1 Introducción al Diseño de Sonido en Csound

Richard Boulanger (Traducción: Servando Valero)

Csound es un programa de síntesis de sonido increíblemente potente y versátil. Al disponer de una caja de herramientas con más de 450 módulos de procesado de señal, uno puede usar Csound para modelar virtualmente cualquier sintetizador o procesador multi-efectos comercial. Csound transforma literalmente un ordenador personal en una *workstation* (estación de trabajo) de audio digital de alta fidelidad, un entorno en el que los mundos del diseño de sonido, la investigación acústica, la producción de audio digital y la composición de música por ordenador se unen para dar lugar al instrumento expresivo definitivo. Sin embargo, como con cualquier instrumento musical, el verdadero virtuosismo es, en realidad, el producto tanto del talento como de la dedicación personal. Pronto descubrirás que Csound es el instrumento musical definitivo. Pero se requiere práctica. A cambio, Csound recompensará tu dedicación produciendo algunas de las más ricas texturas y únicos y hermosos timbres que hayas oído jamás. En el mundo sonoro de Csound el conocimiento y la experiencia son la clave… y tu imaginación la única limitación.

El objetivo de este capítulo es prepararte para comenzar a hollar el sendero del descubrimiento y la maestría de Csound. A lo largo de dicho camino examinaremos un amplio espectro de técnicas de síntesis y procesado de señal y veremos como implementarlas en Csound. Para cuando lleguemos al final del capítulo habremos explorado un buen número de las muchas posibilidades que ofrece Csound. Te animo a que compiles, escuches, estudies y modifiques cada uno de mis instrumentos básicos de ejemplo. Haciéndolo adquirirás una clara comprensión y apreciación del lenguaje, al mismo tiempo que sientas las sólidas bases sobre las cuales construir tu propia librería personal de instrumentos, ya sean originales o modificados. Además, trabajar con los conceptos básicos que se cubren en este capítulo te preparará para comprender, apreciar y aplicar mejor los más avanzados modelos de síntesis y procesado de señal que presentarán mis colegas y amigos en los capítulos posteriores de este libro.

Además, en el CD-ROM que acompaña al libro hay miles de instrumentos y cientos de piezas escritos en Csound. Cada uno abre una puerta a uno de los múltiples mundos de Csound. De hecho, llevaría toda una vida explorarlos todos al detalle. Obviamente, una manera de proceder sería compilar todas las orquestas del CD-ROM, seleccionar las que te suenen más interesantes y luego simplemente copiarlas en tus propias composiciones. Esta librería de *presets* bien podría ser justo la colección de sonidos originales que andabas buscando y entonces tu viaje habría terminado.

Sin embargo, creo que lo más recomendable sería leer, compilar, escuchar y, por último, estudiar las técnicas de síntesis y procesado de señal que te hayan impresionado más, modificando las orquestas de Csound existentes que emplean dichas técnicas. Después, deberías intentar expresar esta comprensión a través de tus propias composiciones, tus propios “paisajes tímbricos” y “collages sonoros”. Seguramente, mediante este proceso de descubrimiento “activo”, empezarás a desarrollar tu propia librería personal de Csound y por último tu propia “voz”.

Para seguir el camino que te propongo, necesitarás comprender la estructura y la sintaxis del lenguaje de Csound. Pero tengo la certeza de que, con ese conocimiento, serás capaz de traducir tu experiencia personal en el mundo del audio y la síntesis, ya en originales y hermosos instrumentos sintéticos, ya en algunas esculturas sonoras verdaderamente únicas y vívidas.

Con ese fin, empezaremos por aprender la estructura y la sintaxis del lenguaje de las orquestas y partituras de Csound. Entonces continuaremos explorando varios algoritmos de síntesis y técnicas de programación en Csound. Finalmente, avanzaremos hasta ver algunos ejemplos de procesado de señal. A lo largo de nuestro recorrido, cubriremos algunos conceptos básicos de audio digital y aprenderemos algunos trucos de programación de software de síntesis. Para comprender mejor los algoritmos y el flujo de la señal, se presentarán diagramas de flujo con la mayoría de nuestros instrumentos en Csound. Así mismo, te propondré un buen número de ejercicios que te ayudarán a comprender detalladamente las muchas formas en que realmente puedes trabajar con el programa.

No te saltes los ejercicios. Y no los leas simplemente. ¡Hazlos! Son la clave para desarrollar verdadera fluidez con el lenguaje. De hecho, puede sorprenderte descubrir que estos ejercicios te enseñan más sobre cómo trabajar con Csound que cualquiera de las descripciones que los preceden. Al final, deberías haber adquirido una buena y sólida base sobre la que construir tu propia librería de instrumentos y habrás pavimentado el camino para desarrollar un conocimiento más profundo de los capítulos que siguen.

Así que sigue las instrucciones del CD-ROM: instala Csound en tu ordenador, compila y escucha unas cuantas orquestas de prueba para asegurarte de que todo está funcionando bien y… ¡manos a la obra!

¿Qué es Csound y Cómo Funciona?

Csound es un renderizador de sonido. Funciona traduciendo primero una serie de *instrumentos* en formato de texto, localizados en el fichero *orquesta*, a una estructura de datos residente en la máquina y comprensible para el ordenador. Luego ejecuta esos instrumentos definidos por el usuario interpretando una lista de eventos que contiene las *notas* y los *parámetros*, que son los datos que el programa lee: puede ser un fichero *partitura* en formato de texto, un fichero MIDI generado por un secuenciador, los datos enviados por un controlador MIDI en tiempo real, datos de audio en tiempo real, o los enviados a través de un dispositivo no MIDI como, por ejemplo, un teclado ASCII o un ratón.

Dependiendo de la velocidad de tu ordenador (y de la complejidad de los instrumentos en el fichero orquesta) la ejecución de esta "partitura" podrá ser, bien escuchada en tiempo real, bien escrita directamente a un fichero en tu disco duro. Este proceso completo es llamado "renderizado de sonido", análogamente al proceso de "renderizado de imágenes" utilizado en el mundo de los gráficos por ordenador.

Una vez renderizado, podrás escuchar el archivo de sonido resultante abriéndolo con tu editor de sonido preferido y reproduciéndolo a través del conversor digital-analógico (DAC) incluido en la placa madre de tu ordenador o a través del DAC de tu tarjeta de sonido.

Por lo tanto, en Csound trabajamos básicamente con dos ficheros de texto interdependientes y complementarios: el fichero *orquesta* y el fichero *partitura*. Puedes dar a estos ficheros el nombre que quieras. Normalmente, damos el mismo nombre a los dos ficheros y los diferenciamos por una extensión de tres caracteres, *.orc* para el fichero orquesta y *.sco* para el fichero partitura. El nombre depende de ti. En este primer capítulo, he llamado a los ficheros *etude1.orc* y *etude1.sco, etude2.orc* y *etude2.sco, etude3.orc* y *etude3.sco,* etc. Estas "orquestas-estudios" contienen seis instrumentos cada una (*instr 101 – 106, instr 107 – 112*, *instr 113 – 118*, etc.). Dando el mismo nombre al fichero partitura que al fichero orquesta correspondiente, podremos mantener nuestra librería de instrumentos bien organizada. Te recomiendo que hagas lo mismo. De hecho, todas las orquestas y partituras en este libro y el CD-ROM que lo acompaña siguen este convenio de nombres.

El Fichero Orquesta

El fichero orquesta de Csound está dividido en dos partes: la sección de *cabecera* y la sección de *instrumentos*.

La Sección de Cabecera

En la sección de cabecera definimos las frecuencias tanto de muestreo como de control a las que los instrumentos serán renderizados, así como el número de canales de la salida. La cabecera de la orquesta que usaremos a lo largo de todo el texto es la siguiente:

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

Figura 1.1 "Cabecera" de la orquesta de Csound por defecto.

El código de esta cabecera asigna una frecuencia de muestreo (*sr*) de 44.1 KHz (44100 Hz), una frecuencia de control (*kr*) de 4410 Hz y un valor 10 a la variable ksmps (*ksmps = sr/kr*). La cabecera indica también que la orquesta debe renderizar un fichero de sonido mono, estableciendo el número de canales (*nchnls*) en 1. (Si quisiéramos renderizar un fichero de sonido estéreo, simplemente daríamos un valor 2 a *nchnls*).

La Sección de Instrumentos

En Csound, los instrumentos se definen (y por tanto de diseñan) interconectando “módulos” u *opcodes* (del inglés *operation code*, código de operación) que generan o modifican señales. Estas señales se representan mediante *símbolos*, *etiquetas* o *nombres de variable* que se pueden “pasar” de un opcode a otro. Cada instrumento está delimitado por las sentencias **instr** y **endin** y se representa por un número distintivo. Un único fichero orquesta puede contener virtualmente cualquier número de instrumentos. De hecho, en Csound todo es un instrumento, tu *sampler* de 8000 voces, tu sinte FM de 4000 voces, tu sinte *waveguide* de 2000 voces, tu ecualizador de 1000 bandas, tu mesa mezcladora automatizada de 500 canales, las 250 líneas de retardo de tu *flanger* fractal, tu reverb por convolución, tu espacializador vectorial, o lo que sea… Para Csound cada uno de estos dispositivos de síntesis, procesado de señal o aparatos de estudio son meramente *instr 1*, *instr 2*, *instr 3*, *instr 4*, etc.

La Sintaxis de la Orquesta

En el fichero orquesta de Csound, la sintaxis de la sentencia de un opcode genérico es:

```
Salida      Opcode          Argumentos          Comentarios (opcional)
```

En el caso del opcode **oscil**, esto se traduce como:

```
salida                  amplitud    frecuencia    función #     ; COMENTARIO

a1          oscil       10000,      440,          1             ; OSCILADOR
```

Estudio de Diseño de Sonido 1: Una Orquesta de 6 Instrumentos

En nuestro primer fichero orquesta, *instr 101* usa un oscilador con búsqueda en tabla (*table-lookup oscillator*), opcode **oscil**, para ejecutar una onda sinusoidal de 440 Hz con un valor de amplitud de 10000. En la figura 1.2 podemos ver el diagrama de flujo de *instr 101* y, en la figura 1.3, el código real de la orquesta de Csound.

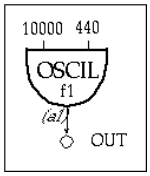


Figura 1.2 Diagrama de flujo de *instr 101*, un instrumento basado en un simple oscilador con búsqueda en tabla con frecuencia y amplitud fijas.

```
          instr       101               ; OSCILADOR SIMPLE
a1        oscil       10000, 440, 1
          out         a1
          Endin
```

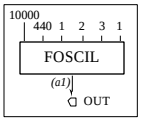
Figura 1.3 Código de la Orquesta de *instr 101*, un instrumento de frecuencia y amplitud fijas que usa el opcode del oscilador con búsqueda en tabla de Csound.

El diagrama de flujo de *instr 101* muestra claramente como la salida del oscilador, etiquetada como *a1*, es pasada a la entrada del opcode **out** que escribe la señal en el disco duro.

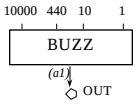
Csound renderiza los instrumentos línea a línea, de arriba a abajo. Los argumentos de entrada se especifican a la derecha del nombre del opcode. Las salidas van a la izquierda. Los caracteres que siguen a un punto y coma (;) son ignorados. Se consideran comentarios.

En *instr 101*, tal y como se muestra en la figura 1.3, los argumentos de entrada del oscilador son 10000 (para la amplitud), 440 (para la frecuencia) y 1 (para el número de la tabla que contiene la función de la onda que el oscilador ha de leer). El opcode oscilador renderiza el sonido 44100 veces por segundo con dichos valores y escribe el resultado en la variable *a1*. Los valores muestreados en la variable local *a1* pueden entonces ser leídos como entradas por los opcodes subsiguientes, como, por ejemplo, el opcode **out**. De esta forma, los nombres de las variables funcionan como si fueran los "cables de conexión" (*patch cords*) de un sintetizador analógico tradicional. Con estos cables de conexión virtuales uno puede dirigir las señales de audio y de control a cualquiera de los instrumentos, usándolas para: asignar un nuevo valor a un parámetro, controlar dinámicamente un parámetro (como si giráramos un botón), o como si fuera la entrada de audio de algún opcode de procesado de señal.

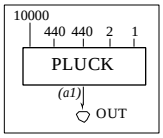
En la figura 1.4 puedes ver que *instr 102 – 106* usan el mismo simple diseño que *instr 101* (un generador de señal escribiendo su salida en el disco duro). Hemos reemplazado el opcode **oscil** por otros opcodes de síntesis más potentes como: **foscil**, un sintetizador FM sencillo con 2 osciladores; **buzz**, que usa una serie aditiva de cosenos armónicamente relacionados; **pluck**, un sintetizador *waveguide* sencillo basado en el algoritmo de Karplus-Strong; **grain**, un sintetizador granular asíncrono; y **loscil**, un sintetizador que lee en bucle (*looping*) tablas de onda previamente muestreadas.



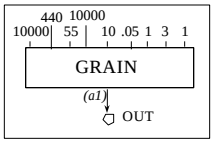

```
        Instr       102                 ; FM SIMPLE
a1      Foscil      10000, 440, 1, 2, 3, 1
        Out         a1
        Endin
```




```
        Instr       103                 ; BUZZ SIMPLE
a1      Buzz        10000, 440, 10, 1
        out         a1
        endin
```

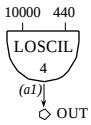



```
        instr       104                 ; WAVEGUIDE SIMPLE
a1      pluck       10000, 440, 440, 2, 1
        out         a1
        endin
```




```
        instr       105                 ; SINTESIS GRANULAR SIMPLE
a1      grain       10000, 440, 55, 10000, 10, .05, 1, 3, 1
        out         a1
```

```
        endin
```




```
        instr       106                 ; REPRODUCTOR DE MUESTRAS SIMPLE
a1      loscil      10000, 440, 4
        out         a1
        endin
```

Figura 1.4 Diagramas de flujo y código de la orquesta para *instr 102 – instr 106*, una colección de instrumentos con frecuencia y amplitud fijas que usan diferentes métodos de síntesis para producir una única nota con la misma amplitud (10000) y frecuencia (440).

Obviamente la estructura de "único generador de señal" de todos estos instrumentos es idéntica. Pero una vez que los renderices oirás que sus sonidos son bastante diferentes. Incluso aunque cada uno se ejecuta con una frecuencia de 440 Hz y una amplitud de 10000, el algoritmo de síntesis subyacente que cada opcode usa es fundamentalmente distinto y, por tanto, se requiere la especificación de distintas series de parámetros en cada caso. De hecho, estos seis opcodes generadores de señal (**oscil**, **foscil**, **buzz**, **pluck**, **grain** and **loscil**) representan el núcleo de la tecnología de síntesis subyacente en muchos de los más populares sintetizadores comerciales de hoy día. Puede decirse que en Csound cada opcode es un sintetizador completo. Bueno… puede que no sea un sintetizador demasiado excitante o versátil, pero… en combinación con otros opcodes, Csound puede, y lo hará, llevarte mucho más allá de cualquier implementación comercial.

El Fichero Partitura

Echemos ahora un vistazo al fichero partitura de Csound que "toca" los instrumentos de esta orquesta. Como el fichero orquesta, el fichero partitura tiene dos partes: *tablas* y *notas*. En la primera parte, usamos las subrutinas de dibujo de funciones matemáticas (**GENS**) para generar directamente las tablas de función (*f-tables*) y/o rellenarlas mediante la lectura de ficheros de sonido desde el disco duro. En la segunda parte, introducimos las *sentencias de nota*. Estos eventos de nota "tocan" los instrumentos y les pasan parámetros de interpretación, tales como ajustes de frecuencia, niveles dinámicos, frecuencias de vibrato o duraciones de ataque.

Las Rutinas GEN

Las subrutinas de generación de funciones se llaman **GENS**. Cada una de estas subrutinas (más de 20) está optimizada para generar un tipo específico de funciones o tablas de onda. Por ejemplo, las subrutinas **GEN5** y **GEN7** construyen funciones mediante segmentos de curvas exponenciales o líneas rectas; las subrutinas **GEN9** y **GEN10** generan formas de onda complejas compuestas por la suma ponderada de ondas sinusoidales simples; la subrutina **GEN20** genera funciones de "ventana" estándar como las de Hamming o Kaiser, que son las que se usan típicamente para el análisis espectral y en los envolventes granulares; la subrutina **GEN21** calcula tablas con distribuciones aleatorias diferentes, como, por ejemplo, la Gausiana, la de Cauchy o la de Poison; y la subrutina **GEN1** transfiere los datos de un fichero de sonido pregrabado a una tabla de función para su posterior procesado con uno de los opcodes de Csound, como, por ejemplo, el oscilador en bucle **loscil**.

