Programación de Música Electrónica en Pd

Johannes Kreidler

<www.kreidler-net.de>

27-01-2009

Historia de revisión		
Revision 0.71e	27-01-2009	JK
Traducción al español: Raúl Lacabanne		

Resumen

Pd fue iniciado por el ingeniero en software estadounidense Miller Puckette, quien previamente co-desarrolló el software, bien conocido y similarmente estructurado, Max/Msp. Pd no es un software comercial; no fue desarrollado por una corporación y no está a la venta. En cambio, es “open source”: su código fuente no es propiedad (patentada) de una corporación, sino que se encuentra disponible libremente para todos. Una desventaja de esto es que hasta el momento no ha existido un manual de operaciones detallado para usuarios que carezcan de experiencia en programación. En oposición a una corporación— que tiene un interés monetario en asegurarse que los usuarios nóveles puedan operar fácilmente un software nuevo— el movimiento open source carece de dicha fuerza conductora para hacerse asimismo accesible. Este libro es un intento de llenar dicho hueco.

Este tutorial está diseñado para estudio personal, principalmente para compositores. Comienza con una explicación de programación básica y principios acústicos y gradualmente avanza hacia las técnicas más avanzadas de procesamiento de música electrónica. La propuesta didáctica del libro se concentra primariamente en la escucha, la cual consideramos es una forma más rápida y amena de absorber nuevos conceptos que a través de fórmulas abstractas.

Los patches que ilustran el texto se encuentran disponibles para su descarga.

Tabla de Contenidos

Prólogo

Este libro es el resultado de mi experiencia enseñando música electrónica. A través del proceso de enseñanza, me he familiarizado con los puntos más problemáticos que los estudiantes encuentran —especialmente cuando la lengua nativa del estudiante no coincide con el lenguaje que conduce las lecciones.

Pd (Pure Data) es un lenguaje de programación profesional y de alta performance para el procesamiento de sonidos electrónicos. Es open source, es decir, de libre disponibilidad en Internet. Una desventaja de esto es que Pd sólo se discute en ciertas instituciones o foros de Internet. La compleja terminología técnica que utilizan en estos sitios es realmente difícil de entender para los principiantes.. Este libro libro ayudará a los usuarios nóveles a despejar estos primeros obstáculos en el aprendizaje de Pd.

El principal diseñador de Pd, Miller Puckette, también a escrito un libro sobre teoría y tecnología sobre el procesamiento de música electrónica con Pd. Seguramente no exista mejor profesor de un lenguaje de programación que la persona que lo diseñó; su acercamiento primario científico ciertamente cubre todo el material en una forma total y sistemática. Sin embargo, su método de enseñanza puede ser difícil de comprender. Mi experiencia pedagógica ha sido la que demanda a los lectores el texto de Puckette, grandes cantidades de matemáticas, ciencia informática y competencias terminológicas.

Este libro está diseñado para estudio personal, principalmente para compositores. Comienza con una explicación de programación básica y principios acústicos y gradualmente avanza hacia las técnicas más avanzadas de procesamiento de música electrónica. El texto asume algunos conocimientos en física e intencionalmente han sido omitidas las explicaciones de conceptos físicos básicos. La propuesta didáctica del libro se concentra primariamente en la escucha, la cual consideramos es una forma más rápida y amena de absorber nuevos conceptos que a través de fórmulas abstractas. En términos de matemáticas, sólo explico lo que es absolutamente necesario para comprender un concepto de procesamiento dado. Explico las varias técnicas desde una perspectiva composicional en lugar de intentar una discusión desde un acercamiento basado en las ciencias informáticas, matemáticas o física de los fenómenos o estructuras de procesamiento. Por lo tanto, las decisiones y comentarios que he realizado son absolutamente subjetivas y están abiertas a debate.

Este libro no habría podido ser posible sin el apoyo del Prof. Mathias Spahlinger, la supervisión experta del Prof. Orm Finnendahl, las sugerencias y patches dela comunidad Pd, la edición del manuscrito y los esfuerzos de codificación en DocBook-XML de Esther Kochte. También quiciera agradecer a Mark Barden por la traducción al inglés y a la Musikhochschule Freiburg y al estado de Baden-Württemberg por financiar el proyecto, el cual —en el espíritu del movimiento open source —hace posible que todos aquellos interesados puedan utilizar el libremente este libro en Internet. Espero que esto incremente el interés el la música electrónica, enriqueciendo indirectamente el discurso estético de la Nueva Música.

Johannes Kreidler, Enero 2008

Introducción a la metodología de este libro

El siguiente material comienza con conocimientos básicos informáticos. Por lo tanto los primeros pasos son descritos de manera meticulosa.

Pd puede ejecutarse en diferentes plataformas (como Linux, OS X, o Windows) y este libro no presupone una plataforma específica. Los problemas relacionados a los sistemas operativos no serán discutidos, dado que simplemente se encuentran más allá del alcance de este tutorial (y también es posible que los cambios —actualizaciones, arreglos de problemasm etc.— ocurran en un futuro cercano). Asumimos por lo tanto qye Pd ha sido correctamente instalado y ha sido integrado con el entorno de hardware (consulte un foro de Internet para resolver cualquiera de estos problemas, por ejemplo “Pd-list”).

Cómo utilizar este libro: cada lección comprende una parte de teoría, una parte de practica y un apéndice, como así también aspectos individuales que son examinados con gran detalle al final de cada sección. Esta información exhaustiva está dirigida a usuarios avanzados y no es esencial para adquirir conocimientos básicos de trabajo en Pd. Recomiendo trabajar a través de todo el libro sin consultar primero estos detalles adicionales, para luego volver a ellos y aprenderlos más tarde.

Aquí y allá se discutirá conceptos fundamentales de acústica. Los ejercicios contienen no sólo problemáticas composicionales específicas, sino también aplicaciones útiles para las necesidades diarias de los músicos —por ejemplo, herramientas como un metrónomo o un afinador. En este aspecto, este tutorial podría ser utilizado por intérpretes así como por compositores.

Capítulo 1. Introducción a Pd

Table of Contents

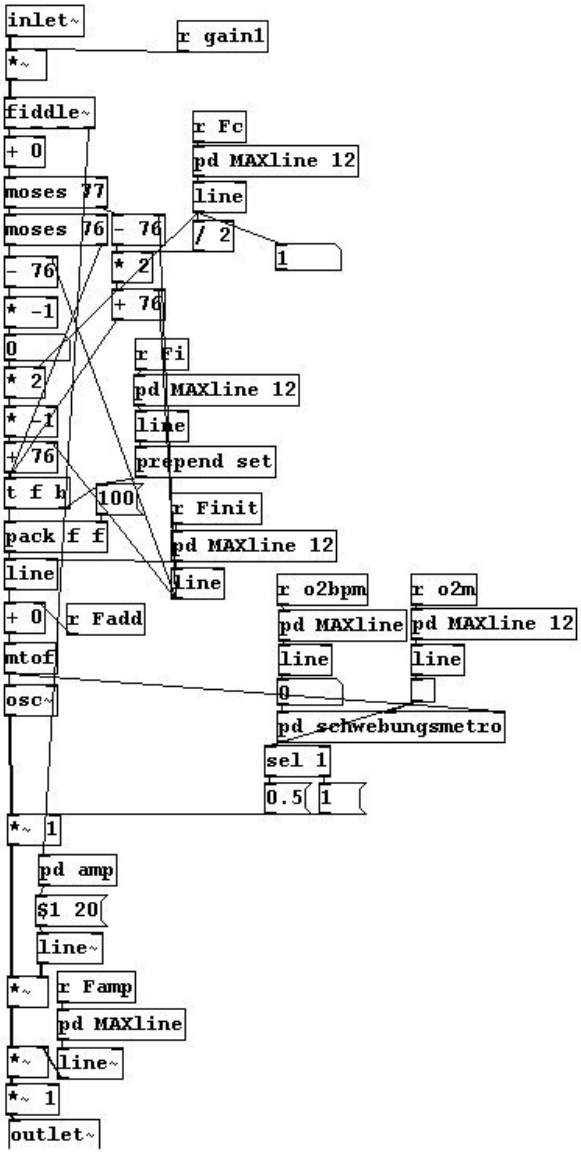
1.1 Observaciones generales

Pd (Pure Data) es un lenguaje de programación para música electrónica. Crear música en una computadora es referido técnicamente como DSP (digital signal processing: “procesamiento de señal digital”). “Digital” significa que la información es representada mediante dígitos — los ordenadores, como sabes, trabajan sólo con números. “Señal” es el término técnico para un modo especial de operación informática que trata con el sonido. “Procesamiento” refiere a funciones ejecutadas por un ordenador.

Pd fue iniciado por el ingeniero en software estadounidense Miller Puckette, quien previamente co-desarrolló el software, bien conocido y similarmente estructurado, Max/Msp. Pd no es un software comercial; no fue desarrollado por una corporación y no está a la venta. En cambio, es “open source”: su código fuente no es propiedad (patentada) de una corporación, sino que se encuentra disponible libremente para todos. Esto también significa que, siempre y cuando tengamos el suficiente conocimiento, cualquiera puede cambiar el programa. Hoy en día, muchos otros programadores, músicos, ingenieros en sonido, y compositores acompañan a Miller Puckette para continuar con el desarrollo de Pd. Como resultado de esto, no existe una versión final o definitiva de Pd; el programa se encuentra bajo desarrollo constante. Además de la gran ventaja de una disponibilidad libre en Internet, es también “democráticamente” expandido y optimizado a nivel profesional. Una desventaja de esto es que hasta el momento no ha existido un manual de operaciones detallado para usuarios que carezcan de experiencia en programación. En oposición a una corporación— que tiene un interés monetario en asegurarse que los usuarios nóveles puedan operar fácilmente un software nuevo— el movimiento open source carece de dicha fuerza conductora para hacerse asimismo accesible. Este libro es un intento de llenar dicho hueco.

En términos precisos, Pd es un “entorno de programación gráfica en tiempo real para el procesamiento de audio”. Tradicionalmente, los programadores trabajan con lenguajes de programación basados en texto. Ellos crean lo que es llamado “código”, el cual es procesador por un ordenador para producir un resultado. Para llevar a cabo sus funciones de programación, Pd utiliza objetos visuales que el usuario ubica y altera en la pantalla. Estos objetos visuales — pequeñas cajas que pueden ser conectadas entre ellas — nos retrotraen a los estudios analógicos que fueron utilizados para producir música electrónica antes de la llegada del ordenador: varios dispositivos — hoy simbolizados mediante pequeñas cajas — son conectados entre ellos utilizando líneas que — como cables — simbolizan conecciones físicas entre las cajas. (Debido a este tipo de conexión, Pd es referido como a un lenguaje de programación orientado a flujo de datos.)

Un estudio analógico – los dispositivos están conectados mediante cables.



Las cajas de Pd están conectadas entre sí.

Una gran ventaja en Pd es el aspecto del “tiempo real”. Esto significa que, en contraposición a la mayoría de los entornos de programación donde se debe entrar primero un texto que debe ser procesado posteriormente por el ordenador antes de que se obtenga un resultado, los cambios en Pd pueden ser realizados durante la performance. Como en un instrumento clásico, el usuario escucha el resultado instantáneamente y puede cambiarlo inmediatamente. Esto hace que Pd sea especialmente adecuado para su uso en vivo.

Pd se ha convertido en mucho más que un lenguaje de programación para música electrónica. A partir de que los usuarios de todo el mundo pueden participar en el proyecto , existen módulos programados por los usuarios que son llamados “externals (externos)”: video, conección a redes, integración con joysticks, etc. Librerías enteras de estos módulos también existen (“librerías externas”). Algunos de estos externos han sido integrados en la versión regular de Pd.

1.2 Instalación y configuración de Pd

Los lectores de este libro deben tener instalado Pd en sus ordenadores para que puedan probar los procesos descritos. Sin esta experiencia práctica simultánea resultará dificil entender este tutorial.

Primero necesitas un ordenador con al menos 128 MB de memoria principal (RAM), un procesador de 500 Mhz y cerca de 500 MB de espacio en el disco duro (¡estos son los requerimientos mínimos!). Pd trabaja con los siguientes sistemas operativos: Linux, OS X, y Windows.

Luego necesitas descargar de Internet la última versión de Pd-extended. Ingresa “Pd-extended” en un buscador de Internet. Como la dirección del portal de descarga puede cambiar en el futuro, no daremos un vínculo al sitio en este texto. Pd-extended es una versión del software original (tambien llamado “Pd vanilla”) que ha sido expandido con numerosas librerías. La mayoría de los ejercicios descritos aquí funcionan con la versión original de Pd, pero no todos ellos. Los objetos extra en Pd-extended hacen al progrma mucho más práctico en general. Este tutorial asume una versión 0.39 o superior de Pd-extended.

Una vez que Pd ha sido instalado, lo abrimos desde el directorio Pd/bin/. Una ventana aparece. Este es el centro de control principal, por así decirlo. Aquí puedes probar si Pd está funcionando correctamente: en el menú principal, clic en **Media > Test Audio and MIDI**. Bajo “TEST SIGNAL”, clica primero en la caja contigua a “-40”, luego en la caja contigua a “-20”. Debes escuchar un tono sinusoidal saliendo de los altoparlantes del ordenador. Si no ocurre esto, necesitas ajustar las configuraciones de hardware (bajo **Media > Audio settings**). No podemos dar aquí más información acerca de los problemas que surgen en esta etapa. Para encontrar ayuda sobre cómo resolver cualquier problema, por favor consulte “Pd-list”, un foro de usuarios de Pd en Internet, Si tiene conectado un micrófono, debería encontrar que los dígitos, en al menos las dos cajas de la parte extrema izquierda debajo de “AUDIO INPUT”, cambian en respuesta al sonido tomado por el micrófono. Puede trabajar con el programa sin un micrófono. (Sin embargo, llegando al final del Capítulo 3, necesitará un micrófono para probar algunos de los ejemplos.)

Capítulo 2. Programando con Pd por primera vez

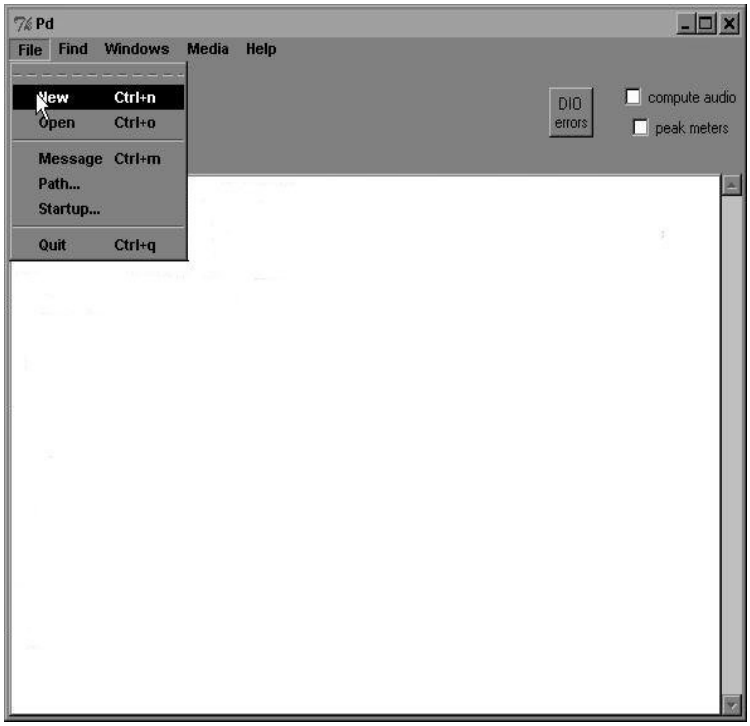
Tabla de contenidos

En este capítulo todavía no nos concentraremos en producir música, sino más bien en entender la manera en que los ordenadores y Pd se encargan de los datos. Trabajaremos con ejemplos de escucha práctica lo más frecuente que podamos para evitar innecesariamente explicaciones técnicas abstractas y áridas. Sin embargo, la manera precisa en que los ordenadores producen sonido no será explicado hasta el Capítulo 3. Ud. debe construir en Pd por sí mismo los patches de ejemplo. La experiencia de primera mano le ayudará a solidificar los conceptos presentados. A partir del Capítulo 3, puede encontrar los patches más grandes en: http://www.kreidler-net.de/pd/patches/patches.zip

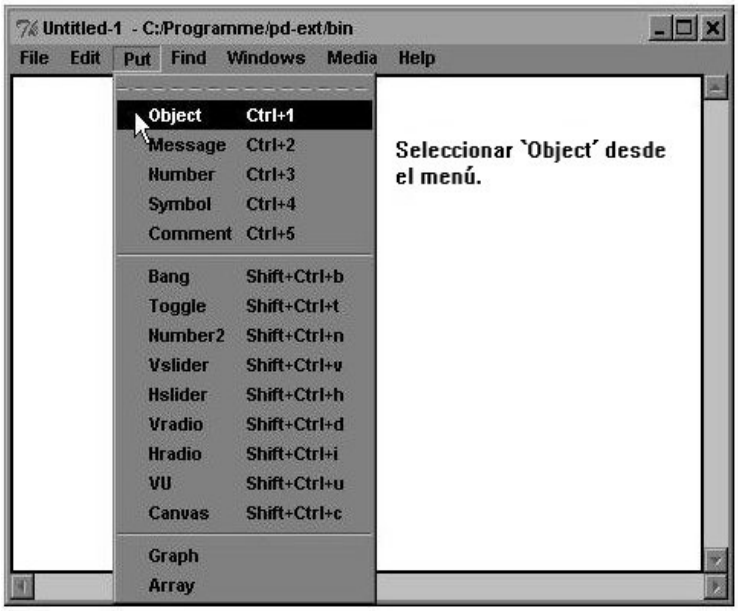
2.1 Introducción

2.1.1 Un ejemplo simple

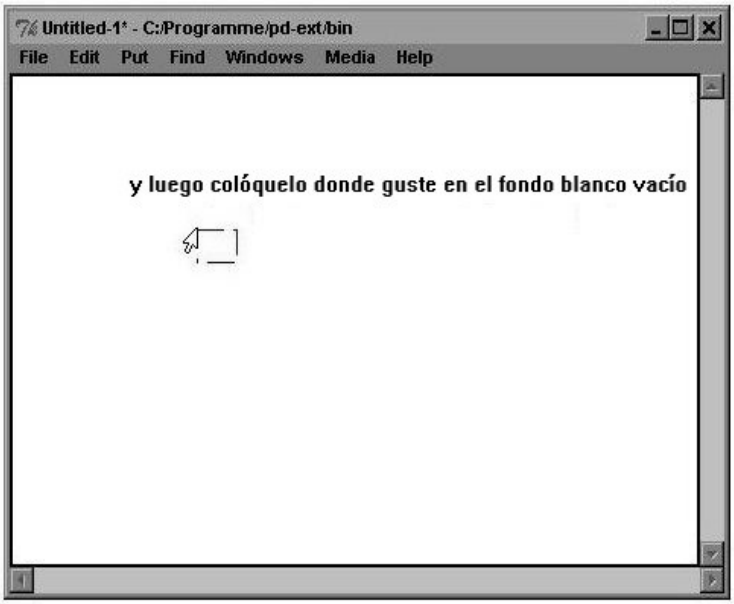
Una vez que haya iniciado Pd, aparece en la pantalla la ventana principal de Pd. Abra una nueva ventana de programación clicando **File > New**.



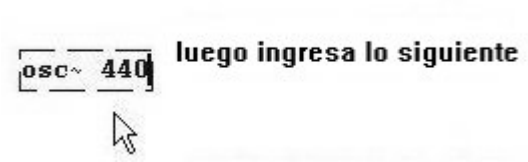
Una nueva ventana se abre. Añada una caja objeto bajo **Put > Object**, o con el teclado, utilizando el comando: **Ctrl-1** (esto funciona bajo Windows; otras plataformas pueden tener diferentes comandos).



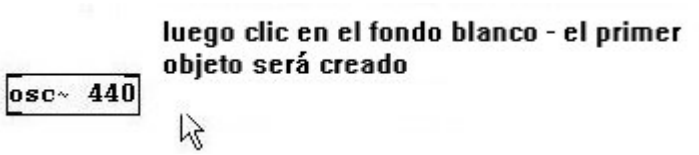
... ahora debe ver una caja azul anclada al cursor del ratón ...



Luego clic en algún lugar de la superficie blanca in la ventana nueva para disociar la caja objeto del cursor del ratón. Tipee lo siguiente en la caja: “osc~ 440”.

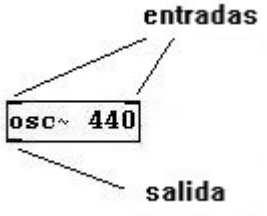


Para aceptar lo que haya tipeado en la caja, clic afuera de la caja sobre la superficie blanca:

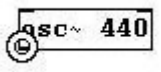


(El signo "~" significa "tilde (en castellano: virgulilla)"; necesitará utilizar este signo muy seguido en Pd. La correspondencia numérica en el código ASCII es 126).

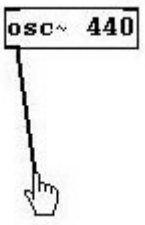
Ahora puede observar una caja rectangular con pequeños rectángulos negros sobre y debajo de las esquinas. Los rectángulos superiores son llamados "inlets (entradas)", el rectángulo inferior es un "outlet (salida)".



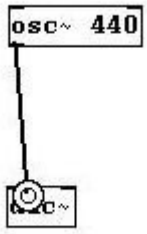
Si posiciona el cursor sobre el rectángulo de salida, éste cambia a un círculo (el cual se asemeja a un zócalo abierto para un cable).



Ahora, haga clic sobre el rectángulo y mueva el ratón mientras mantiene presionado el botón del mismo. Verá que se dibuja una linea la cual puede ser pensada como si fuera un cable.

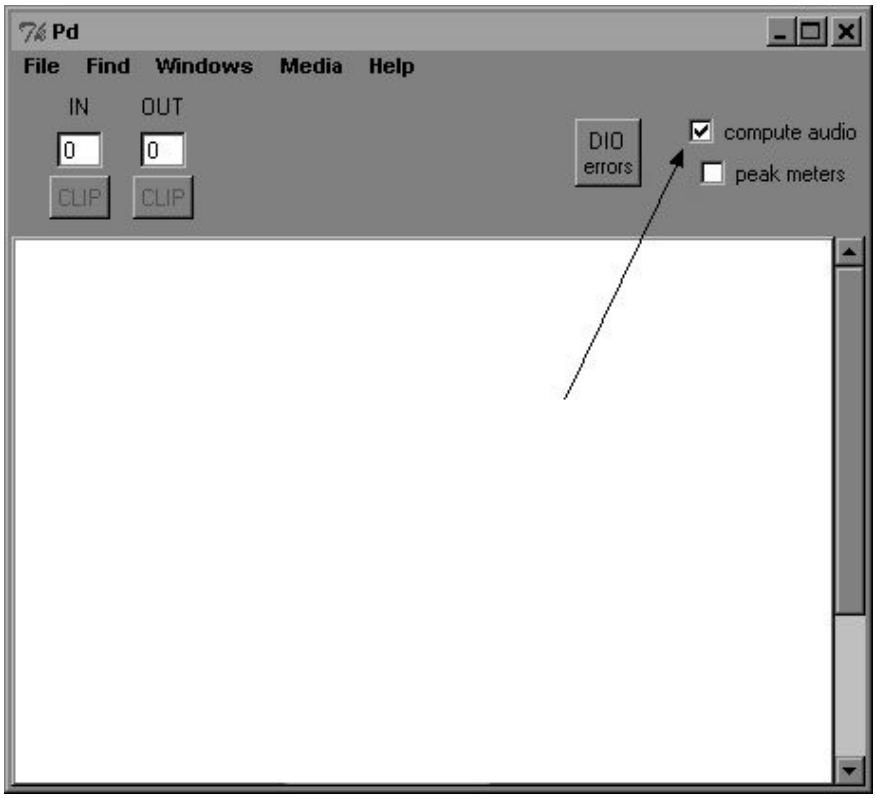


Pero como todavía no ha creado un objeto al cual vincular este cable, el mismo se desaparecerá cuando libere el botón del ratón. Cree otro objeto como en anterior y escriba dentro del mismo "dac~". Posicionelo debajo del objeto "osc~" mediante un clic sin soltar el botón, verá que la caja se pinta de azul. Luego despliegue un cable desde la salida de "osc~" y conéctelo a la entrada izquierda de "dac~". El cursor cambia a un círculo cuando se encuentra encima de la entrada mencionada.



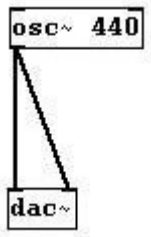
Libere el botón del mouse. El "cable" ahora conecta "osc~ 440" a "dac~". Debería estar escuchando un tono. Si esto no ocurre, verifique en la ventana principal de Pd que la caja a la izquierda de "compute audio" esté validada (en Linux: verifique que la caja esté coloreada en rojo). Si no es así,

valídela con un clic:



(La función “compute audio” le permite programar en Pd sin generar sonido. Esto le permite al ordenador ahorrar mucho poder de procesamiento innecesario — aunque esto probablemente un aspecto no necesario con los ordenadores de hoy en día.)

Escuchamos un tono. Para ser específicos, es un A4 (a' en el sistema alemán), también llamado A440, la altura de afinación de concierto que posee una frecuencia de 440 Hertz (los significados de “frecuencia” y “Hertz” serán explicados más adelante). Ahora conecte también la salida de “osc~ 440” con la entrada derecha de “dac~”.

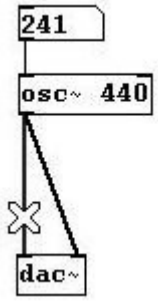


Debería escuchar el tono procediendo de ambos altoparlantes del ordenador. Ahora cree una caja de número (**Put** > **Number** o el comando **Ctrl-3**) y vincule su salida en la entrada izquierda del objeto “osc~”. Luego necesita cambiar a lo que se llama “Execute mode (modo Ejecución)” (**Edit** > **Edit mode**, o el comando **Ctrl-E**; el cursor cambia a un puntero). Clic en la caja de número manteniendo presionado el botón, y mueva el ratón hacia arriba y abajo:

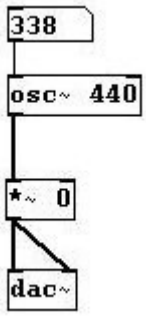
Esto cambia los números y la altura. El valor debería ser al menos 100; este rango puede ser ajustado de manera fina manteniendo presionada la tecla SHIFT mientras se clica y mueve el ratón como se describió anteriormente.

