Musical Applications of Digital Synthesis and Processing Techniques

Realisation using *Csound* and the *Phase Vocoder*

Rajmil Fischman

April 1991

Preface 2006

Dear Reader

It is more than fifteen years since these pages were written, and, undoubtedly, many changes and advances have taken place than have had a profound influence in the development and use of Csound and the frequency domain processing tools (Chapter 11 – The Phase Vocoder) demonstrated in the text.

Furthermore, when the book was written, the only personal computer (apart from mainframes and minicomputers) in which CSound was implemented was the Atari ST.

However, although the some of the procedural indications have changed, most of the syntax of CSound and the Composers' Desktop Project (CDP) version of the CARL phase vocoder processes have remained the same. Therefore, except for the interface commands for running CSound, the orchestras and scores should work as they are given in the text, and the phase vocoder processes are still achievable using the latest version of the CDP.

Nevertheless, the following pointers might help you run CSound in a more contemporary environment and also find further information regarding the Phase Vocoder processes:

1. The command for running CSound given in chapter 1 (page 6 onwards) should be replaced by that running on an current computer. For instance, on a PC, typical commands look as follows:
    a. To run CSound in real-time, directly to the soundcard

        csound -W -o dac

    b. To make CSound create an audio file that can be auditioned later

        csound -W -o <output name>.wav

        where *<output name>* is the name of the audio file you wish to create.

2. Better still, it is worth using existing available public domain interfaces, such as the wonderful BLUE, by Steven Yi, which works in any platform and available for free download at http://www.csounds.com/stevenyi/blue/ .
3. Information on the CDP is found at http://www.composersdesktop.com/.
4. Finally, I have published a relatively more recent article, which develops the issues covered in chapter 11 to more depth and breath and contains a large number of additional examples:

   Fischman R, 1997. 'The Phase Vocoder: Theory and Practice' in Organised Sound **2**(2). Cambridge University Press, Cambridge, UK, pp 127-145.

Audio examples appear in the Organised Sound **2**(3) CD.

Last time I checked, the was available for free download in PDF format at http://mustech.robodreams.com/ceos/reading/rajmil.pdf

I hope that you find the texts useful

Rajmil Fischman
29/12/06

Notes

(1) *Csound* by Barry Vercoe. Massachussets Institute of Technology (MIT). Cambridge. Massachussets. USA.

(2) *Phase Vocoder* by Mark Dolson. Computer Audio Research Laboratory (CARL), Centre for Music Experiment, University of California, San Diego, USA.

Contents

Prefacio 2006

Querido Lector

Hace ya más de quince años desde que escribí estas páginas, y sin duda, mucho ha cambiado desde entonces. Estos cambios han afectado el desarrollo y uso de CSound y de los útiles para procesamiento en el dominio de la frecuencia que este texto demuestra (Capítulo 11 – El Phase Vocoder).

Es más, cuando el libro estaba siendo escrito, la única computadora personal (fuera de los mainframes y minicomputadoras) en el cual se había implementado CSound era la Atari ST.

No obstante, aunque algunas instrucciones para lanzar el programa hayan cambiado, la mayoría de la sintáctica de CSound y los procesos obtenibles con la versión Composers' Desktop Project (CDP) del phase vocoder de CARL se han mantenido intactos. Es por eso que las orquestas y partituras que aparecen en el texto funcionan aún, y los procesos del phase vocoder se pueden realizar usando las versiones más recientes del CDP.

De todos modos, espero que las sugerencias que aparecen a continuación te ayuden a procesar en CSound usando equipo moderno y te aporten más información sobre el phase vocoder:

5. El comando para lanzar a CSound que aparece en el capítulo 1 (a partir de la página 6) deberá ser sustituido con instrucciones adecuadas para computadores más recientes. Por ejemplo, en la PC, los comandos típicos serían:
    a. Para lanzar CSound en tiempo real, enviando los resultados directamente a la tarjeta de sonido

       csound -W -o dac

    b. Para crear un archivo sonoro con CSound:

       csound -W -o <nombre del salida>.wav

       donde *<nombre de salida>* es el nombre del archivo que se desee crear.

6. Sería mejor aún usar alguno de los programas gráficos que procesan con CSound, tal como el maravilloso BLUE, de Steven Yi, que funciona en cualquier plataforma y puede ser bajado gratuitamente de http://www.csounds.com/stevenyi/blue/ .
7. Puedes obtener información sobre el CDP en la página http://www.composersdesktop.com/.
8. Finalmente, quisiera informarte que he publicado un artículo que es relativamente más reciente. Éste ensancha y profundiza los conceptos discutidos en el capítulo 11, y provee ejemplos adicionales:

   Fischman R, 1997. 'The Phase Vocoder: Theory and Practice' in Organised Sound **2**(2). Cambridge University Press, Cambridge, UK, pp 127-145.

Los ejemplos de audio aparecen en el CD de Organised Sound **2**(3).

La última vez que busqué el artículo, encontré la siguiente página, que permite bajarlo gratuitamente en formato PDF: http://mustech.robodreams.com/ceos/reading/rajmil.pdf

Espero que este texto te sea provechoso.

Rajmil Fischman
29/12/06

Preface

It has been said more than once that the advent of computer technology and its application to music has given the electroacoustic composer command over a vast universe of sound that could only be the dream of his predecessors.

While, at least in theory, it is reasonable to assume the validity of this statement, it is also important to bear in mind that the same developments have given rise to new problems confronting anyone willing to take full advantage of the new developments. For instance, the psychoacoustic effects of sound perception within a musical context are immediately apparent and much more significant than when dealing with traditional instruments: A concept like timbre, which is reasonably defined within the orchestral palette, takes new significance as a time-varying, multidimensional entity, in which there is no one to one correspondence between the spectrum and timbral identity.

Another well known difficulty is that involved in achieving sounds that are 'alive', rather than of neutral 'electronic' quality, experienced by anyone that has used a synthesizer, regardless of whether it is implemented as a piece of hardware or in a software package.

In fact, a typical learning curve includes a threshold, below which the sounds that are produced are very similar to each other and usually of no great interest. A possible explanation is that while the various synthesis or processing techniques are reasonably simple, it is very difficult to find a correlation between these and the resulting sounds. The latter can only be discovered by experience, either one's own, or that appearing in literature.

At the time of writing, there is still very little literature sharing the results of this type of experience. It is true that there are publications which contain the principles of various techniques used to synthesize and process sounds, but few of these go a step beyond, showing what can be done with the techniques and - not less important - showing their audible results. It is the wish of the author to share his own limited experience in the hope that it contributes to make the learning process shorter and more pleasant and that it suggests directions for possible further experimentation.

For practical purposes, *Csound* (1) was chosen in order to illustrate the results. This is because of its flexibility, as well as for being available in several microcomputers, particularly within the Composers' Desktop Project (CDP) Atari based system. However, alas, flexibility comes at the expense of simplicity. Therefore, and in view of the fact that *Csound*'s original manual assumes

previous knowledge of similar computer music systems, the first part of this book consists of a stepwise approach towards the mastering of the basic tools offered by this package: it can be skipped by the initiated. It should be clear, though, that this is not a substitute for the manual and there are no pretensions to exhaustively cover all the possibilities offered by *Csound*.

The second part deals with the principles of linear techniques: *additive* and *subtractive synthesis*, and the third with non-linear methods: *amplitude modulation*, *waveshaping* and *frequency modulation*. In the fourth part, devoted to processing, the principles and some applications of the *Phase Vocoder* (2) are discussed.

Finally, an appendix containing time-varying reverberation instruments for both mono and stereo soundfiles is presented.

All the examples were recorded to a cassette tape hereby included, and the code for these is available in floppy disk.

The author would like to express his deepest gratitude to those that helped directly and indirectly in this research. Special thanks to Richard Orton, who first introduced him to MUSIC 11, and in general to computer music. Also thanks to Trevor Wishart, Tom Endrich and Andrew Bentley for all he was able to learn from their experience and for being responsible, together with Richard Orton, for the Composers' Desktop Project. To John Paynter, who, in spite of not being directly involved in electroacoustic music, provided, as a musician and composer, invaluable input from the aesthetic point of view which reverted back into the use of computer music techniques. Thanks to David Malham and Martin Atkins for being instrumental in the realisation of the CDP. Finally, but not least, special thanks to his colleagues at Keele University Music Department for their encouragement in a stimulating environment and for their constructive feedback.

Rajmil Fischman
Keele, April 1991.

PART I

Introduction to *Csound*

1. Fundamental Concepts

Csound is a high level computer synthesis and processing language that serves the musician as a tool to create and manipulate samples representing sounds, by means of a variety of techniques. It is implemented in the CDP system and the results can be stored in magnetic media (usually a hard disk or DAT tape). Therefore, the sampled sounds can be played through loudspeakers. It is also possible to display them graphically as waveforms.

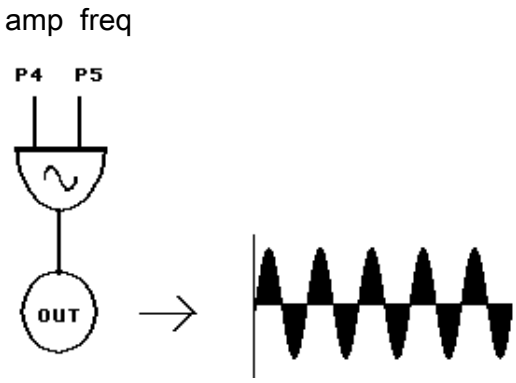
Sampled sound is stored in a computer as a file, more specifically as a *soundfile*. At this point, it is worth noticing the difference between MIDI (Music Instrument Digital Interface) information and soundfile information. The former contains messages that activate synthesizers and other hardware, it does not contain actual sounds. The latter, on the other hand, consists of actual sampled sound.

1.1 Orchestra and Score

The basic concept behind the implementation of *Csound* resembles the way sounds were initially produced in analogue synthesizers. Modules known as *oscillators* were patched together to create more complex units. These oscillators produce basic types of waveforms like sines, square and triangular waves, sawtooth, white noise, etc.

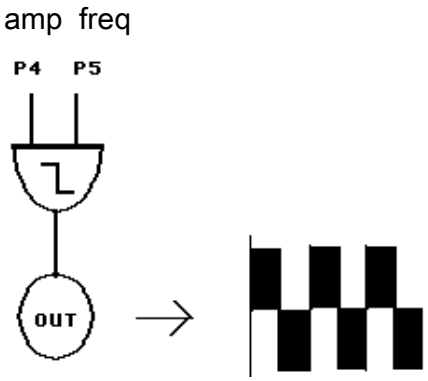
In order to illustrate this principle, take for example a pure sine tone : a sine oscillator can be connected directly to the output. Both the frequency and amplitude of the sine wave depend on the data fed to the oscillator by means of control knobs. This data can be variable (it can assume several values according to the position of the knobs). Diagram 1 shows a graphical representation of this setting. **P3** represents the variable amplitude and **P4** the variable frequency.

Diagram 1 Pure sine tone oscillator



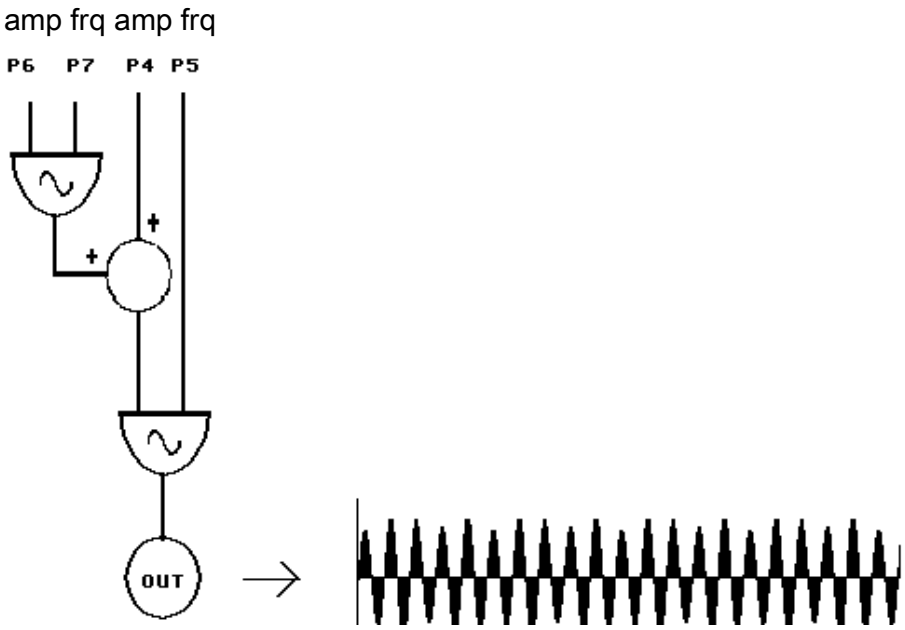
If, instead of a sine wave, a square wave is needed, a different oscillator can be used. The setting, though, would remain similar, as shown in diagram 2.

Diagram 2 Square wave oscillator



If some tremolo - (small sub-audio amplitude fluctuation) - is desired so that the sound is more realistic, then, instead of controlling the amplitude only through the knob; another oscillator with small amplitude and slower frequency could be added. This can be seen in diagram 3. **P6** and **P7** represent the amplitude and frequency of the second oscillator. The amplitude of the first oscillator fluctuates between **P4** - **P6** and **P4** + **P6**. The rate at which the fluctuation repeats itself is **P7**.

Diagram 3 Tremolo



All the previous examples have two common features:

1. A constant configuration: the schematic connections are always the same.

2. A set of variables: amplitude and frequency values could be changed.

A further analogy can be drawn to a musician playing an instrument. In this case, the constant configuration consists of the morphology of instrument and the musician's own anatomy. When required to play, the performer is given a score which consists of a chronological list of musical events describing them in terms of specific pitches to play, durations, dynamics, etc. All these are values given to variables that determine the actual sounds that are consequently produced with the existing configuration.

Csound's approach is based, at least in principle, on the concept of a more or less constant configuration of *instruments*, called the *orchestra*, and a chronological list of events, called the *score*. Each event invokes an instrument and specifies a set of values given to its corresponding variables.

In order to make it possible to use different scores with the same orchestra the *orchestra* and the *score* are stored in the computer as separate text (ASCII) files. This resembles even more the situation of a musician who can play different pieces on his instrument.

It should be pointed out that the orchestra-score paradigm is not always the best approach to sonic composition. For example, sometimes it is desirable to have a so called 'instrument' that itself creates a series of events. Furthermore, there is a preconception regarding the traditional approach to timbre, by which a given instrument will always have a recognisable timbral identity, which, of course, is not true of a *Csound* instrument. This will become obvious in later examples.

1.2 Creating a Sound

As mentioned above, the orchestra and the score are stored as text files in a computer. These are created using a text editor and subsequently, have to be translated to computer language, that is, they have to be *compiled*. In this case the equivalent of the compiler is the actual program called *Csound*.

However, there is a difference between *Csound* and a real compiler like those used in *C* or *PASCAL*. Csound does not only translate the score and orchestra into computer language commands, but also causes the computer to carry out these. Therefore, in the end, the computer directly produces a soundfile when *Csound* code is compiled.

Before entering into details on how to compile, there is one more issue that has to be clarified : *Csound* allows the user to specify events in any chronological order within a section. This means that it assumes the responsibility of ordering events chronologically. For this reason, there is a program called CSORT.PRG that sorts the events creating a new version of the score in a file called SCORE.SRT . It is this file, and not the original score that is used in conjunction with the orchestra when compiling.

In order to show how a sound is produced once the orchestra and score are created, it will be assumed that these exist and are respectively called CS1.ORC and CS1.SC and that the soundfile produced after running *Csound* is called CS1.OUT .

There are two steps to be followed in order to create the soundfile:

1. Sort the score chronologically.

2. Compile using a given orchestra.

The following commands will do exactly that

csort cs1.sc

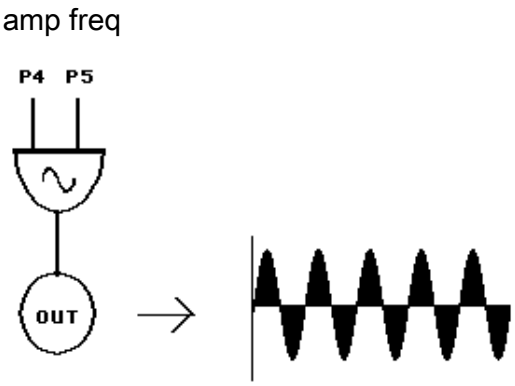
csound -Ncs1.out cs1.orc

The first command is quite obvious. The second command contains the *flag* -N, which tells the compiler that the following letters represent the name of the soundfile to be created. If the name of the output is not specified, the default name CSOUND.OUT will be given to the soundfile. The name of the orchestra is then specified. It is not necessary to specify the name of score because *Csound* will always use the sorted version contained in the file SCORE.SRT .

An example orchestra and score are given below, that can be compiled in order to create a soundfile. The meaning of their contents will become clearer in the following chapters.

The orchestra is called CS1.ORC and represents an instrument with the configuration given in diagram 4.

Diagram 4 CS1.ORC




```
; This is a comment, Csound will ignore it
; CS1.ORC

sr = 44100
kr =  4410
ksmps = 10
nchnls = 1

instr 1

a1   oscil   p4, p5, 1; oscillator
        out     a1; output

endin
```

The score file is called CS1.SC. It instructs this instrument to produce a sine wave of frequency 440 hz.

```
; This is a comment, Csound will ignore it
; CS1.SC
```

```
f1 0 512 10 1

t 0 60

i1 0 2 32000 440

e
```

Csound tape example 1 contains the admittedly not very exciting 440 hz sine wave.

1.3 Some Shortcuts

It is quite tedious to repeatedly type two commands with as much wording as CSORT and CSOUND need. In fact, if *Csound* is used from a command line environment like the COMMAND.TOS program, a *batch-file* can be written. A batch-file is a text file that contains a list of commands. When the computer is told to run a batch-file, it reads the commands one by one and performs them as if they were successively typed in via the keyboard by the user.

For example a batch-file called PERF.BAT can be written that gets the name of the orchestra and the name of the score and creates a file called CSOUND.OUT:

```
csort %1

csound %2
```

%1 and %2 mean respectively the first and second name typed after the name of the batch-file.

In our previous example the line

perf cs1.sc cs1.orc

will sort the score and create the soundfile called CSOUND.OUT .

Another possibility suitable to the case in which the orchestra and the score have the same name (like in the previous case: CS1), is to create the following batch-file

csort %1.sc

csound -N%1.out %1.orc

which will produce a soundfile with the extension .OUT .

2. Syntax of the Orchestra

As with any other computer language, there is a set of syntactic rules that has to be observed when creating a *Csound* orchestra and score.

A look at CS1.ORC in the previous chapter gives an idea of how an orchestra file should look. In the first place, it is possible to include *comments* giving explanatory remarks or other information which are ignored by *Csound* but, at the same time, very useful to the user. Comments are always preceded by semicolons **';'** . Anything contained between a semicolon and the end of a line is ignored.

The rest of the orchestra consists of different parts which are described in the following sections.

2.1 The Header

The first four lines that are not comments constitute the *header*. This is used to define quantities that are needed in order to produce sampled sound. These are:

2.1.1 Sampling Rate (sr)

The first line defines the *sampling rate*. In the case of CS1.ORC, it is 44100 samples/second.

2.1.2 Control Rate (kr)

Lets consider the oscillator producing a tremolo effect discussed in the previous chapter (see diagram 3). A reasonably fast tremolo would have a frequency of about 15 hz. Therefore, sampling this oscillator at 44100 hz can be unnecessarily wasteful with regard to computational power and time. It is for this reason that an alternative slower sampling rate called *control rate* is defined. Its name stems from the fact that it is not used for oscillators that produce sound but for modules that control other oscillators.

Another common example of control rate use can be found in the generation of amplitude envelopes.

2.1.3 Samples Skipped (ksmps)

Using the slower control rate is equivalent to skipping samples at the actual sampling rate. *ksmps* indicates how many samples are skipped. Therefore

$$ksmps = \frac{sr}{kr}$$

NOTE: The sampling rate has to be a multiple of the control rate.

2.1.4 Number of channels (nchnls)

This states how many audio channels are being used. There can be 1 channel (mono), 2 (stereo) and 4 (quad).

2.2 Instrument Definition

Next after the header is the statement *instr 1* . This tells the computer that an instrument will be defined by the subsequent lines. In this case it is instrument number 1. A number has to be given because there may be more than one instrument in the orchestra.

Then, the statements defining the instrument follow. Statements contain variables, constants or arithmetic expressions using variables and constants. In the case of CS1.ORC, there are two statements. As will be seen below, the first one defines the oscillator and the second the output.

Finally, *endin* is used to signal that this is the end of the definition of instrument 1.

2.3 General Form of the Orchestra File

It is now possible to define the general form of an orchestra file defining **n** instruments.

Header (sr, kr, ksmps, nchnls, etc.)

instr 1 statements ...
endin

instr 2 statements ...
endin
.
.
instr n statements ...
endin

2.4 OSCIL and OUT

So far no attempt has been made to explain the statements defining instrument 1. In fact, both statements are perhaps the most popular and useful in *Csound*. We will now proceed to describe them.

2.4.1 OSCIL

This represents the simplest oscillator. The basic waveform can be a sine, a square wave, triangular, sawtooth and virtually anything else.

Its general form is

ar oscil amp, freq, func (, phase)

where

ar is used to name the output of the oscillator.

amp is the amplitude. It can be a variable whose value is given in the score. The

maximum value the amplitude can attain is 32,767 (2^{16}- 1) since soundfiles are currently sampled at 16 bit resolution.

freq — is the frequency in cycles per second (CPS or hz). It can be a variable whose value is given in the score.

func — refers to a function defining the basic waveform (sine, square, etc.). The function is defined in the score.

phase — is the phase of the waveform. It is in brackets because it does not have to be specified, in which case it is assumed to have a default value of 0.

In CS1.ORC the output is called **a1**. The amplitude is given by the variable **p4**. This is a special kind of variable. A **p** followed by a number tells the computer that this is a *parameter* that will be given in the score. **p4** means that the fourth parameter given in any line of the score referring to this specific instrument (*instr 1*) will determine the amplitude.

Similarly, the frequency is given by **p5**. Again, this means that the fifth parameter in any line referring to instrument 1 determines the frequency.

Finally, the waveform is 1. This tells *Csound* that the score will contain a definition of the shape of one cycle of the oscillator identified by the number 1. The cycle so defined will be repeated by the oscillator to produce a continuous waveform.

2.4.2 OUT

OUT is used in order to produce an actual output. In other words, it tells the computer to create a soundfile.

The general form is

out *signal*

where

signal is the actual waveform that is going to be contained in the soundfile. It is usually an output or an arithmetic combination of outputs of one or more oscillators

In CS1.ORC, the statement

out a1

means that the soundfile created contains **a1**: the output of the oscillator.

Any combination of outputs is possible. For example, if an instrument has three oscillators with outputs represented by a1, a2 and a3 then the statement

out a1 + a2/2 + a3*1.5

effectively mixes the three with different gains.

3. Syntax of the Score

It is now possible to proceed to see how the score works. Again, a look at the example CS1.SC in the first chapter, will prove helpful.

Comments can also be written into the score starting with a semicolon **;**'. It is also permissible to use the letter **c** instead.

Each line in the score starts with a letter. The possible letters are: **a**, **c**, **e**, **f**, **i**, **s**, and **t**. They are followed by numbers which are the actual data. A letter tells *Csound* what type of information the data refers to. For example, the letter **f** means that the information in that line refers to the basic waveform of an oscillator. the letter **t** refers to *tempo* , **i** invokes an instrument, and so on.

3.1 Function Statements - GEN routines

The first line appearing after the comments in CS1.SC is a statement starting with the letter **f**. It is called a *function statement* or *f-statement*, because it provides the computer with a mathematical function that describes the shape of one cycle of the waveform that will be used by an oscillator in an instrument.

Once the computer gets the information given by the f-statement it produces a sampled version of the cycle of that waveform called the *function table*, because it is used as a basic table from which actual waveforms can be built by interpolation in order to get the right frequency, and then by repetition of the cycle in order to produce a periodic signal.

Before we look at the actual information given in an f-statement, a new comparison with the analogue synthesizer may prove helpful. In chapter 1, it was seen that the same configuration used in diagram 1 with a sine generator could be used with a square generator in diagram 2, thus producing a different waveform. Therefore, the basic cycle of a waveform depends on the signal generator used by the oscillator.

In a similar fashion, different signal generators available in *Csound* can produce the most diverse signals. These generators, known as *GEN routines*, are each called by using an identification number. For example, GEN10 is used to produce sine waves and

combinations of a fundamental and its harmonics, GEN9 is used to produce a combination of non-harmonic and/or out of phase sine waves, GEN7 is used to produce waveforms that can be built of straight lines like square and triangular waves, and so on.

The actual result depends on the data fed into the GEN routines, and this is exactly what an f-statement provides.

The general form of an f-statement is

f number time size GEN-number data

number — is a number given to the specific function table resulting from this statement, so it can be addressed by the orchestra. In the example, the third parameter of the oscillator defined in the orchestra is 1. This means that it will use the waveform cycle defined by **f1**. Using a negative number cancels a previous definition of a function.

time — indicates at what time (in beats) is the signal generator first going to be used. If the function number is negative, it indicates a time at which this function definition will be cancelled.

size — indicates the number of samples used to produce the table. Obviously if more samples are used, the shape of the cycle will be defined more accurately. On the other hand, this requires more computing time.

The size has to be a power of 2 (2, 4, 8, 16, 32, 64, etc.). A reasonably practical value is 512.

GEN-number — indicates the specific GEN routine number used.

data — is the specific data required by that GEN routine.
For example, GEN7 and GEN10 need different parameters: while the first of them requires break points that are going to be joined by straight lines to produce the waveform cycle, the latter requires the number of harmonic

and its relative amplitude.

In the CS1.SC, the statement

f1 0 512 1 0 1

tells *Csound* that the table containing the basic cycle of the waveform is table 1. It is used at time = 0 (the beginning of the score), it is defined by 512 samples and it uses GEN10.

The data required by GEN10 is the relative values of the harmonics. We see that only the first harmonic or fundamental is used. Thus, the waveform will be a sine wave.

3.2 **Tempo Statement**

Next in the CS1.SC comes a line starting with the letter **t**: a *tempo statement* or *t-statement*.

Time is specified in beats, like in instrumental music, therefore the tempo, or *number of beats per minute* has to be defined.

Its general form is

t beat# speed beat# speed .....

beat# is the beat number at which the tempo is specified.

speed is the number of beats per minute (metronome mark).

In short, a tempo statement contains a series of beat numbers and speeds. The computer interpolates the speeds in between the beats, therefore producing *accelerando* or *ritardando* effects.

In CS1.SC, the tempo is constant and specified once at the beginning as 60 MM. Therefore every beat lasts exactly a second and in this case, counting beats is the same as counting seconds.

If there is no tempo statement in the score, the default value of 60 MM is used automatically.

3.3 Instrument Statements - Parameter Fields

The next line starts with the letter **i**. This is an instrument statement or *i-statement*, because it defines an event for a specific instrument in the orchestra. The data obviously varies according to the configuration of the instrument.

There are some parameters pertinent to a particular instrument that have no meaning for others. This also happens in traditional music: it is quite improbable to find an indication like *sul tasto* in a horn part! On the other hand, all events have to be given a starting time and a duration. Therefore, it is useful to specify all the indispensable data first: instrument number, start time and duration, and then allow for specific parameters to be given for each particular instrument.

This leads to the general form of an i-statement

i instr-number start-time duration other parameters...

where

instr-number — is the identification number of the instrument as defined in the orchestra. In the example, it is instrument 1 (**i1**).

start-time — is the time in beats at which the event starts. In the example, it is 0 which means that the event is produced at the very beginning.

duration — is the duration in beats of the event. Sometimes an instrument in the orchestra uses the duration as a parameter. For example, the speed of a tremolo can depend on how long is the sound. It may be remembered that in order to use data from the score, special variables that use the letter **p** and a number are used. In this case, since the duration is the third parameter, the variable representing it in any instrument is **p3**.

other parameters — These are the corresponding values given to variables represented by p4, p5, p6, etc. in the orchestra. This purely depends on the particular configuration of the instrument. But it is common practice to assign p4 to the amplitude, and p5 to the fundamental frequency or to the pitch if any of those is defined in the instrument.

In CS1.SC there is only one instrument line

i1 0 2 3 2000 440

It addresses instrument 1 (**i1**), which should produce an event at time = 0 (second parameter), with a duration of 2 beats (third parameter), an amplitude of 32000 (fourth parameter, corresponding to **p4** in the orchestra) and a frequency of 440 hz (fifth parameter, corresponding to **p5**).

3.4 End of Score

Finally, the last line contains the single letter **e** which stands for: *end of score*.

3.5 Sections

When scores became too long, it is useful to divide them into sections.

The advantage of this is that time is reset at the beginning of each section. That is, time starts at 0. Functions can also be redefined.

In order to separate sections, the letter **s** is used.

3.6 General Form of the Score File

It is now possible to define the general form of a score file.

f-statements

t-statement

i-statements

s

f-statements

t-statement

i-statements

s

.
.
.

e

3.7 **Example - Use of GEN10**

The following example uses the same orchestra as CS1.ORC. This time, though, four sine waves will be produced.

261.625 hz	corresponding to middle C
329.627 hz	corresponding to E above
391.995 hz	corresponding to G above
466.163 hz	corresponding to Bb above.

C will sound at the beginning, then E will be added, then G and finally Bb. In order to do so, GEN10 is used to produce a sine wave. Therefore, the statement defining **f1** in CS1.SC can still be used.

The score looks as follows

```
f1 0 512 10 1

;              p3      p4      p5
;instr  start  dur     amp     freq
i1      0      5       8000    261.625
i1      1      4       .       329.627
i1      2      3       .       391.995
i1      3      2       .       466.163
e
```

A new feature of *Csound* can be noticed: If a variable repeats itself, a dot '.' can be used instead to avoid typing long numbers. In this case, the amplitude is the same for all the notes, so instead of the value corresponding to p4, a dot can be used in the second, third and fourth lines.

It is important to bear in mind that the amplitudes are added, so that if the notes are to be heard together, the sum of their amplitudes can not be greater than 32,767 which is the maximum amplitude value that can be represented with 16 bits, otherwise the sound will be distorted. If this happens, *Csound* will issue a message while compiling.

Diagram 5 Waveform produced by the f-statement: f1 0 512 10 1

GEN10 can also be used to produce more complex waveforms. The following score uses the same fundamentals for its waveforms, but the generating function is defined differently.

```
f1 0 512 10 1 0 .5 0 .7 0 .9 0 2

;                 p3    p4       p5
;instr  start  dur   amp     freq
i1      0       5     8000   261.625
i1      1       4     .        329.627
i1      2       3     .        391.995
i1      3       2     .        466.163
e
```

The meaning of the definition of f1 is

The fundamental has a relative amplitude of 1

The second harmonic has a relative amplitude of 0

The third harmonic has a relative amplitude of .5

The fourth harmonic has a relative amplitude of 0

The fifth harmonic has a relative amplitude of .7

The sixth harmonic has a relative amplitude of 0

The seventh harmonic has a relative amplitude of .9

The eight harmonic has a relative amplitude of 0

The ninth harmonic has a relative amplitude of 2

In other words, this produces an instrument with odd harmonics.

Diagram 6 Waveform produced by the f-statement :

f1 0 512 10 1 0 .5 0 .7 0 .9 0 2

Finally, another example score

```
f1 0 512 10 1 .5 .7 .9 .2 .87 .76

;                 p3      p4      p5
;instr  start   dur      amp     freq

i1      0       5        8000    261.625
i1      1       4         .      329.627
i1      2       3         .      391.995
i1      3       2         .       466.163

e
```

This time f1 defines an oscillator with 7 harmonics.

Diagram 7 Waveform defined by the f-statement

f1 0 512 10 1 .5 .7 .9 .2 .87 .76

Csound tape example 2 is the realization of the score CS2.SC in which the three previous scores are played one after the other. This is possible since it is allowed to redefine f1 after the end of a section. In order to have a gap of 1 beat, a dummy note of 0 amplitude is introduced.

It can be noticed that each note starts and ends with a click. The next chapter will deal with ways of avoiding this problem. Apologies for the time being.

Orchestra and score:

```
; CS2.ORC       Simple Oscillator
;               Uses table defined by f1

; NOTE :        Clicks will be heard since
;               there is no envelope.

sr = 44100
kr =  4410
ksmps = 10
nchnls = 1

instr 1

      a1      oscil   p4, p5, 1       ; oscillator
              out     a1              ; output
```

```
endin

; CS2.SC

f1 0 512 10 1

;             p3      p4      p5
;instr  start   dur     amp     freq

i1      0       5       8000    261.625
i1      1       4       .       329.627
i1      2       3       .       391.995
i1      3       2       .       466.163
i1      5       1       0       0

s

f1 0 512 10 1 0 .5 0 .7 0 .9 0 2

;             p3      p4      p5
;instr  start   dur     amp     freq

i1      0       5       8000    261.625
i1      1       4       .       329.627
i1      2       3       .       391.995
i1      3       2       .       466.163
i1      5       1       0       0

s

f1 0 512 10 1 .5 .7 .9 .2 .87 .76

;             p3      p4      p5
;instr  start   dur     amp     freq
```

```
i1    0     5    8000   261.625
i1    1     4    .      329.627
i1    2     3    .      391.995
i1    3     2    .      466.163

e
```

It now remains to be seen how to combine the different timbres in the same section. This requires two things :

1. More than one waveform table is needed. This is easily accomplished by defining the tables f2, f3, f4, etc. with different parameters for GEN10.

2. Our instrument has to be told which waveform to use. This suggests a variable that represents the table number. Thus, a new parameter, **p6**, can be introduced in *oscil* that gets the table number. When **p6** assumes the value 1, **f1** will be used. When **p6** is 2, **f2** will be used and so on.

This is applied to CS3.ORC and CS3.SC

```
; CS3.ORC       -       Simple Oscillator
;               Uses table defined by function number
;               indicated in p6.

; NOTE :        Clicks will be heard since
;               there is no envelope.

sr = 44100
kr =  4410
ksmps = 10
nchnls = 1
```

```
instr 2

        a1      oscil   p4, p5, p6      ; oscillator
                out     a1              ; output

endin

; CS3.SC

f1 0 512 10 1 0 .5 0 .7 0 .9 0 2
f2 0 512 10 1 .5 .7 .9 .2 .87 .76
f3 0 512 10 1 .7 0 .9 0 .87 0 .76
f4 0 512 10 1

t 0 120

;               p3      p4      p5          p6
;instr  start   dur     amp     freq        func no.
i2      0       10      8000    261.625     1
i2      2       8       .       329.627     2
i2      4       6       .       391.995     3
i2      6       4       .       466.163     4
e
```

Csound tape example 3 is the realization of the above orchestra and score.

4. Envelopes

The sounds produced so far have a problematic feature: when they end (and sometimes even when they start), a click is heard. Clicks like this are usually due to quick changes in the amplitude of a waveform. This usually happens when a sound is interrupted and the value of the current sample is much larger than zero, therefore producing a sudden drop to silence (zero amplitude), as shown in diagram 8.

Diagram 8 Appearance of Clicks

Click No Click

In order to rid the sounds of these clicks, a mechanism that produces smooth transitions between silence and actual sounds is needed. The best way of achieving this is by means of an envelope.

Envelopes also shape sounds, and in some cases even help determine their timbre. For example, a piano sound without its characteristic attack will not be recognizable. Furthermore, reversing a low piano note will actually produce something similar to a bowed string.

4.1 Rates

Before dealing in more detail with the production of envelopes it is worth remembering that the *control rate* provides the means of saving some computer calculation time by setting a lower sampling rate for signals that are expected to change relatively slow.

Envelopes are usually slow changing signals. It makes sense to assign the output of an envelope generator to a variable that is sampled at the control rate. In *Csound*, control variables always start with the letter **k**, as opposed to audio signals, sampled at the *audio*

rate, which start with the letter **a**. For example, *k1*, *kenv*, *k56*, represent control signals, while *a3*, *asig*, *aleft*, represent audio signals.

Furthermore, there are some cases in which it is necessary to calculate the value of a variable only once, at the beginning of an event. This type of variable is called *initialization variable*. Its value is calculated only once per event. This is referred by the manual as the *initialization rate*. In *Csound* initialization variables always start with the letter **i**. *i1* and *ifreq* are examples of initialization variables.

Finally, there are variables, like the values in the header of the orchestra (sampling rate, control rate, etc.) that are set only once in the whole 'piece'. These are called *setup rate* variables and are described in the manual. However, for practical purposes only the three rates need to be considered:

a-rate	For audio signals.	Variables beginning with **a**
k-rate	For control signals.	Variables beginning with **k**
i-rate	For values that are calculated once, at the beginning of an event.	Variables beginning with **i**

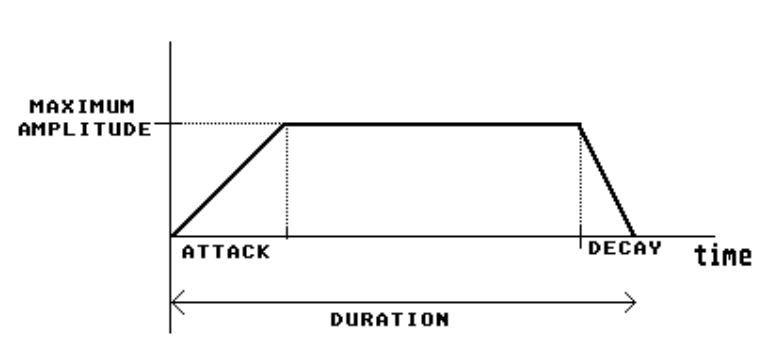
4.2 Envelope Generators

We will now proceed to discuss some of the most common ways of creating envelopes in *Csound*.

4.2.1 LINEN

The first and most direct way of producing an envelope is by using the LINEN statement. It produces a linear envelope as shown in the following diagram.

Diagram 9 Linear Envelope



Its general form is

kr linen amp, attack, dur, decay

where

amp is the maximum amplitude.

attack is the duration of the attack.

dur is the duration of the sound.

decay is the duration of the decay.

4.2.2 LINSEG

Sometimes, more complicated shapes are desired. LINSEG is used do this. By specifying a list of amplitudes and the time it takes to reach each amplitude from the previous one, the envelope can be calculated.

For example if an event lasts 3 seconds, and the desired envelope starts at 0, then increases in .5 sec to an amplitude of 20,000, then decreases to 8,000 during the following 1.2 seconds, followed by an increase to 15,000 after .6 sec and finally decreases to 0 for the remaining 1.1 seconds, the amplitudes are

0 , 20000 , 8000 , 15000 , 0

The time durations are

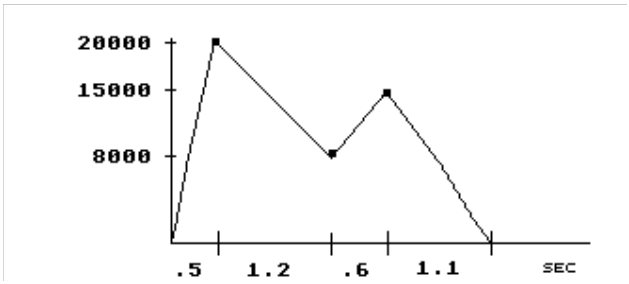
From 0 to 200000	.5 sec
From 20000 to 8000	1.2 sec
From 8000 to 150000	.6 sec
From 15000 to 0	1.1 sec

and the following list of numbers represents the envelope

0 ,	0.5 ,	20000 ,	1.2 ,	8000 ,	0.6 ,	15000 ,	1.1 ,	0
↑	↑	↑	↑	↑	↑	↑	↑	↑
amp	dur	amp	dur	amp	dur	amp	dur	amp

Then straight linear interpolation is used to calculate the values in between, as the following diagram shows

Diagram 10



The general form of LINSEG is therefore

kr linseg amp1, dur1, amp2, dur2, amp3, dur3, ...

where

amp1, amp2, amp3... are the amplitudes.

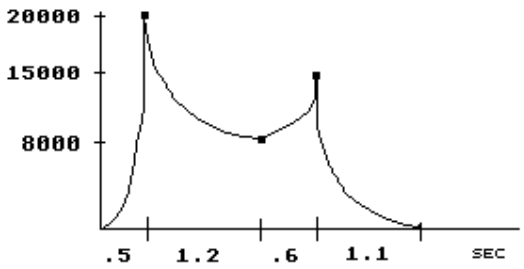
dur1, dur2, dur3... are the time durations.

4.2.3 EXPSEG

This envelope generator works in a similar fashion to LINSEG, the only difference being that instead of interpolating straight lines between the given points, it uses exponential segments.

It is worth remembering that an exponential function can never be 0. Therefore if the same values as before were to be used, the computer would issue an error message. In order to avoid this, a very small value, like 0.0001 can be given. For example, if an EXPSEG generator is given the same values as before replacing the zeros at the beginning and the end with 0.0001 the result is :

Diagram 11



4.2.4 Other Means of Producing Envelopes

There are at least two other ways of producing an envelope. One is the envelope generator ENVLPX. It is explained in the *Csound* manual and will not be covered here.

The second method is achieved by using an oscillator to generate an envelope. The envelope generator must have a period of oscillation equal to the duration of the event. In order to achieve an envelope oscillator with a period **T** equal to **p3** the duration of the event its frequency must be

freq = 1/T = 1/p3

The advantage of using an oscillator is that it can take advantage of the GEN routines in order to generate most diverse and capricious envelopes.

Following is an example of an instrument that uses an oscillator to generate an envelope with its shape of the determined by the function table 1. The shape of the audio waveform is determined by the function table 10.

```
instr 3
k1      oscil   p4, 1/p3, 1; envelope
a1      oscil   k1, p5, 10; audio signal
        out     a1; output
endin
```

4.3 **Example - Three Different Envelopes**

CS4.ORC contains three instruments that play a waveform with different envelopes.

instr 4 uses LINEN with an attack given by the parameter **p6** and a decay time given by **p7** . As usual, **p3** represents the duration.

instr 5 uses LINSEG with an attack that is .05 of the duration of the note (**p3**), then a decrease to .4 of the amplitude that takes .3 of the duration of the note, followed by an increase to .75 of the amplitude in .15 of the duration of the note, and finally a decay to zero in the remaining time (.5 of the duration).

instr 6 uses EXPSEG with the same amplitudes and durations as above.

In the score, a 2 second sound coming from the same signal generator is played in each instrument giving different results according to the envelope as shown in diagrams 12, 13 and 14. The results can be heard in *Csound* tape example 4.