Qué tablas de función se requieren y cómo las usarán los instrumentos de tu orquesta depende totalmente de ti, el diseñador de sonido. Algunas veces es una cuestión de sentido común. Otras veces es simplemente una cuestión de preferencia o hábito. Por ejemplo, debido a que *instr 106* usaba el oscilador de lectura de muestras en bucle **loscil**, necesitabamos cargar una muestra en la orquesta. Escogí **GEN1** para hacerlo. Mientras que en *instr 102*, debido a que usabamos el opcode **foscil**, podríamos haber escogido modular la frecuencia de dos ondas cualquiera, pero decidí usar la aproximación tradicional y modulé dos ondas sinusoidales definidas con **GEN10**.

La Sintaxis de la Partitura

En el fichero partitura de Csound, la sintaxis de la sentencia de función (*f-statement*) es:

```
f   número   instante    Tamaño de     Rutina GEN    parámetro1   parámetro...    ; COMENTARIO
             de carga    la tabla
```

Si quisiéramos generar una onda sinusoidal de 16 puntos, podríamos escribir la siguiente sentencia:

```
f   101    0     16    10     1         ; UNA ONDA SINUSOIDAL
```

Como resultado, la tabla de función (*f 101*) generaría la función que se muestra en la figura 1.5

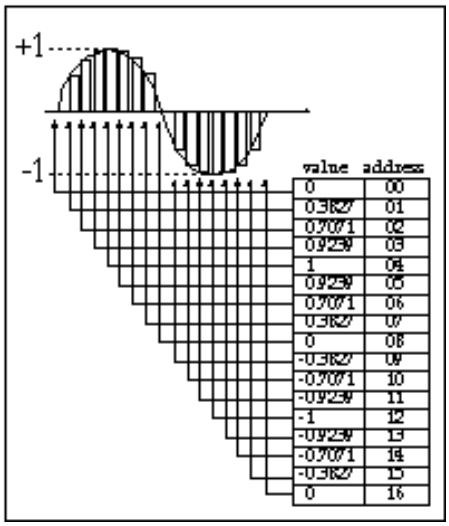


Figura 1.6 Una función seno con 16 puntos definida por GEN10 con los argumentos *f 101 0 16 10 1*

Como puedes ver, una onda sinusoidal dibujada con una resolución de 16 puntos no es particularmente suave. La mayoría de las funciones deben tener una longitud igual a una potencia de 2. Para las tablas de onda sintetizadas normalmente especificamos tamaños de tablas de función de entre 512 (5k) y 8192 (8k). En nuestra primera partitura, *etude1.sco,* definimos la siguientes funciones, usando **GEN10, GEN20** y **GEN1**:

```
f 1    0    4096    10     1
f 2    0    4096    10     1 .5 .333 .25 .2 .166 .142 .125 .111 .1 .09 .083 .076 .071 .066 .062
f 3    0    4097    20     2 1
f 4    0    0       1      "sing.aif" 0 4 0
```

Figura 1.6 Tablas de función definidas en el fichero partitura *etude1.sco*.

Todas estas cuatro funciones se cargan en el instante 0. *f1* y *f2* usan ambas **GEN10** para rellenar tablas de 4K (4096 elementos) con un ciclo de una onda sinusoidal (*f1*) y con una onda diente de sierra con los 16 primeros armónicos (*f 2*), respectivamente. **GEN20** se usa para rellenar una tabla de 4K (*f3*) con una ventana *Hanning* para ser usada por el opcode **grain**. Finalmente, *f4* usa **GEN1** para rellenar una tabla con un fichero de sonido mono de 16 bits en formato AIFF con una frecuencia de muestreo de 44.1 KHz de un cantante masculino cantando la sílaba “la” a una frecuencia de 440 Hz durante 3 segundos. Esta “muestra” es usada por el opcode **loscil**. (Observa que la longitud de *f4* es 0. Esto le indica a la subrutina **GEN1** que lea la longitud real según los datos de la cabecera del fichero de sonido “sing.aif”. En este caso específico, esa longitud sería de 132300 muestras, 44100 muestras por segundo durante 3 segundos).

El Listado de Notas

En la segunda parte del fichero partitura de Csound escribimos las “notas”. Como en el fichero orquesta, cada *sentencia de nota* en el fichero partitura ocupa una única línea. Las sentencias de nota (o *sentencias-i*) llaman a un instrumento para activarlo en un instante determinado y durante una duración determinada. Además, cada sentencia de nota puede ser usada para pasar un número virtualmente ilimitado de parámetros a un instrumento y dichos parámetros pueden ser cambiados de nota a nota.

Como la orquesta, que renderiza un sonido línea a línea, el fichero partitura también se lee línea a línea, nota a nota. Sin embargo, las “notas” pueden tener el mismo *instante de comienzo* y ser así interpretadas simultáneamente. En Csound, uno debe ser siempre consciente del hecho de que cuandoquiera que dos o más notas se ejecuten simultáneamente o cuandoquiera que éstas se pisen, sus amplitudes se sumarán. Esto puede causar frecuentemente “muestras fuera de rango”, o el llamado efecto *“cilpping”* (Discutiremos esto en detalle dentro de poco).

Puedes haberte dado cuenta de que en la orquesta, las comas separaban los argumentos de un opcode cualquiera. Aquí, en la partitura, cualquier número de espacios o tabulaciones separan tanto los argumentos de una tabla de función como los campos de parámetros (o *campos-p*) de las sentencias-i. Nunca se usan comas.

Para mantener las cosas claras y en orden, los diseñadores de sonido usan normalmente tabulaciones para separar los campos-p. Esta práctica mantiene los campos-p alineados en columnas y facilita tanto la lectura como la depuración. No es obligatorio, pero sí altamente recomendable.

Los Primeros Tres Campos-p

En todas las sentencias de nota, el “significado” de los tres primeros campos-p (o columnas) está reservado. Dichos campos especifican el *número de instrumento*, el *instante de comienzo* y la *duración*.

```
;     p1               p2            p3
i     instrumento #    comienzo      duración
```

Tú, el diseñador de sonido, determinas la función de todos los demás campos-p. Normalmente, *p4* se reserva para las amplitudes y *p5* para las frecuencias. Este convenio es el que ha sido adoptado en este capítulo y a lo largo de todo el texto. En nuestra primera partitura, *etude1.sco,* cada instrumento, *instr 101 – instr 106*, ejecutaba consecutivamente una única nota con una duración de tres segundos. Al estar los instantes de comienzo de cada nota separados cada 4 segundos, se producirá un segundo de silencio entre cada evento de audio.

```
; P1                  P2                P3
; INSTRUMENTO #       COMIENZO          DURACION
i 101                 0                 3
i 102                 4                 3
i 103                 8                 3
i 104                 12                3
i 105                 16                3
i 106                 20                3
```

Figura 1.7 La sencilla partitura usada para ejecutar los instrumentos 101 a 106, mostrados en la figura 1.2 y 1.4

Ejercicios para el Estudio 1

• Compila los ficheros orquesta y partitura: *etude1.orc* & *etude1.sco.*
• Reproduce y escucha atentamente los diferentes timbres de cada instrumento.
• Modifica la partitura y cambia la duración de cada nota.

• Haz que todas las notas empiecen al mismo tiempo.
• Deshabilita (convirtiéndolas en comentarios) varias notas para que no suenen.
• "Corta y pega" varias copias de las notas y cambia los instantes de comienzo (*p2*) y las duraciones (*p3*) de dichas copias para hacer que los mismos instrumentos empiecen y acaben en distintos momentos.
• Crea un canon al unísono con *instr 106*.
• Busca y lee acerca de los opcodes usados en *instr 101 – 106* en el Manual de Referencia de Csound.

```
Ar   oscil    xamp,   xcps,    Ifn[,    iphs]
Ar   foscil   xamp,   kcps,    Kcar,    kmod,     kndx,      ifn[,    iphs]
Ar   buzz     xamp,   xcps,    Knh,     ifn[,     iphs]
Ar   pluck    kamp,   kcps,    Icps,    ifn,      imeth[,    iparm1   iparm2]
                                                              ,
Ar   grain    xamp,   xpitch,  Xdens    kampoff   kpitchof   kgdur,   igfn,     iwfn,    Imgdur
                                ,         ,         f,
Ar   losci    xamp,   kcps,    Ifn[,    ibas][,   imod1,     ibeg1,   iend1][   imod2,   ibeg2,   iend2]
                                                                        ,
```

• En el fichero orquesta, modifica los argumentos de frecuencia y amplitud de cada instrumento.
• Cambia las razones de frecuencia de la portadora y la moduladora en el instrumento que usa **foscil.**
• Cambia el número de armónicos en el instrumento que usa **buzz.**
• Cambia la función inicial del instrumento que usa **pluck.**
• Cambia la densidad y la duración del instrumento que usa **grain.**
• Haz tres copias de *f 4* y renuméralas como *f 5*, *f 6* y *f 7*. Rellénalas con tus propias muestras ("tusonido1.aif," "tusonido2.aif," "tusonido3.aif"). Crea varias copias de *instr 106* y renuméralas como *instr 66*, *instr 67* y *instr 68*. Edita dichos instrumentos para que cada uno pueda leer un archivo de sonido diferente y a una distinta altura. Reproduce las diferentes muestras simultáneamente.
• En el fichero *etude1.orc* duplica y renumera cada copia. Dale valores diferentes a los parámetros de cada versión de los instrumentos duplicados. Reproduce los 12 instrumentos simultáneamente. Ajusta las amplitudes para que no tengas *muestras fuera de rango*.

Sonido, Señales y Muestreo

Para apreciar mejor cómo funciona Csound, será mejor que te asegures de que comprendes las propiedades acústicas del sonido y cómo es representado en un ordenador.

La sensación de sonido resulta de la respuesta simpática de nuestro tímpano a las compresiones y dilataciones de las moléculas de aire proyectadas hacia afuera en todas las direcciones por una fuente de vibración. Este patrón vibratorio de variaciones de presión se llama *forma de onda*. Al observar la figura 1.8, podemos ver que las moléculas de aire estarán comprimidas cuando la forma de onda esté por encima del eje x (valores de abscisas positivos) y dilatadas cuando esté por debajo de éste (valores negativos). La figura 1.8 muestra un único ciclo de una onda cuadrada tanto en el dominio del tiempo como en de la frecuencia.

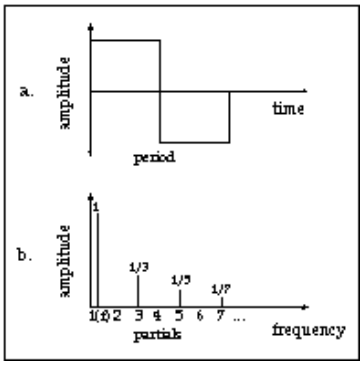


Figura 1.8 Representaciones de una onda cuadrada en el dominio temporal (a) y en el de frecuencia (b).

La representación en el dominio temporal (a) dispone el tiempo en las abscisas y las amplitudes en las ordenadas. La representación en el dominio de la frecuencia (b) dispone las frecuencias en el eje de abscisas y las amplitudes en el eje de ordenadas.

Nosotros experimentamos el sonido en el dominio temporal como variaciones de presión, pero en el dominio de la frecuencia lo hacemos como variaciones espectrales. El oído actúa como un *transductor*, convirtiendo el movimiento mecánico del tímpano (a través de los huesecillos: martillo, yunque y estribo) en la membrana de la ventana oval, que causa que una onda se propague a través del fluido de la coclea y estimule las células capilares de la membrana basilar. Estos pelillos actúan como un analizador espectral de alta resolución que transmite la información de esta compleja serie de frecuencias al cerebro a través de los nervios unidos a dichos pelillos. Con esta increíblemente sensible serie de sensores, transductores y transmisores, nosotros, al resonar con el mundo que nos rodea, analizamos, codificamos, clasificamos y percibimos activa y continuamente las complejas características de las frecuencias de las ondas sonoras.

Normalmente, empleamos un transductor diferente (el micrófono) para convertir las ondas acústicas en señales que podamos visualizar y manipular en el ordenador. Este proceso se llama *muestreo* o *sampling* y se ilustra en la figura 1.9.

Cuando sampleamos (muestreamos) una onda sonora, usamos primero un micrófono para convertir una onda de presión acústica en una onda eléctrica análoga, o *señal analógica*. Entonces pasamos esta señal a través de un filtro *anti-aliasing* pasa-bajos para eliminar las frecuencias por encima de la mitad de la frecuencia de muestreo. De hecho, un sistema digital no puede representar exactamente una señal por encima de la mitad de la frecuencia de muestreo (esta "frecuencia espejo" se conoce como frecuencia *Nyquist*). Entonces, después de eliminar con el filtro pasa-bajos las frecuencias altas que no podemos representar con exactitud, procedemos a "medir" o "muestrear" ("samplear") la amplitud de la señal con un convertidor analógico-digital (ADC).

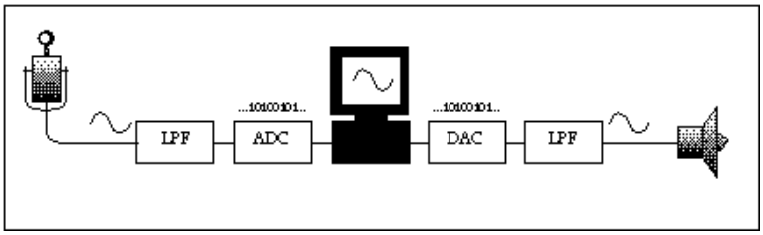


Figura 1.9 Grabación digital ("muestreo") y reproducción.

Si tienes un sistema lineal de 16 bits, samplearás la onda analógica con 16 bits de precisión (con valores en el rango de -32768 a 32767 o 2^{16}) tomando una nueva muestra a la frecuencia de muestreo (44100 veces cada segundo según nuestra cabecera por defecto). En esencia hemos "cuantizado" esta señal analógica en una serie de pequeñas instantáneas (o muestras) – literalmente estamos tomando miles de "pequeñas muestras" de la señal. Puedes ver claramente cómo se cuantiza la onda sinusoidal de la figura 1.5, donde cada dirección guarda la amplitud de la señal en ese preciso instante en el tiempo.

Para oir un sonido desde el ordenador, tenemos que reconvertir la señal digital (esta secuencia de "muestras") en una señal analógica (con un voltaje que varía continuamente) usando un convertidor digital-analógico (DAC) seguido por filtro pasa-bajos suavizante. ¿Se entiende todo esto? Bueno, basta de conceptos básicos por el momento. Volvamos a Csound.

Estudio de Diseño de Sonido 2: Campos de Parámetros en la Orquesta y la Partitura

En nuestra segunda orquesta modificaremos *instr 101 – 106* para que puedan ser actualizados y alterados desde la partitura. En vez de fijar el valor de los argumentos de los opcodes en la orquesta, como hicimos en *etude1.orc*, los definiremos según valores "p", que corresponden a los campos-p (o números de columna) en la partitura. De esta manera, cada argumento puede recibir un valor completamente diferente, tomándolo directamente de cada sentencia de nota.

En *instr 107*, por ejemplo, los campos-p se aplicarán a cada uno de estos argumentos del opcode **oscil**: la amplitud (*p4*), la frecuencia (*p5*) y la tabla de onda (*p6*), tal y como se muestra en la figura 1.10.

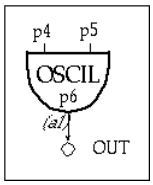


Figura 1.10 Diagrama de flujo de *instr 107*, un instrumento con un oscilador simple que usa campos-p en lugar de valores fijos.

```
        instr       107             ; OSCILADOR CON CAMPOS-P
a1      oscil       p4, p5, p6
        out         a1
        endin
```

Figura 1.11 Código de la orquesta para *instr 107*, un instrumento con un oscilador simple cuyos argumentos están tomados de los campos-p.

Así, usando el fichero partitura de la figura 1.12, podemos volver a usar el mismo instrumento para ejecutar una secuencia de tres octavas descendentes seguidas por un arpegio de La Mayor.

```
; P1        P2          P3          P4          P5          P6
;           COMIENZO    DURACION    AMPLITUD    FRECUENCIA  FORMA DE ONDA
INSTRUM

i 107       0           1           10000       440         1
i 107       1.5         1           20000       220         2
i 107       3           3           10000       110         2
i 107       3.5         2.5         10000       138.6       2
i 107       4           2           5000        329.6       2
i 107       4.5         1.5         6000        440         2
```

Figura 1.12 Listado de notas para *instr 107*, que usa campos-p para "ejecutar" 6 notas (algunas pisándose) con diferentes frecuencias, amplitudes y formas de onda.