Otra manera de ingresar valores en la caja de número es clicar en la misma, ingresar un valor en el teclado y presionar ENTER

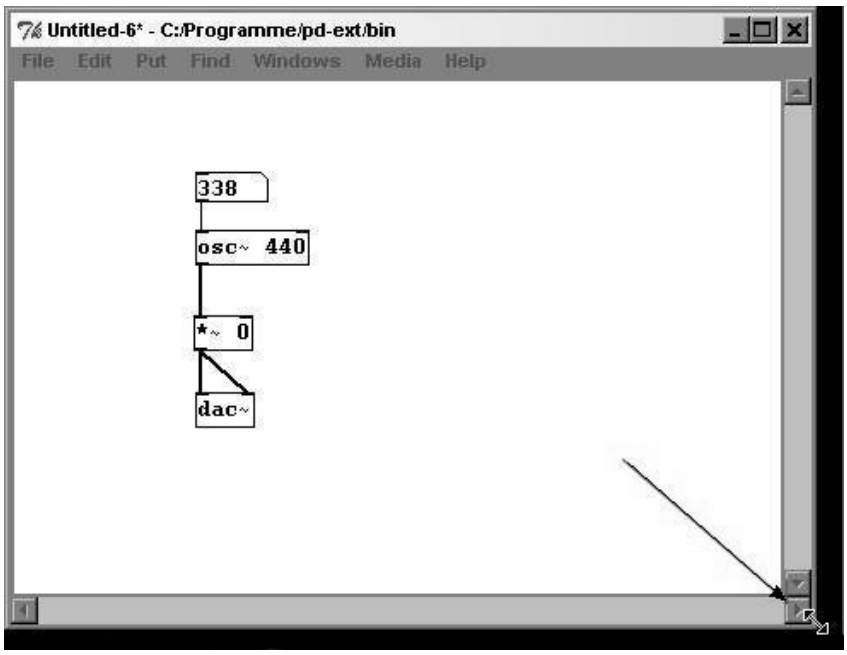
Ahora regrese al modo anterior, el "Edit mode (modo de edición)" (**Edit > Edit mode**, o el comando **Ctrl-E**). Mueva el cursor, el cual habrá cambiado nuevamente a Mano, sobre la conección entre "osc~" y "dac~". El cursor se convierte en una X. Clic sobre el mismo, el cual se coloreará en azul.



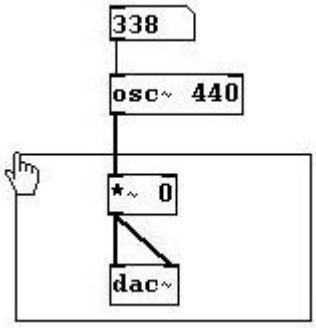
Luego vaya a **Edit > Cut** o simplemente presione BACKSPACE. Esto elimina la conección. Corte también la otra conección a "dac~". Ahora cree un nuevo objeto donde los cables solían estar: "*~ 0" y conectelo a los otros objetos como se muestra a continuación:



Hagamos un poco más de espacio: Alargue la ventana clicando en la esquinainferior derecha, mantenga el botón presionado, y tire hacia abajo y a la derecha.

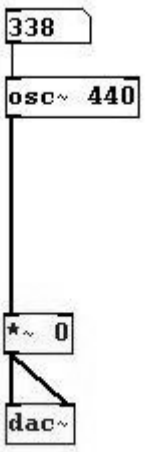


Luego clique, manteniendo el botón presionado, en la parte derecha del fondo blanco cerca del objeto “dac~” y dibuje un rectángulo que incluya los objetos”dac~” y “*~”.



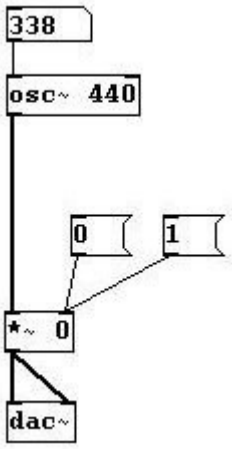
Así es cómo Ud. Selecciona una parte del patch. (También puede borrar cajas de esta manera. Luego de seleccionarlas, vaya a **Edit > Cut** o simplemente presione BACKSPACE.)

Cuando usted libera el botón del ratón, ambos objetos se colorean en azul. Clic en alguno de estas cajas seleccionadas, mantenga el botón presionado, y muevalos hacia abajo para liberar espacio.



Para deseleccionar estos objetos, sólo cliquee en cualquier lugar del fondo blanco.

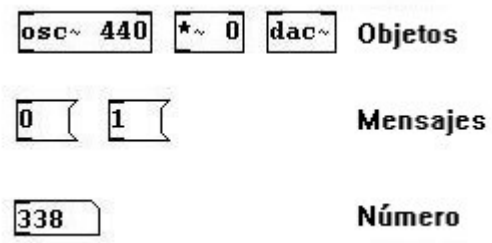
Luego cree dos cajas “Message (mensaje)” (**Put** > **Message** o **Ctrl-2**) como se muesta abajo e ingrese “0” en uno y “1” en el otro.



Cambie a Modo Ejecución (**Edit > Edit mode** o **Ctrl-E**) y cliquee en las dos cajas de mensaje alternativamente: clicando 1 enciende el sonido, clicando 0 lo apaga.

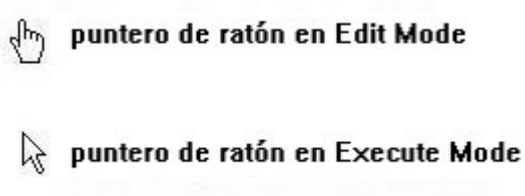
2.1.2 Elementos de superficie en Pd

El ejemplo previo cubre la mayoría de los elementos en Pd. Veámos un poco mas de cerca dichos elementos – utilizamos tres tipos distintos de cajas: *Objecto*, *Mensaje*, y *Número*.



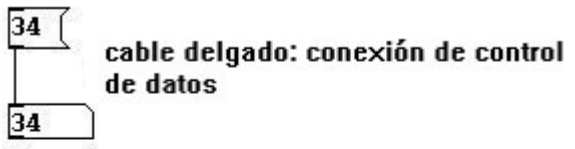
Las cajas Objeto son rectangulares, las cajas Mensaje tienen las esquinas salientes en el lado derecho, y las cajas Número tienen una hendidura en la esquina superior derecha.

Todas estas cajas tienen entradas y salidas. Las entradas están siempre en la parte superior, las salidas en la parte inferior. Siempre puedes conectar una salida a una entrada (en este orden). Existe un modo Edición y un modo Ejecución. El modo Edición se utiliza para programar y el modo Ejecución se utiliza para hacer funcionar el programa, Puedes saber en qué modo te encuentras mirando el cursor:

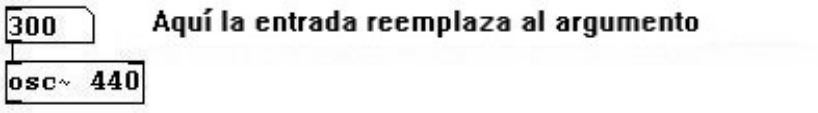


Veamos con detenimiento: Existen dos tipos de cables, uno grueso y otro delgado. Un cable delgado conecta la caja Número al objeto “osc~” y un cable grueso sale del objeto “osc~”. Los cables gruesos transmiten *señales*, mientras que los cables delgados transmiten sólo *datos de control*. Con “compute audio” en la ventana principal de Pd, determinamos si debemos o no enviar las señales mediante el marcado o desmarcado de la casilla de verificación. Además, todos los objetos que producen señals o que trabajan con señales como entrada (entrada = lo que se dirige a un inlet;

salida = lo que resulta de un outlet), poseen una tilde o virgulilla (“~”") después de su nombre ; ¡otros objetos no lo tienen! Estos dos niveles son llamados “nivel de control” (donde sólo fluyen datos de control, también llamado “dominio de mensaje”) y el “nivel de señal” (donde fluyen señales, también llamado “dominio de señal”).



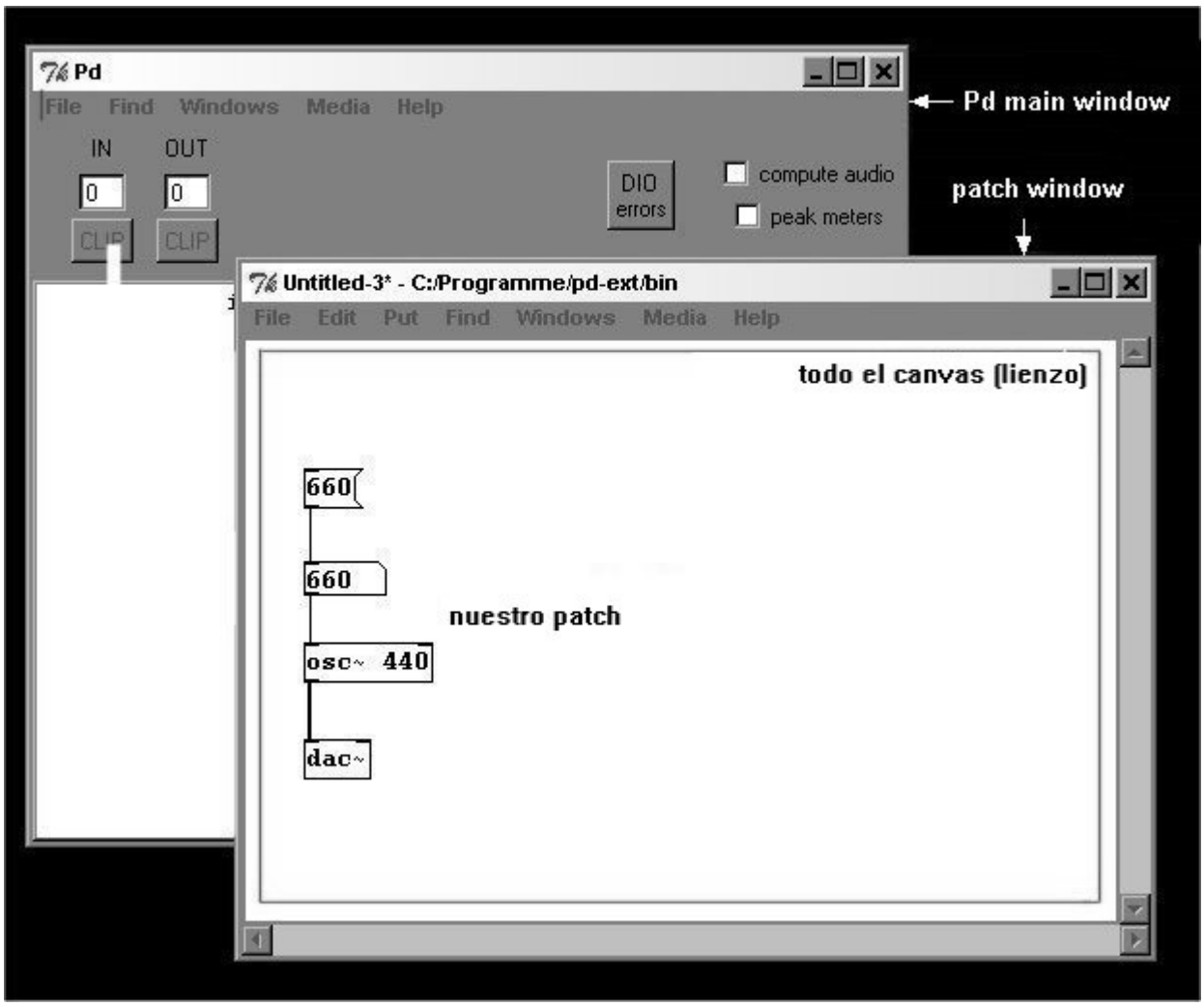
El primer Objeto que creaste fue “osc~ 440”, el cual es un “oscilador”, y escuchaste un tono sinusoidal de 440 Hertz (el significado de “Hertz” será explicado más tarde). Luego creaste una caja Número e ingresaste allí nuevos valores, los cuales causaron cambios en la frecuencia del tono. Esta es la estructura básica en Pd: un Objeto tiene un nombre (si produce señales, una tilde sigue al nombre), luego tenemos un expacio, y luego le siguen uno o varios *argumentos* (en este caso, el argumento inicial fue “440”). En la mayoría de los Objetos, los argumentos pueden ser reemplazados con nuevos valores que se encuentran conectados a las entradas (al contrario de lo que ocurre aquí con el Objeto “osc~”, el valor cambiado usualmente ingresa en la entrada extrema derecha).



Si ingresamos nuevos valores de esta manera, el argumento escrito en la caja es ignorado (en este caso, 300 reemplaza a 440).

Podemos ingresar información mediante cajas Número o cajas Mensaje. Las cajas Mensaje también permiten ingresar letras, los cuales son llamados *símbolos*. Toda esta información es referida como *atoms* (átomos). Un átomo en una caja Mensaje o en una caja Número (para más sobre átomos, ver 2.1.5).

Otro término importante: El programa que escribes es llamado *patch*. Un patch aparece en principio como un fondo blanco sobr el cual escribes un programa. Este fondo blanco también es llamado *canvas*.



2.1.3 Resumen

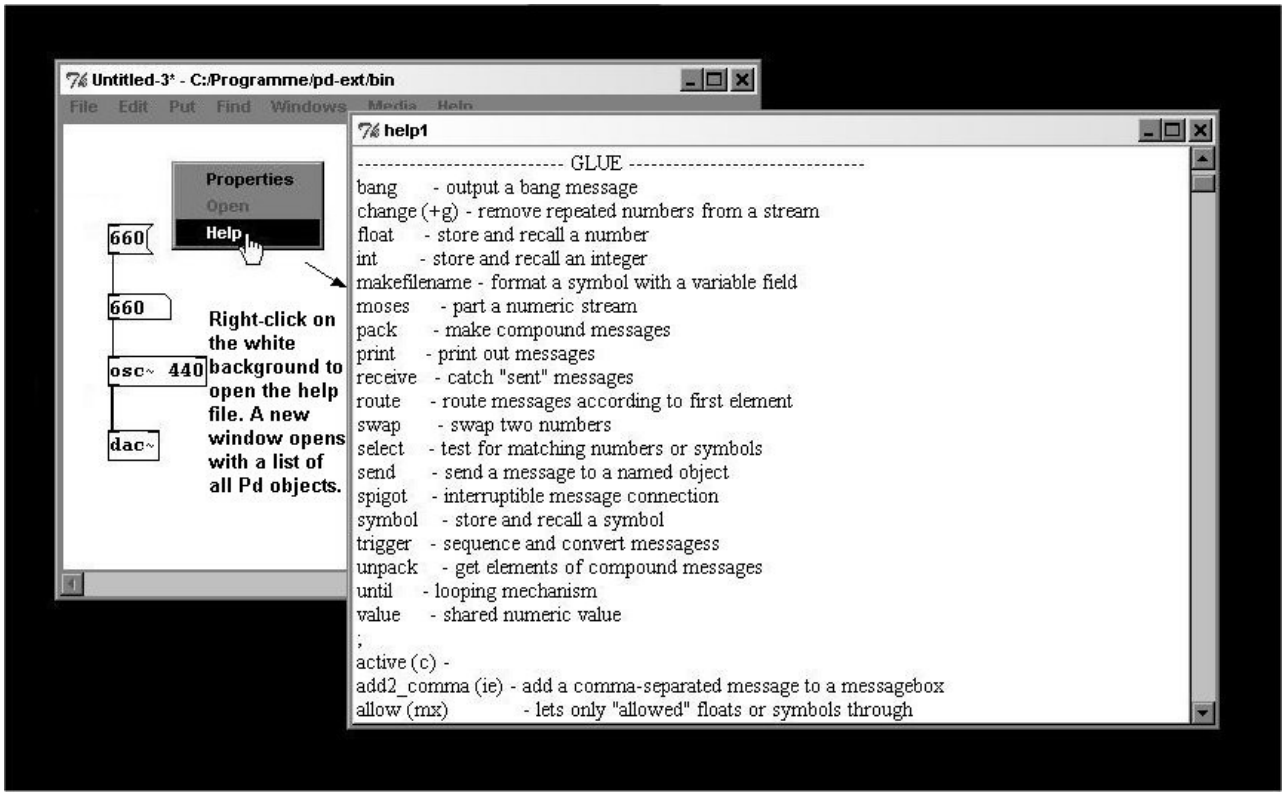
- Existen dos modos: modo Edición y modo Ejecución (puedes cambiar entre ellos con **Ctrl-E** o bajo **Edit > Edit mode**). Programas todas las partes de un patch en modo Edición y comienzas todas las operaciones y sonidos en modo Ejecución.
- Dentro de un patch, existen el nivel de control y el nivel de señal (los Objetos de controls no tienen una tilde a continuación de sus nombres y están conectados con cables delgados; los Objetos de señal tienen una tilde al final y están conectados con cables gruesos). El nivel de señal sólo se activa si "compute audio" ha sido activado en la ventana principal de Pd.
- Los elementos de un patch son *Objetos*, *Mensajes* y *Números*.
- Un Objeto a menudo posee uno o varios argumentos (a.k.a.
- An object often has one or several arguments (alias "argumentos de creación"), los cuales pueden ser cambiados utilizando una entrada.
- Un Mensaje es un valor fijo en el modo Ejecución y es salvado en el patch. Cuando una caja de mensaje es clicada, su contenido en enviado a todos los objetos conectados a su salida. En cambio, las cajas Número pueden ser alteradas en el modo Ejecución y los cambios en sus valores no son salvados.

2.1.4 Apéndice

Algunas cosas adicionales que pueden facilitar su trabajo en Pd:

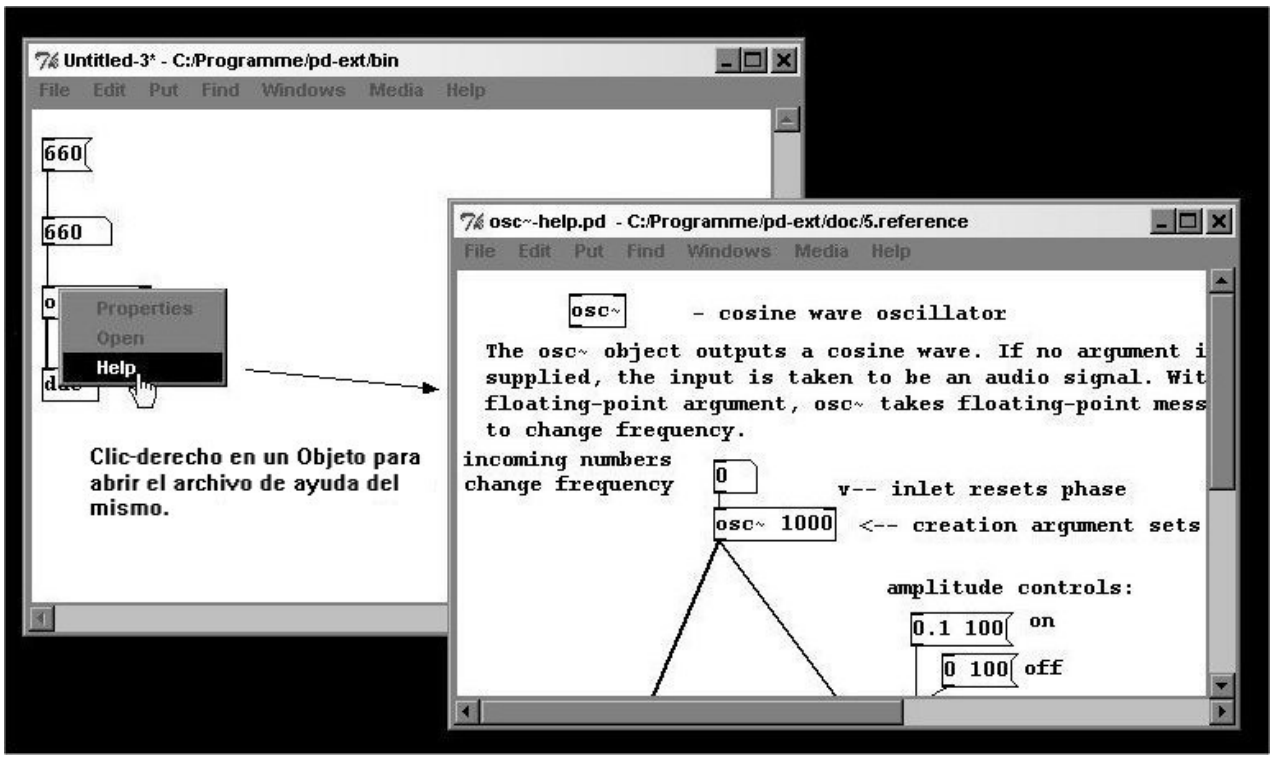
2.1.4.1 Lista de todos los objetos

Si clica en la superficie blanca (“canvas”) con el botón derecho y abre el menú **Help**, aparece una lista con todos los Objetos de Pd.



2.1.4.2 Archivo de Ayuda

Si clica con botón derecho sobre un Objeto, un menú desplegable aparece donde puedes seleccionar el archivo de **Help** (ayuda) donde podrá contar con una explicación detallada del mismo.



2.1.4.3 Duplicación

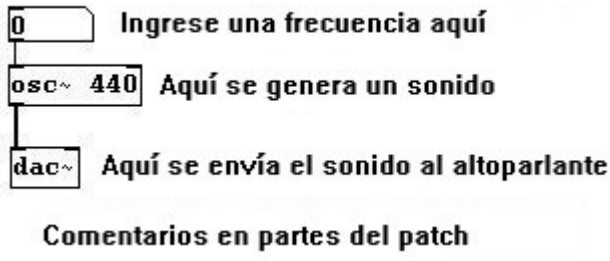
Pronto encontrarás muy útil duplicar partes de patches. Para poder realizarlo, selecciona un área de manera tal que las cajas seleccionadas se vuelvan azul (como fue descrito en 2.1.1, en el contexto de hacer más espacio en el canvas) e ir a **Edit** > **Duplicate** o **Ctrl-D**. Esto duplica el área seleccionaday la copia aparece como área seleccionada la cual puede ser movida a continuación (clic en una de las cajas seleccionadas, mover a un lugar deseado con el ratón, y liberarlo).

2.1.4.4 Atajos

Es mucho más rápido y más confortable trabajar si utiliza los “atajos de teclado”. Muchas funciones que puedes seleccionar desde los menues desplegables también están disponibles como atajos de teclado. Estos comandos aparecen a continuación de la función en los menúes desplegables.It is much faster and more comfortable to work if you use "keyboard shortcuts".

2.1.4.5 Comentarios

La programación se puede complicar rápidamente. Para ayudarte a recordar el sentido de un patch dado, es recomendable que *agregue comentarios* en el patch. Los comentarios pueden ser añadidos bajo **Put > Comment** (o **Ctrl-5**). De esta manera puede escribir lo que necesite para explicar su patch.



Si ha podido entender todo hasta aquí, entonces Ud. Entiende los fundamentos esenciales de la interfaz de usuario de Pd. Ahora podemos ingresar a la estructura de la programación misma.

2.1.5 Para los interesados, especialmente: Atoms

Un Mensaje para un Objeto tiene dos partes: un designador de método (selector) y ninguno, uno, o varios valores (argumentos). Por ejemplo, si el Mensaje es “5”, entonces el mensaje real es “float 5” y está compuesto de los átomos “float” y “5”. El Mensaje “bang” está compuesto sólo del selector “bang” por lo tanto no tiene argumentos. El Mensaje “1 2 3 4 5” es realmente un mensaje “list 1 2 3 4 5”.

Existen tres clases de átomos: un Número (lenguaje de programación = “float (número con decimal)” con un valor de 32-bit, un Símbolo, el cual es una cadena de letras, o un puntero, el cual es una clase de dirección (esto será cubierto en el Capítulo 5.2.3).

El Mensaje “float 5” está compuesto por los dos tipos de designación Símbolo y Float. El tipo Símbolo tiene un valor de “float” (una cadena de texto) y el tipo Float tiene un valor de “5”.

El selector siempre es un Símbolo.

The selector is always a symbol. Dado que los Objetos pueden reaccionar de manera diferente a diferentes mensajes, el selector en primer lugar realiza una más precisa determinación preliminar.

2.2 El nivel de control

En principio nos pondremos a repasar en Pd, aquello que concierne al funcionamiento básico del nivel de control. Como ya hemos mencionado, Pure Data trabaja sólo con datos, es decir, con números (y la ayuda de letras). (Sin embargo, en los ejemplos trabajaremos a la brevedad con sonido procesado.)

