Capítulo 4

Automatización y admnistración de las voces

A veces queremos controlar los objetos musicales o los eventos como agregados, en lugar de hacerlo individualmente. Los agregados podrían tomar la forma de una serie de eventos espaciados en el tiempo, en los cuales los detalles de los eventos siguen un arco más grande (por ejemplo las notas de una melodía). O los individuales podrían sonar simultáneamente como las voces en un acorde o los parciales en un sonido complejo. A veces algunas o todas las propiedades de los elementos individuales son mejor inferidas de las propiedades que tiene el todo.

Una rica colección de herramientas e ideas surgen en el repertorio de la música electrónica para describir comportamientos individuales de los agregados. En este capítulo cubrimos dos clases de tales herramientas: los generadores de envolvente y los bancos de sonidos. Los generadores de envolvente automatizan el comportamiento en el tiempo, y los bancos de voces lo hacen sobre los agregados de procesos simultáneos (tales como los generadores de señal).

4.1 Generadores de envolvente

Un *generador de envolvente* (a veces y con más justicia, llamado un *generador de transiente*) hace que una señal de audio se eleve suavemente y luego caiga para controlar la fortaleza de una nota musical. Los generadores de envolvente fueron tocados anteriormente en la sección 1.5. El control de la amplitud por multiplicación (figura 1.4) es la forma más directa y ordinaria para utilizarlo, pero hay muchos otros usos posibles.

Los generadores de envolvente han tenido muchas formas a lo largo de los años, pero el más simple y perenne favorito es el generador de envolvente *ADSR*. “ADSR” es un acrónimo de “Attack, Decay, Sustain, Release” <Ataque, Decaimiento, Sostenimiento y Liberación>, los cuatro segmentos de la salida del generador. El generador ADSR es encendido y apagado por una secuencia de control llamada “disparo”. Al “encender” con el disparador el generador ADSR este ejecuta sus segmentos de ataque, decaimiento y sostenimiento. Al “apagar” con el disparador inicia el segmento de liberación. La figura 4.1 muestra la representación en diagrama de bloques de un generador de envolventes.

Hay cinco parámetros controlando el generador ADSR. El primero, un parámetro de *nivel* ajusta el valor de la salida al final del segmento de ataque (que es normalmente el valor más alto de salida del generador ADSR). El segundo y el tercero son los parámtros de *ataque* y *decaimiento* que dan el tiempo de duración de estos segmentos respectivamente. El cuarto es un parámetro de *sostenimiento* que da el nivel del segmento sostenimiento, como una fracción del parámetro de nivel. Finalmente el parámetro *liberación* proporciona la duración de dicho segmento. Estos cinco valores, junto con los tiempos de encendido y apagado en el disparo, determinan totalmente la salida del generador ADSR. Por ejemplo la duración de la porción del sostenimiento es igual al tiempo entre los “disparos” de encendido y apagado, menos las duraciones de los segmentos de ataque y decaimiento.

La aplicación clásica de una envolvente ADSR es usándola en un teclado o secuenciador musical controlado por voltaje, para hacer notas musicales con un sintetizador. Al oprimir y liberar una tecla (por ejemplo) se deberían generar los disparos de “encendido” y “apagado”. El generador ADSR debería controlar entonces la amplitud de la síntesis de tal manera que las “notas” inicien y terminen con las teclas. Adicional a la amplitud, el generador ADSR puede ser utilizado (y usualmente se hace) para controlar el timbre, lo cual puede hacer evolucionar el timbre de manera natural en el transcurrir de cada nota.

4.2 Formas de amplitud lineales y curvas

Suponga que desea iniciar gradualmente una señal en el transcurso de diez segundos -eso es, usted quiere multiplicarla por una señal de control de amplitud $y[n]$ la cual se eleva de 0 a 1 en valor sobre $10R$ muestras donde R es la velocidad de las muestras. La opción más obvia podría ser la de una rampa lineal: $y[n] = n/(10R)$. Pero esta no dará como resultado la producción de un incremento suave en la intensidad percibida. En el primer segundo $y[n]$ se eleva de -∞ dB a -20 dB, en los siguientes cuatro a otros 14 dB y en los cinco restantes, únicamente un restante de 6 dB. En la mayor parte del período de diez segundos el incremento en la amplitud será barely no-sé perceptible.

Otra posibilidad sería elevar $y[n]$ exponencialmente, de tal manera que se eleve a una velocidad constante en decibeles. Se deberá fijar la amplitud inicial en lo inaudible, por decir 0 dB (si fijamos la unidad a 100 dB). Ahora tenemos el problema opuesto: para los primeros cinco segundos el control de la amplitud se elevará de 0 dB (inaudible) a 50 dB (pianísimo); esta parte del inicio de la nota debería tomarse únicamente el primer segundo aproximadamente.

Una progresión más natural debería ser sin embargo considerar el inicio de la nota como una sucesión de dinámicas, 0-ppp-pp-p-mp-mf-f-ff-fff, tomando cada paso aproximadamente un segundo.

Una entrada de nota idealmente debería obedecer a una escala entre la logarítmica y la lineal. Una opción algo arbitraria, pero útil en la práctica, es la curva cuártica:

$$y[n] = (n/N)^4$$

donde N es el número de muestras sobre el cual está la entrada (en el ejemplo anterior este número es $10R$). Así en el primer segundo de los diez estaríamos elevándonos a -80 dB, a los cinco segundos a -24, y a los nueve, a casi -4 dB.

