A Beginning Tutorial

by Barry Vercoe, Massachusetts Institute of Technology

The Orchestra File

Csound runs from two basic files: an orchestra file and a score file. The orchestra file is a set of instruments that tell the computer how to synthesize sound; the score file tells the computer when. An instrument is a collection of modular statements which either generate or modify a signal; signals are represented by symbols, which can be "patched" from one module to another. For example, the following two statements will generate a 440 Hz sine tone and send it to an output channel:

```
asig    oscil   10000, 440, 1
        out     asig
```

The first line sets up an oscillator whose controlling inputs are an amplitude of 10000, a frequency of 440 Hz, and a waveform number, and whose output is the audio signal asig. The second line takes the signal asig and sends it to an (implicit) output channel. The two may be encased in another pair of statements that identify the instrument as a whole:

```
instr 1
        asig    oscil   10000, 440, 1
                out     asig
endin
```

In general, an orchestra statement in Csound consists of an action symbol followed by a set of input variables and preceded by a result symbol. Its action is to process the inputs and deposit the result where told. The meaning of the input variables depends on the action requested. The 10000 above is interpreted as an amplitude value because it occupies the first input slot of an oscil unit; 440 signifies a frequency in Hertz because that is how an oscil unit interprets its second input argument; the waveform number is taken to point indirectly to a stored function table, and before we invoke this instrument in a score we must fill function table #1 with some waveform.

The output of Csound computation is not a real audio signal, but a stream of numbers which describe such a signal. When written onto a sound file these can later be converted to sound by an independent program; for now, we will think of variables such as asig as tangible audio signals.

Let us now add some extra features to this instrument. First, we will allow the pitch of the tone to be defined as a parameter in the score. Score parameters can be represented by orchestra variables which take on their different values on successive notes. These variables are named sequentially: p1, p2, p3, ... The first three have a fixed meaning (see the Score File), while the remainder are assignable by the user. Those of significance here are:

p3 - duration of the current note (always in seconds).
p5 - pitch of the current note (in units agreed upon by score and orchestra).

Thus in

```
asig    oscil    10000, p5, 1
```

the oscillator will take its pitch (presumably in cps) from score parameter 5.

If the score had forwarded pitch values in units other than cycles-per-second (Hertz), then these must first be converted. One convenient score encoding, for instance, combines pitch class representation (00 for C, 01 for C#, 02 for D, ... 11 for B) with octave representation (8. for middle C, 9. for the C above, etc.) to give pitch values such as 8.00, 9.03, 7.11. The expression

```
cpspch(8.09)
```

will convert the pitch A (above middle C) to its cps equivalent (440 Hz). Likewise, the expression

```
cpspch(p5)
```

will first read a value from p5, then convert it from octave.pitch-class units to cps. This expression could be imbedded in our orchestra statement as

```
asig    oscil    10000, cpspch(p5), 1
```

to give the score-controlled frequency we sought.

Next, suppose we want to shape the amplitude of our tone with a linear rise from 0 to 10000. This can be done with a new orchestra statement

```
amp    line    0, p3, 10000
```

Here, amp will take on values that move from 0 to 10000 over time p3 (the duration of the note in seconds). The instrument will then become

```
instr 1
        amp     line    0, p3, 10000
        asig    oscil   amp, cpspch(p5), 1
        out     asig
endin
```

The signal amp is not something we would expect to listen to directly. It is really a variable whose purpose is to control the amplitude of the audio oscillator. Although audio output requires fine resolution in time for good fidelity, a controlling signal often does not need that much resolution. We could use another kind of signal for this amplitude control

```
kamp   line    0, p3, 10000
```

in which the result is a new kind of signal kamp. Signal names up to this point have always begun with the letter a (signifying an audio signal); this one begins with k (for control). Control signals are identical to audio signals, differing only in their resolution in time. A control signal changes its value less often than an audio signal, and is thus faster to generate. Using one of these, our instrument would then become (next page):

```
instr 1
        kamp    line    0, p3, 10000
        asig    oscil   kamp, cpspch(p5), 1
        out     asig
endin
```

This would likely be indistinguishable in sound from the first version, but would run a little faster. In general, instruments take constants and parameter values, and use calculations and signal processing to move first towards the generation of control signals, then finally audio signals. Remembering this flow will help you write efficient instruments with faster execution times.

