irenew(char menu, float tmpfloat);
void iwrite();

void drawenv(World w, float lvl0, float lvl1, float lvl2,
float lvl3, float lvl4, float tm1, float tm2, float tm3, float tm4);


/* subroutines for calculating modified wave file */

void recalculate();

float timestretch(float time, float pos);
float lint(float y[], float x);
float resfct(float f,float f0, float resonance);

/* functions from fftpack  */

void srffti_(int *n, float wsave[]);
void srfftf_(int *n, float r[], float wsave[]);
void srfftb_(int *n, float r[], float wsave[]);
```

```
/* some flags */

int    done = 0;
int    stop = 0;



/* names for input and output files   */

char   infile[60]  = "in.wav";
char   tempfile[30] = "tmp.wav";
char   tem2file[30] = "tm2.wav";
char   outfile[60]= "out.wav";


AFILE *inptr;
AFILE *tmpptr;
AFILE *outptr;

/*  counters for old and new sample      */

long int   channels;
long int   length1,length2;

/* sample rate                           */

float  samplerate;

/* parameters                            */

float  flvl0 = 0.5, flvl1=0.5,flvl2=0.5,flvl3=0.5,flvl4=0.5;
float  ftime1=0.25,ftime2=0.5,ftime3=0.75,ftime4=1.0;
float  flforate=0.3,flfooffset=0.2,flfoamp= 0.00;
int    flfoform= 1;

float  alvl0=0,alvl1=1.0, alvl2=0.8, alvl3=0.7,alvl4=0.0;
float  atime1=0.05,atime2=0.15,atime3=.75,atime4=1.0;
```

```
float  alforate= 0.3,alfooffset=0.2, alfoamp = 0.0;
int    alfoform=1;

float  ilvl0=0,ilvl1=1,ilvl2=0.8,ilvl3=0.7,ilvl4=0.0;
float  itime1=0.1, itime2=0.25, itime3=0.75, itime4=1.0;
float  resonance=0;

/*----------------------------------------------------------------*/
/*                        main menu                               */
/*----------------------------------------------------------------*/

void main(int argc, char* argv[3]) {

int j;
char wstring[40];

InitX();
mainwindow = CreateWindow(500,265,"Synth - Main Menu");

freqwindow = CreateWindow(600, 530, "Frequency menu");
ampwindow  = CreateWindow(600, 530, "Amplitude menu");
filtwindow = CreateWindow(600, 465, "Filter menu");

if(argc>1)
  {strcpy(infile, argv[1]);}
if(argc>2)
  {strcpy(outfile,argv[2]);}
if(argc>3)
  {fprintf(stderr, "Usage: synth [<input.wav>] [<output.wav>]\n");
   fprintf(stderr, "Additional arguments ignored\n");}

/*   show parameters of input file   */

inptr= AFopenRead(infile, &length1, &channels, &samplerate, NULL);
AFclose(inptr);

mainwrite= CreateWorld(mainwindow, 0, 0, 300, 260, 0.0,0.0,300.0,260.0,0,0);

mainrenew();
```

```
ShowWindow(mainwindow);

while(done == 0)
 {
   if (GetEvent(&mainmenu,1)){
     switch(mainmenu.type) {
     case Expose:
          mainrenew();
  break;
     case KeyPress:
       switch(ExtractChar(mainmenu)){
       case 'r':
          recalculate();
  mainrenew();
          break;
       case 'f':
          freqpage();
  mainrenew();
          break;
       case 'a':
          amppage();
  mainrenew();
          break;
       case 'i':
          filtpage();
  mainrenew();
          break;
       case 'x':
          done = 1;
          break;

       } /*switch*/
     }  /*switch */
   }   /*if*/


 }    /*while*/
```

```
ExitX();