La figura 4.3 muestra las tres funciones de transferencia de amplitud:

$$f_1(x) = x \text{ (lineal)}$$

$$f_2(x) = 10^{2(x\ -1)} \text{ (dB a lineal)}$$

$$f_3(x) = x^4 \text{ (cuártica)}$$

La segunda función hace la conversión de dB a lineal, arreglada de tal manera que el rango de entrada, de 0 a 1, corresponde a 40 dB. (Este rango de entrada de 40 dB corresponde a un rango de dinámica razonable, permitiendo 5 dB por cada 8 pasos en la dinámica.) La curva cuártica imita bien la curva exponencial (dB) para las amplitudes más altas, pero cae más rápidamente en las amplitudes pequeñas, llegando al cero real correctamente (toda vez que la curva exponencial cae únicamente a 1/100).

Podemos pensar las tres curvas de las funciones de transferencia que se muestran, desde un control abstracto (con rango de 0 a 1) a una amplitud lineal. Después de escoger una función de transferencia f apropiada, podemos calcular la señal de control correspondiente; si queremos elevarnos sobre una rampa de N muestras desde el silencio hasta la unidad de ganancia, la señal de control deberá ser:

$$y[n] = f(n/N)$$

Un diagrama de bloques para esto se muestra en la figura 4.4. Introducimos aquí un nuevo tipo de bloque para representar la aplicación de *función de transferencia*. Por ahora no nos preocuparemos de su implementación; dependiendo de la función deseada, esto debería ser mejor hacerlo aritméticamente o

utilizando la lectura de tablas.

4.3 Cambios de control continuos y discontinuos

Los algoritmos de síntesis varían ampliamente en su capacidad para tratar con los cambios de control discontinuos. Hasta ahora en este capítulo hemos asumido que los controles deben cambiar continuamente, y que el generador de envolvente ADSR se convierte en el que idealmente realiza tales controles. Puede incluso suceder que la amplitud máxima de una nota sea menor que el valor corriente de su predecesora (utilizando el mismo generador) y la envolvente ADSR simplemente descenderá la rampa (en lugar de ascender) a un nuevo valor para un ataque.

Esto no es necesariamente deseable, siempre y cuando, en situaciones donde un generador de envolvente esté encargado de algunos aspectos del timbre: sin embargo, por ejemplo no queremos que la agudeza de una nota decrezca durante el ataque a una más suave, si no que salte a un valor más bajo de tal forma que siempre pueda elevarse durante el ataque.

Esta situación también puede suceder con las envolventes de afinación: se puede querer deslizar la altura de una nota a la siguiente, o se puede querer que la trayectoria de la afinación de cada nota inicie de nuevo en un punto independiente del sonido previo.

Dos situaciones se presentan cuando queremos hacer cambios discontinuos a los parámetros de síntesis: podemos hacerlo simplemente sin interrupción (por ejemplo haciendo un cambio discontinuo en la altura); o en cambio, no podemos, como en el caso de un índice de tabla de onda (el cual hace un cambio discontinuo en la salida). Hay incluso parámetros que *posiblemente* no pueden cambiarse de manera continua; por ejemplo, una selección entre un conjunto de tablas de onda. En general, los cambios discontinuos de la fase en un oscilador o de la amplitud de una señal ocasionará un artefacto audible, pero los incrementos de fase (tales como alturas) pueden ir por saltos sin malos resultados.

En los casos en los que un cambio de parámetro no se puede hacer de manera continua por una u otra razón, existen por lo menos dos estrategias para realizar el cambio limpiamente: el *muteo* y el *cambio-y-rampa*.

4.3.1 Muteo

La técnica del *muteo* se aplica a una envolvente en una salida de amplitud, la cual es rápidamente llevada en rampa a cero antes de que el parámetro cambie y luego es restaurada más tarde. Puede ser o no el caso de que los cambios discontinuos resultarán en una señal que se eleve lentamente desde cero posteriormente. En la figura 4.5 (parte a), tomamos el ejemplo de una envolvente de amplitud (la salida de la señal de un generador de ADSR), y asumimos que el cambio discontinuo está al iniciar una nueva nota desde la amplitud cero.

Para cambiar la salida del generador ADSR de manera dicontinua, lo *reajustamos*. Esta es una operación diferente del disparo; el resultado es hacer que salte a un nuevo valor, después del cual se puede simplemente dejarlo ahí o redisparar el generador ADSR.

Debajo de la salida del generador ADSR vemos el muteo apropiado de la señal, el cual llega en rampa a cero para preparar la discontinuidad. La cantidad de tiempo que dejamos para el muteo debe ser pequeña (de tal manera que interrumpa el sonido previo en tan poco como sea posible) pero no tan pequeña que cause artefactos audibles en la salida. Un ejemplo de trabajo de este tipo de muteo se mostró ya en la página “81”; allí dejamos 5 milisegundos para el descenso de la rampa. La señal del muteo se multiplica por la salida del proceso para eliminar el click.

La figura 4.5 (parte b) muestra la situación en la cual suponemos que el cambio discontinuo está entre dos valores que posiblemente no son cero. Aquí la señal de muteo debe no únicamente bajar la rampa como antes (con anticipación a la discontinuidad) si no que también debe subirla posteriormente. El tiempo de la bajada de la rampa no necesita ser igual al de la subida; estos se pueden escoger, como siempre, escuchando el sonido a la salida.

En general, el muteo presenta la dificultad de que usted debe comenzar la operación del muteo con anticipación al cambio de control deseado. En ajustes en tiempo real, esto usualmente requiere que nosotros intencionalmente retrasemos el cambio de control. Esta es otra razón para mantener el tiempo de muteo tan bajo como sea posible. (Es más, es mala idea tratar de minimizar el retraso condicionalmente, omitiendo el período de descenso de la rampa cuando este no se requiere; un retraso constante es mucho mejor que uno que varíe, incluso si este es más pequeño en promedio.)