En nuestro próximo ejemplo del uso de los campos p, representado en las figuras 1.13, 1.14 y 1.15, nuestro más bien limitado *instr 102* ha sido transformado en *instr 108*, un instrumento más versátil musicalmente, capaz de un largo abanico de colores tonales.

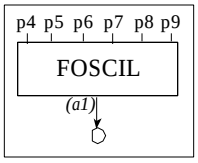


Figura 1.13 Diagrama de flujo de *instr 108*, un instrumento FM simple con campos-p en cada parámetro.

```
      instr       108                ; P-FIELD FM
a1    foscil      p4, p5, p6, p7, p8, p9
      out         a1
      Endin
```

Figura 1.14 Código de la orquesta de *instr 108*, un instrumento FM simple que recibe sus argumentos de los campos-p.

```
; P1      P2       P3       P4       P5       P6       P7       P8       P9
; INSTR   COM      DUR      AMP      FREC     C        M        INDICE   FORMA DE ONDA

i 108     7        1        10000    440      1        2        3        1
i 108     8.5      1        20000    220      1        .5       8        1
i 108     10       3        10000    110      1        1        13       1
i 108     10.5     2.5      10000    130.8    1        2.001    8        1
i 108     11       2        5000     329.6    1        3.003    5        1
i 108     11.5     1.5      6000     440      1        5.005    3        1
```

Figura 1.15 Listado de notas para *instr 108*, en el que se usan nueve campos-p para ejecutar un sintetizador FM con diferentes comienzos, duraciones, amplitudes, frecuencias, razones de frecuencia e índices de modulación.

En el fragmento de la partitura que se muestra en la figura 1.15, se ha asignado un distinto campo-p a cada uno de los argumentos de **foscil** para que puedan ser así alterados de nota a nota. En este caso, *p4* = amplitud, *p5* = frecuencia, *p6* = coeficiente de la portadora, *p7* = coeficiente de la moduladora, *p8* = índice de modulación y *p9* = forma de onda. De esta manera, a los 7 segundos de empezar *etude2.sco, instr 108* ejecuta seis notas consecutivas. Todas ellas usan *f 1* (una onda sinusoidal en *p9*). Las dos primeras notas, por ejemplo, están separadas por una octava (*p5* = 440 y 220) pero tienen diferentes coeficientes *c:m ratios* (*p7* = 2 y 13) y diferentes índices de modulación. (*p8* = 3 y 8), dando como resultado dos timbres muy distintos. Obviamente, los campos-p de la orquesta nos permiten obtener una gran variedad de alturas y timbres, incluso usando el más simple de los instrumentos.

Ejercicios para el Estudio 2

- Compila la orquesta y la partitura: *etude2.orc* & *etude2.sco.*
- Reproduce y escucha atentamente los diferentes timbres de cada nota e instrumento.
- Modifica el fichero partitura y cambia los instantes de comienzo, las duraciones, amplitudes y frecuencias de cada nota.
- Busca de nuevo y lee acerca de los opcodes usados por *instr 107 – 112* en el Manual de Referencia de Csound y concentra tu estudio y experimentación sólo en una técnica de síntesis a la vez.
- Explora los efectos de diferentes proporciones C:M en *instr 108*.
- Sin cambiar el coeficiente C:M, explora el efecto de índices de modulación bajos y altos.
- Compara la diferencia tímbrica cuando se modula usando una onda sinusoidal (*f 1*) y una onda diente de sierra (*f 2*).
- Usando *instr 109*, compón una progresión a cuatro voces en la que el bajo y el tenor tengan más armónicos que la contralto y la soprano.

- Usando *instr 109* y *112* simultáneamente, ejecuta la misma partitura que compusiste para *instr 109* doblando ahora las partes.
- Usando *instr 110*, experimenta con los diferentes métodos del opcode **pluck** (mira el Manual de Referencia de Csound para los argumentos adicionales)
- Usando *instr 110*, experimenta con diferentes tablas de función de inicialización – *f 1* y *f 2*. Intenta también inicializar usando ruido y compara el timbre resultante.
- Explora los diferentes parámetros del opcode **grain**.
- Crea una serie de estudios cortos para cada instrumento por separado.
- Crea una serie de estudios cortos para varios instrumentos al mismo tiempo. Recuerda ajustar tus niveles de amplitud para que no tengas muestras *fuera de rango*.
- Baja las frecuencias de muestreo y de control en la cabecera. Recompila algunos de tus instrumentos modificados. ¿Notas alguna diferencia en la calidad del sonido? ¿Notas algún cambio en el brillo? ¿Notas algún pequeño ruidito? (Discutiremos la teoría que se esconde detrás de este fenómeno un poco más tarde)

Amplitudes y Clipping

Como dijimos anteriormente, si tienes un convertidor de 16 bits en tu sistema (que es lo normal) puedes expresar 2^{16} posibles valores de amplitud (esto es 65536 en el rango -32768 a +32767). Esto se traduce en un rango dinámico de más de 90 dB (normalmente se consigue un rango de 6 dB por bit de resolución). Pero, si has estado haciendo los ejercicios, habrás notado probablemente que las amplitudes en Csound son aditivas. Esto significa que si un instrumento tiene una amplitud de 20000 y simultáneamente se tocan dos notas en dicho instrumento, le estás pidiendo a tu convertidor que produzca una señal con una amplitud de 40000 más o menos. El problema es que tu convertidor de 16 bits sólo puede representar valores hasta 32000 y, por tanto, obtendrás mensajes de *muestras fuera de rango* en tu trabajo con Csound y el fichero de sonido resultante aparecerá "cortado" como se muestra en la figura 1.16. A este efecto se le da el nombre de *clipping*.

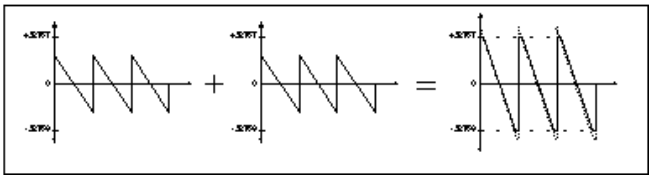


Figura 1.16 Clipping como resultado de añadir dos ondas de grandes amplitudes.

Tratar con amplitudes es uno de los aspectos más problemáticos al trabajar con Csound. No hay respuesta fácil. El problema yace en el hecho de que las amplitudes de Csound son simples representaciones matemáticas de "la señal". Estas mediciones no tienen para nada en cuenta la naturaleza acústica o perceptual "del sonido".

En dos palabras, un desplazamiento lineal que doble la amplitud de una onda no será necesariamente percibido como una intensidad el doble de fuerte. Un buen libro de Acústica te ayudará a apreciar la complejidad del problema. En el mundo de Csound, recuerda siempre que cuando dos o más notas suenan juntas, sus amplitudes se suman. Si los números suman algo más de 32000 tu señal sonará cortada. Anque Csound tiene algunos opcodes y herramientas que te ayudarán a tratar con este problema de "las muestras fuera de rango", ninguno de los actuales opcodes o convertidores de valor lo solucionará de verdad. La mayor parte del tiempo tendrás que recortar los niveles y renderizar el fichero otra vez (y otra y otra...), hasta que consigas que las amplitudes caigan en un rango que tu sistema soporte.

Frecuencias de Datos

Como has podido ver en los dos primeras orquestas de estudio, podemos definir y actualizar parámetros (argumentos) como "constantes en punto flotante", tanto directamente en la orquesta o indirectamente mediante campos-p. Pero la potencia real de Csound se deriva del hecho de que uno puede actualizar parámetros usando

variables a cualquiera de las cuatro frecuencias de actualización disponibles: **configuración, inicialización de nota** (tipo-**i**), **control** (tipo-**k**)**, audio** (tipo-**a**), donde:

- Las variables del tipo-**i** son modificadas y actualizadas cada nueva nota.
- Las variables del tipo-**k** son modificadas y actualizadas a la frecuencia de control (*kr*).
- Las variables del tipo-**a** son modificadas y actualizadas a la frecuencia de audio (*sr*).

Tanto las variables de tipo-**i** como las del tipo-**k** son "escalares". Esencialmente, toman sólo un valor a la vez. Las variables de tipo-**i** se usan principalmente para asignar valores a los parámetros y definir las duraciones de las notas. Estas variables son evaluadas en "tiempo de inicialización" y permanecen constantes durante toda la duración de la nota.

Las variables de tipo-**k** se usan principalmente para almacenar y actualizar envolventes y señales de control (sub-audio). Estas variables se recalculan a la frecuencia de control (4410 veces por segundo), según venga definida en la cabecera de la orquesta por la constante *kr*.

Las variables de tipo-**a** son "arrays" (cadenas) o "vectores" de información. Estas variables se usan para almacenar y actualizar datos tales como las señales de salida de un oscilador o un filtro, que cambian a la frecuencia de muestreo de audio (44100 veces por segundo), según venga ésta definida en la cabecera de la orquesta por la constante *sr*.

Se puede asignar o identificar la frecuencia a la que la variable será actualizada por la primera letra del nombre de la variable. Por ejemplo, la única diferencia entre los dos osciladores de abajo es que uno se calcula a frecuencia de muestreo y el otro a frecuencia de control. Ambos usan el mismo opcode, **oscil** y ambos tienen los mismos argumentos. Lo único que difiere es, entonces, la resolución (precisión) de la señal de salida.

```
; SALIDA        OPCODE       AMP,         FRC,       FUNC      ; COMENT

ksig            oscil        10000,       1000,      1         ; 1000 HZ SINE - F 1
asig            oscil        10000,       1000,      1         ; 1000 HZ SINE - F 1
```

Figura 1.17 Contrastando las salidas asig (audio) y ksig (control) de dos opcodes **oscil**.

Dada la configuración de nuestra cabecera por defecto, *sr* = 44100 and *kr* = 4410, *ksig* sería renderizada a una frecuencia de 4.41 KHz y *asig* a una de 44.1 KHz. En este caso, la salida resultante sonaría bastante similar porque ambas tienen bastante resolución para calcular exactamente la onda sinusoidal de 1000 Hz. Sin embargo, si los argumentos fueran diferentes y las formas de onda tuvieran armónicos adicionales, como los que tiene por ejemplo la onda diente de sierra definida por *f 2* en la figura 1.18, la frecuencia de control de 4410 muestras por segundo no representaría con precisión la forma de la onda y se produciría un efecto que se conoce como "aliasing" (Veremos este efecto en detalle más tarde).

```
; SALIDA        OPCODE       AMP,         FRC,       FUNC      ; COMENT

ksig            oscil        10000,       1000,      2         ; 1000 HZ SAW - F 2
asig            oscil        10000,       1000,      2         ; 1000 HZ SAW - F 2
```

Figura 1.18 Una onda diente de sierra "submuestreada" (dados *kr* = 4410 y una frecuencia 1000), lo que resulta en una señal de salida *ksig* con aliasing.

Debes darte cuenta que se deja en tus manos, como diseñador de sonido, decidir la frecuencia más apropiada, eficiente y efectiva a la que renderizar tus opcodes. Por ejemplo, podrías renderizar todos tus osciladores de baja frecuencia (LFOs) y envolventes a frecuencia de audio, pero Csound tardaría más en calcular las señales y el aumento de resolución sería, en la mayoría de los casos, imperceptible.

Nombres de Variable

En los instrumentos que hemos diseñado hasta ahora hemos usado y hablado de *a1*, *asig*, k1 y *ksig*, en muchos casos indiscriminadamente. ¿Por qué nombres diferentes para la misma cosa? Csound ya es bastante difícil en sí mismo. ¿Por qué complicarnos la vida?

Bueno, cuando se trata de dar nombres a las variables, Csound sólo requiere que dichos nombres empiecen por las letras **i**, **k**, o **a**. De esta manera, el programa puede determinar a qué frecuencia renderizar esa específica línea de código. Luego puede seguir cualquier cadena de caracteres.

Por ejemplo, podrías llamar a la salida del opcode **loscil** **a**1, **a**sig, **a**muestra, or **a**quebonitosonido. Cada uno de estos nombres de variable sería reconocido y ejecutado sin error por Csound. De hecho, si las líneas de código de cada uno de ellos tuviera los mismos parámetros, sonarían todos exactamente igual al renderizarlos, sin importar el nombre que les dieras. Por tanto, depende de ti, el diseñador de sonido, adoptar un sistema para referenciar las variables que sea claro, consistente y significativo… para ti.

```
a1              loscil     10000, 440, 4         ; REPRODUCCION DE LA MUESTRA EN F4 A 440HZ
                out        a1

asig            loscil     10000, 440, 4         ; REPRODUCCION DE LA MUESTRA EN F4 A 440HZ
                out        asig

amuestra        loscil     10000, 440, 4         ; REPRODUCCION DE LA MUESTRA EN F4 A 440HZ
                out        amuestra

aquebonitoson   loscil     10000, 440, 4         ; REPRODUCCION DE LA MUESTRA EN F4 A 440HZ
                out        aquebonitoson
```

Aliasing y el Teorema de Muestro

Revisemos un poco más de teoría antes de adentrarnos en diseños de instrumentos más complejos. Como dijimos antes, la “onda diente de sierra subsampleada” (*ksig*) de la figura 1.18 es un buen ejemplo de “aliasing” y una demostración del “Teorema de Muestreo”. En pocas palabras, el Teorema de Muestreo dice que, en el dominio digital, para reconstruir (trazar, dibujar o reproducir) con precisión una forma de onda a una determinada frecuencia, se necesita una frecuencia de muestreo el doble de la frecuencia más alta que se quiera renderizar. Este límite superior “fijo” en la mitad de la frecuencia de muestreo se conoce como *frecuencia Nyquist.* Con una frecuencia de audio de 44100 Hz se pueden renderizar con precisión sonidos con frecuencias (incluyendo parciales) de hasta 22050 Hz, discutiblemente por encima del umbral de audición humano. Y con una frecuencia de control de 4410 Hz se pueden renderizar sonidos de hasta 2205 Hz. Ciertamente eso daría un LFO increíblemente rápido y parece un poquito exagerado para señales de control que cambian lentamente, pero debes darte cuenta de que ciertos segmentos de los envolventes de amplitud cambian de manera extremadamente rápida. Los “controladores de alta resolución” pueden reducir el “ruido de cremallera” que resulta a veces de esas rápidas transiciones.

La figura 1.19 ilustra gráficamente el fenómeno conocido como “aliasing”. Aquí, se produce un “alias” o frecuencia espejo al subsamplear una determinada frecuencia. En este caso particular, la frecuencia de nuestra onda sinusoidal original es de 5 Hz. Estamos sampleando esta onda a 4 Hz (recuerda que el mínimo para obtener una reproducción fidedigna sería 10 Hz, es decir 2 veces la frecuencia más alta), obteniendo como resultado un sonido de 1 Hz. Como puedes ver en la figura, los valores devueltos por el proceso de muestreo trazan una onda sinusoidal de 1 Hz, no una de 5 Hz. La frecuencia “aliasing” real es la diferencia entre la frecuencia de muestreo y la frecuencia de la muestra (incluyendo los parciales).

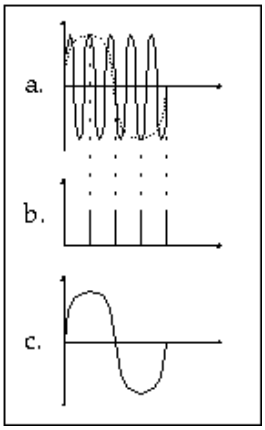


Figura 1.19 Aliasing. Una onda sinusoidal de 5 Hz (a) es subsampleada 4 veces por segundo (b) lo que da como resultado la reproducción incorrecta de una onda sinusoidal de 1 Hz (c).

Para experimentar y comprender en su totalidad este fenómeno, sería muy educativo volver a los primeros instrumentos de este capítulo y probar con variables a diferentes frecuencias (te recomiendo que dupliques y renumeres todos los instrumentos. Luego transforma todas las variables de audio *asig* y *a1* en variables de control *ksig* y *k1* y renderiza de nuevo. Te sorprenderán, e incluso agradarán, algunos de los resultados de "baja fidelidad"). Dejemos este tema por el momento y sigamos adelante.