2.2.1 Operaciones matemáticas y orden

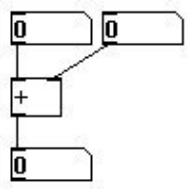
2.2.1.1 Teoría

2.2.1.1.1 Funciones matemáticas básicas

Como ya lo hemos dicho, los ordenadores trabajan sólamente con números. Pd lo hace con *números* y *símbolos* ("symbols"), en otras palabras: letras. Pero los números tienen mayor significancia; en el primer ejemplo vimos cómo algunos parámetros importantes del sonido, tales como altura o volumen, no son determinados en Pd usando indicaciones musicales tradicionales como *C4* para la altura o *pianissimo* para la dinámica, sino que se utilizan números exclusivamente. Es por esta razón que dedicaremos tiempo a aprender lo esencial sobre cómo Pd procesa números en el nivel de control:

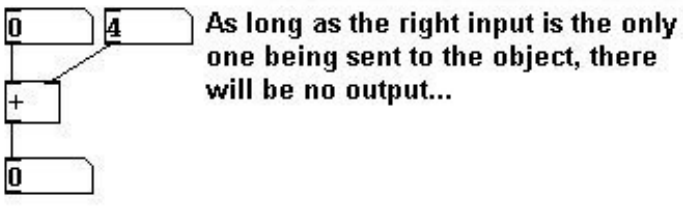
Puede ingresar números en las cajas Número y Mensaje. Algunos Objetos permiten realizar

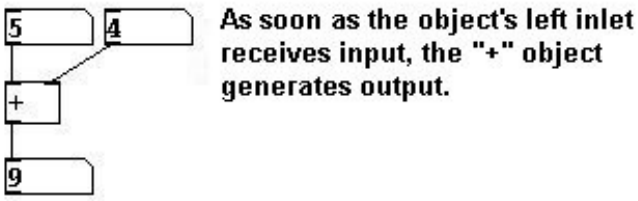
operaciones matemáticas usando estos números. Cree el Objeto "+" y conecte cajas Número a las entradas derecha e izquierda, como así también a su salida:



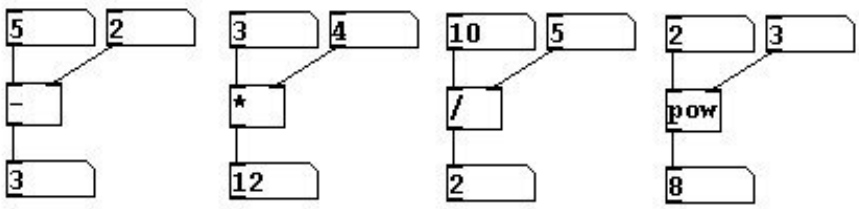
Ingrese un 4 en la caja Número superior derecha (en Execute Mode: clic en la caja Número una vez, tipee el número, presione ENTER) y 5 en la caja superior izquierda. El número 9 -la suma de 4 y 5- aparece en la caja inferior. El Objeto "+" tiene dos entradas en las cuales podemos ingresar números, y una salida la cual expide el resultado procesado por el Objeto (en este caso, un proceso de adición).

Este ejemplo ilustra una importante regla en Pd: con los Objetos de control que poseen varias entradas, usted debe ingresar los datos en las entradas de derecha a izquierda. En otras palabras, un objeto recibe ingresos pero sólo crea una egreso cuando recibe un ingreso en la entrada del extremo izquierda. (Distinguimos entre entradas "pasivas", las cuales no causan un cambio inmediato, y entrada "activa", la cual provoca un cambio inmediato y visible cuando se ingresan datos en ella.) Encontraremos constantemente esta regla en los demás Objetos.

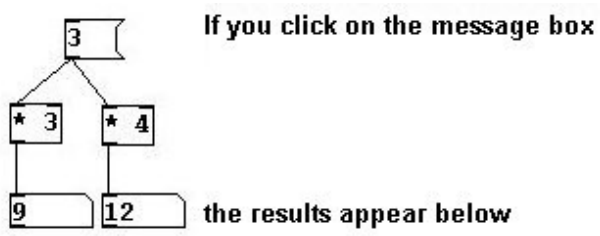




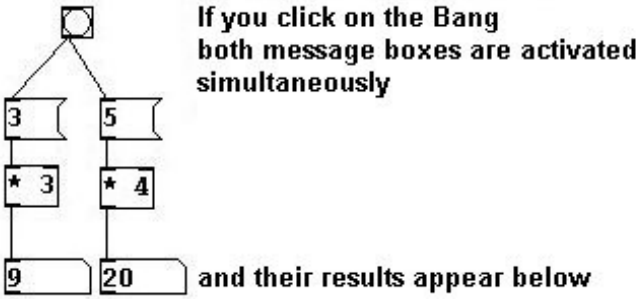
Las otras operaciones matemáticas básicas -sustracción, multiplicación, división y potencias- siguen el mismo principio:



Si quiere realizar varias operaciones usando un número, por ejemplo, 3*3 y 3*4 entonces sólo conecte la caja de Número o Mensaje a varias entradas (por una cuestión de simplicidad usaremos argumentos ("* 3" y "* 4") en vez de entradas para los factores. En los ejemplos previos, usamos entradas en vez de argumentos. Si ingresamos un Objeto sin un argumento, Pd asume un valor de "0" para el argumento):

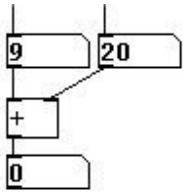


Si quiere realizar dos cálculos diferentes al mismo tiempo, tiene que transformar un clic de ratón en muchos usando un "bang" (**Put** > **Bang** o **Shift-Ctrl-B**). Puede hacer clic en el bang en Execute Mode.

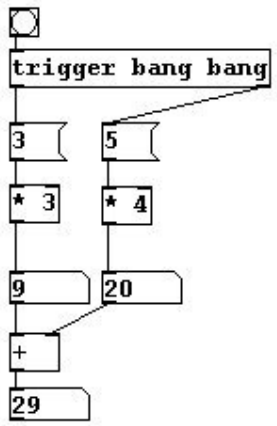


2.2.1.1.2 Orden

Si luego quiere sumar estos dos resultados, debe asegurarse que estos ingresen al Objeto "+" en el orden correcto, es decir de derecha a izquierda.

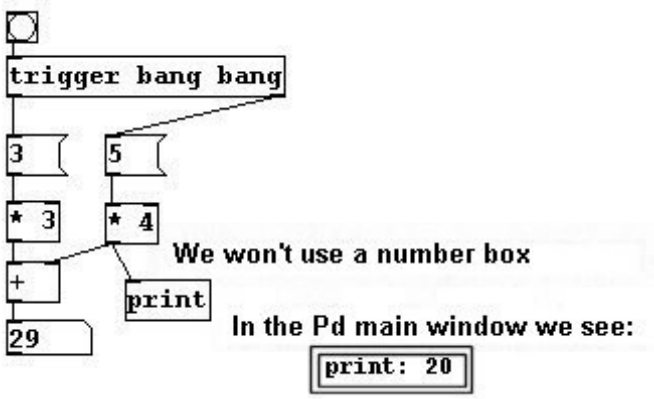


Para lograr esto, existe lo que se llama Objeto "trigger":



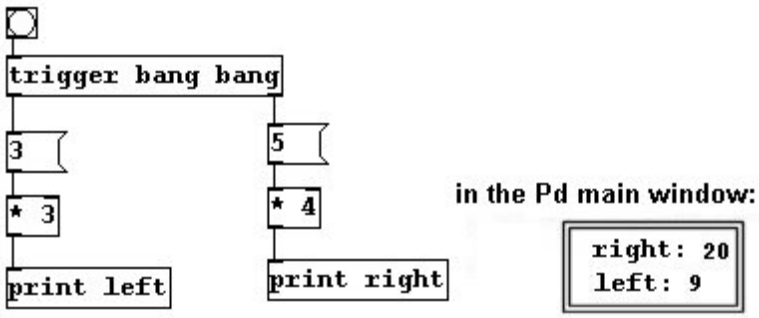
El Objeto "trigger" puede recibir: bang, número, símbolo, puntero o lista (más tarde ampliaremos sobre punteros y listas). Una vez iniciado de esta manera, "trigger" egresa esta entrada o la transforma en un bang a la salida, en el orden *de derecha a izquierda*. La salida del Objeto "trigger" es determinada por sus argumentos (bang, float, symbol, pointer, list). En el presente caso, los argumentos usados corresponden a dos bangs, por lo tanto son creadas automáticamente dos salidas (una salida es creada por cada argumento ingresado).

Para ahorrar espacio, puede omitir las cajas Número de las primeras operaciones para los resultados que quiere sumar, simplemente usando las salidas superiores directamente como entradas de las inferiores. Si alguna vez quiere saber qué valor está siendo enviado, le basta con agregar un Objeto "print" a la salida.



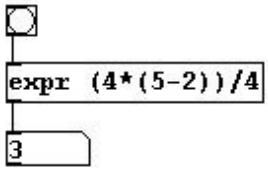
Una entrada del Objeto "print" aparecerá en la ventana principal de Pd. Todos los errores que ocurren también aparecen en esta ventana. Por ejemplo, si intentamos crear el Objeto no existente "zzzgghhh", este no será creado en el patch y aparecerá un mensaje de error ("zzzgghhh ... couldn't create") en la ventana principal de Pd.

Puede usar esta ventana para reconocer la manera en que "trigger" trabaja mediante la creación de numerosos Objetos "print" y dandole a los mismos diferentes argumentos. Los resultados aparecen en la ventana principal de Pd, uno debajo del otro, es decir, uno luego del otro cronológicamente (para profundiza en el orden de operaciones, cf. 2.2.1.4):



2.2.1.1.3 Expresiones

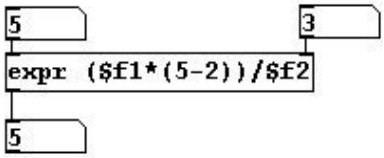
Expresiones matemáticas más extensas pueden ser programadas usando el Objeto "expr". En este caso, el argumento es la expresión misma (usando paréntesis cuando sea necesario, ¡como si estuvieramos nuevamente en una clase de matemáticas!):



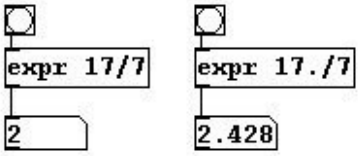
Para generar el resultado, debe ingresar un bang.

También podría usar "variables"; son llamadas $f1, $f2, $f3 etc. (el conteo comienza en 1). Esto crea entradas de izquierda a derecha, en donde se deben ingresar los valores para las variables (como siempre, la salida sólo es generada cuando la entrada extrema izquierda recibe un valor. Es

por esto que se debe asegurar que todos los demás valores hayan sido ingresados antes del correspondiente a la entrada extrema izquierda).



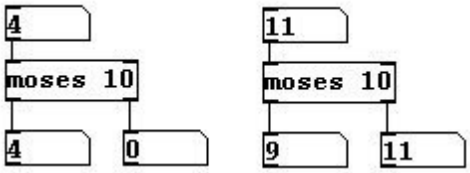
Note Bien: Si desea que una operación "expr" (sin entradas) genere un 'número decimal' (es decir, uno con decimales y no sólo un número entero), debe incluir un punto decimal en uno de los valores de la operación (para ampliar sobre números deciales, vea 2.2.1.4).



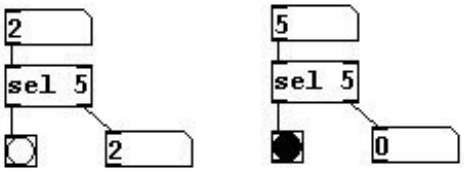
Las funciones exponenciales (alias de las operaciones 'elevado a la potencia de') siguen la siguiente sintaxis: "expr pow ([Base], [Exponente])". Por ejemplo, para elevar 2 al cubo: "expr pow (2, 3)".

2.2.1.1.4 Otras operaciones matemáticas

"moses": Se espera a un número como entrada, "moses" decide, evaluando si este número es */menor que/* o */mayor o igual que/* el argumento, y que salida la enviará. Si en "moses" incluye un argumento de 10 e ingresa una entrada menor a 10, este ingreso saldrá por la salida izquierda. Si la entrada es de 10 o un valor más grande, será enviado a la salida derecha.

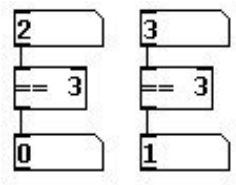


"select" (usualmente abreviado como "sel"): Recibe un número como entrada, la salida es un bang sólo cuando la entrada es igual al argumento. Cualquier otro número que reciba como entrada egresa por la salida extrema derecha.



Evaluaciones condicionales

"= =": Si la entrada izquierda es *equivalente* al argumento o a la entrada derecha, l salida es 1, de lo contrario 0:



"> =": Si la entrada izquierda es *mayor o igual que* el argumento o a la entrada derecha, la salida es 1, de lo contrario 0.

">": Si la entrada izquierda es *mayor que* el argumento o a la entrada derecha, la salida es 1, de lo contrario 0.

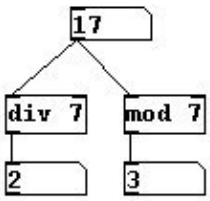
"! =": Si la entrada izquierda es *no equivalente* al argumento o a la entrada derecha, la salida es 1, de lo contrario 0.

"<": Si la entrada izquierda es *menor que* el argumento o a la entrada derecha, la salida es 1, de lo contrario 0.

"< =": Si la entrada izquierda es *menor o igual que* el argumento o a la entrada derecha, la salida es 1, de lo contrario 0.

Otros dos temas matemáticos:

El resultado de una operación de división (cociente) puede ser expresado en forma decimal (17 / 7 = 2.428) o en forma de 'resto': 17 / 7 = 2 resto 3. Un cociente con un "resto" puede ser obtenido en Pd con "div" y "mod":



Luego, existen también otras operaciones matemáticas importantes (para información más específica sobre estas funciones consulte un manual de matemáticas de nivel superior):

"sin" = Seno

"cos" = Coseno

"tan" = Tangente

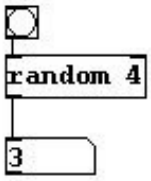
"log" = Logaritmo (natural)

"abs" = Valor absoluto

"sqrt" = Raiz cuadrada

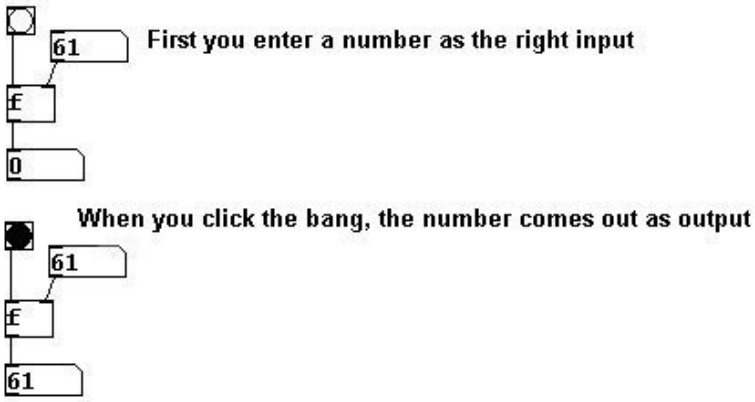
Finalmente, existe un algoritmo (los algoritmos son operaciones matemáticas que el ordenador calcula utilizando valores ingresados):

"Random" crea un número aleatorio dentro de un rango dado. El límite inferior tiene un valor por defecto de 0, el límite superior es ingresado como argumento (sólo números enteros). El límite superior es exclusivo; es decir, si ingresa "random 4", cada vez que el Objeto reciba un bang como entrada, seleccionará de manera aleatoria una salida como 0, 1, 2, o 3.



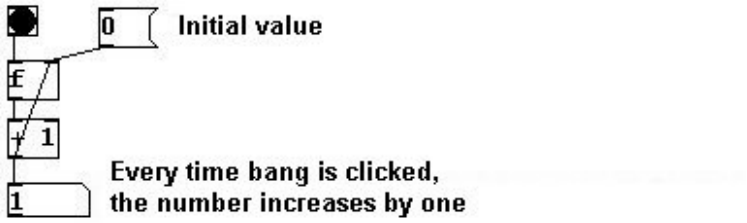
2.2.1.1.5 Objeto float y contador

Otro Objeto importante en el contexto de las operaciones numéricas, es el Objeto "float" (abreviado: "f"). Este Objeto es usado para almacenar números. Cuando ingresa un número en la entrada derecha, es almacenado en el mismo para su uso posterior. Si envía un bang a la entrada izquierda, el número almacenado es dirigido a la salida (para ampliar sobre "float", cf. 2.2.1.4).



También puede enviar un número directamente a la entrada izquierda. Esto causa su egreso inmediato como salida. El número también es almacenado en el Objeto para un uso posterior y puede ser reenviado usando un bang.

En Pd, habitualmente querrá utilizar un "contador" que incremente su valor de entrada inicial usando números enteros. Aquí tiene un ejemplo:



Explicación:

Primero ingresa al Objeto "f" un valor inicial de "0". La primera vez que haga un clic en el bang conectado a la entrada izquierda, "f" envía un 0 al Objeto "+ 1". Entonces este Objeto genera 0 + 1 = 1. Este 1 luego se dirige a la entrada derecha del Objeto "f". La próxima vez que envíe un bang, este 1 será enviado como salida al Objeto "+ 1", que en su momento generará el valor 2.

2.2.1.1.6 Resumen

- Estos Objetos para operaciones matemáticas demuestran claramente una regla importante en Pd: las entradas para un Objeto de control siempre deben ser ingresadas *de derecha a izquierda*. Para asegurarnos esto, necesitamos emplear seguido un Objeto "trigger", el cual

envía salidas de derecha a izquierda, una detrás de la otra.

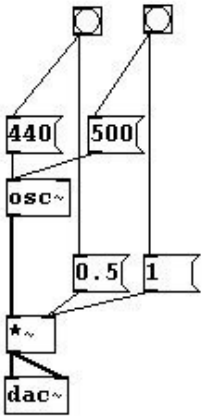
- Un "bang" responde como un clic de ratón, el cual puede ser enviado o recibido.
- El Objeto "print" imprime en la ventana principal de Pd las salidas generadas en el tiempo de ejecución de su patch. Las salidas enviadas una detrás de la otra en tiempo aparecen una debajo de la otra en la lista; es decir, la salida que se encuentra al final de la lista es la más reciente.

2.2.1.2 Aplicaciones

Ahora veamos cómo aplicar estos conceptos (todo lo relacionado al sonido será explicado más tarde):

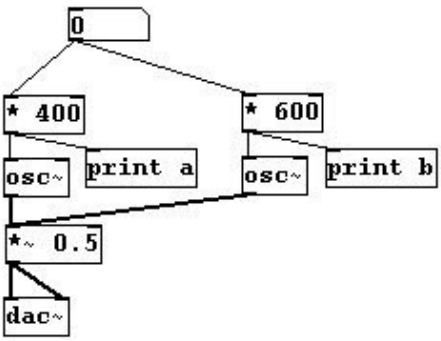
2.2.1.2.1 Dos frecuencias – dos niveles de volumen

Si quiere alternar entre dos frecuencias -un tono grave suave y un tono agudo intenso- puede utilizar el siguiente patch. Para seleccionar el tono deseado haga clic en su respectivo bang:



2.2.1.2.2 Un intervalo

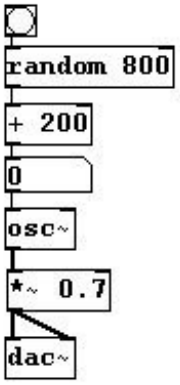
Para producir una diada, necesitará dos Objetos "osc~". En el siguiente patch, el mover la caja Número hacia arriba y abajo producirá un intervalo musical (en este caso, una quinta perfecta) a varias alturas:



Dado que se encuentran presentes Objetos "print", las frecuencias de estos dos tonos ser´´a presentados en la ventana principal de Pd.

2.2.1.2.3 Melodía aleatoria

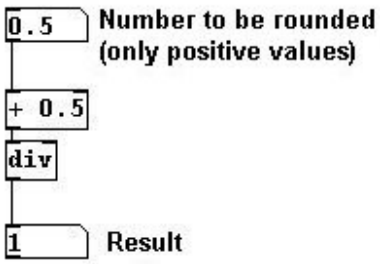
¡Indeterminación!



Cada bang que envíe generará una altura entre 200 y 1000 Hetz -una melodía aleatoria.

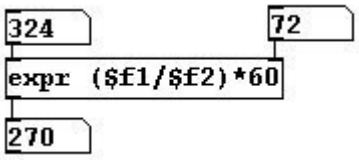
Ahora, algunos otros ejemplos de operaciones matemáticas:

2.2.1.2.4 Redondeo



2.2.1.2.5 ¿Cuánto dura esta partitura?

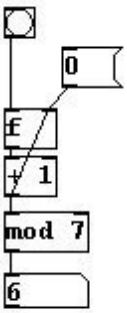
Un valor que los compositores necesitan calcular una y otra vez: usted ha escrito una pieza con 324 negras a un tempo de negra = 72. ¿Cuánto dura la misma en segundos?



Resultado: 270 segundos o 4 minutos y 30 segundos.

2.2.1.2.6 Contando una serie

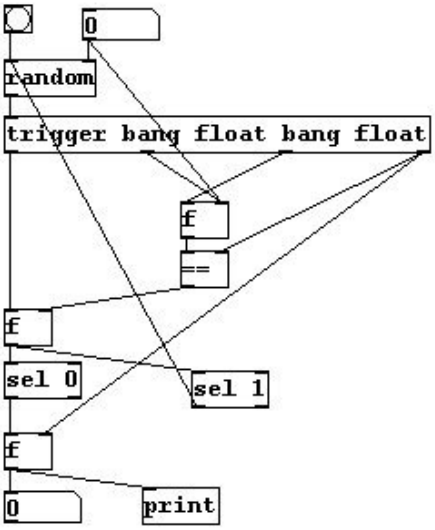
Este contador repasa una serie que se extiende de 0 a 6; luego del 6, recomienza desde 0.



2.2.1.2.7 Aleatoriedad sin repetición

Si ha entendido todo hasta aquí, debería ser capaz de llevar a cabo el siguiente desafío -pero queda advertido, no es sencillo:

Cree un patch que genere números aleatorios donde el mismo número no ocurra dos veces de manera consecutiva (al contrario de lo que ocurre con el Objeto "random" normal). Cuando haya terminado, compare su patch con la solución. ¡Buena suerte!



2.2.1.2.8 Más ejercicios

a) Cree dos melodías aleatorias que se ejecuten simultáneamente.

b) Cree un partch donde dos bangs seleccionen dos intervalos diferentes de su elección (como el ejemplo de *dos bangs/dos frecuencias*).

c) Use "expr" para calcular funciones exponenciales, por ej.: $y = x^2$, $y = x^{(2+x)}$; ó $y = 1 - (2^x)$.

2.2.1.3 Apéndice

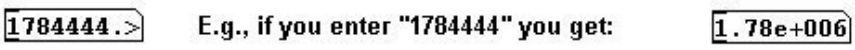
2.2.1.3.1 Entrada de bang

Un bang representa a un clic de ratón. Puede hace un clic sobre el mismo y transmitirlo; es decir, puede recibir un clic como entrada y luego enviar un clic como salida. Sin embargo, esta entrada no tiene que ser necesariamente un clic. El Objeto "bang" convierte cualquier entrada de control que recibe, en bang. Por ejemplo, podría usar un número:

2.2.1.3.2 Cómo se representan los números

Los números con muchos lugares decimales no pueden ser leidos por completo en una caja Número normal. Sin embargo puede alargar la caja Número, clic derecho sobre la misma, seleccione "Properties", ingrese un valor para "width" (ancho), y luego clic en "Ok".

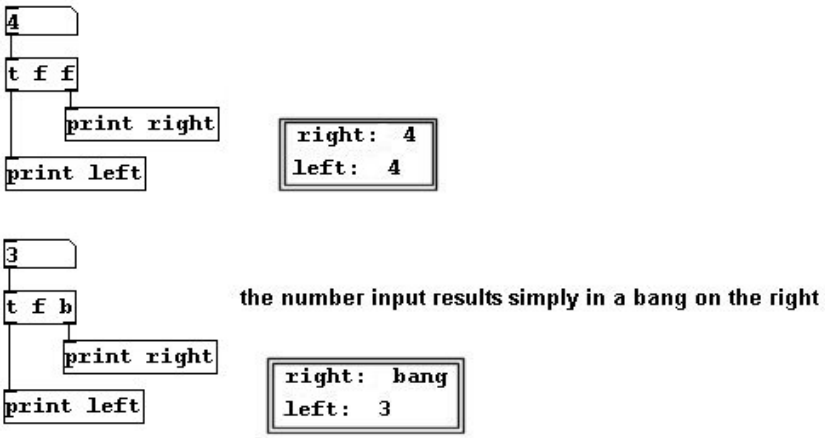
Otro aspecto importante relacionado a los números mayores a 999999. Los mismos son representados de forma simplificada, a saber como producto (con un máximo de dos lugares decimales) de 1000000. El número 1000000 es representado como "e+006".



Lo mismo corresponde para números menores a -999999 y para aquellos entre 1 y -1 con más de cuatro lugares decimales.

2.2.1.3.3 Más sobre trigger

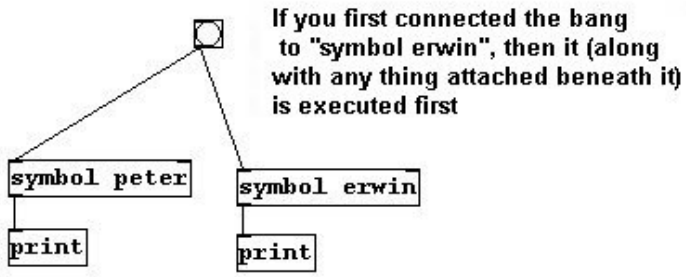
El Objeto "trigger" es capaz de distribuir no sólo bangs sino también números (posteriormente aprenderemos aún más posibilidades). Generalmente es abreviado como "t" y en vez de ingresar los argumentos como "bang" y "float", puede usar sólo "b" y "f":



2.2.1.4 Para los interesados, especialmente

2.2.1.4.1 Sobre la secuencialidad

Por defecto, los Objetos y conexiones son (actualmente) procesados en la secuencia (en tiempo) en que fueron creados:



¡Por supuesto no puede ser deducido desde la vista gráfica y por lo tanto debe ser evitado!

2.2.1.4.2 Con respecto a float

"f" significa "floating point" (punto flotante). En términos precisos, indica un número que posee lugares decimales por lo tanto no estamos en presencia de un número entero. Si quiere trabajar en Pd con números enteros, puede utilizar el Objeto "int" (abreviado: "i") en vez de "float". Al contrario de lo que ocurre en Max/MSP, Pd trabaja con números decimales por defecto.

2.2.2 Diferentes tipos de datos

2.2.2.1 Teoría

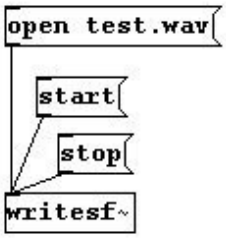
2.2.2.1.1 Bang – un objeto de Interfaz Gráfica de Usuario

Un "bang" se presenta mediante la combinación de letras b-a-n-g. Las combinaciones de letras, llamadas *symbols (símbolos)*, son la segunda forma de datos que Pd usa (además de los números). Algunos Objetos reconocen ciertas palabras y trabajan con las mismas como entradas. Muchos objetos reaccionan al símbolo "bang". Dado que este se utiliza frecuentemente, existe una representación gráfica especial para el "bang", un cículo que se oscurece cuando se activa (**Put** > **Bang**). A esto se lo llama objeto "GUI" (GUI = graphical user interface (interfaz gráfica de usuario), es decir, una representación gráfica de algo y/o un gráfico que puede ser cambiado para producir y enviar nuevos valores).