4.3.2 Interruptor-y-rampa

La técnica *interruptor-y-rampa* también busca remover discontinuidades resultado de los cambios de control discontinuos, pero lo hace de manera diferente: sintetizando una discontinuidad opuesta la cual adicionamos para cancelar la original. La figura 4.6 muestra un ejemplo en el cual un sonido percusivo (una sinusoide envuelta) comienza una nota en la mitad de la nota previa. El ataque del sonido no deriva de la envolvente de amplitud, si no de la fase inicial de la sinusoide, como es usualmente apropiado para sonidos percusivos. La gráfica inferior en la figura muestra una señal de audio compensadora con una discontinuidad opuesta, la cual puede ser sumada a la superior para remover la discontinuidad. Las ventajas de esta técnica sobre el muteo son, primero, que aquí no se requiere de retraso entre la decisión de hacer un ataque y el sonido del ataque; y segundo, que cualquier artefacto que aparezca con esta técnica es más probable que se pueda enmascarar con la aparición de un nuevo sonido.

La figura 4.7 muestra cómo la técnica interrupción-y-rampa puede realizarse en un diagrama de bloques. La caja marcada con comillas (“...”) puede contener cualquier algoritmo de síntesis, el cual queremos interrumpir discontinuamente de tal manera que re-inicie de cero (como por ejemplo lo hace en la parte (a) de la figura previa). Al mismo tiempo que disparamos los cambios de control necesarios (ejemplificados por el generador ADSR de la parte superior), también reiniciamos y disparamos otro generador ADSR (en la mitad a la derecha) para cancelar la discontinuidad. La discontinuidad es menos que el último valor de salida de la síntesis, justo antes de reajustarse a cero.

Para hacer esto medimos el nivel al que el generador de envolvente ADSR debe saltar. Este es su propio nivel corriente (que no debería ser cero). Los dos son sumados (con el objeto `+~` en la parte inferior), y luego se toma una instantánea. La generador de envolvente de cancelación (a la derecha) es reiniciado continuamente a este nuevo valor, y luego disparado para descender la rampa a cero. La salida del objeto `+~` (la suma de la salida del sintetizador y de la señal de cancelación de discontinuidad) es la señal de-click-ada.

4.4 Polifonía

En la música, el término polifonía es usualmente utilizado para significar “más de una voz separadas cantando o tocando en diferentes tonos una de la otra”. Cuando hablamos de instrumentos musicales electrónicos utilizamos el término para decir “mantener varias copias del mismo proceso en paralelo”. Usualmente llamamos cada copia una “voz” manteniendo la analogía, aun cuando las voces no necesitan ser tocadas de manera separada con sonidos distintos.

En este lenguaje, un piano es un instrumento polifónico, con 88 “voces”. Cada voz del piano es capaz normalmente de tocar exactamente una nota. Aquí nunca se pregunta cuál voz usar para tocar una nota en una altura determinada, y tampoco

se pregunta por la ejecución de varias notas simultáneamente en la misma altura.

Muchos instrumentos musicales electrónicos polifónicos se aprovechan de manera más flexible del manejo de las voces. La mayoría de los programas de síntesis (como Csound) utilizan un esquema de asignación dinámica de voces, esto es, en efecto, que una nueva voz se crea por cada nota en la partitura. En sistemas tales como Max o Pd, los cuales están orientados para el uso interactivo en tiempo real, un *banco de voces* es localizado con anticipación, y el trabajo que se debe hacer (tocar notas o lo que sea), se distribuye entre las voces en el banco.

Esto se diagrama en la figura 4.8, donde cada una de las voces produce una señal de salida, las cuales se suman todas para producir la salida total de las voces del banco. Frecuentemente las voces individuales requerirán algunas salidas separadas; por ejemplo, ellas podrían tener salidas por varios canales de tal manera de que puedan ser panoramizadas individualmente; o podrían tener envíos de efectos individuales de tal manera que cada una puede tener su propio nivel de envío.

4.5 Asignación de las voces

Es frecuentemente deseable automatizar la selección de las voces para asociarlas con *tareas* individuales (tales como notas para tocar). Por ejemplo, un músico tocando el teclado no puede escoger en la práctica cuál voz suena con cada nota. Para automatizar la selección de las voces necesitamos un algoritmo de asignación de las voces, para usarse como se muestra en la figura 4.9.

Armados con un algoritmo apropiado de asignación de las voces, la fuente de control no necesita preocuparse con el detalle de qué voz se ocupa de qué tarea; a los generadores de notas algorítmicos y a los secuenciadores frecuentemente se les confía esto. De otro lado, la escritura musical para ensambles frecuentemente especifica explícitamente cuál instrumento toca qué nota, de tal manera que las notas se conectarán una con la otra de extremo a extremo de la manera que queremos. Un algoritmo sencillo de asignación de voces trabaja como se muestra en la figura 4.10. Aquí suponemos que el banco de voces tiene únicamente dos voces, y tratamos de asignar voces para las tareas *a*, *b*, *c* y *d*. Las cosas van sin sobresaltos hasta que llega la tarea *d*, porque entonces vemos que no hay voces libres (están tomadas por *b* y *c*). Podríamos ahora elegir realizar o no la tarea *d* o tomar la voz de la tarea *b* o *c*. En la práctica, la mejor opción es tomar una. En este ejemplo en particular, escogimos tomar la voz de la tarea más antigua, *b*.

Si conocemos la longitud de las tareas *b* y *c* por fuera de la ejecución de la tarea *d*, podremos ser capaces de escoger mejor cuál voz tomar. En este ejemplo podría haber sido mejor tomar la de la *c*, ya que *d* y *b* deben sonar juntas en el final y no la *d* únicamente. En algunas situaciones esta información estará disponible cuando debamos tomar la decisión, y en otras (la entrada de un teclado en vivo, por ejemplo), no.

4.6 Etiquetas de las voces

Suponga ahora que estamos utilizando en banco de voces para tocar notas, como en el ejemplo de arriba, pero suponga que las notas *a*, *b*, *c* y *d* tienen todas la misma altura y aún más, que tienen otros parámeros idénticos. Cómo podemos diseñar una secuencia de control de tal manera que cuando cualquier nota se apague, sepamos cuál fue?