Diagram 12 Instrument 4 :Use of LINEN

Diagram 13 Instrument 5 :Use of LINSEG

Diagram 14 Instrument 6 :Use of EXPSEG

The orchestra and the score are listed below:

```
; CS4.ORC       -       Simple Oscillators
;with envelopes

sr = 44100
kr =  4410
ksmps = 10
nchnls = 1

instr 4; uses LINEN

;p4amplitude
;p5frequency
;p6attack
;p7decay
;p8function table

k1linenp4, p6, p3, p7; envelope
a1oscilk1, p5, p8; oscillator
outa1             ; output

endin

instr 5; uses LINSEG

;p4amplitude
;p5frequency
;p6function table

i1=.05*p3                      ;dur1
i2=.15*p3                      ;dur2
i3=.3*p3                      ;dur3
i4=.5*p3                      ;dur4
k1linseg0,i1,p4,i2,.3*p4,i3,.75*p4,i4,0 ; envelope
```

```
a1oscil   k1, p5, p6                    ; oscillator
out     a1                              ; output
endin


instr 6; uses EXPSEG

;p4amplitude
;p5frequency
;p6function table

i1=.05*p3                               ;dur1
i2=.15*p3                               ;dur2
i3=.3*p3                                ;dur3
i4=.5*p3                                ;dur4
k1expseg  .0001,i1,p4,i2,.3*p4,i3,.75*p4,i4,.0001 ;env
a1oscil   k1, p5, p6                            ; osc
outa1                                   ; out
endin




; CS4.SC

f5 0 512 10 1 .5 .7 .9 .2 .87 .76

;Uses simple linear envelope
;               p3      p4      p5      p6      p7      p8
;instr  start   dur     amp     freq    rise    decay   func

i4      0       2       30000   120     .05     1       5

s
```

```
;Uses linear envelope
;             p3      p4      p5      p6
;instr  start   dur     amp     freq    func

i5      1       2       30000   120     5

s

;Uses exponential envelope
;             p3      p4      p5      p6
;instr  start   dur     amp     freq    func

i6      1       2       30000   120     5

e
```

5. Some Useful Devices

In addition to statements representing oscillators, envelopes and filters, *Csound* offers a series of facilities in the shape of functions that can perform quick operations to relieve the user from doing tedious calculations. For example, there is a function to convert dB to amplitude and another to convert amplitude back to dB.

Some of these facilities will be surveyed in this chapter. At the same time, new examples of slightly more sophisticated instruments will be shown.

5.1 LINE

This time we will begin with an example. An instrument will be designed that has some voice-like qualities. As an example it will be used to play the first two *Kyrie Eleison* of Palestrina's *Missa Papae Marcelli*.

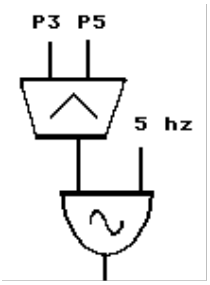
The properties that add interest to the instrument are

1. An amplitude that 'swells' to up to an additional 1/5 of the maximum amplitude indicated by **p4**. In order to do this, an envelope generator that produces a straight line is used.

   Diagram 15 Line Envelope Generator

2. A 5 hz vibrato (frequency fluctuation), of width that varies from 0 to 1/100 of the frequency (**p5**) in the first half of the note and then decreases to 0 at the end. This is achieved by using an envelope to determine the vibrato width, which is then fed into an oscillator that produces the actual vibrato subsequently added to the actual frequency.

Diagram 16 Vibrato Module



The amplitude swell can be implemented by using LINSEG. However, this is an opportunity to introduce a new statement: LINE.

This statement is a variation of LINSEG that produces only one linear segment. Its general form is

kr line amp1, dur, amp2

where

amp1 is the first amplitude

amp2 is the last amplitude

dur is the duration.

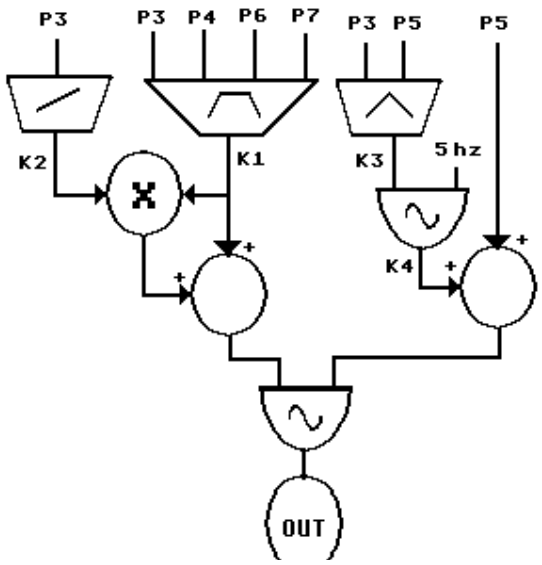
Therefore this statement is equivalent to

kr linseg amp1, dur, amp2

Since the former two improvements take care of adding higher harmonics and some dynamic variation, the sound generator does not need to be very complicated. In fact, a sine wave is used.

The whole instrument is shown in diagram 17

Diagram 17 Instrument 7



The orchestra and score are listed below. *Csound* tape example 5 contains the results.

```
; CS5.ORC       -       Oscillator with variable width
;                       vibrato and swell.

;p4     amplitude
;p5     frequency
;p6     attack
;p7     decay
;p8     function table

sr = 44100
kr =  4410
ksmps = 10
nchnls = 1

instr 7

    i1   =      1/5             ; maximum swell
      i2   =      p5/100          ; maximum vib width
      i3   =      p3/2            ; half a cycle
```

```
        k1   linen   p4,p6,p3,p7          ; envelope
        k2   line    0, p3, i1            ; swell
        k3   linseg  0, i3, i2, i3, 0     ; vib width
        k4   oscil   k3, 5, 1             ; vibrato
        a1   oscil   k1*(1+k2), p5+k4, p8 ; oscillator
        out     a1                  ; output

endin
; CS5.SC
; First two Kyrie Eleison from Palestrina's Missa Papae Marcelli

f1 0 8192 10 1

;       SOPRANO
;               p3      p4      p5      p6      p7      p8
;instr  start   dur     amp     freq    attack  decay   func
i7      1       1.5     4000    587.329 .05     .05     1    ; D
i7      +       0.5     .       .       .       .       .    ; D
i7      +       1       .       .       .       .       .    ; D
i7      +       0.75    .       783.991 .       .       .    ; G
i7      +       0.25    .       698.456 .       .       .    ; F
i7      +       .       .       659.255 .       .       .    ; E
i7      +       .       .       587.329 .       .       .    ; D
i7      +       .       .       523.251 .       .       .    ; C
i7      +       .       .       493.883 .       .       .    ; B
i7      +       .       .       440     .       .       .    ; A
i7      +       .       .       391.995 .       .       .    ; G
i7      +       1       .       .       .       .       .    ; G
i7      +       0.5     .       369.994 .       .       .    ; F#
i7      +       1       .       391.995 .       .       .    ; G
; second phrase
i7      +       .       .       587.329 .       .       .    ; D
i7      +       0.75    .       493.883 .       .       .    ; B
i7      +       0.25    .       440     .       .       .    ; A
```

```
i7    +      .    .     493.883 .      .      .   ; B
i7    +      .    .     523.251 .      .      .   ; C
i7    +      0.5  .     587.329 .      .      .   ; D
i7    +      0.75 .     659.255 .      .      .   ; E
i7    +      0.25 .     587.329 .      .      .   ; D
i7    +      .    .     523.251 .      .      .   ; C
i7    +      .    .     440     .      .      .   ; A
i7    +      0.75 .     587.329 .      .      .   ; D
i7    +      0.25 .     523.251 .      .      .   ; C
i7    +      1    .     523.251 .      .      .   ; C
i7    +      0.5  .     493.883 .      .      .   ; B
i7    +      2    .     523.251 .      .      .   ; C


;       ALTO
;               p3     p4     p5     p6     p7     p8
;instr  start   dur    amp    freq   attack decay  func

i7    3      1.5  .     391.995 .      .      .   ; G
i7    +      0.5  .     .       .      .      .   ; G
i7    +      1    .     .       .      .      .   ; G
i7    +      2    .     523.251 .      .      .   ; C
i7    +      .    .     493.883 .      .      .   ; B
; second phrase
i7    10.5   1    .     391.995 .      .      .   ; G
i7    +      0.5  .     .       .      .      .   ; G
i7    +      1    .     .       .      .      .   ; G
i7    +      0.5  .     440     .      .      .   ; A
i7    +      1    .     391.995 .      .      .   ; G
i7    +      0.5  .     349.228 .      .      .   ; F
i7    +      1    .     391.995 .      .      .   ; G
i7    +      2    .     329.627 .      .      .   ; E
```

```
;       TENOR 1
;               p3      p4      p5      p6      p7      p8
;instr  start   dur     amp     freq    attack  decay   func

i7      0       1.5     4000    293.664 .05     .05     1    ; D
i7      +       0.5     .       .       .       .       .    ; D
i7      +       1       .       .       .       .       .    ; D
i7      +       0.75    .       391.995 .       .       .    ; G
i7      +       0.25    .       349.228 .       .       .    ; F
i7      +       0.5     .       329.627 .       .       .    ; E
i7      +       .       .       293.664 .       .       .    ; D
i7      +       1.5     .       329.627 .       .       .    ; E
i7      +       0.5     .       293.664 .       .       .    ; D
i7      +       1       .       261.625 .       .       .    ; C
; second phrase
i7      +       1       .       293.664 .       .       .    ; D
i7      +       0.5     .       246.941 .       .       .    ; B
i7      +       .       .       195.997 .       .       .    ; G
i7      +       0.75    .       391.995 .       .       .    ; G
i7      +       0.25    .       349.228 .       .       .    ; F
i7      +       0.5     .       329.627 .       .       .    ; E
i7      +       .       .       293.664 .       .       .    ; D
i7      +       0.5     .       261.625 .       .       .    ; C
i7      +       1       .       329.627 .       .       .    ; E
i7      +       0.5     .       293.664 .       .       .    ; D
i7      +       0.5     .       329.627 .       .       .    ; E
i7      +       0.25    .       293.664 .       .       .    ; D
i7      +       .       .       261.625 .       .       .    ; C
i7      +       1       .       293.664 .       .       .    ; D
i7      +       2       .       261.625 .       .       .    ; C


;       TENOR 2
;               p3      p4      p5      p6      p7      p8
```

```
;instr  start   dur     amp     freq    attack  decay   func

i7      8       1.5     4000    293.664 .05     .05     1    ; D
i7      +       0.5     .       .       .       .       .    ; D
i7      +       1       .       .       .       .       .    ; D
i7      +       0.75    .       391.995 .       .       .    ; G
i7      +       0.25    .       349.228 .       .       .    ; F
i7      +       .       .       329.627 .       .       .    ; E
i7      +       .       .       293.664 .       .       .    ; D
i7      +       .       .       261.625 .       .       .    ; C
i7      +       .       .       246.941 .       .       .    ; B
i7      +       0.5     .       220     .       .       .    ; A
i7      +       0.5     .       246.941 .       .       .    ; B
i7      +       1       .       261.625 .       .       .    ; C


;       BASS 1
;               p3      p4      p5      p6      p7      p8
;instr  start   dur     amp     freq    attack  decay   func

i7      2       1.5     4000    195.997 .05     .05     1    ; G
i7      +       0.5     .       .       .       .       .    ; G
i7      +       1       .       .       .       .       .    ; G
i7      +       1.5     .       261.625 .       .       .    ; C
i7      +       0.5     .       246.941 .       .       .    ; B
i7      +       1       .       220     .       .       .    ; A
i7      +       2       .       195.997 .       .       .    ; G
; last chord
i7      16      2       .       130.812 .       .       .    ; C


;       BASS 2
;               p3      p4      p5      p6      p7      p8
;instr  start   dur     amp     freq    attack  decay   func
```

```
i7    9     1.5   4000   195.997 .05    .05    1    ; G
i7    +     0.5   .      .       .      .      .    ; G
i7    +     1     .      .       .      .      .    ; G
i7    +     1.5   .      261.625 .      .      .    ; C
i7    +     0.5   .      246.941 .      .      .    ; B
i7    +     1     .      220     .      .      .    ; A
i7    +     3     .      391.995 .      .      .    ; G

e
```

A new feature can be noticed in the score: when consecutive events are played, a '**+**' in the start time field means 'play after the previous sound has finished'. This effectively relieves us from having to calculate the start time for each event.

5.2 EXPON

There is an equivalent of LINE that produces an exponential between to amplitudes. It is called EXPON and it uses the same parameters as LINE.

Also

kr expon amp1, dur , amp2

is equivalent to

kr expseg amp1, dur, amp2

It is now possible to produce an instrument that uses an exponential swell instead of the linear one:

```
instr 8; Oscillator with varying  vibrato rate and ;width, and exponential 'swelling' amplitude.
    i1  =     1/5              ; maximum swell
      i2  =     p5/100           ; maximum vib width
```

```
        i3   =      p3/2              ; half a cycle
        k1   linen   p4,p6,p3,p7        ; envelope
        k2   expon   .0001, p3, i1      ; exponential swell
        k3   linseg  0, i3, i2, i3, 0   ; vib width
        k4   oscil   k3, 5, 1           ; vibrato
        a1   oscil   k1*(1+k2), p5+k4, p8 ; oscillator
        out    a1                ; output
endin
```

The previous score can be used.

5.3 Pitch Representation

The fact that pitches are represented by frequencies makes the task of producing the score in the previous examples a very tedious and unnatural one. Fortunately, there are two alternatives to using frequencies.

5.3.1 PCH Representation

This is akin to tempered tuning: A pitch is represented by a number consisting of an integer representing the octave and a fractional part usually ranging from .00 to .12 representing the number of semitones above C.

The middle C octave is represented by the integer 8. Therefore, the lower octave is 7, two octaves lower 6, and so on. In a similar way, the octaves above are 9, 10, etc.

Examples

8.07is 7 semitones above middle C: G.
6.10is 10 semitones above C two octaves below: Bb.

Microtones can be achieved by using more decimal places. For example

8.065 is a quarter tone above F#.

Finally, if the number of semitones indicated by the decimal part is bigger than 12, this will

indicate more than an octave above C.
For example

8.15 is 15 semitones above middle C: Eb a minor tenth above. This is equivalent to Eb, a minor third above C in the next octave: 9.03.

5.3.2 OCT Representation

This representation uses a number consisting of an integer part that represents the octave, and a fraction that represent an actual subdivision of the octave.

For example
8.5 is half an octave above middle C. This is F#. If PCH representation was used, this would be represented as 8.06.

The score in the previous example can now be written using PCH representation. As an example, the first phrase of Tenor 1 is listed below. The whole score is given as CS6.SC in the floppy disk containing the code for the examples.

```
;       TENOR 1
;               p3      p4      p5      p6      p7      p8
;instr  start   dur     amp     PCH     attack  decay   func

i9      0       1.5     4000    8.02    .05     .05     1   ; D
i9      +       0.5     .       .       .       .       .   ; D
i9      +       1       .       .       .       .       .   ; D
i9      +       0.75    .       8.07    .       .       .   ; G
i9      +       0.25    .       8.05    .       .       .   ; F
i9      +       0.5     .       8.04    .       .       .   ; E
i9      +       .       .       8.02    .       .       .   ; D
i9      +       1.5     .       8.04    .       .       .   ; E
i9      +       0.5     .       8.02    .       .       .   ; D
i9      +       1       .       8.00    .       .       .   ; C
```

5.3 **Pitch Converters**

The use of PCH notation makes the score easier to write and understand. However, this presents a problem in the orchestra: OSCIL needs data expressed in cycles per second (CPS or hz). Therefore, it is necessary to translate PCH data into frequency.

Csound provides a function that can convert PCH information into cycles per second: CPSPCH.

The general form of this converter is

variable = cpspch(PCH)

where

variable is equal to the frequency corresponding to the PCH representation.

Instrument 9 is an altered version of instrument 7 that can cope with PCH notation thanks to the variable **i0**, which is made equal to the frequency corresponding to **p5** (given in PCH representation) and use subsequently instead of **p5**.

```
i0 = cpspch(p5)
```

The whole instrument is given next

```
; CS6.ORC-Oscillator with variable width
;       vibrato and swell.
;           Accepts pitch and converts it ;to frequency.

;p4    amplitude
;p5    frequency
;p6    attack
;p7    decay
;p8    function table

sr = 44100
kr =  4410
```

```
ksmps = 10
nchnls = 1

instr 9

    i0=   cpspch(p5)          ; pitch to frequency
    i1=       1/5             ; max swell
    i2=       i0/100          ; maximum vib width
    i3=       p3/2            ; half a cycle
    k1linen   p4,p6,p3,p7     ; envelope
    k2line    0, p3, i1       ; swell
    k3linseg  0, i3, i2, i3, 0    ; vib width
    k4oscil   k3, 3, 1        ; vibrato
    a1oscil   k1*(1+k2), i0+k4, p8 ; oscillator
          out     a1          ; output
endin
```

Csound tape example 6 shows the result of using CS6.ORC and CS6.SC which is identical to example 5.

There are other converters that transform OCT to frequency, PCH to OCT, etc., namely
CPSPCHconverts PCH to CPS.
CPSOCTconverts OCT to CPS.
OCTCPSconverts CPS to OCT.
OCTPCHconverts PCH to OCT.
PCHOCTconverts OCT to PCH.
There is no conversion from CPS to PCH.

5.4 Value Converters

Csound offers functions that help avoid tedious calculations called *value converters*. In this section, two of these will be used to produce an instrument that expands or squeezes the octave around middle C according to a factor given by **p9**.

The principle behind this new instrument is: At the beginning of each note, the given pitch is converted into semitones above or below middle C. This is stored in the variable **ist** (if ist is positive then the pitch is above middle C otherwise it is below). Then, that number of semitones is multiplied by the factor given by p9. Finally, it is added to middle C.

The number of semitones above (below) middle C is found as follows:

Middle C in PCH notation is octave 8. That is 8 x 12 = 96 semitones.

The pitch given is separated into

1. Its integer part, representing the number of octaves, which is subsequently multiplied by 12.

2. Its fractional part, representing the number of semitones above that octave. In order to find the actual number of semitones, this fraction has to be multiplied by 100.

Therefore, the total number of semitones is

integer part x 12 + fractional part x 100

Finally, the variable **ist** is equal to

ist = total number of semitones - 96

For example, if G, 7 semitones above middle C is given, its PCH representation will be 8.07.

integer part = 8->8 x 12 = 96 semitones

fractional part = .07->.07 x 100 = 7 semitones

total number of semitones = 96 + 7 = 103
ist = 103 - 96=7 semitones above

The integer and fractional parts of the PCH representation are obtained by respectively using the converters INT and FRAC.

instr 10 is the implementation of an instrument that can squeeze or expand intervals above and below middle C.

```
; CS7.ORC-Oscillator with variable width
;              vibrato and swell. Accepts pitch and converts
;to frequency and squeezes or expands the
;octave above and below middle C

;p4     amplitude
;p5     frequency
;p6     attack
;p7     decay
;p8     function table
;p9     squeeze/expansion factor

sr = 44100
kr =  4410
ksmps = 10
nchnls = 1

instr 10

;squeeze or compress

    ist  = int(p5)*12 + frac(p5)*100 - 96
                                    ; semitones above
                                    ; middle C
       itot = 96 + ist*p9          ; total semitones
```

```
        ioc  =     int(itot/12)         ; octave
        ist  =     itot - ioc*12        ; semitones above
        i0   =     ioc+ist/100          ; to PCH notation                    ; middle C

;the rest is identical to instrument 9

        i0   =     cpspch(i0)           ; PCH to freq
        i1   =     1/5                  ; max swell
        i2   =     i0/100               ; max vib width
        i3   =     p3/2                 ; half a cycle
        k1   linen p4,p6,p3,p7          ; envelope
        k2   line  0, p3, i1            ; swell
        k3   linseg0, i3, i2, i3, 0     ; vib width
        k4   oscil   k3, 3, 1           ; vibrato
        a1   oscil   k1*(1+k2), i0+k4, p8 ; oscillator
          out     a1                    ; output
endin
```

The score in the two previous examples can be used with instrument 10 by adding parameter **p9**.

Csound tape example 7 contains two versions: First, the octave is squeezed by .65 and then it is expanded by 1.35. They do not appear to do much good to this piece. However, this instrument could be used to investigate non-rigid transformations of the tempered or any other scale. Non-rigid transformations do not preserve interval distances.

The previous instrument can still be made more sophisticated. The centre pitch can be given in the score and set as a variable in the orchestra. All that has to be done is convert it to semitones.

Instrument 11 is a realization of this enhancement and *Csound* tape example 8 uses the first note in (yet) the same score - D above middle C - as the centre pitch and expands the octave by a factor of 1.3 .

```
; CS8.ORC-Oscillator with variable width
;             vibrato and swell.
;             Accepts pitch and converts to frequency.
;             and squeezes or expands the octave above and
;             below a centre pitch given by p10

;p4     amplitude
;p5     frequency
;p6     attack
;p7     decay
;p8     function table
;p9     squeeze/expansion factor
;p10    centre pitch

sr = 44100
kr =  4410
ksmps = 10
nchnls = 1

instr 11



;Convert centre pitch to semitones

        icst=         int(p10)*12 + frac(p10)*100

;squeeze or compress

     ist  = int(p5)*12 + frac(p5)*100 - icst
                                       ; semitones above
                                       ; centre pitch
        itot = icst + ist*p9        ; total semitones
```

```
        ioc  =     int(itot/12)         ; octave
        ist  =     itot - ioc*12        ; semitones above
        i0   =     ioc+ist/100          ; to PCH notation                                    ; middle C

;the rest is identical to instrument 9

        i0   =     cpspch(i0)           ; PCH to freq
        i1   =     1/5                  ; max swell
        i2   =     i0/100               ; max vib width
        i3   =     p3/2                 ; half a cycle
        k1   linen p4,p6,p3,p7          ; envelope
        k2   line  0, p3, i1            ; swell
        k3   linseg0, i3, i2, i3, 0     ; vib width
        k4   oscil   k3, 3, 1           ; vibrato
        a1   oscil   k1*(1+k2), i0+k4, p8 ; oscillator
          out     a1                    ; output

endin
```

5.5 **Example - Fan Out**

After the lengthy discussion on pitch representation and manipulation, *Csound* may appear to be a note oriented synthesis language. This is not necessarily so, since an event does not have to be a note. On the contrary, sounds with very complex morphologies can be produced, as will be shown in later sections.

The next example shows an instrument that, given a certain frequency, 'fans out' 3 upper and 3 lower glissandi. The width of each glissando is a multiple of a fraction of the original frequency. The fraction is determined by **p8** in the score.

For example, if 440 hz and a factor of .1 are given

The separation in hz will be

440 hz x .1=44 hz

The upper glissandi will fan out to 440 + 44 = 484 hz
484 + 44 = 528 hz
528 + 44 = 572 hz

and the lower will fan to

440 – 44 = 396 hz
396 – 44 = 352 hz
352 – 44 = 308 hz

Two versions are given: One with a linear fan-out (instrument 12) and one with an exponential fan-out (instrument 13).

Csound example 9 shows the results of using instrument 12 with CS9.SC (given below).

```
; CS9.ORC       -       Fan Out Instrument
;               From one frequency, 3 upper and 3 lower glissandi
;               fan out. They are separated by a fraction of the
;               frequency given in p9

sr = 44100
kr =  4410
ksmps = 10
nchnls = 1

instr 12;       linear fan-out
```

```
;p4 :   amplitude
;p5 :   frequency
;p6 :   attack
;p7 :   decay
;p8 :   fraction
;p9 :   function table

    ihz     =       p8*p5; separation in hz
    iu1     =       p5+ihz; first fan up freq.
    iu2     =       iu1+ihz; second fan up freq.
    iu3     =       iu2+ihz; third fan up freq.
    il1     =       p5-ihz; first fan down freq.
    il2     =       il1-ihz; second fan down freq.
    il3     =       il2-ihz; third fan down freq.

    k1      linen   p4, p6, p3, p7  ; envelope
; linear glissandi

    ku1     line    p5, p3, iu1; first glissando up
    ku2     line    p5, p3, iu2; second gliss. up
    ku3     line    p5, p3, iu3; third gliss. up
    kl1     line    p5, p3, il1; first gliss down
    kl2     line    p5, p3, il2; second gliss down
    kl3     line    p5, p3, il3; third gliss down

; each glissando has its own oscillator

    al1     oscil   k1, kl1, p9     ; oscillator
    al2     oscil   k1, kl2, p9     ; oscillator
    al3     oscil   k1, kl3, p9     ; oscillator
    au1     oscil   k1, ku1, p9     ; oscillator
    au2     oscil   k1, ku2, p9     ; oscillator
au3     oscil   k1, ku3, p9     ; oscillator
```

```
; mix and output
out     (au1+au2+au3+al1+al2+al3)/6

endin


instr 13;         exponential fan-out

;p4 :   amplitude
;p5 :   frequency
;p6 :   attack
;p7 :   decay
;p8 :   fraction
;p9 :   function table

    ihz     =       p8*p5; separation in hz
    iu1     =       p5+ihz; first fan up freq.
    iu2     =       iu1+ihz; second fan up freq.
    iu3     =       iu2+ihz; third fan up freq.
    il1     =       p5-ihz; first fan down freq.
    il2     =       il1-ihz; second fan down freq.
    il3     =       il2-ihz; third fan down freq.

    k1      linen   p4, p6, p3, p7  ; envelope

; linear glissandi. In this case CARE must be taken in order ; to avoid iu1, iu2, iu3, il1, il2,
il3  being negative or ; zero since an exponential can never be zero or change ; sign. This
is done by not allowing p8 to be larger than ; 1/3

    ku1     expon   p5, p3, iu1; first glissando up
    ku2     expon   p5, p3, iu2; second gliss. up
    ku3     expon   p5, p3, iu3; third gliss. up
    kl1     expon   p5, p3, il1; first gliss down
```

```
    kl2     expon   p5, p3, il2; second gliss down
    kl3     expon   p5, p3, il3; third gliss down

; each glissando has its own oscillator

    al1     oscil   k1, kl1, p9     ; oscillator
    al2     oscil   k1, kl2, p9     ; oscillator
    al3     oscil   k1, kl3, p9     ; oscillator
    au1     oscil   k1, ku1, p9     ; oscillator
    au2     oscil   k1, ku2, p9     ; oscillator
au3     oscil   k1, ku3, p9     ; oscillator

; mix and output
out     (au1+au2+au3+al1+al2+al3)/6

endin

;CS9.SCExample score

f1 0 512 10 1 .7 .4 1 .3 .8 .95

;               p3      p4      p5      p6      p7      p8      p9
;instr  start   dur     amp     freq    attack decay  fract. func

i12     0       1       15000   300     .05     1     .3      1
i12     2       6       4500    60      .05     1     .2      1
i12     4.5     4       .       680     .       .     .3      1
i12     4.7     3.5     .       400     .       .     .1      1
i12     5       3       .       560     2       1     .15     1
i12     6.4     1.5     .       230     .       .5    .25     1
i12     6.7     1.3     .       1200    .       .5    .06     1
i12     6.8     3       .       2000    .       1     .13     1

e
```

PART II

Linear Synthesis Techniques

6. Additive Synthesis

Additive synthesis is one of the oldest techniques. The main assumption is that sound can be decomposed into a unique sum of pure sine waves - called *partials* - each with its own amplitude, frequency and phase. The description of a sound in terms of the frequencies and amplitudes of its partials is called the *spectrum*.

6.1 Static Spectrum

The simplest approach to additive synthesis is to imagine that the spectrum of a sound is always the same throughout its duration. In other words, the spectrum is assumed to be *static*.

If the sound waveform is represented by a function of time: s(t), this function can be expressed as follows

$$s(t) = A_0 + A_1 \sin(2 \pi f_1 t + p_1) + A_2 \sin(2 \pi f_2 t + p_2) + \ldots$$

$$= A_0 + \sum_i \{ A_i \sin(2 \pi f_i t + p_i) \}$$

$$\pi = 3.1415927\ldots$$

where the amplitudes A_0, A_1, A_2 ... the frequencies f_0, f_1, f_2 ... , and phases p_0, p_1, p_2 ... are all constant: they do not change as time goes by.

It can then be inferred that:

1. Different sets of amplitudes and frequencies (different spectra) produce different sounds. Experiments seem to indicate that the phase is not a significant factor in sound recognition, but rather has to do more with positioning a sound in space.
2. Two sounds that have the same component frequencies but differ in the relative amplitudes of these partials, have different timbre.

3. Sounds that have partials that are close to multiples of the lowest frequency (fundamental) tend to be more 'pitched' than those that do not. When there is such a relation, the partials are also called *harmonics* and the spectrum is said to be *harmonic*.

6.1.1Harmonic Spectrum

Harmonic static spectrum has been dealt with when examining GEN10, which requires the relative amplitude of each harmonic. Examples can be found in chapter 3, section 3.10, (CS2.ORC and CS2.SC).

This is an appropriate place to introduce an statement very similar to OSCIL. This statement is called OSCILI and requires exactly the same parameters as OSCIL.

ar oscili amp, freq, func (, phase)

The only difference between these two is that OSCILI uses a more accurate method of producing a waveform, therefore the sounds so produced are cleaner. But, as expected, there is a price that has to be paid: OSCILI is about twice as slower to compile than OSCIL.

In order to enjoy the speed of OSCIL with the accuracy of OSCILI, larger function tables can be used with OSCIL (8192 points or more) which sample the cycle of a waveform more accurately at the expense of more memory needed to store the function table. A few table sizes of 8192 should not be a terrible problem to handle for a 512 Kbyte or 1 Mbyte RAM microcomputer.

6.1.2 Inharmonic Spectrum

Inharmonic spectra occur when the frequencies of the partials are not multiples of a fundamental. This requires a signal generator that allows specifying the frequency of each partial as well as its amplitude.

Theoretically, such a generator is available in GEN9, which also allows specification of the phase. Thus, for each partial GEN9 requires three values

freq ratio amplitude phase

where

amplitude is the relative amplitude of that partial, like in GEN10.

freq ratio is the ratio between the frequency of that partial and the fundamental. For example, if the fundamental is 100 hz and the partial is 141 hz, the ratio is

141 / 100=1.41

phase is the phase of the partial given in degrees.

For example, the function
f2 0 4096 9 1 1 0 1.41 .8 20 1.89 .9 45 2.3 .7 90

defines a waveform that has the following 4 partials.

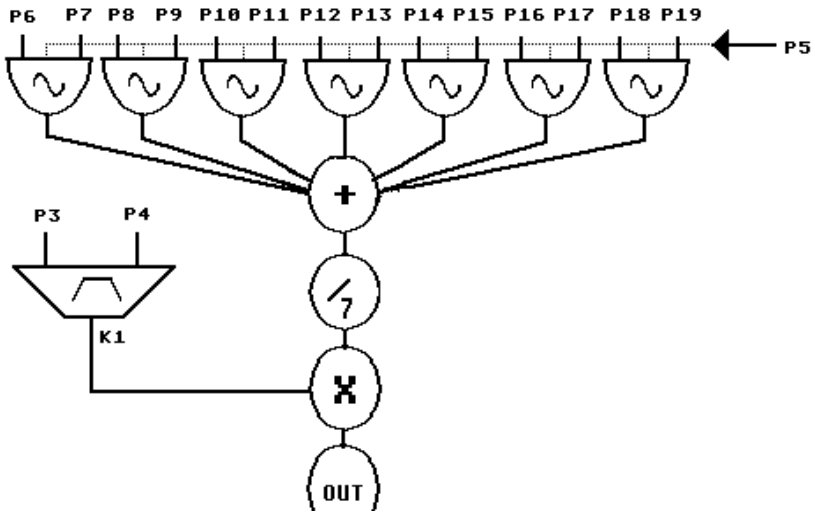
	freq ratio	amplitude	phase
fundamental	1	1	0
2nd partial	1.41	.8	20
3rd partial	1.89	.9	45
4th partial	2.3	.7	90

If the fundamental were 100 hz, then the partials would be: 141, 189 and 230 hz.

However, this will not really produce the effect of a sound that has different inharmonic spectrum. What GEN9 really does is to produce a cycle that can be repeated, with no regard to the fact that the partials are not necessarily contained a whole number of times in that cycle, thus truncating their continuation. In the end, because of the fact that a cycle is repeated, a pitched sound is produced regardless of how complicated the data given to GEN9 may be. Effectively, because of the truncation of the partials, the spectrum of the resulting sound is distorted and does not necessarily contain the specified frequencies.

This can be shown if an instrument that truly produces inharmonic spectra is devised. The instrument is readily done by dedicating an oscillator to each partial and mixing the result at the end, as shown in diagram 18. Instrument 14 (shown below in CS10.ORC) is a realisation of this diagram.

Diagram 18 Instrument that Produces Seven Partials



In order to prove that the results given by this instrument and GEN9 are different, CS10.ORC and CS10.SC implement a waveform that is generated both by a simple oscillator using GEN9, and by instrument 14 using pure sines. For the sake of simplicity, the phases are left to be 0. This comparison can be heard in *Csound* tape example 10. First the GEN9 generated signal is played, then the one produced with instrument 14, which is the truly unpitched sound.

Waveform description

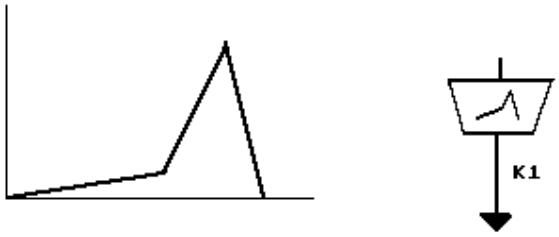
Partial number	RelativeFrequency	Relative Amplitude
fundamental	1	1
2	0.8	1.41
3	0.9	1.89
4	0.7	2.3
5	0.65	2.6
6	0.93	3.2
7	0.94	3.5

```
; CS10.ORC
sr = 44100
kr =  4410
ksmps = 10
nchnls = 1

instr 3        ; Simple Oscillator
               ; Uses table defined by function number
               ; indicated in p8.

       k1linen   p4, p6, p3, p7  ; envelope
       a1     oscil   k1, p5, p8      ; oscillator
              out     a1         ; output
endin

instr 14       ; Inharmonic oscillator. Up to 7 partials.

;p4 : overall amplitude
;p5 : fundamental frequency
;p6, p8, p10, p12, p14, p16, p18 : relative amplitudes
;of partials
;p7, p9, p11, p13, p15, p17, p19 : relative frequencies
;of partials

    i1      =       p5*p7           ; 1 partial freq
    i2      =       p5*p9           ; 2 partial freq
    i3      =       p5*p11          ; 3 partial freq
    i4      =       p5*p13          ; 4 partial freq
    i5      =       p5*p15          ; 5 partial freq
    i6      =       p5*p17          ; 6 partial freq
    i7      =       p5*p19          ; 7 partial freq
    k1      linen   p4, .1, p3, .1  ; envelope
    a1      oscil   p6, i1, 1       ; 1 partial
    a2      oscil   p8, i2, 1       ; 2 partial
```

```
    a3      oscil   p10, i3, 1      ; 3 partial
    a4      oscil   p12, i4, 1      ; 4 partial
    a5      oscil   p14, i5, 1      ; 5 partial
    a6      oscil   p16, i6, 1      ; 6 partial
    a7      oscil   p18, i7, 1      ; 7 partial

    ; mix and output
       out     k1*(a1+a2+a3+a4+a5+a6+a7)/7
endin

;CS10.SC
f1 0 4096 10 1
f2 0 4096 9  1 1 0   1.41 .8 0   1.89 .9 0  2.3 .7 0
  2.6 .65 0   3.2 .93 0   3.5 .94 0

; Using GEN9
;               p3      p4      p5      p6      p7      p8
;instr start    dur     amp     freq    rise    decay   func
i3     0        2       30000   300     .1       .1     2
s

; Using 7 oscillators
;               p3      p4              p5      p6,p8...p18     p7,p9...p19
;instr  start   dur     amp             fund    relative        relative
                        overall         freq    amplitudes      frequencies
i14     2       2       30000           300     1               1
                                                .8              1.41
                                                .9              1.89
                                                .7              2.3
                                                .65              2.6
                                                .93              3.2
                                                .94              3.5
e
```

A snapshot of the waveforms is shown in diagram 19. Observe the periodicity of the GEN9 waveform.

Diagram 19



1. Using GEN9

2. Using Independent Oscillators in *instr 14*

6.2 Dynamic Spectrum

So far a static spectrum has been assumed. However, sounds in nature have a *dynamic* spectra. That is, they change as time goes by. In the early days, this was one of the main reasons for the 'dead' quality of synthesized sounds.

From a formal point of view a waveform is now represented as:

$$s(t) = A_{0(t)} + A_{1(t)} \sin(2 \pi f_{1(t)}t + p_{1(t)}) + \ ...$$

$$= A_{0(t)} + \sum_{i} \{ A_{i(t)} \sin(2 \pi f_{i(t)}t + p_{i(t)}) \}$$

where $A_{0(t)}$, $A_{1(t)}$, $A_{2(t)}$ ... , $f_{0(t)}$, $f_{1(t)}$, $f_{2(t)}$ ... and $p_{0(t)}$, $p_{1(t)}$, $p_{2(t)}$ ... are now themselves functions of time, changing as the sound evolves.

A dynamic spectrum gives sounds their morphology. Therefore, regardless of which

technique is in use, the main aim is always to produce such spectrum and to control its evolution. In additive synthesis, this is done by varying the amplitudes and frequencies of the partials through time, which amounts to giving each of these components an individual amplitude and frequency envelope.

Csound offers a module called ADSYN. This used to require some tedious preparation, but there is a graphical interface, ADSYN DRAW, created by Richard Orton at York University, through which the envelopes for each partial are drawn using a mouse. Information about this program is available in the CDP users manual.

6.3 Example - A Dynamic Spectrum Simple Instrument

In spite of having a graphics interface like ADSYN DRAW, producing a sound by creating each envelope is still a very lengthy process that implies accurate knowledge of the envelopes of each partial. This is extremely time consuming, especially if a wide range of timbres and morphologies are needed.

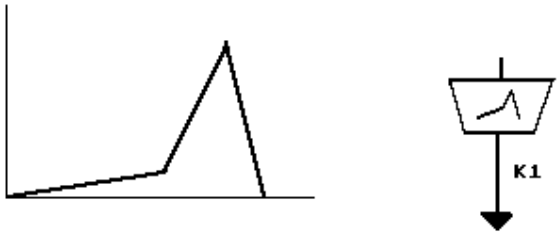
A great deal of work can be saved if the synthesis of a sound is approached in a more or less intuitive way, based on a reasonable assumption of what the desired characteristics are; therefore having a rough idea of how the spectrum varies, how the envelope of that sound affects its morphology, etc. Once the rough sound is produced, it can be refined.

As an example, an instrument will now be produced to comply with one requirement. To produce a dynamic spectrum that results in a very definite gesture.

This will be achieved by making some modifications to *instr 14*, as follows

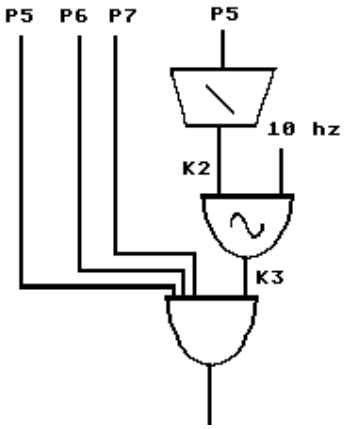
1. An envelope that rises in two stages: slowly at first and then more abruptly as shown in diagram 20. The parameter **p20** is used to determine the position of the breakpoints.

Diagram 20 Overall envelope Shape



2. A vibrato that changes from a maximum width of 1/10 of the fundamental at the beginning of the sound (when it is not very loud), to almost nothing at the end. This vibrato affects each partial differently since it is a frequency added up and not a factor that multiplies all the spectrum. For each oscillator, the diagram looks as follows

Diagram 21 Oscillator for One Partial



In order to obtain a richer spectrum, the generating function is not necessarily restricted to being a sine wave.

```
; CS11.ORC
sr = 44100
kr =  4410
ksmps = 10
```

```
nchnls = 1

instr 15    ;Inharmonic oscillator. Up to 7 partials with
            ;   vibrato shape determined by function 2

;p4 : overall amplitude
;p5 : fundamental frequency
;p6, p8, p10, p12, p14, p16, p18 : relative amplitudes
;of partials
;p7, p9, p11, p13, p15, p17, p19 : relative frequencies
;of partials
;p20 : envelope shaper
;p21 : function table

      i1     =      p5*p7          ; 1 partial freq
      i2     =      p5*p9          ; 2 partial freq
      i3     =      p5*p11          ; 3 partial freq
      i4     =      p5*p13          ; 4 partial freq
      i5     =      p5*p15          ; 5 partial freq
      i6     =      p5*p17          ; 6 partial freq
      i7     =      p5*p19          ; 7 partial freq
      ifrq1  =       1/p3           ; freq = 1/dur

      ; overall envelope

      k1     linseg  0,p3-.7*p20,p4/4,.3*p20,p4,.4*p20,0

      ; frequency vibrato envelope

      k2     line    1, p3, 0
      k3     oscil   k2*p5/10, 10, 2 ; frequency change

      a1     oscil   p6, i1+k3, p21  ; 1 partial
      a2     oscil   p8, i2+k3, p21  ; 2 partial
```

```
        a3      oscil   p10, i3+k3, p21 ; 3 partial
        a4      oscil   p12, i4+k3, p21 ; 4 partial
        a5      oscil   p14, i5+k3, p21 ; 5 partial
        a6      oscil   p16, i6+k3, p21 ; 6 partial
        a7      oscil   p18, i7+k3, p21 ; 7 partial

        ; mix and output

              out     k1*(a1+a2+a3+a4+a5+a6+a7)/7

endin
```

Csound tape example 11 contains the sounds produced with the following score:

```
;CS11.SC

f1 0 1024 10 1
f2 0 1024 10 1
f3 0 1024 10 1 .5 .6 .9 1 .3 .4 .2

t 0 80

;           p3    p4      p5     p6..p18 p7..p19 p20      p21
;instr start dur   amp     fund  relat.  relat.  env      func
;                  overll  freq  amps    freqs   shaper

i15   0     .5    10000   300     1       1
                          .8      1.41
                          .9      1.89
                          .7      2.3
                          .65     2.6
                          .93     3.2
                          .94     3.5     .15     1
i15   1.3   .7    10000    300     1       1
```

				.8	1.41		
				.9	1.89		
				.7	2.3		
				.65	2.6		
				.93	3.2		
				.94	3.5	.15	1
i15	1.8	.5	15000	900	1	1	
				.8	1.41		
				.9	1.89		
				.7	2.3		
				.65	2.6		
				.93	3.2		
				.94	3.5	.15	1
i15	1.9	3	19000	40	1	1	
				.8	1.41		
				.9	1.89		
				.7	2.3		
				.65	2.6		
				.93	3.2		
				.94	3.5	.8	1
i15	4.4	2	19000	600	1	1	
				.8	1.41		
				.9	1.89		
				.7	2.3		
				.65	2.6		
				.93	3.2		
				.94	3.5	1	3

e

6.4 Example - Risset's Beating Harmonics

Jean-Claude Risset's beating harmonics constitute one of the most successful displays of ingenuity achieved with additive synthesis. The principle behind this sound (shown in *Csound* tape example 12), is the very simple fact that when two sine waves are mixed, they produce a cosine wave of half the difference of their frequencies that serves as an envelope to a sine wave of half of the sum of those frequencies.