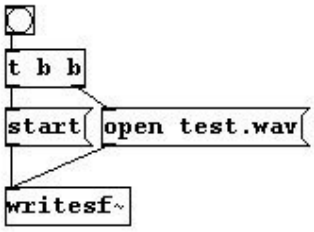
2.2.2.1.2 Mensajes

En este contexto, analicemos el Objeto "writesf~" (introducimos aquí un Objeto de audio por el hecho de que los símbolos son frecuentemente utilizados con dichos Objetos; "writesf~" será explicado con detalles en el capítulo dedicado al audio). Este Objeto almacena sonido en un archivo

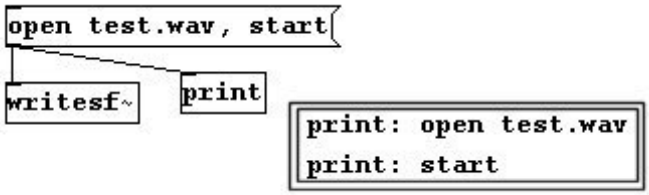
WAV. Funciona de la siguiente manera: primero determinamos en una caja Mensaje el *nombre de archivo* en el cual el sonido será guardado en formato WAV: "open [nombre del archivo]". Por ejemplo, si desea llamar al archivo "test.wav", entonces deberá ingresar "open test.wav". Luego usamos los Mensajes "start" y "stop" para comenzar y finalizar la grabación.



Generalmente selecciona un nombre y luego comienza la grabación. Aquí el orden es importante -antes que el Objeto "writesf~" pueda comenzar a grabar, tiene que saber previamente cuál es el nombre del archivo sobre el cual tiene que almacenar los datos. Esto puede ser resuelto usando un "trigger":

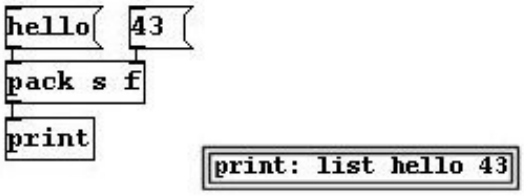


Pero los Mensajes también pueden ser enviados uno detrás del otro escribiéndolos en la misma caja Mensaje, separados por una coma:



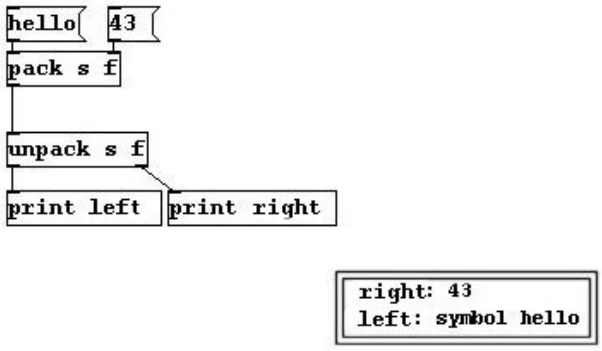
2.2.2.1.3 Listas

El Mensaje "open test.wav" es en realidad la conjunción de dos símbolos (porque consiste de dos palabras separadas por un espacio). Llamamos "lista" a este tipo de sucesión de dos o más símbolos (o números). El Objeto "pack" puede crear una lista a partir de varios "elementos". Como argumentos, ingrese las indicaciones que especifiquen qué tipo de elementos debe contener la lista. Un número, como ocurre en "trigger", es expresado con "float"* (ó "f"), un símbolo con "symbol" (ó "s"). Si quiere crear una lista que contenga los Mensajes "hello" y "43", use el Objeto "pack" como se muestra a continuación:



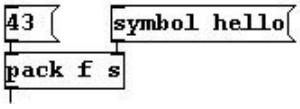
Una vez más: sólo cuando la entrada extrema izquierda recibe un ingreso, obtenemos una salida

(precisa). (Si primero ingresa el Mensaje "hello" en "pack", sin haber ingresado previamente "43", obtendremos como resultado "list hello 0".) Hasta este punto, la salida del Objeto "pack" pueede ser vista con un Objeto "print". Este muestra "list hello 43". Sin embargo, los elementos de esta lista pueden ser devueltos usando el Objeto inverso (Pd posee muchos Objetos inversos) "unpack", el cual trabaja de acuerdo al mismo principio que "pack", excepto que lo que aparece aquçi como salidas corresponde a las entradas del Objeto "pack".

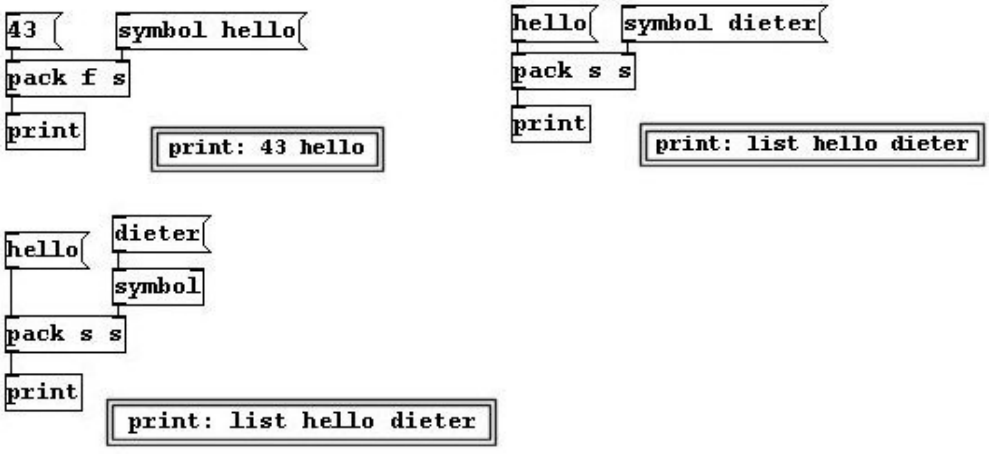


Ahora "print" muestra "43" y "symbol hello". Todo aquello que no corresponda a un número es precedido por una indicación (llamada "selector") de su tipo de dato.

También debemos aclarar: si usa un símbolo en cualquier entrada de un Objeto "pack" que no sea la extrema izquierda, debe aparecer de esta manera:



Un problema con "pack s s": la primer entrada es la única que no tiene que ser etiquetada específicamente como un símbolo. El segundo símbolo debe o bien ser precedido por la palabra "symbol" en la caja Mensaje, o bien el Mensaje debe ser convertido usando un Objeto "symbol":



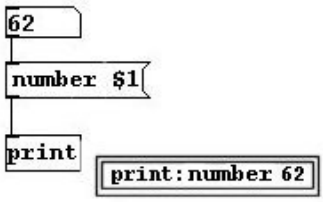
Un último aspecto sobre las listas: una lista que comience con un número no necesita ser explícitamente etiquetada como lista; sin embargo, si comienza con un símbolo, debemos usar la palabra "list".

2.2.2.1.4 Mensajes con variables

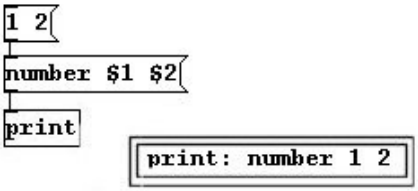
Veamos más de cerca las cajas de Mensaje:

Podemos integrar variables en el contenido de una caja de Mensaje. Esto se realiza de forma similar

al aprendido con "expr", pero con algunas diferencias: primero, las variables son simplemente llamadas "$1, $2", etc. Si ingresa un número como entrada para el Mensaje "number $1", la salida del mismo corresponderá a la expresión completa, esto es, incluido el número ingresado.



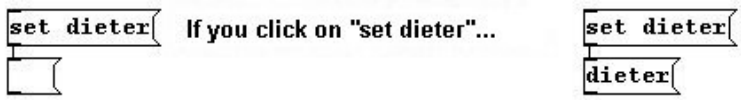
El uso de varias variables -por ejemplo, "number $1 $2"- no crea un número correspondiente de entradas (como sí lo hace "expr"), sino que mantiene una única entrada. Necesita ingresar una lista de números en el mismo:



Los símbolos deben ser identificados como tales:

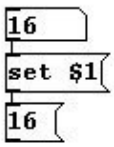
2.2.2.1.5 Mensajes: Set

También puede redefinir por completo los contenidos de la caja Mensaje gracias al símbolo "set":



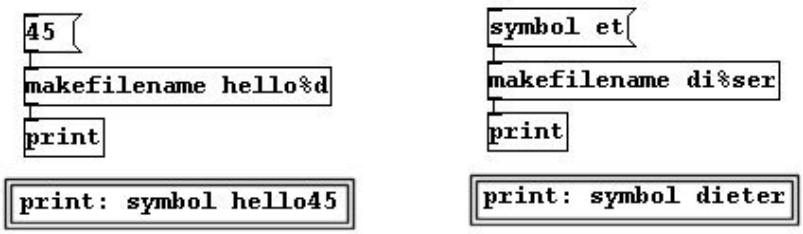
Este símbolo cambia los contenidos de una caja Mensaje en Execute Mode (cf. el último punto en 2.1.3).

Por ejemplo, si utiliza una variable podría convertir la salida de una caja Número en un Mensaje:



2.2.2.1.6 Makefilename

Normalmente no es posible incluir una variable sin un espacio que la separe. Sin embargo, el Objeto "makefilename" lo hace posible. Las variables que pueden ser incluidas in los argumentos son "%d" para dígitos y "%s" para símbolos:



2.2.2.1.7 Openpanel

El Objeto "readsf~" reproduce un archivo de sonido existente, por ejemplo, uno que esté almacenado en un disco duro. Necesita el Mensaje "open [nombre del archivo de sonido]". "nombre del archivo de sonido" se refiere al lugar donde el archivo se encuentra grabado en un dispositivo de almacenamiento de datos. Si quiere usar "readsf~" en un patch que se encuentra salvado en el directorio *c:/Pd/Pd-patches/* para ejecutar un archivo de sonido llamado "hallo.wav" que también se encuentra guardado en el mismo directorio, sólo debe ingresar "open hallo.wav". Sin embargo, si "hallo.wav" se encuentra alojado en el directorio *c:/Pd/*, debe ingresar "../hallo.wav" o si no se encuentra salvado en *c:/Pd/Pd-patches/soundfiles/*, entonces ingrese "/soundfiles/hallo.wav". En el caso que se encuentre en *c:/soundfiles/* use "open ../../soundfiles/hallo.wav". O si se encuentra en otro disco, por ejemplo, *d:/soundfiles*, entonces ingrese "open d:/soundfiles/hallo.wav".

A veces, cuando trabajamos con estos complicados nombre de ruta de directorio, puede resultarnos más facil expresarlos usando "openpanel". Cuando este recibe un bang se abre una ventana con los contenidos disponibles de todos los discos presentes en el ordenados. Cuando realiza un doble clic en un archivo, "openpanel" ingresa en Pd toda la ruta necesaria para el acceso del archivo (como un símbolo):

Si un patch todavía no ha sido salvado, Pd (bajo Windows) asume que la ruta corresponde a *pd/bin*.

2.2.2.1.8 Almacenamiento simple de datos

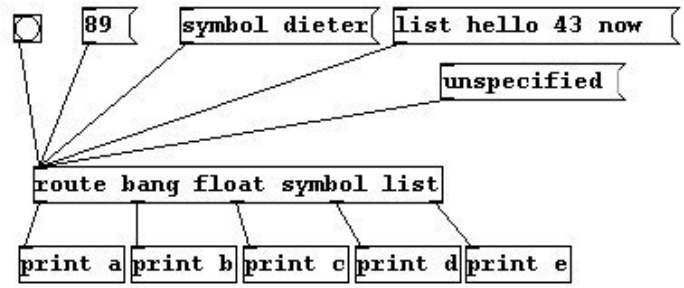
Como ya hemos explicado en relación al Objeto "float" (2.2.1.1.5), los datos pueden ser salvados dentro de un patch usando los Objetos "float", "symbol", y "lister" (pero se pierden cuando cerramos el patch). "float" y "lister" son generalmente abreviados como "f" y "l".

La entrada derecha recibe un número, un símbolo, o una lista que será almacenado en el objeto . Estos datos almacenados son enviados como salida cuando el Objeto recibe un bang en su entrada izquierda.

Un número, símbolo o lista también pueden ser enviados directamente en la entrada izquierda; son entonces directamente dirigidos a la salida (y también son almacenados en el Objeto mismo).

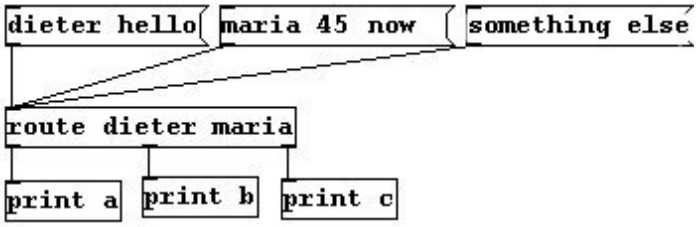
2.2.2.1.9 Route

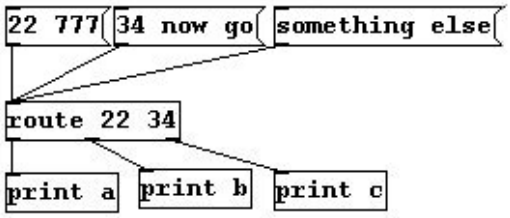
El Objeto "route" puede ser usado para clasificar varios tipos de datos. También puede distribuir los tipos de datos (número, símbolo, lista, bang)...

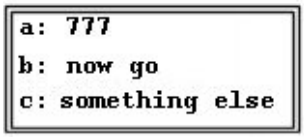


(Todo lo que no pueda ser distribuido es enviado a la salida extrema derecha.)

... como así también ordenar listas de acuerdo a nombres que usted haya definido:





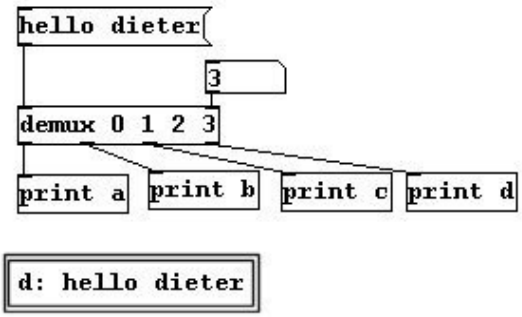


Los números y símbolos no pueden ser combinados aquí. Por ejemplo, "route 22 dieter" no funcionará.

2.2.2.1.10 Demultiplex

El Objeto "route" distribuye un ingreso a varias salidas de acuerdo a un prefijo. El Objeto "demultiplex" (o "demux", ambos en la versió Pd-extended) distribuye un ingreso a varias salidas de acuerdo al ingreso de otra entrada. Primero "demux" recibe los números de las salidas como argumento, comenzando por 0: "demux 0 1 2 3".

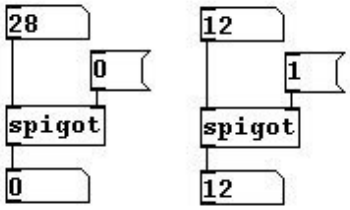
En este ejemplo, existen dos entradas ("demux" siempre posee sólo dos entradas) y cuatro salidas (una para cada uno de los cuatro argumentos). Ingrese un número en la entrada derecha que corresponda a un número de una salida. Ahora, cualquier cosa que ingrese (número, símbolo, o lista) en la entrada izquierda, saldrá por la salida que corresponda al número ingresado en la entrada derecha:



Observe que, con frecuencia, Pd comienza a contar no con 1, sino con 0.

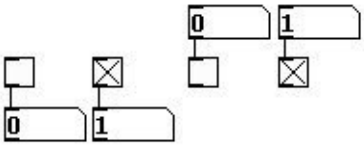
2.2.2.1.11 Spigot

Otro Objeto importante es "spigot". Dependiendo si su entrada derecha ingresa un 0 o un 1, "spigot" no permitirá o sí permitirá el paso de un ingreso de la entrada izquierda -como una compuerta que se encuentra cerrada o abierta.



2.2.2.1.12 Toggle

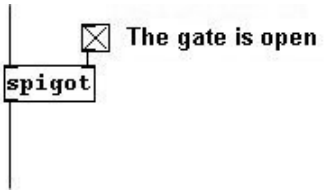
Como ya habrá observado con "spigot", "= =", y otras evaluaciones relacionales, 0 y 1 se presentan con frecuencia en Pd. Dada esta frecuencia -y de manera similar a bang- existe un Objeto gráfico que permite cambiar de 0 a 1 y viceversa, llamado "toggle" (**Put** > **Toggle** o **Shift-Ctrl-T**).



"toggle" parece un conmutador de encendido/apagado y a menudo puede ser pensado como tal. Pero siempre debe recordar que el ordenador siempre lo interpreta simplemente como un cambio

entre 0 y 1.

Al relacionar un "toggle" a un "spigot" puede observar más claramente si la "compuerta" se encuentra abierta o cerrada. U observar si una evaluación relacional entrega un resultado positivo o negativo:

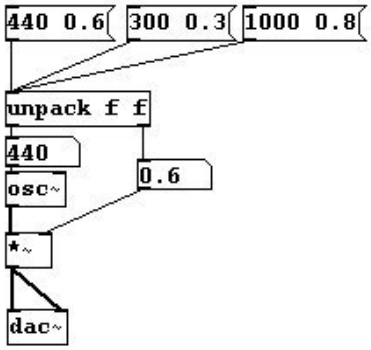


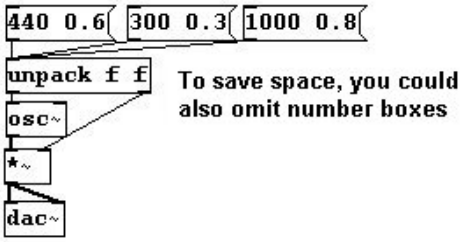
2.2.2.2 Aplicaciones

Veamos cómo trabajan estos conceptos en la práctica:

2.2.2.2.1 Usar listas con alturas y dinámicas

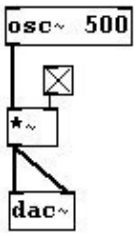
Usar una lista para asignar alturas acopladas con dinámicas, a un oscilador:





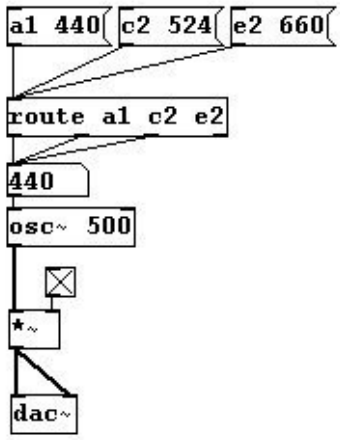
2.2.2.2.2 Conmutador encendido/apagado

En el primer ejemplo, vimos que un tono podría ser encendido o apagado usando "1" y "0". Podría usar un "toggle" para esto según lo siguiente:



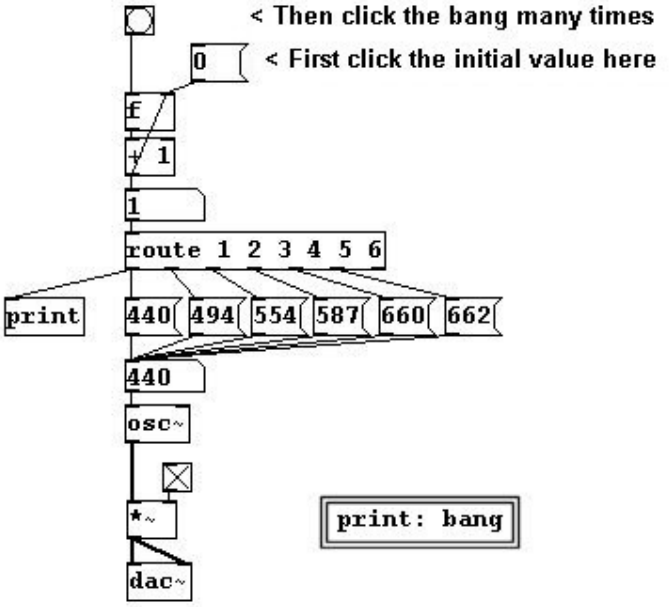
2.2.2.2.3 Alturas con nombres

Para asignar alturas con nombres (elegidos libremente) a un oscilador:



2.2.2.2.4 Una secuencia simple

Aquí tenemos un contador que envía al oscilador una altura concreta cada vez que recibe un bang:

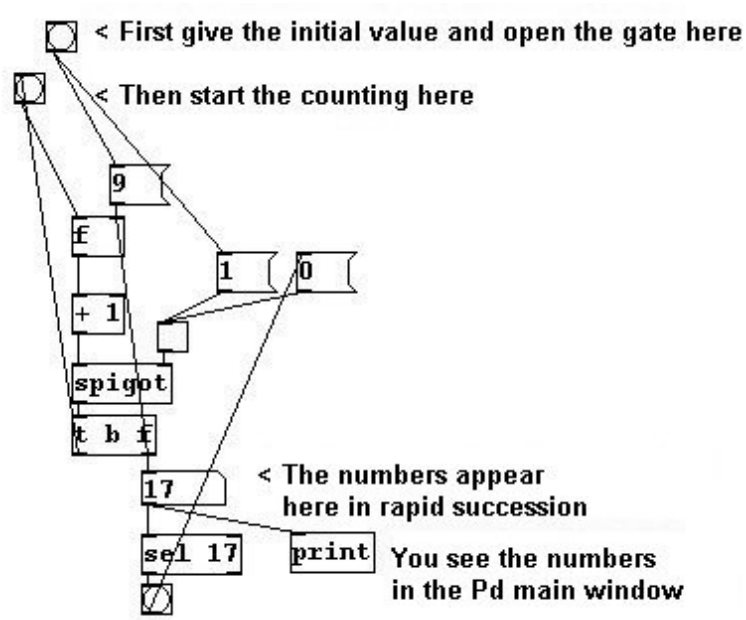


Cuando, en vez de una lista, "route" recibe un valor de entrada equivalente a uno de sus argumentos, envía un bang a la salida correspondiente. En este ejemplo, "route" funciona como una combinación de varios selectores (otra posibilidad sería relacionar una serie de Objetos "sel" al contador: "sel 1", "sel 2", etc.).

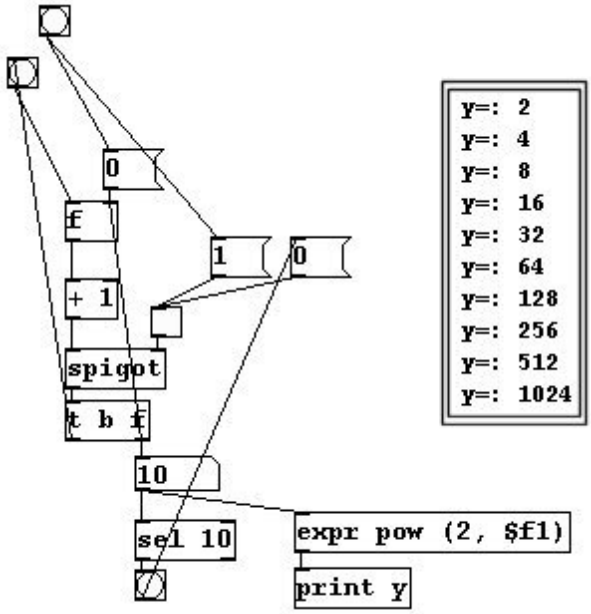
La salida extrema derecha del Objeto "route" no necesita ser conectada a nada ya que la entrada siempre corresponde a los argumentos del Objeto "route".

2.2.2.2.5 Un contador limitado

Aquí tenemos un contador que comienza en 10 y se detiene en 17:



Este podría ser util para, digamos, calcular rápidamente valores para una función matemática dentro de un rango dado. Este ejemplo muestra la simple función cuadrática $y = 2^x$ para el rango de 1 a 10:



Para recursiones (donde una salida es retroalimentada como entrada) como estas, debe ser muy cauteloso para evitar un bucle infinito. Si reanudamos este patch luego de que ya haya sido ejecutado una vez sin reingresar el valor inicial sino sólo abriendo la compuerta y comenzando el cálculo nuevamente, comenzará por arriba del valor diez y continuará contando por siempre (ya que el Objeto "sel 10", el cual finaliza el cálculo, nunca tendrá lugar).

2.2.2.2.6 Más ejercicios

a) Crear una secuencia de listas con alturas y dinámicas.

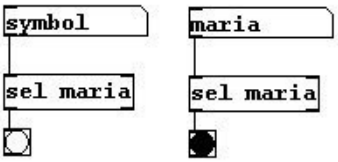
b) Crear un patch que le permita usar una lista de dos números, los cuales representan el primero y

último valor en un rango de *x*, para calcular valores de *y* en una ecuación -por ejemplo, valores para la función $y = 3^x$ desde $x = -2$ hasta $x = 4$.