Esta cuestión no viene al caso si la fuente de control es un teclado debido a que es imposible tocar más de una nota simultáneamente con una sola tecla. Pero podría fácilmente presentarse algorítmicamente, o simplemente como resultado de la mezcla de dos secuencias de teclados juntas. Aún más, el apagado de las notas es únicamente el ejemplo más simple de un problema más general, el cual es cómo,

una vez terminada una tarea en un banco de voces, podemos recobrarla para la misma voz con el fin de guiar su evolución como una función de las entradas en tiempo real, o de cualquier otro factor que no es posible predecir.

Para tratar con situaciones como esta podemos adicionar una o más etiquetas al mensaje de inicio de la nota (o en general, a una tarea). Una etiqueta es cualquier conjunto de datos con el cual podemos identificar más tarde la tarea, los cuales podemos utilizar depués para buscar la voz a la que fue asignada.

Tomando de nuevo el ejemplo de la figura 4.10, hay una forma en la que podríamos escribir aquellas cuatro tareas como una secuencia de control:

tiempo-de-inicio	tiempo-de-finalización	nota	...
1	3	60	...
2	8	62	...
4	6	64	...
5	8	65	...

En esta representación no necesitamos etiquetas debido a que cada mensaje (cada línea de texto) contiene toda la información que requerimos para especificar la tarea completa. (Aquí hemos asumido que las tareas *a*, ..., *d* son de hecho notas musicales con alturas 60, 62, 64 y 65.) En efecto, estamos representando cada tarea como un evento singular en una secuencia de control (sección 3.3).

De otro lado, si suponemos ahora que no conocemos con anticipación la longitud de cada nota, una mejor representación podría ser esta otra:

tiempo	etiqueta	acción	parámetros
1	a	start	60 ...
2	b	start	62 ...
3	a	end	
4	c	start	64 ...
5	d	start	65 ...
6	c	end	
7	b	end	
8	d	end	

Aquí cada nota ha sido dividida en dos eventos separados para iniciar y para finalizar. Las letras *a*, ..., *d* son utilizadas como etiquetas; sabemos cuál inicio va con con cuál final ya que sus etiquetas son las mismas. Note que la etiqueta no necesariamente está del todo relacionada con la voz que se utilizará para tocar cada nota.

El MIDI normalizado no suministra etiquetas; en el uso normal, la altura de una nota sirve también como su etiqueta (de tal manera que las etiquetas están siendo re-utilizadas continuamente.) Si dos notas tienen la misma altura deben ser dirigidas de manera separada (para deslizar sus alturas en direcciones diferentes, por ejemplo) y el canal MIDI debe ser usado como una etiqueta (adicional a la nota).

En los programas de computación de música en tiempo real es usualmente necesario pasar alternativamente entre los eventos-por-tarea y el etiquetado anterior, ya que la primera representación está mejor dispuesta para almacenar una edición gráfica, mientras que la segunda está mejor dispuesta para las operaciones en tiempo real.

4.7 Encapsulado Pd

Los ejemplos para este capítulo utilizarán el mecanismo de *encapsulado* Pd el cual permite la construcción de parches que pueden ser reutilizados cualquier

número de veces. Una o más caja de objetos en Pd pueden ser *subparches*, los cuales son parches separados encapsulados dentro de las cajas. Vienen en dos tipos: *subparches encendido-apagado* y *abstracciones*. En cualquier caso el subparche aparece como una caja de objeto en otro parche denominado *padre*.

Si usted escribe “pd” o “pd mi-nombre” dentro de una caja de objeto, este crea un subparche encendido-apagado. Los contenidos de este subparche son salvados como parte del parche padre, en un archivo. Si hace algunas copias de un subparche, puede cambiarlas individualmente. De otro lado puede invocar una abstracción escribiendo dentro de la caja el nombre de un parche Pd salvado como un archivo (sin la extensión “.pd”). En esta situación Pd leerá ese archivo dentro del subparche. De esta manera los cambios en el archivo se propagarán donde quiera que la abstracción se invoque.

Un subparche (sea encendido-apagado o abstracción) puede tener diferentes entradas o salidas que aparecen en la caja del parche padre. Esto se especifica utilizando los siguientes objetos:

`inlet`, `inlet~`: crea entradas para la caja de objeto que contiene al subparche. La versión `inlet~` crea una entrada de señal de audio, mientras que `inlet` crea crea una para secuencias de control. En cualquier caso, lo que sea que llegue a la entrada de la caja en el parche padre, llega al `inlet` o al `inlet~` en el subparche.

`outlet`, `outlet~`: objetos correspondiente para las salidas de los subparches.

Pd provee un mecanismo de paso-de-argumento para que usted pueda parametrizar invocaciones diferentes de una abstracción. Si en una caja de objeto usted escribe “$1”, esto significa “el primer argumento creado en mi caja del parche padre”, de manera similar para “$2” y así sucesivamente. El texto en una caja de objeto se interpreta en el momento en que se crea la caja, a diferencia del texto en una caja de mensaje. En las cajas de mensajes el mismo “$1” significa “el primer argumento del mensaje que acabo de recibir” y es interpretado siempre y cuando llegue un nuevo mensaje.

Un ejemplo de una abstracción utilizando entradas, salidas y parametrización, se muestra en la figura 4.11. En la parte (a), un parche invoca el objeto `plusminus`, y la salida es la suma y la diferencia de 8 y 5.

El objeto `plusminus` no está definido por Pd, pero sí está definido por el parche que reside en el archivo denominado “plusminus.pd”. Este parche se muestra en la parte (b) de la figura. El objeto `inlet` y los dos objetos `outlet` corresponden a las entradas y las salidas del objeto plusminus. Los dos argumentos “$1” (para los objetos `+` y `-`) son reemplazados por 5 (el argumento creado del objeto `plusminus`).