}  /* end of main*/

/*-----                 ---------------------------------------*/
/* mainrenew writes stuff to the main menu when necessary      */
/*-----                 ---------------------------------------*/

void mainrenew()
{
char wstring[60];

sprintf(wstring,"Input file:        %s",infile);
  WDrawString(mainwrite,50, 33, wstring, 1);
sprintf(wstring,"Length:           %i samples",length1);
  WDrawString(mainwrite,50, 66, wstring,1);
sprintf(wstring,"Sample rate:   %6.2f Hz", samplerate);
  WDrawString(mainwrite,50, 100,wstring,1);
sprintf(wstring,"Channels:        %i",channels);
  WDrawString(mainwrite,50, 133,wstring,1);
sprintf(wstring,"Output file:      %s",outfile);
  WDrawString(mainwrite,50, 166,wstring,1);

InitButtons(mainwindow, "b,recalculate,r,b,frequency page,f,b,amplitude page,\
a,b,filter page,i,b,exit,x",200);

return;
}

/*-----------------------------------------------------------------*/
/*          subroutines for calculating new wave file             */
/*-----------------------------------------------------------------*/

/*------                                          ---------------*/
/*      recalculate basically does all the work...                */
/*------                                          ---------------*/


void recalculate()
```

```
{
float pos,time;
int i,j,l,m,n;
int nsamp;
int blocksize;
float x, xtemp[9];
int   xnr, xcount=0;
float y[2050], y2a[2050];
float sampbuff[4100],fftbuff[4100],backbuff[4100];
float wtemp[8300];
float yout[8200];
float sampout[16400];
float cutoff;
char wstring[40];


/*----                                                 -----------*/
/*                 1st task: frequency                            */
/*----                                                 -----------*/

/*      important: always proclaim that you ARE working  ;-)      */
  WDrawString(mainwrite,50, 250,"Working...                  ",1);

inptr= AFopenRead(infile, &length1, &channels, &samplerate, NULL);
tmpptr = AFopenWrite(tempfile, 261, channels, samplerate, NULL);

for(j=0;j<channels;j++)
  {xtemp[j]=0.0;}

for(i = 0; 2048*channels*i<length1; i=i+1){

   nsamp = AFreadData(inptr,2048*channels*i,sampbuff, 2048*channels);
   blocksize = channels*2048;

       for(j=0; j<channels; j++){
       for(l=0; l<2048;l++){
 y[l]=sampbuff[l*channels + j];
         }/* for(l) */
     x=xtemp[j];
```

```
    xnr=0;

     while(x<=2048.0){
      yout[xnr]= lint(y,x);
      x = x + timestretch((float)(xcount+xnr)/(samplerate*(float)channels),
               ((float)(i*blocksize)+x*(float)channels)/((float)(length1)));
      sampout[xnr*channels + j] = yout[xnr];
      xnr= xnr+1;
     }


      xtemp[j] = x - 2048.0;
      xcount= xcount + xnr;
    }/* for(j) */
  AFwriteData(tmpptr, sampout, channels*xnr);
  } /*for(i)*/

AFclose(tmpptr);
AFclose(inptr);
/*----                                              -----------------*/
/*          2nd task: amplitude                                    */
/*----                                              -----------------*/

tmpptr=AFopenRead(tempfile,&length2,&channels,&samplerate,NULL);
outptr=AFopenWrite(tem2file,261, channels,samplerate,NULL);

for(i=0; 2048*channels*i<length2; i++)
 {

  nsamp = AFreadData(tmpptr, 2048*channels*i, sampbuff, 2048*channels);
  blocksize = channels*2048;

    for(j=0; j<channels;j++){
      for(l=0; l<2048; l++){
        y[l]=sampbuff[l*channels + j];
        pos=(float)(i*blocksize +l*channels)/((float)(length2));
        time=(float)(i*blocksize +l*channels)/(samplerate*(float)channels);
        y[l]=(env(alvl0,alvl1,alvl2,alvl3,alvl4,atime1,atime2,atime3,atime4,
              pos)+
```

```
            lfo(alfoform, alfooffset,pos,alfoamp, 20.0*alforate, time))
            *y[l];


       sampout[l*channels + j]=y[l];
     }
    }
    AFwriteData(outptr, sampout, blocksize);
}
AFclose(tmpptr);
AFclose(outptr);