Assuming that the frequencies of the sine waves are f_1 and f_2. Then m(t), the result of mixing them can be written as follows

$$m(t) = \sin(2\pi f_1 t) + \sin(2\pi f_2 t)$$

but from trigonometry:

$$\sin(a) + \sin(b) = 2\left[\cos\left(\frac{a-b}{2}\right)\sin\left(\frac{a+b}{2}\right)\right]$$

therefore

$$m(t) = 2\left[\cos\left(2\pi\,\frac{f_1 - f_2}{2}\right)\sin\left(2\pi\,\frac{f_1 - f_2}{2}\right)\right]$$

For example, if the frequency of the one of the sine waves is 100 hz and the second one is 150 hz, a cosine of 25 hz will serve as an envelope to a 175 hz sine wave.

Now, if the values of the frequencies are carefully chosen so that they are very close, it is possible to end up with a very slow changing cosine that serves as an envelope to a sine that is itself very close to the original frequencies.

For example, 110 and 110.03 hz will give an envelope of 0.015 hz (one cycle will take about 66 seconds) and an audible sine of 110.015 hz. When the 0.015 hz cosine reaches its maximum value, the 110.015 hz sine will be loud. Conversely, when the cosine value is

close to 0, the sine will be attenuated.

If instead of mixing a two sines, two complex harmonic signals are mixed, then each of the harmonics of one signal will interact with those of the other to produce envelopes that have different rates. Especially important are the corresponding harmonics of 110 and 110.03 which will still be very close creating slow envelopes. For example, the corresponding second harmonics are 220 and 220.06 hz. Their interaction will produce an envelope of 0.03 hz (1 cycle lasts ~33 seconds) shaping a 220.03 hz sine wave. The corresponding third harmonics will produce an envelope of 0.045 hz (1 cycle lasts ~22 seconds) shaping a 330.045 sine, and so on.

Therefore, different harmonics will be heard at different times, corresponding to the maximum value of their envelopes.

Finally, if more harmonic rich signals are mixed, their harmonics will interact with each other creating the beating effect. A good set of values is 109.88, 109.91, 109.94, 109.97, 110, 110.03, 110.06, 110.09 and 110.12 hz, used in CS12.SC . The beauty of the idea lays in the simplicity of its orchestra, which of an oscillator with an envelope (instrument 3). The orchestra and score are listed below.

```
;CS12.ORC      This produces a simple oscillator with an
;envelope needed to produce RISSET's beating ;harmonics

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

instr 3

;p4 : amplitude
;p5 : fundamental
```

```
        k1      linen p4, .5, p3, .5; envelope
        a1      oscil   k1, p5, 1; oscillator
                out     a1

endin


;       CS12.SC
;       This produces the original Risset beating harmonics

f1 0 8192 10 1 1 1 1 1 1 1 1 1 1 1

;               p3      p4      p5
;instr  start   dur     amp     fund

i3      0       35      3500    110
i3      .       .       .       110.03
i3      .       .       .       110.06
i3      .       .       .       110.09
i3      .       .       .       110.12
i3      .       .       .       109.97
i3      .       .       .       109.94
i3      .       .       .       109.91
i3      .       .       .       109.88

e
```

7. Subtractive Synthesis

This is another technique that has been used since the early days of electronic synthesis. The principle behind it is simple: a source signal that is rich in partials is passed through filters that enhance or attenuate some of these, literally shaping the sound.

RICH SIGNAL ---> FILTER --> OUTPUT

7.1 Input - Sources

The main synthetic sources used as input to the filters are noise generators and pulses. Square waves are also rich in harmonics and can be used.

7.1.1 Noise Generators

The spectrum of ideal *white noise* contains all possible frequencies evenly distributed up to half of the sampling rate (Nyquist frequency). Therefore, it is very useful for subtractive synthesis purposes.

The basic white noise generator in *Csound* is RAND, which generates *pseudo*-random numbers every sample. The prefix 'pseudo' is used because these are not real random numbers picked by chance, but rather the result of a formula that simulates randomness by producing a sequence of values that can not be perceived as having any repetition pattern. This formula takes a *seed* number as a starting value from which it generates the rest. Therefore, if the same seed is used, the numbers generated will always be the same.

When the purpose of using RAND is to produce white noise, using the same seed should not make any difference, but if a pseudo-random generator is used to produce a slow sequence of values, like for example, a pitch series, this fact must be taken into account, in which case repetition can be avoided by using a different seed each time a new sequence is produced.

RAND produces values between -1 and +1. To change this range, it is possible to multiply each number by an amplitude. Thus, the resulting series can be set to be between -amplitude and +amplitude.

The general statement for RAND is

ar rand amp (,seed)

where

ar is the output signal.

amp is the amplitude that determines the range. For example, if numbers between -32000 and 32000 are desired, amp = 32000.

seed is the starting value used by the random generator. It is optional, therefore if it is not given, a default value of .5 is taken.

As stated above, white noise has a uniform frequency distribution. Sometimes, noise that has a greater density at lower frequencies is desired sounding more like a rumble.

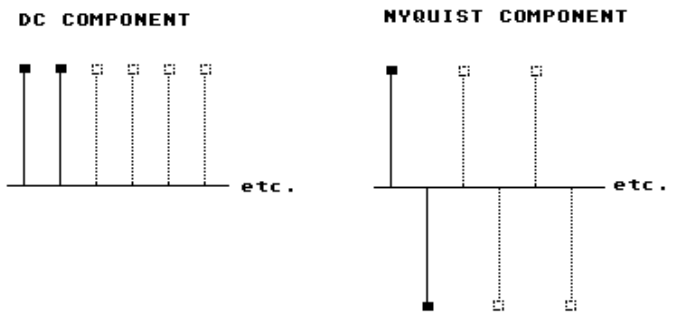
In order to find out how to influence the frequency distribution, a rule of thumb stating that very quick changes in amplitude are a sign of existing high frequencies will be adopted (this is why clicks, that are a sudden change in amplitude, can sometimes be removed by appropriate low pass filters).

This rule can give a heuristic explanation to the uniform distribution of frequencies in white noise: since RAND produces a random number at each sample, it could produce two consecutive random numbers with the same value, which
could be interpreted as part of a constant or 0 hz frequency signal. On the other hand, it could produce 1 and -1 as two consecutive values, which can be interpreted as being part of the component that can change the quickest, having a cycle of two samples, thus a

frequency of half of the sampling rate. But this is no other than the Nyquist frequency. Any two changes between these extremes will belong to other frequencies, larger drops or increments in amplitude will belong to higher frequencies and viceversa. This is illustrated in diagram 22.

The immediate conclusion is that in order to lower the probability of having high frequencies, the rate at which random numbers are produced must be lowered: instead of producing a random number each sample (that is at a frequency equal to half of the sampling rate), it could be produced at a lower frequency.

Diagram 22 Extreme Cases in White Noise



RANDH is an alternative to RAND that can generate random numbers at any frequency. It will hold to the value of the last number generated until it is time to produce a new one. Its general form is

ar randh amp, freq, (,seed)

where

ar is the output.

freq is the frequency at which the random numbers are produced. If the frequency is equal to the sampling rate, the random numbers will be produced every sample, thus the effect is the same as with RAND.

seed is the seed, is also optional and has the same default as with RAND.

RANDH can then produce a signal that has a higher density in the low part of the spectrum. If an even higher predominance of low frequencies is desired, there are still some corners that can be cut. The samples at which the numbers change to a new value, are points of very quick amplitude variation. If they are smoothed, making a gradual transition between two values then those high frequency components can be further neutralised, as shown in diagram 23, which displays the result of series of random numbers produced at a frequency of 400 hz (a new number every 2.5 milliseconds), once by changing values change abruptly and then by changing them gradually.

Diagram 23 Random Numbers Produced Every 2.5 msec

23.1 Abrupt change between values.

23.2 Gradual change between values.

For the reasons mentioned above, there is a third generator in *Csound* called RANDI, which produces random numbers at a given frequency but instead of changing abruptly from one number to the other, it interpolates between them. Its general form has the same variables and default seed as RANDH.

ar randi amp, freq (,seed)

In order to hear the difference between the three, instruments using each of the random generators are implemented in CS13.ORC and compiled with CS13.SC .

```
; CS13.ORC      -       Use of RAND, RANDH, RANDI

sr = 44100
kr =  4410
ksmps = 10
nchnls = 1


instr 16       ;       uses RAND

;p4  : amplitude
;p5  : not used
;p6  : attack
;p7  : decay


k1      linen   p4, p6, p3, p7  ; envelope
    a1      rand    k1              ; noise source
            out     a1              ; output
endin



instr 17       ;       uses RANDH

;p4  : amplitude
;p5  : random oscillator frequency
;p6  : attack
;p7  : decay


k1      linen   p4, p6, p3, p7  ; envelope
        a1      randh   k1,p5           ; noise source
                out     a1              ; output
```

```
endin
instr 18       ;       uses RANDI

;p4  : amplitude
;p5  : random oscillator frequency
;p6  : attack
;p7  : decay


k1      linen   p4, p6, p3, p7  ; envelope
        a1      randi   k1,p5           ; noise source
                out     a1              ; output
endin
```

The score produces :

1.2 seconds of 'white noise'.

2.2 seconds of noise produced with RANDH at a frequency of 400 hz.

3.2 seconds of noise produced with RANDI at the same frequency of 400 hz.

The results can be heard in *Csound* tape example 13. The second sound is perceived as being lower than the first, and the third is heard as being even lower.

```
;CS13.SC        Noise Generators

;               p3      p4      p5      p6      p7
;instr  start   dur     amp     freq    attack  decay

;       RAND
i16     0       2       10000   0       .1      .1
```

```
;       RANDH   400 hz
i17     3       2       10000   400     .1      .1


;       RANDI   400 hz
i18     6       2       10000   400     .1      .1

e
```

7.1.2 Pulse Generators

An ideal pulse is a signal containing an infinite number of partials. In practice, the most direct way of obtaining an approximation of a pulse is by having as many partials as possible, all with the same relative amplitude. However care must be taken: if a large number of partials is taken indiscriminately, there is the danger of exceeding the Nyquist frequency, which means aliasing !

Csound provides a generator that produces pulses - with as many partials as required - which repeat periodically. The actual sonorous result is similar to a buzz. For this reason, the statement is called ... BUZZ. Its general form is

ar buzz amp, freq, nharm, func (, phase)

where

ar is the output.

amp is the overall amplitude.

freq is the frequency of the fundamental.

nharm is the required number of harmonics. In order to avoid exceeding the Nyquist frequency the number of harmonics must be less than the number of times the fundamental is contained in the Nyquist frequency. Thus, the following formula can be used to obtain the maximum possible partials

nharm = int((sr/2) / fundamental)

where sr, is the variable used by *Csound* to store the sampling rate. int() is used because it is necessary to round down the result.

func is a function table containing a sine wave.

phase is the optional phase of the signal.

CS14.ORC contains an instrument that uses BUZZ. The score CS14.SC produces pulses that have a fundamental of 40 hz and 5, 15 and 45 partials respectively. The cycles so produced can be seen in diagram 24.

```
; CS14.ORC      -       Use of BUZZ
sr = 44100
kr =  4410
ksmps = 10
nchnls = 1


instr 19

;p4  : amplitude
;p5  : fundamental
;p6  : attck
;p7  : decay
;p8  : number of partials

k1      linen   p4, p6, p3, p7; envelope
        a1      buzz    k1, p5, p8, 1      ; pulse generator
                out     a1                 ; output
endin
```

```
; CS14.SC

f1 0 8192 10 1


;               p3   p4    p5   p6   p7p8
;instrstartduramp freq atckdecayNo. of
;                                 partials

i19     0         2    30000 40   .1   .1  5

i19      3        2    .     40   .1   .1        15

i19      6        2    .     40   .1   .1        45

e
```

Diagram 24 Shape of One Pulse Generated by BUZZ with a Fundamental of 40 hz.

24.1 Using 5 harmonics.

24.2 Using 15 harmonics.

24.3 Using 45 harmonics.

Csound tape example 14 contains the sounds produced.

BUZZ can be used in order to produce inharmonic spectra. In order to do this a fundamental that is below the auditory range (less than ~15 hz) should be used. This way, the missing fundamental will not be heard and the lowest audible frequency will not be at integer ratios with the other partials.

There is an alternative generator, called GBUZZ, that allows control over low frequency and high frequency partials. This is done by varying their relative amplitudes using a factor that multiplies each partial and increases or decreases with the harmonic number. It is also possible to start at any partial, not necessary from fundamental.

The general form of GBUZZ is

ar gbuzz amp, freq, npart, lowest, fact, func (,phase)

where

ar is the output.

amp is the overall amplitude.

freq is the frequency of the fundamental.

lowest is the lowest partial relative to the fundamental that should be produced. Compare this with BUZZ, in which the partials will always start with the fundamental.

npart is the required number of partials starting from *lowest*. For example if a fundamental of 100 hz is specified and the pulse is to contain 10 partials starting from the fifth (500 hz), then

freq = 100
lowest = 5
npart = 10

The actual components of the pulse will be 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, and 1400 hz.

This time, in order to avoid exceeding the Nyquist frequency, the number of the highest partial must be less than the number of times the fundamental is contained in the Nyquist frequency. Thus, the following formula can be used to obtain the maximum possible partials

nharm = int((sr/2)/freq) - lowest + 1

fact is the factor that multiplies the relative amplitude of each partial according to its harmonic number.

The amplitude of the lowest partial is left as it is.

The amplitude of the next partial is multiplied by *fact*. The amplitude of the following partial is multiplied by $fact^2$. The next one is multiplied by $fact^3$, and so on.

If *fact* is bigger than 1, the higher harmonics will be amplified. If, instead, it is smaller than 1, they will be attenuated.

fact can be made to vary in time, by assigning it to a control rate variable.

func is the function table. It has to be a cosine wave.

phase is the optional phase of the signal.

For example, if the fundamental is 20 hz, the beginning partial is the fifth (5 x 20 = 100 hz), a total of 4 partials is requested and the factor is .5, the relative amplitudes of the partials will be

Partial	Frequency	Relative Amplitude
fifth	100 hz	1
sixth	120 hz	0.5
seventh	140 hz	$0.5^2 = .25$
eight	160 hz	$0.5^3 = .125$

CS15.ORC contains an instrument that use GBUZZ with partials
specified by the score and an amplitude factor that progresses linearly from a value at the beginning of the event to a different value at the end.

The score produces a sound with a fundamental of 30 hz, a lowest partial of 150 hz, a total of 30 partials and a factor that changes from 0.7 to 1.7.

```
; CS15.ORC      -       Use of GBUZZ
sr = 44100
kr =  4410
ksmps = 10
nchnls = 1

instr 20
;p4  : amplitude
;p5  : fundamental
;p6  : attack
;p7  : decay
;p8  : beginning value of amplitude multiplier
;p9  : final value of amplitude multiplier
;p10 : lowest partial number
;p11 : total number of partials

        k2   line   p8,p3,p9                    ; changing multiplier
        k1   linen  p4,p6,p3,p7                 ; envelope
        a1   gbuzz  k1,p5, p11,p10,k2,1         ; pulse generator
        out    a1                               ; output
endin

;CS15.SC
f1 0 512 9 1 1 90; cosine wave
;           p3  p4    p5    p6   p7     p8   p9   p10  p11

;instr start dur amp   freq  atck decay  beg  end  low  numb
;                                          mult mult  part part
i20    0     4   25000 30    .1      .5        .7    1.7    5    30
e
```

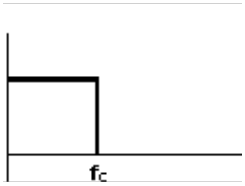
The result can be heard in *Csound* tape example 15.

7.2 Shaping of the Spectrum with Filters

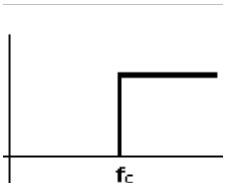
The four basic types of ideal filters are shown in diagram 25. Apart from these there is the *all-pass* filter, which only delays a signal, changing its phase. The *all-pass* will not be discussed here.

Diagram 25Basic Types of Filters

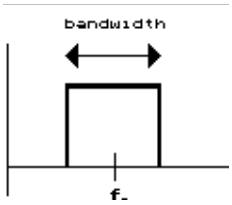
1.Low-Pass



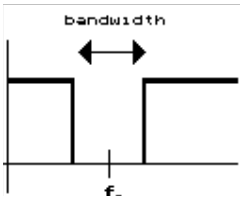
2.High-Pass



2.Band-Pass



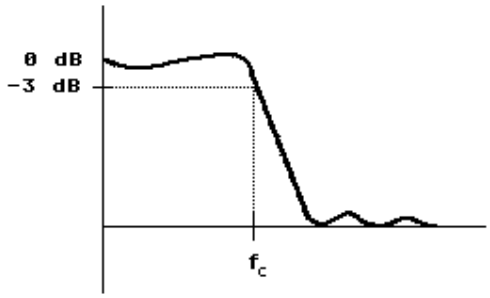
3.Band-Stop



All these filters have a *cut-off* value $\mathbf{f_c}$ that determines the limit for frequencies which will be passed or stopped. However, the figures above, are only ideal approximations. Real filters have *ripples* and a *transition region*, as shown in diagram 26. Therefore, for a real filter, the cut-off is defined as the frequency at which a signal is attenuated by -3 dB.

In addition, the transition region is characterized by a parameter called *roll-off*, that measures its steepness. The roll-off is defined as the change in the intensity of a signal when its frequency is doubled. In other words, it measures the change of the frequency response in dB per octave .

Diagram 26 Cut-off Frequency for a Real Life Filter

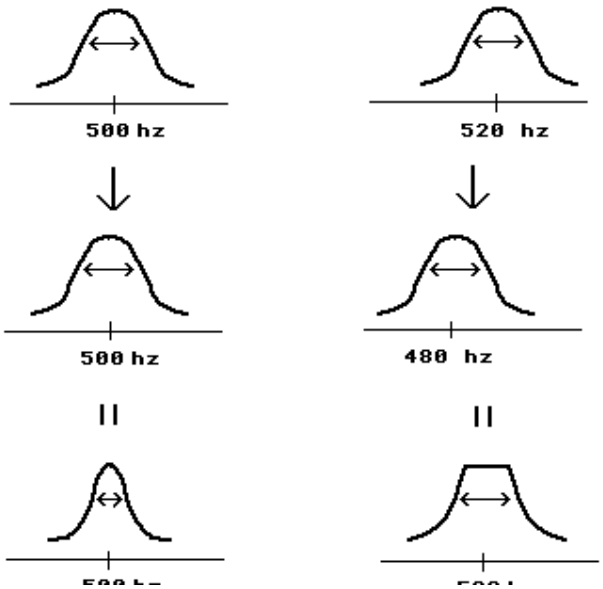


The sharpness of a filter can be increased by narrowing the transition region (which is equivalent to increasing the value of the roll-off). This is achieved by connecting it with an identical copy of itself. Doing this usually narrows the bandwidth, which is not always desirable.

In order to avoid a narrower bandwidth, the filters can have their cut-off frequencies slightly shifted above and below the actual cut-off.

Diagram 27 shows qualitatively the result of connecting two band-pass filters with centres at 500 hz, and compares it with the connection of two filters centred at 480 hz and 520 hz respectively. The bandwidth in the second case is better preserved. As an additional benefit, flatter pass and attenuation regions are produced.

Diagram 27 Connected Band-Pass Filters



Same Bandwidth Shifted Bandwidths

We will now proceed to discuss the implementation of the four basic filters in *Csound*.

7.2.1 Low-Pass and High-Pass: TONE and ATONE

Csound provides a *first order recursive* filter. A filter is called recursive when the value of its current output depends on previous outputs. It is of the first order when it only depends on its immediate previous value. Therefore, if x_n is the nth sample of the input, then y_n is the nth sample of the output and y_{n-1} is the immediate previous sample of that same output. The mathematical expression for the first order recursive can then be written as follows

$$y_n = c_0 x_n + d_1 y_{n-1}$$

The cut-off frequency and whether the filter is a low-pass or a high-pass depend on the coefficients $\mathbf{c_0}$ and $\mathbf{d_1}$. However, this should not trouble the *Csound* user, since there are two different statements, each representing a different type, which require the value of the cut-off frequency, rather than those of the coefficients. The statements representing those filters are:

TONE for a low-pass.

ATONE for a high-pass.

The general statements are :

ar tone ain, cutoff (, init)

ar atone ain, cutoff (, init)

where

ar is the output.

ain is the input.

cutoff is the cut-off frequency in hz.

init describes the initial conditions of the filter: it is always necessary to state the initial value of the output, because the current output is dependent on its immediate previous value. Therefore, the first output has to be determined in order to start this process.

If init is 0, the initial value is 0. This is also the default value.

If a non-zero value is given, the initial output recalls the value obtained the last time the filter was used .

7.2.2 Band-Pass and Band-Stop: RESON and ARESON

These are *second order* filters (they depend on the previous two values of either their input or their output). The band-pass filter is recursive - its current output depends on the two previous outputs - and is represented by the following expression:

$$y_n = c_0x_n + d_1y_{n-1} + d_2y_{n-2}$$

The band-stop is actually non-recursive: it only depends on its two previous inputs. Its representation is given below.

$$y_n = c_0x_n + c_1x_{n-1} + c_2x_{n-2}$$

The coefficients $\mathbf{c}_0$, $\mathbf{d}_1$and $\mathbf{d}_2$ and $\mathbf{c}_0$, $\mathbf{c}_1$and $\mathbf{c}_2$ determine the centre frequency and bandwidth of the band-pass and band-stop, respectively. This, again, does not have to worry the user since the statements in *Csound* do the calculations themselves. The statements are

RESON for a band-pass.

ARESON for a band-stop.

Their general form is

ar reson ain, cfreq, bw, scaling (,init)

and

ar areson ain, cfreq, bw, scaling (,init)

where

ar is the output.

ain is the input.

cfreq is the centre frequency in hz.

bw is the bandwidth.

scaling is a scaling factor. When a signal is filtered, the components that are attenuated do not contribute to the average intensity. This may result in an output with a much lower loudness. For this reason, it is sometimes necessary to amplify the output.

If the scaling is 0, the signal is not amplified.

If the scaling is 1, the pass-band frequencies are amplified to the maximum amplitude of the signal.

If the scaling is 2, the average intensity is kept. It is not clear from the manual what will higher scaling values do. Experience shows that there is some amplification.

NOTES:

1. The scaling factor changes the steepness of the transition between pass and stop bands.

2. If a large scaling is used, the amplitude may become larger than 32,767 or smaller than -32,768 causing distortion.

init describes the initial conditions of the filter, and acts in the same way as with TONE and ATONE.

If init is 0, the initial values are 0. This is also the default.

If a non-zero value is given, the initial outputs (RESON) or inputs (ARESON) recall values obtained the last time the filter was used.

7.3 Output: BALANCE

As stated above, when a signal is filtered, the components that are attenuated do not contribute to the average intensity, resulting in a lower level signal.

Apart from using a scaling factor of 1 or 2 within the filter, there is another method used in order to amplify the output: the average intensity of the output is made equal to the average intensity of the input or of another signal used for this purpose. This technique is usually safer for avoiding amplitudes larger than 32,767.

In *Csound*, a special filter called BALANCE is used:

ar balance ain, acomp, (, lowcutoff, init)
where

ar is the output.

ain is the signal that has to be balanced.

acomp is the signal to which it is compared.

lowcutoff is the cut-off frequency of an internal low-pass filter built into BALANCE. The purpose of this is to stop frequencies below the auditory range that do not contribute to the intensity of the perceived sound. If no value is given, the default is 10 hz.

init determines the initial conditions of BALANCE in the same way as with TONE and RESON.

A typical example of the use of a filter and BALANCE is now given. A signal **ain** is sent through a high-pass filter of 100 hz cut-off. Then the output of the filter is balanced with the original signal.

```
aout    atone       ain, 100
aout    balance     aout, ain
```

7.4 Example -Filtered Noise

The principle behind subtractive synthesis is the filtering of sounds with rich spectrum. It is highly desirable that the output has a dynamic spectrum. Therefore, it is necessary that at least the sound source or the filter (or both) change their spectral characteristics as time goes by.

Accurate control of filter characteristics is possible when using a computer. For example, the cut-off frequency of a low-pass filter can be driven by a control signal that changes in time. Combinations of several filters can be used to shape a signal to a tighter degree. However, it is worth bearing in mind that increasing quantity and complexity increases the number of operations a computer has to perform resulting in slower processing.

The following example, instrument 21, uses white noise as a source which is passed through a band-pass filter with centre frequency and bandwidth controlled respectively by the time varying control signals **k2** and **k3**.

k2 varies, according to the shape of a linear oscillator, between the centre frequencies **p7** and **p8**. **k3** varies, according to the shape of an exponential oscillator, with bandwidths that are between **p9**/100 of the minimum centre frequency and **p10**/100 of the maximum centre frequency. A graphic representation of this instrument is given in diagram 28.

The orchestra, CS16 is given below, followed by a score which consists of two events:

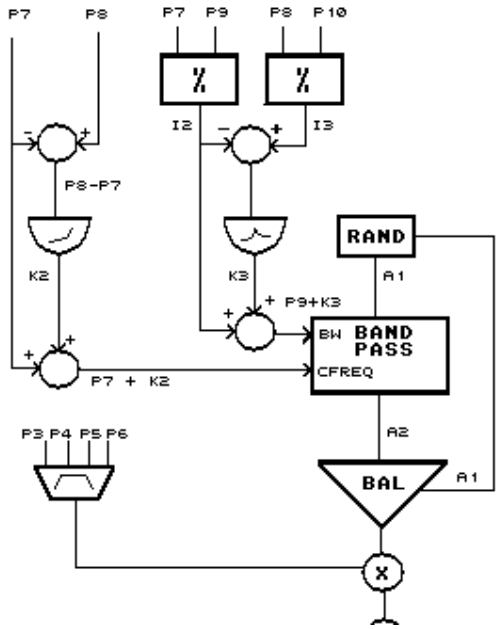
1. Simulation of a blow by increasing the centre frequency from 260 to 650 hz and the bandwidth from 0.1% to 20% of the centre frequency.

2. The reverse of 1: The centre frequency decreases from 650 to 260 hz and the

bandwidth from 20% to .1 %.

Csound tape example 16 contains the two events produced.

Diagram 28 Instrument 21: White Noise Shaped with a Variable Centre Frequency and Variable Bandwidth filter.




```
; CS16.ORC      -       FILTERED NOISE
;               White noise is passed through a band-pass filter
;               with varying centre frequency and bandwidth.


sr = 44100
kr =  4410
ksmps = 10
nchnls = 1

instr 21

;p4  : amplitude
```

```
;p5  : attack
;p6  : decay
;p7  : minimum centre frequency
;p8  : maximum centre frequency
;p9  : minimum bandwidth in %
;p10 : maximum bandwidth in %

i1=       1/p3; one cycle
    i2   =       p9*p7/100            ; minimum bandwidth
    i3   =       p10*p8/100           ; maximum bandwidth
    k1   linen   p4, p5, p3, p6       ; envelope
    k2   oscil   p8-p7, i1, 1         ; varying centre freq
    k3   oscil   i3-i2, i1, 2         ; varying bandwidth
    a1   rand    1                    ; random numbers
    a2   reson   a1, p7+k2, i2+k3, 4  ; filter
    a2   balance a2,a1                ; power balance
         out     k1*a2                ; output
endin

;CS16.SC
;       FROM PITCH TO NOISE SLIDING UP

f1 0 1024 7 0 512 .3 512 1; controls centre frequency
f2 0 1024 5 .001 512 1 512 .4; controls bandwidth

;           p3   p4      p5     p6    p7    p8     p9    p10
;instr start dur  amp     attack dec     centre     Bandwidth
;                                       frequency      in %
;                                       min   max    min   max

i21    0     2    10000   .5     .7    260   650    .1    20
i21    2.9   .1      0   .      .     .     .      .     .

s
```

```
;       FROM NOISE TO PITCH SLIDING DOWN

f1 0 1024 7 1 512 .3 512 0; controls centre frequency
f2 0 1024 5 .4 512 1 512 .001; controls bandwidth

;           p3   p4      p5     p6    p7    p8     p9    p10
;instr start dur  amp     attack dec     centre     Bandwidth
;                          frequency      in %
;                          min   max    min   max

i21    0     2    10000   .7     .5    260   650    .1    20

e
```

7.5 **Formants**

It has been found that one of the major factors influencing timbre recognition in traditional music is the enhancement of frequencies in certain regions of the audible spectrum that particular types of instruments show. The enhanced regions are called *formants*.

This assumption is strengthened by the fact that there are significant differences in waveform characteristics when different pitches produced with the same instrument are examined.

Formants are due to the physical characteristics of an instrument and are independent of the pitch produced.

For example, an oboe has its first and second formants at 1400 and 3000 hz respectively. This means that if a 440 hz A pitch is produced, the harmonics falling in the 1400 and 3000 hz regions will be strongly emphasised as shown by the underlined values given below.

440 880 <u>1320</u> 1760 2200 2640 <u>3080</u> 3520

Formant theory also applies to the most versatile of all instruments: the human voice. Speech production theories describe human voice articulation in terms of pulses or bands of noise produced by air passing through the vocal chords - pulses for voiced sounds and noise for unvoiced sounds like P, T, K and other consonants. These sounds are then filtered by the shape of the vocal tract, which changes dynamically according to the position of the tongue, lips, jaws, etc, thus changing the formant regions. This principle is used to produce very articulated sounds with instrument 22: A very rich pulse (maximum possible harmonics)is passed through five band-pass filters in parallel with centre frequencies in different regions of the audible spectrum: 500, 1000, 2000, 3300 and 4800 hz. These frequencies and bandwidths are made fluctuate by random number interpolators (RANDI).

To enhance the articulation of the sounds the following steps are taken:

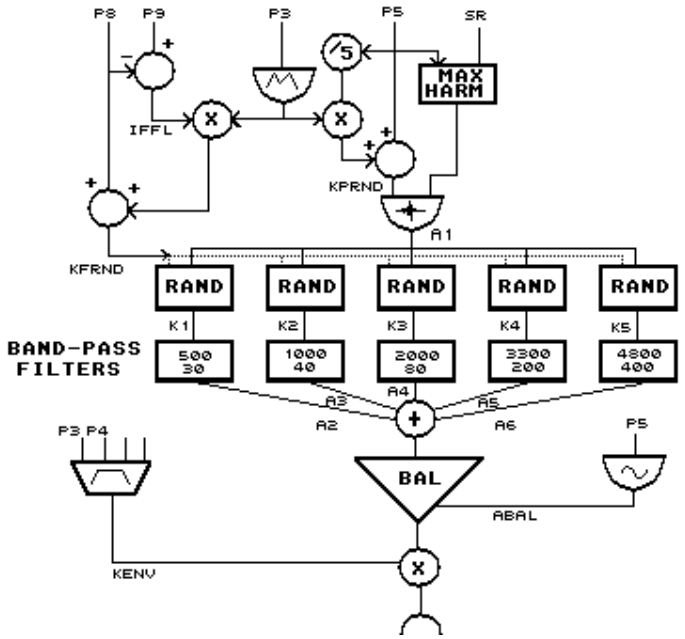
1. The frequency of the random number generators is varied by a function that makes it fluctuate between a minimum and a maximum value given in the score by p8 and p9. The control variable determining this frequency is **kfrnd**. In this particular score, the frequency fluctuates between 3 and 50 hz.

2. The fundamental frequency of the pulse generator is made to fluctuate by up to 1/5 of its value using the same function. The change in pitch provides for expressivity in a similar way to that in which the human voice is articulated. The control variable determining the frequency of the fundamental is **kprnd**.

The output of the filters is balanced with a sine wave especially generated for this purpose. The reason for not using the pulse generator to balance it is that a pulse is an extreme type of signal with most of its energy bursting very quickly followed by almost silence until the next pulse appears. Therefore if it were used for balancing purposes, the output would be distorted.

The graphical representation of this instrument 22 is shown below.

Diagram 29 Voice-like Instrument 22



The score contains a short and very articulated sound, using a fundamental of 70 hz and a random number frequency that varies between 10 and 15 hz, followed by two other events producing distorted versions by changing the fundamental, the duration and random number frequency more drastically. Both sounds were used towards the end of the piece *Dreams of Being* and can be heard in in *Csound* tape example 17.

The orchestra and the score are given next.

```
; CS17.ORC      -       FILTERED PULSE WITH FORMANTS
;               A pulse is passed through 5 parallel bandpass
;               filterswith varying centre frequency and
;         bandwidth.

sr = 44100
kr =  4410
ksmps = 10
nchnls = 1

instr 22
```

```
;p4  : amplitude
;p5  : fundamental
;p6  : attack
;p7  : decay
;p8  : minimum frequency of RANDI
;p9  : maximum frequency of RANDI

ip3     =       1/p3            ; one cycle
    ipfl    =       p5/5            ; pitch fluctuation
    iffl    =       p9-p8           ; freq fluctuation of RANDI

    inh     =       int(sr/2/(p5+ipfl)); maximum harmonics
    kenv    linen   p4,p6,p3,p7     ; envelope

    krand   oscil   .5, ip3, 2      ; oscil between -.5 and .5
    krand   =       krand + .5      ; correct between 0 and 1

    kfrnd   =       p8+iffl*krand   ; actual frequency of RANDI
    kprnd   =       p5+ipfl*krand   ; variable pitch

    a1      buzz    1,kprnd,inh,1   ; pulse
    abal    oscil   1,p5,1          ; balancing sine wave

;random number generator for each filter

    k1      randi   1,kfrnd,.12     ; random generator
    k2      randi   1,kfrnd,.23     ; for each filter
    k3      randi   1,kfrnd,.34     ; each with a
    k4      randi   1,kfrnd,.45     ; different seed
    k5      randi   1,kfrnd,.56     ;

;formant filters
```

```
    a2 reson a1, 500+k1*100, 30*(1+k1), 0   ; 500 hz bandpass
    a3 reson a1, 1000+k2*200, 40*(1+k2), 0  ; 1000 hz bandpass
    a4 reson a1, 2000+k3*300, 80*(1+k3), 0  ; 2000 hz bandpass
    a5 reson a1, 3500+k4*500, 200*(1+k4), 0 ; 3500 hz bandpass
    a6 reson a1, 4800+k5*800, 400*(1+k5), 0 ; 4800 hz bandpass

;mix outputs

    a7      =       a2+a3+a4+a5+a6

    a7      balance a7,abal         ; balance intensity

            out kenv*a7                     ; output
endin



;CS17.SC

f1 0 4096 10 1
f2 0 512 7 0 50 .6 50 .8 50 .3 50 .7 50 .9 50 .4
50 .1 50 .01 62 .2

;               p3   p4      p5     p6    p7    p8      p9
;instr  start  dur  amp     fund.  atck dec   random freqs.
;                        freq              minmax

i22     0       .5  5000     70     .1    .1    10      15

i22     1       5    2500    490    4      .5    40      45
i22     1       5    2500    615    4      .5    40      45
i22     3.5     7    2500    60      .5   1.5     3      50

e
```

7.6 Example - Extension of Risset's Beating Harmonics Principle

In the previous chapter, Risset's beating harmonics were shown as an example of ingenuity using additive synthesis. However, once the basic sound was achieved, there was little else that could be done to modify it. In fact, the frequencies of the components had to be carefully chosen in such a way that very slow envelopes were produced. Also, the results were all more or less harmonics of a fundamental drone.

This section will deal with an extension of this principle using subtractive synthesis techniques to produce an instrument that affords great flexibility. The main advantages of this extended technique are now listed:

1. The beating partials do not have to be harmonics. Any reasonable frequency values can be ascribed to the beating partials.

2. The frequencies of the beating partials can change in time.

3. The envelopes can have any frequency.

4. The envelopes can have any shape.

5. It is possible to filter any sound. Thus, advantage can be taken of gesture and morphology of concrete sound sampled into the system.

The first step in order to arrive at this desired instrument will be to explain and emulate the beating harmonics of the previous chapter. In it, nine frequencies were chosen as follows:

	110.12	=	110 + 0.12	=	110 + 4 x 0.03
	110.09	=	110 + 0.09	=	110 + 3 x 0.03
	110.06	=	110 + 0.06	=	110 + 2 x 0.03
	110.03	=	110 + 0.03	=	110 + 1 x 0.03
110.00 hz					
	109.97	=	110 - 0.03	=	110 - 1 x 0.03
	109.94	=	110 - 0.06	=	110 - 2 x 0.03
	109.91	=	110 - 0.09	=	110 - 3 x 0.03
	109.88	=	110 - 0.12	=	110 - 4 x 0.03

It can be seen that 4 frequencies were chosen above and 4 below a central value of 110 hz. Instead of 110 hz a general value **f** could be chosen, with **n** frequencies above it and **n** frequencies below: a total **2n+1** frequencies. A general increment represented by the letter **d** could replace the value of 0.03 hz. The frequencies used can now be represented as follows:

f + n x d
.
.
f + 2 x d
f + 1 x d
f
f - 1 x d
f - 2 x d
.
.
f - n x d

If sine waves are assumed, then the output could be written as

$$
\begin{aligned}
m(t) = {} & \sin[2\pi ft] + \\
& \sin[2\pi (f+d)t] + \sin[2\pi (f-d)t] + \\
& \sin[2\pi (f+2d)t] + \sin[2\pi (f-2d)t] + \\
& . \\
& . \\
& \sin[2\pi (f+nd)t] + \sin[2\pi (f-nd)t]
\end{aligned}
$$

but it may be remembered that

$$
\sin(a) + \sin(b) = 2\left[\cos\left(\frac{a-b}{2}\right)\sin\left(\frac{a+b}{2}\right)\right]
$$

therefore

$$
\sin[2\pi (f+nd)t] + \sin[2\pi (f-nd)t] = 2\cos[2\pi ft]\sin[2\pi (nd)t]
$$

and

$$
\begin{aligned}
m(t) = {} & \sin[2\pi ft] + \\
& 2\sin[2\pi ft]\cos[2\pi dt] + \\
& 2\sin[2\pi ft]\cos[2\pi (2d)t] + \\
& . \\
& . \\
& 2\sin[2\pi ft]\cos[2\pi (nd)t]
\end{aligned}
$$

$$
\begin{aligned}
= {} & \sin[2\pi ft]\{1 + 2\cos[2\pi dt] \\
& + 2\cos[2\pi (2d)t] \\
& . \\
& . \\
& + 2\cos[2\pi (nd)t]\}
\end{aligned}
$$

$$
m(t) = \sin[2\pi ft]\left\{1 + 2\sum_{i=1}^{n}\cos[2\pi (id)t]\right\} \qquad (7.1)
$$

Expression (7.1) can be interpreted as a sine wave of frequency **f** with an envelope that is the result of a sum of cosines of frequencies **d**, **2d**, **3d** ... **nd**. In the previous example, the value of **d** was 0.03 hz. The envelope in the curly brackets is just a cosine oscillator of fundamental frequency **d**, with a DC component (frequency = 0hz) and **n** harmonics and can be easily implemented using GEN9.

f1 0 512 9 0 .5 01 1 901 1 901 1 901 1 90 . . .

Now, if complex signals are used, then the corresponding harmonics will generate similar waveforms, the difference being in the values of the frequency **f** and increment **d**. For example, if we take the **kth** harmonic, its frequency will be **kf** and the increment will be **kd**. We can then write

$$m_k(t) = \sin[2\pi(kf)t] \{ 1 + 2 \sum_{i=1}^{n} \cos[2\pi(ikd)t] \} \qquad (7.2)$$

The envelope in the curly brackets still has the same shape as above, with a different fundamental frequency.

In order to generate any harmonic, a very narrow band-pass filter can be imposed on a rich pulse. Then the previous envelope can be imposed. The implementation is shown in CS18.ORC and CS18.SC. *Csound* tape example 18 contains a pulse followed by the resulting beating harmonics after it is filtered.

```
; CS18.ORC
; Filters the output of BUZZ and gives it an amplitude envelope
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
```

```
instr 23

;p4 : amplitude factor
;p5 : fundamental
;p6 : number of partials
;p7 : frequency of envelope
;p8 : envelope function
;p9 : centre frequency of band-pass filter
;p10: bandwidth as a percentage of centre frequency

        ibw     =       p8*p7/100       ; % bandwidth to hz
        kenv    oscil   p4, p7, p8      ; envelope
        ain     buzz    1, p5, p6, 2    ; pulse generator
        afilt   reson   ain, p9, p10, 1 ; filter
                out     kenv*afilt; output

endin



; CS18.SC
;       It uses f1 as an envelope

f1 0 1024 9 0.001 1 0 1 2 90 2 2 90 3 2 90 4 2 90 ; harmonics env
f2 0 8192 10 1; sine for BUZZ

;           p3   p4      p5    p6     p7   p8   p9      p10
;inst  start dur  ampl    fund  No of  envl envl centre  bw in
;               factor  freq  parts  freq func freq    % of
;                                                 centre

i23    0     35   .4      20    250    .03  1     100     1
i23    .     .    .       .     .      .06  .     200     .
i23    .     .    .       .     .      .09  .     300     .
i23    .     .    .       .     .      .12  .     400     .
```

```
i23  .  .  .  .  .  .15  .  500  .
i23  .  .  .  .  .  .18  .  600  .
i23  .  .  .  .  .  .21  .  700  .
i23  .  .  .  .  .  .24  .  800  .
i23  .  .  .  .  .  .27  .  900  .
i23  .  .  .  .  .  .30  .  1000  .

e
```

There is no reason to limit the source to a pulse generator. In fact, it is possible to process an existing soundfile by reading it into an instrument. In order to do this two things are necessary:

1. The soundfile has to be renamed to *soundin.nnn*, where *nnn* is a number between 1 and 999. *soundin.1*, *soundin.33* and *soundin.763* are examples of possible names.

2. The instrument has to include the *Csound* statement SOUNDIN. The general form of SOUNDIN is

soundin nnn, skip

where

nnn is the number assigned to the soundfile *soundin.nnn*.

skip is the number of beats that are to be skipped from the beginning of the soundfile.

Now it is possible to produce an instrument that has all the features mentioned at the beginning of this section in addition to variable bandwidth of the filters. This is instrument 24, implemented in CS19.ORC.