2.2.2.3 Apéndice

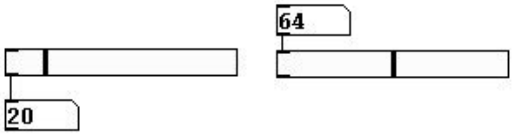
2.2.2.3.1 Cajas Símbolo

Las cajas Símbolos funcionan de manera análoga a las cajas Número (pero son raramente usadas en Pd). Por ejemplo, "sel" también podría ser usado con símbolos:



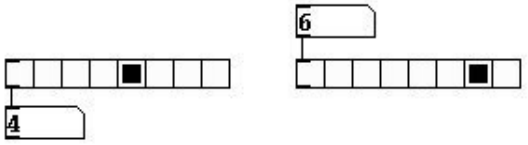
2.2.2.3.2 Slider

Existen otros dos Objetos GUI en el nivel de control: *slider* y *radio*. *Slider* (deslizador) (**Put > HSlider** o **VSlider** o sus atajos) es una representación gráfica de una caja Número. Sin embargo se encuentra restringido a un rango (con una configuración por defecto de 0 a 127):



2.2.2.3.3 Radio

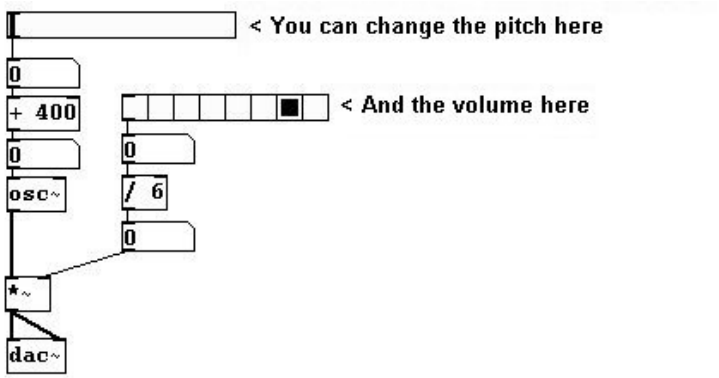
Radio (caja de opciones) (**Put** > **Hradio** o **Vradio**) es también una representación gráfica de una caja Número pero extremadamente limitada: sólo unos pocos números (por defecto de 0 a 7) pueden ser enviados a la salida, lo cual es logrado simplemente gracias a un clic sobre alguna de las celdas.



Sliders y *radios* pueden ser horizontales o verticales; esta es sólo una diferencia en la apariencia y no afecta su funcionamiento.

2.2.2.3.4 Usar slider y radio

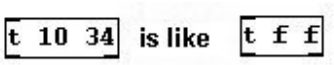
Varias altura pueden ser seleccionadas con un *slider* y varios niveles dinámicos con un *radio*:



Esto crea una interfaz visualmente clara para cambiar los parámetros de un patch. Es especialmente útil para usar en performances en un escenario en vivo .

2.2.2.4 Para los interesados, especialmente: Otras especificaciones de tipo y más sobre cajas

En Pd, una especificación "float" puede (por ejemplo, como "trigger") muchas veces ser expresada con un número en vez de hacerlo con "f" (el valor puede a veces jugar un rol, pero no siempre -por ejemplo, podría ser válido con "f" o "pack" pero no con "t"):



Sin embargo, como esto perjudica la claridad, no recomendamos el uso de números.

Algunas observaciones con respecto a las cajas: 1. Estrictamente hablando, todas las cajas son Objetos que pueden senviar y recibir mensajes como así también reaccionar a estos mensajes de acuerdo a sus (las cajas) características. 2. Las conexiones muestran qué Objeto envía mensajes a otro Objeto. Si una salida de Objeto es conectada a entradas de diversos Objetos, entonces todos estos Objetos reciben el mensaje. El orden no es definido (intencionalmente). 3. Existen Objetos GUI que crean y envían mensajes basadas en la interacción del usuario. Ejemplos de Objetos GUI: bang, toggle, slider, y canvas.

2.2.3 Operaciones en el tiempo

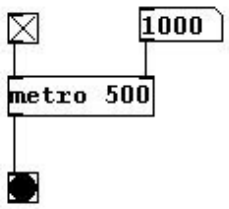
La música, como es sabido, ocurre en el tiempo. Entonces resulta fundamental que un lenguaje de programación de audio tenga la capacidad de controlar la secuencia cronológica (es decir, que las duraciones/ritmos y secuencias de eventos puedan ser creados).

2.2.3.1 Teoría

2.2.3.1.1 Metro

El primer Objeto básico para controlar la secuencia cronológica se llama "metro". Como el nombre indica, se trata de un metrónomo. Cuando lo enciende o lo apaga (usando 1/0 o un toggle en la entrada izquierda), obtenemos bangs que a un intervalo regular determinado por el argumento o la

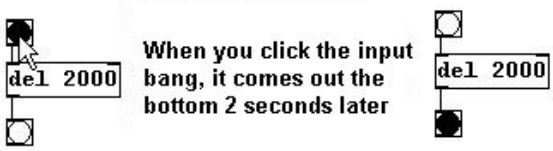
entrada derecha.



El tempo se configura en **milisegundos** (ms). Si quiere enviar un bang una vez por segundo, ingrese "metro 1000", "metro 2000" para un bang cada dos segundos, "metro 500" para un bang cada ½ segundo (equivalente a negra = 120).

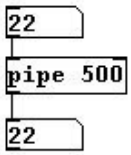
2.2.3.1.2 Delay

"delay" ("del") retarda un bang entrante la cantidad de milisegundos especificada en el argumento o en la entrada derecha:

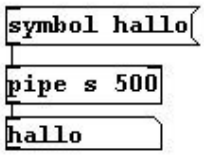


2.2.3.1.3 Pipe

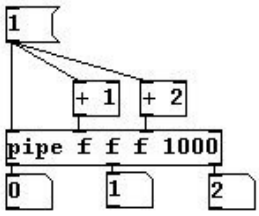
"pipe" cumple la misma función que "delay" pero retrasa números y símbolos. La duración del retardo es ingresada como argumento. Por defecto, "pipe" espera un número como entrada.



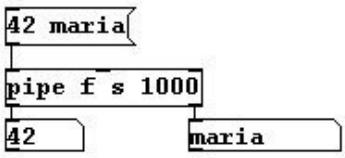
Si quiere enviar un símbolo por medio de este, primero se debe declararlo como argumento (con "s", como en "route"). Segundo, a continuación debe determinar la duración (como argumento a como entrada derecha):



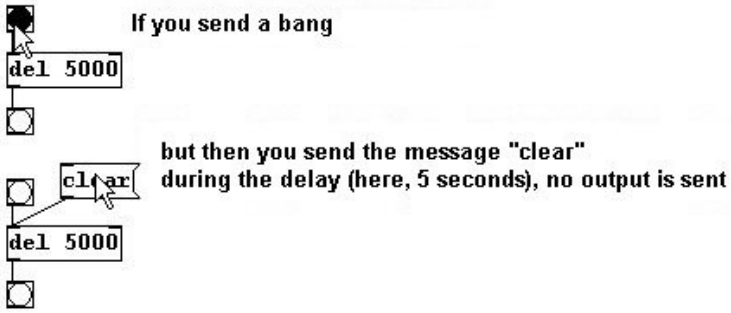
También "pipe" -como "pack"/"unpack"- puede tener varias entradas y salidas:



"pipe" manipula listas como "route":

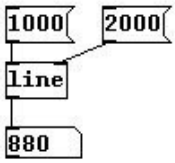


Si una entrada se encuentra 'esperando' en un Objeto "del" o "pipe" , esta puede ser eliminada antes de ser enviada mediante el uso de los Mensajes "clear" or "stop":

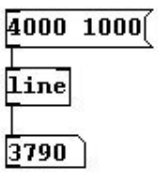


2.2.3.1.4 Line

Con "line", puede crear una serie de números en el tiempo. En otras palabras, puede ordenar al programa que comience a contar dentro de un rango limitado, los valores de inicio y finalización que usted determine. Generalmente "line" no continen argumentos. La entrada derecha corresponde a la *duración* de la serie de números (por defecto 0). La entrada izquierda corresponde al *valor de destino* (por defecto 0; puede ser ingresado como argumento).

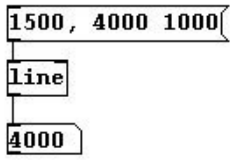


Si ingresa un nuevo valor de destino en la izquierda, Pd salta inmediatamente hacia ese valor. Esto ocurre así porque el valor de la entrada derecha de reinicializa automáticamente a 0, por lo tanto debe ser reingresado (esta es una excepción en Pd; normalmente los Objetos de Pd almacenan los ingresos de las entradas "pasivas" hasta el momento de su reinicialización. Alternativamente, puede ingresar ambos valores (*valor de destino* y *duración*) como una lista:



Si usted hace un clic sobre la caja Mensaje, no ocurre nada ya que "line" ha llegado a 4000 por lo tanto se mantiene allí. Si ingresa en la lista un nuevo *valor de destino* -por ejemplo, 50- "line" cuenta atrás desde 4000 hasta 50 (en 1000 ms).

Si quiere comenzar con un número particular y contar hacia otro número dentro de un cierto marco de tiempo, primero debe ingresar un valor en la entrada izquierda (sin haber ingresado nada en la entrada derecha). Esto provocará que "line" salte al *valor de origen* (con una sóla caja Mensaje); luego puede ingresar la lista. Como mencionamos en 2.2.2.1.2, puede incluir varios mensajes en una sóla caja Mensajes, siempre y cuando los separe mediante comas, como el ejemplo siguiente:

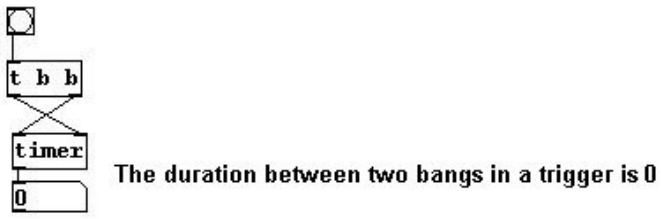


En este ejemplo, cada vez que realiza un clic en la caja Mensaje, "line" cuenta de 1500 a 4000 en 1000 ms.

2.2.3.1.5 Timer

El Objeto "timer" cumple un funcionamiento similar a un cronómetro. El tiempo medido se inicia siempre cuando se introduce un bang en la entrada izquierda (el cual, por supuesto, debe ingresar primero) y obtenemos el resultado como salida cuando ingresamos un bang en la entrada derecha (en ms):

Aquí puede observar que las operaciones realizadas con "trigger" no involucran una expedición de tiempo por parte del ordenador, incluso aunque ocurran una a continuación de la otra:



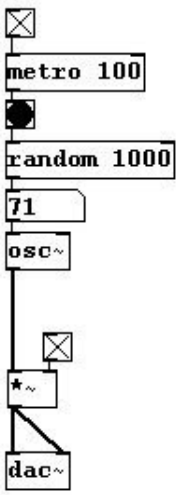
(cf. 2.2.2.2.5)

El Objeto "timer" es (innecesariamente de alguna manera) una excepción a la regla de Pd en donde las entradas siempre deben darse de derecha a izquierda.

2.2.3.2 Aplicaciones

2.2.3.2.1 Melodía aleatoria automática

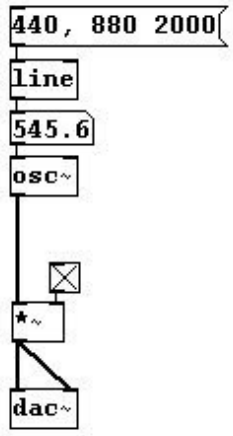
Ahora estamos en condiciones de llevar a cabo configuraciones musicales algo más complejas. Por ejemplo, una rápida melodía aleatoria que corre automáticamente:



¡Intente modificar la velocidad del metrónomo del ejemplo anterior con una caja Número en la entrada derecha de "metro"!

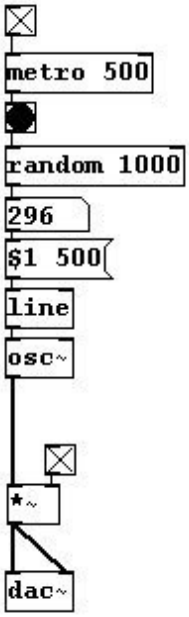
2.2.3.2.2 Glissando

También puede crear un glissando:



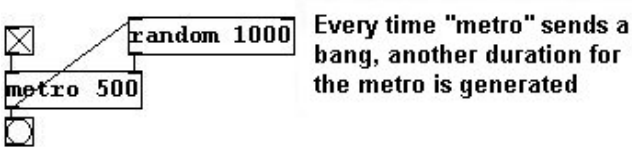
2.2.3.2.3 Melodía de glissando

O combinar los últimos dos patches para crear una melodía de glissando aleatoria:

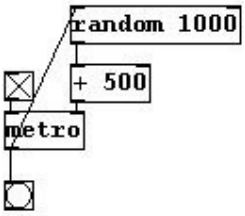


2.2.3.2.4 Ritmos irregulares aleatorios

También puede crear ritmos irregulares basados en una selección aleatoria:



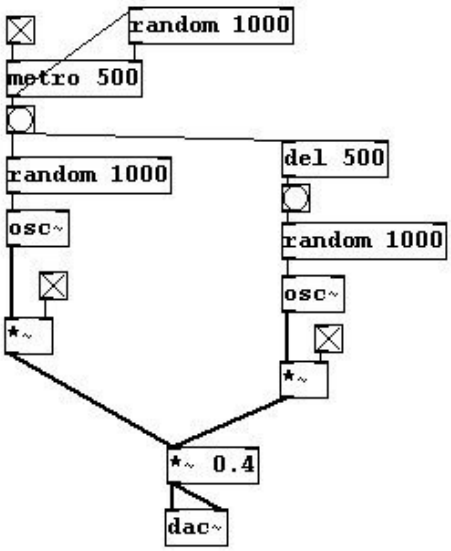
En este ejemplo, los bangs del Objeto "metro" son enviados a un intervalo de entre 0 y 999 ms (seleccionados aleatoriamente). Si quiere, digamos, duraciones de entre 500 y 1500 ms, sólo tiene que agregar una simple adición:



Una adición a una operación matemática de este tipo (aquí "+ 500") se la llama "offset" (desplazamiento).

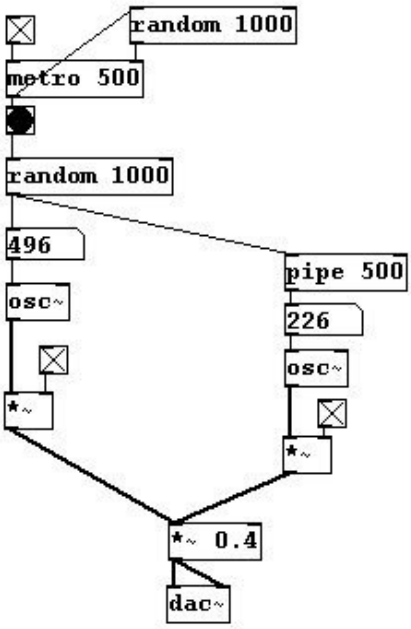
2.2.3.2.5 Cánones

Estos ritmos pueden luego ser conectados a un generador aleatorio y transferidos a otro oscilador para realizar un canon rítmico:



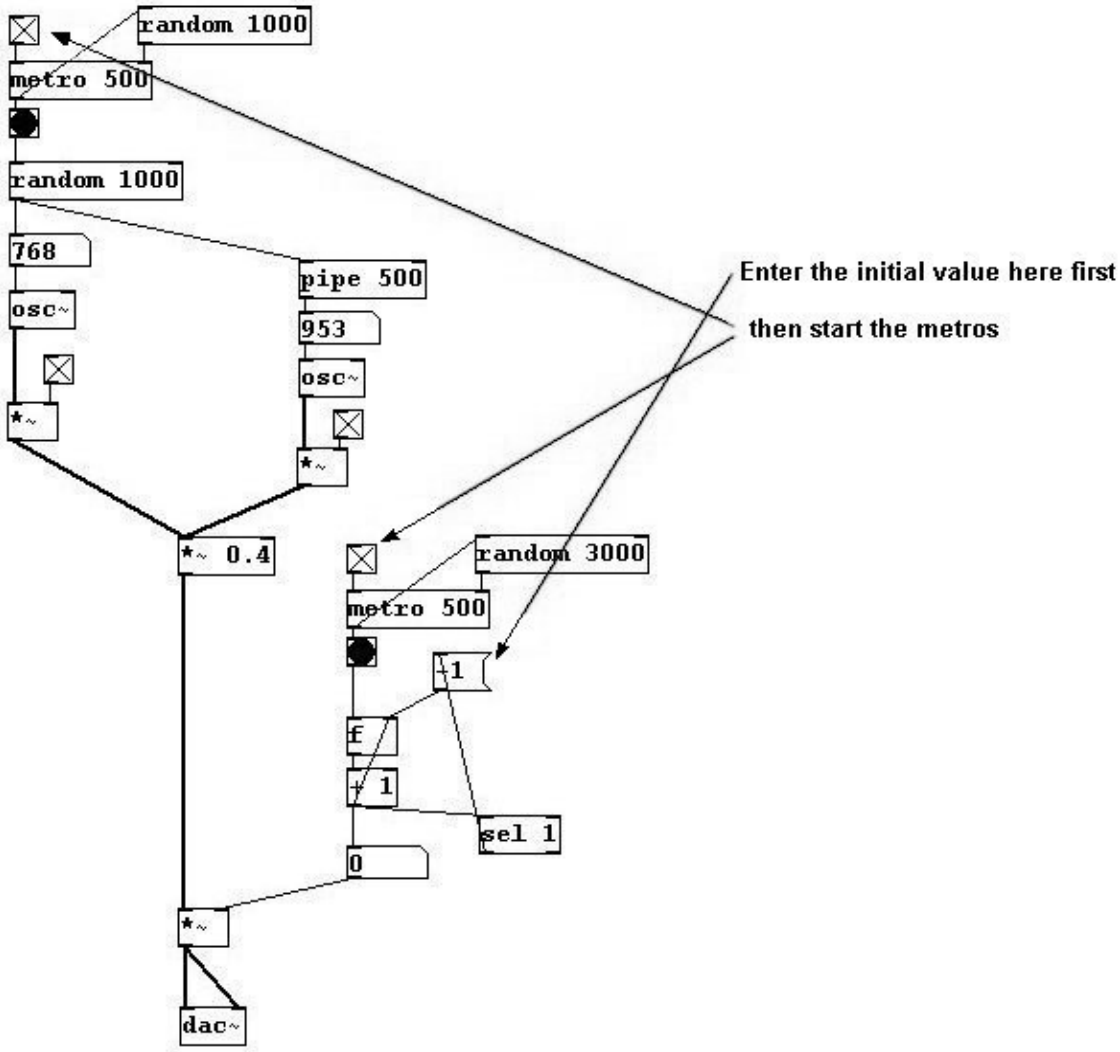
(El "*~ 0.4" será explicado más adelante.)

O puede hacer un canon verdadero:



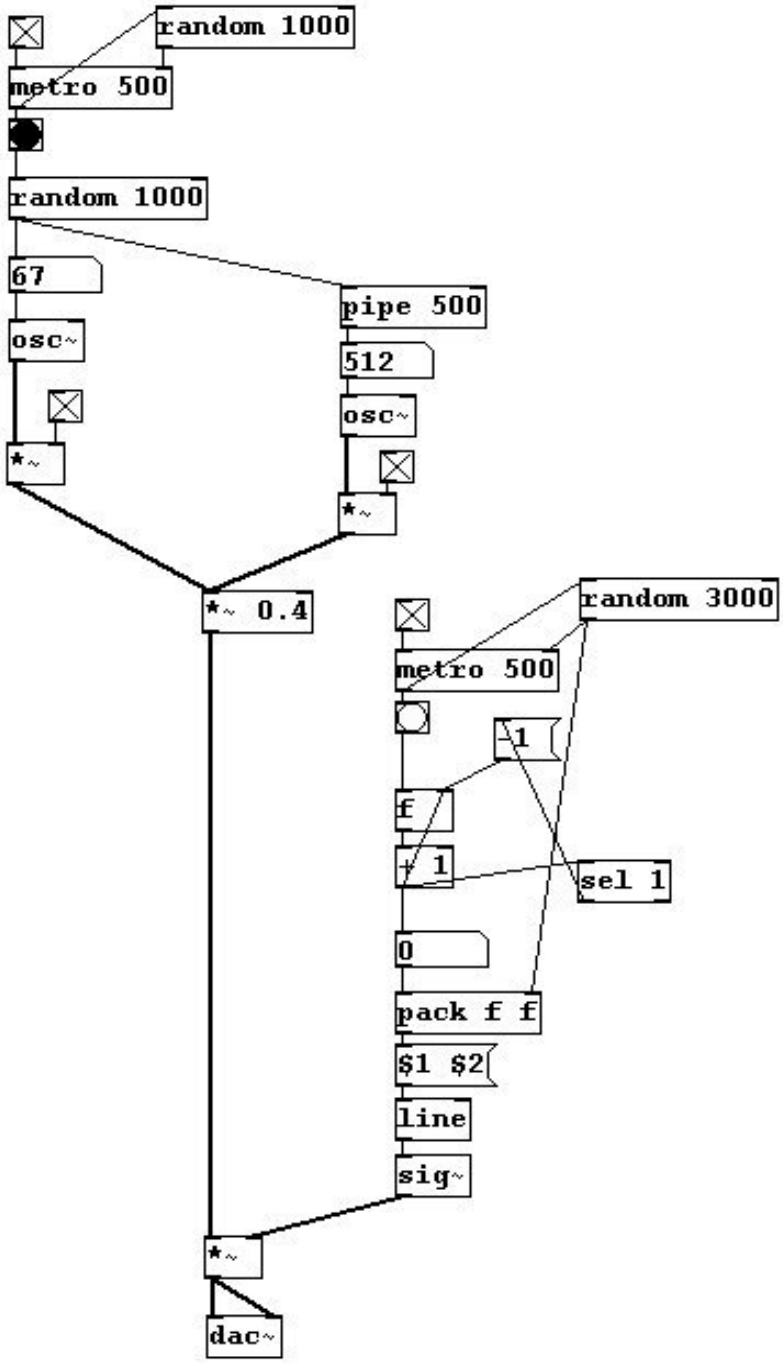
2.2.3.2.6 Silencios

También puede incluir silencios automáticos:



2.2.3.2.7 Crescendo/Decrescendo

O un crescendo y decrescendo ("sig~" también será explicado más adelante):



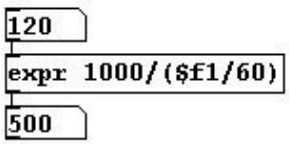
Aquí puede ver nuevamente que, por el momento, Pd sólo usa números para los cálculos. Un crescendo utiliza una serie de número tal como lo hace un glissando. Podría tanbién decir: un crescendo equivale a un glissando dinámico.

2.2.3.2.8 Metrónomo

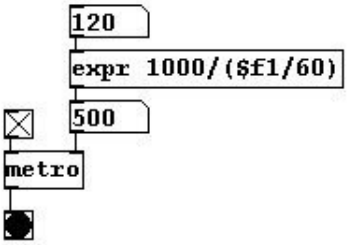
Podría construir un metrónomo como el siguiente:

Primero, realicemos un modelo visual funcional, es decir, uno en donde la señal del metrónomo crea un bang visible. Las marcas de un metrónomo están dadas en pulsos por minuto, tal como encontramos en una partitura musical: negra = 60, negra = 100 etc.

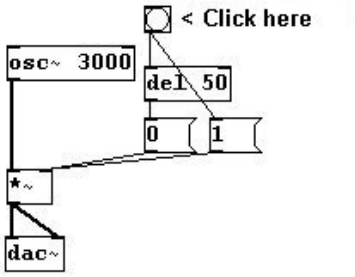
Entonces debe convertir ppm (pulsos por minuto) en milisegundos:



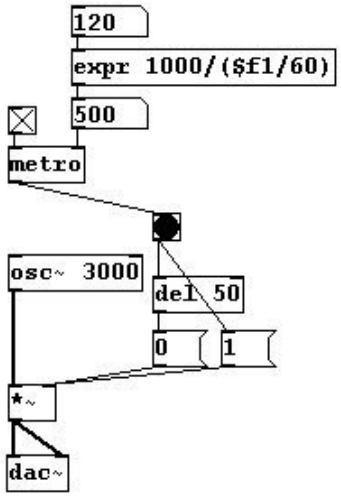
Ahora use este resultado como entrada para un metrónomo.



Ahora no sólo queremos ver los impulsos como un bang, sino también oirlos. Usemos el patch de sonido anterior y configurelo para que se escuche un tono corto con cada bang, Puede crearlo así:



Combinamos los dos últimos:



Una vez realizadas las conexiones, tenemos el metrónomo finalizado. Más adelante aprenderá cómo incorporar una señal de sonido alternativa.

2.2.3.2.9 Más ejercicios

a) Cree una melodía aleatoria que salte al próximo tono dos veces por segundo (otra alternativa: con

un glissando).

b) Cree un metrónomo con ritmos irregulares aleatorios (con un tempo promedio ajustable).

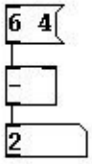
c) Cree un metrónomo que pulse cinco veces en tempo negra = 60 y cinco veces en tempo negra = 100.

d) Cree una melodía aleatoria que cambie cada dos segundos desde un registro bastante agudo hasta un registro bastante bajo.

2.2.3.3 Apéndice

2.2.3.3.1 Distribución de listas

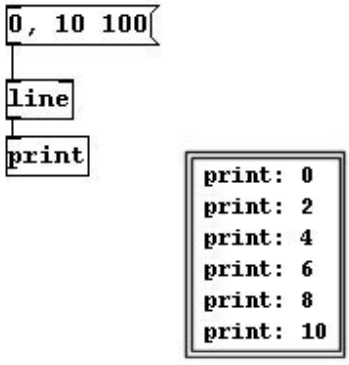
Como observó en "line", en Pd puede ingresar una lista en la entrada extrema izquierda de un Objeto que posee varias entradas en lugar de conectar algo a todas las entradas del Objeto (sin embargo, existen Objetos en los cuales esto no funcionará). Los elementos de una lista son distribuidos a las entradas de derecha a izquierda:



2.2.3.3.2 Resolución de tiempo para el control de datos

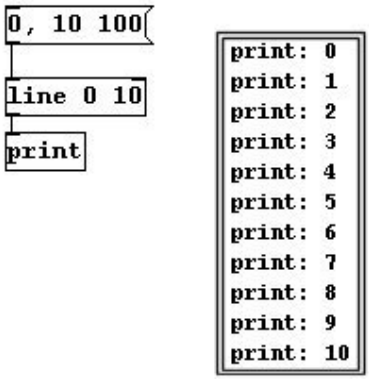
La resolución de tiempo para las tareas en el nivel de control se da en milisegundos.

Sin embargo, esto frecuentemente no configura con antelación. Puede imaginar que un cálculo en milisegundos requiere muchos recursos de procesador (también llamado CPU). Para "line", por ejemplo, la preconfiguración se da en pasos que ocurren en intervalos de 20 ms:



Si quiere que cuente de 0 a 10 en 100 ms, el ordenador ejecuta un paso cada 20 ms; es por esta razón que los números de salida poseen diferencias.