We are now ready to create our first orchestra file. Type in the following orchestra using the system editor, and name it "intro.orc".

```
sr = 44100      ; audio sampling rate is 44.1 kHz
kr = 4400       ; control rate is 4410 Hz
ksmps = 10      ; number of samples in a control period (sr/kr)
nchnls = 1      ; number of channels of audio output


instr 1
        kctrl   line    0, p3, 10000    ; amplitude envelope
        asig    oscil   kctrl, cpspch(p5), 1       ; audio oscillator
        out     asig            ; send signal to channel 1
endin
```

It is seen that comments may follow a semi-colon, and extend to the end of a line. There can also be blank lines, or lines with just a comment. Once you have saved your orchestra file on disk, we can next consider the score file that will drive it.

The Score File

The purpose of the score is to tell the instruments when to play and with what parameter values. The score has a different syntax from that of the orchestra, but similarly permits one statement per line and comments after a semicolon. The first character of a score statement is an opcode, determining an action request; the remaining data consists of numeric parameter fields (pfields) to be used by that action.

Suppose we want a sine-tone generator to play a pentatonic scale starting at C-sharp above middle-C, with notes of 1/2 second duration. We would create the following score:

```
f1      0       256     10      1       ;  a sine wave function table

;  a pentatonic scale
i1      0       .5      0       8.01
i1      .5      .       .       8.03
i1      1.0     .       .       8.06
i1      1.5     .       .       8.08
i1      2.0     .       .       8.10
e
```

The first statement creates a stored sine table. The protocol for generating wave tables is simple but powerful. Lines with opcode f interpret their parameter fields as follows:

p1 - function table number being created
p2 - creation time, or time at which the table becomes readable
p3 - table size (number of points), which must be a power of two or one greater
p4 - generating subroutine, chosen from a prescribed list.

Here the value 10 in p4 indicates a request for subroutine GEN10 to fill the table. GEN10 mixes harmonic sinusoids in phase, with relative strengths of consecutive partials given by the succeeding parameter fields. Our score requests just a single sinusoid. An alternative statement:

```
f1 0 256 10 1 0 3
```

would produce one cycle of a waveform with a third harmonic three times as strong as the first.

The i statements, or note statements, will invoke the p1 instrument at time p2, then turn it off after p3 seconds; it will pass all of its p-fields to that instrument. Individual score parameters are separated by any number of spaces or tabs; neat formatting of parameters in columns is nice but unnecessary. The dots in p-fields 3 and 4 of the last four notes invoke a carry feature, in which values are simply copied from the immediately preceding note of the same instrument. A score normally ends with an e statement.

The unit of time in a Csound score is the beat. In the absence of a tempo statement, one beat takes one second. To double the speed of the pentatonic scale in the above score, we could either modify p2 and p3 for all the notes in the score, or simply insert the line

```
t 0 120
```

to specify a tempo of 120 beats per minute from beat 0.

Two more points should be noted. First, neither the f statements nor the i statements need be typed in time order; Csound will sort the score automatically before use. Second, it is permissable to play more than one note at a time with a single instrument. To play the same notes as a three-second pentatonic chord we would create the following:

```
f1      0       256     10      1       ;  a sine wave function table


; five notes at once
i1      0       3       0       8.01
i1      0       .       .       8.03
i1      0       .       .       8.06
i1      0       .       .       8.08
i1      0       .       .       8.10
e
```

Now go into the editor once more and create your own score file. Name it "intro.sco". The next section will describe how to invoke a Csound orchestra to perform a Csound score.