This orchestra was run with CS19.SC in order to produce one of the basic sounds used in the composition of *Los Dados Eternos*. This time the beating partials, at 100, 200, 350, 550, 800, 1100, 1450, 1850, 2300, 2800, 3350 and 3950 hz were clearly not harmonic. The envelope repetition frequencies, from 1 to 12 hz were two orders of magnitude faster than those in the original Risset sound. The input, *soundin.1* was reminiscent of a low cry and the result kept the gestural characteristics of the original. These two were subsequently spliced together. *Csound* tape example 19 contains the original sound, followed first by the processed result and then by their splice.

```
; CS19.ORC
; Filters soundfile SOUNDIN.NNN and gives it an envelope.
; The bandwidth of the filter fluctuates according to function 2

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

instr 24

;p4 : skip
;p5 : frequency of envelope
;p6 : envelope function
;p7 : centre frequency of band-pass filter
;p8 : minimum bandwidth in %
;p9 : maximum bandwidth in %
;p10: scaling of filter
;p11: input file

    icyc    =       1/p3                ; one cycle
    iminbw  =       p8*p7/100           ; min % bw to hz
    imaxbw  =       p9*p7/100           ; max % bw to hz
    kenv    oscil   p10, p5, p6         ; envelope
    kbw     oscil   imaxbw-iminbw, icyc, 2  ; bw fluctuation
```

```
    ain     soundin p11, p4                 ; input soundfile
    afilt   reson   ain, p7, iminbw+kbw, 1  ; filter
            out     kenv*afilt; output

endin



; CS19.SC
; This inputs the soundfile SOUNDIN.1 and filters it.
; It uses f1 as an envelope and f2 as bandwidth fluctuation

f1 0 1024 9 0.001 1 0 1 2 90 2 2 90 3 2 90 4 2 90
f2 0 1024 5 1 700 0.0001 324 0.0001

;          p3    p4   p5     p6     p7     p8  p9  p10     p11
;inst start dur   skip envel  envel  ctre  bw in %  scaling input
;                  freq   func   freq  min  max factor  sfile

i24   0     4.47  0     1     1       100  .5   100 5       4
i24   .     .     .     2     .       200  .    .   .       .
i24   .     .     .     3     .       350  .    .   .       .
i24   .     .     .     4     .       550  .    .   .       .
i24   .     .     .     5     .       800  .    .   .       .
i24   .     .     .     6     .      1100  .    .   .       .
i24   .     .     .     7     .      1450  .    .   .       .
i24   .     .     .     8     .      1850  .    .   .       .
i24   .     .     .     9     .      2300  .    .   .       .
i24   .     .     .    10     .      2800  .    .   .       .
i24   .     .     .    11     .      3350  .    .   .       .
i24   .     .     .    12     .      3950  .    .   .       .

e
```

PART III

Non-Linear Synthesis Techniques

8. Amplitude Modulation (AM)

Amplitude modulation includes all the techniques that produce changes in the spectrum of a sound by controlling its amplitude. The effectiveness of these techniques is due to the fact that they distort the input signal, changing its shape in such a way that new frequency components that were not there in the first place are created: when the shape of a waveform is altered, its spectrum changes.

This is different from filtering because of the fact that digital filters are *linear*: they do not distort the signal and, as a consequence, they do not create new frequencies that were not previously contained in the input. On the other hand, effective amplitude modulation must be essentially *non-linear*: it must produce distortion.

Linear procedures process samples in only three possible ways

1. By delaying them.

2. By scaling, which is equivalent to multiplying a sample by a constant.

3. By adding or subtracting scaled samples.

Non-linear processes can raise a sample to a power, or multiply it by different coefficients depending on its value, etc.

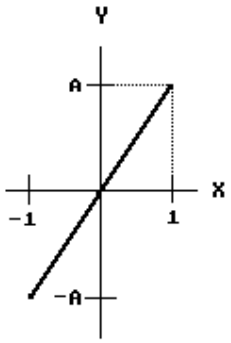
To illustrate the difference, imagine the following linear amplifier:

$y = Ax$

All this does is multiply each sample by the number **A**. If **x** varies between -1 and 1, the

values of **y** against those of **x** can be plotted to obtain the following diagram

Diagram 30 Linear amplifier y = Ax



Therefore, if a sampled sine wave of frequency **f** is passed through this amplifier, the input sine wave is represented by

$X = \sin(2 \pi ft)$

The output of the amplifier will be

$Y=A\sin(2 \pi ft)$

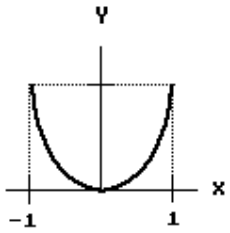
Thus, the output consists of a sine wave of exactly the same frequency as the input, scaled by a factor A.

If, instead, the sine wave is passed through a non-linear device represented by

$y = Ax^2$

then, plotting the values of **y** against **x** produces the following curve

Diagram 31 Non-linear amplifier $y = Ax^2$



Feeding the sine wave to this function will produce

$Y=A\,[\,\sin(\,2\,\pi\,ft\,)\,]^2$

But, from trigonometry :

$$\sin^2 x = \frac{1}{2} - \frac{1}{2}\cos 2x$$

thus

$$Y = \frac{A}{2} - \frac{A}{2}\cos(\,2(2\,\pi\,f)t\,)$$

$$= \frac{A}{2} - \frac{A}{2}\cos(\,2\,\pi\,(2f)t\,)$$

There are now two terms contributing to the components of the output:

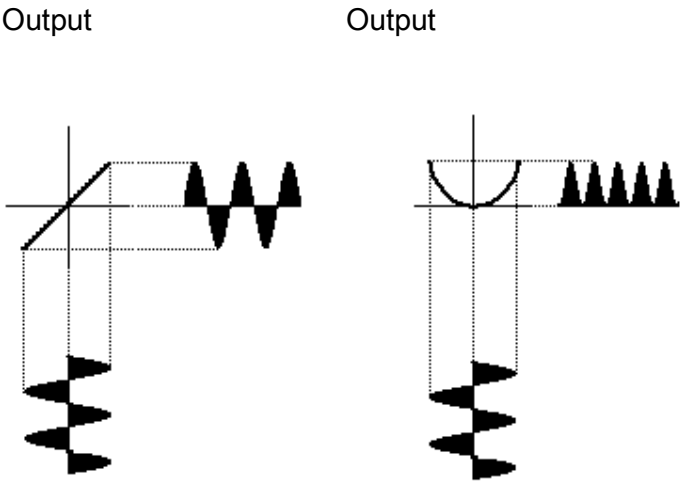
a constant which represents a component at 0 hz

a component at twice the frequency of the input.

Therefore, two components that did not exist in the input were created.

The respective outputs of the linear and the non-linear devices are shown next.

Diagram 32 Linear Non-Linear



These two devices are implemented in CS20.ORC. CS20.SC sends a 440 hz waveform first through a linear amplifier and then through the non-linear device. *Csound* tape example 20 contains the results: when the linear amplifier is used, the frequency is conserved, but in the case of the non-linear amplifier, the result is an octave higher (twice the frequency) as predicted. The DC component is not heard therefore it does not contribute to the perceived intensity. As a result, the non-linear output is not as loud.

```
; CS20.ORC      -       Linear and Non-linear amplifiers

sr = 44100
kr =  4410
ksmps = 10
nchnls = 1

instr 25        ;       linear  amplifier y = Ax

;p4  : amplitude
;p5  : frequency
;p6  : attack
;p7  : decay
;p8  : function
;p9  : amplitude factor A


kenv    linen   1,p6,p3,p7      ; envelope
    ain     oscil   1,p5,p8         ; oscillator

    aout    =       p4*ain          ; y = Ax
            out     kenv*aout       ; output
endin


instr 26        ;                               2
                ;       non-linear amplifier y = Ax

;p4  : amplitude factor A
;p5  : frequency
;p6  : attack
;p7  : decay
;p8  : function
```

```
    kenv    linen   1,p6,p3,p7      ; envelope
    ain     oscil   1,p5,p8         ; oscillator
                                ;       2
    aout    =       p4*ain*ain      ; y = Ax
            out     kenv*aout       ; output
endin


;CS20.SC

f1 0 8192 10 1

;               p3      p4      p5      p6      p7      p8
;instr  start   dur     amplif  freq    attack  decay   func
;                       factor

;linear amplifier
i25     0       2       25000   440     .3      .3      1

;non-linear amplifier
i26     3       2       25000   440     .3      .3      1

e
```

8.1 **The Simplest Amplitude Modulator**

One of the simplest ways to achieve non-linear amplitude modulation is by multiplying two sine waves.

Take two sine waves of frequencies $\mathbf{f_c}$ and $\mathbf{f_m}$. The former will be called the *carrier* and the latter the *modulator*. If the carrier is multiplied by the modulator, the output will be

$$o(t) = \sin(2\pi f_c t) \sin(2\pi f_m t)$$

but from trigonometry

$$\sin(a)\sin(b) = \frac{1}{2}[\cos(a+b) - \cos(a-b)]$$

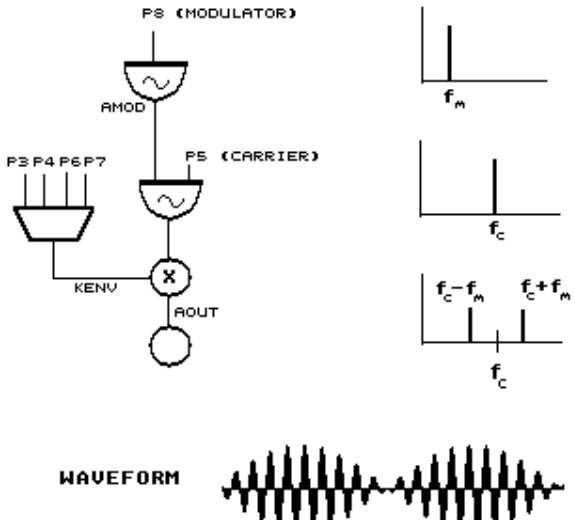
therefore

$$o(t) = \frac{1}{2}[\cos(2\pi(f_c+f_m)t) - \cos(2\pi(f_c-f_m)t)]$$

A look at this expression reveals that two components are created. One at the frequency $\mathbf{f_c + f_m}$ and one at $\mathbf{f_c - f_m}$. The amplitude of each component is half of the amplitude of the overall output. In other words, the resulting spectrum consists of two components, one of them $\mathbf{f_m}$ hz below the carrier and the other $\mathbf{f_m}$ hz above it. These components are called *sidebands*.

Diagram 33 shows an instrument that multiplies two waveforms, illustrating the changes in the spectrum as well as displaying snapshots of possible inputs and the output they produce.

Diagram 33 Side Band AM Modulator



Waveforms

Carrier

Modulator

Output

It is worth discussing the difference frequency $f_c - f_m$:

If the frequency of the carrier and the modulator are very close, $f_c - f_m$ will be a very small number. For values of less than 15 hz, it will not be heard, becoming an envelope.
If the frequency of the modulator is higher than the frequency of the carrier, $f_c - f_m$ will be negative. A negative frequency of a cosine wave reflects as a positive frequency of the same value and phase because of the identity

cos(a) = cos(-a)

For example, if the frequency of the carrier is 300 hz and the frequency of the modulator is 500 hz, the resulting frequencies will be

300 + 500 = 800 hz

and

300 - 500 = -200 --> 200 hz.

CS21.ORC contains *instr 27*, which is the realization of the instrument shown in Diagram 33. CS21.SC produces examples of a carrier of 300 hz and for different modulators of 15, 110, 310, and 500 hz. In each example, the carrier and the modulator are first played separately using *instr 3* followed by the modulation. The resulting frequencies are:

Carrier(hz)	Modulator(hz)	Components(hz)
300	15	315 and 285
300	110	410 and 190
300	310	610 and 10 (not heard)
300	500	800 and 200

The orchestra and score are listed below and the sounds can be heard in *Csound* tape example 21.

; CS21.ORC - Amplitude Modulator

```
sr = 44100
kr =  4410
ksmps = 10
nchnls = 1


instr 3         ; Used to play pure sinewaves

    k1      linen   p4, p6, p3, p7  ; envelope
    a1      oscil   k1, p5, p8      ; oscillator
            out     a1              ; output

endin



instr 27        ;       Side bands only

;p4  : amplitude
;p5  : carrier
;p6  : attack
;p7  : decay
;p8  : modulator
;p9  : function

    kenv    linen   p4, p6, p3, p7  ; envelope

    acarr   oscil   1, p5, p9       ; carrier
    amod    oscil   1, p8, p9       ; modulator

    aout    =       kenv*acarr*amod ; modulation
            out     aout            ; output

endin
```

```
;CS21.SC

f1 0 8192 10 1



;SECTION 1      carrier 300 hz, modulator 15 hz

;              p3      p4      p5      p6     p7    p8
;instr3 start   dur     amp     carr.   attack dec   func
 i3      0       2      25000   300     .3     .3    1
i3      3       2      25000    15     .3     .3    1

;              p3      p4      p5      p6     p7    p8   p9
;inst27 start   dur     amp     carr.   attack dec   mod. func.

i27     6       2      25000   300     .3    .3    15   1

s



;SECTION 2      carrier 300 hz, modulator 110 hz

;              p3      p4      p5      p6     p7    p8
;instr3 start   dur     amp     carr.   attack dec   func

i3      1       2      25000   300     .3     .3    1
i3      4       2      25000   110     .3     .3    1

;              p3      p4      p5      p6     p7    p8    p9
;inst27 start   dur     amp     carr.   attack dec   mod.  func.
```

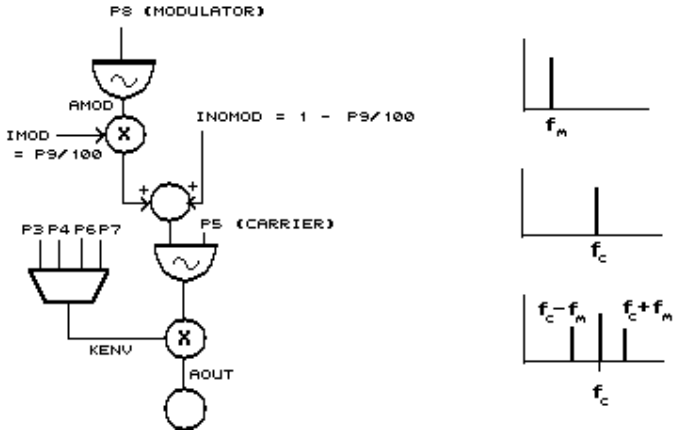
```
i27     7       2      25000   300     .3     .3     110  1

s


;SECTION 3      carrier 300 hz, modulator 310 hz

;               p3      p4      p5      p6     p7    p8
;instr3 start   dur     amp     carr.   attack dec   func

i3      1       2      25000   300     .3     .3    1
i3      4       2      25000   310     .3     .3    1

;               p3      p4      p5      p6     p7    p8    p9
;inst27 start   dur     amp     carr.   attack dec   mod.  func.

i27     7       2      25000   300     .3     .3    310   1

s

;SECTION 4      carrier 300 hz, modulator 500 hz

;               p3      p4      p5      p6     p7    p8
;instr3 start   dur     amp     carr.   attack dec   func

i3      1       2      25000   300     .3     .3    1
i3      4       2      25000   500     .3     .3    1

;               p3      p4      p5      p6     p7    p8    p9
;inst27 start   dur     amp     carr.   attack dec   mod.  func.

i27     7       2      25000   300     .3     .3    500   1

e
```

In side band modulation, the frequency of the carrier is lost. Sometimes it is desirable to keep some of it. In order to do this, the modulator multiplies only part of the carrier, as shown in diagram 34.

CS22.ORC contains an instrument that realizes this: In addition to the carrier and the modulator frequencies, it gets from the score the percentage of the waveform that is to be modulated.

CS22.SC produces 2 sounds. Both have a carrier of 600 hz and a modulator of 15 hz. In the first sound, 90% of the carrier is modulated. In the second only 10%. *Csound* tape example 22 contains the results. It may be noticed that the second sound is actually a sine wave with a tremolo. Therefore, tremolo is a special case of amplitude modulation, when only a small percentage of a signal is modulated at a frequency that is below the audible range. Small amounts of tremolo can add to the liveliness of a sound. Even better effects are achieved by using an interpolating random number generator (RANDI) at frequencies of arround 15 hz.

Diagram 34 Side Bands and Carrier.
Only part of carrier is modulated.



Waveforms

modulation of 10% of carrier

modulation of 90% of carrier

```
; CS22.ORC      -       Amplitude Modulator

sr = 44100
kr =  4410
ksmps = 10
nchnls = 1


instr 28        ;       Side bands and carrier

;p4  : amplitude
;p5  : carrier
;p6  : attack
;p7  : decay
;p8  : modulator
;p9  : percentage of carrier to be modulated
;p10 : generating function of carrier
;p11 : generating function of modulator

imod    =       p9/100.00               ; modulated part
    inomod  =     1 - imod              ; unmodulated part

    kenv    linen  p4, p6, p3, p7       ; envelope

    acarr   oscil  1, p5, p10           ; carrier
    amod    oscil  1, p8, p11           ; modulator
```

```
    aoutm   =      acarr*amod*imod       ; modulated signal
    aoutnm  =      acarr*inomod          ; unmodulated signal
           out    kenv*( aoutm+aoutnm ) ; output


endin
;CS22.SC

f1 0 8192 10 1

;            p3  p4      p5    p6     p7   p8  p9    p10  p11
;instr start dur amp     carr.  attack dec  mod mod%  carr mod.
;    func func

i28     0     2   25000   600   .3     .3   15  90    1    1

i28     3     .   .       .     .      .    .   10    1    1

e
```

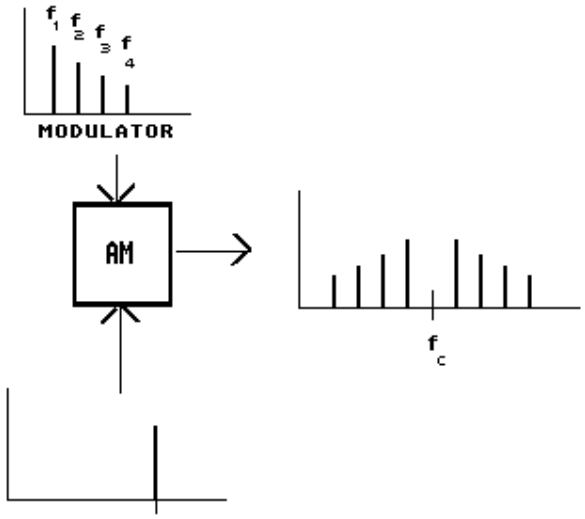
8.2 **Ring Modulation**

This technique is just a generalisation of the case of two sine waves that are multiplied. For example, if a carrier that is not a sine wave, but rather a waveform with component frequencies f_{c1}, f_{c2}, f_{c3}, etc. modulating it with a sine wave of frequency f_m will produce a pair of frequencies for each component of the carrier:

$f_{c1} + f_m$ and $f_{c1} - f_m$
$f_{c2} + f_m$ and $f_{c2} - f_m$
$f_{c3} + f_m$ and $f_{c3} - f_m$
.
.
etc.

This can be seen in diagram 35.

Diagram 35



As an example, if a modulator with components at 50, 100 and 150 hz, and a carrier of 500 hz are given. The resulting frequencies will be

50 hz modulating 500 hz: 450and550 hz
100 hz modulating 500 hz: 400and600 hz
150 hz modulating 500 hz: 350and650 hz

Instrument 26 can now be used with a modulator that is a complex waveform. CS23.ORC is identical to CS22.ORC and CS23.SC contains
a complex modulator defined by function 2. A carrier of 300 hz is completely modulated by

a modulator with a fundamental of 212 hz and components at 424, 636, 848 and 1060 hz.

The resulting frequencies are:

212 hz on 300 hz : 512 and 88.
424 hz on 300 hz : 724 and -124 --> 724 and 124 hz.
636 hz on 300 hz : 936 and -336 --> 936 and 336 hz.
848 hz on 300 hz : 1148 and -548 --> 1148 and 548 hz
1060 hz on 300 hz: 1360 and -860 --> 1360 and 860 hz

The result can be heard in *Csound* tape example 23. The score is now listed.

```
;CS23.SC

f1 0 512 10 1
f2 0 512 10 1 .8 .7 .6 .5

;          p3   p4     p5    p6    p7    p8   p9    p10   p11
;inst start dur  amp    carr  attack dec   mod  mod%  carr  mod
;                                          func  func

i28   0    2    25000   300   .3    .3    212  100   1     2

e
```

8.3 Carrier to Modulator Ratio

In the last example, it could be noticed that the resulting sound was not definitely pitched, more like a bell. The reason for this is that the the frequency of the carrier is not a multiple of the modulator, in other words, the ratio between the frequencies cannot be represented as a ratio of integers.

In fact, the ratio between the carrier and the modulator is a very important parameter when determining the quality of the resulting spectrum. Therefore in order to have an idea of

what the output is going to be, it is worthwhile knowing the effects of the carrier to modulator ratio on the spectrum.

Since the intention is just to evaluate qualitatively the results rather than to produce a rigorous description of all the possible cases, the following assumptions will be adopted:

1. The carrier is a pure sine wave of frequency $\mathbf{f_c}$.

2. The modulator has at least two partials (otherwise this reverts to the simple case of the product of two sine waves). Its fundamental is $\mathbf{f_m}$.

3. The partials are actual harmonics.

4. They are consecutive (there are no harmonics missing between the highest and the lowest component).

The following cases can then occur:

8.3.1 c/m = 1 , which means that $\mathbf{f_c = f_m}$

In this case, the fundamental of the output is equal to the frequency of the carrier (= modulator fundamental). Also, if the modulator has M harmonics, the output will have M+1.

To illustrate this, assume the simplest case, in which the carrier is a sine wave of frequency $\mathbf{f_c}$ and the modulator has two partials: $\mathbf{f_c}$ (= $\mathbf{f_m}$) and $\mathbf{2f_c}$, therefore M = 2. The resulting components will be:

$f_c + f_{m1}$	=	$f_c + f_c$	=	$2f_c$
$f_c + f_{m2}$	=	$f_c + 2f_c$	=	$3f_c$
$f_c - f_{m1}$	=	$f_c – f_c$	=	0
$f_c - f_{m2}$	=	$f_c - 2f_c$	=	$-f_c$ --> f_c

It can be seen that the output has 2 + 1 = 3 harmonics out which $\mathbf{f_c}$ is the fundamental.

8.3.2 c/m = 1/N , which means that $\mathbf{f_m = Nf_c}$

Here, the modulator is a multiple of the carrier. This is a generalization of the previous case. The fundamental will be

(N-1) x f_c

For example, if the carrier is a sine wave of 100 hz and the modulator consists of two partials at 300 and 600 hz, the modulator fundamental is 3 times the frequency of the carrier, therefore the fundamental will be

(3-1) x 100=200 hz

This can be corroborated by looking at the resulting components.
100 + 600 = 700 hz
100 + 300 = 400 hz
100 - 300 = -200 hz --> 200 hz
100 - 600 = -500 hz --> 500 hz

The lowest component is 200 hz, as expected.

It can be noticed that there is a gap between the fundamental and the next harmonic of the resulting sound. In fact, if the frequency of the modulator becomes higher, the gap between the fundamental and the harmonics will grow. In general, this can be associated with the production of brighter sounds.

8.3.3 c/m = N , which means that $\mathbf{f_c = Nf_m}$

Here, the carrier is a multiple of the modulator. Assuming that the modulator M harmonics, there are two possibilities:

1. If N is larger than M, or in other words, if the c/m ratio is larger than the highest harmonic, then the fundamental of the output will be

(N - M) x f

For example, if the carrier frequency is 1000 hz, and the modulator consists of the three components 200, 400 and 600 hz, the fundamental will be:

(5 - 3) x 200 = 2 x 200 = 400 hz

which is the difference between the carrier and the highest partial,namely 600 hz.

Summarizing: the new fundamental will be the difference between the carrier frequency and the highest component of the modulator.

2. If M is larger or equal than N, that is, if the highest harmonic is larger or equal than the c/m ratio, the fundamental of the resulting sound will be equal to the fundamental of the modulator.

   For example, if the carrier frequency is 400 hz, and the modulator consists of the three components 200, 400 and 600 hz, the fundamental will be 200 hz.

   This can be checked by finding all the frequencies produced.

   200 modulating 400 --> 200 and 600 hz
   400 modulating 400 --> 0 and 800 hz
   600 modulating 400 --> -200 and 1000 hz
   The component at 0 hz (DC) is not heard, thus the output is perceived as having a fundamental is 200 hz, as expected.

8.3.4 Non-Integer Ratio

In the three previous cases the carrier to modulator ratio could be expressed as the ratio between two integers. Next the case in which c/m cannot be expressed as a ratio of integers is discussed.

Before we proceed with this case, it should be made clear that in computer terms there is

no such a thing as an irrational number - (a number that cannot be expressed as a ratio of integers) - since computers deal with finite values. For example, the number 3.1415927 can be expressed as

$$\frac{31'415,927}{10'000,000}$$

which is formally a ratio of integers. However, when these integers are large, the practical results are identical to the case of irrational numbers.

Back to this case, when the ratio between carrier and modulator starts deviating from the (small) integer to integer relationship, the sound will become less and less pitched.

Values that are close but are not exactly an integer ratio produce a desirable effect closer to the characteristics of real life instruments. Since the partials are almost harmonics but not quite, the beating patterns between them produce a livelier sound.

CS24.ORC contains *instr 29*, which is a modification of *instr 28*.
Instead of being given the frequency of the modulator, it
requires the carrier to modulator ratio (c/m).

CS24.SC uses a carrier of 250 hz, a modulator with 5 partials and different c/m ratios to produce the four following sounds.

1. The carrier is half of the modulator (c/m = .5). The result will be a pitched sound at 250 hz.

2. The carrier is twice the modulator (c/m = 2). c/m is smaller than the number of partials, which is 5, therefore, according to section 8.3.3, the result will be the fundamental of the modulator. Since c/m = 2, the fundamental of the modulator is half of that of the carrier, i.e. 125 hz. Thus, the resulting sound will be an octave lower.

3. c/m is close to .5 (q/m = .501). Still a pitched sound of about 250 hz but livelier.

4. c/m is far from .5 (c/m = .35355). The sound is more like a bell.

Csound tape example 24 demonstrates these sounds. The orchestra and the score are listed below.

```
; CS24.ORC      -       Amplitude Modulator

sr = 44100
kr =  4410
ksmps = 10
nchnls = 1


instr 29        ;       Side bands and carrier
                ;       carrier and modulator with
                ;       different generating functions

;p4  : amplitude
;p5  : carrier
;p6  : attack
;p7  : decay
;p8  : carrier to modulator ratio. Should not be 0.
;p9  : percentage of carrier to be modulated
;p10 : carrier function table
;p11 : modulator table

    imod    =       p9/100.00           ; modulated part
    inomod  =       1 - imod            ; unmodulated part

    imf     =       p5/p8               ; modulator frequency
```

```
    kenv    linen  p4, p6, p3, p7        ; envelope

    acarr   oscil  1, p5, p10            ; carrier
    amod    oscil  1, imf, p11           ; modulator

    aoutm   =      acarr*amod*imod       ; modulated signal
    aoutnm  =      acarr*inomod          ; unmodulated signal
            out    kenv*(aoutm + aoutnm) ; output

endin

;CS24.SC

f1 0 512 10 .9 1 .78 .07 .24 .53 .09 .46
f2 0 512 10 1 .46 .56 .87 .35

;           p3   p4      p5    p6     p7  p8     p9    p10  p11
;instr start dur  amp     carr. attack dec c/m    mod%  carr mod
;                                             func func

i29    0     1.5  25000   250   .1     1   .5     95    1    2

i29    2.5   .    .       .     .      .   2      .     .    .

i29    5     .    .       .     .      .   .501   .     .    .

i29    7.5   .    .       .     .      .   .35355 .     .    .

e
```

8.4 **Dynamic Spectrum - Examples**

As usual, the best situation is one in which a dynamic spectrum can be produced and controlled. In order to achieve this, the following parameters can be made to change in

time:

1. The overall amplitude.

2. The frequency of the carrier.

3. The Carrier to Modulator Ratio (c/m)
4. The percentage of the carrier that is modulated.

Instrument 28, in CS25.ORC is an implementation of a ring modulator in which these four parameters change each according to different function tables. This instrument is very versatile and can produce the most dissimilar sounds.

CS25.SC uses it to produce the following:

8.4.1 Section 1

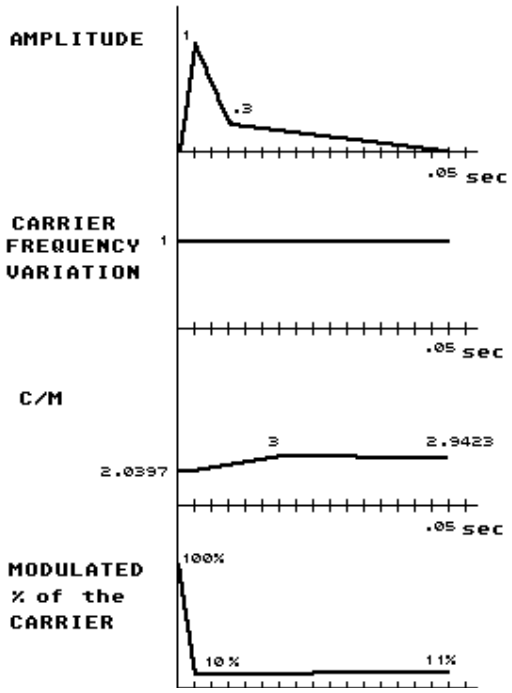
A short pitched percussive sound, similar to that made by a hollow wood instrument, is produced by changing very quickly from a highly inharmonic to a more or less harmonic spectrum.

This is done by varying the c/m ratio from 2.0397 to 3 and the modulated percentage from 1 to 0 at the beginning of the sound in 1/16th of the total duration. The sound has to be very short, about .05 seconds or less. Otherwise, the percussive part will be smeared.
If the duration through which the change from inharmonic to harmonic spectrum takes place is made longer, the result is a less pitched sound, and if it is made shorter, a more pitched and metallic sound, like a glockenspiel or vibraphone, will be heard.

The pitch of the carrier is left constant and the overall envelope has a very sharp attack and a quick decay in two stages.

The shape of the four time varying parameters can be seen in the following diagram.

Diagram 36 Short Percussive Pitched Sound Funtions



8.4.2 Section 2

A low bouncy sound is followed by a long glissando that fans out when it reaches its lowest point.

The bouncy quality is produced just by extending the duration of the percussive sound of the previous section. In order to achieve a softer attack the overall amplitude, c/m ratio and percentage of modulation change in a more gradual fashion. This can be seen in diagram 37.

Diagram 37 Low Bouncy Sound Functions

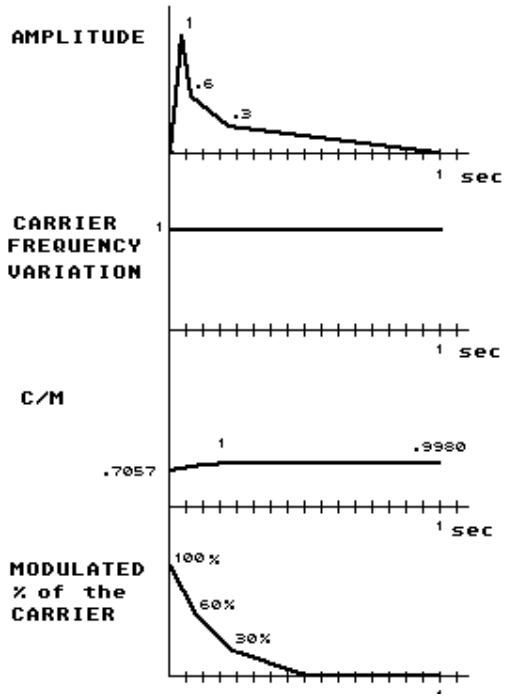


Diagram 38 describes the function tables of the second sound in this section. It should be noticed that there is a sudden increase of overall amplitude after about 2 seconds, that coincides more or less with the quickest frequency slide. When the overall amplitude becomes more or less stable, the fan out effect is achieved by changing the c/m ratio from 1 (harmonic) to .7001 (inharmonic). In order to avoid discontinuities, the modulation percentage increases gradually at this point giving the impression of a real fan out of frequencies.

Finally, it can also be seen that during the first 2 seconds, the percentage of modulation goes from 0 to 100% and back to 0, adding life to the pitched glissando sound, that first goes slightly up and then begins to descend, following the frequency function table.

8.4.3 Section 3

A bell-like sound is produced. The main characteristic of bell sounds is that they are initially very inharmonic and dissolve more or less into a sine wave, while the amplitude

envelope attacks very quickly (when the bell is hit), decays quickly at the beginning and slows its decay as time goes by. Therefore c/m has to be very inharmonic (1.4142) and has to control 100% of the sound at the beginning. The modulated percentage has to decrease to 0 in order to turn completely into the carrier: a sine wave. This is shown in diagram 39.

Diagram 38 Glissando - Fan Out Functions.

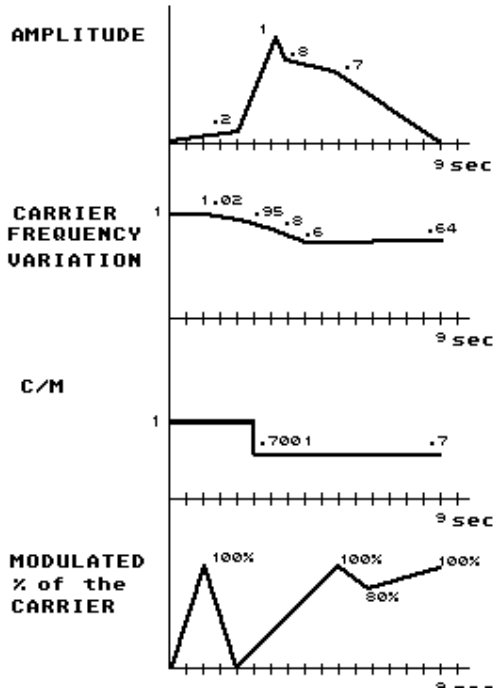
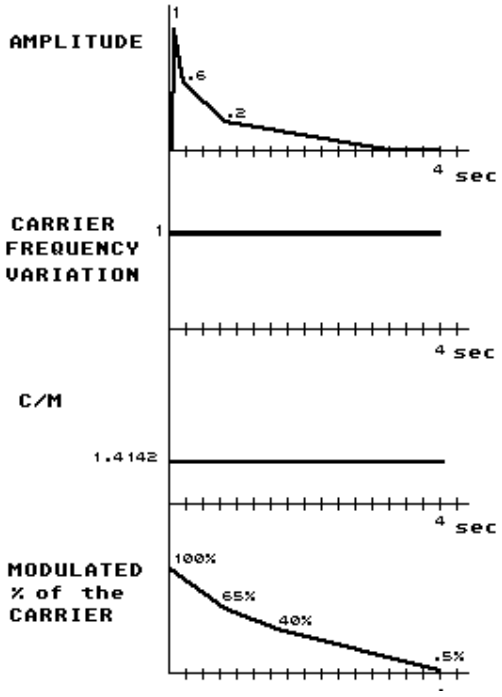


Diagram 39 Bell-like Sound Functions



The orchestra and the score are listed below. The score, CS25.SC, shows a new feature of *Csound*. Whenever it is required to interpolate between two values in the same parameter field (in this case **p4**), the symbol '**>**' can be used. In the current score, the symbol is used to produce a gradual increase in amplitude amounting to a *crescendo*.
The sounds produced can be heard in *Csound* tape example 25.

```
; CS25.ORC      -       Dynamic Spectrum Ring Modulator

sr = 44100
kr =  4410
ksmps = 10
nchnls = 1


instr 30        ; Variable parameter amplitude modulation
                ; instrument.
                ; The following can change in time :
                ;overall amplitude, carrier frequency,
                ;   c/m ratio, percentage of the carrier that
```

```
            ;   is modulated.

;p4  : max amplitude
;p5  : highest carrier pitch
;p6  : max carrier to modulator ratio (c/m)
;p7  : function table controlling amplitude
;p8  : function table controlling carrier pitch
;p9  : function table controlling c/m ratio
;p10 : function table controlling modulation percentage

        ;       one cycle

        ip3     =       1.0/p3

        ;       pitch frequency conversion

        ifr     =       cpspch(p5)

        ; envelopes

        kamp    oscil   p4,ip3,p7       ; amplitude
        kcar    oscil   ifr,ip3,p8      ; carrier freq
        kcmr    oscil   p6,ip3,p9       ; c/m
        kmp     oscil   1,ip3,p10       ; modulation %

        ;oscillators

        acarr   oscil   1, kcar, 10     ; carrier
        amod    oscil   1, kcar/kcmr,11 ; modulator

        aoutm   =       acarr*amod*kmp  ; modulated signal
        aoutnm  =       acarr*(1-kmp)   ; unmodulated signal

        ;mix and output
```

```
        out     kamp*(aoutm+aoutnm)

endin


;CS25.SC

;SECTION 1

t 0 120 4 120 6 140 6 250 8.6 40 8.8 60

;       percussive sound functions

f1  0 512 7 0         32  1       64    .3    384   0
f2  0 512 7 1        512  1
f3  0 512 7 0.6799    32  .6799   160   1     384   .9808
f4  0 512 7 1         32  .1      414   .11

;       carrier and modulator functions

f10 0 8192 10 1                             ; carrier
f11 0 8192 10 1 .9 .8 .7 .6 .5 .6 .7 .8         ; modulator

;           p3   p4     p5        p6      p7   p8   p9   p10
;instr start dur  max    highest   max     amp  carr c/m  mod%
;               amp    carr.     c/m     func func func func
;                      pitch

i30    0     .1    5000    9.03    3       1    2     3   4
i30     .2   .     1000     .      .       .    .     .   .
i30     .4   .     >        .      .       .    .     .   .
i30     .6   .     >        .      .       .    .     .   .
i30     .8   .     >        .      .       .    .     .   .
```

i30 1 . 25000 . . . . . .

i30 2 . . 10.02 . . . . . .

i30 3.5 . 15000 9.07 . . . . . .

i30 3.7 . 15000 . . . . . .

i30 3.85 . 15000 . . . . . .

i30 4 . 15000 10.04 . . . . . .

i30 4.4 . > 9.10 . . . . . .

i30 4.8 . > 10.06 . . . . . .

i30 5.2 . > 10.02 . . . . . .

i30 5.6 . > 10.08 . . . . . .

i30 6 . 32000 11.01 . . . . . .

i30 6.2 . . . . . .

i30 6.4 . . . . . .

i30 6.6 . > . . . . . .

i30 6.8 . > . . . . . .

i30 7.0 . > . . . . . .

i30 7.2 .09 > . . . . . .

i30 7.4 .085 > . . . . . .

i30 7.6 .08 > . . . . . .

i30 7.8 .075 > . . . . . .

i30 8.0 .07 > . . . . . .

i30 8.2 .065 > . . . . . .

i30 8.4 .06 > . . . . . .

i30 8.6 .05 8000 . . . . . .

s

```
;SECTION 2

;       low 'bounce' functions

f1  0 1024 7  0       32  1   64  .6    128  .3  800  0
f2  0 512  7  1       512  1
f3  0 512  7  .7057    64  .45 64  1     384  .9980
f4  0 512  7  1       64  .6  64  .3    128  0

;       gliss and fan sound functions

f5  0 512 7 0    128   .2   64  1      32  .8    96 .7 192 0
f6  0 512 7 1     64  1.02  64 .9      64  .8    64 .6 256 .64
f7  0 512 7 1    128  1      0 .7001  384  .7
f8  0 512 7 0     64  1     64 0      192  1     64 .8 128 1

;       carrier and modulator functions

f10 0 8192 10 1                         ; carrier
f11 0 8192 10 1 .9 .8 .7 .6 .5 .6               ; modulator



;           p3   p4     p5        p6      p7   p8   p9   p10
;instr start dur  max    highest   max     amp  carr c/m  mod%
;               amp    carr.     c/m     func func func func
;                      freq      pitch

i30    1     1    30000  6.01      1       1    2    3    4
i30    1.75  9    30000  9.01      1       5    6    7    8

s

; SECTION 3
```

```
;       Bell-like sound functions

f1  0 2048 7  0    4  1    128  .6   280  .2   1200  0
f2  0 512  7  1  512  1
f3  0 512  7  1  512  1
f4  0 512  7  1  100  .65  100  .4   312  .05

;       carrier and modulator functions

f10 0 8192 10 1                        ; carrier
f11 0 8192 10 1 .9 .8 .7 .6            ; modulator

;           p3   p4     p5        p6      p7   p8   p9   p10
;instr start dur  max    highest   max     amp  carr c/m  mod%
;                amp    carr.     c/m     func func func func
;                       freq      pitch

i30    1     4    30000  8.06      1.4142  1    2    3    4

e
```

9. Waveshaping

In the previous chapter, the outputs of a sine wave going through a linear amplifier and a non-linear processor that squares it were discussed. Each of these devices was characterized by a function that relates the input with the output. The function characterizing the linear amplifier was given by

$y = Ax$

and the squaring device was represented by the relation

$y = Ax^2$

These functions were also described graphically in diagrams 30 and 31 above.

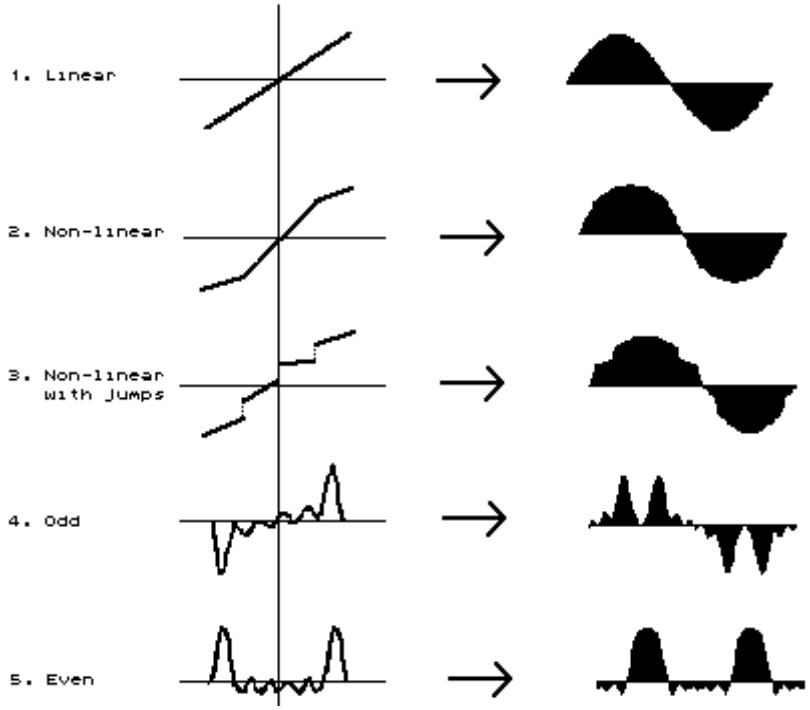
In general, every device, linear or non-linear can be described in terms of a mathematical relation between the input and the output. This relation is called the *transfer function*. If the input and the transfer function are known, it is possible to predict what the output will be. Therefore, the process can now be described as follows:

SIGNAL--->TRANSFER FUNCTION--->OUTPUT

The process by which a signal is distorted by a non-linear transfer function in order to alter its spectrum is called *waveshaping.* Examples of non-linear transfer functions can be found in a device that outputs the square of a signal, or a processor that multiplies it by a gain factor that depends on the actual value of the signal itself.