Dicho intervalo (en milisegundos) puede ser ajustado en el Objeto "line" como segundo argumento (el primero entrega el *valor de destino* primario para el proceso de conteo, el cual es eventualmente reemplazado por la entrada):



Debe ser advertido que el resultado sólo será "limpio" mientras la potencia de procesamiento del ordenador sea lo suficientemente alto. De lo contrario, tendremos errores.

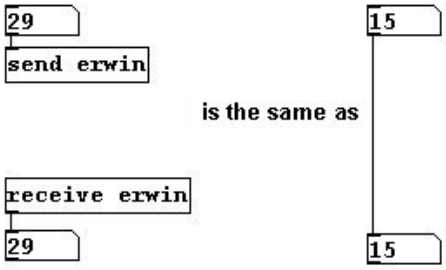
2.2.4 Misceláneos

Para mejorar el "manejo" de Pd, existen varias opciones adicionales.

2.2.4.1 Envío y recepción

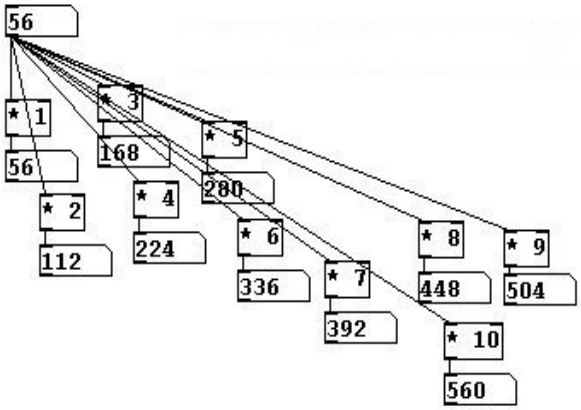
2.2.4.1.1 Send/Receive

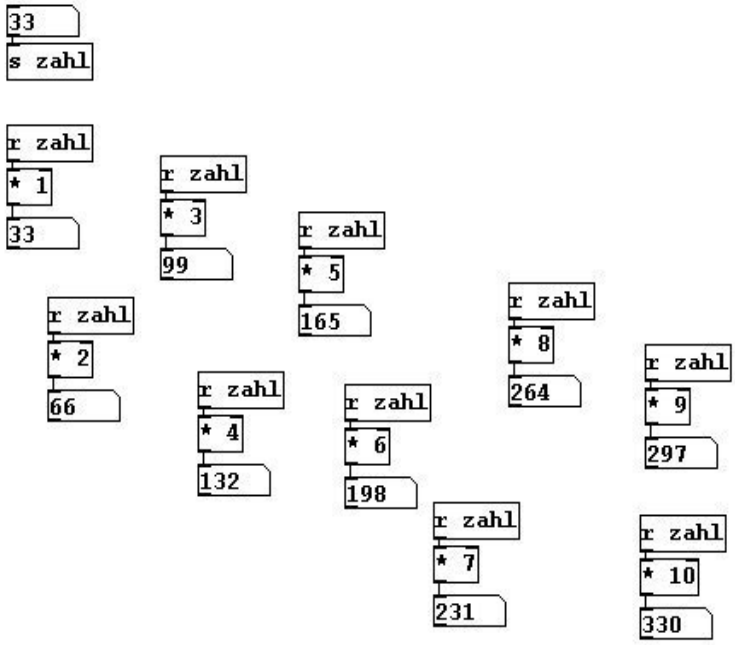
Para evitar la conexión de todas las cajas mediante 'cables,' es posible usar los Objetos "send" y "receive" para enviar y recibir datos.



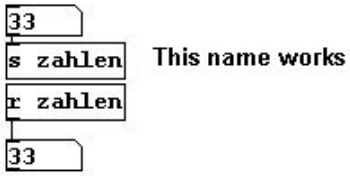
El argumento para un Objeto "send" puede ser un nombre cualquiera. Un Objeto "receive" que poseea un argumento con el mismo nombre que un Objeto "send", recibe su entrada y la envía a través de él.

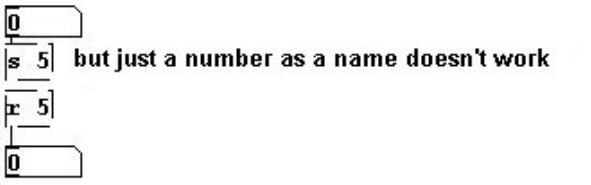
Los Objetos "send"/"receive" (o "s" y "r") son útiles cuando necesita enviar datos numéricos a diferentes locaciones (aunque esto pueda hacer más dificil la comprensión del patch).

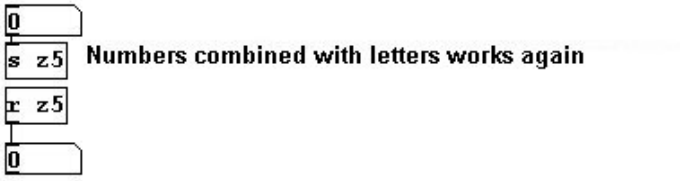




Semejante libertad en la elección de los nombres (regresaremos a esto más tarde) requiere que los mismos siempre sean ingresados sin espacios entre letras; números individuales no están permitidos.

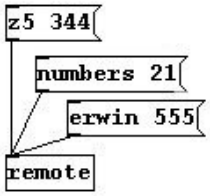


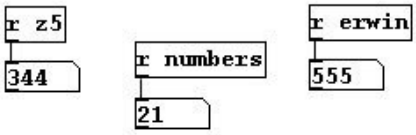




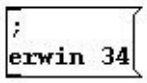
2.2.4.1.2 Envio de listas

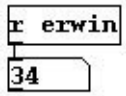
Si tiene diferentes receptores (con nombres diferentes), puede usar "remote" para distribuir Mensajes desde un "punto de distribución" central (similar a "route"). Usted ingresa al Objeto una lista cuyo primer elemento corresponde al nombre del receptor y cuyo segundo elemento es el mensaje mismo. El Objeto "remote" es parte de Pd-extended.





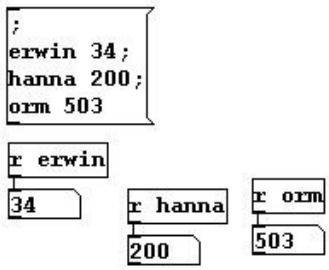
Otra posibilidad disponible consta en preceder la lista con un punto y coma en la caja Mensaje. En este caso necesita un realizar un clic en la caja Mensaje para producir el envío del mismo.





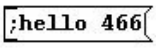
2.2.4.1.3 Una serie de envio de listas

También puede enviar muchos mensajes diferentes en una misma caja Mensaje (con un sólo clic):

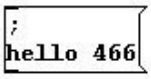


En las cajas Mensaje, existen dos signos de puntuación que tienen una importancia especial: la *coma* (una serie de muchos mensajes) y el *punto y coma* (el cual representa al transmisor).

Note Bien: en las cajas Mensajes, Pd automáticamente salta una linea después del punto y coma. Si escribe lo siguiente:



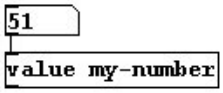
y luego duplica la entrada (**Ctrl-D**) o cierra y reabre el patch, verá esto:

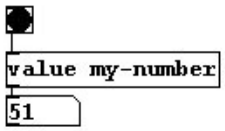


Este también ocurre con los *comentarios* (2.1.4.5.).

2.2.4.1.4 Value

Otra manera de enviar un valor es determinarlo globalmente, es decir, para todo el ámbito del patch. Esto se logra con "value". De cualquier nombre como argumento e ingrese el valor como entrada. En cualquier otra parte del patch, puede recuperar el valor usando un bang seguido del mismo Objeto y argumento.:

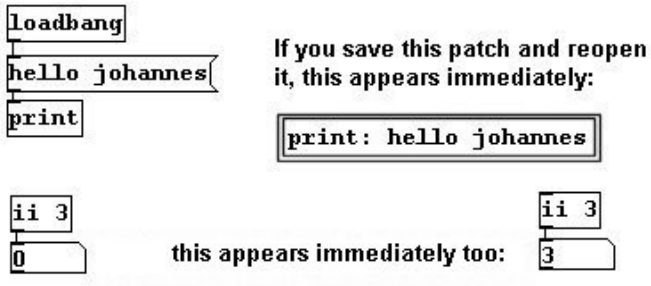




2.2.4.2 Loadbang

Probablemente quiera conservar varios valores para la próxima vez que abra el patch o quizá haya algún valor particular que quiera que reciba un bang justo en la apertura de un patch. Para lograr cualquiera de estos casos, podría utilizar el Objeto "loadbang" (envia un bang en cuanto se abre un

patch) o "init" (abreviación: "ii"), el cual envía un número o símbolo (o una lista de números, de símbolos, o de números y símbolos) como salida (Pd-extended).



2.2.4.3 Opciones GUI

GUI significa "graphical user interface" (interfaz gráfica de usuario) y se refiere a todos los Objetos gráficos especiales en Pd. Objetos GUI son *cajas Número y Símbolo*, *bang*, *toggle*, *slider*, *radio*, *canvas*, como así también *array* y *VU* (los últimos dos serán explicados más adelante). Todos los Objetos GUI poseen funciones extendidas. Para acceder a las mismas, realice un clic-derecho en el Objeto y selecciones "Properties" del menú desplegable.

A continuación nos enfocaremos en cada Objeto GUI y veremos la configuración de sus parámetros:

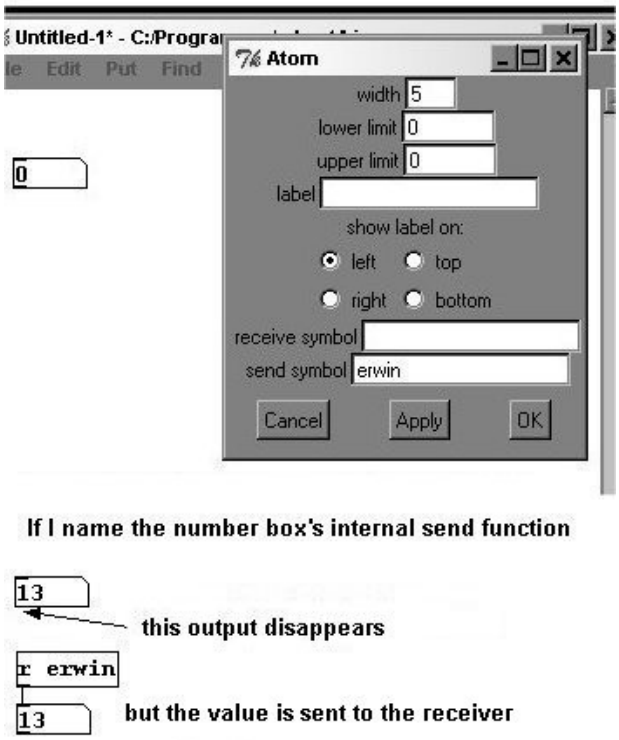
2.2.4.3.1 Number and symbol box

width (ancho) se refiere al ancho de la caja. Puede resultar útil ajustar esta configuración cuando se trabaja con número muy grandes o némeros con muchos lugares decimales.

Use *lower and upper limit* (límite inferior y superior) para ajustar el rango de valores que la caja aceptará (por ejemplo, puede requerir el rango de 0 a 1000).

Con *label* (etiqueta) puede asignar el nombre de la caja para su visualización (incluyendo la posición donde el nombre aparecerá).

Con *receive/send*,(recibir/enviar) puede construir una función de envío y recepción dentro de la caja. Por ejemplo, si ingresa "post1" en 'send' y tiene un receptor en algún lado llamado "post1", éste recibirá todos los ingresos realizados en la caja Número. Lo mismo ocurre con la función "receive".



Como puede observar en el gráfico anterior, la entrada y salida desaparecen cuando las funciones de recepción y envío son activadas.

Los cambios son efectudos cuando se realiza un clic sobre "apply" u "ok".

2.2.4.3.2 Bang

size (tamaño) se refiere al tamaño variable del objeto gráfico (en píxeles).

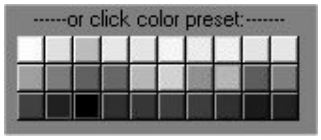
intrrpt/hold (interruptor/mantener) indica cuánto tiempo se activa el bang (en milisegundos).

init (inicialización) significa que el valor (en este caso, un bang) será enviado en cuanto sea abierto el patch (como ocurre con 'loadbang').

El *send symbol / receive symbol* (enviar símbolo / recibir símbolo) trabaja como la función interna "send" / "receive" alojada dentro de la caja Número.

Con *name* (nombre) puede crear una "etiqueta", como ocurre en la caja Número. La posición es determinada en valores *x* e *y*. Además, puede definir es estilo de fuente y el tamaño así como los colores para el fondo, el frente y el nombre. Primero seleccione el elemento que quiere cambiar, ...

... luego seleccione el color de las opciones de color predefinidas:



En "compose color" (componer color) puede también generar el color de su elección.

"backgd" se refiere al color de fondo, es decir, el color del breve encendido cuando el bang resulta activado (debido a un ingreso o a un clic de ratón).

Todos los cambios toman efecto cuando realiza un clic en "apply" u "ok".

2.2.4.3.3 Toggle

Este Objeto trabaja de manera análoga a un bang, exceptuando el *value* (valor), por defecto, 'toggle' alterna entre 0 y 1. Aquí puede ingresar otro valor reemplazando el 0 (el 1 siempre permanece como 1).

2.2.4.3.4 Slider

width (ancho): ancho en píxeles

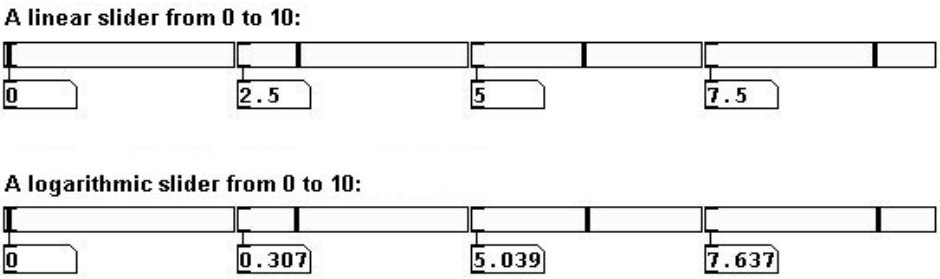
height (alto): alto en píxeles

bottom (inferior): valor del slider cuando se encuentra en el límite inferior

top (superior): valor del slider cuando se encuentra en el límite superior

lin: Adición/sustracción alternada entre lineal o logarítmica dentro del rango definido anteriormente. Si realiza un clic sobre el mismo, "log" aparece como opción actual. La opción actual es siempre la que aparece visible.

Note Bien: Con "log", '0' no puede ser usada como cifra de referencia.



init (inicialización): El valor inferior del rango será enviado automáticamente como salida cuando se abra el patch.

steady on click (clic constante): El cursor del slider es movido mientras se mantiene presionado el botón del ratón. Si realiza un clic sobre "steady on click", entonces aparece la opción "jump on click", la cual produce el movimiento inmediato del cursor mediante un clic, en el cualquier lugar que se desee dentro del rago del Objeto GUI slider.

El resto trabaja de la misma manera que en 'bang.'

2.2.4.3.5 Radio

Con 'radio,' todo funciona de la misma manera que los demás exceptuando el parámetro *number*: el mismo indica el número de celdas contenidas en el Objeto.

2.2.4.3.6 Canvas

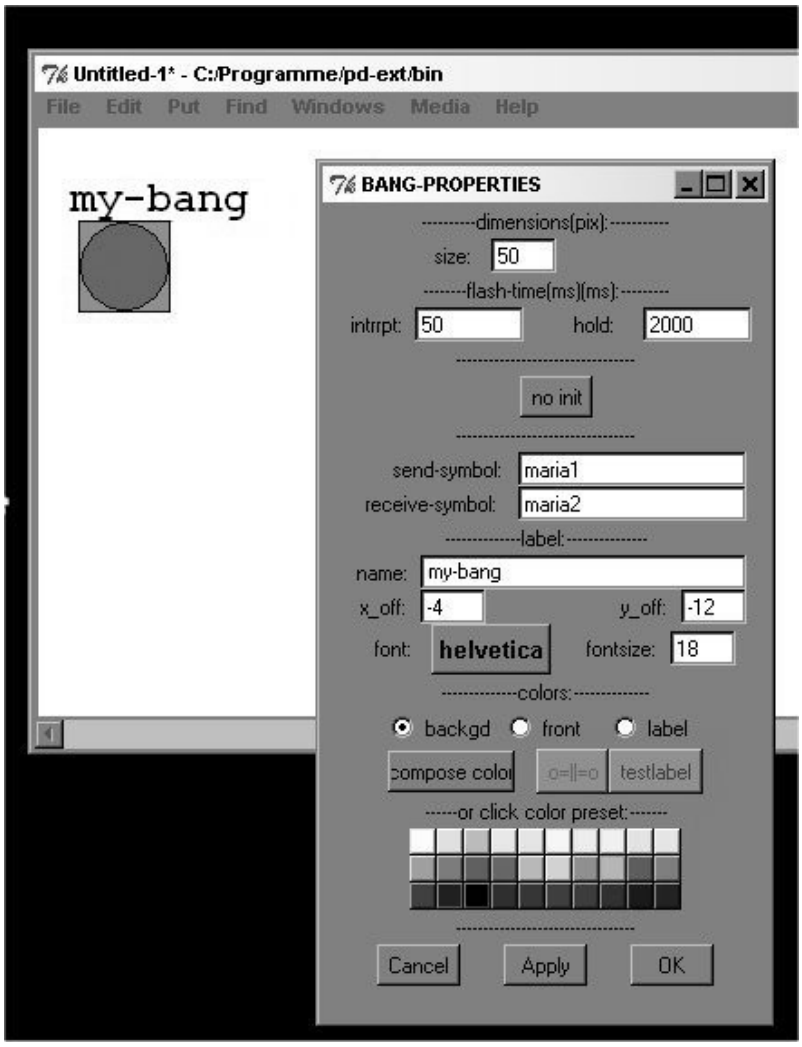
Al final de 2.1.2. observamos que la superficie blanca en la cual se depositan los objetos es llamada "canvas". También podemos agregar otras superficies coloreadas (**Put** > **Canvas**). No tienen otra

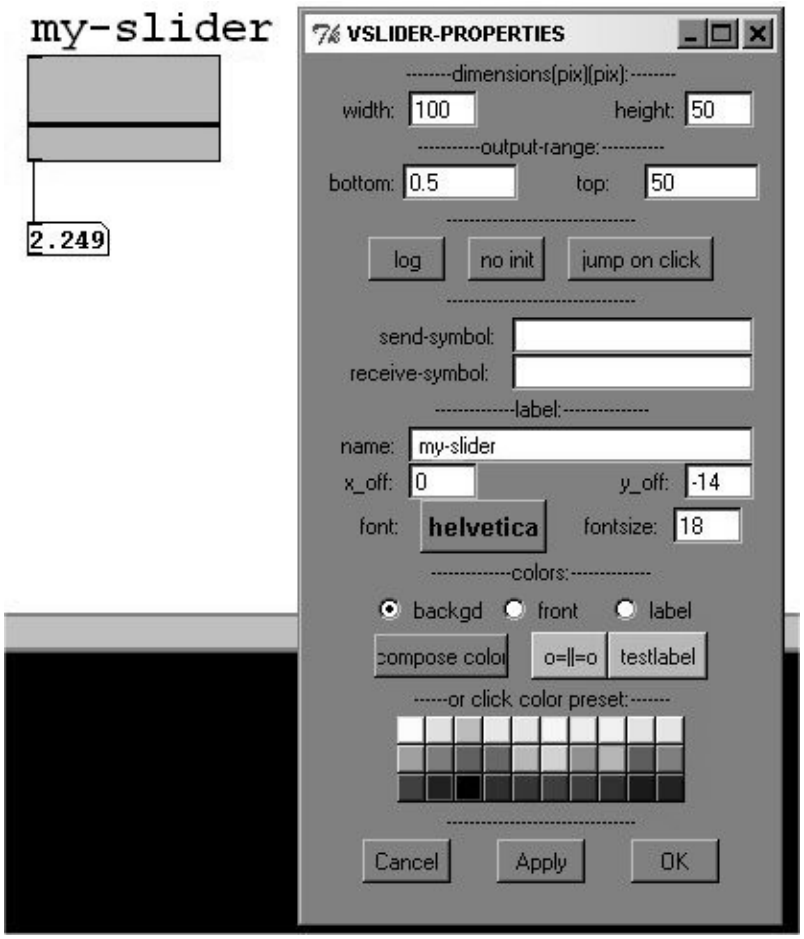
función que la de ser una superficie coloreada.

La superficie contiene un cuadrado azul en la esquina superior izquierda -ese es el objeto real. Toda la superficie es un producto de este objeto, su salida por así decirlo. Cubre todos los Objetos que fueron creados antes del mismo y yace debajo de todos los Objetos que cree a posteriori.

Los parámetros son los mismos que los que relacionesdos a los otros Objetos GUI.

2.2.4.3.7 Ejemplos de Objetos GUI alterados



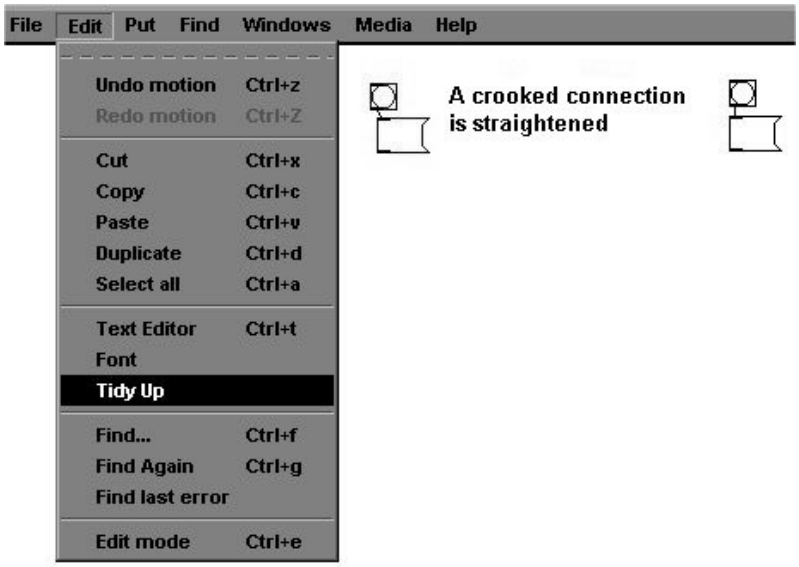


2.2.4.3.8 Cambio de tamaño de fuente

Puede cambiar la configuración del tamaño de fuente para todas las cajas bajo **Edit** > **Font**.

2.2.4.3.9 Arreglar las cosas

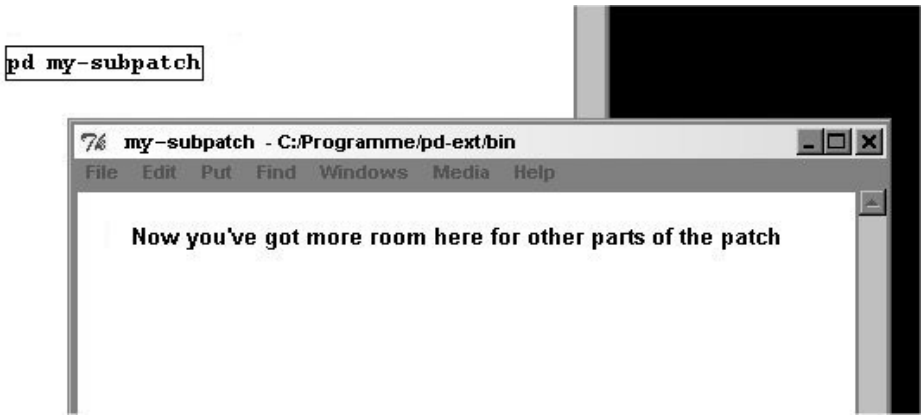
Para enderezar conexiones de cables, puede seleccionarlas y luego dirigirse a **Edit** > **Tidy up**. Sin embargo, esta opción no siempre funciona correctamente.



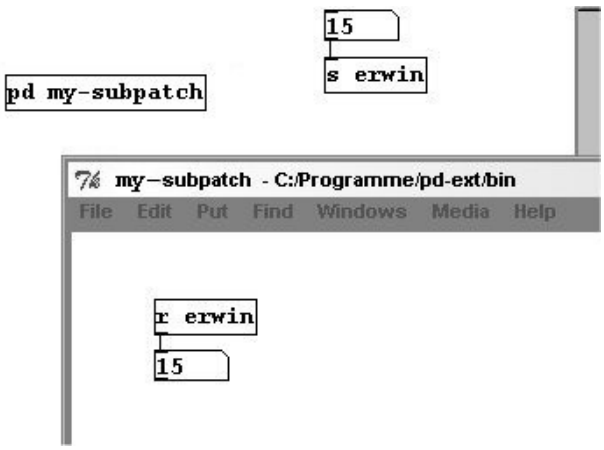
2.2.4.4 Subpatches

2.2.4.4.1 Espacio

A lo largo de la programación de un patch, en algún punto probablemente se encuentre sin espacio para seguir creando Objetos. Por esta razón puede guardar partes de un patch en lo que se llama "Subpatch". Si crea un Objeto "pd" e ingresa un nombre (sin espacios) como argumento -por ejemplo, "pd my-subpatch" – una nueva ventana es abierta. (Una vez que haya cerrado dicha ventana, puede volver a abrirla en el modo ejecución haciendo clic sobre el mismo Objeto una sóla vez.) Ahora tiene espacio para crear nuevos elementos dentro del patch.