/*----                                                     ----------*/
/*       3rd task: filter                                           */
/*----                                                     ----------*/

tmpptr=AFopenRead(tem2file,&length2,&channels,&samplerate,NULL);
outptr=AFopenWrite(outfile,261, channels,samplerate,NULL);
blocksize = 4096;


/*   initialize temporary buffer for fft  */
srffti_(&blocksize, wtemp);


/*   read samples    */

for(i=0; 2048*channels*i<length2; i++)
 {
  nsamp = AFreadData(tmpptr, 2048*channels*i, sampbuff, 2048*channels);

  for(j=1; j<=channels;j++){
     for(l=0; l<2048; l++){
       fftbuff[l]=sampbuff[(l)*channels + (j-1)];
       fftbuff[l+2048]= 0.0;  /* padding with zeros  */
      }/* for (l) */

     srfftf_(&blocksize, fftbuff, wtemp);
```

```
     }
     for (m = 0; m< 8; m++){
       pos = (float)(i*channels*2048+m*256*channels)/(float)length2;
       cutoff=env(ilvl0,ilvl1,ilvl2,ilvl3,ilvl4,
                   itime1,itime2,itime3,itime4,pos);

       for (n = 0; n<=4096; n++){
         backbuff[n]=resfct((float)n/4096.0,cutoff,resonance)*fftbuff[n];
         }
       srfftb_(&blocksize, backbuff,wtemp);
       for(n = 0;n<= 255;n++)
         { sampout[n*channels + m*256*channels] =
     backbuff[n + m*256]/8096.0;}
     }/*for (m)  */
  } /* for (j)  */
  AFwriteData(outptr,sampout,2048*channels);
 } /* for (i) */
AFclose(tmpptr);
AFclose(outptr);

  WDrawString(mainwrite,50, 250,"Done recalculating   ",1);

}

/*--------------------------------------------------------------*/
/*          timestretch                                         */

float timestretch(float time, float pos)
{
return  0.5*fexp(
        (env(flvl0,flvl1,flvl2,flvl3,flvl4,ftime1,ftime2,ftime3,ftime4,pos) +
         lfo(flfoform, flfooffset, pos, flfoamp,20.0*flforate, time)/3.0)
         *2*flog(2));}



/*--------------------------------------------------------------*/
/*           linear interpolation between sample points         */
```

```
float lint(float y[],float x)
{
int xint= (int)x;
float xround = (float)((int)x);

if(x<0)
 return ((2-x)*y[1]+(x-1)*y[2]);
else if(x>2047)
 { return (2047-x)*y[2046]+ (x-2046)*y[2047];}
else
 {return (xround+1.0-x)*y[xint] + (x - xround)*y[xint+1];}
}

/*-----------------------------------------------------------------*/
/*           resonance function                                    */

float resfct(float f,float f0, float resonance)
{
  float r;
  float eps = 1e-8;
  float damp = 0.95; /* prevents resonance catastrophe */
  r = 2*f0*f0*(1 - damp*resonance);

    return (f0*f0 +eps)/(sqrt(pow((f*f -f0*f0),2)+ r*f*f)+eps);
  /* eps prevents numerical divergence */
}

/*-----------------------------------------------------------------*/
/*            envelope generator                                   */


float env(float lvl0, float lvl1,float lvl2, float lvl3, float lvl4,
          float time1, float time2, float time3, float time4,
          float time)

  {


if(time <= time1)
```

```
  {if (time1==0.0) return lvl1;
  else  return lvl0 + (time/time1)*(lvl1-lvl0);}

else if(time <=time2)
          return lvl1 - (lvl1-lvl2)*(time - time1)/(time2 - time1);

else if(time <=time3)
          return lvl2 - (lvl2-lvl3)*(time - time2)/(time3 - time2);

else if(time <= time4)
          return lvl3 - (lvl3-lvl4)*(time - time3)/(time4 - time3);

else
          return lvl4;
  }

/*-----------------------------------------------------------------*/
/*          low frequency oscillator                               */


float lfo(int form, float offset,float pos, float amp, float freq, float time)
{
float pi = 3.1415927;
float modulo = (freq*time) -(float)((int)(freq*time));
float stor;