Csound tape example 26 contains the outputs obtained by passing a sine wave through devices with different transfer functions. These can be seen in the following diagram.

Diagram 40 Examples of Transfer Functions



The first is an example of a linear filter, the sine wave is not distorted but rather only multiplied by a constant factor.

In the second case, a non-linear function is used. If the value of the sine wave is below .7, it will be multiplied by a certain value but if it is larger than .7, it will be multiplied by a smaller instead value. The effect is one of compression around an amplitude of .7.

In case 3, the transfer function has abrupt leaps. This means that the value of the output signal will change very quickly at some points. But quick changes in amplitude mean high frequency components: abrupt leaps or sharp changes in direction of the transfer function produce infinite harmonics, exceeding the Nyquist frequency and causing aliasing.

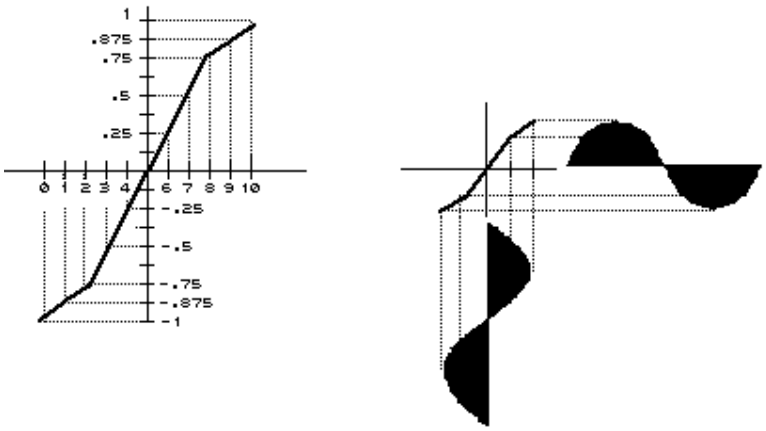
In case 4, the function is symmetrical with respect to the centre. A function that has this property is called *odd* and its output contains only odd harmonics (the fundamental, third harmonic, fifth, etc.).

Conversely, when a function is symmetrical with respect to the vertical axis, like in case 5, it is called *even* and only produces even harmonics. For this reason, the frequency of the fundamental (first harmonic) will not appear in the output, which sounds an octave above, as can be heard in the tape example.

9.1 A Waveshaping Instrument

This section, shows how to build a basic waveshaping instrument. The principle is simple: the transfer function can be represented by a table that consists of a series of numbers representing the values that should multiply the amplitude of the input. Each of this values can be addressed by indicating its position in the table, referred as the *index*. Diagram 41 shows a function and its representation in a table that has 11 points.

Diagram 41 Waveshaping Function Table



TABLE

Index	0	1	2	3	4	5	6	7	8	9	10
Value	-1	-.875	-.75	-.5	-.25	0	.25	.5	.75	.875	1

If a sine wave is fed to the input, its value determines an index in the table. The value of the function corresponding to that index determines the output. From the diagram, it can be seen that when the value of the sine wave is 0, the corresponding multiplier is the middle of the table, in this case: the fifth value (index = 5).

When the sine wave is -1, the multiplier is the first number of the table (index = 0). When it is 1, the multiplier is the last value of the table (index = 10).

This relationship between the sine wave and the index can be put as follows:

index=(value of sine + 1) x 5

If the value is 1 : index = (1+1)x5 = 2x5 = 10
If the value is 0 : index = (0+1)x5 = 1x5 = 5
If the value is -1: index = (-1+1)x5 = 0x5 = 0

It may be noticed that the size of the table is 10 (= 5x2), thus a generalization of the formula can be written for any size:

Index = (value of sine + 1) x (table size/2)

For values in between, the result of this formula may not be an integer, therefore it has to be rounded down to an integer value. This means that if the table is very small, a large error will be introduced when rounding. Therefore, the table has to be quite large (ideally 65536 for 16 bit resolution but sizes of 8192 may work reasonably well).

The previous formula can be made simpler if instead of changing from 0 to 10, the index changed from -5 to 5. Then

index=value of sine x table size/2

If the value is 1 : index = 1x5 = 5
If the value is 0 : index = 0x5 = 0
If the value is -1 : index = -1x5 = -5

This is equivalent to starting the table in the middle: if the index is positive the values after the middle of the table are referred, and if the index is negative, the reference is to the values before the middle. Therefore, it is useful to have an *offset* that indicates where to start the table. In the previous case the offset is 5 and the index varies from -5 to +5.

Generalizing again, in order to have the middle of the table as a starting point:

1.The offset must be

offset = (table size - 1) / 2

2.The index should vary between *+offset* and *-offset*.

In the previous example:

offset = (11 - 1)/2 = 10/2 = 5

and the index varies between -5 and 5.

The previous table is easy to handle since it has a middle value, but if instead of having 11 entries, the table had 12, it would not be possible to find a middle value. Nevertheless, *Csound* allows definition of the size of a function as a power of two less one (2^n- 1), which will always be odd.

Now that indexing of a function table has been explained, it is necessary to examine the statement that finds the value corresponding to the index. It is actually called TABLE, and its general form is now given

artableindex, func (, mode, offset, wrap)
where

ar is the table value corresponding to the index.

Index is exactly what it means.

func is the function table number (in the score).

mode is an optional facility in *Csound* that allows normalization, which means that the index can be specified as a fraction between 0 and 1, in which case *mode* must be 1 (*normalized mode*). Otherwise it should be set to 0 (*raw mode)*, its default value.

offset is an optional offset from the beginning of the table. Thus, the table can begin from any point. This is very useful for inputs that can assume negative values, like a sine or a cosine, in which case the index is taken from the middle of the table.

For example, if a sine wave that changes between -1 and 1 is waveshaped with a table of size *tablesize*: if the table is read in the raw mode, it can be started at the middle by making

offset = int((tablesize-1)/2)

and the index can be made to vary from *-offset* to *+offset*. Alternatively, if the table is read in normalized mode, the offset has to be offset = .5

wrap is an option that enables the use of indexes larger than the table size. If an index is bigger than the size of the table, and *wrap* is set to 1, it is 'wrapped

around' to the beginning.

For example if the size of the table is 33 and the index is 35, the 'wrapped index' is 2. Arithmetically, this can be expressed as

wrapped index = 35 modulo 33 = 2

The default value is 0, in which case there is no wrap.

CS26.ORC contains the implementation of a waveshaping instrument. CS26.SC was used to produce the examples of waveshaping functions displayed above (diagram 40).

```
; CS26.ORC      -       Waveshaper


sr = 44100
kr =  4410
ksmps = 10
nchnls = 1


instr 31        ;       Basic Waveshaping instrument

;p4  : amplitude
;p5  : frequency
;p6  : attack
;p7  : decay
;p8  : oscillator function
;p9  : waveshaping function

  ioffset =    .499  ; offset
       k1      linen   p4, p6, p3, p7   ; envelope
       a1      oscil   ioffset, p5, p8  ; oscillator
```

```
        awsh    table   a1,p9,1,ioffset  ; waveshaping value
                out     k1*awsh          ; output

endin

;CS26.SC

;       OSCILLATOR

f1 0 8192 10 1

;       WAVESHAPING FUNCTIONS

;linear
f2 0 8193 7 -1 8193  1

;non-linear
f3 0 8193 7 -1 2048 -0.7  4097   0.7  2048  1

;leaps
f4 0 8193 7 -1  2048  -0.5  0  -0.3  2048  0.1  0  0.4  2048  0.5
 0  0.7  2049  1

;odd function
f5 0 8193 9  1 1  180   2 .9 180   3 .8 180   4 .7 180   5 .6 180

;even function
f6 0 8193 9  1 1  90   2 .9  90   3 .8  90   4 .7  90   5 .6 90

;EVENTS

;           p3  p4    p5    p6     p7   p8     p9
;inst start dur amp   freq  attack dec  osc    wsh
;                                   func   func
```

```
i31   0     1   32000 440   .1    .1    1      2
i31   1.5   1   .     .     .1    .1    1      3
i31   3     1   .     .     .1    .1    1      4
i31   4.5   1   .     .     .1    .1    1      5
i31   6     1   .     .     .1    .1    1      6
```

e

9.2 Avoiding aliasing

When examining the examples of transfer functions in diagram 40, a certain difficulty has been pointed out: if a transfer function has sharp changes of direction or contains leaps, aliasing will take place because an infinite number of harmonics will be produced. Therefore, it is necessary to use only well behaved functions. This is ensured by utilizing polynomials.

In principle, any smooth function can be modelled by a polynomial. Furthermore, there is another characteristic that makes polynomials very desirable waveshapers: The highest harmonic produced can be predicted. In fact, it can be shown that the highest harmonic produced by waveshaping a sine wave through a polynomial transfer function is the one corresponding to the the highest power (the *order*) of that polynomial.

For example, the highest harmonic produced by the waveshaping function : $1 + x + 5x^2 + x^4$ is the fourth harmonic because the highest power in the polynomial is 4. It is now easy to avoid aliasing if the frequency of the input and the order of the transfer function are known.
Csound offers a GEN routine that produces polynomials: GEN3. It is given the coefficients (multipliers of the powers of x). Formally, the data is given is as follows:

minin maxin c0 c1 c2 c3 ...

where

minin is the minimum input value expected. Usually sine waves between -1 and 1 are used. When this is the case, *minin* is -1.

maxin is the maximum input value expected. If a sine wave between -1 and 1 are used. When this is the case, maxin is 1.

c0, c1.. are the coefficients of the polynomial. *c0* is a constant, *c1* multiplies x, *c2* multiplies x^2 ...

For example, in order to produce the polynomial

$1 + 2x + 3x^2 + 5x^3 + x^4$

assuming an input waveform between -1 and 1, a function can be defined as follows:

f1 0 8192 3 -1 1 1 2 3 5 1

9.3 **Chebyshev Polynomials**

It is very helpful to know what the highest harmonic is in order to avoid aliasing. It would be even more useful if it were possible to predict accurately the relative amplitude of each harmonic, since this would allow greater control of the spectrum. Fortunately, there is a set of polynomials called *Chebyshev Polynomials* that allow this type of control.

Each Chebyshev Polynomial produces a different harmonic when a sine wave is used as an input. Take for example the polynomial that produces the second harmonic (second order polynomial):

$T_2(x) = 2x^2 - 1$

Applying a sine wave to this transfer function and remembering that

$$\sin^2 a = \frac{1}{2} - \frac{1}{2}\cos(2a)$$

we obtain

$$T_2(\sin(2\pi ft)) = 2\left(\frac{1}{2} - \frac{1}{2}\cos(2(2\pi ft))\right) - 1$$

$$= 1 - \cos(2(2\pi f)t) - 1$$

$$= \cos(2\pi(2f)t)$$

This is a cosine of twice the frequency, corresponding to the second harmonic.

The first five Chebyshev polynomials of *first kind* are:

$T_0(x)$	=	1	Produces a DC component
$T_1(x)$	=	x	Produces the fundamental
$T_2(x)$	=	$1 - 2x^2$	Produces the second harmonic
$T_3(x)$	=	$4x^3 - 3x$	Produces the third harmonic
$T_4(x)$	=	$8x^4 - 3x^2 + 1$	Produces the fourth harmonic

It is now possible to control each resulting harmonic when a sine wave is used by combining the polynomials with different relative amplitudes. For example, in order to obtain a spectrum described below

Harmonic	Relative Amplitude
1	1
2	.3
3	0
4	.7

the following transfer function can be used

$$y = T_1(x) + .3\ T_2(x) + .7\ T_4(x)$$

If GEN3 is used, the each Chebyshev polynomial has to be replaced and then the expression has to be simplified. Nevertheless, this is not necessary because *Csound* provides a GEN routine especially designed for use with Chebyshev transfer functions. This is GEN13.

The data required by GEN13 is

ampin scaling h0 h1 h2 h3 ...

where

ampin — is the maximum input amplitude expected. Usually sine waves between -1 and 1 are used. When this is the case, ampin = 1.

scaling — is a scaling factor that multiplies the input after it is waveshaped. This is also usually 1 because the common way to control the amplitude is by using an envelope within the orchestra.

h0, h1.. — are the relative amplitudes of the Chebyshev polynomials, or in other words, the relative amplitudes of the harmonics. It is worth remembering that *h0* refers to a DC component, and not to the fundamental (*h1*).

In the next example, instrument 31 is used to produce the first 9 harmonics of a sine wave, which can be heard in *Csound* tape example 27. First, the sine wave is played with instrument 3, and then, GEN13 produces a DC component (that will not be heard), followed by the fundamental and the harmonics up to the ninth. This is done by setting all the relative amplitudes to zero except the one that represents the desired harmonic.

```
; CS27.ORC

sr = 44100
kr =  4410
ksmps = 10
nchnls = 1

instr 3         ;       Simple Oscillator

      k1      linen   p4, p6, p3, p7  ; envelope
      a1      oscil   k1, p5, p8      ; oscillator
              out     a1              ; output

endin



instr 31        ;       Basic Waveshaping instrument

;p4  : amplitude
;p5  : frequency
;p6  : attack
;p7  : decay
;p8  : oscillator function
;p9  : waveshaping function
```

```
 ioffset =    .5 ; offset
      k1      linen   p4, p6, p3, p7   ; envelope
      a1      oscil   ioffset, p5, p8  ; oscillator
      awsh    table   a1,p9,1,ioffset  ; waveshaping value
              out     k1*awsh          ; output

endin

;CS27.SC
; Use of GEN13 to produce a harmonic series out of a sine
; wave.

f1  0 8192 10 1                    ; sine wave

f2  0 8193 13 1 1 1                ; DC
f3  0 8193 13 1 1 0 1              ; 1 harmonic (fund)
f4  0 8193 13 1 1 0 0 1            ; 2 harmonic
f5  0 8193 13 1 1 0 0 0 1          ; 3 harmonic
f6  0 8193 13 1 1 0 0 0 0 1        ; 4 harmonic
f7  0 8193 13 1 1 0 0 0 0 0 1      ; 5 harmonic
f8  0 8193 13 1 1 0 0 0 0 0 0 1    ; 6 harmonic
f9  0 8193 13 1 1 0 0 0 0 0 0 0 1  ; 7 harmonic
f10 0 8193 13 1 1 0 0 0 0 0 0 0 0 1  ; 8 harmonic
f11 0 8193 13 1 1 0 0 0 0 0 0 0 0 0 1 ; 9 harmonic

;       SINE WAVE

;           p3    p4      p5    p6        p7      p8
;instr start dur   amp     freq  attack    decay   func

i3      0    1     25000   200   .1        .1      1

;       HARMONICS
```

```
;          p3  p4    p5    p6     p7    p8     p9
;inst start dur amp   freq  attack dec  osc    wsh
;                              func   func

i31   1    .5  25000 200   .1     .1   1       2
i31   +    .   .     .     .      .    .       3
i31   +    .   .     .     .      .    .       4
i31   +    .   .     .     .      .    .       5
i31   +    .   .     .     .      .    .       6
i31   +    .   .     .     .      .    .       7
i31   +    .   .     .     .      .    .       8
i31   +    .   .     .     .      .    .       9
i31   +    .   .     .     .      .    .       10
i31   +    .   .     .     .      .    .       11

e
```

9.4 Distortion Index and Dynamic Spectrum

So far, sine waves with amplitude 1 have been considered. The effects of having an amplitude different than 1 will now be examined. Take for example the sine wave:

$$A \sin(2\pi f t)$$

Suppose this waveform is sent through a device with the following transfer function

$$y = x + x^3$$

The output will be

$A \sin(2\pi ft) + A^3 \sin^3(2\pi ft)$

After some manipulation, the result is:

$$\left(A + \frac{3A^3}{4}\right) \sin^3(2\pi ft) - \frac{A^3}{4} \sin(2\pi (3f)t)$$

It can be noticed that the relative amplitudes of the harmonics produced are dependent on the amplitude of the sine wave:

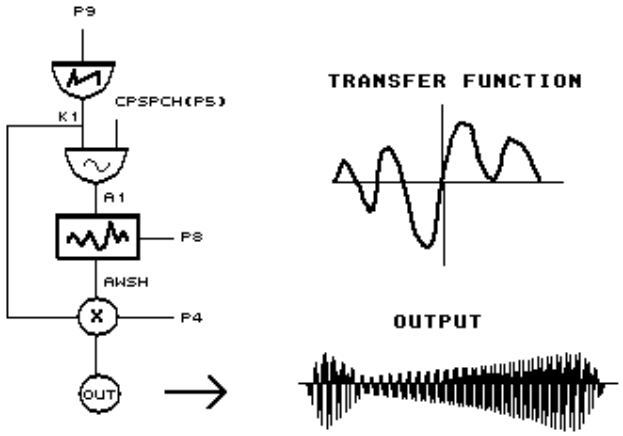
$h_1 = A + 3A^3/4$
$h_3 = A^3/4$

Even more more important is the fact that each relative amplitude depends on the amplitude of the input in a different way. This means that if the amplitude of the input changes, each partial will be affected differently. Therefore, by changing A in time, the spectrum of the output will be modified, since the relative amplitudes of the partials will change with respect to one another. For this reason, **A** is called the *modulation index*.

In general, the higher the partial, the quicker it will grow when **A** grows or decrease when **A** decreases.

Finally, in order to achieve dynamic spectrum, it is enough to make the modulation index change in time. CS28.ORC contains an instrument that implements a waveshaper that takes account of the envelope, as shown in diagram 42.

Diagram 42




```
; CS28.ORC

sr = 44100
kr =  4410
ksmps = 10
nchnls = 1

instr 32        ; Dynamic Waveshaping instrument
                ; Distortion index changes according to
                ; function indicated by p11

;p4  : amplitude
;p5  : frequency
;p6  : oscillator function
;p7  : waveshaping function
;p8  : envelope (distortion index) function


        ifr     =     cpspch(p5)        ; pitch to freq
        ip3     =     1/p3              ; one cycle
        ioffset =     .5          ; offset
        k1      oscil 1, ip3, p8        ; envelope (dist idx)
        a1      oscil offset, ifr, p6   ; oscillator
        awsh    table k1*a1,p7,1,ioffset ; waveshaping value
```

```
        out   k1*p4*awsh        ; max amp & output

endin



;CS28.SC

;sine wave
f1  0 8192  10 1

;waveshaper
f2  0 8193  13 1 1 0 1 .7 -.8 -.3 .1 .8 -.9 -1 1

envelope (distortion index)
f3  0  512   7 0 64 1 64 .2 184 .5 152 1 48 0

;           p3   p4     p5    p6    p7    p8
;instr start dur  amp    pitch osc   wsh   index
;                        func  func  func

i32    0     1    15000  7.06  1     2     3
i32    +     0.5  .      8.00  .     .     .
i32    +     0.25 .      7.01  .     .     .
i32    +     0.25 .      6.02  .     .     .
i32    2.5   2.5  20000  5.08  .     .     .

e
```

Two warnings should be issued about a variable index:

1. In order to achieve effective waveshaping, the changes in envelope must happen in the range where the transfer function changes the most. For example, if the function has rapid changes for amplitudes between .8 and 1 and is fairly linear for amplitudes

less than .8, a sine wave with an amplitude of .7 will hardly experience any distortion when that specific function.

2. Since waveshaping depends on the amplitude of a signal, it is usually necessary to produce wide changes in the envelope in order to get reasonable effects. But this also means changes in intensity, which may require balancing the signals after they have been waveshaped (using BALANCE, for example).

It is also important to notice that only harmonic spectra is produced when waveshaping. In order to achieve an inharmonic spectrum it is necessary to have an inharmonic input or to do something to the input before or after it is passed through the waveshaper. For example, it could be ring modulated.

Another way of producing a dynamic spectrum is by changing the transfer function in time.

CS29.ORC contains an instrument that simulates this effect by waveshaping the same signal through two different waveshapers and then cross-fading the outputs. While the output of one of the waveshapers fades out the other fades in. In order to achieve a rich spectrum, the input is ring modulated before it is passed through the waveshapers.

CS29.SC contains events that use a sine wave carrier, a more complex modulator with an irrational c/m to produce an inharmonic spectrum and a very smooth symmetrical varying index. The smooth index function consists of half a sine wave that is achieved by using GEN9 with a harmonic number of .5 (half of the fundamental). When the frequency of this kind of oscillator is set to 1/p3, it will only produce half a cycle for the whole duration of the event.

The sounds produced with CS29.ORC and CS29.SC can be heard in *Csound* tape example 29. The orchestra and score are now listed.

```
; CS29.ORC

sr = 44100
```

```
kr =  4410
ksmps = 10
nchnls = 1

instr 33        ; Dynamic Waveshaping instrument
          ; changes linearly between two different
          ; transfer functions that waveshape the same
               ; signal

;p4  : amplitude
;p5  : frequency
;p6  : oscillator function
;p7  : beginning waveshaping function
;p8  : final waveshaping function
;p9  : index function
;p10 : c/m ratio

       ip3     =       1/p3              ; one cycle

       icarr   =       cpspch(p5)        ; carrier freq
       imod    =       icarr*p10         ; modulator freq
       ioffset =       .5                ; offset
       k1      oscil   1, ip3, p9        ; envelope
       kmix    line    1, p3, 0          ; mix proportions
       acarr   oscil   ioffset, icarr, 1  ; carrier
       amod    oscil   1, imod,p6        ; modulator

       a1      =       acarr*amod        ; modulated signal

       awsh1   table   k1*a1,p7,1,ioffset ; 1 waveshaping
       awsh2   table   k1*a1,p8,1,ioffset ; 2 waveshaping

       ; mix output

            out     k1*p4*( kmix*awsh1 + (1-kmix)*awsh2 )

endin
```

```
;CS29.SC

f1 0 8192 10 1                          ; carrier
f2 0 8192 10 1 .7 .3 .8 .4                  ; modulator

; waveshaper 1
f3 0 8193 13 1 1 0 1 -.9 -.8 .7 .6 -.5 -.4 .3 .2

; waveshaper 2
f4 0 8193 13 1 1 0 .1 .1 .2 1 .3 .2 .1

; envelope (index function)
f5 0 1024 9 .5 1 0



;       p3  p4     p5   p6   p7   p8   p9   p10
;ins stt dur amp    pch  mod  frst last idx  ring
;                       func wsh  wsh  func c/m
;                           func func
i33  0   5   25000  7.08 2     3   4     5   .35355
i33  6   2   .      7.06 .     .   .     .   .354
i33  7   6   .      8.05 .     .   .     .   .35355

e
```

10. Frequency Modulation (FM)

Frequency modulation is another non-linear technique used in sound synthesis. However, while amplitude modulation controls the spectrum of sounds by manipulating their amplitude, FM does this by manipulating their frequency.

As a matter of fact, there is a carrier, a modulator and the carrier to modulator ratio also plays an important role. There is also an index, related to frequency (rather than to amplitude), that can help predict some of the characteristics of the resulting spectrum.

10.1 The Basic Frequency Modulator

One of the advantages of frequency modulation is that very simple waveforms can be used to produce a very complex spectrum. The basic frequency modulator consists of a *carrier* sine wave that changes its frequency according to another sine wave: the *modulator*. The corresponding mathematical expression is

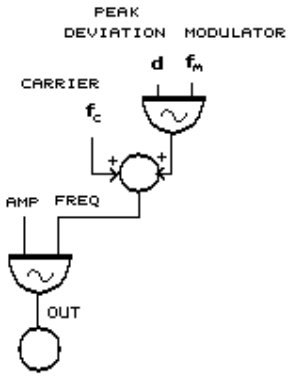
$$f(t) = A \sin(2 \pi f_c t + d \sin(2 \pi f_m t)) \qquad (10.1)$$

where

f_c is the carrier frequency.
f_m is the modulator frequency.
A is the amplitude of the carrier (also the amplitude of the output).
d is the modulator amplitude

The basic frequency modulator is shown in diagram 43.

Diagram 43 Basic FM Instrument



f_c carrier frequency.

f_m modulator frequency.

A amplitude

d modulator amplitude

It can be shown that expression (10.1) is equivalent to a sum of simpler sine waves of frequencies

f_c

$f_c + f_m$ $f_c - f_m$

$f_c + 2 \times f_m$ $f_c - 2 \times f_m$

$f_c + 3 \times f_m$ $f_c - 3 \times f_m$

.

.

etc.

For example, if the carrier is 1000 hz and the modulator 100 hz, the resulting frequencies are:

1000 hz

1000 + 100 = 1100 hz	1000 - 100 = 900 hz
1000 + 200 = 1200 hz	1000 - 200 = 800 hz
1000 + 300 = 1300 hz	1000 - 300 = 700 hz

.

.

etc.

Expression (10.1) can be written as follows:

$$
\begin{aligned}
f(t) = \; & A_0 \sin(2\pi f_c t) + \\
& A_1 \sin(2\pi (f_c+f_m)t) + A_1 \sin(2\pi (f_c-f_m)t) + \\
& A_2 \sin(2\pi (f_c+2f_m)t) + A_2 \sin(2\pi (f_c-2f_m)t) + \\
& A_3 \sin(2\pi (f_c+3f_m)t) + A_3 \sin(2\pi (f_c-3f_m)t) + \\
& A_4 \sin(2\pi (f_c+4f_m)t) + A_4 \sin(2\pi (f_c-4f_m)t) +
\end{aligned}
$$

.

.

etc.

$$
\begin{aligned}
= \; & A_0 \sin(2\pi f_c t) + \\
& A_1 \times (\sin(2\pi (f_c+f_m)t) + \sin(2\pi (f_c-f_m)t)) + \\
& A_2 \times (\sin(2\pi (f_c+2f_m)t) + \sin(2\pi (f_c-2f_m)t)) + \\
& A_3 \times (\sin(2\pi (f_c+3f_m)t) + \sin(2\pi (f_c-3f_m)t)) + \\
& A_4 \times (\sin(2\pi (f_c+4f_m)t) + \sin(2\pi (f_c-4f_m)t)) +
\end{aligned}
$$

.

.

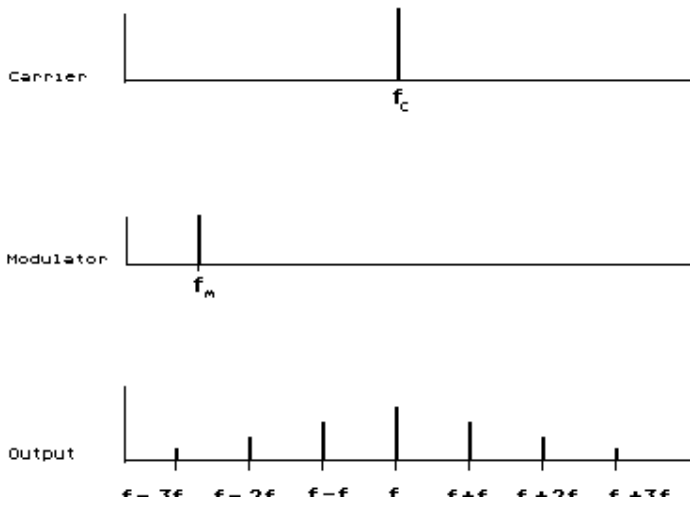
etc. (10.2)

or more concisely

$$f(t) = \sum_i A_i [\sin(2\pi (f_c+if_m)t) + \sin(2\pi (f_c-if_m)t)] \qquad (10.3)$$

The physical interpretation of expressions (10.2) and (10.3) is that frequencies are produced on both sides of the carrier. These frequencies are appear at intervals of $\mathbf{f_m}$ hz. They are also called *side bands*. Diagram 44 displays the spectrum obtained using a carrier $\mathbf{f_c}$ and a modulator $\mathbf{f_m}$. It may be noticed that the components at both sides of the carrier have the same amplitudes.

Diagram 44 Spectrum Produced by FM



The basic FM oscillator can be implemented in *Csound* by straightforward realization of diagram 43, as shown in *instr 34*. There is, however, a special FM statement in *Csound* that will be introduced later. For the time being, until some concepts are clarified, *instr 34* will be used.

```
instr 34        ; Basic FM instrument

;p4  : carrier amplitude
;p5  : carrier frequency
;p6  : attack
;p7  : decay
;p8  : modulator amplitude
;p9  : modulator frequency
;p10 : oscillator function

    kenv linen   p4,p6,p3,p7      ; envelope
    amod oscil   p8,p9,p10        ; modulator
    aout oscil   kenv,p5+amod,p10 ; modulated carrier
    out     aout            ; output

endin
```

10.2 Distortion Index

Expression (10.1) and diagram 43 show that the amplitude of the modulator, **d**, is actually the maximum possible fluctuation, or *peak deviation* from the frequency of the carrier. Therefore the frequency of the resulting waveform changes between the values

$$f_c - d \quad \text{and} \quad f_c + d$$

If **d** is very small compared with $\mathbf{f_m}$, the effect of the modulator on the frequency of the overall waveform will be very small, whereas if **d** is relatively large, it will affect the waveform drastically. In order to show the effects of **d**, *instr 34*, is used by CS30.SC to produce five sounds with a carrier of 400 hz, a modulator of 440 hz and different values of **d**: 0, 10, 100 800 and 1600 hz.

The score and orchestra are listed below. The results can be heard in tape example 30:

When **d** = 0 the result is no modulation at all. Thus, a pure sine wave at the frequency of the carrier is heard. This is in accordance with the mathematical representation of FM.

$f(t) = A \sin(2 \pi f_c t + 0 \times \sin(2 \pi f_m t))$

$= A \sin(2 \pi f_c t)$

When **d** = 10 very little distortion is produced.
When **d** = 100 the distortion begins to be noticeable.
When **d** = 800 the distortion is more significant.

When **d** = 1600 the sound becomes completely distorted.

```
; CS30.ORC      Basic FM instrument

sr = 44100
kr =  4410
ksmps = 10
nchnls = 1

instr 34        ; Basic FM instrument

;p4  : carrier amplitude
;p5  : carrier frequency
;p6  : attack
;p7  : decay
;p8  : peak deviation
;p9  : modulator frequency
;p10 : oscillator function

    kenv    linen   p4,p6,p3,p7         ; envelope
    amod    oscil   p8,p9,p10           ; modulator
```

```
    aout    oscil   kenv,p5+amod,p10     ; modulated carrier
            out     aout                 ; output

endin

;CS30.SC        Modulation of a 400 hz sine wave by a
;               440 hz sine wave and different deviation
;               values

f1  0 8192 10 1 ; sine wave

;             p3   p4   p5   p6    p7   p8   p9   p10
;instr startdurampcarrattackdecpeakmodfunc
;                     freq           dev  freq

i34    0      1.5 15000 400  .1    .1     0 440  1
i34    2       .   .    .   .     .     10 .    .
i34    4       .   .    .   .     .    100 .    .
i34    6       .   .    .   .     .    800 .    .
i34    8       .   .    .   .     .   1600 .    .

e
```

So far we have seen that when the peak deviation is very small in relation to the modulator, the effect it produces is minor, but as it increases its distorting effect is more significant. Therefore, it makes sense to device a way of measuring how large the peak deviation is in comparison to the modulating frequency. The simplest way of comparing two quantities is by finding their ratio. For this reason, the *modulation index* I is defined as follows:

$$I = \frac{\text{peak deviation}}{\text{modulator frequency}} = \frac{d}{f_m}$$

then

If **d** is 0, then **I** = 0.
If **d** is very small compared to $\mathbf{f_m}$ then **I** is much smaller than 1 (**I** << 1).
If **d** is equal to $\mathbf{f_m}$, then **I** = 1.
If **d** is larger than $\mathbf{f_m}$, then **I** > 1.

The conclusions drawn from example 30 can now be rephrased:

When **I** = 0 no distortion is produced.
When **I** << 1 very little distortion is produced.
When **I** = 1 the distortion begins to be noticeable.
When **I** = 2 the distortion grows.
When **I** = 4 (I >> 1) the sound becomes completely distorted.

It can be proved that the reason for distortion is the fact that when the index grows, the amplitudes of the sidebands further away from the carrier also grow, therefore, more partials, corresponding to these distant sidebands, can be heard. At the same time, the intensity of the carrier decreases with an increasing index.

As a rule of thumb, it is useful to consider the number of pairs of sidebands that became audible equal to the index. Thus, in total, the sound will have

$$2 \times I + 1 \text{ partials.}$$

For example:

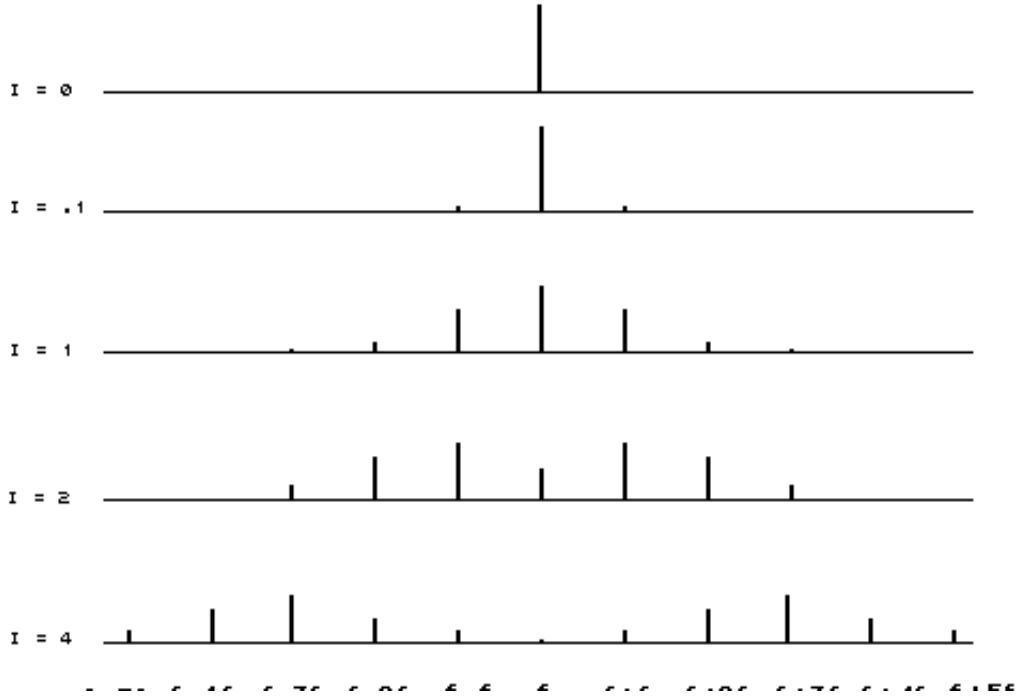
If I = 0,there will be only one significant partial: the carrier.

If I = 1,there will be one partial at each side of the carrier. In total 2x1 + 1 = 3 partials, corresponding to $f_c - f_m$, f_c and $f_c + f_m$.

If I = 2,there will be 2 partials at each side of the carrier. In total 2x2 +1 = 5 partials, etc.

Diagram 45 shows the relative amplitudes of the partials produced by frequency modulation for five values of **I** (0, .1, 1, 2, 4). These are not to scale and just give an idea of which partials are prominent.

Diagram 45 Relative Amplitudes of Partials Produced by Different Values of the Index.



In this section, it has been shown that the modulation index controls the relative amplitudes of the partials. In general it can be said that the modulation index determines how many partials will be heard.

10.3 **Reflected Frequencies**

By now, it should be clear that if the modulation index grows, more and more sidebands can be heard. At some point the sidebands produced by the difference between $\mathbf{f_c}$ and multiples of $\mathbf{f_m}$ will either be zero or negative.

In order to interpret these values it is necessary to look at expressions (10.2) or (10.3) which consist of a sum of sine waves. In the case of 0hz

$\sin(2\ /7\ (0)\ t) = \sin(0) = 0$

Therefore, the resulting waveform has no DC component which is usually an advantage FM has over waveshaping.

In the case of -200 hz, using the trigonometric identity

$\sin(a) = -\sin(-a) = \sin(a-\ /7)$

it can be seen that

$\sin[2\ \pi\ (-f)t] = -\sin[2\ \pi\ ft] = \sin[2\ \pi\ ft -\ \pi]$

This means that the components with negative frequencies always reflect themselves as positive frequencies with negative amplitudes, which is the same as to say that they have a phase shift of π (they will lag or be in front by half a cycle).

For example, if we take a carrier of 400 hz and a modulator of 200 hz, the first three pairs of sidebands are:

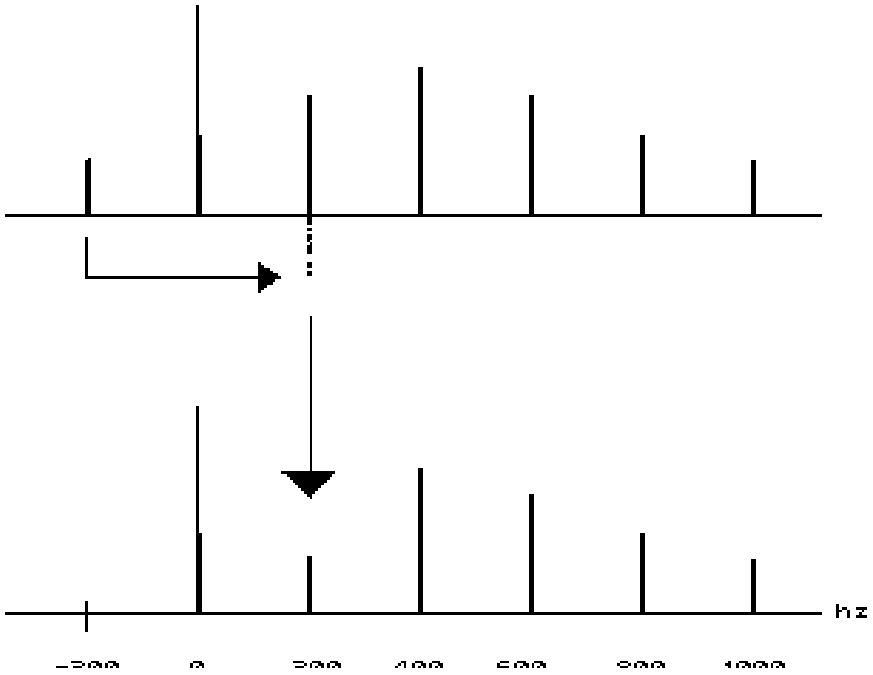
400 + 200 = 600 hz400 - 200 = 200 hz

400 + 400 = 800 hz400 - 400 = 0 hz

400 + 600 = 1000 hz400 - 600 = -200 hz ---> 200 hz

Therefore, when the spectrum is viewed, the partial at -200 hz reflects as a partial of 200 hz with negative amplitude as shown in the following diagram.

Diagram 46 Reflected frequency of -200 hz and final result, when a carrier of 400 hz is modulated by a a 200 hz signal



It is worth pointing out that in this case the final amplitude of the partial at 200 hz is the result of adding the amplitudes of the reflected -200 hz and the original 200 hz sideband.

10.4 Carrier to Modulator Ratio (c/m)

If a 400 hz carrier is modulated by a 200 hz signal. The first few sidebands are:

200 hz and 600 hz
0 hz and 800 hz
-200hz (which reflects as 200 hz) and 1000 hz
-400hz (which reflects as 400 hz) and 1200 hz

It can be seen that the resulting partials are:

200, 400, 600, 800, 1000, 1200 hz, etc.

Therefore, the fundamental is 200 hz, which is equal to the frequency of the modulator.

If, instead, a 200 hz carrier is modulated by a 400 hz signal, the first few sidebands are:

-200hz (which reflects as 200 hz) and 600 hz
-600hz (which reflects as 600 hz) and 1000 hz
-1000hz (which reflects as 400 hz) and 1400 hz
-1400hz (which reflects as 400 hz) and 1800 hz

Now, the resulting partials are:
200, 600, 1000, 1400 hz, etc.

Therefore, the fundamental is 200 hz, which, this time, is the frequency of the carrier.

In both cases, the resulting partials are harmonics (multiples of the fundamental); however, in the first example, all the harmonics are present, whereas in the second, only the odd harmonics (1, 3, 5, etc.) appear.

If a third case is taken with a carrier of 200 hz and a modulator of 283 hz, the first few

sidebands are:

- 83 hz (which reflects as 83 hz) and 483 hz
-366 hz (which reflects as 366 hz) and 766 hz
-649 hz (which reflects as 649 hz) and 1049 hz

Therefore, the partials are:

83, 200, 366, 483, 649, 766, 1049 hz, etc.

The fundamental is 83 hz this time, which is neither the frequency of the carrier nor of the modulator. Furthermore, the spectrum is not harmonic.

The resulting sounds in each case can be heard in *Csound* tape example 31, which is the output produced by using CS31.ORC and CS31.SC . The sounds are the outputs of:

1.Carrier = 400 hz. Modulator = 200 hz.
2.Carrier = 200 hz. Modulator = 400 hz.
3.Carrier = 200 hz. Modulator = 283 hz.

In order to preserve the value of the index (**I** = 2) throughout these three cases, the peak deviation has to be different for each case. Since

$I = d/f_m$

then

$d = f_m \times I$

therefore

In case 1: d = 200 x 2 = 400 hz.

In case 2: d = 400 x 2 = 800 hz.

In case 3: d = 283 x 2 = 566 hz.

CS31.ORC contains *instr 34*, already listed above. CS31.SC is shown below.

```
;CS31.SC        Different carrier-modulator combinations


f1  0 8192 10 1 ;sine wave

;             p3   p4   p5   p6     p7   p8   p9   p10
;instr startdurampcarrattackdecpeakmodfunc
;                    freq          dev  freq


i34    0       1.5 15000 400  .1     .1   400  200  1
i34    2        .   .    200  .      .    800  400  .
i34    4        .   .    200  .      .    566  283  .

e
```

In the previous example, the fundamental and the harmonicity of the spectrum could have been predicted because they depend on the ratio between the carrier and the modulator **c/m**. This is similar, but much simpler case of ring modulation and can be summarized in as follows:

10.4.1 Harmonic Spectrum (c/m = Ratio of Integers)

Whenever the carrier to modulator ratio can be represented as a ratio of (small) integers, the spectrum will be harmonic. This can be expressed as

$f_c/f_m = N_1/N_2$ where N_1 and N_2 are integers.

Furthermore, if this expression can be simplified so that N_1 and N_2 have no common factors then there is more information that can be inferred:

10.4.1.1The fundamental is

fund = $f_c/N_1 = f_m/N_2$

In the first case of the previous example, the carrier to modulator ratio is

400/200 = 2/1

therefore

fund = 400/2 = 200/1 = 200 hz

In the second case the carrier to modulator ratio is

200/400 = 1/2

therefore

fund = 200/1=400/2=200 hz

10.4.1.2 The carrier is always the N_1 harmonic of the output spectrum.

In the first two cases of the previous example the carrier to modulator ratios are

respectively

400/200=2/1The carrier is the second harmonic ($N_1 = 2$)

and

200/400=1/2The carrier is the fundamental ($N_1 = 1$)

10.4.1.3Different values of N_2

If N_2 is 1, all the harmonics will appear. This can be corroborated by the first case above.