The Csound Command

To request your orchestra to perform your score, type the command

csound intro.orc intro.sco (or simply “cs intro.orc intro.sco”

Or if you wish to suppress the display of the funtion tables, do:

csound -d intro.orc intro.sco (or simply “cs intro.orc intro.sco”

The resulting performance will take place in three phases:

1. sort the score file into chronological order. If score syntax errors are encountered they will be reported on your console.

2. translate and load your orchestra. The console will signal the start of translating each instr block, and will report any errors. If the error messages are not immediately meaningful, translate again with the verbose flag turned on:

csound -v intro.orc intro.sco

3. fill the wave tables and perform the score. Information about this performance will be displayed throughout in messages resembling

B 4.000 .. 6.000 T 3.000 TT 3.000 M 7929. 7929.

A message of this form will appear for every event in your score. An event is defined as any change of state (as when a new note begins or an old one ends). The first two numbers refer to beats in your original score, and they delimit the current segment of sound synthesis between successive events (e.g. from beat 4 to beat 6). The second beat value is next restated in real seconds of time, and reflects the tempo of the score. That is followed by the Total Time elapsed for all sections of the score so far. The last values on the line show the maximum amplitude of the audio signal, measured over just this segment of time, and reported separately for each channel.

Console messages are printed to assist you in following the orchestra's handling of your score. You should aim at becoming an intelligent reader of your console reports. When you begin working with longer scores and your instruments no longer cause surprises, the above detail may be excessive. You can elect to receive abbreviated messages using the -m option of the Csound command.

When your performance goes to completion, it will have created a sound file named test in your soundfile directory. You can now listen to your sound file by typing

play test.wav (or simply “p test.wav”

If your machine is fast enough, and your Csound module includes user access to the audio output device, you can hear your sound as it is being synthesized by using the command:

csoundplay

More about the Orchestra

Suppose we next wished to introduce a small vibrato, whose rate is 1/50 the frequency of the note (i.e. A440 is to have a vibrato rate of 8.8 Hz.). To do this we will generate a control signal using a second oscillator, then add this signal to the basic frequency derived from p5. This might result in the instrument

```
instr 1
        kamp    line    0, p3, 10000
        kvib    oscil   2.75, cpspch(p5)/50, 1
        a1      oscil   kamp, cpspch(p5)+kvib, 1
        out     a1
endin
```

Here there are two control signals, one controlling the amplitude and the other modifying the basic pitch of the audio oscillator. For small vibratos, this instrument is quite practical; however it does contain a misconception worth noting. This scheme has added a sine wave deviation to the cps value of an audio oscillator. The value 2.75 determines the width of vibrato in cps, and will cause an A440 to be modified about one-tenth of one semitone in each direction (1/160 of the frequency in cps). In reality, a cps deviation produces a different musical interval above than it does below. To see this, consider an exaggerated deviation of 220 cps, which would extend a perfect 5th above A440 but a whole octave below. To be more correct, we should first convert p5 into a true decimal octave (not cps), so that an interval deviation above is equivalent to that below. In general, pitch modification is best done in true octave units rather than pitch-class or cps units, and there exists a group of pitch converters to make this task easier. The modified instrument would be

```
instr 1
        ioct    =       octpch(p5)
        kamp    line    0, p3, 10000
        kvib    oscil   1/120, cpspch(p5)/50, 1
        asig    oscil   kamp, cpsoct(ioct+kvib), 1
        out     asig
endin
```

This instrument is seen to use a third type of orchestra variable, an i-rate variable. The variable ioct receives its value at an initialization pass through the instrument, and does not change during the lifespan of this note. There may be many such init time calculations in an instrument. As each note in a score is encountered, the event space is allocated and the instrument is initialized by a special pre-performance pass. i-rate variables receive their values at this time, and any other expressions involving just constants and i-rate variables are evaluated. At this time also, modules such as line will set up their target values (such as beginning and end points of the line), and units such as oscil will do phase setup and other bookkeeping in preparation for performance. A full description of init-time and performance-time activities, however, must be deferred to a general consideration of the orchestra syntax.