Hemos visto ya una abstracción en los ejemplos: el objeto `output~` introducido en la figura 1.10 (página “16”). En ese ejemplo se ve también que una abstracción puede mostrar controles como parte de su caja en el parche padre; vea la documentación de Pd para una descripción de esta característica.

4.8 Ejemplos

Generador de envolvente ADSR

El ejemplo D01.envelope.gen.pd (figura 4.12) muestra cómo el objeto `line~` puede ser utilizado para generar una envolvente ADSR para controlar un parche de síntesis (en la figura se muestra únicamente la envolvente ADSR). El botón “attack” tiene dos efectos cuando se presiona. El primero (a la izquierda en la figura) es ajustar el objeto `line~` con su segmento de ataque, con una meta de 10 (la amplitud pico) sobre 200 milisegundos (el tiempo de ataque). Segundo, el botón de ataque ajusta un objeto `delay 200`, de tal manera que después de que el

segmento de ataque haya pasado, el segmento de decaimiento pueda iniciar. El segmento de decaimiento cae a 1 (el nivel de sostenimiento) luego de otros 2500 milisegundos (el tiempo de decaimiento).

El botón “release” envía el mismo objeto `line~` a cero en más de 500 milisegundos (el tiempo de liberación). También, en caso de que el objeto `delay 200` esté activo en el momento de que el botón “release” se presiona, un mensaje “stop” es enviado a este. Esto previene al generador ADSR de lanzar su segmento de decaimiento después de lanzar el segmento de liberación.

En el ejemplo D02.adsr.pd (figura 4.13) encapsulamos el generador ADSR en una abstracción Pd (denominada `adsr`) de tal manera que se pueda replicar con facilidad. El diseño de la abstracción `adsr` hace posible controlar los cinco parámetros ya sea suministrando argumentos creados o conectando secuencias de control en sus entradas.

En este ejemplo, los cinco argumentos creados (1, 100, 200, 50 y 300) especifican el nivel pico, el tiempo de ataque, el tiempo de decaimiento, el nivel del sostenimiento (como un porcentaje del nivel pico), y el tiempo de liberación. Hay seis entradas de control: la primera para disparar el generador ADSR, y las otras para actualizar los valores de los cinco parámetros. La salida de la abstracción es una señal de audio.

Esta abstracción está realizada como se muestra en la figura 4.14. (Puede abrir este subparche haciendo click en el objeto `adsr` del parche.) Los únicos objetos de señal son `line~` y `outlet~`. Los tres objetos `pack` corresponden a los tres objetos de mensaje de la figura 4.12 anterior. De izquierda a derecha, ellos se ocupan de los segmentos de ataque, decaimiento y liberación.

El segmento de ataque va a un objetivo especificado como “$1” (el primer argumento creado de la abstracción) en “$2” milisegundos; estos valores se pueden sobreescribir enviando números a las entradas “peak level” y “attack”. El segmento de liberación es similar pero más sencillo, ya que el objetivo es siempre cero. La parte difícil es el segmento de decaimiento, que debe ajustarse después de un retraso igual al tiempo de ataque (el objeto `del $2`). El nivel de sostenimiento es calculado a partir del nivel pico y del porcentaje de sostenimiento (multiplicando los dos y dividiendo por 100).

La entrada del disparador, si envía un número diferente de cero, dispara un arranque (los segmentos de ataque y decaimiento), y si envía cero, dispara el segmento de liberación. Aún más, el generador ADSR puede ser reiniciado en cero enviando un disparo negativo (lo cual genera también un arranque).

Funciones de transferencia para control de amplitud

La sección 4.2 describió el uso de las envolventes ADSR para controlar la amplitud, para lo cual los segmentos exponencial o de curva cuártica usualmente proporcionan resultados sonoros más naturales que los segmentos lineales. Los parches D03.envelope.dB.pd y D04.envelope.quartic.pd (el último mostrado en la figura 4.15) demuestra el uso de segmentos decibel y cuártico. Adicional a la amplitud, en el ejemplo D04.envelope.quartic.pd la frecuencia del sonido está controlada también, utilizando formas lineales y cuárticas para su comparación.

Ya que convertir decibeles a unidades de amplitud lineal es una operación costosa (por lo menos comparada con la de un oscilador o la de un generador de rampa), el ejemplo D03.envelope.dB.pd utiliza lectura de tabla para implementar la función de transferencia necesaria. Esto tiene la ventaja de la eficiencia, pero la desventaja de que debemos decidir el rango de los valores admisibles con anticipación (que son aquí de 0 a 120 dB).

Para los segmentos cuárticos como en el ejemplo D04.envelope.quartic.pd no se requiere lectura; simplemente elevamos al cuadrado la señal de salida del objeto

line~, dos veces, sucesivamente. Para compensar la elevación a la cuarta potencia, el valor meta enviado al objeto line~ debe ser la raíz cuarta del valor deseado. Así, los mensajes que hacen la rampa de frecuencia o amplitud son primero desempacados para separar el valor meta del intervalo de tiempo, se toma la raíz cuarta del valor meta (por la vía de la sucesión de dos raíces cudradas), ylos dos se envían luego al objeto line~. Hemos hecho uso aquí de un nuevo objeto Pd:

unpack: desempaca una lista de números (y/o símbolos) y los distribuye en salidas separadas. Como es usual las salidas aparecen en orden de derecha a izquierda. El número de las salidas y sus tipos están determinadas por los argumentos creados. (Ver también pack, página “50”).