if(form == 1){

stor=amp*sin(2*pi*freq*time);}


else if(form ==2){

if(modulo<=0.5) stor = -amp;
else stor= amp;}

else if(form ==3)
{
stor= 2*amp*(modulo-0.5);}
```

```
else if(form ==4){
if(modulo<0.5) stor = amp*(1 + 4*modulo);
else  stor = amp*(3 - 4*modulo);
}

if((pos+0.1)<offset) return 0;

else if (pos<offset) return (1-10.0*(offset-pos))*stor;

else return stor;}




/*--------------------------------------------------------------*/
/*         subroutines for editing frequency parameters          */
/*--------------------------------------------------------------*/

/*   getting parameters for frequency page  */

void freqpage()
{
 World envelope;
 XEvent freqevent;

 int leave = 0;
 int temp = 1;
 float tempfloat=0;
 char  fmenu;

 feg = CreateWorld(freqwindow,
          5, 5, 350, 350, 0, 1,1 , 0, 0, 0);

 fvalues = CreateWorld(freqwindow,
          365, 0, 65, 420, 0,0,1,12,0, 0);

 freqwrite();
```

```
InitButtons(freqwindow, "b, Level 1,a,b,Level 2,b,b,Level 3,c,b,Level 4,d,b,Time 1,e,b,Time 2,f,b,Time 3,g,b,Time 4,h,b, Lfo Form,i,b, Lfo Rate,j,b, Lfo Offset,k,b, Lfo Amplitude,l,b,Decrease(-),-,b, Increase(+),+,b,return,x", 150);

ShowWindow(freqwindow);
while(leave == 0)
  {
  if (GetEvent(&mainmenu,1)){

    switch(mainmenu.type) {
    case Expose:
       freqwrite();
       InitButtons(freqwindow, "b, Level 1,a,b,Level 2,b,b,Level 3,c,b,Level 4,d,b,Time 1,e,b,Time 2,f,b,Time 3,g,b,Time 4,h,b, Lfo Form,i,b, Lfo Rate,j,b, Lfo Offset,k,b, Lfo Amplitude,l,b,Decrease(-),-,b, Increase(+),+,b,return,x", 150);
        break;
    case KeyPress:
    switch(ExtractChar(mainmenu)){
      case 'a':
fmenu = 'a'; tempfloat = flvl1;break;
      case 'b':
fmenu = 'b';tempfloat = flvl2;break;
      case 'c':
  fmenu = 'c';tempfloat = flvl3;break;
     case 'd':
fmenu = 'd';tempfloat = flvl4;break;
      case 'e':
fmenu = 'e';tempfloat = ftime1;break;
      case 'f':
fmenu = 'f';tempfloat = ftime2;break;
      case 'g':
fmenu = 'g';tempfloat = ftime3;break;
      case 'h':
fmenu = 'h';tempfloat = ftime4;break;
      case 'i':
        flfoform = flfoform + 1;
        if (flfoform == 5) flfoform = 1;
        freqwrite();
        break;
      case 'j':
        fmenu = 'j'; tempfloat = flforate;break;
      case 'k':
        fmenu = 'k'; tempfloat = flfooffset; break;
```

```
       case 'l':
         fmenu = 'l'; tempfloat = flfoamp;break;

       case '-' :
           tempfloat = tempfloat -0.01;
           if(tempfloat<0.0) tempfloat = 0.0;
           frenew(fmenu, tempfloat);
           break;
       case '+':
           tempfloat = tempfloat + 0.01;
 if(tempfloat>1.0) tempfloat = 1.0;
         frenew(fmenu, tempfloat);  /* assign tempfloat to
                                       corresponding parameter */
         break;
       case 'x':
         leave =1;
 HideWindow(freqwindow);
         break;
        }; /*switch */
     }/*switch*/
   }/*if*/
 }/*while */