If N_2 is even, the spectrum contains only odd harmonics. The second case above corroborates this.

If $N_2 = 3$, every third harmonic is missing.

An example is now given: if the carrier is 200 hz and the modulator is 300 hz, then

c/m = 2/3

The sidebands are

-100 (reflects as 100) and 500 hz
-400 (reflects as 400) and 800 hz
-700 (reflects as 700) and 1100 hz
-1000 (reflects as 1000), etc.

The fundamental is then: 100 hz.
The carrier is the second harmonic: 200 hz.
The harmonics are:

100, 200, 400, 500, 700, 800, 1000, 1100 hz, etc.

It can be noticed that every third harmonic (300, 600, 900, etc.) is missing. If instead of using a modulator of 300 hz, a 100 hz is used, the harmonics appearing in the output will be
100, 200, 300, 400, 500, 600, 700, 800 hz, etc.

In this case, no harmonics are missing, but if the same index value of, say, 4 is used, then the highest harmonic heard will be 600 hz whereas in the case of a 300 hz modulator, the highest harmonic heard will be 800 hz, making the sound brighter (higher harmonic content).

As a general rule of thumb, in order to achieve brighter or sounds with the same index, the carrier to modulator ratio should be increased, and viceversa.

Csound tape example 32 illustrates this. It contains two sounds, both with a 400 hz carrier. The first sound has a 400 hz modulator, and the second a 800 hz modulator. In both cases, the modulation index is 3 (the peak deviation is adjusted according to the modulator frequency).

It can be noticed that the second sound is brighter and somewhat thinner than the first since it contains higher and more spaced harmonics. The harmonic content of each of the sounds can now be calculated.

The modulation index is 3, therefore, the number of partials heard is approximately

2x3 + 1 = 7

There will be 3 partials above and three partials below the fundamental (negative partials will reflect). In the case of the first sound (c/m = 1/1), the fundamental is 400 hz and the harmonics are

400, 800, 1200 and 1600 hz

In the case of the second sound (c/m = 1/2) therefore, the harmonics of are

400, 1200, 2000 and 2800 hz

The sounds in example 32 were produced by compiling CS32.ORC and CS32.SC. The orchestra uses *instr 34*. The score is given below.

```
;CS32.SC   Same carrier (400 hz) with two different ;  modulators (400 and 800 hz)
f1  0 8192 10 1 ;sine wave

;             p3   p4   p5   p6   p7   p8   p9   p10
;instr startdurampcarratckdecpeakmodfunc
;                    freq          dev  freq

i34    0       1.5 15000 400  .1   .1   1200 400  1
i34    2        .   .    .    .    .    2400 800  .

e
```

10.4.2Inharmonic Spectrum (c/m is not a Ratio of Integers)

When c/m cannot be expressed as a ratio of two small integers, the resulting spectrum will be inharmonic.

The third sound in *Csound* example 31 is an illustration of inharmonic spectrum. The carrier is 200 hz and the modulator 283 hz.

$$c/m = \frac{200}{283} \sim \frac{1}{2^{1/2}}$$

The closer c/m is to a ratio of integers, the more harmonic the spectrum. A spectrum that is almost harmonic has a lively character, as explained in chapter 8, section 8.3.4

Csound tape example 33 contains three sounds with a carrier of 300 hz and the following carrier to modulator ratios: 1, 1.003 and 1.4142 (~ $2^{1/2}$).

The first sound is completely harmonic.

The second is more lively because of its slight inharmonicity, which produces beats between the partials.

The third sound is completely inharmonic.

The orchestra uses *instr 34*. The score is listed below.

```
;CS33.SC        carrier = 300 hz
;               c/m =   1       (modulator = 300)
;                       1.003   (modulator = 300.9)
;                       1.4142  (modulator = 424.26)
;               index = 2

f1  0 8192 10 1 ;sine wave

;          p3   p4   p5   p6   p7   p8    p9    p10
;instr stt durampcarratckdecpeak mod func
;                  freq          dev   freq

i34    0   1.5 15000 300 .1   .3  600    300    1
i34    2    .   .    .    .   601.8  300.9  .
```

i34 4 . . . . . 848.52 424.26 .

e

In this section, it has been shown how the carrier to modulator ratio determine the values of frequencies in spectrum of the output.

10.5 FOSCIL

Now that the concepts of *index* and *carrier to modulator ratio* have been clarified, it is possible to discuss a special oscillator offered by *Csound*: FOSCIL.

The general form of FOSCIL is

arfoscilamp, freq, carr, mod , index, func, (, phase)

where

amp is the amplitude.

freq is a common frequency from which the carrier and the modulator can be obtained by multiplying respectively by *carr* and *mod*. For example, if

Freq = 100 hz
carr = 1
mod = 2

the carrier is 100 x 1 = 100 hz
and the modulator is 100 x 2 = 200 hz

This makes it easy to see the carrier to modulator ratio. In this example

$$\frac{carr}{mod} = \frac{1}{2}$$

carr is a number that multiplies *freq* in order to obtain the carrier frequency. It does not have to be an integer.

mod is a number that multiplies *freq* in order to obtain the modulator frequency. It does not have to be an integer.

index is the modulation index.

func is the function defining the basic waveform of the carrier and the modulator.

phase is the phase of the waveform. It does not have to be specified, in which case it is assumed to be 0.

As in the case of OSCIL, there is a more accurate (but slower) statement that has exactly the same parameters as FOSCIL, called FOSCILI. However, accuracy can be achieved by using FOSCIL with a large function table (8192 samples or more).

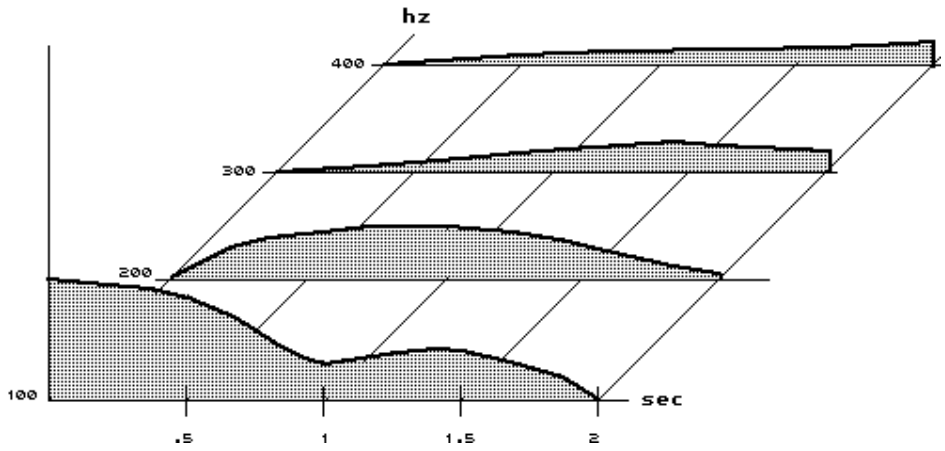
10.6 Dynamic Spectrum

The big success of FM in its initial days was due to the alternative it offered to the production of dynamic spectra which did not involve a great number of parameters. Obviously, the carrier frequency can be made to vary in time, as well as the carrier to modulator ratio (equivalent to varying the value of the modulator). But the parameter that produced the real revolution when FM synthesis first appeared was the modulation index.

As the index changes in time, different sidebands become more or less prominent

therefore changing the composition of the spectrum. For example, if we take a 100 hz carrier, a 100 hz modulator (c/m = 1) and make the index increase from 0 to 4 during 2 seconds, the beginning of the sound will be a pure sine wave of 100 hz, and as time goes by, the second, third, fourth and fifth harmonics will, respectively, be more prominent. The following diagram is a qualitative representation of the resulting dynamic spectrum.

Diagram 47 $f_c = f_m = 100$ hz
Index changes from 0 to 4 in 2 sec



This sound can be heard in *Csound* tape example 34, the orchestra and score of which are given next.

```
; CS34.ORC      -      FM synthesis instrument using FOSCIL
;                     Produces dynamic spectrum by varying
;                     the modulation index

sr = 44100
kr =  4410
ksmps = 10
nchnls = 1


instr 35        ;      General FM instrument
```

```
;p4  : amplitude
;p5  : pitch (to be converted to frequency)
;p6  : carrier/frequency
;p7  : modulator/frequency
;p8  : maximum index
;p9  : oscillator function (usually a sine wave)
;p10 : index function

icyc    =       1/p3            ; one cycle

    ; envelope with attack = decay = .1 sec

    kenv    linen   p4,.1,p3,.1

    ; index

    kidx    oscil   p8,icyc,p10

    ; FM oscillator

    a1      foscil  kenv,p5,p6,p7,kidx,p9

    ; output

        out     a1

endin




;CS34.SC        dynamic spectrum index changes
;               from 0 to 4 in 2 seconds
```

```
f1 0 8192 10 1          ; oscillator function
f2 0 512  7 0 512 1     ; index function

;             p3   p4   p5   p6   p7   p8      p9     p10
;inst startdurampfreqcarrmodmaxfunc
;                                  index   oscidx

i35   0       2   15000 100  1    1    4       1      2

e
```

Proper manipulation of the modulation index yields a great diversity of sounds. Some of them are shown in the next section. It should be pointed out that the following examples only take advantage of a time varying index, and that more variety can be achieved by changing the carrier to modulator ratio and the frequency of the carrier, as well as by mixing more than one FOSCIL module to obtain, for example, formant regions.

10.7 Examples

The first three examples are all implemented using CS35.ORC and CS35.SC and can be heard in *Csound* tape example 35.

10.7.1Bell-like Sounds

One of the easiest sounds to produce with FM are bells. As discussed in Chapter 8, Section 8.4.3, the main characteristic of bell sounds is that they are initially very inharmonic, with very high partials that decay quickly leaving the lower partials to resonate.

This can be translated in terms of **c/m** and index.

Inharmonic spectrum -> c/m irrational. 1.215 is a good approximation.

high partials dominant ->high index = 6

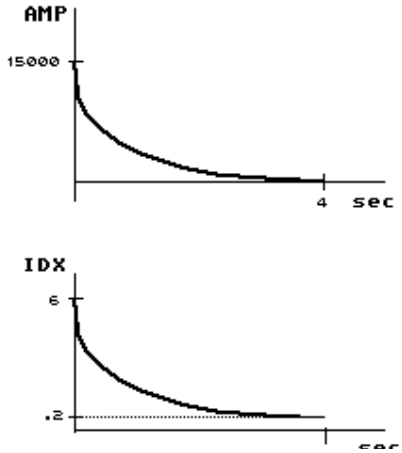
low partials dominant -> low index = 0.2

High partials that are initially dominant and decay very quickly means that the actual index should be initially high and decay rapidly. An exponential decay matches this quite well.

Therefore, the carrier to modulator ratio can be kept constant at 1.215, while the index should be a control variable that changes exponentially from 6 to 0.2 through the duration of the note.

Finally, the whole amplitude envelope that starts abruptly (when the bell is hit) and decays rapidly in the beginning and slowly afterwards, can also be matched by an exponential.

Diagram 48 Amplitude Envelope and Index for a Bell



10.7.2Drum-like Sounds

In this case a few changes can be made to the bell parameters. The principal change consists of shortening the duration of the whole sound to 0.5 seconds. This makes it more percussive in comparison with the slow decaying bell. The amplitude and index envelope follow exactly the same shape. Finally, the following changes are made:

1.In order to simulate hitting a membrane, which is more flexible than a metallic bell, the

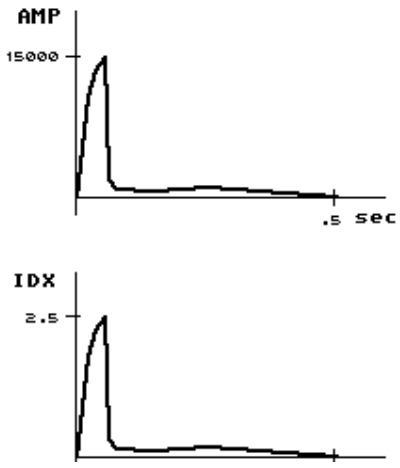
envelope does not start abruptly at its highest value but rather starts from 0 and then increases for about 0.02 seconds until it reaches its maximum value and then decays quickly for 0.025 seconds. This produces a softer attack.

2.After it decays, there is some increase in the amplitude and index which corresponds to the membrane still being deformed after it has been hit, thus producing higher harmonics.

3.The highest value of the index is only 2.5, as compared with 6 for the bell, therefore lower harmonics are heard, especially after the initial decay. This produces an overall darker sound which is more characteristic of a drum.

4.c/m = 1/2.21, therefore the harmonics are more spaced, producing a slightly hollower sound when the index is high.

Diagram 49 Drum-like Sound Amplitude Envelope and Index



10.7.3Knock on Wood

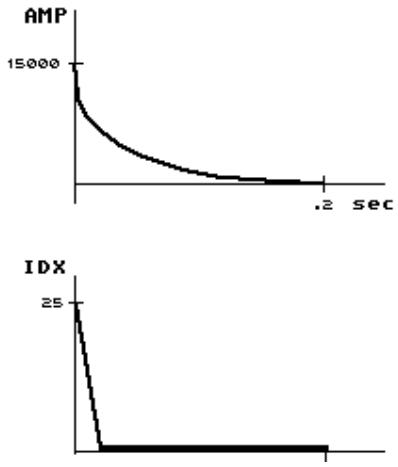
A knock on wood is characterized by being very short and changing very quickly from very high harmonics to almost the fundamental. In order to achieve this sound:

1.The overall duration of the sound is shortened even further to 0.2 seconds.

2.The amplitude envelope is left to be exponential. Due to the short duration the result is a very rapid decay.

3.The index falls linearly from 25 to 0 in 0.025 of a second.

Diagram 50 Knock on Wood



The orchestra and score are listed below.

```
; CS35.ORC      -   FM synthesis instrument using FOSCIL
;                  Produces all kinds of percussive sounds

sr = 44100
kr =  4410
ksmps = 10
nchnls = 1


instr 36  ;    General FM instrument, with envelope changing
          ;according to function table given by p10 and
          ;    index changing according to function table given
          ;    by p11.
```

```
;p4  : amplitude
;p5  : pitch (to be converted to frequency)
;p6  : carrier/frequency
;p7  : modulator/frequency
;p8  : maximum index
;p9  : oscillator function (usually a sine wave)
;p10 : amplitude function
;p11 : index function

    icyc  =       1/p3            ; one cycle
    ifreq =       cpspch(p5)      ; frequency

    ; envelope

    kenv  oscil   p4,icyc,p10

    ; index

    kidx  oscil   p8,icyc,p11

    ; FM oscillator

    a1    foscil  kenv,ifreq,p6,p7,kidx,p9

    ; output

        out     a1

endin

;CS35.SC  Different Percussive sounds obtained by changing
;        envelopes, index functions and durations of the sounds

f1 0 8192 10 1          ; oscillator function
```

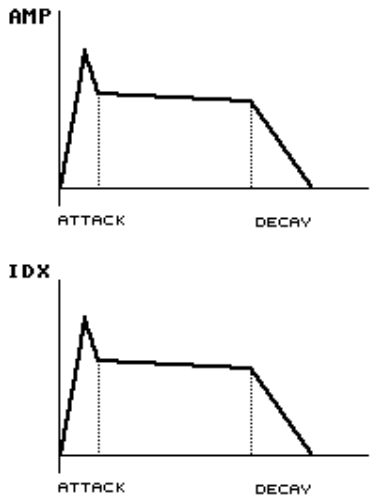
```
f2 0 512  5 1 512 .0001 ; amplitude function
f3 0 512  5 1 512 .2    ; index function

;       BELLS

;         p3   p4   p5       p6   p7   p8       p9   p10  p11
;inst sttdurampppitchcarrmodmax        functions
;                                    indexoscampidx

i36    0  4   15000  7.11   1   1.215 6       1   2   3
i36    5  .   10000  8.11    .   .   .        .   .   .
i36    6  .   10000  8.07    .   .   .        .   .   .
i36    7  .   10000  8.09    .   .   .        .   .   .
i36    8  .   10000  8.02    .   .   .        .   .   .
i36   10  .   10000  8.02    .   .   .        .   .   .
i36   11  .   10000  8.09    .   .   .        .   .   .
i36   12  .   10000  8.11    .   .   .        .   .   .
i36   13  .   10000  8.07    .   .   .        .   .   .

s

;       DRUMS

;index and amplitude function
f2 0 2048 7 0 20 .75 35 .9 35 1 104 .15 64 .1 186 .08 630 .15
964 0

;         p3   p4   p5       p6   p7   p8       p9   p10  p11
;inst sttdurampppitchcarrmodmax        functions
;                                    indexoscampidx

i36   0   .5  15000  5.07    1    2.21  2.5     1   2   2
i36   +   .5  15000  6       .    .     .       .   .   .
i36   +   .5  15000  6.05    .    .     .       .   .   .
```

```
s

;       KNOCK ON WOOD

f2 0 512  5 1 512 .0001         ; amplitude function
f3 0 512  7 1 64 0 448 0        ; index function



;         p3   p4   p5        p6   p7   p8        p9   p10  p11
;inst sttduramppitchcarrmodmax        functions
;                                    indexoscampidx

i36   2   .2  15000  6       1  1.4142 25      1   2   3
i36   2.5 .   >      .       .  .    .       .   .   .
i36   3   .   25000  .       .  .    .       .   .   .


e
```

10.7.4Brass-like Sounds

A well know characteristic of a brass sound is that the louder it is played, the higher the harmonics that will be heard. In fact, the brilliance of the brass is due to higher harmonics appearing in the spectrum. This can be easily simulated by making the index follow the same shape as the amplitude envelope as can be seen in the following diagram:

Diagram 51 Brass-like Tone Amplitude Envelope and Index



The spectrum should be harmonic and all the harmonics should be present, therefore c/m should be close to 1. In order to give the sounds a slightly livelier characteristic, the spectrum can be slightly detuned by making c/m = 1.001 or 1.002, instead of 1.

The maximum index is usually 5 for low to middle register sounds (middle C to D a ninth above). For higher sounds a value of 7 is used to give more brightness.

Finally, the initial blow of the attack is achieved by an overshoot in the contours of the envelope and index.

The instrument used is a modification of the general FM instrument in which the envelope and index are adapted to this special case. Softer articulation can be achieved by varying the attack time and fade out can be accomplished by increasing the decay.

The result of using CS36.SC can be heard in *Csound* tape example 36.

```
; CS36.ORC      -       Brass-like tones instrument

sr = 44100
kr =  4410
ksmps = 10
nchnls = 1

instr 37        ;       FM instrument that generates
                ;       brass-like tones
;p4  : amplitude
;p5  : pitch (to be converted to frequency)
;p6  : attack
;p7  : decay
;p8  : carrier/frequency
;p9  : modulator/frequency
;p10 : maximum index
;p11 : oscillator function (usually a sine wave)

    ifreq  =       cpspch(p5)      ; frequency
    ihatck =       p6*.5           ; half of attack time
    ist1   =       p4*.75          ; steady state amplitude 1
    ist2   =       p4*.6           ; steady state amplitude 2
    istd   =       p3-p6-p7        ; duration of steady state

    ; envelope

    kenv   linseg  0,ihatck,p4,ihatck,ist1,istd,ist2,p7,0

    ; index

    kidx   =       kenv*p10/p4

    ; FM oscillator
```

```
    a1     foscil  kenv,ifreq,p8,p9,kidx,p11

    ; output

          out     a1
endin

;CS36.SC        Brass-like sounds

f1 0 8192 10 1          ; oscillator function

;             p3p4    p5      p6   p7   p8   p9    p10  p11
;inst stt duramp pitchatckdeccarrmod maxosc
;                                          idx  func

i37   0     .1  10000 8.01    .05   .04  1   1.002  5   1
i37   +     .   >     .     .     .    .    .     .   .
i37   +     .   >     .     .     .    .    .     .   .
i37   +     .9  15000 .     .2    .1   .    .     5   .
i37   +     .1  10000 .     .05   .04  1   1.002  5   1
i37   +     .   >     .     .     .    .    .     .   .
i37   +     .   >     .     .     .    .    .     .   .
i37   +     .9  15000 .     .2    .1   .    .     5   .
i37   +     .1  10000 .     .05   .04  1   1.002  5   1
i37   +     .   >     .     .     .    .    .     .   .
i37   +     .   >     .     .     .    .    .     .   .
i37   +    1.8  18000 8.04    .2    .4   .    .     5   .

i37   +     .1  12000 8.01    .05   .04  1   1.002  5   1
i37   +     .   >     .     .     .    .    .     .   .
i37   +     .   >     .     .     .    .    .     .   .
i37   +     .9  18000 8.04    .2    .1   .    .     5   .
i37   +     .1  14000 8.01    .05   .04  1   1.002  5    1
i37   +     .   >     .     .     .    .    .     .    .
```

i37 + . > . . . .

i37 + .9 22000 8.04 .2 .1 . . 5 .

i37 + .1 14000 8.01 .05 .04 1 1.002 5 1

i37 + . > 8.04 . . . . . .

i37 + . > 8.08 . . . . . .

i37 + 2.1 25000 9.01 .2 .2 . 1.001 5 .

s

i37 .15 .15 22000 8.11 .08 .04 1 1.002 5 1

i37 + .8 23000 8.09 .1 .1 . . . .

i37 + .15 22000 8.08 .08 .04 . . . .

i37 + .8 23000 8.06 .1 .1 . . . .

i37 + .15 22000 8.04 .08 .04 . . . .

i37 + .8 23000 8.09 .1 .1 . . . .

i37 + .15 22000 8.08 .08 .04 . . . .

i37 + .8 23000 8.06 .1 .1 . . . .

i37 + .15 22000 8.04 .08 .04 . . . .

i37 + .25 22000 8.06 .08 .04 . . . .

i37 + .25 > 8.09 .08 .04 . . . .

i37 + .25 > 9.01 .08 .04 . . . .

i37 + 1.8 28000 9.04 .2 .1 . 1.001 7 .

i37 + .75 > 9.06 . . . . . .

i37 + 3 32000 9.09 . 1.5 . 1.0005 . .

e

PART IV

Sound Processing

11. The Phase Vocoder

Sound processing is one of the most exciting areas of computer music. This is especially true because of the possibilities opened by applications of the Fast Fourier Transform (FFT) to the analysis and resynthesis of dynamic spectra.

A typical approach consists of analysing a sound, obtaining a numeric description of its evolving spectrum, which can then be processed and finally resynthesized. This is schematically described below.

ANALYSIS --->PROCESSING --->RESYNTHESIS

One of the most powerful tools that enable musicians to do precisely this is the *Phase Vocoder*. This chapter will discuss its use and some of applications, but in order to do this, it is worth spending some time on the principles governing the Fast Fourier Transform. This subject will be treated, as much as it is possible, in a qualitative way in order to avoid the mathematical complications usually associated with Fourier Analysis, which are not always fruitful to those interested in making music and for which there is already an extensive bibliography.

11.1 Time Domain and Frequency Domain

There are two ways of describing a digital filter. One is by representing its behaviour after a very loud and extremely short signal - an *impulse* - is applied to it. The filter will react to this signal and will exhibit one and only one characteristic output as time goes by. This type of representation is called the *impulse response*, which describes the behaviour of the filter through time. Therefore, when the impulse response is used, the filter is said to be characterized in the *time domain*. An example of this is the famous bullet test, used to measure the reverberation of a room: in principle, a gunshot is fired and then the reverberating sound is recorded and measured. The room is considered to be a filter to which an impulse (the gunshot) is applied.

The second way of representing a filter is by describing how it attenuates or amplifies each possible frequency. In other words, a diagram is plotted showing the amplitude of the output when pure sine waves of different frequencies are used separately as inputs. This is its *frequency response* and, in this case, the filter is characterized in the *frequency domain*. Examples of frequency responses were given when describing the behaviour of the low-pass, high-pass, band-pass and band-stop basic filters in Chapter 7, Section 7.2.

11.2Discrete Fourier Transform

In the analogue world, an impulse can be visualized as a very quick spike in a voltage source. A digital impulse would be the sampling of this signal as shown in the diagram 52.

But the analogue impulse has a very particular characteristic: Its spectrum contains all the frequencies. This means that when the Fourier method is applied to the analogue impulse in order to find its partials, the result will tell us that it has partials at all possible frequencies and that the coefficients of those are all 1.

Diagram 52 Analogue and Digital Impulses

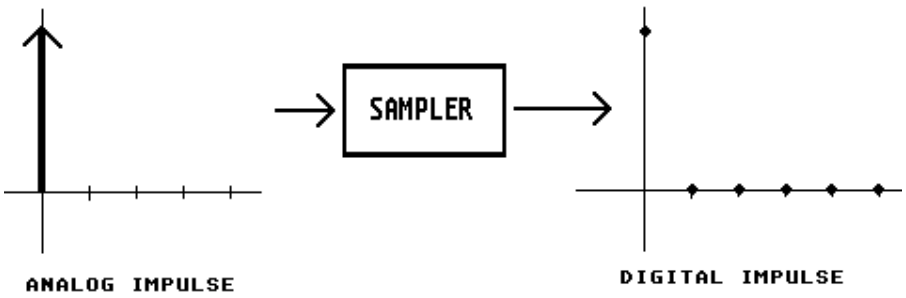
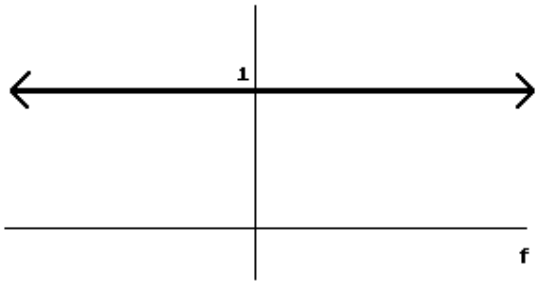


Diagram 53 Fourier Components of Analogue Impulse



The immediate conclusion is that applying an analogue impulse to an analogue filter is equivalent to applying all possible frequencies simultaneously. However, the frequency response is the behaviour of the filter at all possible frequencies when they are applied one at the time. Therefore, in the end, the two representations are equivalent. The link between them is provided by the Fourier decomposition into partials, or, in other words, by using the Fourier Transform.

Since the digital impulse can be seen as a sampled analogue impulse and the digital frequency response can be obtained by applying sampled sine waves at all frequencies, the digital impulse and frequency responses are obtained by using samples of the same signals that produced their analogue counterparts and it makes sense to find a digital version of the Fourier Transform that will link between them. It can be formally demonstrated that there is such a transform. It is called the *Discrete Fourier Transform* (DFT), which can be viewed as a sampled version of the analogue transform that turns the digital impulse response into the digital frequency response.

In order to convert the frequency response into the impulse response a reverse of the DFT called the *Inverse Discrete Fourier Tranmsform* (IDFT) is used.

11.3 Sampling the Frequency Response

If the digital responses are conceived as being obtained by means of using sampled versions of corresponding analogue signals, and the DFT is viewed as a sampled version of the analogue Fourier Transform, then the digital impulse and time responses themselves can be visualized as a sampled versions of their analogue counterparts.

Each sampled point of the frequency response is produced by calculating the DFT for a

certain frequency value. This means computer time and power used for each calculation, therefore in order to avoid wasting this time and power the response should not be oversampled. In other words, a minimum number of samples should be taken such that information is not lost.

In order to find this minimum, we will imagine a digital impulse applied to a non-recursive filter. We would like to be able to isolate each component in the output and find out its amplitude and phase. This will be done by sampling an analogue impulse using a limited number of samples, then finding what are the maximum and minimum frequencies that can be represented with that number of samples. The minimum frequency will give the distance between the values of two adjacent sampled components. As with all sampling, no information can be gathered between two samples which applies to any frequency laying in between the values of those sampled components. Consequently, there is no extra benefit obtainable from trying to gather information to an accuracy of less of the minimum frequency that can be sampled.

The minimum number of samples of the frequency domain is therefore

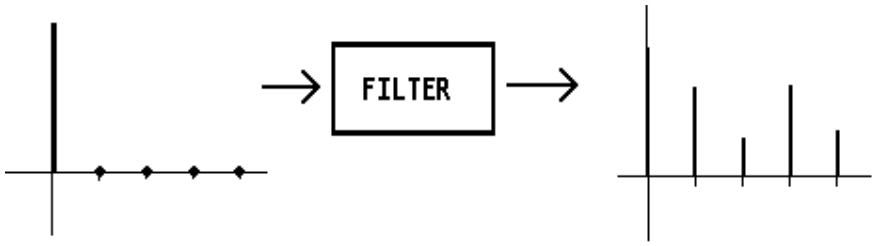
$$\text{Number of sampled frequencies} = \frac{\text{maximum freq}}{\text{minimum freq}} \qquad (11.1)$$

11.3.1 Sampling an Impulse

In practice, when we want to use a computer to calculate the DFT it is impossible to sample the impulse to infinity. In any case, there is no need to do so because we are dealing with non recursive filters, which do not feed back their output, therefore after a while the impulse response becomes zero. It is enough to take as many samples are

necessary to include all the non-zero outputs of the filter. In order to simplify we will assume a filter that requires only 5 samples, shown in diagram 54, and a sampling rate of 20 samps/sec.

Diagram 54 Filter with 5 sample Impulse Response



11.3.2Finding the Maximum Frequency

Since it was assumed that the input is the sampled version of a real impulse, all the frequencies should be contained. However, in order to avoid aliasing, only frequencies up to the Nyquist component (= sampling rate/2) can be sampled. This makes the maximum frequency equal to 10 hz.

11.3.3Finding the Minimum Frequency

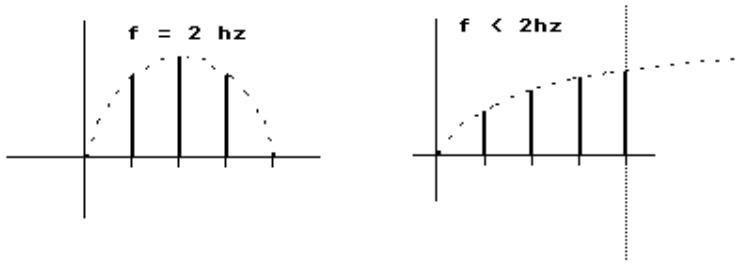
Our main purpose is to be able to find the amplitude and phase of as many partials as we can. Our limitation is the number of samples of the impulse response, in this case: 5.

If the filter could only let through one frequency out of the whole spectrum the output would be a sine wave. Now, in order to obtain the information (amplitude and phase), we must be sure that the number of samples taken contains at least half a cycle of that sine wave (the other half can be constructed out of the first one because the shape of a sine wave is known).

In the present example, it is necessary that the 5 samples represent half a cycle. Therefore, 10 samples portray a whole cycle and since the sampling rate is 20 samps/sec, there will be two cycles in one second, that is a frequency of 2 hz.

If we try to identify frequencies smaller than 2 hz, it will not be feasible to produce a complete half cycle, so there is no point in detecting frequencies of less than 2 hz, since they will not render enough information. This can be seen in diagram 55.

Diagram 55



Enough Information Not Enough Information

Two conclusions can now be drawn:

1.The minimum frequency that can be made a sampled point and still give useful information is 2 hz. Thus, it is enough to space the sampling points every 2 hz up to 10 hz. And from here, the sampled points of the frequency response will be 2, 4, 6, and 8 hz.

2.The number of frequency samples excluding 0 hz is 5 (2, 4, 6, 8 and 10hz), and is equal to the number of points taken from the impulse response. This is in accordance with expression (11.1) and should be expected since both, frequency and time domain

representations are equivalent.

If a resolution of 2 hz is not enough, the only way of improving this is by using a filter that needs more points in order to represent its impulse response. For example, if 10 samples had been taken, then these 10 could be used to portray half a cycle of a sine wave, 20 samples would portray the whole cycle, and at a sampling rate of 20 samples/sec, this would mean a frequency of 1 hz. Therefore doubling the length of the impulse response halves the minimum frequency.

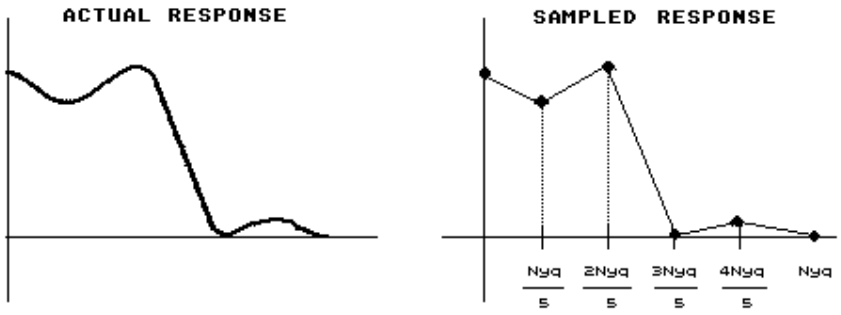
A quick way of finding the minimum frequency is by dividing the Nyquist frequency by the length in number of samples that are taken at the output of the filter in order to find the DFT. In general if the filter's impulse response is **N** samples long, we only need to sample its frequency response at the following frequency values.

$$0,\ \frac{Nyq}{N},\ \frac{2Nyq}{N},\ \frac{3Nyq}{N},\ \ldots\ \frac{(N-1)Nyq}{N}\ \text{and}\ Nyq$$

where *Nyq* is the Nyquist frequency = sampling rate/2

The final conclusion is that for a filter of length **N**, we only need to take **N** sampled points of the frequency response in order to obtain enough information. We will also obtain **N** sampled frequencies. In other words, the whole frequency range up to Nyquist is divided into **N** equal parts. More sampled points are unnecessary since they do not render any extra information. Diagram 56 shows the actual frequency response of a 5 point impulse response filter and the sampled frequency response.

Diagram 56



11.4 DFT of a Digital Signal

The DFT was seen to be very useful when going back and forth between the time and frequency domain. But there is more to it than just representing filters.

If we examine the procedure we used to arrive at the frequency response by means of the DFT, it can be noticed that the impulse response of the output of a filter was taken and used to calculate the sampled points of the frequency response of the filters, and that this was the same as finding the frequency components of that output.

So, for any arbitrary signal, its frequency components can be calculated by imagining it is the output of a filter. If **N** samples of that signal are taken, a filter of length **N** (its impulse response has **N** terms before it turns into 0) can be assumed.

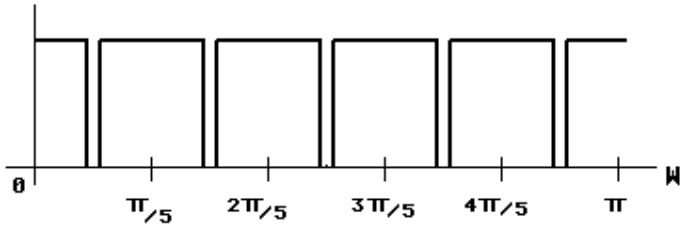
11.5 The Filter Bank Approach

The previous approach for visualizing the DFT of an arbitrary signal was useful in order to find out how to optimize the calculations in a computer.

Sometimes, it is easier to visualize the DFT of an arbitrary signal as the output of a bank of band-pass filters that have that signal for an input. This is similar to a graphic equalizer with centre frequencies at the sampled points and bandwidth at half way between them. If **N** points of a signal are taken in order to obtain the DFT, this can be seen as feeding the signal to a bank of **N** filters, since this would mean **N** sampled frequencies.

Diagram 57 shows a filter bank equivalent to a 5 point DFT.

Diagram 57 5 Point DFT Filter Bank Equivalent



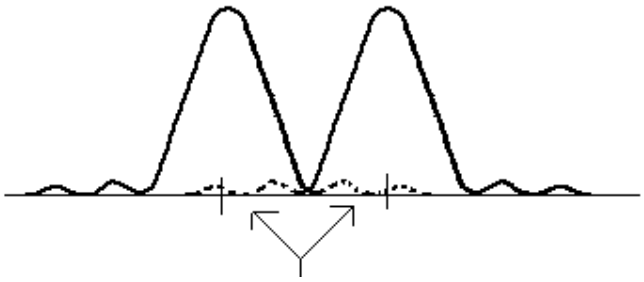
It may be noticed that the first channel looks like a low-pass filter and the last like a high-pass, making the total number add up to 6 filters. While theoretical considerations prove that these two represent only one filter that wraps around, for practical purposes these can be viewed as being independent. In other words, if **N** points are taken then **N + 1** filters will be produced: a low-pass, **N - 1** band-pass and a high-pass.

If a signal has components within a certain bandwidth, these will show in the output of the respective filter. Each of those filters is usually called a *channel*.

Obviously, if there are more filters, the bandwidth will be narrower and the analysis will be more accurate. But it was seen above that in order to get **N** meaningful divisions of the frequency range, it is necessary to analyse a signal that is **N** samples long. Therefore, in order to improve the accuracy of the DFT more samples of a signal need to be analysed.

It is worth remembering that a non-ideal filter has ripples and a transition region. This condition can be improved (but not completely eradicated) when the length of the filter is increased. Thus, when the DFT produces a bank of filters the neighbouring frequencies will not be completely filtered out and they will overlap with the results of the other filters. This is known as *spectral leakage* and it is shown in diagram 58.

Diagram 58 Spectral Leakage



11.6 The Fast Fourier Transform (FFT)

As stated above, the accuracy of spectral analysis depends on the number of samples that are analysed. For musical purposes, for example, a 1024 point DFT is a reasonable compromise.

But a 1024 point requires 1'049,600 additions and sums done by a computer. This means long computation times even by the quickest machines and DSP processors.

For this reason, computer scientists found it necessary to produce an algorithm that reduces dramatically the number of operations done, by avoiding redundancy in the calculation of the DFT. For obvious reasons, this algorithm is called the *Fast Fourier Transform* (FFT).

As a comparison, an implementation of a 1024 FFT requires 'only' 10,240 additions and multiplications: It is 1000 times faster than the ordinary DFT. As the number of points increases, the FFT becomes more and more efficient in comparison with the DFT.

It must be stressed that the FFT is not an approximation of the DFT but rather the real thing. The only constraint it presents is that the number of points **N** must be a power of 2.

11.7 The Phase Vocoder

The Phase Vocoder is one of the most striking musical applications of FFT. The principle behind it is quite simple: since the Fourier components of sounds with dynamic spectrum change as time goes by, the spectrum is analysed at successive instants, rather like snapshots of its state, or like frames from which a moving picture is composed. The numerical results of the analysis of each frame are stored in the computer as pairs of

amplitude-frequency, corresponding to each channel in the filter bank, which can then be processed and used for resynthesis.

The reader may ask why is it necessary to store frequency if the centre of each channel can be easily determined. The answer is that although the centre is known, the frequency fluctuates above or below this value due to the fact that the phase of each component is not constant but also changes in time: a sine wave that changes its phase also has to change its frequency in order to be able to lag behind or leap ahead the normal development of its cycle. In some cases a channel may even have a frequency value that is lower than the one preceding it.

The fact that changes in frequency are taken into account in order to describe phase fluctuations is what makes the Phase Vocoder more accurate in its analysis than its predecessor, the *Channel Vocoder*, which only depicts changes in the amplitude of each component.

11.7.1 Direct Filtering

One of the most straightforward applications of the the Phase Vocoder is its use as a very sharp filter, by omitting some of the channels in the filter bank when resynthesizing. This is done with options **-i** and **-j**. The resynthesized sound will only contain the channels starting at the value indicated by **-i** and ending at the value indicated by **-j**.

PVoc example 1 contains an oboe low C (~261.6 hz) sampled at 44100 samps/sec, followed by three resynthesized versions:
1. *obclow*. Only the fundamental was allowed to pass, by implementing a low-pass filter with ~270 hz cut-off.

2. *obcmid*. Frequencies between ~270 hz and ~860 hz (second and third harmonics) were allowed to pass by implementing a band-pass filter.

3. *obchigh*. Frequencies between ~860 hz and 22050 (Nyquist) - from the fourth harmonic

onwards - were allowed to pass by implementing a high-pass filter of ~860 hz cut-off.

The oboe sound was analysed using 2048 channels, in which case the option **-N**, which indicates twice the number of channels, must be 4096.

In order to calculate the values of the channels corresponding to the frequency values a formula can be deduced. If 2048
channels are used then it is possible to imagine a bank of 2049 filters, of which one is a low-pass filter and one is a high-pass (see Section 11.5, diagram 57).

Assuming ideal filters, the bandwidth of each will be

$$Bw = \frac{\text{Nyquist}}{\text{Channels}} = \frac{22050}{2048} \qquad (11.2)$$

The low-pass (channel 0) begins at 0 hz and ends at *bw*/2.

The first band-pass filter (channel 1) has its centre frequency at

cf1 = end of lowpass + bw/2

= bw/2 + bw/2 = bw

The second band-pass (channel 2) at

cf2 = end of first band-pass + bw/2

= 2 x bw

The third band-pass (channel 3) at

$$cf3 = 3 \times bw$$

The **kth** band-pass (channel k) at

$$cfk = k \times bw$$

Therefore, if a frequency **f** is desired as the centre frequency of a channel, then

$$k \times bw/2 = f \quad (11.3)$$

therefore

$$k = \frac{f}{bw} \quad (11.4)$$

Replacing (11.2) in (11.4) we obtain

$$\mathbf{k = \frac{f \times channels}{Nyquist}} \quad \mathbf{(11.5)}$$

In this particular case

1. For 0 hz, k = 0.

2. For 270 hz, $k = \frac{270 \times 2048}{22050} \sim 25$

3. For 860 hz, $k = \frac{860 \times 2048}{22050} \sim 80$

In order to process these sounds a batchfile containing the following commands was used:

```
# analysis
sfpvoc -N4096 -A obc obc.dat
#
# synthesis
sfpvoc -N4096 -S -i0 -j24 obc.dat obclow
sfpvoc -N4096 -S -i25 -j80 obc.dat obcmid
sfpvoc -N4096 -S -i81 -j2048 obc.dat obchigh
```

11.7.2Transposition

In several samplers, transposition is achieved by either skipping samples if a sound is to be transposed upwards or by
interpolating between samples to lower a sound.

This however has the same effect as playing a tape at different speeds, thus affecting the duration of a sound and is particularly problematic, for example, when willing to mix a pitched sound and its transpositions to create chords, because the higher transpositions

will have shorter durations.

The Phase Vocoder transposes in the frequency domain by multiplying the fundamental and all the partials by the same factor before resynthesizing the sound, leaving the original duration unaltered. There are also options that allow to keep formant regions unchanged.

Pvoc example 2 contains a *mlflute*, a sampled flute tone, which is followed by a transposition by a factor of 2.1, first using the Phase Vocoder to produce the soundfile *mflute.tr* and then by means of skipping samples using *ftrans* from the GROUCHO package to produce the soundfile *mflute.ftr.* The batchfile used is now listed.

```
# Pvoc Transposition
sfpvoc -N2048 -P2.1 mflute mflute.tr
# Ftrans
ftrans mflute mflute.ftr
```

The transposition produced by skipping samples shortened the sound considerably.