Los dos parches siguientes, D05.envelope.pitch.pd y D06.envelope.portamento.pd, utilizan el primero, un generador de envolvente ADSR para hacer una envolvente de altura y el segundo, un solo objeto line~, también como control de la altura, para hacer portamento. En ambos casos los segmentos exponenciales son deseables, y se calculan utilizando lectura de tabla.

Síntesis aditiva: la campana de Risset

El mecanismo de abstracción de Pd, que utilizamos antes para hacer un generador de envolvente re-utilizable, es también útil para hacer bancos de voces. Utilizaremos aquí abstracciones para organizar bancos de osciladores para síntesis aditiva. Hay muchas formas posibles de organizar bancos de osciladores, además de las que se muestran aquí.

La organización más simple y directa de las sinusoides es formar parciales y adicionarlos a una nota. El resultado es monofónico, en el sentido que el parche ejecutará una sola nota a la vez, la cual, sin embargo consistirá de algunas sinusoides cuyas frecuencias y amplitudes individuales dependen de aquellas cuya nota estamos tocando, y también de su lugar individual en una serie armónica (o inarmónica) de series de sobretonos.

El ejemplo D07.aditive.pd (figura 4.16) utiliza un banco de 11 copias de una abstracción denominada partial (figura 4.17) en una imitación de un instrumento campana bien conocido de Jean-Claude Risset. Tal como se describe en [DJ85, pág. 94] el sonido de la campana tiene 11 parciales, cada uno con su propia amplitud, frecuencia y duración relativas.

Para cada nota, la abstracción partial computa una envolvente de amplitud simple (cuártica) consistente únicamente de un segmento de ataque y de decaimiento; sin segmento de sostenimiento o de liberación. Esto se multiplica por una sinusoide y el producto se adiciona en un bus de suma. Dos nuevos tipos de objetos son introducidos para implementar el bus de suma:

catch~: define y da salida a un bus de suma. El argumento creado (“sum-bus” en el ejemplo) le da al bus de suma un nombre para que el objeto throw~ en la parte inferior se pueda referir a este. Usted puede tener tantos buses de suma como quiera (y por lo tanto objetos catch~) pero deben tener todos nombres diferentes.

throw~: adición a un bus de suma. El argumento creado selecciona cuál bus de suma utilizar.

La interface de control es cruda: las cajas de números controlan la frecuencia “fundamental” de la campana y su duración. Al enviar un mensaje “bang” al objeto s trigger comienza la nota. (la nota entonces decae sobre el período de tiempo controlado por el parámetro de duración; no hay disparo separado para detener la nota). No hay control de amplitud excepto vía el objeto output~.

Los cuatro argumentos para cada invocación de la abstracción partial

especifican:

1. la amplitud. Es la amplitud del parcial en su pico, al final del ataque y al comienzo del decaimiento de la nota.

2. la duración relativa. Esta está multiplicada por la duración general de la nota (controlada en el parche principal) para determinar la duración de la porción de decaimiento de la sinusoide. Los parciales individuales pueden tener así diferentes tiempos de decaimiento de tal forma que algunos parciales mueren más rápido que otros, bajo el control general del parche principal.

3. la frecuencia relativa. Al igual que con la duración relativa, esta controla cada frecuencia de los parciales como un múltiplo de la frecuencia general controlada en el parche principal.

4. la desafinación. Una frecuencia en Hertz que se añade al producto de la frecuencia global y la frecuencia relativa.

A interior de la abstracción `partial`, la amplitud es tomada simplemente de manera directa del argumento "$1" (multiplicando por 0.1 para ajustar las amplitudes individuales altas); la duración es calculada del objeto `r duration`, multiplicándolo por el argumento "$2". La frecuencia es computada como $fp + d$ donde f es la frecuencia global (que viene del objeto `r frequency`), p es la frecuencia relativa del parcial, y d es la frecuencia de desafinación.

Síntesis aditiva: control de la envolvente espectral

El siguiente parche de ejemplo, D08.table.spectrum.pd (figura 4.18), muestra una aplicación muy diferente de la síntesis aditiva. Aquí las sinusoides son manejadas por la abstracción `spectrum-partial` mostrada en la figura 4.19. Cada parcial computa su propia amplitud periódicamente (cuando es disparada por el objeto `poll-table`), utilizando un objeto `tabread4`. Los contenidos de la tabla (los cuales tienen un rango nominal de 50 dB) son convertidos a unidades lineales y utilizados como un control de amplitud en la forma usual.

<En el banco de osciladores de D08.table.spectrum.pd, se diferencia uno de otro por un solo argumento que puede hacer las veces de número de parcial (spectrum-partial 1, spectrum-partial 2, etc). Ese número se multiplica por la frecuencia, dando como resultado un múltiplo entero de esta última. Esa frecuencia-múltiplo se convierte a unidades MIDI a las que se resta el número whammybar (que está en las mismas unidades), lo que actúa como un filtro. Sin embargo, la línea de amplitudes dibujada en spectrum-tab es la que actúa en primer lugar como filtro inicial, aumentando o disminuyendo amplitudes en determinadas frecuencias.>

Un ejemplo similar, D09.shepard.tone.pd (que no se muestra), fabrica un sonido Shepard usando la misma técnica. Aquí las frecuencias de las sinusoides barren en un rango fijo, saltando desde el final al comienzo y repitiendo. La envolvente espectral está ordenada para tener un pico en la mitad del rango de afinación y caer hasta ser inaudible en los extremos del rango de tal manera que escuchamos un barrido continuo pero no el salto. El resultado es un famoso acertijo auditivo, un sonido indefinidamente ascendente o descendente.