}

/*------------------------------------------------------------------*/
/*    frenew updates values of parameters, calls freqwrite           */


void frenew(char menu, float tmpfloat)
{

  switch(menu){
   case 'a':
      flvl1 = tmpfloat;break;
   case 'b':
      flvl2 = tmpfloat;break;
   case 'c':
```

```
      flvl3 = tmpfloat;break;
   case 'd':
      flvl4 = tmpfloat;break;
   case 'e':
      ftime1 = tmpfloat;
      if (ftime1> ftime2) ftime1= ftime2;
      break;
   case 'f':
      ftime2 = tmpfloat;
      if(ftime2<ftime1) ftime2=ftime1;
      if(ftime2>ftime3) ftime2=ftime3;
      break;
   case 'g':
      ftime3 = tmpfloat;
      if(ftime3<ftime2) ftime3=ftime2;
      if(ftime3>ftime4) ftime3=ftime4;
      break;
   case 'h':
      ftime4 = tmpfloat;
      if(ftime4<ftime3) ftime4=ftime3;
      break;
    case 'j':
      flforate = tmpfloat; break;
   case 'k':
      flfooffset = tmpfloat; break;
   case 'l':
      flfoamp = tmpfloat; break;

   }/*switch*/

   freqwrite();

}

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

/* freqwrite writes values of frequency parameters to screen   */

void freqwrite()
```

```
{
 char str[12];

 ClearArea(freqwindow,0,0, 355,355);
 drawenv(feg, flvl0, flvl1,flvl2, flvl3,flvl4, ftime1,ftime2,
        ftime3,ftime4);

 sprintf(str,"%4.2f", flvl1);
  WDrawString(fvalues,0.0, 0.6, str, 1);
 sprintf(str,"%4.2f", flvl2);
  WDrawString(fvalues,0.0, 1.6, str, 1);
 sprintf(str,"%4.2f", flvl3);
  WDrawString(fvalues,0.0, 2.6, str, 1);
 sprintf(str,"%4.2f", flvl4);
  WDrawString(fvalues,0.0, 3.6, str, 1);
 sprintf(str,"%4.2f", ftime1);
  WDrawString(fvalues,0.0, 4.6, str, 1);
 sprintf(str,"%4.2f", ftime2);
  WDrawString(fvalues,0.0, 5.6, str, 1);
 sprintf(str,"%4.2f", ftime3);
  WDrawString(fvalues,0.0, 6.6, str, 1);
 sprintf(str,"%4.2f", ftime4);
  WDrawString(fvalues,0.0, 7.6, str, 1);

 WDrawString(fvalues,0.0, 8.6, "                ",1);
    if(flfoform ==1)
 WDrawString(fvalues,0.0, 8.6, "Sine", 1);
    else if (flfoform ==2)
 WDrawString(fvalues,0.0, 8.6, "Square", 1);
    else if (flfoform ==3 )
 WDrawString(fvalues,0.0, 8.6, "Sawtooth", 1);
    else if (flfoform ==4)
 WDrawString(fvalues,0.0, 8.6, "Triangle", 1);
    else
 WDrawString(fvalues,0.0, 8.6, "undefined", 1);

 WDrawString(fvalues, 0.0, 9.6, "            ",1);
 sprintf(str,"%5.2f",20.0*flforate );
  WDrawString(fvalues,0.0, 9.6, str, 1);
```

```
 sprintf(str,"%5.2f",flfooffset );
  WDrawString(fvalues,0.0, 10.6, str, 1);
 sprintf(str,"%5.2f",flfoamp );
  WDrawString(fvalues,0.0, 11.6, str, 1);
}/* end of freqwrite */



/*------------------------------------------------------------------*/
/*          subroutines for amplitude parameters                    */
/*      (basically the same with other parameter names)             */
/*------------------------------------------------------------------*/


/*   getting parameters for amplitude page  */

void amppage()
{
 World envelope;


 XEvent ampevent;
 int leave = 0;
 int temp = 1;
 float tempfloat=0;
 char  amenu;


 aeg = CreateWorld(ampwindow,
          5, 5, 350, 350, 0, 1,1 , 0, 0, 0);

 avalues = CreateWorld(ampwindow,
          365, 0, 65, 420, 0,0,1,12,0, 0);

 awrite();