Estudio de Diseño de Sonido 3: Cuatro Técnicas de Envolvente

Se dice a menudo que un ordenador es capaz de producir cualquier sonido imaginable. Y "matemáticamente" esto es cierto. Pero si lo es, ¿por qué son esos sonidos computerizados y sintéticos tan "estériles", "monótonos" y "apagados"? A mi oído, lo que hace a un sonido interesante y atractivo es el sutil, dinámico e interdependiente comportamiento de sus tres parámetros principales, altura, timbre y sonoridad. Y lo que hace que Csound sea un lenguaje de síntesis verdaderamente potente es el hecho de que uno puede literalmente conectar la salida de cualquier opcode a la entrada de virtualmente cualquier opcode o argumento, consiguiendo por consiguiente un grado insuperable de control dinámico de los parámetros. Modificando sutilmente (o dramáticamente) cada uno de los argumentos de entrada de tus opcodes, tus "sintéticos" y "computerizados" sonidos tomarán vida.

Hasta este punto lo que hemos hecho ha sido esencialmente "dar puerta" a nuestros instrumentos de Csound, simplemente poniéndolos a todo volumen. No estoy seguro de que algún instrumento acústico se comporte de esa manera. Lógicamente, aplicando alguna forma de control sobre el envolvente total de estos instrumentos avanzaríamos un gran trecho hacia el objetivo de hacerlos más "musicales". Y añadiendo otros "controles paramétricos dinámicos" renderizaríamos sonidos incluso más seductores.

En *instr 113*, representado en las figuras 1.20 y 1.21, se usa el opcode **linen** de Csound para controlar dinámicamente el parámetro de amplitud del oscilador, funcionando así como un típico generador de envolvente de ataque-caída.

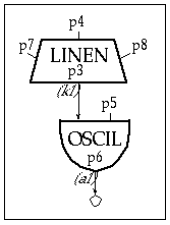


Figura 1.20 Diagrama de flujo de *instr 113*, un ejemplo de la salida de un opcode controlando un parámetro en la entrada de otro. En este caso, obtenemos el control dinámico de las amplitudes, modificando el parámetro de amplitud del opcode **oscil** con la salida de otro, el opcode **linen**.

```
        instr       113                 ; OSCILADOR SIMPLE CON ENVOLVENTE DINAMICO
k1      linen       p4, p7, p3, p8              ; P3=DUR, P4=AMP, P7=ATAQUE, P8=CAIDA
a1      oscil       k1, p5, p6                  ; P5=FREC, P6=FORMA DE ONDA
        out         a1
        endin
```

Figura 1.21 El código de la orquesta para *instr 113*, un instrumento con un oscilador simple con control sobre su envolvente de amplitud.

En *instr 115*, representado en las figuras 1.22 y 1.23, se usa un opcode **linen** para aplicar un envolvente de amplitud dinámico. Pero esta vez, el envolvente es conseguido multiplicando la salida del opcode **linen** (*k1*) por la salida del opcode **buzz** (*a1*). De hecho, la multiplicación se realiza en el argumento de entrada del opcode **out** (*k1 * a1*). Aquí no sólo vemos una manera diferente de aplicar un envolvente a una señal (multiplicándola por un controlador), sino también que es posible ejecutar operaciones matemáticas con las variables en los argumentos de un opcode.

En la figura 1.22, podemos observar también que en ese instrumento se usa un opcode **expon** para movernos exponencialmente desde el valor en *p10* al valor en *p11*, a lo largo de la duración de la nota (*p3*), barriendo el número de cosenos armónicos que **buzz** produce. El efecto es muy parecido a cerrar lentamente un filtro pasa bajos resonante y es otro método sencillo de obtener control dinámico sobre el timbre.

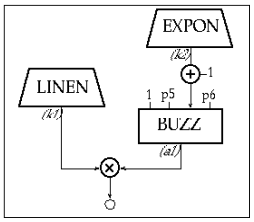


Figura 1.22 Diagrama de flujo de *instr 115* que muestra cómo controlar la amplitud multiplicando dos salidas y cómo controlar dinámicamente un argumento.

```
      Instr       115                 ; BARRIDO ESPECTRAL DE BUZZ CON ENVOLVENTE
k1    Linen       p4, p7, p3, p8
k2    Expon       p9, p3, p10
a1    Buzz        1, p5, k2+1, p6
      Out         k1*a1
      Endin
```

Figura 1.23 Código de la orquesta para *instr 115*, un instrumento con control dinámico sobre la amplitud y el contenido espectral.

Si has estado hojeando el Manual de Referencia de Csound probablemente notarás que muchos opcodes, como por ejemplo **oscil**, tienen versiones de tipo-**k** (control) y de tipo-**a** (audio). En *instr 117*, que se muestra en la figura 1.24, usamos un opcode **linen** a frecuencia de muestreo como generador de envolvente. Para hacerlo, pasamos la salida del opcode **grain** a la entrada de amplitud de **linen**, como se puede observar en los caracteres en negrita de la figura 1.25. Evidentemente esta aproximación usa **linen** para realizar el envolvente de la señal que viene del sintetizador granular (**grain**). De hecho, “envolvemos” literalmente la señal antes de mandarla como salida.

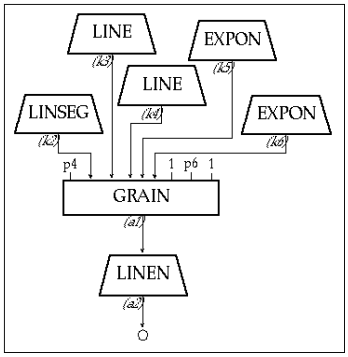


Figura 1.24 Diagrama de flujo de *instr 117* que muestra como controlar la amplitud pasando la señal (*a1*) a través de un envolvente a frecuencia de audio (*a2*).

```
      instr         117               ; GRANULOS PASADOS A TRAVES DE UN ENVOLVENTE
k2    linseg        p5, p3/2, p9, p3/2, p5
k3    line          p10, p3, p11
k4    line          p12, p3, p13
k5    expon         p14, p3, p15
k6    expon         p16, p3, p17
a1    grain         p4, k2, k3, k4, k5, k6, 1, p6, 1
a2    linen         a1, p7, p3, p8
      out           a2
      endin
```

Figura 1.25. Código de la orquesta para *instr 117*, un instrumento de síntesis granular con control dinámico sobre múltiples parámetros. Observa que la salida de **grain** (*a1*) se pasa al argumento de amplitud de un opcode **linen** trabajando a frecuencia de audio para modelar el sonido con un único envolvente de amplitud.

Envolventes

Tengo que admitir que cuando era un joven estudiante de música electrónica, siempre andaba un poco “confuso” por el uso del término “envolvente” en el mundo de la síntesis de audio. Pensaba en los “envolventes” como bolsas de plástico herméticas en las que podías meter la mozarella para que se conservara bien en el frigorífico y jamás pude entender el porqué de tal asociación mental. Pero el algoritmo usado en *instr 117* ejemplifica bastante bien esa metáfora al menos para mi y espero que para ti también. Aquí vemos que el opcode **linen** “envuelve” completamente la señal en esa “extraña bolsa” de ataque-caida y la envía a la salida (de la misma manera que envolvemos el queso y lo metemos en el frigorífico). La figura 1.26 es otra manera de visualizar el proceso. Primero observamos la señal bipolar de audio tal cual. Entonces observamos el envolvente de amplitud unipolar ADSR (del inglés *attack-decay-sustain-release*, ataque-decaimiento-sostenimiento-caída). A continuación vemos el envolvente aplicado a la señal de audio. En la fase final podemos observar la señal bipolar de audio cuya amplitud ha sido proporcionalmente modificada por el “perfil” del ADSR.

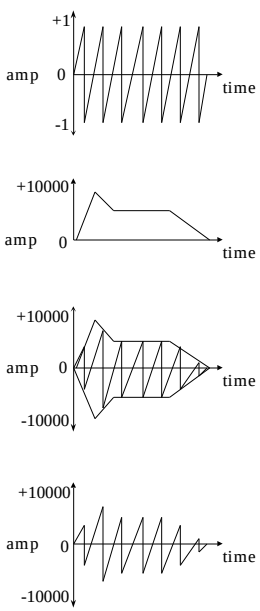


Figure 1.26 "Envolviendo" una señal.

Otra manera de mirar la figura 1.26 sería considerar que nuestra señal bipolar es escalada (multiplicada) por un envolvente ADSR unipolar que "perfila" simétricamente la señal unipolar. El resultado es que la señal unipolar es "envuelta" en el "plástico" del ADSR. Apliquemos este nuevo nivel de comprensión al diseño de un nuevo instrumento.

En *instr 118*, representado en la figura 1.27 y 1.28, ilustramos otra manera de aplicar un envolvente a una señal en Csound. En este caso, usamos un oscilador cuyo parámetro de frecuencia viene definido como *1/p3*. Usemos varios valores de ejemplo para averiguar cómo esta simple expresión nos ayuda a calcular la frecuencia de sub-audio correcta que transformará nuestro oscilador periódico en un generador de envolvente "aperiódico".

Por ejemplo, si la duración de la nota era de 10 segundos y la frecuencia de nuestro oscilador se definió, por tanto, como 1/10 Hz, se tardaría 10/10 Hz en "leer" completamente un ciclo de la tabla de función en *p7*. De esta forma, configurar la frecuencia de un oscilador como "1 dividido por la duración de la nota", o 1/*p3*, nos garantiza que dicho generador periódico de señal solamente generará 1 período, o lo que es lo mismo, sólo leerá un ciclo completo de su tabla de función durante el curso de cada nota.

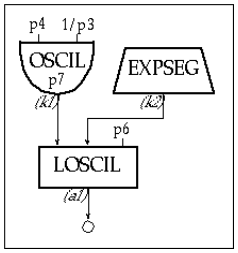


Figura 1.27 Diagrama de flujo de *instr 118*, un instrumento con un oscilador como generador de envolvente.

```
Instr        118                              ; LOSCIL CON UN ENVOLVENTE OSCIL
```

```
k1      oscil       p4, 1/p3, p7
k2      expseg      p5, p3/3, p8, p3/3, p9, p3/3, p5
a1      loscil      k1, k2, p6
        out         a1
        endin
```

Figura 1.28 Código de la orquesta para *instr 118*, un instrumento de reproducción de muestras que usa un oscilador para generar el envolvente y la modulación dinámica de frecuencia.

En *instr 118* las funciones de envolvente llamadas por *p7* (*f 6*, *f 7 and f 8*) usan **GEN7** y **GEN5** para dibujar varios contornos unipolares, tanto lineales como exponenciales. Es muy importante darse cuenta de que es *ilegal* usar un valor 0 en cualquier función exponencial, como las que son generadas por la subrutina **GEN5** o por el opcode **expseg**. Te darás cuenta, por tanto, que *f 8*, que usa **GEN5**, empieza y termina con un valor de .001 en vez de 0.

```
f 6    0   1024    7   0 10 1 1000 1 14 0                    ; ENVOLVENTE AR LINEAL
f 7    0   1024    7   0 128 1 128 .6 512 .6 256 0           ; ENVOLVENTE ADSR LINEAL
f 8    0   1024    5   .001 256 1 192 .5 256 .5 64 .001      ; ADSR EXPONENCIAL
```

Figura 1.29 Funciones de envolventes lineales y exponenciales usando **GEN5** y **GEN7**.

La técnica empleada para el envolvente de *instr 118* (un oscilador usado como un generador de envolvente) tiene varias ventajas. Primero, puedes crear una librería completa de presets de curvas de envolventes y cambiarlas de nota a nota si es necesario. Segundo, debido a que el generador de envolvente es de hecho un oscilador, puedes hacer "bucles" con el envolvente o "redispararlo" durante el transcurso de la nota para crear interesantes efectos de amplitud basados en el LFO. En *instr 119*, mostrado en la figura 1.30, *p8* determina el número de repeticiones que tendrán lugar durante el transcurso de la nota. Si damos a *p8* un valor 10 y *p3* es 5 segundos, el instrumento "disparará" el envolvente 2 veces por segundo. En cambio, si la duración de la nota fuera de 1 segundo (*p3* = 1), entonces el envolvente se dispararía 10 veces por segundo.

```
        instr       119                     ; REDISPARANDO FOSCIL CON UN ENVOLVENTE OSCIL
k1      oscil       p4, 1/p3 * p8, p7       ; P8=FRECUENCIA DE DISPARO POR DURACION DE LA NOTA
k2      line        p11, p3, p12
a1      foscil      k1, p5, p9, p10, k2, p6
        out         a1
        endin
```

Figura 1.30 Código de la orquesta para *instr 119*, un instrumento FM con un oscilador envolvente en el que *p8* determina la frecuencia de "disparo" de dicho envolvente.

Funciones Unipolares y Bipolares

Normalmente pensamos en un oscilador como algo que produce un "sonido" al "tocar" diferentes formas de onda o muestras. Sin embargo, hemos visto que el oscilador con búsqueda en tabla de Csound es capaz de leer cualquier función unipolar o bipolar a cualquier frecuencia. Obviamente, este generador de señal puede ser utilizado de la misma forma como fuente de control o como fuente de audio. A diferencia de los sintetizadores comerciales, en Csound la función de un opcode viene definida por el uso y el usuario. Hasta ahora, hemos estado usando varias rutinas **GEN** de Csound para generar funciones, tanto unipolares como bipolares, y es importante que nos aseguremos de que entendemos la diferencia.

La mayoría de las formas de onda de audio, como las que crea **GEN10**, son bipolares – moviéndose simétricamente por encima y por debajo del eje de abscisas. Por otra parte, la mayoría de las funciones de envolvente, como las que hemos creado usando **GEN5** y **GEN7**, son unipolares – moviéndose solamente en una dirección, normalmente positiva. En Csound, las funciones bipolares se normalizan por defecto en un rango de –1 a +1 y todas las funciones unipolares se normalizan en el rango de 0 a +1, como se muestra en la figura 1.31.

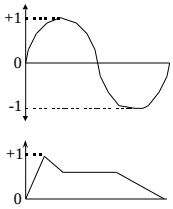


Figura 1.31 Funciones bipolar (de –1 a +1) y unipolar (de 0 a 1).

Si quieres ignorar el proceso de normalización por defecto de Csound, debes usar un signo (-) antes del número de la rutina **GEN**, como aparece en *f 3* y *f 4* en la figura 1.32.

```
f 1    0   512    10     1                       ; NORMALIZADA BIPOLAR SINUSOIDAL
f 2    0   512    7      0 6 1 500 1 6 0         ; NORMALIZADA UNIPOLAR ENVELOPE
f 3    0   512    -10    .3 .013 .147 .026       ; NON-NORMALIZED BIPOLAR SUM-OF-SINES
f 4    0   512    -7     440 256 220 256 440     ; NON-NORMALIZED UNIPOLAR ENVELOPE
```

Figura 1.32 Dos funciones normalizadas (*f 1* y *f 2*) y dos funciones no normalizadas (*f 3* and *f 4*).

Ejercicios para el Estudio 3

• Compila la tercera orquesta y partitura de Csound: *etude3.orc* y *etude3.sco*.
• Reproduce y escucha los diferentes timbres y curvas de envolvente de cada nota y cada instrumento.
• Modifica el fichero orquesta y cambia los nombres de las variables por unos más significativos. Por ejemplo, renombra todas las variables **a**1 como **a**sig1 y todas las variables **k**1 como **k**env1.
• Busca en el Manual de Referencia de Cosund, los opcodes que aparecen en *instr 113 – 119*:

```
kr    linen       kamp,     irise,      idur,     idec
ar    linen       xamp,     irise,      idur,     idec
kr    line        ia,       idur1,      ib
kr    expon       ia,       idur1,      ib
kr    linseg      ia,       idur1,      ib[,      idur2,       ic[...]]
kr    expseg      ia,       idur1,      ib[,      idur2,       ic[...]]
```

• Modifica la duración de ataque (*p7*) y de caída (*p8*) de los opcodes **linen** en los *instr 113 –117*.
• Añade un envolvente de altura a los *instr 113*, *114* y *115* añadiendo un opcode **linseg** a cada instrumento y sumando su salida a *p5*.
• Experimenta con los controles dinámicos de los parámetros de **grain** que aparecen el *instr 117*.
• En *instr 113 – 117*, substituye los envolventes basados en el oscilador **oscil** por los basados en el opcode **linen**.
• Usa **GEN5** y **GEN7** para diseñar varias funciones de envolvente adicionales. Intenta imitar las características de ataque de un piano – *f 9*, una mandolina – *f 10*, una tuba – *f 11*, un violín – *f 12* y una voz masculina cantando un "la" – *f 13*. Aplica estos envolventes a tus nuevas versiones de *instr 113 – 117*.
• Siguiendo los ejemplos de las figuras que has estudiado hasta ahora, dibuja diagramas de flujo para *instr 112*, *113*, *114* y *119*.