<En D09.shepard.tone.pd la fase desciende de 10000 a 0 para volver a iniciar, de dos en dos (ver el -2), cada 50 milisegundos. Ese número se recibe dentro de cada subparche y se le suma la fase relativa, que es el único argumento creado para cada subparche. El rango resultante de las fases se normaliza entre -1 y 1 mediante la operación *fase-total**0.0002-1. Este número se utiliza para computar el descenso de la frecuencia en cada subparche que en su punto máximo es 120+60 ("interval" menos "pitch") y en su mínimo -120+60 (120 es un intervalo elegido). Pero también ayuda a computar el descenso en

la amplitud con la fórmula de tipo gaussiano 1/(e^(*fase-total*fase-total*10)). Aquí 10 es un módulo elegido para obtener una forma determinada para la caída de amplitud. Este uso doble y simultáneo de lo que llamamos *fase-total* permite hacer coincidir la llegada a la nota central (60 en este caso) con la máxima amplitud sonora 1.>

La técnica de sintetizar a una envolvente espectral específica puede ser generalizada de muchas maneras: la envolvente puede ser hecha para variar en el tiempo ya sea como el resultado de un análisis de una señal en vivo, o por el cálculo de un conjunto de reglas composicionales, o por el cruzamiento atenuado entre un conjunto de envolventes espectrales prediseñadas o por el pandeo-de-frecuencias de envolventes, para nombrar algunas pocas posibilidades.

Síntesis polifónica: el muestreador

Nos movemos ahora a un ejemplo utilizando la asignación dinámica de voces tal como se describió en la sección 4.5. En los ejemplos de la síntesis aditiva mostrados previamente, cada voz es utilizada para un propósito fijo. En el presente ejemplo, asignamos las voces desde un banco según se necesiten para ejecutar las notas en una secuencia de control.

El ejemplo D11.sampler.poly.pd (figura 4.20) muestra el sampleador polifónico, que usa la abstracción sampvoice (cuyo interior se muestra en la figura 4.21). Las técnicas para alterar la afinación y otros parámetros en un muestrador de un disparo se ven en el ejemplo D10.sampler.notes.pd (que no se muestra) el cual a su vez deriva del muestrador de un disparo del capítulo previo (C05.sampler.oneshot.pd, mostrado en la figura 3.14.)

Los objetos sampvoice están organizados en un tipo diferente de bus de suma al de los anteriores; aquí, cada uno adiciona su propia salida a la señal en su entrada, y pone la suma en su salida. En la parte inferior de los ocho objetos, la salida de esta manera mantiene la suma de todos los ocho objetos. Este tiene la ventaja de ser más explícito que los buses throw~ / catch~, y es preferible cuando el atestamiento visual no es un problema.

La tarea principal del parche es, sin embargo, distribuir los mensajes “note” a los objetos sampvoice. Para hacer esto debemos introducir algunos nuevos objetos Pd:

mod: Módulo de enteros. Por ejemplo, 17 mod 10 da 7, -2 mod 10 da 8. Existe también un objeto de división de enteros denominado div; dividir 17 por 10 vía div da 1, y -2 por 10 da -1.

poly: Asignador de voces polifónico. Los argumentos creados dan el número de voces en el banco y una bandera (1 si la voz que se toma se necesita, 0 si no). Las entradas son una etiqueta numérica a la izquierda y una bandera a la derecha indicando el inicio o la parada de una voz con la etiqueta dada (los números que no son cero significan “start” y cero, “stop”). Las salidas son, a la izquierda, el número de la voz, la etiqueta de nuevo al centro, y la bandera inicio/parada a la derecha. En las aplicaciones MIDI, la etiqueta puede ser la nota y la bandera inicio/parada puede ser la velocidad de la nota.

makenote: Suministra mensajes retrasados de nota-apagada para que coincidan con los mensajes nota-encendida. Las entradas son una etiqueta y bandera de inicio/parada (“nota” y “velocidad” cuando se usa MIDI) y la duración deseada en milisegundos. Los pares etiqueta/bandera se repiten para las dos salidas tal como ellas son recibidas; luego, después del retraso, la etiqueta es repetida con la bandera cero para detener la nota después de la duración deseada.

Los mensajes “note” contienen campos para nota, amplitud, duración, número de muestra, sitio de inicio en la muestra, tiempo de aparición y tiempo de decaimiento. Por ejemplo, el mensaje

60 90 1000 2 500 10 20

(si es recibido por el objeto r note) significa tocar una nota a la altura 60 (en unidades MIDI), amplitud 90 dB, un segundo de longitud, desde la tabla de onda denominada “sample2”, comenzando en el punto 500 milisegundos en la tabla de onda, con tiempos de aparición y de decaimiento de 10 y 20 milisegundos.

Luego de desempacar el mensaje en sus siete componentes, el parche crea una etiqueta para la nota. Para hacer esto, primero el objeto t b f da como salida un disparo después de que el último de los siete parámetros aparece separadamente. La combinación de los objetos +, f, y mod actúa como un contador que se repite después de un millón de veces, escencialmente generando un número único correspondiente a la nota.

El paso siguiente es utilizar el objeto poly para determinar cuál voz ejecuta qué nota. El objeto poly espera mensajes separados para iniciar y detener las tareas (es decir, las notas). De esta manera la etiqueta y la duración son alimentados primero al objeto makenote, cuyas salidas incluyen una bandera (“velocidad”) a la derecha y una etiqueta de nuevo a la izquierda. Por cada etiqueta que recibe makenote, se genera la salida de dos pares de números, uno para iniciar la nota, y el otro, después de un retraso igual a la duración de la nota, para detenerla.

Habiendo tratado poly de separar esta entrada, ahora tenemos que desnudar los mensajes correspondientes a los finales de las notas, puesto que nosotros realmente necesitamos únicamente mensajes combinados de “nota” con campos de duración. El objeto stripnote hace esta tarea. Finalmente, el número de voz que hemos calculado es antepuesto a los siete parámetros con los que comenzamos (el objeto pack), de tal manera que la salida del objeto pack aparece así:

4 60 90 1000 2 500 10 20

donde “4” es el número de la voz a la salida del objeto poly. El número de la voz es utilizado para enrutar el mensaje hacia la voz deseada utilizando el objeto route. El objeto sampvoice apropiado obtiene entonces la lista original iniciando con “60”.