11.7.3Changing the Duration of a Sound

For the reasons mentioned above, if the duration of a sound is altered by skipping or by interpolating samples its pitch will be affected. On the other hand, the Phase Vocoder interpolates frames of the spectrum evolution, causing it to evolve at a slower or faster rate while preserving its contents.

PVoc example 3 contains

1.*ahh*. A recorded cackle.

2. *ahhx4*. Time stretch of *ahh* by a factor of 4.

3. *ahhx16*. Time stretch of *ahhx4* by a factor of 4 (16 times the original duration).

It may be noticed that when the sound is stretched drastically, new artifacts appear, that the original did not seem to have. However, these are not always undesirable, as may be gathered from the previous examples. The batchfile used is now listed:

```
sfpvoc -N4096 -T4.0 ahh ahhx4
sfpvoc -N4096 -T4.0 ahhx4 ahhx16
```

The Phase Vocoder cannot produce dynamic changes in the duration of a sound (acceleration or slowing down). However, there is a program written by Trevor Wishart: *specstr*, which - given a list of breakpoints containing each a time from the beginning of the sound and a stretching factor - can be applied to the analysis file of that sound to achieve this. An example will be given below.

11.7.4Frequency Shifting and Stretching

Two more programs written by Wishart, enable the user to *shift* some or all of the components of a sound. This process multiplies some of the frequencies in the spectrum by the same factor, thus keeping their original ratio.

For example, if a signal of frequencies 100, 200, 300, 400 and 500 hz has its last three partials (ratio 3:4:5) shifted by a factor of 1.5 (the partials at 100 and 200 hz will remain untouched), the resulting signal will have components at

100 hz (unchanged)
200 hz (unchanged)

1.5 x 300 = 450 hz

1.5 x 400 = 600 hz

1.5 x 500 = 750 hz

The ratio between the shifted frequencies 450, 600 and 750 is still 3:4:5.

Alternatively, the frequencies could be *stretched*, which is a process that multiplies successive components by a factor that increases or decreases with frequency. Therefore , the components do not preserve their original ratio.

In the previous example, if the last three partials are stretched by a factor of which is initially 1.3 and increases by 0.3 every 100 hz, the result will be:

100 hz (unchanged)

200 hz (unchanged)

300 x 1.3 = 390 hz

400 x 1.6 = 640 hz

500 x 1.9 = 1800 hz

The ratio between the three stretched frequencies is now 39:64:180 and not 3:4:5.

specsh and *spece* are used to respectively shift and stretch sounds, by applying them to the analysis data produced by the Phase Vocoder. Both of these programs allow a time variable factor.

Pvoc example 4 contains the sound *inh4*, the end of which is first shifted in 3 different ways. This is followed by the same end of the file but this time stretched in 3 different ways.

The process involves the following steps:

1.Cut of beginning and end of *inh4*, saved respectively as *inh4b* and *inh4e*.

2.Analysis of *inh4e*, saved as *inh4e.dat*.

3.Shift of spectrum applying *specsh* to inh4e.dat producing three different versions of data files: *inh4she1.dat*, *inh4she2.dat*, *inh4she3.dat*. The parameters used correspond to the following cases:

a.Shift frequencies *above* channel 15 with a factor that is initially 1 for .1 seconds, then increases to a maximum of 1.9 for .1 seconds and remains like that.
b.The same but with a maximum factor of 2.4.

c. The same but with a maximum factor of 3.45.

4.Stretch of spectrum applying *spece* to inh4e.dat producing three different versions of data files: *inh4ste1.dat*, *inh4ste2.dat*, *inh4ste3.dat*. The parameters used correspond to the following cases:

a.Stretch frequencies *above* channel 15 with a factor that is initially 1 for .06 seconds, then increases for .1 seconds to a value between 1 (for channel 15) to 1.9 (at Nyquist).
It then remains like that.

b.The same but with a maximum factor of 2.4.

c.The same but stretching from channel 4, with a maximum factor of 4.6 at Nyquist.

5.Synthesis of *inh4sh1e*, *inh4sh2e*, *inh4sh3e*, *inh4st1e*, *inh4st2e*, *inh4st3e*, from the respective analysis files created above.

6.Splice of these endings with *inh4b* to create *inh4sh1*, *inh4sh2*, *inh4sh3*, *inh4st1*, *inh4st2* and *inh4st3*.

The batchfile used is shown below:

```
#       cut
cut inh4 inh4b 0 2.3
cut inh4 inh4e 2.3 2.7
#       analysis
sfpvoc -N2048 -A inh4e inh4e.dat
#       shift
specsh inh4e.dat inh4she1.dat 1 15 1.9 .1 .1 0 0
specsh inh4e.dat inh4she2.dat 1 15 2.4 .1 .1 0 0
specsh inh4e.dat inh4she3.dat 1 15 3.45 .1 .1 0 0
#       stretch
spece inh4e.dat inh4ste1.dat 0  15 1.9 .06 .1 0 0 .5
spece inh4e.dat inh4ste2.dat 0  15 2.4 .06 .1 0 0 .5
spece inh4e.dat inh4ste3.dat 0  4 4.6 .06 .1 0 0 .5
#       resynthesis
sfpvoc -N2048 -S inh4she1.dat inh4she1
sfpvoc -N2048 -S inh4she2.dat inh4she2
sfpvoc -N2048 -S inh4she3.dat inh4she3
sfpvoc -N2048 -S inh4ste1.dat inh4ste1
sfpvoc -N2048 -S inh4ste2.dat inh4ste2
sfpvoc -N2048 -S inh4ste3.dat inh4ste3
#       splice
splice -w0 inh4sh1 inh4b inh4she1
splice -w0 inh4sh2 inh4b inh4she2
splice -w0 inh4sh3 inh4b inh4she3
```

```
splice -w0 inh4st1 inh4b inh4ste1
splice -w0 inh4st2 inh4b inh4ste2
splice -w0 inh4st3 inh4b inh4ste3
```

11.7.5 Spectral Interpolation

Vocinte is another program by Wishart that interpolates between the spectrum of one sound and another sound, which, if used judiciously can produce a convincing transformation of one sound into the other.

For instance, *Pvoc* example 5 shows a transformation of an oboe sound into a cry used in *Los Dados Eternos*. First, the oboe sound is played, then the cry, and finally the transformation.

The process is simple and consists of cutting the transition sections of each of sounds, then each section is analysed and a new analysis is produced by using the analysed sections as inputs. Then the interpolated analysis is synthesized. Finally, the latter sound is splice with the beginning of the first sound and the end of the second.

In this particular case, the soundfiles are *obD* and *alhmix4* and the transition takes 2 seconds. The batchfile used is now listed:

```
#cuts
cut obD obDs 0 1.2
cut obD obDe 1.2 3.34
cut alhmix4 alhmix4s 0 2.011
cut alhmix4 alhmix4s 2.011 3.75
#analysis
sfpvoc -N4096 -A obDs obDs.dat
sfpvoc -N4096 -A alhmix4s alhmix4s.dat
#interpolation
vocinte obDs.dat alhmix4s.dat otg1m.dat .01 2 .01 2 2 2
#synthesis
```

```
sfpvoc -N4096 -S otg1m.dat otg1m
#splices
splice -w0 otg1s obDs otg1m
splice otg1 otg1s alhmix4s
```

Spectral interpolation does not guarantee perceptually convincing transformations. Care is needed in order to achieve these. Sometimes a simple solution can solve certain problems. The next example consists of a double transformation from a high oboe note to a squeaking door to a revolving sound, also used in *Los Dados Eternos*.

oboe--> squeaking door --> revolving sound
The initial result was unsuccessful in the first (oboe to door) transformation because of a region where neither one nor the other sound could be identified. In order to correct this, this region was cut off with a simple digital cut and splice program, which solved the problem.

The sounds can be heard in *PVoc* example 6: First comes the oboe note, then the door, then the revolving sound. This is followed by the first (unsuccessful) version. The corrected version is then heard.

A final example of a simultaneous double transformation uses the sounds given in *Pvoc* example 7. A mix of the fundamental and the high harmonics of the oboe low C is converted into one sound while the mid harmonics are converted into another sound. Each of these transformations is then put into a different stereo channel, and both are played together. Because of the characteristics of our auditory perception, left and right channels are added together, so that an oboe sound is heard ... until different harmonics convert into different sounds giving the impression of splitting the oboe into these, each on a different speaker.

PVoc example 7 contains the following sounds:

1.Left channel: Oboe fundamental and high harmonics, followed by second sound, followed by the transformation of one into the other.

2.Right channel: Oboe mid frequencies, followed by another sound followed by the transformation of one into the other.

3.Left and right channel played together. It is worth hearing this while changing the balance between the speakers.

11.7.6Cross-Synthesis

The last example in this chapter uses a technique by which the formants of one sound are applied to the spectrum of another. For instance, formants of speech could be imposed on white noise. a rich sound. Again, a program produced by Wishart, this time called *speccros*, can be used to achieve this.

In the current example, from *Los Dados Eternos*, cross-synthesis, aided by variable time stretch using *specstr*, was used in order to produce a voiced sound that sounds like a small devil distorting the words 'Dominus Deus'.

The process consisted of the following steps:

1.Cross-synthesis of 'Dominus Deus' with a previously processed sampled guiro.

2.Transposition by resampling in order to raise the pitch.

3.Variable time stretch not only to slow down the sound without changing pitch but also in order to give it more gestural edge.

PVoc example 8 contains the following sounds: 'Dominus Deus', processed guiro, cross-synthesis of the first sound applied to the second, transposition by resampling, variable timestretch.

Coda

Computer technology is changing at an increasing pace and its musical applications are no exception. Most of the techniques covered here were produced in a non-real-time system. However signs are that the realisation in a real-time system, enabling their use in performance, is only a few steps away.

New DSP chips are constantly being developed, improving on the previous versions. At the same time, new computer environments, more friendly and powerful, are making it possible to get closer, for the first time, to a system in which the composer does not have to be computer literate to the extent of a programmer or system analyst, unless by choice.

There is one thing, however, that computer technology can do nothing about, and that is our perception, with its limitations. It is up to the musician to take this into account. Mathematical principles can be realised by means of sound, but that does not give them automatically musical sense.

It is the belief of the author that our limitations, far from being a curse, are the most wonderful devices: because of them, illusions can be created, and what is music without illusion?

Appendix

Reverberation Instruments

The following two instruments are implementations of *Csound* that use the REVERB statement in order to implement instruments that produce dynamic reverberation. In other words, the reverberation time can be controlled by a function table, thus changing the characteristics of the environment. *instr 37* does this to a mono file, while *instr 38* works on stereo files, with an independent function table controlling each channel.

```
;       REVERBM.ORC     Reverberates soundfile in MONO

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

instr 37        ;       for MONO files

;p4 : amplitude SCALING FACTOR
;p5 : input file number
;p6 : skip
;p7 : maximum reverberation time
;p8 : reverberation function

        nchnls  = 1                     ; set channels

        icyc    =       1/p3            ; one cycle

        krtime  oscil   p7,icyc,p8      ; variable reverb time
        ain     soundin p5,p6           ; input file
        arev    reverb  ain,krtime,0    ; reverberate
                out     p4*(ain+arev)   ; output original + reverb

endin
```

```
;       REVERBS.ORC     Reverberates soundfile in STEREO

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2


instr 38        ;       STEREO files

;p4 : amplitude SCALING FACTOR
;p5 : input file number
;p6 : skip
;p7 : reverberation time, left channel
;p8 : reverberation function left channel
;p9 : reverberation time, right channel
;p10: reverberation function left channel

        nchnls = 2                      ; set channels

        icyc    =       1/p3            ; one cycle

        krtimel oscil   p7,icyc,p8      ; variable left reverb time
        krtimer oscil   p9,icyc,p10     ; variable left reverb time

        air,ail soundin p5,p6           ; input file
        arevl   reverb  ail,krtimel     ; reverberate left
        arevr   reverb  air,krtimer     ; reverberate right

        ; output original + reverb

                outs    p4*(ail+arevl),p4*(air+arevr)

endin
```

```
; This is a comment, Csound will ignore it
; CS1.ORC       -       Simple Oscillator

; NOTE :        Clicks will be heard since
;               there is no envelope.

sr = 44100
kr =  4410
ksmps = 10
nchnls = 1

instr 1

        a1      oscil   p4, p5, 1       ; oscillator
                out     a1              ; output

endin
```

```
; This is a comment, Csound will ignore it
; CS1.SC

f1 0 512 10 1

i1 0 2 8000 440

e
```

Assignment :Produce a soundfile called CS1.OUT using the orchestra and the score in this example.

```
; CS2.ORC       Simple Oscillator
;             Uses table defined by f1

; NOTE :       Clicks will be heard since
;             there is no envelope.

sr = 44100
kr =  4410
ksmps = 10
nchnls = 1

instr 1

      a1      oscil   p4, p5, 1       ; oscillator
              out     a1              ; output
endin



; CS2.SC

f1 0 512 10 1



;             p3      p4      p5
;instr  start   dur     amp     freq

i1      0       5      8000    261.625
i1      1       4       .      329.627
i1      2       3       .      391.995
i1      3       2       .      466.163
i1      5       1      0       0

s
```

```
f1 0 512 10 1 0 .5 0 .7 0 .9 0 2

;             p3      p4      p5
;instr  start   dur     amp     freq

i1      0       5      8000    261.625
i1      1       4      .       329.627
i1      2       3      .       391.995
i1      3       2      .       466.163
i1      5       1      0       0

s

f1 0 512 10 1 .5 .7 .9 .2 .87 .76

;             p3      p4      p5
;instr  start   dur     amp     freq

i1      0       5      8000    261.625
i1      1       4      .       329.627
i1      2       3      .       391.995
i1      3       2      .       466.163

e
```

```
; CS3.ORC       -       Simple Oscillator
;               Uses table defined by function number
;               indicated in p6.

; NOTE :        Clicks will be heard since
;               there is no envelope.


sr = 44100
kr =  4410
ksmps = 10
nchnls = 1

instr 2

        a1      oscil   p4, p5, p6      ; oscillator
                out     a1              ; output

endin


; CS3.SC

f1 0 512 10 1 0 .5 0 .7 0 .9 0 2
f2 0 512 10 1 .5 .7 .9 .2 .87 .76
f3 0 512 10 1 .7 0 .9 0 .87 0 .76
f4 0 512 10 1

t 0 120

;               p3      p4      p5      p6
;instr  start   dur     amp     freq    func no.

i2      0       10      8000    261.625 1
```

```
i2    2    8    .    329.627 2
i2    4    6    .    391.995 3
i2    6    4    .    466.163 4

e
```

Assignment :Use the previous instrument with an envelope table generated by GEN7.

4. Envelopes

```
; CS4.ORC       -       Simple Oscillators
;                       with envelopes

sr = 44100
kr =  4410
ksmps = 10
nchnls = 1

instr 4

; p4: amplitude
; p5: frequency
; p6: attack
; p7: decay
; p8: function table

        k1      linen   p4, p6, p3, p7  ; envelope
        a1      oscil   k1, p5, p8      ; oscillator
                out     a1              ; output

endin


instr 5

; p4: amplitude
; p5: frequency
; p6: function table

    i1    =       .05*p3                    ;dur1
    i2    =       .15*p3                    ;dur2
    i3    =       .3*p3                     ;dur3
```

```
    i4    =       .5*p3                       ;dur4
    k1    linseg  0,i1,p4,i2,.3*p4,i3,.75*p4,i4,0 ; envelope
    a1    oscil   k1, p5, p6                  ; oscillator
          out     a1                          ; output
endin



instr 6

; p4: amplitude
; p5: frequency
; p6: function table

    i1    =       .05*p3                      ;dur1
    i2    =       .15*p3                      ;dur2
    i3    =       .3*p3                       ;dur3
    i4    =       .5*p3                       ;dur4
    k1    expseg  .0001,i1,p4,i2,.3*p4,i3,.75*p4,i4,.0001 ; envelope
    a1    oscil   k1, p5, p6                  ; oscillator
          out     a1                          ; output
endin
```

```
; CS4.SC

f5 0 512 10 1 .5 .7 .9 .2 .87 .76

;             p3      p4      p5      p6      p7      p8
;instr  start   dur     amp     freq    rise    decay   func

i4      0       2       30000   120     .05     1       5

s

;             p3      p4      p5      p6
;instr  start   dur     amp     freq    func

i5      1       2       30000   120     5

s

;             p3      p4      p5      p6
;instr  start   dur     amp     freq    func

i6      1       2       30000   120     5

e
```

Assignment :Produce a musical phrase that has more than one voice.

5.Some Useful Devices

```
; CS5.ORC       -       Oscillator with variable width
;                       vibrato and swell.

;p4     amplitude
;p5     frequency
;p6     attack
;p7     decay
;p8     function table

sr = 44100
kr =  4410
ksmps = 10
nchnls = 1

instr 7

        i1      =       1/5                     ; max swell
        i2      =       p5/100                  ; max vib width
        i3      =       p3/2                    ; half a cycle
        k1      linen   p4,p6,p3,p7             ; envelope
        k2      line    0, p3, i1               ; swell
        k3      linseg  0, i3, i2, i3, 0        ; vib width
        k4      oscil   k3, 5, 1                ; vibrato
        a1      oscil   k1*(1+k2), p5+k4, p8    ; oscillator
                out     a1                      ; output

endin


; CS5.SC
; First two Kyrie Eleison from Palestrina's Missa Papae Marcelli
```

```
f1 0 8192 10 1

;       SOPRANO
;               p3      p4      p5      p6      p7      p8
;instr  start   dur     amp     freq    attack  decay   func

i7      1       1.5     4000    587.329 .05     .05     1     ; D
i7      +       0.5     .       .       .       .       .     ; D
i7      +       1       .       .       .       .       .     ; D
i7      +       0.75    .       783.991 .       .       .     ; G
i7      +       0.25    .       698.456 .       .       .     ; F
i7      +       .       .       659.255 .       .       .     ; E
i7      +       .       .       587.329 .       .       .     ; D
i7      +       .       .       523.251 .       .       .     ; C
i7      +       .       .       493.883 .       .       .     ; B
i7      +       .       .       440     .       .       .     ; A
i7      +       .       .       391.995 .       .       .     ; G
i7      +       1       .       .       .       .       .     ; G
i7      +       0.5     .       369.994 .       .       .     ; F#
i7      +       1       .       391.995 .       .       .     ; G

; second phrase
i7      +       .       .       587.329 .       .       .     ; D
i7      +       0.75    .       493.883 .       .       .     ; B
i7      +       0.25    .       440     .       .       .     ; A
i7      +       .       .       493.883 .       .       .     ; B
i7      +       .       .       523.251 .       .       .     ; C
i7      +       0.5     .       587.329 .       .       .     ; D
i7      +       0.75    .       659.255 .       .       .     ; E
i7      +       0.25    .       587.329 .       .       .     ; D
i7      +       .       .       523.251 .       .       .     ; C
i7      +       .       .       440     .       .       .     ; A
i7      +       0.75    .       587.329 .       .       .     ; D
```

```
i7     +      0.25   .      523.251 .      .      .    ; C
i7     +      1      .      523.251 .      .      .    ; C
i7     +      0.5    .      493.883 .      .      .    ; B
i7     +      2      .      523.251 .      .      .    ; C

;      ALTO
;              p3     p4     p5     p6     p7     p8
;instr  start  dur    amp    freq   attack decay  func

i7     3      1.5    .      391.995 .      .      .    ; G
i7     +      0.5    .      .      .      .      .    ; G
i7     +      1      .      .      .      .      .    ; G
i7     +      2      .      523.251 .      .      .    ; C
i7     +      .      .      493.883 .      .      .    ; B
; second phrase
i7     10.5   1      .      391.995 .      .      .    ; G
i7     +      0.5    .      .      .      .      .    ; G
i7     +      1      .      .      .      .      .    ; G
i7     +      0.5    .      440    .      .      .    ; A
i7     +      1      .      391.995 .      .      .    ; G
i7     +      0.5    .      349.228 .      .      .    ; F
i7     +      1      .      391.995 .      .      .    ; G
i7     +      2      .      329.627 .      .      .    ; E

;      TENOR 1
;              p3     p4     p5     p6     p7     p8
;instr  start  dur    amp    freq   attack decay  func

i7     0      1.5    4000   293.664 .05    .05    1    ; D
i7     +      0.5    .      .      .      .      .    ; D
i7     +      1      .      .      .      .      .    ; D
i7     +      0.75   .      391.995 .      .      .    ; G
i7     +      0.25   .      349.228 .      .      .    ; F
i7     +      0.5    .      329.627 .      .      .    ; E
```

```
i7    +      .     .     293.664 .      .      .    ; D
i7    +      1.5   .     329.627 .      .      .    ; E
i7    +      0.5   .     293.664 .      .      .    ; D
i7    +      1     .     261.625 .      .      .    ; C
; second phrase
i7    +      1     .     293.664 .      .      .    ; D
i7    +      0.5   .     246.941 .      .      .    ; B
i7    +      .     .     195.997 .      .      .    ; G
i7    +      0.75  .     391.995 .      .      .    ; G
i7    +      0.25  .     349.228 .      .      .    ; F
i7    +      0.5   .     329.627 .      .      .    ; E
i7    +      .     .     293.664 .      .      .    ; D
i7    +      0.5   .     261.625 .      .      .    ; C
i7    +      1     .     329.627 .      .      .    ; E
i7    +      0.5   .     293.664 .      .      .    ; D
i7    +      0.5   .     329.627 .      .      .    ; E
i7    +      0.25  .     293.664 .      .      .    ; D
i7    +      .     .     261.625 .      .      .    ; C
i7    +      1     .     293.664 .      .      .    ; D
i7    +      2     .     261.625 .      .      .    ; C


;      TENOR 2
;             p3     p4     p5     p6     p7     p8
;instr  start  dur    amp    freq   attack decay  func

i7    8      1.5    4000   293.664 .05    .05    1    ; D
i7    +      0.5    .      .       .      .      .    ; D
i7    +      1      .      .       .      .      .    ; D
i7    +      0.75   .      391.995 .      .      .    ; G
i7    +      0.25   .      349.228 .      .      .    ; F
i7    +      .      .      329.627 .      .      .    ; E
i7    +      .      .      293.664 .      .      .    ; D
i7    +      .      .      261.625 .      .      .    ; C
```

```
i7    +     .     .     246.941 .     .     .    ; B
i7    +     0.5   .     220   .     .     .    ; A
i7    +     0.5   .     246.941 .     .     .    ; B
i7    +     1     .     261.625 .     .     .    ; C


;     BASS 1
;           p3    p4    p5    p6    p7    p8
;instr  start  dur   amp   freq  attack decay  func

i7    2     1.5   4000  195.997 .05   .05   1    ; G
i7    +     0.5   .     .     .     .     .    ; G
i7    +     1     .     .     .     .     .    ; G
i7    +     1.5   .     261.625 .     .     .    ; C
i7    +     0.5   .     246.941 .     .     .    ; B
i7    +     1     .     220   .     .     .    ; A
i7    +     2     .     195.997 .     .     .    ; G

; last chord
i7    16    2     .     130.812 .     .     .    ; C

;     BASS 2
;           p3    p4    p5    p6    p7    p8
;instr  start  dur   amp   freq  attack decay  func

i7    9     1.5   4000  195.997 .05   .05   1    ; G
i7    +     0.5   .     .     .     .     .    ; G
i7    +     1     .     .     .     .     .    ; G
i7    +     1.5   .     261.625 .     .     .    ; C
i7    +     0.5   .     246.941 .     .     .    ; B
i7    +     1     .     220   .     .     .    ; A
i7    +     3     .     195.997 .     .     .    ; G

e
```

```
; CS6.ORC       -       Oscillator with variable width
;                       vibrato and swell.
;                       Accepts pitch and converts to frequency.

;p4     amplitude
;p5     frequency
;p6     attack
;p7     decay
;p8     function table

sr = 44100
kr =  4410
ksmps = 10
nchnls = 1

instr 9
        i0      =       cpspch(p5)              ; pitch to freq
        i1      =       1/5                     ; max swell
        i2      =       i0/100                  ; max vib width
        i4      =       p3/2                    ; half a cycle
        k1      linen   p4,p6,p3,p7             ; envelope
        k2      line    0, p3, i1               ; swell
        k3      linseg  0, i3, i2, i3, 0        ; vib width
        k4      oscil   k3, 3, 1                ; vibrato
        a1      oscil   k1*(1+k2), i0+k4, p8    ; oscillator
                out     a1                      ; output

endin


; CS6.SC
; First two Kyrie Eleison from Palestrina's Missa Papae Marcelli

f1 0 8192 10 1
```

```
;       SOPRANO
;               p3      p4      p5      p6      p7      p8
;instr  start   dur     amp     PCH     attack  decay   func

i9      1       1.5     4000    9.02    .05     .05     1   ; D
i9      +       0.5     .       .       .       .       .   ; D
i9      +       1       .       .       .       .       .   ; D
i9      +       0.75    .       9.07    .       .       .   ; G
i9      +       0.25    .       9.05    .       .       .   ; F
i9      +       .       .       9.04    .       .       .   ; E
i9      +       .       .       9.02    .       .       .   ; D
i9      +       .       .       9.00    .       .       .   ; C
i9      +       .       .       8.11    .       .       .   ; B
i9      +       .       .       8.09    .       .       .   ; A
i9      +       .       .       8.07    .       .       .   ; G
i9      +       1       .       .       .       .       .   ; G
i9      +       0.5     .       8.06    .       .       .   ; F#
i9      +       1       .       8.07    .       .       .   ; G
; second phrase
i9      +       .       .       9.02    .       .       .   ; D
i9      +       0.75    .       8.11    .       .       .   ; B
i9      +       0.25    .       8.09    .       .       .   ; A
i9      +       .       .       8.11    .       .       .   ; B
i9      +       .       .       9.00    .       .       .   ; C
i9      +       0.5     .       9.02    .       .       .   ; D
i9      +       0.75    .       9.04    .       .       .   ; E
i9      +       0.25    .       9.02    .       .       .   ; D
i9      +       .       .       9.00    .       .       .   ; C
i9      +       .       .       8.09    .       .       .   ; A
i9      +       0.75    .       9.02    .       .       .   ; D
i9      +       0.25    .       9.00    .       .       .   ; C
i9      +       1       .       9.00    .       .       .   ; C
i9      +       0.5     .       8.11    .       .       .   ; B
```

```
i9     +      2      .      9.00    .      .      .   ; C

;      ALTO
;              p3     p4     p5     p6     p7     p8
;instr  start   dur    amp    PCH    attack  decay   func

i9     3      1.5    .      8.07    .      .      .   ; G
i9     +      0.5    .      .       .      .      .   ; G
i9     +      1      .      .       .      .      .   ; G
i9     +      2      .      9.00    .      .      .   ; C
i9     +      .      .      8.11    .      .      .   ; B
; second phrase
i9     10.5   1      .      8.07    .      .      .   ; G
i9     +      0.5    .      .       .      .      .   ; G
i9     +      1      .      .       .      .      .   ; G
i9     +      0.5    .      8.09    .      .      .   ; A
i9     +      1      .      8.07    .      .      .   ; G
i9     +      0.5    .      8.05    .      .      .   ; F
i9     +      1      .      8.07    .      .      .   ; G
i9     +      2      .      8.04    .      .      .   ; E

;      TENOR 1
;              p3     p4     p5     p6     p7     p8
;instr  start   dur    amp    PCH    attack  decay   func

i9     0      1.5    4000   8.02    .05    .05    1   ; D
i9     +      0.5    .      .       .      .      .   ; D
i9     +      1      .      .       .      .      .   ; D
i9     +      0.75   .      8.07    .      .      .   ; G
i9     +      0.25   .      8.05    .      .      .   ; F
i9     +      0.5    .      8.04    .      .      .   ; E
i9     +      .      .      8.02    .      .      .   ; D
i9     +      1.5    .      8.04    .      .      .   ; E
i9     +      0.5    .      8.02    .      .      .   ; D
```

```
i9    +    1    .    8.00   .    .    .  ; C
; second phrase
i9    +    1    .    8.02   .    .    .  ; D
i9    +    0.5  .    7.11   .    .    .  ; B
i9    +    .    .    7.07   .    .    .  ; G
i9    +    0.75 .    8.07   .    .    .  ; G
i9    +    0.25 .    8.05   .    .    .  ; F
i9    +    0.5  .    8.04   .    .    .  ; E
i9    +    .    .    8.02   .    .    .  ; D
i9    +    0.5  .    8.00   .    .    .  ; C
i9    +    1    .    8.04   .    .    .  ; E
i9    +    0.5  .    8.02   .    .    .  ; D
i9    +    0.5  .    8.04   .    .    .  ; E
i9    +    0.25 .    8.02   .    .    .  ; D
i9    +    .    .    8.00   .    .    .  ; C
i9    +    1    .    8.02   .    .    .  ; D
i9    +    2    .    8.00   .    .    .  ; C


;       TENOR 2
;             p3    p4    p5    p6    p7    p8
;instr  start  dur   amp   PCH   attack decay  func

i9    8    1.5   4000  8.02  .05   .05   1  ; D
i9    +    0.5   .     .     .     .     .  ; D
i9    +    1     .     .     .     .     .  ; D
i9    +    0.75  .     8.07  .     .     .  ; G
i9    +    0.25  .     8.05  .     .     .  ; F
i9    +    .     .     8.04  .     .     .  ; E
i9    +    .     .     8.02  .     .     .  ; D
i9    +    .     .     8.00  .     .     .  ; C
i9    +    .     .     7.11  .     .     .  ; B
i9    +    0.5   .     7.09  .     .     .  ; A
i9    +    0.5   .     7.11  .     .     .  ; B
```

```
i9     +      1      .      8.00   .      .      .   ; C


;      BASS 1
;              p3     p4     p5     p6     p7     p8
;instr  start  dur    amp    PCH    attack decay  func

i9     2      1.5    4000   7.07   .05    .05    1   ; G
i9     +      0.5    .      .      .      .      .   ; G
i9     +      1      .      .      .      .      .   ; G
i9     +      1.5    .      8.00   .      .      .   ; C
i9     +      0.5    .      7.11   .      .      .   ; B
i9     +      1      .      7.09   .      .      .   ; A
i9     +      2      .      7.07   .      .      .   ; G
; last chord
i9     16     2      .      7.00   .      .      .   ; C

;      BASS 2
;              p3     p4     p5     p6     p7     p8
;instr  start  dur    amp    PCH    attack decay  func

i9     9      1.5    4000   7.07   .05    .05    1   ; G
i9     +      0.5    .      .      .      .      .   ; G
i9     +      1      .      .      .      .      .   ; G
i9     +      1.5    .      8.00   .      .      .   ; C
i9     +      0.5    .      7.11   .      .      .   ; B
i9     +      1      .      7.09   .      .      .   ; A
i9     +      3      .      7.07   .      .      .   ; G

e
```

```
; CS7.ORC       -       Oscillator with variable width
;                       vibrato and swell.
;                       Accepts pitch and converts to frequency.
;                          and squeezes or expands the octave above ;                          and below
middle C

;p4     amplitude
;p5     frequency
;p6     attack
;p7     decay
;p8     function table
;p9     squeeze/expansion factor

sr = 44100
kr =  4410
ksmps = 10
nchnls = 1

instr 10

        ist     =       int(p5)*12 + frac(p5)*100 - 96                                  ; semitones
above                                           ; middle C
        itot    =       96 + ist*p9             ; total semitones
        ioc     =       int(itot/12)                    ; octave
        ist     =       itot - ioc*12           ; semitones above
                                        ; middle C
        i0      =       ioc+ist/100             ; to PCH notation
        i0      =       cpspch(i0)              ; PCH to freq
        i1      =       1/5                     ; max swell
        i2      =       i0/100                  ; max vib width
        i3      =       p3/2                    ; half a cycle
        k1      linen   p4,p6,p3,p7             ; envelope
        k2      line    0, p3, i1               ; swell
        k3      linseg  0, i3, i2, i3, 0        ; vib width
```

```
        k4      oscil   k3, 3, 1                ; vibrato
        a1      oscil   k1*(1+k2), i0+k4, p8    ; oscillator
                out     a1                      ; output

endin




; CS8.ORC       -       Oscillator with variable width
;                       vibrato and swell.
;                       Accepts pitch and converts to frequency.
;                       and squeezes or expands the octave above and
;                       below a centre pitch given by p10

;p4     amplitude
;p5     frequency
;p6     attack
;p7     decay
;p8     function table
;p9     squeeze/expansion factor
;p10    centre pitch
```

```
sr = 44100
kr =  4410
ksmps = 10
nchnls = 1

instr 11

      icst    =       int(p10)*12 + frac(p10)*100
                                    ; semitones of
                                    ; centre pitch
      ist     =       int(p5)*12 + frac(p5)*100 - icst                    ; semitones
above                                       ; middle C
      itot    =       icst + ist*p9          ; total semitones
      ioc     =       int(itot/12)                ; octave
      ist     =       itot - ioc*12          ; semitones above
                                    ; middle C
      i0      =       ioc+ist/100             ; to PCH notation
      i0      =       cpspch(i0)              ; PCH to freq
      i1      =       1/5                   ; max swell
      i2      =       i0/100                ; max vib width
      i3      =       p3/2                  ; half a cycle
      k1      linen   p4,p6,p3,p7             ; envelope
      k2      line    0, p3, i1             ; swell
      k3      linseg  0, i3, i2, i3, 0        ; vib width
      k4      oscil   k3, 3, 1              ; vibrato
      a1      oscil   k1*(1+k2), i0+k4, p8    ; oscillator
            out     a1                    ; output

endin
```

Assignment :

1.Produce a score for instrument 11 in which a short musical phrase is played without squeezing/stretching the intervals and then the same phrase with different squeezing/stretching factors.(1 week from now)

2.Produce an instrument that, given a frequency 'fans out' for half of its duration and then 'fans in' for the other half using 3 upper and 3 lower glissandi to frequencies determined by a factor given in the score. (2 weeks from now)

For example, if 400 hz and a factor of .1 are given, the upper glissandi will 'fan' to
440 hz x 1.1 =484 hz
484 hz x 1.1 =532.4 hz
532.4 hz x 1.1 =585.64 hz

and the lower will 'fan' to
484 hz / 1.1 =400 hz
400 hz / 1.1 =363.64 hz
363.64 hz / 1.1 =330.58 hz

Produce a short musical passage using this instrument.

5.Additive Synthesis

```
; CS10.ORC
sr = 44100
kr =  4410
ksmps = 10
nchnls = 1

instr 3         ;       Simple Oscillator
                ;       Uses table defined by function number
                ;       indicated in p8.

        k1      linen   p4, p6, p3, p7  ; envelope
        a1      oscil   k1, p5, p8      ; oscillator
                out     a1              ; output
endin



instr 14        ;       Inharmonic oscillator. Up to 7 partials.

;p4: overall amplitude
;p5: fundamental frequency
;p6, p8, p10, p12, p14, p16, p18: relative amplitudes of partials
;p7, p9, p11, p13, p15, p17, p19: relative frequencies of partials
                        i1      =       p5*p7           ; 1 partial freq
        i2      =       p5*p9           ; 2 partial freq
        i3      =       p5*p11          ; 3 partial freq
        i4      =       p5*p13          ; 4 partial freq
        i5      =       p5*p15          ; 5 partial freq
        i6      =       p5*p17          ; 6 partial freq
        i7      =       p5*p19          ; 7 partial freq
        k1      linen   p4, .1, p3, .1  ; envelope
        a1      oscil   p6, i1, 1       ; 1 partial
        a2      oscil   p8, i2, 1       ; 2 partial
```

```
        a3      oscil   p10, i3, 1      ; 3 partial
        a4      oscil   p12, i4, 1      ; 4 partial
        a5      oscil   p14, i5, 1      ; 5 partial
        a6      oscil   p16, i6, 1      ; 6 partial
        a7      oscil   p18, i7, 1      ; 7 partial

        ; mix and output

                out     k1*(a1+a2+a3+a4+a5+a6+a7)/7 endin


;CS10.SC

f1 0 4096 10 1
f2 0 4096 9 1 1 0    1.41 .8 0    1.89 .9 0   2.3 .7 0   2.6 .65 0            3.2 .93 0   3.5 .94 0

;               p3      p4      p5      p6      p7      p8
;instr  start   dur     amp     freq    rise    decay   func

i3      0       2       20000   300     .1      .1      2
s
```

```
;            p3      p4      p5      p6,p8... p18    p7,p9... p19
;instr  start  dur     amp     fund    relative        relative
;                      overall freq    amplitudes      frequencies

i14     2       2       20000   300     1               1
                                        .8              1.41
                                        .9              1.89
                                        .7              2.3
                                        .65             2.6
                                        .93             3.2
                                        .94             3.5


e




; CS11.ORC

sr = 44100
kr =  4410
ksmps = 10
nchnls = 1

instr 15        ;    Inharmonic oscillator. Up to 7 partials with                ;    vibrato shape
determined by function 2

;p4 : overall amplitude
;p5 : fundamental frequency
;p6, p8, p10, p12, p14, p16, p18: relative amplitudes of partials
;p7, p9, p11, p13, p15, p17, p19: relative frequencies of partials
;p20 : envelope shaper
;p21 : function table
```

```
                    i1      =       p5*p7           ; 1 partial freq
        i2      =       p5*p9           ; 2 partial freq
        i3      =       p5*p11          ; 3 partial freq
        i4      =       p5*p13          ; 4 partial freq
        i5      =       p5*p15          ; 5 partial freq
        i6      =       p5*p17          ; 6 partial freq
        i7      =       p5*p19          ; 7 partial freq
        ifrq1   =       1/p3            ; freq = 1/dur
        ; overall envelope
        k1      linseg  0,p3-.7*p20,p4/4,.3*p20,p4,.4*p20,0
        ; frequency vibrato envelope
        k2      line    1, p3, 0
        k3      oscil   k2*p5/10, 10, 2 ; frequency change

        a1      oscil   p6, i1+k3, p21  ; 1 partial
        a2      oscil   p8, i2+k3, p21  ; 2 partial
        a3      oscil   p10, i3+k3, p21 ; 3 partial
        a4      oscil   p12, i4+k3, p21 ; 4 partial
        a5      oscil   p14, i5+k3, p21 ; 5 partial
        a6      oscil   p16, i6+k3, p21 ; 6 partial
        a7      oscil   p18, i7+k3, p21 ; 7 partial

        ; mix and output
              out     k1*(a1+a2+a3+a4+a5+a6+a7)/7 endin


;CS11.SC

f1 0 4096 10 1
f2 0 4096 10 1
f3 0 4096 10 1 .5 .6 .9 1 .3 .4 .2

t 0 80
```

```
;           p3      p4      p5      p6..p18 p7..p19 p20     p21
;instr  start dur     amp     fund    relat.  relat.  env     func
;                   overall freq    amps    freqs   shaper

i15     0       .5      10000   300     1       1                               .8      1.41
                                        .9      1.89
                                        .7      2.3
                                        .65     2.6
                                        .93     3.2
                                        .94     3.5     .15     1

i15     1.3     .7      10000   300     1       1                               .8      1.41
                                        .9      1.89
                                        .7      2.3
                                        .65     2.6
                                        .93     3.2
                                        .94     3.5     .15     1

i15     1.8     .5      15000   900     1       1                               .8      1.41
                                        .9      1.89
                                        .7      2.3
                                        .65     2.6
                                        .93     3.2
                                        .94     3.5     .15     1

i15     1.9     3       19000   40      1       1                               .8      1.41
                                        .9      1.89
                                        .7      2.3
                                        .65     2.6
                                        .93     3.2
                                        .94     3.5     .8      1

i15     4.4     2       19000   600     1       1                               .8      1.41
                                        .9      1.89
```

```
                    .7      2.3
                    .65     2.6
                    .93     3.2
                    .94     3.5     1       3

e

;CS12.ORC       This produces a simple oscillator with an
;               envelope needed to produce RISSET'S beating
;               harmonics
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

instr 3

;p4 : amplitude
;p5 : fundamental


        k1      linen p4, .5, p3, .5    ; envelope
        a1      oscil   k1, p5, 1       ; oscillator
                out     a1
endin



;       ORRISS.SC
;       This produces the original Risset gliding harmonics

f1 0 8192 10 1 1 1 1 1 1 1 1 1 1 1

;               p3      p4      p5
```

```
;instr  start   dur     amp     fund

i3      0       35      3500    110
i3      .       .       .       110.03
i3      .       .       .       110.06
i3      .       .       .       110.09
i3      .       .       .       110.12
i3      .       .       .       109.97
i3      .       .       .       109.94
i3      .       .       .       109.91
i3      .       .       .       109.88

e
```

6.Subtractive Synthesis

```
; CS13.ORC      -       Use of RAND, RANDH, RANDI

sr = 44100
kr =  4410
ksmps = 10
nchnls = 1


instr 16              ;       uses RAND

;p4  : amplitude
;p5  : not used
;p6  : attack
;p7  : decay

      k1      linen   p4, p6, p3, p7  ; envelope
      a1      rand    k1              ; noise source
              out     a1              ; output
endin


instr 17        ;       uses RANDH

;p4  : amplitude
;p5  : random oscillator frequency
;p6  : attack
;p7  : decay

      k1      linen   p4, p6, p3, p7  ; envelope
      a1      randh   k1,p5           ; noise source
              out     a1              ; output
endin
```

```
instr 18        ;       uses RANDI

;p4  : amplitude
;p5  : random oscillator frequency
;p6  : attack
;p7  : decay

        k1      linen   p4, p6, p3, p7  ; envelope
        a1      randi   k1,p5           ; noise source
                out     a1              ; output
endin
```

```
;CS13.SC        Noise Generators

;             p3     p4     p5     p6     p7     ;instr  start   dur     amp     freq    attack  decay   ;
;       RAND
i16     0       2       10000   0       .1      .1

;       RANDH   400 hz
i17     3       2       10000   400     .1      .1

;       RANDI   400 hz
i18     6       2       10000   400     .1      .1

e




; CS14.ORC      -       Use of BUZZ

sr = 44100
kr =  4410
ksmps = 10
nchnls = 1

instr 19

;p4  : amplitude
;p5  : fundamental
;p6  : attack
;p7  : decay
;p8  : number of partials

        k1      linen   p4, p6, p3, p7          ; envelope
        a1      buzz    k1, p5, p8, 1           ; oscillator
```

```
        out     a1                  ; output
endin


; CS14.SC

f1 0 8192 10 1


;           p3      p4      p5      p6      p7      p8
;instr  start   dur     amp     freq    attack  decay   No. of
;                                                       parts

i19     0       2       30000   40      .1      .1      5
i19     3       2       .       40      .1      .1      15
i19     6       2       .       40      .1      .1      45

e
```