Estudio de Diseño de Sonido 4: Mezcla, Coro, Tremolo y Vibrato

A continuación mejoraremos la calidad de nuestros instrumentos primero mezclando y luego desafinando nuestros osciladores para crear un pronunciado efecto de "coro" (*chorus*). Después realizaremos un *crossfade* con los opcodes para crear un algoritmo de síntesis híbrida muy poco común en los productos comerciales. Finalmente, daremos vida a nuestros instrumentos introduciendo modulaciones de amplitud (AM) y frecuencia (FM), a frecuencias de sub-audio y de audio. También emplearemos varios de los opcodes de representación en pantalla (display) de

Csound para visualizar estos envolventes, más complejos tanto temporal como espectralmente. De paso, aprenderemos un poco más sobre el lenguaje.

En *instr 120*, representado en las figuras 1.33 y 1.34, mezclamos tres osciladores desafinados entre ellos pero usando todos el mismo opcode **envlpx** como envolvente de amplitud. Usando el opcode **display**, dicho envolvente será dibujado en pantalla, con una resolución configurada para trazar la curva de envolvente a lo largo de toda la duración de la nota (*p3*), representando, por tanto, la curva completa.

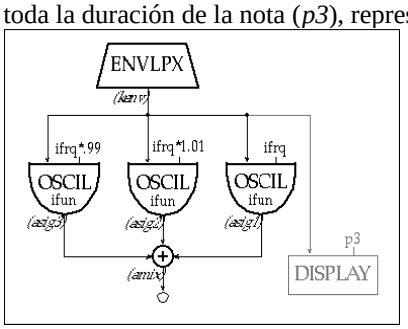


Figura 1.33 Diagrama de flujo de *instr 120* que ilustra tres osciladores en coro con un envolvente común y su representación en pantalla.

```
            instr       120                         ; CHORUSING SIMPLE
idur        =           p3                          ; BLOQUE DE INICIALIZACION
iamp        =           ampdb(p4)
ifrq        =           cpspch(p5)
ifun        =           p6
iatk        =           p7
irel        =           p8
iatkfun     =           p9
kenv        envlpx      iamp, iatk, idur, irel, iatkfun, .7, .01
asig3       oscil       kenv, ifrq*.99, ifun        ; BLOQUE DE SINTESIS
asig2       oscil       kenv, ifrq*1.01, ifun
asig1       oscil       kenv, ifrq, ifun
amix        =           asig1+asig2+asig3           ; MEZCLA
            out         amix
            display     kenv, idur
            endin
```

Figura 1.34 Código de la orquesta para *instr 120,* un instrumento con coro en el que damos nombres de variable a los campos-p. También empleamos un opcode **envlpx**, que se representa en pantalla, para dibujar un envolvente común.

Aunque *instr 120* es todavía bastante simple en su diseño, sirve como modelo de cómo instrumentos más complejos son normalmente "dispuestos" y organizados en Csound. En la figura 1.34 puedes ver que las variables son inicializadas al comienzo del instrumento y nombradas de manera que nos ayuden a identificar su función (lo que resulta en un estilo de código "auto-comentado"). Puedes leer con claridad que la duración del ataque se asigna a *iatk,* desde el valor de la partitura pasado por *p7* (*iatk = p7*), y que la duración de la caída se asigna a *irel,* desde el valor de la partitura pasado por *p9* (*irel = p9*). Y, más importante, al observar dónde son "conectados" en el opcode **envlpx,** puedes ver y recordar qué argumentos corresponden a esos parámetros particulares, haciendo por consiguiente que el opcode mismo sea más fácil de "leer".

Debes darte cuenta que en Csound el signo de igualdad (=) es en realidad el "operador de asignación". Es, de hecho, un opcode. Asignando mnemónicos en "Castellano llano" y nombres abreviados a las variables de tipo-**i,** conseguimos que nuestros instrumentos sean mucho más fáciles de leer, siendo por tanto ésta una práctica altamente recomendable.

Fusión Espectral

A continuación echaremos un vistazo a *instr 122*, que se muestra en las figuras 1.35 y 1.36. Este instrumento usa opcodes **expon** independientes para realizar un crossfade entre un opcode **foscil** y otro **buzz,** que son ambos "fundidos" (transformados/fusionados/transfigurados) con un ataque **pluck,** creando un precioso timbre híbrido. Este instrumento emplea el opcode **dispfft** de Csound para generar y representar en pantalla una Transformada Rápida de Fourier (FFT, del inglés Fast Fourier Transform) de 512 puntos y actualizada cada 250 milisegundos de la señal compuesta. Aunque los opcodes **display** y **dispfft** son una manera maravillosa de "investigar" el comportamiento de tus instrumentos, es importante observar que cuando estés usando tus instrumentos para hacer música, debes siempre acordarte de deshabilitar estos opcodes de impresión en pantalla (**display, print…**), poniendo un punto y coma (;) delante, porque influyen significativamente en el rendimiento de tu sistema. Son educativos e informativos pero realmente funcionan más como herramientas de depuración. Deberías pensar en ellos como tales.

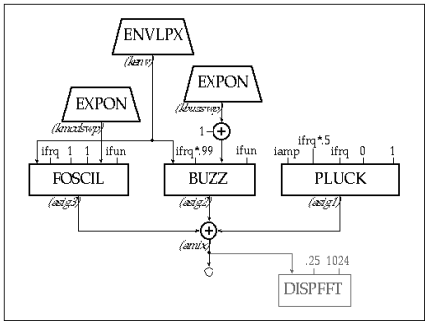


Figura 1.35. Diagrama de flujo de *instr 122* que ilustra la representación en pantalla de una FFT de la mezcla (fusión) y el crossfade ("morphing") de tres opcodes.

```
            instr       122           ; FUSION ESPECTRAL SIMPLE
idur        =           p3
iamp        =           ampdb(p4)
ifrq        =           cpspch(p5)
ifun        =           p6
iatk        =           p7
irel        =           p8
iatkfun     =           p9
index1      =           p10
index2      =           p11
kenv        envlpx      iamp, iatk, idur, irel, iatkfun, .7, .01
kmodswp     expon       index1, idur, index2
kbuzswp     expon       20, idur, 1
asig3       foscil      kenv, ifrq, 1, 1, kmodswp, ifun
asig2       buzz        kenv, ifrq*.99, kbuzswp+1, ifun
asig1       pluck       iamp, ifrq*.5, ifrq, 0, 1
amix        =           asig1+asig2+asig3
            out         amix
            dispfft     amix, .25, 1024
            endin
```

Figura 1.36 Código de la orquesta de *instr 122*, un instrumento que muestra la fusión de tres técnicas de síntesis, **pluck**, **foscil**, y **buzz**

En lugar de hacer simplemente un crossfade o mezclar opcodes, como hemos hecho en *instr 120* y *instr 122*, otra aproximación tradicional es *modular* un opcode de audio con la frecuencia y la amplitud de otro. En *instr 124*, mostrado en las figuras 1.37 y 1.38, por ejemplo, la amplitud de un opcode **oscil** funcionando a frecuencia de audio (*asig*) es modulada por la salida del barrido dinámico de un opcode **oscil**, también de tipo-**a** (*alfo*), cuya frecuencia es dinámicamente alterada por un opcode **line** y cuya amplitud es controlada por **expon**.

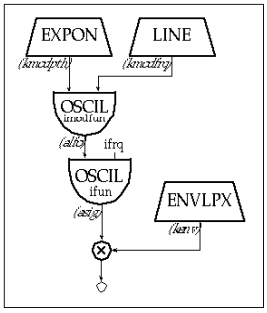


Figura 1.37 Diagrama de flujo de *instr 124*, un instrumento con modulación dinámica de amplitud.

```
            instr       124      ; BARRIDO EN LA MODULACION DE AMPLITUD
idur        =           p3
iamp        =           ampdb(p4)
ifrq        =           cpspch(p5)
ifun        =           p6
iatk        =           p7
irel        =           p8
iatkfun     =           p9
imodp1      =           p10
imodp2      =           p11
imodfr1     =           p12
imodfr2     =           p13
imodfun     =           p14
kenv        envlpx      iamp, iatk, idur, irel, iatkfun, .7, .01
kmodpth     expon       imodp1, idur, imodp2
kmodfrq     line        cpspch(imodfr1), idur, cpspch(imodfr2)
alfo        oscil       kmodpth, kmodfrq, imodfun
asig        oscil       alfo, ifrq, ifun
            out         asig*kenv
            endin
```

Figura 1.38 Código de la orquesta para *instr 124*, un instrumento con modulación de amplitud con envolvente de amplitud y LFO variable independientes.

Esta combinación de osciladores simples puede producir una gran variedad de timbres armónicos e inarmónicos que evolucionen dinámicamente.

A continuación, en *instr 126*, mostrado en las figuras 1.39 y 1.40, presentamos un instrumento con un vibrato simple que usa un opcode **linseg** para retrasar el comienzo de la modulación, lo que resulta en un efecto de vibrato más natural.

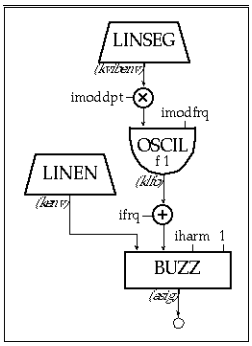


Figura 1.39 Diagrama de flujo de *instr 126*, un instrumento aditivo con vibrato retardado.

```
            instr       126           ; VIBRATO SIMPLE RETARDADO
idur        =           p3
iamp        =           ampdb(p4)
ifrq        =           cpspch(p5)
iatk        =           p6
irel        =           p7
ivibdel     =           p8
imoddpt     =           p9
imodfrq     =           p10
iharm       =           p11
kenv        linen       iamp, iatk, idur, irel
kvibenv     linseg      0, ivibdel, 1, idur-ivibdel, .3
klfo        oscil       kvibenv*imoddpt, imodfrq, 1
asig        buzz        kenv, ifrq+klfo, iharm, 1
            out         asig
            endin
```

Figura 1.40 Código de la orquesta para *instr 126*, un instrumento **buzz** con vibrato retardado.

Incluso estos diseños relativamente simples pueden prestarse a una increíblemente diversa y rica paleta de colores. Tómate tu tiempo para explorarlos y modificarlos.

Convertidores de Valor

En el "bloque" de inicialización de *instr 120*, mostrado en la figura 1.34 (y en todos los instrumentos de este estudio para dicha materia), puedes haber notado que se utilizaron dos de los *convertidores de valor* de Csound, **ampdb** y **cpspch** (*iamp = ampdb(p4)* y *ifrq = cpspch(p5)*). Dichos convertidores nos permiten expresar los datos de frecuencia y amplitud en un formato más familiar e intuitivo que tener que usar los valores en Hz y amplitudes lineales que habíamos usado hasta ahora.

El convertidor de valor **cpspch** leerá un número en notación *octava-punto-nota* y lo convertirá a Hz (por ejemplo, 8.09 = A4 = 440 Hz). La clasificación octava-punto-nota es un sistema de notación taquigráfica en el cual las octavas se representan como números enteros (8.00 = Do central o C4, 9.00 = C5, 10.00 = C6, etc.) y las alturas definidas por el sistema temperado de 12 notas se representan como dos cifras decimales que siguen a la octava (8.01 = C#4, 8.02 = D4, 8.03 = D#4, etc.). La escala mostrada en la figura 1.41 debería ayudarte a tener una idea suficientemente clara del sistema.

NOTA #	Hertz (Hz)	CPSPCH	NUMERO DE NOTA MIDI
C4	261.626	8.00	60
C#4	277.183	8.01	61
D4	293.665	8.02	62
D#4	311.127	8.03	63
E4	329.628	8.04	64
F4	349.228	8.05	65
F#4	369.994	8.06	66
G4	391.955	8.07	67
G#4	415.305	8.08	68
A4	440.000	8.09	69
A#4	466.164	8.10	70
B4	493.883	8.11	71
C5	523.251	9.00	72

Figura 1.41 Una escala cromática empezando en el Do central, especificada usando el convertidor de valor **cpspch** de Csound.

Añadiendo más cifras decimales es también posible especificar microtonos, como se muestra en la figura 1.42.

Como puedes ver, **cpspch** convierte de representación **pch** (octava-punto-nota) a **cps** (ciclos por segundo o Hz). Si estás escribiendo música tonal o microtonal con Csound, puedes encontrar este convertidor de valor particularmente útil.

```
NOTE #          CPSPCH

C4              8.00
C4+             8.005
C#4             8.01
C#4+            8.015
D4              8.02
D4+             8.025
D#4             8.03
D#4+            8.035
E4              8.04
E4+             8.045
F4              8.05
F4+             8.055
F#4             8.06
F#4+            8.065
G4              8.07
G4+             8.075
G#4             8.08
G#4+            8.085
A4              8.09
A4+             8.095
A#4             8.10
A#4+            8.105
B4              8.11
B4+             8.115
C5              9.00
```

Figura 1.42 Una octava de cuartos de tono en temperamento igual, especificada usando el convertidor de valor **cpspch.**

De la misma manera, el convertidor **ampdb** leerá un valor en decibelios y lo convertirá a un valor de amplitud normal como se muestra la figura 1.43.

```
ampdb(42)       =       125
ampdb(48)       =       250
ampdb(54)       =       500
ampdb(60)       =       1000
ampdb(66)       =       2000
ampdb(72)       =       4000
ampdb(78)       =       8000
ampdb(84)       =       16000
ampdb(90)       =       32000
ampdb(96)       =       64000         ; ATENCIÓN: MUESTRAS FUERA DE RANGO!!!
```

Figura 1.43 Conversión de amplitudes usando el convertidor de valor **ampdb**.

Deberías darte cuenta de que aunque la escala logarítmica en decibelios (dB) se percibe de manera lineal, Csound no "usa" realmente dB. El convertidor **ampdb** es una conversión directa sin escalado. Desgraciadamente, aún tendrás que pasar una gran cantidad de tu tiempo ajustando, normalizando y escalando los niveles de amplitud, incluso si estás usando el convertidor **ampdb** de Csound, porque la conversión se realiza antes del renderizado.

Ejercicios para el Estudio 4

• Renderiza la cuarta orquesta y partitura: *etude4.orc* & *etude4.sco*.
• Reproduce y escucha atentamente los efectos tímbricos y de modulación dinámicos de cada nota y cada instrumento.
• Modifica *instr 120* para que puedas hacer un efecto de coro con tres opcodes **foscil** en vez de tres **oscil**.
• De la manera en que se muestra en *instr 126*, añade un vibrato retardado a la versión **foscil** de *instr 120*.
• Realiza el diagrama de flujo de *instr 121* (que no ha sido discutido en esta sección) y añade un vibrato retardado más algunas de tus propias muestras a este sintetizador por tablas de onda.
• Modifica *instr 122* para crear sintetizadores híbridos totalmente diferentes. Quizás puedas añadir opcodes **grain** o **loscil**.
• Realiza el diagrama de flujo de *instr 123* (que no ha sido discutido en esta sección) y cambia los ritmos y las alturas. Intenta modulaciones en frecuencia de audio. Finalmente, crea y usa tu propio conjunto de funciones de modulación de amplitud.
• Modifica *instr 124* para que el barrido no sea tan radical. Añade efectos de coro y vibrato retardado.
• Realiza el diagrama de flujo de *instr 125* (que no ha sido discutido en esta sección). Cambia la frecuencia de modulación y la profundidad usando las funciones existentes. Modula algunas de tus propias muestras.
• Modifica *instr 126* para que las "voces sintéticas" canten melodías y armonías microtonales.
• Realiza el diagrama de flujo de *instr 127* (que no ha sido discutido en esta sección). Diviértete modificándolo con cualquiera de las técnicas y trocitos de código que has desarrollado y dominado hasta ahora.
• En el *Manual de Referencia de Csound*, busca los nuevos opcodes que figuran en *instr 120.127*:

```
kr          envlpx        kamp, irise, idur, idec, ifn, iatss, iatdec[, ixmod]
            print         iarg[, iarg, ...]
            display       xsig, iprd[, inprds][, iwtflg]
            dispfft       xsig, iprd, iwsiz[, iwtyp][, idbouti][, iwtflg]
```

• Crea un nuevo conjunto de funciones de ataque para **envlpx** y úsalas en todos los instrumentos.
• Añade opcodes **print**, **display** y **dispfft** a *instr 123-127* (Recuerda sin embargo deshabilitarlos cuando hagas producciones reales con tus instrumentos).