> <El objeto poly distribuye las voces de manera inteligente, para que su uso sea óptimo. Es decir, tiene en este ejemplo hasta 8 voces a las cuales les puede asignar tareas en tanto la polifonía le exija echar mano de más de una voz. Si la polifonía exige de más voces, por la manera en que está estructurada la pieza, se genera un recorte de voces>

Dentro del objeto sampvoice (figura 4.21), el mensaje es utilizado para controlar los objetos tabread4~ y rodear los objetos line~ y vline~. El control tiene lugar con un retraso de 5 milisegundos como en el primer ejemplo de muestreador. Aquí, sin embargo debemos almacenar los siete parámetros de la nota (antes aquí no había parámetros). Esto se hace utilizando los seis objetos f, más la entrada derecha del objeto delay de la derecha. Estos valores son usados después del retraso de 5 milisegundos. Esto se hace en tándem con el mecanismo de muteo descrito en la página “95”, utilizando otro objeto vline~.

Cuando han pasado los 5 milisegundos, el objeto vline~ a cargo de la generación del índice de la tabla de onda consigue poner sus órdenes en marcha (y simultáneamente, el número de la tabla de onda es ajustado para el objeto tabread4~ y la envolvente de amplitud del generador comienza su ataque.) El índice de la tabla de onda debe ajutarse discontinuamente para iniciar el índice, luego hace la rampa hacia un índice final en un intervalo de tiempo apropiado para obtener la transposición deseada. El índice de inicio en muestras es precisamente 44.1 veces la localización del inicio en milisegundos, más uno, para permitir la tabla de cuatro puntos de interpolación. Este viene a ser el

tercer número en la lista empacada generada por el objeto pack en el centro del parche de voces.

Arbitrariamente decidimos que la rampa tendrá diez mil segundos (este es el "1e+07" que aparece en la caja de mensaje enviada al índice de la tabla de onda del generador), esperando que sea esta la mayor duración de cualquier nota que vayamos a tocar. El índice del final es el índice del inicio, más el número de muestras de la rampa. Con un factor de transposición de uno, nos deberemos mover 441,000,000 de muestras durante 10,000,000 de milisegundos, o proporcionalmente más o menos, dependiendo del factor de transposición. El factor de transposición es computado por el objeto mtof, dividido por 261.62 (la frecuencia correspondiente a la nota MIDI 60) de tal manera que una "altura" especificada como 60 resulte en un factor de transposición de uno.

Estos y otros parámetros son combinados en un mensaje vía el objeto pack de tal manera que las cajas de mensajes siguientes puedan generar los mensajes de control requeridos. La única novedad es el objeto makefilename, el cual convierte números tales como "2" a símbolos tales como "sample2" de tal manera que el objeto tabread4~ de la tabla de onda se puede ajustar.

En la parte inferior del parche de voces veremos cómo un bus de suma se implementa dentro de un subparche; un objeto inlet~ va haciendo la suma de todas las voces precedentes, haciendo que la salida de la voz corriente se adicione en la entrada, y el resultado se envíe a la siguiente voz, vía el objeto outlet~.

<En sampvoice.pd el objeto pack de la parte inferior empaca las variables 1 amplitud (dB), 2 tiempo de entrada (milisegundos), 3 sitio de inicio (en número de muestras), 4 factor de transposición, 5 número de la muestra o archivo de audio. Así, los mensajes por debajo del objeto pack inferior hacen las funciones de:

1er mensaje: ir a 0 en 5 milisegundos, para el vline~ que en este caso apaga los procesos previos al disparo de la nota;

2o mensaje: ir al valor 3, sitio de inicio e ir al valor 4, factor de transposición para el vline~ que hace la lectura de la tabla

3er mensaje: ubica el número del archivo de audio

4o mensaje: ir a 1 en 5 milisegundos, para el vline~ que en este caso mutea el proceso de la envolvente en las discontinuidades, (esto debería ser siempre y cuando se trate de repetir la misma voz antes de que la envolvente termine su trabajo)

5o mensaje: ir a 0 e ir al valor 1, amplitud (dB) en el tiempo de entrada (milisegundos), valor 2. Estos valores son desempacados para ser computados en curva cuártica con vline~.>

Ejercicios

1. Qué entrada para una función de transferencia de cuarta potencia da una salida de -12 dB, si una entrada de 1 da una salida de 0 dB?

2. Un generador de envolvente se eleva desde cero a un valor pico durante su segmento de ataque. Cuántos decibeles menos que el pico tiene la salida alcanzada en la mitad del camino del ataque, asumiendo una salida lineal? Y asumiendo una salida de cuarta potencia?

3. Qué ley exponencial de función de transferencia (es decir, qué opción de n para la función $f(x) = x^n$) debería utilizar si usted desea que el valor en el punto medio sea de -12 dB?

4. Suponga que desea hacer un cruzamiento atenuado de dos señales, es decir, que mientras una señal va entrando, la otra va decayendo. Si tienen igual potencia y

no están correlacionadas un cruzamiento atenuado lineal daría como resultado una caída de 3 decibeles en la mitad del cruzamiento. Qué ley de potencia debería utilizar para mantener la potencia constante a lo largo del cruzamiento?

5. Un acorde de tres notas, que dura 1.5 segundos, suena iniciando una vez cada segundo. Cuántas voces se requieren para sintetizarlo sin que se anule ninguna nota?

6. Suponga que un parche de síntesis da salida entre -1 y 1. Mientras una nota está sonando, una nueva nota ha iniciado utilizando la técnica de toma-de-voz con “descenso-de-rampa”. Cuál es la salida máxima?