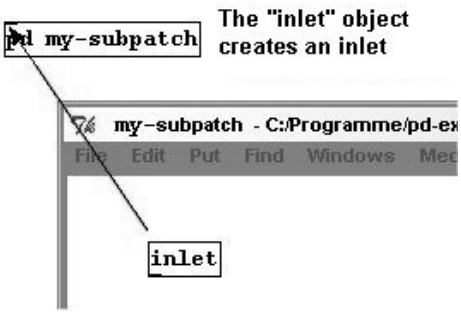


Existen dos maneras de conectar este subpatch a la primera ventana del patch: "send" y "receive" trabajan dentro de la misma ventana y entre ventanas diferentes:

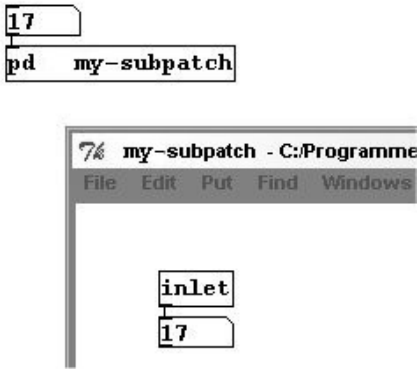


Aquí también puede observar cómo puede alternar independientemente entre el modo ejecución y el modo edición en ventanas distintas.

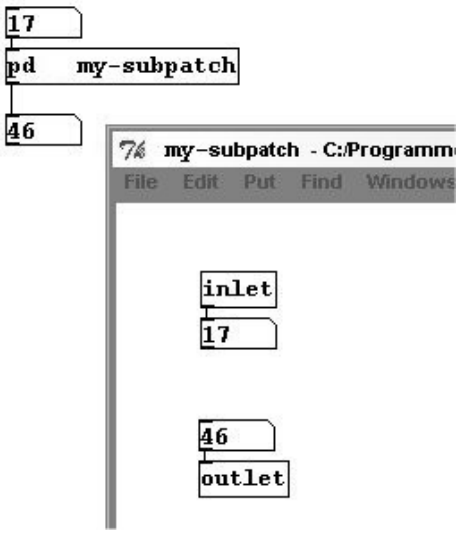
La otra forma de crear una conexión es usando los Objetos "inlet" y "outlet". Si crea un Objeto "inlet" en el subpatch (el cual se encuentra en la primera ventana y es llamado "pd my-subpatch") , el Objeto subpatch mostrará una entrada.



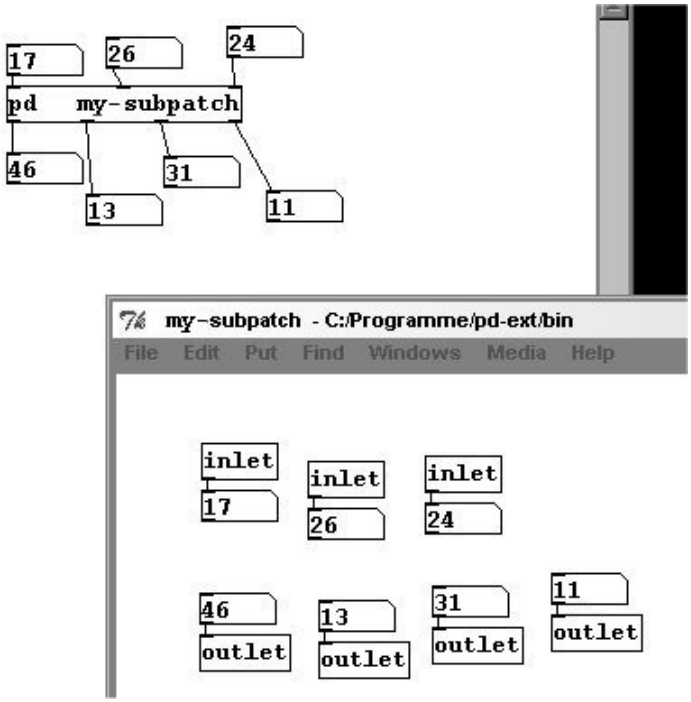
Si ingresa un número como entrada en la primera ventana, este aparecerá en el subpatch.



El Objeto "outlet" trabaja de forma similar:



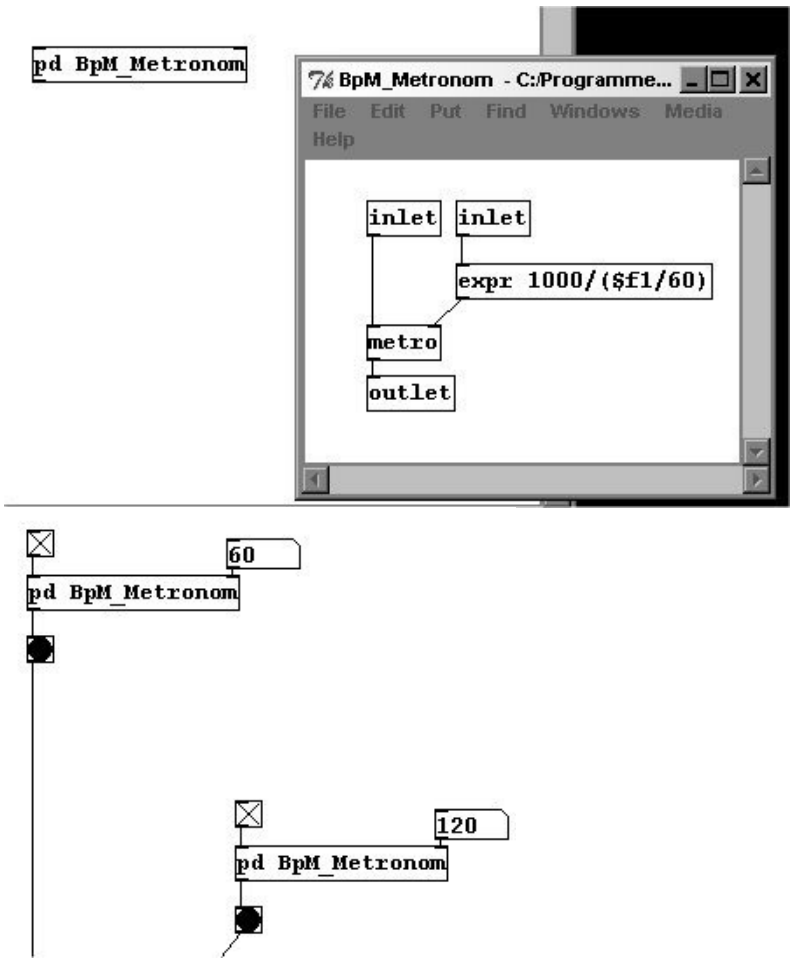
Varios Objetos "inlet" o "outlet" son colocados dentro del subpatch uno al lado del otro análogamente a la distribución en el Objeto subpatch de la primera ventana:



Cuando cierra la ventana del subpatch, el subpatch permanece activo (es decir, continúa ejecutando tareas) siempre y cuando la ventana principal se mantenga abierta.

2.2.4.4.2 Modularización

Los subpatches no sólo solucionan limitaciones espaciales, sino también ayudan a dar una estructura más clara a su patch. Las partes de un patch que completen una tarea específica pueden ser incorporados a un subpatch, para que esta pequeña 'máquina' esté siempre disponible. Llamamos "modulo" a una parte de este tipo. Aquí tiene un ejemplo de esto: un metrónomo que permite ingresar pulsos por minuto en vez de milisegundos:



Los nombres de subpatch que usted elija no pueden contener espacios. En cambio puede utilizar guiones medios o guiones bajos, como lo hacemos en este ejemplo.

Capítulo 3. Audio

Table of Contents

3.1 Lo esencial

De aquí en más, los patches de mayor tamaño se encuentran pre-ensamblados en archivos adicionales (http://www.kreidler-net.de/pd/patches/patches.zip).

Muchas de las funciones descritas en el Capítulo 2 no serán utilizadas en el resto del texto – por ejemplo los Objetos “send” y “receive”- aunque sean muy utilizados en la práctica. Los patches que

encontrará aquí han sido reducidos a lo esencial. Sin embargo, cuando se utilizan las técnicas presentadas en una composición o performance, será necesario almacenarlas como subpatches y conectadas con entradas/salidas, etc. Puede ver un ejemplo de esto aquí: 3.4.2.4.

3.1.1 Altura

Regresemos al primer ejemplo. Escuchaste un tono con una frecuencia de 440 Hertz (luego otras frecuencias, otras alturas)You heard a tone with a frequency of 440 Hertz (later with other frequencies, other pitches). Podías encenderlo o apagarlo – por ejemplo, ajustando la dinámica de alto a inaudible.

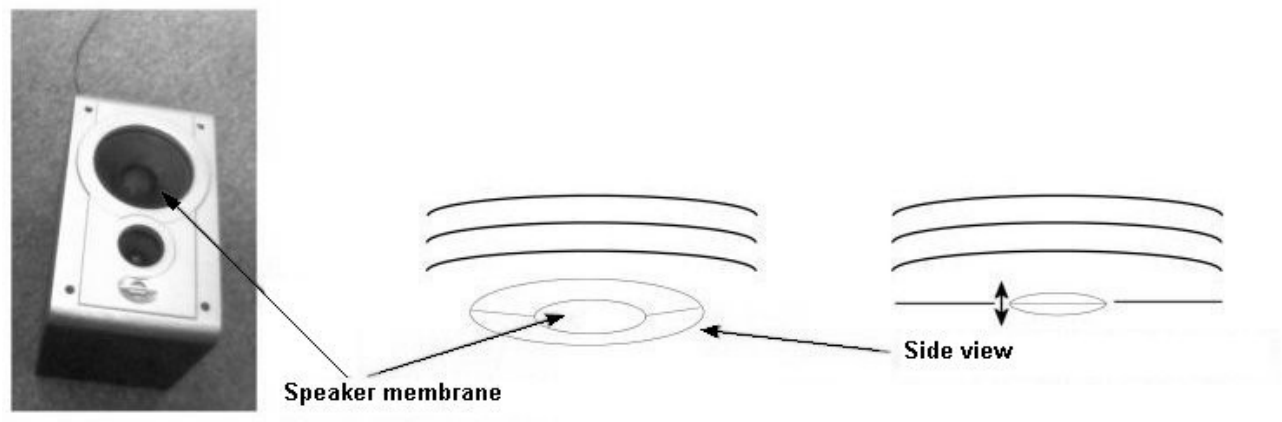
Sobre la altura: Pd trabaja con dos clases de altura – Hertz o números MIDI. Los signos tradicionales: A, Bb, G#, etc. no son utilizados en Pd. En su lugar, todas las alturas cromáticas tienen números MIDI: A4 corresponde al número 69, Bb4 al 70, etc. La otra forma de describir una altura en Pd es en Hetz. Para entender esto, necesitamos sumergirnos un poco en acústica musical.

3.1.1.1 Teoría

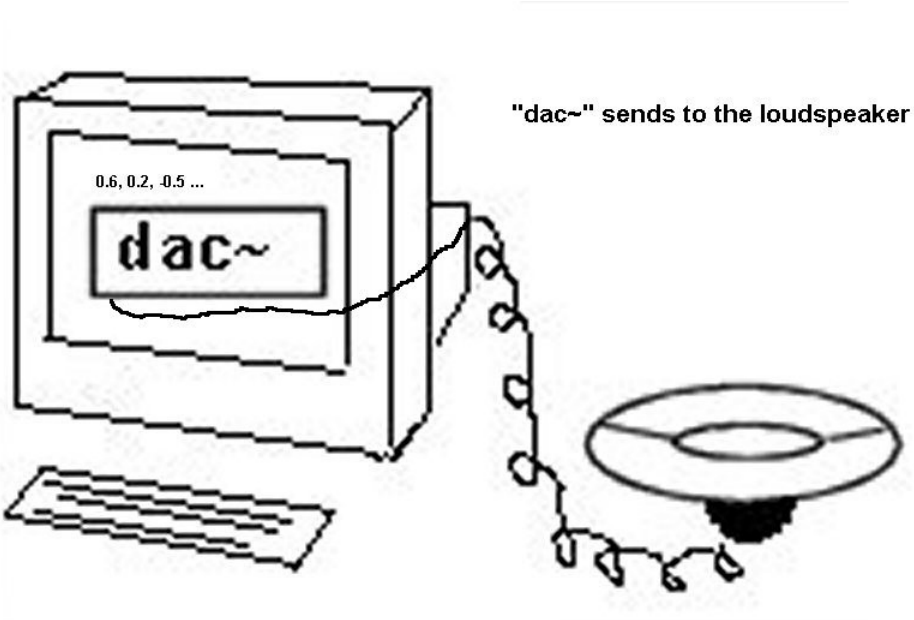
3.1.1.1.1 Controlando altoparlantes digitalmente

El sonido es aire en vibración. Los instrumentos tradicionales son utilizados para poner en vibración al aire a frecuencias específicas. Puedes realizar esto, por ejemplo con cuerdas (violín), labios (trompeta), o membranas (timbal). Incluso tenemos una membrana en el oído que vibra por simpatía a partir de las vibraciones presentes en el aire. Nuestro cerebro transforma estas vibraciones en otra forma diferentes, que es lo que nosotros llamamos sonido.

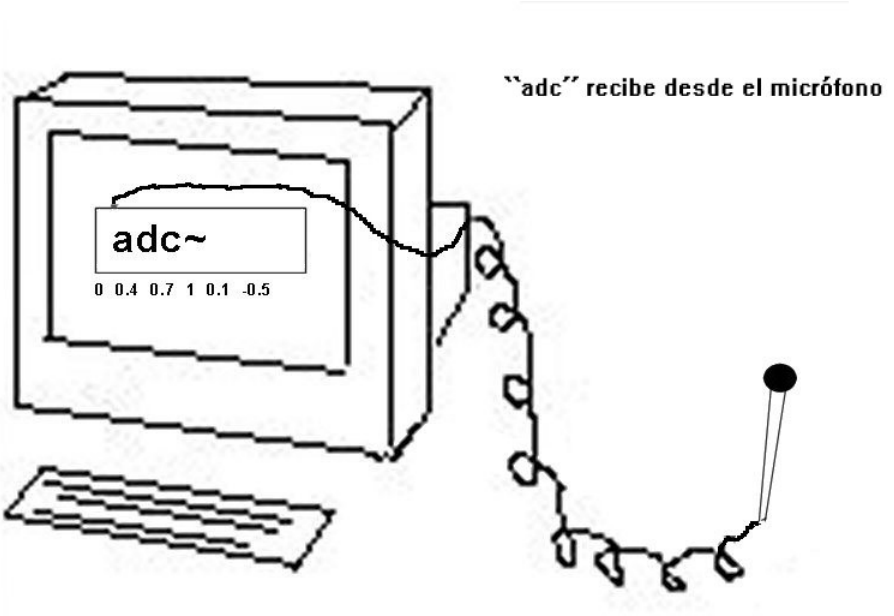
En la música electrónica, utilizamos altoparlantes para generar el sonido. Estos también disponen de una membrana (o varias) que vibra para adelante y para atrás, causando la vibración del aire.



Las vibraciones de esta membrana están controladas por el ordenador. En Pd, el Objeto “dac~” (digital audio converter “conversor de audio digital”) se encarga de esto. Así es como trabaja: el sonido es un fenómeno físico – vibraciones en el aire para ser precisos – y, como tal, es analógico. Sin embargo, los ordenadores sólo trabajan con números, por esto es digital. El Objeto “dac~” convierte los números en sonido al transformar los números en fluctuaciones de corriente eléctrica que, una vez amplificada, causa la vibración simpática de la membrana del altoparlante.



El caso inverso de este proceso sería conectar un micrófono al ordenador. Un micrófono tanbién posee una membrana que responde a las vibraciones del aire y convierte dichas vibraciones en fluctuaciones de corriente eléctrica, que envía al ordenador donde son convertidas en números. En Pd, esta entrada puede ser recibida con el Objeto “adc~”



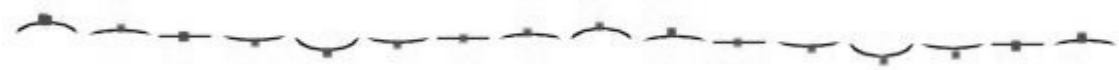
Volvamos un momento a los altoparlantes. Una membrana de un altoparlante se puede mover hacia adelante y hacia atrás. La posición más exterior (la más convexa) es entendida por el ordenador como posición 1. La posición más interior (la más cóncava) corresponde a la posición -1. Cuando la membrana se encuentra en el medio, como cuando reposa, la posición es 0. Todas las demás posiciones son valores que oscilan entre -1 y 1.

En realidad, estos movimientos son tan pequeños y tan rápidos que casi no pueden ser observados a simple vista.

3.1.1.1.2 Ondas

Imaginemos que una membrana se mueve de un extremo límite al otro (más convexo, más cóncavo) a tiempo constante:

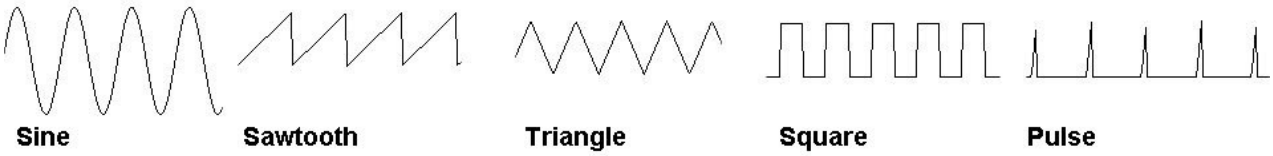
Marquemos las etapas individuales:



De forma abstracta, con el tiempo en el eje X y la posición de la membrana en el eje Y, podríamos representar dicho movimiento el que sigue:

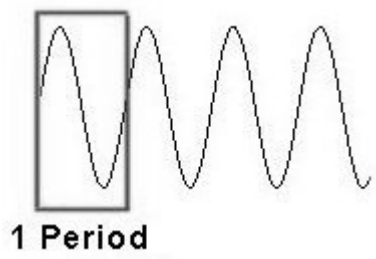
En terminología física, se llama a esto *onda*. Aquí podemos ver claramente la *forma de onda* – un triángulo.

Existen diferentes formas de onda para diferentes tipos de movimiento de membrana. Sus nombres reflejan su semejanza visual:

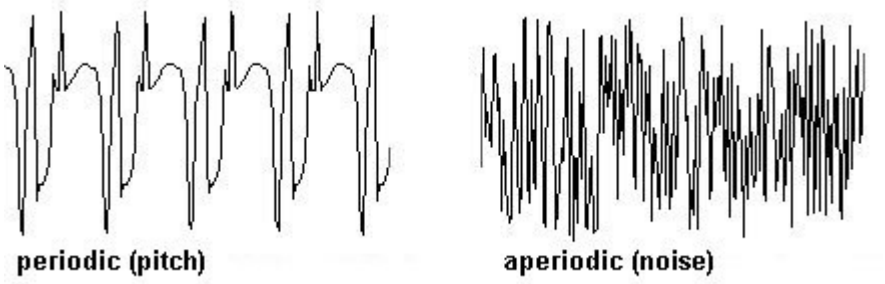


El Objeto “osc~” crea una onda sinusoidal.

Algo importante para recordar en relación a las formas de onda: ellas se repiten de manera constante sin cambiar las características de movimiento. Las vibraciones que exhiben esta cualidad se las llama *periódicas*. Un período es un ciclo completo de una vibración que se repite de manera constante.

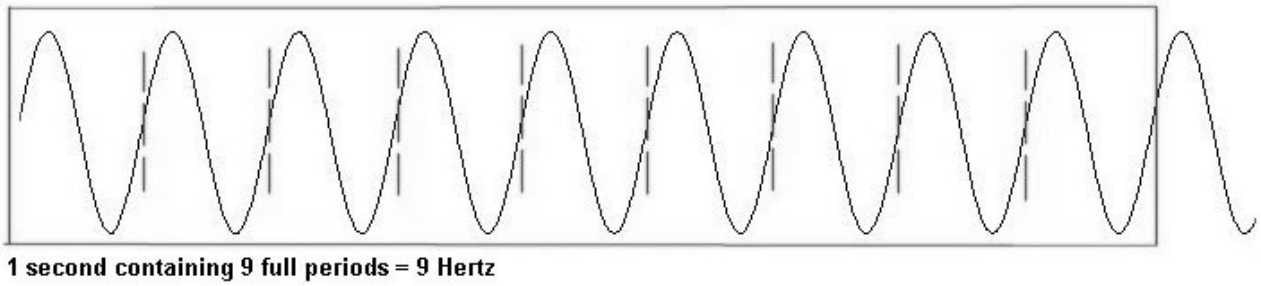


Lo que hace especiales a las las vibraciones periódicas es que las escuchamos como tonos claros con altura definida. En contraste, los ruidos son vibraciones aperiódicas.



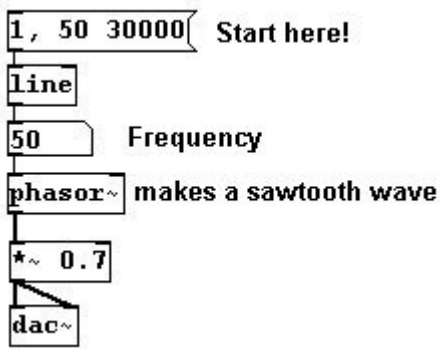
3.1.1.1.3 Medición

Primero hablemos sobre las vibraciones periódicas. Es posible contar simplemente el número de períodos en un segundo. Este número es la frecuencia de la vibración y es medida en "Hertz" (Hz); la frecuencia en este contexto siempre significa qué tan seguido algo se repite en un segundo (expresado matemáticamente: 1/segundo).



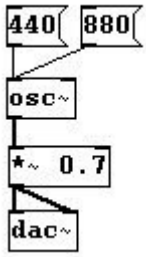
La frecuencia fundamental de un tono determina su altura. A440 (también llamado A4 (La4), la altura estandard que las orquestas utilizan para afinar) significa que el aire vibra periódicamente a la razón de 440 veces por segundo; C5 (Do5) vibra 523 veces por segundo; y el Sol grave de la cuarta cuerda del cello vibra aproximadamente a 100 veces por segundo.

Aquí se puede observar lo siguiente: mientras más baja sea la frecuencia, más baja es percibida la altura por el oído. En efecto, los humanos -en función de la edad- percibimos alturas entre 20 Hz y 15000 Hz. Los niños pueden oir hasta 20000 Hz; las personas mayores pueden llegar a oir hasta 10000 Hz. Los perros y murciélagos superan con creces los 20000 Hz Este límite superior es llamado límite ultrasónico. En contraste a éste tenemos un rango infrasónico, el cual es más bajo que el umbral inferior de audición -por ej. Entre 0 y 20 Hz. Este rango es percibido por nosotros como ritmo. Ud. puede experimentar por su cuenta con el siguiente patch en Pd:

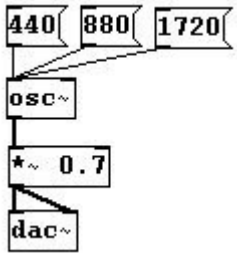


Se puede escuchar un ritmo de clicks (así suena una onda de diente de sierra) que gradualmente se vuelve más rápido. A partir de determinada velocidad (por encima de los 20 clicks por segundo), nuestra percepción procesa de manera diferente y comenzamos a escuchar una altura. En el aire (y en el ordenador), todavía existe un "ritmo". ¡Pero para nuestro oído (ca. 20 Hz.) es una altura! Mientras más rápido sea dicho ritmo, más alta será la altura percibida.

Otra característica que define la escucha humana es que las alturas se oyen logarítmicamente. Esto significa que cuando se dobla una frecuencia dada, la percibimos como salto de octava. Si doblamos la frecuencia de un sonido, por ej., desde un A4 (440 Hz.) a su doble (880 Hz.), escuchamos un A5, el cual equivale a una octava más alta del original:

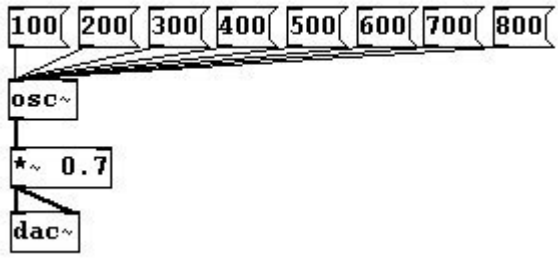


Si queremos escuchar una superior a 880 Hz., tenemos que doblarla nuevamente. 880 + 880 = 1760:

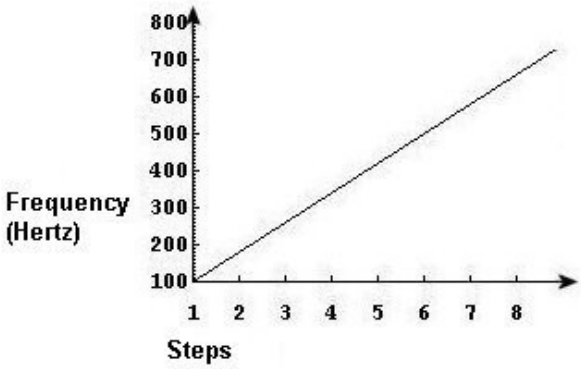


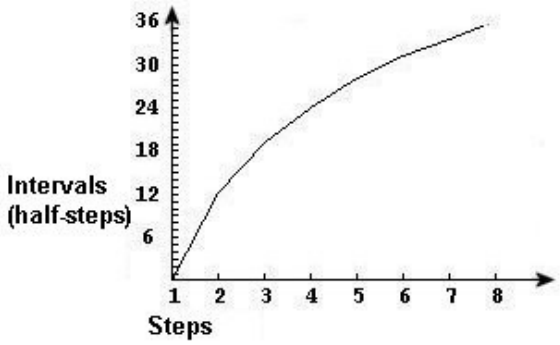
Para clarificar: de 30 Hz. a 60 Hz., escuchamos un salto de octava pero de 1030 Hz. a 1060 Hz., escuchamos sólo un pequeño intervalo. ¡De hecho, el salto de 10000 Hz. a 20000 Hz. equivale a un salto de octava también!