Un Poco de Teoría de Filtros

El siguiente estudio de diseño de sonido es una exploración introductoria de los opcodes de filtrado de Csound. Pero antes de ir demasiado lejos, puede ser de ayuda si revisamos algunos conceptos básicos acerca de los filtros. Cuatro de los tipos más comunes de filtros son: el pasa-altos, el pasa-bajos, el pasa-banda y el para-banda, mostrados en la figura 1.44. En esta figura, una señal consistente en 12 parciales armónicos de igual intensidad (a) es primero filtrada por un pasa-bajos de un solo polo (b), un pasa-altos también unipolar (c), un pasa-banda bipolar (d) y un filtro para-banda también de dos polos (e). Las líneas punteadas están en la banda de parada del filtro y las sólidas en la banda de pasada. La frecuencia de corte es el punto de -3 dB en cada una de las curvas del envolvente espectral representada por la línea sólida.

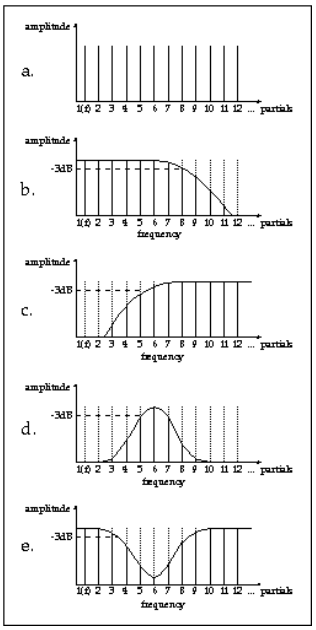


Figura 1.44 Una señal fuente (a) modificada por los cuatro tipos de filtros básicos: **tone -** un pasa-bajos unipolar (b), **atone -** un pasa-altos unipolar (c), **reson -** un pasa-banda bipolar (d) y **areson -** un para-banda bipolar (e).

En Csound, estos filtros corresponderían a los opcodes **tone** (b), **atone** (c), **reson** (d) and **areson** (e). Observa que la línea entrecortada en los -3 dB indica la frecuencia de corte del filtro. ¿Por qué -3 dB? Bueno, ya que la pendiente del filtro es de hecho continua, la frecuencia de corte (f_c) de un filtro debe estar "en algún punto de la curva", habiendo sido definida como el punto en dicho continuo de frecuencia en el cual la banda de pasada se atenúa 3 dB.

Estudio de Diseño de Sonido 5: Ruido, Filtros, Líneas de Retardo y Flangers.

La próxima serie de instrumentos emplea varios modificadores de señal de Csound, con distintas configuraciones en paralelo y en serie, para dibujar y transformar ruido y tablas de onda. En *instr 128*, mostrado en las figuras 1.45 y 1.46, filtraremos dinámicamente el "ruido blanco" producido por el opcode **rand** de Csound. Se usan opcodes **expon** y **line** por separado para modificar independientemente la frecuencia de corte y el ancho de banda del filtro pasa-banda bipolar **reson** de Csound. Al mismo tiempo, se usa un envolvente de amplitud, que se representa en pantalla.

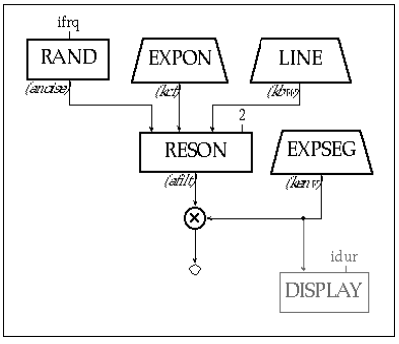


Figura 1.45 Diagrama de flujo de *instr 128*, un instrumento con ruido filtrado por un filtro pasa-banda.

```
        instr       128             ; RUIDO FILTRADO POR UN FILTRO PASA-BANDA
idur    =           p3
iamp    =           p4
ifrq    =           p5
iatk    =           p6
irel    =           p7
icf1    =           p8
icf2    =           p9
ibw1    =           p10
ibw2    =           p11
kenv    expseg      .01, iatk, iamp, idur/6, iamp*.4, idur(iatk+irel+idur/6), iamp*.6, irel,
                    .01
anoise  rand        ifrq
kcf     expon       icf1, idur, icf2
kbw     line        ibw1, idur, ibw2
afilt   reson       anoise, kcf, kbw, 2
        out         afilt*kenv
        display     kenv, idur
        endin
```

Figura 1.46 Código de la orquesta de *instr 128*, un instrumento con ruido filtrado por un filtro pasa-banda con una frecuencia de corte y un ancho de banda variable.

Una Red de Filtros en Cascada.

En *instr 129* a *132*, mostrados en la figura 1.47, una fuente de ruido blanco (**rand**) se pasa a través de una serie de filtros pasa-bajos unipolares (**tone**). La contribución significativa producida por cada polo adicional debería quedar bastante patente en esta serie de ejemplos. De hecho, cada polo incrementa la "pendiente" o "grado de inclinación" de un filtro en 6 dB por octava en la frecuencia de corte. Un diseño de filtros en cascada como este produce una pendiente proporcionalmente más abrupta con cada **tone** adicional, dando lugar de esta manera a un filtro más "efectivo". Por tanto, en nuestro "diseño en cascada", *instr 129* tendría una pendiente correspondiente a una atenuación de 6 dB por octava, *instr 130* tendría una pendiente de 12 dB por octava, *instr 131* una de 18 dB por octava e *instr 132* una de 24 dB por octava. El opcode **dispfft** debería mostrar claramente el efecto progresivo sobre el espectro del ruido en cada instrumento.

```
            instr       129         ; PASA-BAJOS UNIPOLAR
anoise      rand        ifrq
afilt       tone        anoise, kcut
            dispfft     afilt, idur, 4096

            instr       130         ; PASA-BAJOS BIPOLAR
anoise      rand        ifrq
afilt2      tone        anoise, kcut
afilt1      tone        afilt2, kcut
            dispfft     afilt1, idur, 4096

            instr       131         ; PASA-BAJOS TRIPOLAR
anoise      rand        ifrq
afilt3      tone        anoise, kcut
afilt2      tone        afilt3, kcut
afilt1      tone        afilt2, kcut
            dispfft     afilt1, idur, 4096

            instr       132         ; PASA-BAJOS TETRAPOLAR
anoise      rand        ifrq
afilt4      tone        anoise, kcut
afilt3      tone        afilt4, kcut
afilt2      tone        afilt3, kcut
afilt1      tone        afilt2, kcut
```

```
            dispfft     afilt1, idur, 4096
```

Figura 1.47. Fragmentos del código de la orquesta de *instr 129 - 132*, que pasan ruido blanco a través de una cascada de filtros pasa-bajos unipolares.

Representación en Pantalla

Durante los varios ejemplos vistos hasta ahora hemos estado usando los opcodes **display** y **dispfft** de Cosund para observar las señales. Pero, ¿qué está siendo representado exactamente? Y, ¿en qué son estos opcodes diferentes?

Como sabrás, las señales pueden ser representadas tanto en el dominio de la frecuencia como en el del tiempo. De hecho, ambas son representaciones complementarias que ilustran como la señal varía tanto en frecuencia como en amplitud a lo largo del tiempo. El opcode **display** de Csound dibuja las señales en el dominio temporal como un gráfico "amplitud versus tiempo", mientras que el opcode **dispfft** dibuja las señales en el dominio de la frecuencia usando un método conocido como *Transformación Rápida de Fourier*. Ambas nos permiten especificar con que frecuencia se actualizará la salida en pantalla y, por tanto, proporcionan los medios necesarios para observar la evolución de una señal, tanto en el dominio de la frecuencia como en el del tiempo, durante el transcurso de una nota. Así, usabamos **display** en *instr 128* para mirar la forma del envolvente de amplitud de **expseg** y ver la manera en que la amplitud variaba a lo largo de la duración de la nota.

En *instr 129 - 132* usamos el opcode **dispfft** para mirar la forma en que las frecuencias eran atenuadas por nuestra red de filtros. En nuestro diseño, especificando que la FFT (**F**ast **F**ourier **T**ransform, o Transformación Rápida de Fourier) fuera de 4096, dividíamos el espectro de frecuencias en 2048 cuadros de frecuencia, distribuidos linealmente, de aproximadamente 21.5 Hz cada uno (44100/2048 = 21.533), pero podríamos haber dividido el espectro desde 8 bandas (cada una de una anchura de 5512 Hz) hasta 2048 bandas (cada una de 21.5 Hz de ancho). Continuaremos usando estos opcodes para observar, en los dominios del tiempo y la frecuencia, las características de los sonidos que nuestros instrumentos producen. En particular, el opcode **dispfft** nos ayudará a comprender el efecto que los diferentes filtros de Cosund tienen en las señales que pasamos a través de ellos.

Los filtros **tone** y **reson** que hemos usado hasta ahora fueron de los primeros en implementarse en Csound. Se destacan por su eficiencia (se ejecutan rápidamente) e igualmente por su inestabilidad ("estallan"). De hecho, ha sido siempre un buen consejo pasar la salida de estos filtros al opcode **balance** de Csound para mantener las muestras fuera de rango bajo control.

A lo largo de los años, sin embargo, muchos nuevos filtros han sido añadidos al lenguaje de Csound. En los primeros días de la síntesis analógica eran los filtros los que definían el sonido de esos raros y añorados instrumentos "clásicos". Hoy día, en Csound, la familia de filtros *Butterworth* (**butterlp**, **butterhp**, **butterbp** y **butterbr**) suenan de maravilla y están llegando a ser muy comunes en virtualmente todos los diseños instrumentales. Esto es debido en parte al hecho de que los filtros Butterworth tienen: más polos (son más abruptos y por tanto más efectivos en el filtrado), una respuesta a la frecuencia más plana en la banda de pasada (son más suaves y suenan más limpios) y son significativamente más estables (lo que significa que no tienes que preocuparte demasiado por las muestras fuera de rango). En *instr 133*, mostrado en las figuras 1.48 y 1.49, usamos una configuración en paralelo compuesta de un par de filtros **butterbp** de cuatro polos y **butterlp** de cuatro polos como una forma de modelar el filtro pasa-bajos resonante "clásico" que se encuentra normalmente en la primera generación de sintetizadores analógicos.

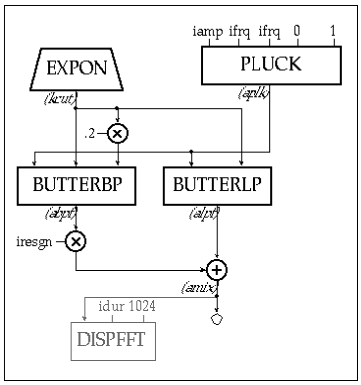


Figura 1.48. Diagrama de flujo de *instr 133*, una configuración de **butterbp** y **butterlp** en paralelo que da lugar a un diseño "clásico" de un filtro pasa-bajos resonante.

```
          instr         133           ; PASA-BAJOS CON RESONANCIA
idur      =             p3
iamp      =             ampdb(p4)
ifrq      =             p5
icut1     =             p6
icut2     =             p7
iresgn    =             p8
kcut      expon         icut1, idur, icut2
aplk      pluck         iamp, ifrq, ifrq, 0, 1
abpf      butterbp      aplk, kcut, kcut*.2
alpf      butterlp      aplk, kcut
amix      =             alpf+(abpf*iresgn)
          out           amix
          dispfft       amix, idur, 1024
          endin
```

Figura 1.49 Código de la orquesta para *instr 133*, un diseño "clásico" de un filtro pasa-bajos resonante.

Como puedes ver y escuchar por la serie previa de ejemplos, el control dinámico paramétrico de los opcodes de filtrado de Csound, combinados en distintas configuraciones en paralelo y en serie, abre la puerta a un amplio mundo de posibilidades en el diseño substractivo de sonido.

Un Resonador con Eco

Un filtro **comb** (peine) es esencialmente una línea de retardo con retroalimentación, como se ilustra en la figura 1.50. Como puedes ver, la señal entra en la línea de retardo y su salida es retardada según la longitud de la línea (25 milisegundos en este caso). Cuando alcanza la salida, es enviada de vuelta a la entrada después de haber sido multiplicada por un factor de ganancia.

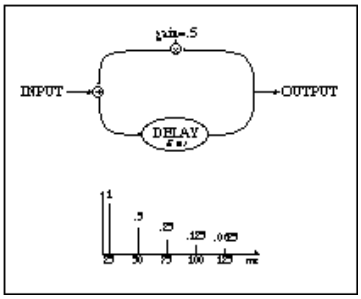


Figura 1.50 Diagrama de flujo de un filtro peine **comb** y su respuesta a la frecuencia.

Al tiempo que tarda la señal en circular de vuelta a la entrada se le llama duración del bucle (*loop-time).* Como se demostró en *instr 135*, mostrado en las figuras 1.51, 1.52 y 1.53, un opcode **diskin** se usa para reproducir (tanto hacia delante como hacia atrás) las muestras y pasarlas directamente del disco al filtro **comb**. Cuando el *loop-time* es largo percibimos ecos discretos, pero cuando es corto el filtro **comb** funciona más como un resonador. Como se muestra en la figura 1.50, la respuesta al impulso de un filtro peine (**comb**) es un tren de impulsos uniformemente distribuidos en el tiempo, en el intervalo de la duración del bucle. De hecho, la frecuencia resonante de este filtro es 1/*loop-time* (la inversa de la duración del bucle). En *instr 135*, esto viene especificado en milisegundos. En los comentarios de la partitura verás dónde convertimos el período del bucle, especificado en milisegundos, en la frecuencia del resonador, especificada en Hz.

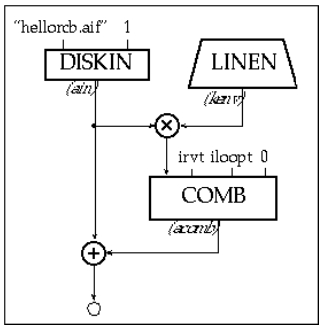


Figura 1.51 Diagrama de flujo de *instr 135*, un instrumento con una línea de retardo y un resonador aplicados sobre un fichero de sonido, usando **diskin** para leer directamente del disco sin el uso de una tabla de función y el filtro **comb** para retardar o resonar.

```
            instr       135           ; ECO-RESONADOR CON DISKIN
idur        =           p3
iamp        =           p4
irvt        =           p5
iloopt      =           p6
kenv        linen       iamp, .01, idur, .01
ain         diskin      "hellorcb.aif", 1
acomb       comb        ain*kenv, irvt, iloopt, 0
            out         ain+acomb
            endin
```

Figura 1.52 Código de la orquesta para *instr 135*, un instrumento con resonancia y eco, usando el opcode **comb**.

```
; INS      ST     DUR    AMP      GANA    LOOPTIME        FRECUENCIA RESONANTE
                                   N
```

```
i 135      0      5      .4      10      .5            ; 1/.5          =     2 Hz
i 135      5      5      .3      5       .25           ; 1/.25         =     4 Hz
i 135      10     5      .3      5       .125          ; 1/.125        =     8 Hz
i 135      15     5      .2      2       .0625         ; 1/.0625       =     16 Hz
i 135      20     5      .2      2       .03125        ; 1/.03125      =     32 Hz
i 135      25     5      .2      2       .015625       ; 1/.015625     =     64 Hz
i 135      30     5      .04     2       .001          ; 1/.001        =    1000 Hz
```

Figura 1.53 Código de la partitura de *instr 135*; la duración del bucle (*p6*) especifica el período y la frecuencia resonante de la línea de retardo retroalimentada.

Aunque podemos variar la duración del bucle en *instr 135* de nota a nota, por su diseño, el opcode **comb** no te permitirá variar dinámicamente este parámetro durante el transcurso de un a nota. Pero el opcode **vdelay** sí. Y las líneas de retardo variables son la clave para el diseño de uno de los más populares procesadores de efectos - el "*flanger*".

En *instr 136*, mostrado en las figuras 1.54 y 1.55, el ruido cae a través de una serie de líneas de retardo variables para conseguir un "flanger". Pasando la salida del opcode **vdelay** a la entrada de otro, la fuerza y el foco de la resonancia característica son enfatizadas (justo al igual que en nuestro ejemplo de **tone** anterior). Además, este pico de resonancia se barre a lo largo de todo el espectro de frecuencias, bajo el control de una variable de baja frecuencia (LFO) cuya frecuencia es dinámicamente modificada por el opcode **line**.