 InitButtons(ampwindow, "b, Level 1,a,b,Level 2,b,b,Level 3,c,b,Level 4,d,b,Time 1,e,b,Time 2,f,b,Time 3,g,b,Time 4,h,b, Lfo Form,i,b, Lfo Rate,j,b, Lfo Offset,k,b, Lfo Amplitude,l,b,Decrease(-),-,b, Increase(+),+,b,return,x", 150);

 ShowWindow(ampwindow);
```

```
while(leave == 0)
  {
  if (GetEvent(&mainmenu,1)){

    switch(mainmenu.type) {
    case Expose:
      awrite();
 InitButtons(ampwindow, "b, Level 1,a,b,Level 2,b,b,Level 3,c,b,Level 4,d,b,Time 1,e,b,Time 2,f,b,Time 3,g,b,Time 4,h,b, Lfo Form,i,b, Lfo Rate,j,b, Lfo Offset,k,b, Lfo Amplitude,l,b,Decrease(-),-,b, Increase(+),+,b,return,x", 150);
     break;
    case KeyPress:
    switch(ExtractChar(mainmenu)){
      case 'a':
amenu = 'a'; tempfloat = alvl1;break;
      case 'b':
amenu = 'b';tempfloat = alvl2;break;
      case 'c':
  amenu = 'c';tempfloat = alvl3;break;
     case 'd':
amenu = 'd';tempfloat = alvl4;break;
      case 'e':
amenu = 'e';tempfloat = atime1;break;
      case 'f':
amenu = 'f';tempfloat = atime2;break;
      case 'g':
amenu = 'g';tempfloat = atime3;break;
      case 'h':
amenu = 'h';tempfloat = atime4;break;
      case 'i':
        alfoform = alfoform + 1;
        if (alfoform == 5) alfoform = 1;
        awrite();
        break;
      case 'j':
        amenu = 'j'; tempfloat = alforate;break;
      case 'k':
        amenu = 'k'; tempfloat = alfooffset; break;
      case 'l':
        amenu = 'l'; tempfloat = alfoamp;break;
```

```
       case '-' :
           tempfloat = tempfloat -0.01;
           if(tempfloat<0.0) tempfloat = 0.0;
           arenew(amenu, tempfloat);
           break;
       case '+':
           tempfloat = tempfloat + 0.01;
 if(tempfloat>1.0) tempfloat = 1.0;
         arenew(amenu, tempfloat);  /* assign tempfloat to
                                       corresponding parameter */
         break;
       case 'x':
         leave =1;
         HideWindow(ampwindow);
         break;
        }; /*switch */
     }/*switch*/
   }/*if*/
 }/*while */
}

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

/*    arenew updates values of parameters, calls awrite to print     */
/*    them and draw envelope                                         */


void arenew(char menu, float tmpfloat)
{

  switch(menu){
   case 'a':
      alvl1 = tmpfloat;break;
   case 'b':
      alvl2 = tmpfloat;break;
   case 'c':
      alvl3 = tmpfloat;break;
   case 'd':
```

```
      alvl4 = tmpfloat;break;
   case 'e':
      atime1 = tmpfloat;
      if (atime1>atime2) atime1= atime2;
      break;
   case 'f':
      atime2 = tmpfloat;
      if(atime2<atime1) atime2=atime1;
      if(atime2>atime3) atime2=atime3;
      break;
   case 'g':
      atime3 = tmpfloat;
      if(atime3<atime2) atime3= atime2;
      if(atime3>atime4) atime3=atime4;
      break;
   case 'h':
      atime4 = tmpfloat;
      if(atime4<atime3) atime4= atime3;
      break;
    case 'j':
      alforate = tmpfloat; break;
   case 'k':
      alfooffset = tmpfloat; break;
   case 'l':
      alfoamp = tmpfloat; break;