Assignment :Produce a pulse with inharmonic spectrum using BUZZ.

```
; CS15.ORC      -       Use of GBUZZ

sr = 44100
kr =  4410
ksmps = 10
nchnls = 1

instr 20

;p4  : amplitude
;p5  : fundamental
;p6  : attack
;p7  : decay
;p8  : lowest amplitude multiplier
;p9  : highest amplitude multiplier
;p10 : lowest partial number
;p11 : total number of partials

    k2      line    p8, p3, p9          ; changing multiplier      k1      linen   p4, p6, p3, p7       ;
envelope
    a1      gbuzz   k1, p5, k2, p10,p11,1 ; oscillator
            out     a1                  ; output
endin



;CS15.SC

f1 0 8192 9 1 1 90


;          p3  p4    p5    p6   p7    p8   p9    p10     p11
;ins start dur amp   freq  atck decay min  max   lowest  partls
;                                fact fact  part
```

```
i20  0     4   25000 30    .1   .5    .7   1.7   5       30

e


; CS16.ORC      -       FILTERED NOISE
;               White noise is passed through a band-pass filter
;               with varying centre frequency and bandwidth.

sr = 44100
kr =  4410
ksmps = 10
nchnls = 1

instr 21

;p4  : amplitude
;p5  : attack
;p6  : decay
;p7  : minimum centre frequency
;p8  : maximum centre frequency
;p9  : minimum bandwidth in %
;p10 : maximum bandwidth in %

    i1   =       1/p3                  ; one cycle
    i2   =       p9*p7/100             ; minimum bandwidth
    i3   =       p10*p8/100            ; maximum bandwidth
    k1   linen   p4, p5, p3, p6        ; envelope
    k2   oscil   p8-p7, i1, 1          ; varying centre freq
    k3   oscil   i3-i2, i1, 2          ; varying bandwidth      a1   rand    1                    ; random
numbers
    a2   reson   a1, p7+k2, i2+k3, 4     ; filter
    a2   balance a2,a1                 ; power balance
```

```
            out     k1*a2                   ; output
endin



;CS16.SC

;       FROM PITCH TO NOISE SLIDING UP

f1 0 1024 7 0 512 .3 512 1
f2 0 1024 5 .001 512 1 512 .4

;           p3   p4     p5    p6     p7      p8      p9      p10
;ins start  dur  amp    atck  decay  min     max     min     max
;                          centre  centre  BW      BW
;                          freq    freq    in %    in %

i21  0      2   10000   .5     .7    260     650     .1      20
i21  2.9    .1  0       .      .     .       .       .       .

s

;       FROM NOISE TO PITCH SLIDING DOWN

f1 0 1024 7 1 512 .3 512 0
f2 0 1024 5 .4 512 1 512 .001

i21  0      2   10000   .7     .5    260     650     .1      20

e
```

```
; CS17.ORC      -       FILTERED PULSE WITH FORMANTS
;A pulse is passed through 4 parallel bandpass ;filters with varying centre frequency and
;bandwidth.
sr = 44100
kr =  4410
ksmps = 10
nchnls = 1

instr 22

;p4  : amplitude
;p5  : fundamental
;p6  : attack
;p7  : decay
;p8  : minimum frequency of RANDI
;p9  : maximum frequency of RANDI

    ip3  =   1/p3              ; one cycle
    ipfl =   p5/5              ; pitch fluctuation
    iffl =   p9-p8             ; freq fluctuation of RANDI
    inh  =   int(sr/2/(p5+ipfl)) ; maximum harmonics

    kenv     linen p4,p6,p3,p7        ; envelope

    krand    oscil .5, ip3, 2       ; oscil between -.5 and .5
    krand    =     krand + .5       ; correct between 0 and 1

    kfrnd    =     p8+iffl*krand    ; actual frequency of RANDI
    kprnd    =     p5+ipfl*krand    ; variable pitch

    a1       buzz  1,kprnd,inh,1    ; pulse
    abal     oscil 1,p5,1           ; balancing sine wave

    k1       randi 1,kfrnd,.12      ; random generator      k2       randi 1,kfrnd,.23      ; for each
```

```
filter
    k3      randi 1,kfrnd,.34     ; each with a     k4      randi 1,kfrnd,.45     ; different seed
    k5      randi 1,kfrnd,.56     ;
    a2 reson a1, 500+k1*100, 30*(1+k1), 0   ; 500 hz bandpass
    a3 reson a1, 1000+k2*200, 40*(1+k2), 0  ; 1000 hz bandpass
    a4 reson a1, 2000+k3*300, 80*(1+k3), 0  ; 2000 hz bandpass
    a5 reson a1, 3500+k4*500, 200*(1+k4), 0 ; 3500 hz bandpass
    a6 reson a1, 4800+k5*800, 400*(1+k5), 0 ; 4800 hz bandpass

    a7  =      a2+a3+a4+a5+a6  ; mix
    a7  balance a7,abal          ; balance intensity
        out kenv*a7              ; output
endin


;CS17.SC

f1 0 4096 10 1
f2 0 512 7 0 50 .6 50 .8 50 .3 50 .7 50 .9 50 .4 50 .1 50 .01 62 .2

;          p3     p4     p5     p6     p7     p8     p9
;ins start  dur    amp    fund.  attack decay  min    max
;                         freq                random freqs.

i22  0      .5    5000   70     .1     .1    10     15

i22  1      5     2500   490    4      .5    40     45
i22  1      5     2500   615    4      .5    40     45
i22  3.5    7     2500   60     .5     1.5   3      50

e
```

```
; CS18.ORC
; Filters the output of BUZZ and gives it an  amplitude envelope

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

instr 23

;p4 : amplitude factor
;p5 : fundamental
;p6 : number of partials
;p7 : frequency of envelope
;p8 : envelope function
;p9 : centre frequency of band-pass filter
;p10: bandwidth as a percentage of centre frequency

        ibw     =       p8*p7/100       ; % bandwidth to hz
        kenv    oscil   p4, p7, p8      ; envelope
        ain     buzz    20000, p5, p6, 2; pulse generator
        afilt   reson   ain, p9, p10, 1 ; filter
                out     kenv*afilt
endin
```

```
; CS18.SC-Uses f1 as an envelope

f1 0 1024 9 0.001 1 0 1 2 90 2 2 90 3 2 90 4 2 90
;f2 0 8192 10 1
f2 0 512 10 1

;          p3   p4     p5     p6     p7     p8    p9      p10
;ins  start dur  ampl   fund   No of  envel  envel centre  bw in  ;                factor freq   part
freq   func  freq   % of  ;                                        centre

i23   0     35   100    20     250    .03    1     100     1
i23   .     .    .      .      .      .06    .     200     .
i23   .     .    .      .      .      .09    .     300     .
i23   .     .    .      .      .      .12    .     400     .
i23   .     .    .      .      .      .15    .     500     .
i23   .     .    .      .      .      .18    .     600     .
i23   .     .    .      .      .      .21    .     700     .
i23   .     .    .      .      .      .24    .     800     .
i23   .     .    .      .      .      .27    .     900     .
i23   .     .    .      .      .      .30    .     1000    .
e




; CS19.ORC      ; Filter soundfile SOUNDIN.NNN and gives it an envelope.
; The bandwidth of the filter fluctuates according to function 2

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
```

```
instr 24

;p4 : skip
;p5 : frequency of envelope
;p6 : envelope function
;p7 : centre frequency of band-pass filter
;p8 : minimum bandwidth
;p9 : maximum bandwidth
;p10: scaling of filter
;p11: input file

    icyc   =       1/p3                ; one cycle
    iminbw =       p8*p7/100           ; min % bandwidth to hz
    imaxbw =       p9*p7/100           ; max % bandwidth to hz
    kenv   oscil   p10, p5, p6         ; envelope
    kbw    oscil   imaxbw-iminbw, icyc, 2 ; bandwidth fluctuation
    ain    soundin p11, p4             ; input soundfile
    afilt  reson   ain, p7, iminbw+kbw, 1 ; filter
           out     kenv*afilt

endin
```

8.Amplitude Modulation (AM)

```
; CS20.ORC      -       Linear and Non-linear amplifiers

sr = 44100
kr =  4410
ksmps = 10
nchnls = 1

instr 25        ;       linear  amplifier y = Ax

;p4  : amplitude
;p5  : frequency
;p6  : attack
;p7  : decay
;p8  : function
;p9  : amplitude factor A

        kenv    linen   1,p6,p3,p7      ; envelope
        ain     oscil   1,p5,p8         ; oscillator
        aout    =       p4*ain          ; y = Ax
                out     kenv*aout       ; output
endin


instr 26        ;                               2
                ;       non-linear amplifier y = Ax

;p4  : amplitude factor A
;p5  : frequency
;p6  : attack
;p7  : decay
;p8  : function
```

```
        kenv    linen   1,p6,p3,p7      ; envelope
        ain     oscil   1,p5,p8         ; oscillator
                                ;   2
        aout    =       p4*ain*ain      ; y = Ax
                out     kenv*aout       ; output
endin


;CS20.SC

f1 0 8192 10 1

;               p3      p4      p5      p6      p7      p8
;instr  start   dur     amplif  freq    attack  decay   func
;                       factor
;linear amplifier
i25     0       2       25000   440     .3      .3      1

;non-linear amplifier
i26     3       2       25000   440     .3      .3      1

e
```

```
; CS21.ORC      -       Amplitude Modulator

sr = 44100
kr =  4410
ksmps = 10
nchnls = 1

instr 3         ; Used to play pure sinewaves

        k1      linen   p4, p6, p3, p7  ; envelope
        a1      oscil   k1, p5, p8      ; oscillator
                out     a1              ; output
endin



instr 27        ;       Side bands only

;p4  : amplitude
;p5  : carrier
;p6  : attack
;p7  : decay
;p8  : modulator
;p9  : function

        kenv    linen   p4, p6, p3, p7  ; envelope
        acarr   oscil   1, p5, p9       ; carrier
        amod    oscil   1, p8, p9       ; modulator
        aout    =       kenv*acarr*amod ; modulation
                out     aout            ; output
endin
```

```
; CS21.SC

f1 0 4096 10 1


;SECTION 1      carrier 300 hz, modulator 15 hz

;              p3      p4      p5      p6      p7      p8
;instr  start   dur     amp     carr.   attack  decay   func

i3      0       2      25000   300     .3      .3      1
i3      3       2      25000    15     .3      .3      1

;           p3   p4      p5      p6      p7      p8      p9
;instr start dur  amp     carr.   attack  decay   mod.    func.

i27     6     2    25000   300     .3      .3      15      1
s
```

```
;SECTION 2      carrier 300 hz, modulator 110 hz

;              p3      p4      p5      p6      p7      p8
;instr  start   dur     amp     carr.   attack  decay   func

i3      1       2       25000   300     .3      .3      1
i3      4       2       25000   110     .3      .3      1

;           p3   p4      p5      p6      p7      p8      p9
;instr start dur  amp     carr.   attack  decay   mod.    func.

i27     7       2       25000   300     .3      .3      110     1
s


;SECTION 3      carrier 300 hz, modulator 310 hz

;              p3      p4      p5      p6      p7      p8
;instr  start   dur     amp     carr.   attack  decay   func

i3      1       2       25000   300     .3      .3      1
i3      4       2       25000   310     .3      .3      1

;           p3   p4      p5      p6      p7      p8      p9
;instr start dur  amp     carr.   attack  decay   mod.    func.

i27     7       2       25000   300     .3      .3      310     1
s


;SECTION 4      carrier 300 hz, modulator 500 hz

;              p3      p4      p5      p6      p7      p8
;instr  start   dur     amp     carr.   attack  decay   func
```

```
i3      1       2       25000   300     .3      .3      1
i3      4       2       25000   500     .3      .3      1

;               p3   p4      p5      p6      p7      p8      p9
;instr start dur  amp     carr.   attack  decay   mod.    func.

i27     7       2       25000   300     .3      .3      500     1

e
```

```
; CS22.ORC      -       Amplitude Modulator

sr = 44100
kr =  4410
ksmps = 10
nchnls = 1


instr 28        ;       Side bands and carrier
                ;       carrier and modulator with              ;       different generating functions

;p4  : amplitude
;p5  : carrier
;p6  : attack
;p7  : decay
;p8  : modulator frequency
;p9  : percentage of carrier to be modulated
;p10 : carrier function table
;p11 : modulator table
```

```
    imod   =     p9/100.00            ; modulated percentage
    inomod =     1 - imod             ; unmodulated percentage

    kenv   linen p4, p6, p3, p7       ; envelope

    acarr  oscil 1, p5, p10           ; carrier
    amod   oscil 1, p8, p11           ; modulator

    aoutm  =     acarr*amod*imod      ; modulated signal
    aoutnm =     acarr*inomod         ; unmodulated signal
           out   kenv*(aoutm + aoutnm) ; output
endin



; CS22.SC

f1 0 4096 10 1


;          p3  p4     p5    p6   p7    p8    p9     p10     p11
;ins  start dur amp    carr. atck decay mod   mod %  carr    mod
;                                        func    func

i28   0     2   25000  600   .3   .3    15    90     1       1
i28   3     .   .      .     .    .     .     10     1       1

e
```

```
;CS23.SC

f1 0 4096 10 1
f2 0 4096 10 1 .8 .7 .6 .5

;          p3  p4     p5    p6   p7    p8    p9     p10     p11
;ins  start dur amp    carr. atck decay mod   mod %  carr    mod
;                                        func    func

i28   0     2   25000  300   .3   .3    212   100    1       2

e




; CS24.ORC      -       Amplitude Modulator

sr = 44100
kr =  4410
ksmps = 10
nchnls = 1

instr 29        ;       Side bands and carrier
                ;       carrier and modulator with              ;       different generating functions
;p4  : amplitude
;p5  : carrier
;p6  : attack
;p7  : decay
;p8  : carrier to modulator ratio. Should not be 0.
;p9  : percentage of carrier to be modulated
;p10 : carrier function table
;p11 : modulator table
```

```
    imod   =     p9/100.00            ; modulated percentage
    inomod =     1 - imod             ; unmodulated percentage

    imf    =     p5/p8                ; modulator frequency

    kenv   linen p4, p6, p3, p7       ; envelope

    acarr  oscil 1, p5, p10           ; carrier
    amod   oscil 1, imf, p11          ; modulator

    aoutm  =     acarr*amod*imod      ; modulated signal
    aoutnm =     acarr*inomod         ; unmodulated signal
           out   kenv*(aoutm + aoutnm) ; output
endin


;CS24.SC

f1 0 4096 10 .9 1 .78 .07 .24 .53 .09 .46
f2 0 4096 10 1 .46 .56 .87 .35

;          p3   p4     p5    p6   p7  p8     p9      p10    p11
;ins start dur  amp    carr. atck dec c/m    mod %   carr   mod
;                                        func   func

i29  0     1.5  25000  250   .1   1    .5     95      1      2
i29  2.5   .    .      .     .    .    2      .       .      .
i29  5     .    .      .     .    .    .501   .       .      .
i29  7.5   .    .      .     .    .    .35355 .       .      .

e
```

```
; CS25.ORC      -       Ring Modulator

sr = 44100
kr =  4410
ksmps = 10
nchnls = 1

instr 30        ;    Variable parameter amplitude modulation
                ;       instrument.
                ;    The following can be made to change in time:
                ;       overall amplitude, carrier frequency,
                ;       c/m ratio, percentage of the carrier to                 ;       be modulated. ;p4  :
max amplitude
;p5  : highest carrier pitch
;p6  : max carrier to modulator ratio (c/m)
;p7  : function table controlling amplitude
;p8  : function table controlling carrier pitch
;p9  : function table controlling c/m ratio
;p10 : function table controlling modulation percentage

        ;       one cycle
        ip3     =       1.0/p3

        ;       pitch frequency convertion
        ifr     =       cpspch(p5)

        ; envelopes
        kamp    oscil   p4,ip3,p7       ; amplitude
        kcar    oscil   ifr,ip3,p8      ; carrier freq
        kcmr    oscil   p6,ip3,p9       ; c/m
        kmp     oscil   1,ip3,p10       ; modulation percentage

        acarr   oscil   1, kcar, 10     ; carrier
        amod    oscil   1, kcar/kcmr,11 ; modulator
```

```
        aoutm   =       acarr*amod*kmp  ; modulated signal
        aoutnm  =       acarr*(1-kmp)   ; unmodulated signal

        ;       mix and output
                out     kamp*(aoutm+aoutnm)
endin


; CS25.SC

;SECTION 1

t 0 120 4 120 6 140 6 250 8.6 40 8.8 60

;       percussive sound functions

f1  0 512 7 0    32     1       64      .3      384     0
f2  0 512 7 1   512     1 f3  0 512 7 0.6799    32      .6799  160     1     384    .9808
f4  0 512 7 1    32     .1      414     .11

;       carrier and modulator functions

f10 0 4096 10 1                         ; carrier
f11 0 4096 10 1 .9 .8 .7 .6 .5 .6 .7 .8         ; modulator

;          p3   p4     p5      p6      p7    p8    p9    p10
;ins start dur  max    highest max     amp   carr  c/m   mod %
;              amp    carr.   c/m     func  func  func  func
;                     pitch

i30  0     .1   5000   9.03    3       1     2     3     4
i30   .2   .    1000    .      3       .     .     .     .
i30   .4   .    >       .      3       .     .     .     .
i30   .6   .    >       .      3       .     .     .     .
```

i30 .8 . > . 3 . . . .

i30 1 . 25000 . 3 . . . .

i30 2 . . 10.02 3 . . . .

i30 3.5 . 15000 9.07 3 . . . .
i30 3.7 . 15000 . 3 . . . .
i30 3.85 . 15000 . 3 . . . .

i30 4 . 15000 10.04 3 . . . .
i30 4.4 . > 9.10 3 . . . .
i30 4.8 . > 10.06 3 . . . .
i30 5.2 . > 10.02 3 . . . .
i30 5.6 . > 10.08 3 . . . .

i30 6 . 32000 11.01 3 . . . .
i30 6.2 . . . 3 . . . .
i30 6.4 . . . 3 . . . .
i30 6.6 . > . 3 . . . .
i30 6.8 . > . 3 . . . .
i30 7.0 . > . 3 . . . .
i30 7.2 .09 > . 3 . . . .
i30 7.4 .085 > . 3 . . . .
i30 7.6 .08 > . 3 . . . .
i30 7.8 .075 > . 3 . . . .
i30 8.0 .07 > . 3 . . . .
i30 8.2 .065 > . 3 . . . .
i30 8.4 .06 > . 3 . . . .
i30 8.6 .05 8000 . 3 . . . .

s

```
;SECTION 2

;       low 'bounce' functions

f1  0 1024 7    0      32  1      64   .6   128  .3    800   0
f2  0 512 7     1     512  1 f3  0 512 7     .7057  64  .8982  64    1    384  .9980
f4  0 512 7     1      64  .6     64   .3   128  0

;       gliss and fan sound functions

f5  0 512 7 0  128  0.2   64  1       32  0.8 96  0.7  192  0
f6   0 512 7 1    64   1.02   64   .95      64   0.8 64   0.6   256   0.64 f7   0 512 7 1   128   1       0
0.7001 384  0.7
f8  0 512 7 0   64  1     64  0      192  1   64  0.8  128  1

;       carrier and modulator functions

f10 0 4096 10 1                         ; carrier
f11 0 4096 10 1 .9 .8 .7 .6 .5 .6               ; modulator

;          p3   p4     p5      p6      p7    p8    p9    p10
;ins start dur  max    highest max     amp   carr  c/m   mod %
;               amp    carr.   c/m     func  func  func  func
;                      pitch

i30  1     1    30000  6.01    1       1     2     3     4
i30  1.75  9    30000  9.01    1       5     6     7     8

s

; SECTION 3

f1  0 2048 7    0    4 1    128  .6  280  .2  1200  0
f2  0 512 7     1  512 1 f3  0 512 7     1  512 1
```

```
f4  0 512 7     1  100 .65  100  .4  312  .05

;       carrier and modulator functions

f10 0 4096 10 1                         ; carrier
f11 0 4096 10 1 .9 .8 .7 .6             ; modulator

;          p3   p4     p5      p6      p7    p8    p9    p10
;ins start dur  max    highest max     amp   carr  c/m   mod %
;              amp    carr.   c/m     func  func  func  func
;                     pitch

i30  1     4    30000  8.06    1.4142  1     2     3     4

e
```

9. Waveshaping

```
; CS26.ORC      -       Waveshaper

sr = 44100
kr =  4410
ksmps = 10
nchnls = 1

instr 31        ;       Basic Waveshaping instrument

;p4 : amplitude
;p5 : frequency
;p6 : attack
;p7 : decay
;p8 : oscillator function
;p9 : waveshaping function

        ioffset =       .499            ; offset
        k1      linen   p4, p6, p3, p7  ; envelope
        a1      oscil   ioffset, p5, p8 ; oscillator
        awsh    table   a1,p9,1,ioffset ; waveshaping value
                out     k1*awsh         ; output
endin



; CS26.SC

;       oscillator

f1 0 8192 10 1
```

```
;       waveshaping functions

f2 0 8193 7 -1 8193  1
f3 0 8193 7 -1 2048 -0.7  4097   0.7  2048  1
f4 0 8193 7 -1 2048 -0.5     0  -0.3  2048  0.1 0 0.4 2048 0.5 0 0.7 2049  1
f5 0 8193 9  1 1 180  2 .9 180   3 .8 180   4 .7 180   5 .6 180
f6 0 8193 9  1 1 90   2 .9  90   3 .8  90   4 .7  90   5 .6 90
 ;          p3      p4      p5      p6      p7      p8      p9
;ins  start dur     amp     freq    attack  decay   osc     wsh
;                                           func    func

i31   0     1      32000    440    .1      .1      1       2
i31   1.5   1      .        .      .1      .1      1       3
i31   3     1      .        .      .1      .1      1       4
i31   4.5   1      .        .      .1      .1      1       5
i31   6     1      .        .      .1      .1      1       6

e
```

```
; CS27.ORC

sr = 44100
kr =  4410
ksmps = 10
nchnls = 1

instr 3         ;       Simple Oscillator

        k1      linen   p4, p6, p3, p7  ; envelope
        a1      oscil   k1, p5, p8      ; oscillator
                out     a1              ; output
endin


instr 31        ;       Basic Waveshaping instrument

;p4 : amplitude
;p5 : frequency
;p6 : attack
;p7 : decay
;p8 : oscillator function
;p9 : waveshaping function

        ioffset =       .5              ; offset
        k1      linen   p4, p6, p3, p7   ; envelope
        a1      oscil   ioffset, p5, p8  ; oscillator
        awsh    table   a1,p9,1,ioffset  ; waveshaping value
                out     k1*awsh         ; output
endin



; CS27.SC
```

```
; Use of GEN13 to produce a harmonic series out of a sine wave

f1  0 8192 10 1                    ; sine wave

f2  0 8193 13 1 1 1                ; DC
f3  0 8193 13 1 1 0 1              ; 1 harmonic (fund)
f4  0 8193 13 1 1 0 0 1            ; 2 harmonic
f5  0 8193 13 1 1 0 0 0 1          ; 3 harmonic
f6  0 8193 13 1 1 0 0 0 0 1        ; 4 harmonic
f7  0 8193 13 1 1 0 0 0 0 0 1      ; 5 harmonic
f8  0 8193 13 1 1 0 0 0 0 0 0 1    ; 6 harmonic
f9  0 8193 13 1 1 0 0 0 0 0 0 0 1  ; 7 harmonic
f10 0 8193 13 1 1 0 0 0 0 0 0 0 0 1 ; 8 harmonic
f11 0 8193 13 1 1 0 0 0 0 0 0 0 0 0 1 ; 9 harmonic

;       SINE WAVE

;             p3     p4     p5     p6     p7     p8
;instr  start  dur    amp    first  attack decay  func
;                           freq

i3     0      1     25000   200    .1     .1     1
```

```
;       HARMONICS

;          p3      p4      p5      p6      p7      p8      p9
;ins  start dur     amp     freq    attack  decay   osc     wsh
;                                          func    func

i31   1    .5     25000    200    .1      .1      1       2
i31   +    .      .        .      .       .       .       3
i31   +    .      .        .      .       .       .       4
i31   +    .      .        .      .       .       .       5
i31   +    .      .        .      .       .       .       6
i31   +    .      .        .      .       .       .       7
i31   +    .      .        .      .       .       .       8
i31   +    .      .        .      .       .       .       9
i31   +    .      .        .      .       .       .       10
i31   +    .      .        .      .       .       .       11

e




; CS28.ORC

sr = 44100
kr =  4410
ksmps = 10
nchnls = 1

instr 32        ;       Dynamic Waveshaping instrument
                ;       Distortion index changes according to
                ;       function indicated by p11
;p4 : amplitude
```

```
;p5 : frequency
;p6 : oscillator function
;p7 : waveshaping function
;p8 : envelope (distortion index) function

    ifr     =       cpspch(p5)          ; pitch to freq
    ip3     =       1/p3                ; one cycle
    ioffset =       .5                  ; offset
    k1      oscil   1, ip3, p8          ; envelope (dist index)
    a1      oscil   ioffset, ifr, p6    ; oscillator
    awsh    table   k1*a1,p7,1,ioffset ; waveshaping value
            out     k1*p4*awsh          ; max amp & output
endin


; CS28.SC

f1  0 8192  10 1                                ; sine wave
f2  0 8193  13 1 1 0 1 .7 -.8 -.3 .1 .8 -.9 -1 1 ; waveshaper
f3  0  512   7 0 64 1 64 .2 184 .5 152 1 48 0    ; dist idx env

;              p3      p4      p5      p6      p7      p8
;instr  start   dur     amp     pitch   osc     wsh     index
;                                       func    func    func

i32     0       1       15000   7.06    1       2       3
i32     +       0.5     .       8.00    .       .       .
i32     +       0.25    .       7.01    .       .       .
i32     +       0.25    .       6.02    .       .       .
i32     2.5     2.5     20000   5.08    .       .       .

e
```

```
; CS29.ORC

sr = 44100
kr =  4410
ksmps = 10
nchnls = 1

instr 33        ;    Dynamic Waveshaping instrument
                ;    changes linearly between two different
                ;    transfer functions that waveshape the same
                ;    signal

;p4  : amplitude
;p5  : frequency
;p6  : oscillator function
;p7  : beginning waveshaping function
;p8  : final waveshaping function
;p9  : index function
;p10 : c/m ratio
    ip3     =       1/p3                ; one cycle
    icarr   =       cpspch(p5)          ; carrier freq
    imod    =       icarr*p10           ; modulator freq
    ioffset =       .5                  ; offset

    k1      oscil   1, ip3, p9          ; envelope
    kmix    line    1, p3, 0            ; mix proportions

    acarr   oscil   .5, icarr, 1        ; carrier
    amod    oscil   1, imod,p6          ; modulator
    a1      =       acarr*amod          ; modulated signal

    awsh1   table   k1*a1,p7,1,ioffset  ; 1 waveshaping value
    awsh2   table   k1*a1,p8,1,ioffset  ; 2 waveshaping value
```

```
    ; mix output
          out     k1*p4*( kmix*awsh1 + (1-kmix)*awsh2 )
endin

; CS29.SC

f1 0 8192 10 1                              ; carrier
f2 0 8192 10 1 .7 .3 .8 .4                    ; modulator

f3 0 8193 13 1 1 0 1 -.9 -.8 .7 .6 -.5 -.4 .3 .2 ; waveshaper 1
f4 0 8193 13 1 1 0 .1 .1 .2 1 .3 .2 .1          ; waveshaper 2

f5 0 1024 9 .5 1 0                           ; envelope

;         p3  p4      p5      p6      p7      p8    p9    p10
;ins strt dur amp     pitch   mod     first   last  index ring
;                          func    wsh     wsh   func  c/m
;                                  func    func  (env)

i33  0    5   25000   7.08    2       3       4     5     .35355
i33  6    2   .       7.06    .       .       .     .     .354
i33  7    6   .       8.05    .       .       .     .     .35355
e
```

10. Frequency Modulation

```
; CS30.ORC      Basic FM instrument

sr = 44100
kr =  4410
ksmps = 10
nchnls = 1

instr 34        ; Basic FM instrument

;p4  : carrier amplitude
;p5  : carrier frequency
;p6  : attack
;p7  : decay
;p8  : peak deviation
;p9  : modulator frequency
;p10 : oscillator function

        kenv    linen   p4,p6,p3,p7        ; envelope
        amod    oscil   p8,p9,p10          ; modulator
        aout    oscil   kenv,p5+amod,p10    ; modulated carrier
                out     aout               ; output
endin


; CS30.SC        Modulation of a 400 hz sine wave by a ;                440 hz sinewave and
different deviation ;              values

f1  0 8192 10 1 ;sine wave

;          p3   p4     p5    p6    p7      p8      p9      p10
;ins  start dur  amp    carr  atck  decay   max     mod     func
```

```
;                   freq              dev     freq

i34   0     1.5  15000  400   .1    .1        0    440     1
i34   2      .   .      .     .     .        10    .       .
i34   4      .   .      .     .     .       100    .       .
i34   6      .   .      .     .     .       800    .       .
i34   8      .   .      .     .     .      1600    .       .
e
```

```
; CS31.SC       Different carrier-modulator combinations

f1  0 8192 10 1 ;sine wave

;          p3   p4     p5    p6    p7      p8      p9      p10
;ins  start dur  amp    carr  atck  decay   max     mod     func
;                   freq              dev     freq

i34   0     1.5  15000  400   .1    .1      400     200     1
i34   2      .   .      200   .     .       800     400     .
i34   4      .   .      200   .     .       566     283     .
e
```

```
; CS32.SC       Same carrier (400 hz) with two different
;              modulators (400 and 800 hz)


f1  0 8192 10 1 ;sine wave

;          p3   p4     p5    p6    p7      p8      p9      p10
;ins  start dur  amp    carr  atck  decay   max     mod     func
;                    freq              dev     freq

i34   0     1.5  15000  400   .1    .1      1200    400     1
i34   2      .   .      .     .     .       2400    800     .
e




; CS33.SC       carrier = 300 hz
;              c/m =   1       (modulator = 300)
;                      1.003   (modulator = 300.9)
;                      1.4142  (modulator = 424.26)
;              index = 2

f1  0 8192 10 1 ;sine wave

;          p3   p4     p5    p6    p7      p8      p9      p10
;ins  start dur  amp    carr  atck  decay   max     mod     func
;                    freq              dev     freq

i34   0     1.5  15000  300   .1    .3      600     300     1
i34   2      .   .      .     .     .       601.8   300.9   .
i34   4      .   .      .     .     .       848.52  424.26  .

e
```

```
; CS34.ORC      -       FM synthesis instrument using FOSCIL
;                      Produces dynamic spectrum by varying
;                      the modulation index

sr = 44100
kr =  4410
ksmps = 10
nchnls = 1

instr 35        ;       General FM instrument

;p4  : amplitude
;p5  : pitch (to be converted to frequency)
;p6  : carrier/frequency
;p7  : modulator/frequency
;p8  : maximum index
;p9  : oscillator function (usually a sine wave)
;p10 : index function

        icyc    =       1/p3            ; one cycle

        ; envelope with attack = decay = .1 sec
        kenv    linen   p4,.1,p3,.1

        ; index
        kidx    oscil   p8,icyc,p10

        ; FM oscillator
        a1      foscil  kenv,p5,p6,p7,kidx,p9

        ; output
                out     a1
endin
```

```
; CS34.SC       dynamic spectrum index changes ;              from 0 to 4 in 2 sec

f1 0 8192 10 1          ; oscillator function
f2 0 512  7 0 512 1     ; index function

;          p3   p4      p5      p6      p7      p8     p9     p10
;ins start dur  amp     freq    carr    mod     max    functions
;                                        idx    osc    idx

i35  0     2    15000   100     1       1       4      1      2

e

; CS35.ORC      -       FM synthesis instrument using FOSCIL
;                       Produces all kinds of percussive sounds

sr = 44100
kr =  4410
ksmps = 10
nchnls = 1

instr 36        ;  General FM instrument, with envelope changing
                ;  according to function table given by p10 and
                ;  index changing according to function table given
                ;  by p11.
;p4  : amplitude
;p5  : pitch (to be converted to frequency)
;p6  : carrier/frequency
;p7  : modulator/frequency
;p8  : maximum index
;p9  : oscillator function (usually a sine wave)
;p10 : amplitude function
```

```
;p11 : index function

        icyc    =       1/p3            ; one cycle
        ifreq   =       cpspch(p5)      ; frequency

        ; envelope
        kenv    oscil   p4,icyc,p10

        ; index
        kidx    oscil   p8,icyc,p11

        ; FM oscillator
        a1      foscil  kenv,ifreq,p6,p7,kidx,p9

        ; output
                out     a1
endin

; CS35.SC       Different Percussive sounds obtained by changing
;               envelopes, index functions and durations of the ; sounds

f1 0 8192 10 1          ; oscillator function
f2 0 512  5 1 512 .0001 ; amplitude function
f3 0 512  5 1 512 .2    ; index function

;       BELLS

;          p3   p4      p5      p6      p7      p8    p9  p10 p11
;ins start dur  amp     pitch   carr    mod     max  functions
;                                        idx  osc amp idx

i36   0    4    15000   7.11    1       1.215   6    1   2   3

i36   5    .    10000   8.11    .       .       .    .   .   .       i36   6    .    10000   8.07    .       .       .
```

```
. . .    i36  7   .  10000  8.09   .    .    .    .    .    i36  8   .  10000  8.02   .
.    .    .    .    .    i36  10   .  10000  8.02   .    .    .    .    .    i36  11   .  10000
8.09   .    .    .    .    .    i36  12   .  10000  8.11   .    .    .    .    .    i36  13
.  10000  8.07   .    .    .    .    .
s

;      DRUMS

;index and amplitude function
f2 0 2048 7 0 20 .75 35 .9 35 1 104 .15 64 .1 186 .08 630 .15 964 0

;         p3   p4     p5     p6     p7     p8   p9  p10 p11
;ins start dur  amp    pitch  carr   mod    max  functions
;                                    idx  osc amp idx

i36  0    .5   15000  5.07   1      2.21   2.5  1   2   2
i36  +    .5   15000  6      .      .      .    .   .
i36  +    .5   15000  6.05   .      .      .    .   .
s

;      KNOCK ON WOOD

f2 0 512  5 1 512 .0001        ; amplitude function
f3 0 512  7 1 64 0 448 0       ; index function

;         p3   p4     p5     p6     p7     p8   p9  p10 p11
;ins start dur  amp    pitch  carr   mod    max  functions
;                                    idx  osc amp idx

i36  2    .2   15000  6      1      1.4142  25   1   2   3
i36  2.5  .    >      .      .      .      .    .   .
i36  3    .    25000  .      .      .      .    .   .

e
```

```
; CS36.ORC      -       Brass-like tones instrument

sr = 44100
kr =  4410
ksmps = 10
nchnls = 1

instr 37        ;       General FM instrument, generates
                ;       brass-like tones
;p4  : amplitude
;p5  : pitch (to be converted to frequency)
;p6  : attack
;p7  : decay
;p8  : carrier/frequency
;p9  : modulator/frequency
;p10 : maximum index
;p11 : oscillator function (usually a sine wave)

    ifreq   =       cpspch(p5)      ; frequency
    ihrise  =       p6*.5           ; half of rise time
    ist1    =       p4*.75          ; steady state amplitude 1
    ist2    =       p4*.6           ; steady state amplitude 2
    istd    =       p3-p6-p7        ; duration of steady state

    ; envelope
    kenv    linseg  0,ihrise,p4,ihrise,ist1,istd,ist2,p7,0

    ; index
    kidx    =       kenv*p10/p4

    ; FM oscillator
    a1      foscil  kenv,ifreq,p8,p9,kidx,p11
```

```
    ; output
          out     a1
endin



; CS36.SC       Brass-like sounds

f1 0 8192 10 1          ; oscillator function

;         p3  p4      p5      p6    p7   p8    p9      p10  p11
;ins stt  dur amp     pitch   atck  dec  carr  mod     max  osc
;                                          idx  func

i37  0    .1  10000   8.01    .05   .04   1    1.002   5     1
i37  +    .   >       .     .     .     .     .     .     .
i37  +    .   >       .     .     .     .     .     .     .
i37  +    .9  15000   .     .2    .1    .     .     5     .

i37  +    .1  10000   .     .05   .04   1    1.002   5     1
i37  +    .   >       .     .     .     .     .     .     .
i37  +    .   >       .     .     .     .     .     .     .
i37  +    .9  15000   .     .2    .1    .     .     5     .

i37  +    .1  10000   .     .05   .04   1    1.002   5     1
i37  +    .   >       .     .     .     .     .     .     .
i37  +    .   >       .     .     .     .     .     .     .
i37  +   1.8  18000   8.04    .2    .4    .     .     5     .

i37  +    .1  12000   8.01    .05   .04   1    1.002   5     1
i37  +    .   >       .     .     .     .     .     .     .
i37  +    .   >       .     .     .     .     .     .     .
i37  +    .9  18000   8.04    .2    .1    .     .     5     .
```

i37 + .1 14000 8.01 .05 .04 1 1.002 5 1

i37 + . > . . . .

i37 + . > . . . .

i37 + .9 22000 8.04 .2 .1 . . 5 .

i37 + .1 14000 8.01 .05 .04 1 1.002 5 1

i37 + . > 8.04 . . . . . .

i37 + . > 8.08 . . . . . .

i37 + 2.1 25000 9.01 .2 .2 . 1.001 5 .

s

i37 .15 .15 22000 8.11 .08 .04 1 1.002 5 1

i37 + .8 23000 8.09 .1 .1 . . . . i37 + .15 22000 8.08 .08 .04 .

. . .

i37 + .8 23000 8.06 .1 .1 . . . .

i37 + .15 22000 8.04 .08 .04 . . . .

i37 + .8 23000 8.09 .1 .1 . . . .

i37 + .15 22000 8.08 .08 .04 . . . .

i37 + .8 23000 8.06 .1 .1 . . . .

i37 + .15 22000 8.04 .08 .04 . . . .

i37 + .25 22000 8.06 .08 .04 . . . .

i37 + .25 > 8.09 .08 .04 . . . .

i37 + .25 > 9.01 .08 .04 . . . .

i37 + 1.8 28000 9.04 .2 .1 . 1.001 7 .

i37 + .75 > 9.06 . . . . . .

i37 + 3 32000 9.09 . 1.5 . 1.0005 . .

e

Assignment :Produce an FM instrument with the following characteristics:

1.The amplitude is controlled by a function table.

2.The index is controlled by a function table.

3.The instrument has two formants at 1300 and 2500 hz.

Produce a short nmuxical fragment in which this instrument is used to play brass-like tones and also non-pitched sounds.

11.The Phase Vocoder

Pvoc 1

```
# analysis
sfpvoc -N4096 -A obc obc.dat
#
# synthesis
sfpvoc -N4096 -S -i0 -j24 obc.dat obclow
sfpvoc -N4096 -S -i25 -j80 obc.dat obcmid
sfpvoc -N4096 -S -i81 -j2048 obc.dat obchigh
```

Pvoc 2

```
# Pvoc Transposition
sfpvoc -N2048 -P2.1 mflute mflute.tr
# Ftrans
ftrans mflute mflute.ftr
```

Pvoc 3

```
sfpvoc -N4096 -T4.0 ahh ahhx4
sfpvoc -N4096 -T4.0 ahhx4 ahhx16
```

Pvoc 4

```
#       cut
cut inh4 inh4b 0 2.3
cut inh4 inh4e 2.3 2.7
#       analysis
```

```
sfpvoc -N2048 -A inh4e inh4e.dat
#       shift
specsh inh4e.dat inh4she1.dat 1 15 1.9 .1 .1 0 0
specsh inh4e.dat inh4she2.dat 1 15 2.4 .1 .1 0 0
specsh inh4e.dat inh4she3.dat 1 15 3.45 .1 .1 0 0
#       stretch
spece inh4e.dat inh4ste1.dat 0  15 1.9 .06 .1 0 0 .5
spece inh4e.dat inh4ste2.dat 0  15 2.4 .06 .1 0 0 .5
spece inh4e.dat inh4ste3.dat 0  4 4.6 .06 .1 0 0 .5
#       resynthesis
sfpvoc -N2048 -S inh4she1.dat inh4she1
sfpvoc -N2048 -S inh4she2.dat inh4she2
sfpvoc -N2048 -S inh4she3.dat inh4she3
sfpvoc -N2048 -S inh4ste1.dat inh4ste1
sfpvoc -N2048 -S inh4ste2.dat inh4ste2
sfpvoc -N2048 -S inh4ste3.dat inh4ste3
#       splice
splice -w0 inh4sh1 inh4b inh4she1
splice -w0 inh4sh2 inh4b inh4she2
splice -w0 inh4sh3 inh4b inh4she3
splice -w0 inh4st1 inh4b inh4ste1
splice -w0 inh4st2 inh4b inh4ste2
splice -w0 inh4st3 inh4b inh4ste3
```

Pvoc 5

```
#cuts
cut obD obDs 0 1.2
cut obD obDe 1.2 3.34
cut alhmix4 alhmix4s 0 2.011
cut alhmix4 alhmix4s 2.011 3.75
#analysis
sfpvoc -N4096 -A obDs obDs.dat
sfpvoc -N4096 -A alhmix4s alhmix4s.dat
#interpolation
vocinte obDs.dat alhmix4s.dat otg1m.dat .01 2 .01 2 2 2
#synthesis
sfpvoc -N4096 -S otg1m.dat otg1m
#splices
splice -w0 otg1s obDs otg1m
splice otg1 otg1s alhmix4s
```

References and Software

Atkins M. C.The Soundfile System. Software and Manual. Composers' Desktop Project. 1990.

Backus J.The Acoustical Foundations of Music.
W. W. Norton & Company, Inc. 1977

Beauchamp J.*Brass Tone Synthesis by Spectrum Evolution Matching with Nonlinear Functions*. pp 35-43. Computer Music Journal 3(2). 1979.

Bentley A.GROUCHO. Programs for Processing Sounds in the Electronic Music Studio, Using the CDP Soundfile System. Composers' Desktop Project. 1987.

Chowning, J M*The Synthesis of Complex Audio Spectra by Means of Frequency Modulation*. pp 46-54. Computer Music Journal 1(1), 1977.

Dodge C., Jerse T. A.Computer Music Synthesis Composition and Performance. Schirmer. 1985.

Fischman R.The Graphic Desktop. Composers' Desktop Project. 1987.

Orton R.ADSYN DRAW. Software and Manual. Composers' Desktop Project. 1988.

Snell J. (Ed.),Computer Music Journal. MIT Press. 1977 Abott C. (Ed.), - 1991
Strawn J. (Ed.)
Kaske S. (Ed.)
Roads C. (Ed.) Travis Pope S. (Ed.)

Roads C. (Ed.),Foundations of Computer Music. MIT Strawn J. (Ed.)Press. 1985.

Strawn J. (Ed)Digital Audio Signal Processing - An Anthology. Wm Kaufmann, Inc. 1985

Vercoe B., Payton W.Csound (Atari Version). Software and Manual. Composers' Desktop Project. 1989.

Williams C. S.Designing Digital Filters. Prentice-Hall Inc. 1986.

Wishart T.On Sonic Art. Trevor Wishart. York, 1985.

Wishart T.The Phase Vocoder. Software and Programs. Composers' Desktop Project Manual. 1986-89.