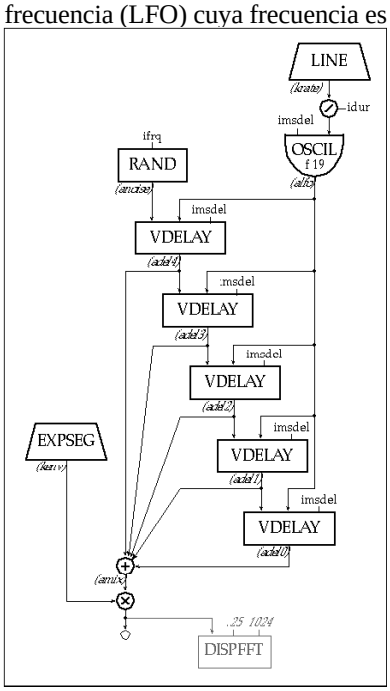


Figura 1.54 Diagrama de flujo de *instr 136*, un "flanger" con **vdelay**.

```
            instr       136             ; FLANGER CON VDELAY
idur        =           p3
iamp        =           p4
ifrq        =           p5
iatk        =           p6
irel        =           p7
irat1       =           p8
irat2       =           p9
imsdel      =           p10
kenv        expseg      .001, iatk, iamp, idur/8, iamp*.3, idur-(iatk+irel+idur/8), iamp*.7,
```

```
                    irel, .01
krate       line    irat1, idur, irat2
alfo        oscil   imsdel, krate/idur, 19
anoise      rand    ifrq
adel4       vdelay  anoise, alfo, imsdel
adel3       vdelay  adel4, alfo, imsdel
adel2       vdelay  adel3, alfo, imsdel
adel1       vdelay  adel2, alfo, imsdel
adel0       vdelay  adel1, alfo, imsdel
amix        =       adel0+adel1+adel2+adel3+adel4
            out     kenv*amix
            dispfft amix, idur, 1024
            endin
```

Figura 1.55 Código de la orquesta para *instr 136*, un instrumento "flanger" con una línea retardo variable.

Sentencias de la Partitura y Atajos en la Lista de Notas.

De acuerdo con que crear y editar listas de notas en el editor de texto no es divertido. Concedido que la lista de notas realmente ofrece el control más exacto y directo sobre el comportamiento de tus instrumentos, pero aún es uno de los aspectos más tediosos y amusicales de trabajar con Csound.

Como se afirmó al principio de este capítulo, Csound lee ficheros MIDI y esto puede resultar ser una manera más intuitiva de generar notas y tocarlas con tus instrumentos de Csound. Sin embargo, para hacerlo así, los instrumentos de Csound deben ser específicamente diseñados para trabajar con MIDI y necesitarás adaptar tus instrumentos tradicionales antes de que puedan trabajar con dispositivos MIDI.

Aunque no se cubre en el texto del Libro de Csound, hay varios capítulos en el CD-ROM dedicados a controlar Csound desde teclados y ficheros MIDI. De hecho, he escrito un complemento a este capítulo titulado *Una Introducción al Diseño de Instrumentos basados en MIDI con Csound* para el CD-ROM que espero te ayude a desarrollar algunos buenos instrumentos MIDI.

Aún así, sin recurrir al MIDI, Csound presenta una colección de **Sentencias y Símbolos de Partitura** (atajos basados en texto) que fueron creados para simplificar el proceso de crear y editar listas de notas. Como las sentencias-f, estos comandos de la partitura empiezan por una letra específica y son, a veces, seguidos por una serie de argumentos. Empleamos muchos de ellos en *etude5.sco.*

La primera sentencia de partitura que empleamos en *etude5.sco* es la **Sentencia de Avance -*a,*** que se muestra en la figura 1.56. La sentencia de avance permite adelantar la cuenta compases de una partitura sin generar ninguna muestra de sonido. Aquí se usa para saltarnos las dos primeras notas de la partitura y empezar el renderizado a los 10 segundos de haber empezado la "pieza". La sentencia de avance puede ser particularmente útil cuando se trabaja en una composición extensa y compleja y queremos retocar algún sonido por ejemplo a mitad o al final de la pieza. En vez de esperar que se renderice la obra entera sólo para escuchar la última nota, podemos avanzar hasta el final de la pieza y renderizar sólo esa sección, ahorrándote horas y horas. La sintaxis de la sentencia de avance se muestra en los comentarios de la figura 1.56.

```
; AVANCE      SIN             INSTANTE DE         DURACIÓN DEL SALTO
              SIGNIFICADO     COMIENZO DEL
                              SALTO
a             0               0                   10

; INS        ST      DUR       AMP       FRQ       ATK       REL       CF1       CF2       BW1       BW2

i 128        1       5         .5        20000     .5        2         8000      200       800       30
i 128        6       5         .5        20000     .25       1         200       12000     10        200

i 128        10      5         .5        20000     .5        2         8000      200       800       30
i 128        14      5         .5        20000     .25       1         200       12000     10        200
```

```
i 128       18      3         .5       20000     .15      .1       800      300      300      40
```

Figura 1.56 Fragmento de la partitura de *etude5.sco* que presenta el uso de la **sentencia de avance**.

La segunda sentencia de partitura que emplearemos en *etude5.sco* es la **Sentencia de Sección -*s***, mostrada en la figura 1.57. La sentencia de sección no tiene argumentos. Simplemente divide una partitura en secciones y te permite empezar a contar de nuevo desde 0. Esto es particularmente útil si queremos repetir un pasaje. Para hacerlo, simplemente insertaríamos una **s** al final de la primera sección, copiaríamos la sección, y la pegaríamos después de la **s**. La figura 1.57, de nuevo de *etude5.sco*, muestra exactamente este uso.

```
; INS       ST          DUR         AMP         FRQ         ATK         REL         CUT1        CUT2

i 129       0           1.5         3           20000       .1          .1          500         500
i 130       2           1.5         3           20000       .1          .1          500         500
i 131       4           1.5         3           20000       .1          .1          500         500
i 132       6           1.5         3           20000       .1          .1          500         500
i 129       8           1.2         1           20000       .01         .01         5000        40
i 130       11          1.2         1           20000       .01         .01         5000        40
i 131       12          1.2         1           20000       .01         .01         5000        40
i 132       13          1.2         1           20000       .01         .01         5000        40
s
; INS       ST          DUR         AMP         FRQ         ATK         REL         CUT1        CUT2

i 129       0           1.5         3           20000       .1          .1          500         500
i 130       2           1.5         3           20000       .1          .1          500         500
i 131       4           1.5         3           20000       .1          .1          500         500
i 132       6           1.5         3           20000       .1          .1          500         500
i 129       8           1.2         1           20000       .01         .01         5000        40
i 130       11          1.2         1           20000       .01         .01         5000        40
i 131       12          1.2         1           20000       .01         .01         5000        40
i 132       13          1.2         1           20000       .01         .01         5000        40
s
```

Figura 1.57 Procedimiento de cortar y pegar para repetir un fragmento de la partitura de *etude5.sco*, presentando el uso de la **sentencia de sección.**

La tercera sentencia de partitura que emplearemos en *etude5.sco*, es la **Sentencia-f Falsa - *f 0*,** mostrada en la figura 1.58. En Csound puedes usar la partitura para cargar tablas de función en memoria en cualquier momento. Esto te permitiría reemplazar una forma de onda o muestra por otra durante el transcurso de una pieza y, aún así, referirlas con el mismo número de tabla en la orquesta. Así mismo, puedes también cargar una tabla de función falsa (*f 0*) en cualquier momento de la partitura como medio de extender la longitud de una sección particular o insertar silencios entre secciones. Como se ilustra en la figura 1.58, usamos una sentencia *f 0* para insertar dos segundos de silencio entre dos secciones de la partitura.

```
; INS       COMIEN    DUR         AMP         FREC        ATAQUE      CAIDA       CORTE1      CORTE2

i 129       0         1.5         3           20000       .1          .1          500         500
i 130       2         1.5         3           20000       .1          .1          500         500
i 131       4         1.5         3           20000       .1          .1          500         500
i 132       6         1.5         3           20000       .1          .1          500         500
i 129       8         1.2         1           20000       .01         .01         5000        40
i 130       11        1.2         1           20000       .01         .01         5000        40
i 131       12        1.2         1           20000       .01         .01         5000        40
i 132       13        1.2         1           20000       .01         .01         5000        40

s
f 0         2         ; FALSA SENTECIA F: DOS SEGUNDOS DE SILENCIO ENTRE SECCIONES
s

; INS       ST      DUR     AMP     FRQ         ATK      REL      CF1       CF2       BW1       BW2
```

```
i 128      0      5      .5      20000      .5      2      8000      200      800      30
i 128      4      5      .5      20000      .25     1      200       12000    10       200
i 128      8      3      .5      20000      .15     .1     800       300      300      40
i 128      10     11     .5      20000      1       1      40        90       10       40
s
```

Figura 1.58 Un fragmento de la partitura *etude5.sco*, que presenta la **falsa sentencia *f 0*,** usada para insertar dos segundos de silencio entre dos secciones de la partitura.

La cuarta serie de atajos de la partitura que emplearemos en *etude5.sco* son los **Símbolos Carry** y **Ramp**, mostrados en la figura 1.59. El símbolo *carry* o acarreo (**.**) copia el valor de un campo-p de una sentencia de nota a la siguiente. El símbolo *ramp* o rampa (<) interpola linealmente los valores de dos campos-p a lo largo de cualquier número de notas (el número de notas determina el número de puntos en la interpolación). El símbolo + sólo funciona para *p2*. Calcula automáticamente el instante de comienzo de la nota actual, añadiendo a su duración (*p2 + p3*) el instante de comienzo de la nota previa. De esta forma, la nota actual será literalmente consecutiva respecto de la nota precedente. Los tres símbolos se usan en la figura 1.59 y la traducción de este sistema taquigráfico se muestra en la figura 1.60.

```
; INS      COM     DUR      AMPDB      FREC        FC1         FC2        GANANCIA DE RESON

i 134      0       .1       90         8.09        8000        80         1
i .        +       .        <          8.095       <           <          <
i .        .       .        .          8.10        .           .          .
i .        .       .        .          8.105       .           .          .
i .        .       .        .          8.11        .           .          .
i .        .       .        .          8.115       .           .          .
i .        .       .        .          9.00        .           .          .
i .        .       .        .          9.005       .           .          .
i .        .       .        .          9.01        .           .          .
i .        .       .        .          9.015       .           .          .
i .        .       .        80         9.02        9000        60         50
```

Figura 1.59 Un fragmento de la partitura *etude5.sco*, **símbolos de acarreo (.), incremento (+) y rampa (<).**

```
; INS      COM     DUR      AMPDB      FREC        FC1         FC2        GANACIA DE RESON

i 134      0       .1       90         8.09        8000        80         1
i 134      .1      .1       89         8.095       8100        78         5
i 134      .2      .1       88         8.10        8200        76         10
i 134      .3      .1       87         8.105       8300        74         15
i 134      .5      .1       86         8.11        8400        72         20
i 134      .5      .1       85         8.115       8500        70         25
i 134      .6      .1       84         9.00        8600        68         30
i 134      .7      .1       83         9.005       8700        66         35
i 134      .8      .1       82         9.01        8800        64         40
i 134      .9      .1       81         9.015       8900        62         45
i 134      1       .1       80         9.02        9000        60         50
```

Figura 1.60 Otra vista del fragmento de *etude5.sco* mostrado en la figura 1.59 en el cual los símbolos **rampa (<), acarreo (.)** y **+** son reemplazados por los valores numéricos reales que representan.

La última sentencia de partitura que emplearemos en *etude5.sco* es la **Sentencia de Tempo - *t*,** mostrada en la figura 1.61. El reloj de la partitura de Csound corre a 60 pulsos por minuto. Por defecto, Csound inserta una sentencia de tempo de 60 (60 pulsos por minuto o 1 pulso por segundo) al principio de cada partitura (*t 0 60*). Obviamente, esto significa que cuando especificas una duración de 1 en *p3*, la nota durará 1 segundo. Afortunadamente, la sentencia-t te permite cambiar este valor por defecto, tanto en forma de variable como en forma de constante. La figura 1.61 ilustra ambos usos.

La sentencia *t 0 120* especificará un tempo constante de 120 pulsos por minuto. Dada esta configuración, el pulso del reloj interno correrá dos veces más rápido y, por tanto, todos los valores en el fichero partitura serán reducidos a la mitad.

La sentencia *t 0 120 1 30* se usa para establecer una variable de tempo. En este caso, el tempo es establecido en 120 en el instante 0 (el doble de rápido que indica la partitura) y tarda un segundo en pasar gradualmente a un nuevo tempo de 30 (el doble de lento que indica la partitura). No hace falta decir que un tempo variable puede hacer tus partituras mucho menos mecánicas y más musicales.

```
; t     0     120                  ; SENTENCIA DE TEMPO FIJO: EL DOBLE DE RAPIDO

; INS   COM   DUR    AMPDB    FREC        FC1         FC2        GANANCIA DE RESON

i 134   0     .1     90       8.09        8000        80         1
i .     +     .      <        8.095       <           <          <
i .     .     .      .        8.10        .           .          .
i .     .     .      .        8.105       .           .          .
i .     .     .      .        8.11        .           .          .
i .     .     .      .        8.115       .           .          .
i .     .     .      .        9.00        .           .          .
i .     .     .      .        9.005       .           .          .
i .     .     .      .        9.01        .           .          .
i .     .     .      .        9.015       .           .          .
i .     .     .      80       9.02        9000        60         50
s
; t     0     120    1     30       ; TEMPO VARIABLE: DEL DOBLE DE RAPIDO AL DOBLE DE LENTO

; INS   COM   DUR    AMPDB    FREC        FC1         FC2        GANANCIA DE RESON

i 134   0     .1     90       8.09        8000        80         1
i .     +     .      <        8.095       <           <          <
i .     .     .      .        8.10        .           .          .
i .     .     .      .        8.105       .           .          .
i .     .     .      .        8.11        .           .          .
i .     .     .      .        8.115       .           .          .
i .     .     .      .        9.00        .           .          .
i .     .     .      .        9.005       .           .          .
i .     .     .      .        9.01        .           .          .
i .     .     .      .        9.015       .           .          .
i .     .     .      80       9.02        9000        60         50
s
```

Figura 1.61 Un fragmento del final de *etude5.sco* en el que la **sentencia de tempo** se usa de manera fija y variable.

Trabajar con el lenguaje de partitura basado en texto de Csound puede ser extremadamente laborioso. De hecho, ha inspirado a muchos estudiantes a aprender a programar en C para generar sus propias listas de notas algorítmicamente. La ejecución en tiempo real y el MIDI son soluciones a esto. Pero aprovechar los atajos de la partitura de Csound puede hacer tu trabajo mucho más fácil y tus gestos musicales, frases y texturas mucho más expresivas.

Ejercicio para el Estudio 5

• Renderiza la orquesta y partitura: *etude5.orc* & *etude5.sco.*
• Reproduce y escucha atentamente las diferentes cualidades del sonido del los distintos filtros y configuraciones de filtrado.
• Busca y lee sobre los nuevos opcodes usados en *instr 128 – 136* en el *Manual de Referencia de Csound.*

```
ar          rand          xamp[, iseed]
ar          tone          asig, khp[, istor]
ar          butterlp      asig, kfreq[, iskip]
```

```
ar          butterbp      asig, kfreq, kband[, iskip]
ar          delayr        idlt[, istor]
ar          comb          asig, krvt, ilpt[, istor]
ar          vdelay        asig, adel, imaxdel [, iskip]
a1[, a2[,   diskin        ifilcod, kpitch[, iskiptim][, iwraparound][, iformat]
  a3, a4]]
```

• En *instr 128*, sustituye el opcode **loscil** por el opcode **rand** y filtra dinámicamente algunas de tus muestras.
• En *instr 128*, sustituye un **butterbp** por un **reson** y escucha la diferencia de calidad.
• Sustituye los filtros **butterlp** por los filtros **tone** en *instr 129 - 132*. Compara la efectividad de ambos.
• Transforma *instr 133* en un instrumento con un filtro pasa-altos resonante.
• Haz un instrumento que combine el diseño en serie de filtros de *instr 132* con el diseño en paralelo del *instr 133*.
• Haz el diagrama de flujo del *instr 134*, un instrumento con una línea de retardo (no discutido en el texto).
• Añadiendo más opcodes **delay** transforma *instr 134* en un instrumento con una línea de retardo múltiple.
• Modifica *instr 135* para hacer un resonador multibanda.
• Añade más **comb**s y **vdelay**s a *instr 135* y crea una línea de retardo múltiple con un super-flanger-resonante-multibanda-reatroalimentado.
• Usando las sentencias de partitura que se cubren en esta sección, vuelve a los estudios 3 y 4. En ellos, repite alguna sección, inserta algunos silencios, cambia el tiempo durante las secciones, y usa las sentencias de avance y rampa en algunos parámetros para explorar mejor el rango de posibilidades que estos instrumentos ofrecen.
• En *instr 136*, sustituye un opcode **diskin** por el opcode **rand** y pásalo por un flanger.
• En *instr 136*, añade y explora distintas modificaciones dinámicas de frecuencia y amplitud del oscilador de control.
• Cambia las formas de onda del oscilador de control en *instr 136* (¡Prueba con **randh**!)
• Añade un filtro pasa-bajos resonante a tus instrumentos con flanger modificados.
• Vete a dar un paseo y escucha el mundo a tu alrededor.