   }/*switch*/

   awrite();
}

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

/*    awrite writes values of frequency parameters to screen   */
/*    and prints envelope                                      */

void awrite()
{
 char str[12];
```

```
ClearArea(ampwindow,0,0, 355,355);
drawenv(aeg, alvl0, alvl1,alvl2, alvl3,alvl4, atime1,atime2,
atime3,atime4);


 sprintf(str,"%4.2f", alvl1);
  WDrawString(avalues,0.0, 0.6, str, 1);
 sprintf(str,"%4.2f", alvl2);
  WDrawString(avalues,0.0, 1.6, str, 1);
 sprintf(str,"%4.2f", alvl3);
  WDrawString(avalues,0.0, 2.6, str, 1);
 sprintf(str,"%4.2f", alvl4);
  WDrawString(avalues,0.0, 3.6, str, 1);
 sprintf(str,"%4.2f", atime1);
  WDrawString(avalues,0.0, 4.6, str, 1);
 sprintf(str,"%4.2f", atime2);
  WDrawString(avalues,0.0, 5.6, str, 1);
 sprintf(str,"%4.2f", atime3);
  WDrawString(avalues,0.0, 6.6, str, 1);
 sprintf(str,"%4.2f", atime4);
  WDrawString(avalues,0.0, 7.6, str, 1);

 WDrawString(avalues,0.0, 8.6, "                ",1);
    if(alfoform ==1)
 WDrawString(avalues,0.0, 8.6, "Sine", 1);
    else if (alfoform ==2)
 WDrawString(avalues,0.0, 8.6, "Square", 1);
    else if (alfoform ==3 )
 WDrawString(avalues,0.0, 8.6, "Sawtooth", 1);
    else if (alfoform ==4)
 WDrawString(avalues,0.0, 8.6, "Triangle", 1);
    else
 WDrawString(avalues,0.0, 8.6, "undefined", 1);

 WDrawString(avalues, 0.0, 9.6, "            ",1);
 sprintf(str,"%5.2f",20.0*alforate );
  WDrawString(avalues,0.0, 9.6, str, 1);
```

```
 sprintf(str,"%5.2f",alfooffset );
  WDrawString(avalues,0.0, 10.6, str, 1);
 sprintf(str,"%5.2f",alfoamp );
  WDrawString(avalues,0.0, 11.6, str, 1);

}/* end of awrite */


/*---------------------------------------------------------------------*/
/*   ...and once more for the filter parameters                        */
/*              long live cut and paste                                */

/*   getting parameters for filter page  */

void filtpage()
{
 World envelope;


 XEvent filtevent;
 int leave = 0;
 int temp = 1;
 float tempfloat=0;
 char  imenu;


 ieg = CreateWorld(filtwindow,
          5, 5, 350, 350, 0, 1,1 , 0, 0, 0);

 ivalues = CreateWorld(filtwindow,
          365, 0, 65, 420, 0,0,1,12,0, 0);

 iwrite();

 InitButtons(filtwindow, "b, Level 1,a,b,Level 2,b,b,Level 3,c,b,Level 4,d,b,Time 1,e,b,Time 2,f,b,Time 3,g,b,Time 4,h,b,Resonance,r,b,Decrease(-),-,b, Increase(+),+,b,return,x", 150);

 ShowWindow(filtwindow);

 while(leave == 0)
```

```
  {
  if (GetEvent(&filtevent,1)){

    switch(filtevent.type) {
    case Expose:
      iwrite();
      InitButtons(filtwindow, "b, Level 1,a,b,Level 2,b,b,Level 3,c,b,Level 4,d,b,Time 1,e,b,Time 2,f,b,Time 3,g,b,Time 4,h,b,Resonance,r,b,Decrease(-),-,b, Increase(+),+,b,return,x", 150);
      break;
    case KeyPress:
    switch(ExtractChar(filtevent)){
      case 'a':
imenu = 'a'; tempfloat = ilvl1;break;
      case 'b':
imenu = 'b';tempfloat = ilvl2;break;
      case 'c':
  imenu = 'c';tempfloat = ilvl3;break;
      case 'd':
imenu = 'd';tempfloat = ilvl4;break;
      case 'e':
imenu = 'e';tempfloat = itime1;break;
      case 'f':
imenu = 'f';tempfloat = itime2;break;
      case 'g':
imenu = 'g';tempfloat = itime3;break;
      case 'h':
imenu = 'h';tempfloat = itime4;break;
      case 'r':
        imenu = 'r';tempfloat = resonance;
      case '-' :
          tempfloat = tempfloat -0.01;
          if(tempfloat<0.0) tempfloat = 0.0;
          irenew(imenu, tempfloat);
          break;
      case '+':
          tempfloat = tempfloat + 0.01;
if(tempfloat>1.0) tempfloat = 1.0;
        irenew(imenu, tempfloat);  /* assign tempfloat to
                                     corresponding parameter */
        break;
```