Variables Globales

Hasta ahora, hemos usado sólo variables de tipo-**i,** de tipo-**k** y de tipo-**a** locales. Estas han sido "locales" a un instrumento. Las variables locales son geniales porque puedes usar el mismo nombre de variable en instrumentos separados sin tener que preocuparte de que los datos de las señales *asig* o *amix* se corrompan o se "derramen" desde un instrumento a otro. De hecho, los delimitadores **instr** y **endin** aislan realmente los bloques de procesado de señal los unos de los otros, incluso si tienen exactamente las mismas etiquetas y nombres de argumentos.

Sin embargo, hay veces en las que te gustaría ser capaz de comunicar instrumentos. Esto haría posible pasar la señal de salida de un instrumento de síntesis a un instrumento de reverberación, de manera similar a como uno pasa las señales de una mesa de mezclas a una unidad de efectos, usando los envíos y los retornos auxiliares. En Csound esta misma operación se consigue mediante las *variables globales.* Las variables globales son aquellas que son accesibles por todos los instrumentos. Y al igual que las variables locales, las globales se actualizan a las cuatro frecuencias básicas: **configuración**, **gi**, **gk** y **ga**, donde:

las variables de tipo-**gi** se cambian y actualizan cada nueva nota.
las variables de tipo-**gk** se cambian y actualizan a frecuencia de control.
las variables de tipo-**ga** se cambian y actualizan a frecuencia de audio.

Debido a que una variable global pertenece a todos los instrumentos y a ninguno, deben ser *inicializadas*. Una variable global es normalmente inicializada en el **instrumento 0** y "rellenada" desde un instrumento "local". ¿Dónde está este misterioso "instrumento 0"? Bueno, el instrumento 0 lo forman en verdad las líneas en el fichero orquesta inmediatamente después de la sección de cabecera y antes de la declaración del primer *instr*. Así, en la figura 1.62, justo después de la cabecera, en el instrumento 0, se limpian e inicializan a 0 las variables *gacmb* y *garvb* (nuestros dos buses globales de efectos).

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

```
gacmb       init     0
garvb       init     0


            instr    137
```

Figura 1.62 Las dos variables globales *gacmb* y *garvb* se inicializan en el **instrumento 0** después de la cabecera y antes del primer *instr*.

Estudio de Diseño de Sonido 6: Reverberación y Panoramización

Pongamos las variables globales en uso y añadamos algún efecto de procesado "externo" a nuestros instrumentos.

Desde dentro de *instr 137*, mostrado en las figuras 1.63 y 1.64, la señal "seca" de **loscil** se añade (se mezcla) a la señal "húmeda" en un bus de reverberación y eco separado.

```
asig        loscil      kenv, ifrq, ifun
            out         asig
garvb       =           garvb+(asig*irvbsnd)
gacmb       =           gacmb+(asig*icmbsnd)
```

Observa que la señal "seca" es aún enviada directamente como salida, usando el opcode **out**, justo como lo hemos estado haciendo desde nuestro primer instrumento. Pero, en este caso, la misma señal se pasa globalmente "fuera del instrumento" y adentro de otros dos, en este caso a *instr 198* (eco) e *instr 199* (reverb) como se muestra en las figuras 1.63 y 1.64.

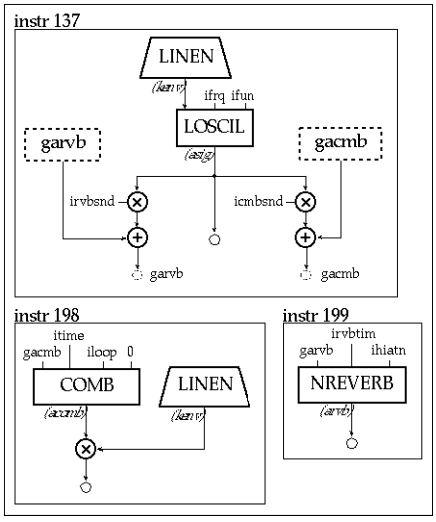


Figura 1.63 Diagrama de flujo para *instr 137*, *198 y 199*, un instrumento con síntesis de tabla de ondas (*instr 137*) y dos efectos globales (*instr 198* y *199*).

```
            instr       137           ; GLOBAL COMB/VERB LOSCIL
idur        =           p3
iamp        =           ampdb(p4)
ifrq        =           cpspch(p5)
ifun        =           p6
iatk        =           p7
irel        =           p8
irvbsnd     =           p9
```

```
icmbsnd     =           p10
kenv        linen       iamp, iatk, idur, irel
asig        loscil      kenv, ifrq, ifun
            out         asig
garvb       =           garvb+(asig*irvbsnd)
gacmb       =           gacmb+(asig*icmbsnd)
            endin

            instr       198           ; GLOBAL ECHO
idur        =           p3
itime       =           p4
iloop       =           p5
kenv        linen       1, .01, idur, .01
acomb       comb        gacmb, itime, iloop, 0
            out         acomb*kenv
gacmb       =           0
            endin

            instr       199           ; GLOBAL REVERB
idur        =           p3
irvbtim     =           p4
ihiatn      =           p5
arvb        nreverb     garvb, irvbtim, ihiatn
            out         arvb
garvb       =           0
            endin
```

Figura 1.64 Código de la orquesta para tres instrumentos que trabajan juntos para añadir reverberación (*instr 199*) y eco (*instr 198*) a un oscilador en bucle (*instr 137*).

Es importante observar que, en el fichero partitura (figura 1.65), los tres instrumentos deben estar activos. De hecho, para evitar ruidos transitorios, los instrumentos globales se dejan normalmente activos durante toda la sección y las variables globales se "limpian" siempre que el instrumento es desactivado (*gacmb = 0* and *garvb = 0*)

```
; INS       COM       DUR       TIEMPO REVERB     ATENUACION DE LAS ALTAS FRECUENCIAS
i 199       0         12        4.6               .8

; INS       COM       DUR       TIEMPO            DURACION DEL BUCLE
i 198       0         6         10                .8
i 198       0         6         10                .3
i 198       0         6         10                .5

; INS       COM       DUR       AMP       FREC1     MUESTRA    ATAQ     CAIDA    REVERB     ECO
i 137       0         2.1       70        8.09      5          .01      .01      .3         .6
i 137       1         2.1       70        8.09      5          .01      .01      .5         .6
```

Figure 1.65 Fichero de la partitura de nuestro instrumento "**comb**/**nreverb loscil**" global. El instrumento **nreverb**, *instr 199* es activado al principio de la partitura y permanece activo durante todo el pasaje. Tres copias de nuestro instrumento global **comb,** *instr 198*, empiezan simultáneamente con diferentes duraciones de bucle. Por último, las dos copias de nuestro instrumento **loscil**, *instr 137*, empiezan una tras la otra.

Nuestro próximo instrumento, *instr 138*, mostrado en las figuras 1.66 y 1.67, está basado en un diseño FM previo, pero ahora el instrumento ha sido mejorado con la capacidad de panoramizar la señal.

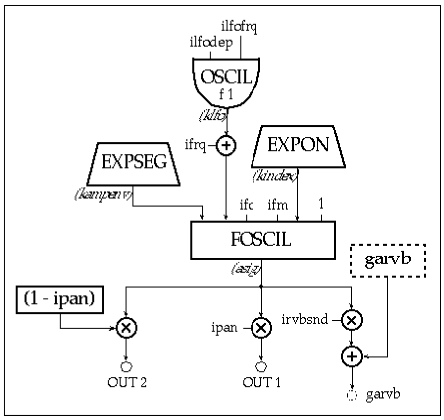


Figura 1.66 Diagrama de flujo de *instr 138*, un instrumento FM dinámico con panoramización y reverberación global.

```
            instr       138             ; BARRIDO FM CON VIBRATO Y PANORAMIZACION DISCRETA
idur        =           p3
iamp        =           ampdb(p4)
ifrq        =           cpspch(p5)
ifc         =           p6
ifm         =           p7
iatk        =           p8
irel        =           p9
indx1       =           p10
indx2       =           p11
indxtim     =           p12
ilfodep     =           p13
ilfofrq     =           p14
ipan        =           p15
irvbsnd     =           p16
kampenv     expseg      .01, iatk, iamp, idur/9, iamp*.6, idur (iatk+irel+idur/9), iamp*.7,
                        irel, .01
klfo        oscil       ilfodep, ilfofrq, 1
kindex      expon       indx1, indxtim, indx2
asig        foscil      kampenv, ifrq+klfo, ifc, ifm, kindex, 1
            outs        asig*ipan, asig*(1-ipan)
garvb       =           garvb+(asig*irvbsnd)
            endin
```

Figura 1.67 Código de la orquesta para *instr 138*, un instrumento FM con vibrato, panoramización discreta y reverberación global.

Deberías observar que en *instr 138* la panoramización se realiza usando una única variable que funciona como la rueda de panoramización en una mesa de mezclas tradicional. ¿Cómo se hace esto?

Bueno, como sabes por el momento, si multiplicamos una señal por un escalar en el rango de 0 a 1, lo que hacemos es controlar eficazmente la amplitud de la señal entre el 0 y el 100%. Así que si multiplicamos la señal y su inversa simultáneamente por el escalar tendríamos dos salidas cuyas amplitudes estarían escaladas entre 0 y 100% pero inversamente proporcionales la una de la otra.

Por ejemplo, si el escalar es 1 y eso corresponde a 1 vez la salida izquierda, tendríamos el 100% de nuestra señal en la izquierda y (1 - 1), o, lo que es lo mismo, el 0% de la señal en la derecha. Si, por otra parte, el escalar de amplitud es .2, entonces quedaría .2 veces, o el 20 % de la señal viniendo de la izquierda y (1 - .2 = .8) y el 80% de la señal viniendo de la derecha. Este algoritmo proporciona un medio simple de usar un único valor para controlar la fuerza a izquierda y derecha de una señal y se usa en *instr 138*, ilustrado en las figuras 1.66 y 1.67.

El fichero *etude6.orc* contiene 4 instrumentos "globo-espaciales" adicionales. Todos están basados en aquellos presentados en estudios previos. Deberías reconocer los algoritmos. Pero todos ellos han sido mejorados con capacidad de panoramización y reverberación global. Te animo a que los estudies y hagas los diagramas de flujo. Cada uno demuestra una aproximación distinta a la panoramización y la reverberación. También te animo a que vuelvas atrás y añadas reverberación global y panoramización a todos los instrumentos que hemos estudiado hasta ahora.

Para finalizar el capítulo I presentaremos *instr 141*, mostrado en las figuras 1.68 y 1.69, que adapta un diseño anterior de modulación de amplitud y añade tanto reverberación global como panoramización basada en un oscilador de baja frecuencia (LFO).

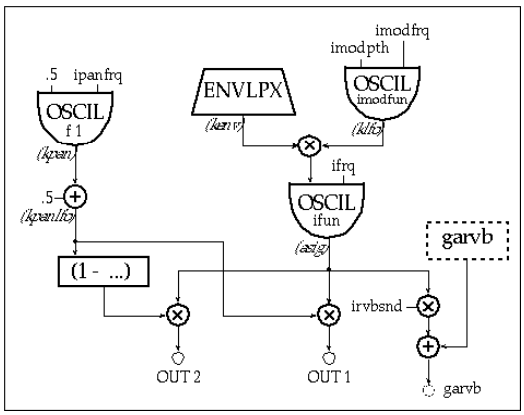


Figura 1.68 Diagrama de flujo de *instr 141*, un instrumento de Modulación de Amplitud con un panoramizador LFO y reverberación global.

```
            instr       141           ; AM CON PANORAMIZADOR LFO
idur        =           p3
iamp        =           ampdb(p4)
ifrq        =           cpspch(p5)
ifun        =           p6
iatk        =           p7
irel        =           p8
iatkfun     =           p9
imodpth     =           p10
imodfrq     =           p11
imodfun     =           p12
ipanfrq     =           p13
irvbsnd     =           p14
kenv        envlpx      iamp, iatk, idur, irel, iatkfun, .7, .01
kpan        oscil       .5, ipanfrq, 1
klfo        oscil       imodpth, imodfrq, imodfun
asig        oscil       klfo*kenv, ifrq, ifun
kpanlfo     =           kpan+.5
            outs        asig*kpanlfo, asig*(1-kpanlfo)
garvb       =           garvb+(asig*irvbsnd)
            endin
```

Figura 1.69 Código de la orquesta para *instr 141*, un instrumento de Modulación de Amplitud con un panoramizador LFO y reverberación global.

Observa aquí que la amplitud del LFO panoramizador es puesta a 5. Esto significa que esta onda sinusoidal bipolar tiene un rango de -5 a +5. Observa, entonces, que lo que hacemos realmente es "inclinar" esta señal bipolar añadiéndole .5 (*kpanlfo = kpan + .5*). Esto convierte la señal en unipolar. Ahora la onda sinusoidal está en el rango

de 0 a 1, con su punto central en .5. Y ¿no es eso perfecto para nuestra rueda de panoramización, que necesita estar en el rango de 0 a 1? Ahora tenemos un sistema de panoramización basado en un LFO.

Ejercicios para el Estudio 6

• Escribe una serie de estudios "musicales" cortos usando tus instrumentos modificados de este capítulo y mándamelos por e-mail.

Conclusión

En este capítulo introductorio he intentado presentar la sintaxis del lenguaje de Csound mientras cubríamos algunos elementos de diseño de sonido. Proporcionada ya esta compresión básica, los siguientes capítulos del libro, escritos por profesores, ingenieros de sonido, programadores y compositores, destacados en todo el mundo, deberían servir para desvelar el poder oculto de Csound y ayudarte a encontrar los tesoros enterrados ahí. Espero sinceramente que, a lo largo del recorrido, no sólo descubras algunos nuevos y exquisitos sonidos, sino que tu trabajo con Csound te proporcione un más detallado conocimiento del hardware de tu actual sintetizador, así como una apreciación y una conciencia más profundas de la naturaleza y el espíritu del sonido musical en sí mismo...
Escucha.

Referencias

Cage, J. 1976. *Silence.* Middletown, CT: Wesleyan University Press.

Chadabe, J. 1997. *Electric Sound: The Past and Promise of Electronic Music*. New York: Prentice Hall.

De Poli, G., A. Piccialli, and C. Roads. 1991. *Representations of Musical Signals*. Cambridge, MA: M.I.T. Press.

De Poli, G., A. Piccialli, S. T. Pope, Stephen and C. Roads. 1997. *Musical Signal Processing*. The Netherlands: Swets and Zeitlinger.

Dodge, C. and T. Jerse. 1997. *Computer Music*. 2nd rev. New York: Schirmer Books.

Eliot, T.S. 1971. *Four Quartets*. New York: Harcourt Brace & Company.

Mathews, Max V. 1969. *The Technology of Computer Music*. Cambridge, MA: M.I.T. Press.

Mathews, Max V. and J. R. Pierce. 1989. *Current Directions in Computer Music Research*. Cambridge, MA: M.I.T. Press.

Moore, R. F. 1990. *Elements of Computer Music*. New York: Prentice Hall.

Pierce, J. R. 1992. *The Science of Musical Sound*. 2nd rev. edn. New York: W. H. Freeman.

Pohlmann, Ken C. 1995. *Principles of Digital Audio*. 3d edn. New York: McGraw-Hill.

Roads, C. 1989.*The Music Machine.* Cambridge, MA: M.I.T. Press.

Roads, C. 1996. *The Computer Music Tutorial*. Cambridge, MA: M.I.T. Press.

Roads, C. and J. Strawn. 1987. *Foundations of Computer Music*. 3d edn. Cambridge, MA: M.I.T. Press.

Selfridge–Field, E. 1997.*Beyond MIDI*. Cambridge: M.I.T. Press.

Steiglitz, K. 1996. *A Digital Signal Processing Primer*. Reading, MA: Addison-Wesley.