```
       case 'x':
         leave =1;
         HideWindow(filtwindow);
         break;
        }; /*switch */
     }/*switch*/
   }/*if*/
 }/*while */
}

/*--------------------------------------------------------------------*/
/*    irenew updates values of parameters, calls iwrite()            */


void irenew(char menu, float tmpfloat)
{

  switch(menu){
   case 'a':
      ilvl1 = tmpfloat;break;
   case 'b':
      ilvl2 = tmpfloat;break;
   case 'c':
      ilvl3 = tmpfloat;break;
   case 'd':
      ilvl4 = tmpfloat;break;
   case 'e':
      itime1 = tmpfloat;
      if (itime1>itime2) itime1= itime2;
      break;
   case 'f':
      itime2 = tmpfloat;
      if(itime2<itime1) itime2=itime1;
      if(itime2>itime3) itime2=itime3;
      break;
   case 'g':
      itime3 = tmpfloat;
      if(itime3<itime2) itime3= itime2;
      if(itime3>itime4) itime3=itime4;
```

```
      break;
   case 'h':
      itime4 = tmpfloat;
      if(itime4<itime3) itime4=atime3;
      break;
   case 'r':
      resonance = tmpfloat;break;


   }/*switch*/

   iwrite();

}


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

/*    iwrite writes values of filter parameters to screen   */


void iwrite()
{
 char str[12];

 ClearArea(filtwindow,0,0, 355,355);

 drawenv(ieg, ilvl0, ilvl1,ilvl2, ilvl3,ilvl4, itime1,itime2,
        itime3,itime4);


 sprintf(str,"%4.2f", ilvl1);
  WDrawString(ivalues,0.0, 0.6, str, 1);
 sprintf(str,"%4.2f", ilvl2);
  WDrawString(ivalues,0.0, 1.6, str, 1);
 sprintf(str,"%4.2f", ilvl3);
  WDrawString(ivalues,0.0, 2.6, str, 1);
 sprintf(str,"%4.2f", ilvl4);
  WDrawString(ivalues,0.0, 3.6, str, 1);
```

```
 sprintf(str,"%4.2f", itime1);
  WDrawString(ivalues,0.0, 4.6, str, 1);
 sprintf(str,"%4.2f", itime2);
  WDrawString(ivalues,0.0, 5.6, str, 1);
 sprintf(str,"%4.2f", itime3);
  WDrawString(ivalues,0.0, 6.6, str, 1);
 sprintf(str,"%4.2f", itime4);
  WDrawString(ivalues,0.0, 7.6, str, 1);
 sprintf(str,"%4.2f", resonance);
  WDrawString(ivalues,0.0, 8.6, str, 1);

}/* end of iwrite */


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

/*  drawenv - draws envelope with given parameters                */

void drawenv(World w,float lvl0, float lvl1, float lvl2, float lvl3,
float lvl4, float tm1, float tm2, float tm3, float tm4)
{
 WDrawRectangle(w, 0,0,1,1,1);

 WDrawLine(w,0, lvl0, tm1, lvl1,2);
 WDrawLine(w,tm1, lvl1, tm2, lvl2,2);
 WDrawLine(w,tm2, lvl2, tm3, lvl3,2);
 WDrawLine(w,tm3, lvl3, tm4, lvl4,2);
 WDrawLine(w,tm4,lvl4,1,lvl4,2);}
```