Otro concepto importante: agreguemos una misma cantidad a una frecuencia fundamental, digamos 100 Hz. -la cual se aproxima a la frecuencia del Sol al aire de un cello- y sumemos sucesivamente 100 Hz.:



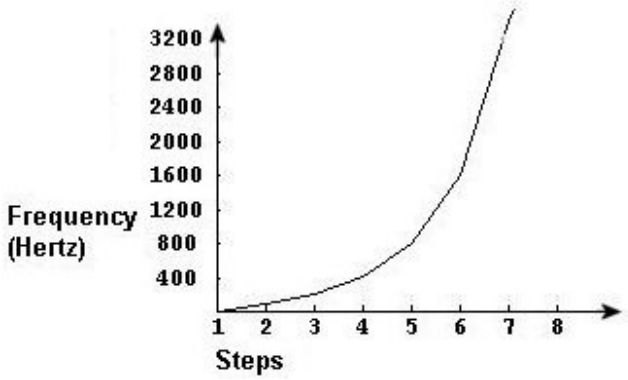
Escuchamos una octava de 100 a 200, una quinta de 200 a 300, una cuarta de 300 a 400, etc. En matemáticas, este es un proceso aditivo en el cual una misma cantidad es agregada una detrás de otra. Nuestro oído, sin embargo, percibe que dicha cantidad se vuelve más pequeña en cada paso:

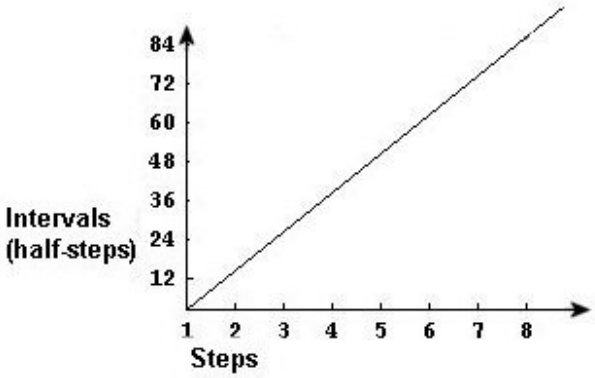




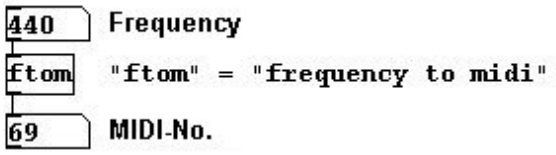
El gráfico a la izquierda muestra una *función lineal*. El de la derecha muestra lo que escuchamos – una *función logarítmica*.

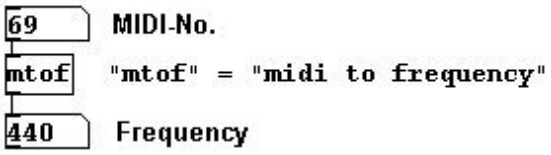
Si queremos escuchar una progresión lineal -por ej. un proceso mediante el cual se agrega un mismo intervalo, en este caso una octava-, la función matemática tiene que ser *exponencial*:





En Pd, la conversión de progresiones lineales a logarítmicas es llevada a cabo utilizando números MIDI y frecuencias. Los números MIDI reflejan la forma en que escuchamos, en donde los intervalos de lo escuchado corresponde a un intervalo equivalente en números MIDI: un número entero por cada semitono. En Pd, podemos convertir los datos de cajas de número en frecuencias y/ o números MIDI de la siguiente manera:





A continuación vemos una pequeña tabla de correspondencia entre frecuencias, números MIDI y los nombres tradicionales de las notas:

Frequenzy (Hz)	MIDI	Note name
65.4	36	C2
100	43	G2
130.8	48	C3
261.6	60	C4
277.1	61	C#4
293.6	62	D4
311.1	63	D#4
329.6	64	E4
349.2	65	F4
370	66	F#4
392	67	G4
415.3	68	G#4
440	69	A4
466.1	70	Bb4
493.8	71	B4
523.2	72	C5
1000	83	B5
4186	108	C8

Nota bien: los osciladores tales como "osc~" o "phasor~" tienen que recibir Hz. en su entrada.

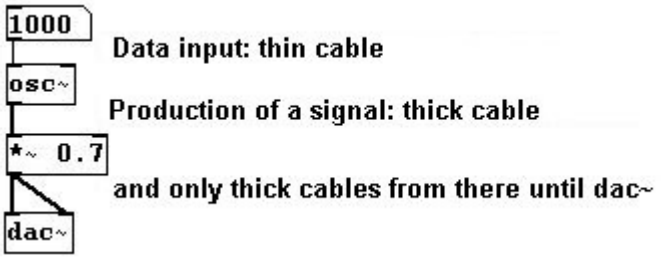
3.1.1.1.4 Frecuencia de muestreo

Debemos recordar que para Pd, el sonido consta sólamente de números. Las posiciones de la membrana de un altoparlante varía entre los números -1 y 1.

Los Objetos como "osc~" generan una secuancia muy rápida entre -1 y 1 que es enviada al parlante mediante el Objeto "dac~". Para ser más específicos, se generan y envían 44100 números por segundo. El altoparlante realiza 44100 pequeños movimientos entre -1 y 1 en el marco de un segundo. Este némero, 44100, es el valor de la llamada *frecuencia de muestreo*.

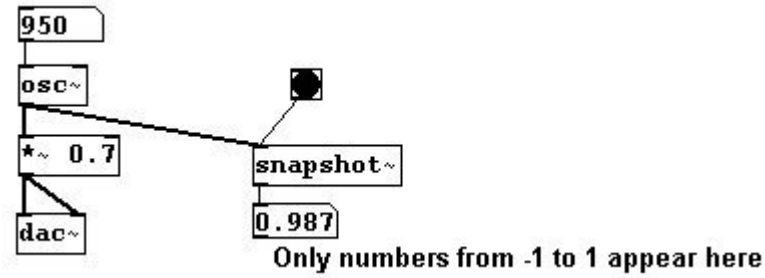
Cada sonido en Pd es producido usando números entre -1 y 1 a una frecuencia de 44100 números por segundo (frecuencia de muestreo). Cada número individual es llamado *muestra*.

Tosos los Objetos de Pd que generan o procesan datos a esta frecuencia poseen una virgulilla "~" dentro de la caja. Estos Objetos se conectan entre sí mediante *cables gruesos*. Llamamos a estas series de números, *señales*.

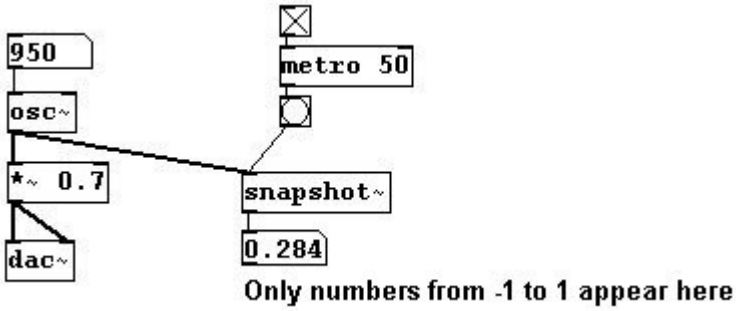


En cuanto lo desee, puede modificar el valor de entrada de frecuencia al Objeto "osc~". El cable para esta conexión es delgado porque la entrada no se encuantra en transmisión constante. Sin embargo, la salida del Objeto "osc~" se encuentra constantemente enviando señales: por ej. números entre -1 y 1, 44100 de ellos por segundo (Hetz significa *ciclos por segundo*).

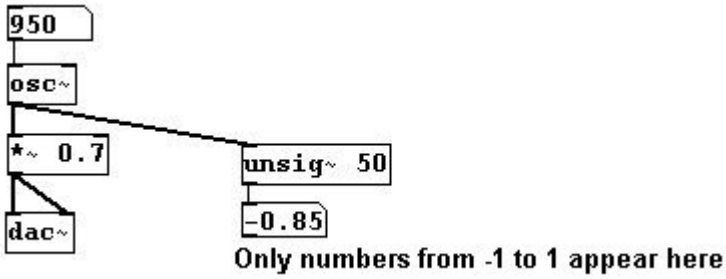
Si quiere observar los números de salida de una caja Objeto "osc~", no lo logrará tan sólo conectando a la salida del mismo una caja Número. Las cajas Número sólo pueden ser utilizadas para conexionesde control y no para conexiones de señal. Las conexiones de señal son muy rápidas: sería imposible ver 44100 números diferentes por segundo. Sin embargo, es posible visualizar algunos números de una señal mediante el Onbejto "snapshot~". En su entrada puede recibir una señal de sonido y un bang (*disparo*) que, una vez pulsado, envía por su salida el número actual. Para monitorear dicho valor conectamos a la salida del Objeto "snapshot~" o bien una caja Número o un Objeto "print":



Si desea un flujo constante de estos números, puede incluir un metrónomo de alta velocidad:



En Pd-extended podría también utilizar "unsig~", el cual cumple una función de metrónomo. Ingrese como argumento el valor de metrónomo:

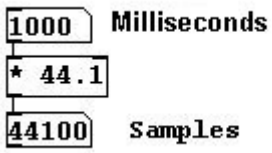


También puede usar "sig~" para convertir números de nivel de control a nivel de señal. Una vez que ingresa un valor a la entrada, se envía por la salida del mismo, dicho número 44100 veces por segundo.

3.1.1.1.5 Muestras – milisegundos

Al igual que la frecuencia (y también la emplitud, como se verá en el siguiente capítulo), existen *dos* unidades diferentes para medir el tiempo en Pd: muestras y milisegundos. Las muestras se utilizan generalmente para contar señales mientras que los milisegundos se utilizan para el control de datos.

Para convertir la duración medida en milisegundos a duración medida en muestras:

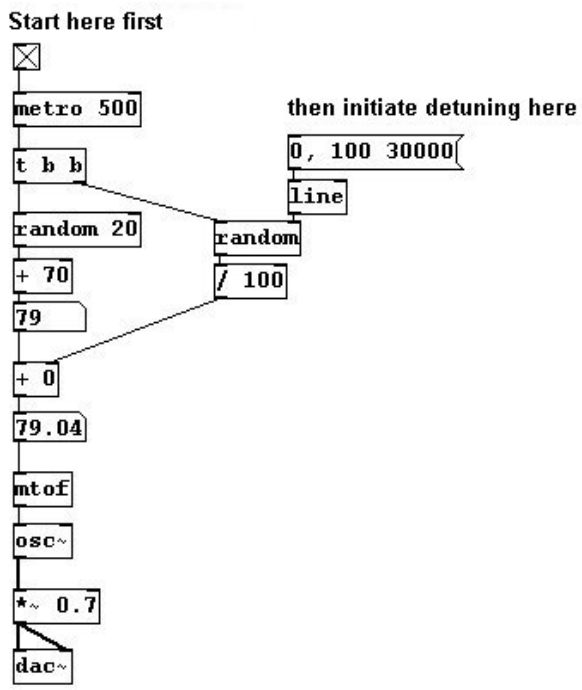


3.1.1.2 Aplicaciones

3.1.1.2.1 Temperado – Aleatorio

Los valores MIDI aleatorios son desplazados gradualmente. (Transición desde la afinación de temperamento igual a una afinación aleatoria):

patches/3-1-1-2-1-random-offset.pd



3.1.1.2.2 Más ejercicios

a) Crear un efecto de glissando que se escuche como lineal y otro que se escuche como logarítmico desde C3 hasta C6.

b) Crear una escala de cuarto de tono.

3.1.1.3 Apéndice

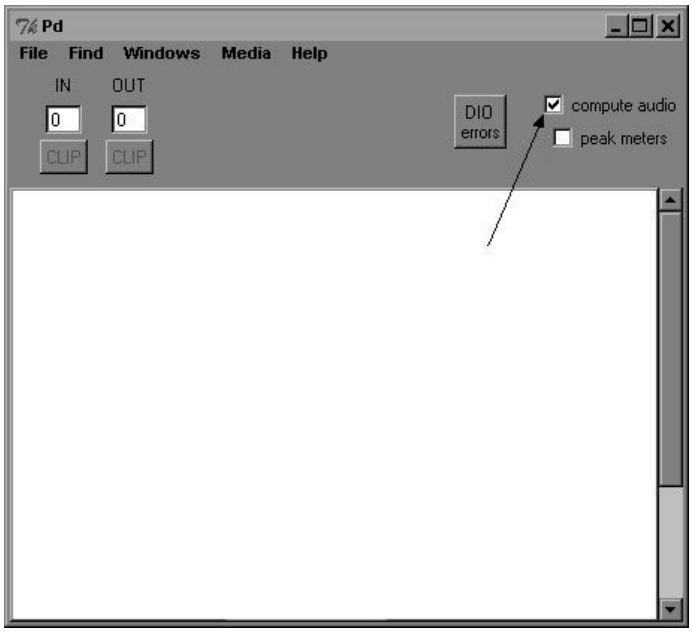
3.1.1.3.1 Teorema de Nyquist

El número 44100 fue elegido por una buena razón. Como mencionamos anteriormente, los seres humanos pueden escuchar, como límite superior, hasta los 20000 Hz. En 1928, el físico norteamericano Harry Nyquist (1889-1976) propuso una teoría en donde establecía que, para representar correctamente una señal de sonido de forma digital, era necesaria utilizar una frecuencia de muestreo de al menos el doble de la frecuencia de la señal a muestrear ("Teorema de muestreo Nyquist-Shannon"). Concretamente esto significa que necesitamos los valores máximo y mínimo por cada período para representar correctamente el contorno básico de una forma de onda, por ej.: dos puntos por período.

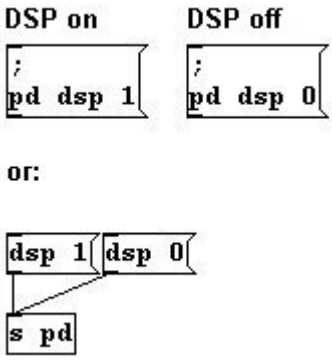
Para una onda de 20000 Hz, que equivale a 20000 períodos por segundo, necesitamos al menos 40000 puntos por segundo para representarla correctamente. Para asegurar que todo el espectro de audición humana fuera incluído, se eligió una frecuencia de muestreo de 44100 para el primer soporte de audición digital comercial: el CD de audio. Esto significa que ondas de hasta 22050 Hz. pueden ser registradas y reproducidas en y a partir de dicho soporte. Para los ordenadores existe una amplia selección de anchos de banda de frecuencia, siendo la más baja de 8000 Hz. para los sonidos básicos del sistema operativo. Las grabaciones de audio de alta fidelidad trabajan con frecuencias de muestreo de 48000 Hz. (48 kHz = kiloHertz, donde kilo = 1000), 96 kHz., o hasta 192 kHz.

3.1.1.3.2 DSP

Ya hemos mencionado que el procesamiento digital simultáneo de numerosas señales demanda muchos recursos en el ordenador. Imagine trabajar con 100 Objetos "osc~". Cada uno genera 44100 números por segundo y estos tienen que ser sincronizados entre sí. Por esta razón Pd ofrece la opción de desactivar el DSP (Digital Signal Processing = *procesamiento de señal digital*) en su ventana principal. Esto le ahorra trabajo innecesario al procesador.



También puede activar o desactivar el DSP mediante un mensaje; el destino "pd" es en este caso el programa mismo:



Con respecto a la música por ordenador, mientras más rápido sea el procesador, más alta resultará la performance del mismo.With regard to computer music, the faster the processor, the higher the performance.

Pd reduce su carga de trabajo al procesar bloques de muestras en vez de hacerlo individualmente una por una. Esto mejora en gran medida el rendimiento. El tamaño de bloque estandard es de 64 muestras, pero esta configuración puede sermodificada. Más sobre este tema en 3.8.1.1

3.1.1.4 Para interesados particularmente

3.1.1.4.1 Conversión da- / ad-

Hemos declarado anteriormente que el Objeto "dac~" envía números generados por Pd a la membrana del altoparlante (3.1.1.1.1), pero por supuesto que esto es una sobre simplificación. Estrictamente hablando, la placa de sonido del ordenador convierte los números en corriente eléctrica mediante voltaje variable ("conversión digital-analógica" o "conversión da"); la posición de la membrana es determinada por la cantidad de voltaje. De manera inversa, las fluctuaciones de la membrana de un micrófono son convertidas en una corriente variable, que luego es digitalizada por la placa de sonido del ordenador ("conversión analógica-digital" o "conversión ad").

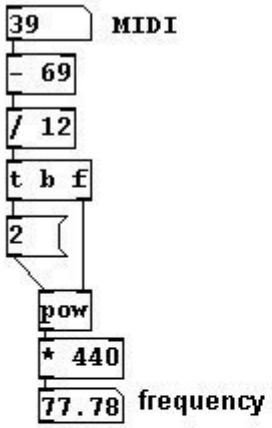
3.1.1.4.2 Ondas sonoras

Al contrario de las ondas de agua, las ondas sonoras se comportan de manera longitudinal. Las ondas longitudinales, también llamadas ondas de compresión, están caracterizadas por el hecho de que vibran a lo largo de la dirección de su movimiento (las ondas transversales, por otro lado, vibran a lo largo de un eje perpendicular a la dirección del movimiento). Para mayor explicación consulte un manual de física de nivel medio.

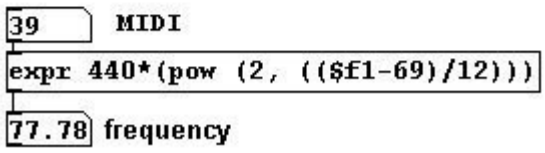
3.1.1.4.3 Convertir números MIDI en frecuencias

El Objeto "mtof" convierte números MIDI en frecuencias. La expresión para este cálculo es:

$$f = 440 \cdot 2^{(m-69)/12}$$



In one expression:



Para calcular la frecuencia fundamental de una altura en temperamento igual, que se encuentra a una determinada distancia de una frecuencia dada, utilizamos la siguiente expresión:

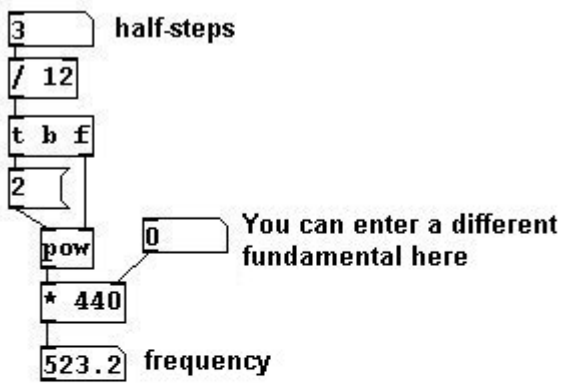
$$f = g \cdot 2^{a/12}$$

'f' es la frecuencia incógnita, 'g' la frecuencia de una altura dada y 'a' el intervalo en semitonos.

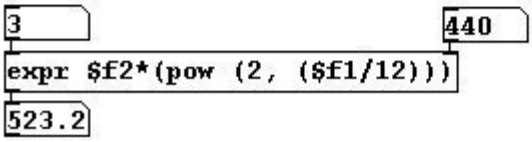
Por ejemplo, si queremos calcular la frecuencia de C5 y sabemos que A4 tiene una frecuencia de 440 Hz:

$$f = 440 \cdot 2^{3/12} = 523.2$$

En Pd:



In one expression:



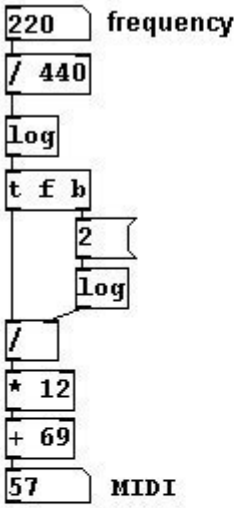
Para la operación inversa -convertir una frecuencia a numero MIDI- la expresión que aplicamos es:

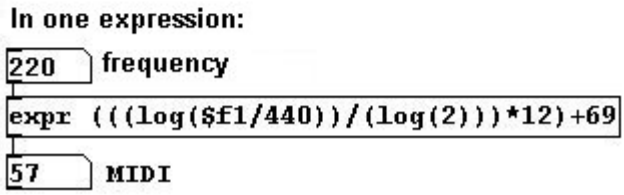
$$m = 69 + 12 \cdot \log_2(f/440)$$

Sin embargo, en Pd sólo contamos con el logaritmo natural basado en el número de Euler (la constante matemática 'e'); entonces también necesitamos esta otra expresión:

$$\log_a(b) = \frac{\log_e(b)}{\log_e(a)}$$

Programado en Pd:





3.1.1.4.4 Periodicidad del ruido

Hemos hablado sobre la no periodicidad en los ruidos. Sin embargo, podemos imaginar un ruido que dure 10 segundos y que luego se repita precisamente como antes.Dicho ruido tendría teóricamente una frecuencia de 0,1 Hz. Entonces podríamos definir más precisamente a un ruido como un tipo de sonido que es aperiódico o que tiene un período menor a 20 Hz. Es más, podríamos también aseverar que las frecuencias de un ruido tienen un tono fundamental que es menor a 20 Hz.

Se han realizado muchos experimentos interesantes en el campo de la acústica, por ejemplo, los concernientes al efecto Doppler o los relacionados al cálculo de la longitud de las ondas sonoras. Para más información consulte los manuales de acústica más destacados.

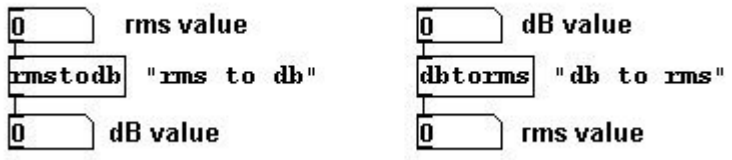
3.1.2 Volumen

3.1.2.1 Teoría

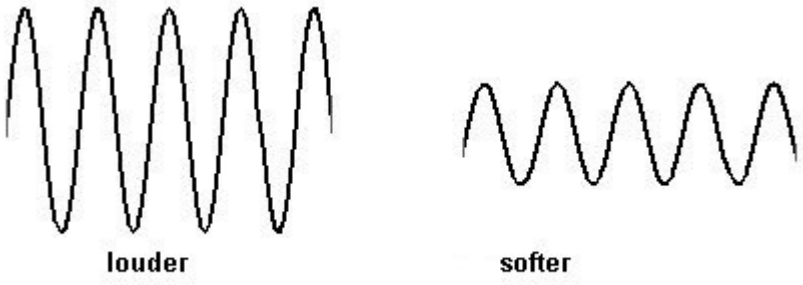
3.1.2.1.1 Medición

El próximo parámetro sonoro que tomaremos en consideración será el de *volumen*. Tradicionalmente en la música escribimos, notamos la sensación de volumen utilizando signos dinámicos o matices tales como pianissimo, piano, mezzoforte, etc. Su uso es subjetivo y variable dependiendo principalmente del instrumento en cuestión. En la física, la cual es el modelo que toma Pd para este parámetro, el volumen es representado con valores objetivos mediante la unidad *decibelio* (dB) o en la unidad *valor eficaz* (RMS = "root mean square" = valor cuadrático medio o media cuadrática). Ambas unidades son comparables a los nuémeros MIDI y frecuencias para las alturas. El decibelio (dB) refleja lo que oimos, donde una octava en volumen corresponde a 6 dB. El rango de la escala se extiende de 0 a 130 dB -donde un valor entre 15 y 20 dB corresponde al

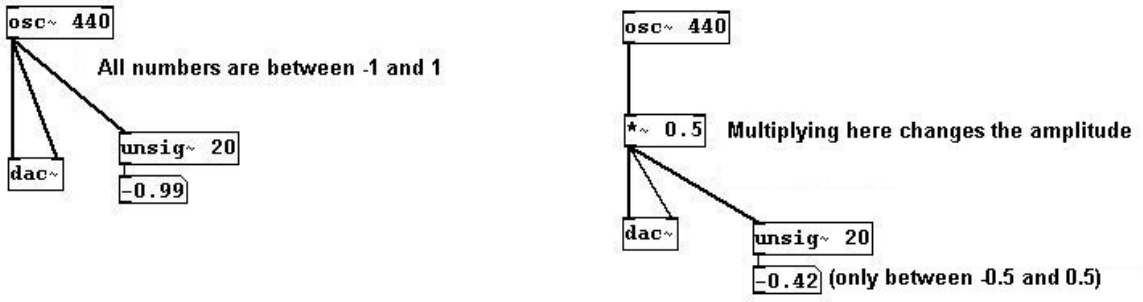
silencio absoluto mientras que cualquier valor que supere los 120 dB es capaz de causar un serio daño de audición. Luego de los 130 dB sólo percibimos dolor. El valor eficaz (RMS), como ocurre con las frecuencias, no corresponde a lo que oimos sino que son valores entre 0 y 1, donde 0 corresponde a 0 dB y 1 a 100 dB. RMS hace referencia a la raiz cuadrada de la media aritmética de los cuadrados de los valores de amplitud que toma como referencia para su cálculo. Entonces, estos valores son en principio elevados al cuadrado, luego se promedian (sumando todos los valores y dividiéndolos por el número de elementos), y en último lugar se aplica a dicho resultado la raiz cuadrada. Los valores RMS para una señal de audio es calculada utilizando una porción de dicha señal con una duración específica; para una señal altamente fluctuante como una frecuencia de altura, nos da una idea de la amplitud promedio de dicha señal. Los siguientes Objetos pueden ser utilizados en Pd para la conversión de uno a otro:



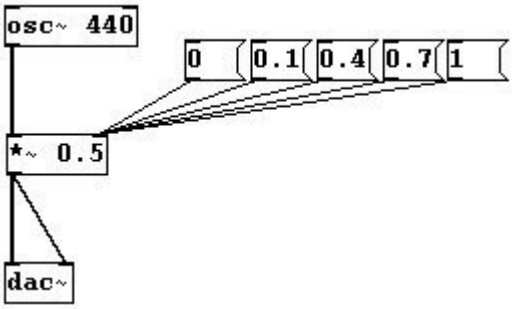
El volumen de un vibración es determinada por su amplitud, la cual es el grado en que la membrana es desplazada a ambos lados de su posición neutral en reposo (posición cero). Mientras más grande es el movimiento de la membrana, percibimos el sonido con mayor intensidad. Su representación puede verse así:



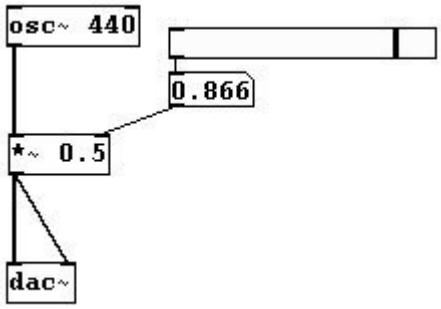
Enfaticemos aún más: Pd sólo trabaja con números hasta que, mediante la ayuda del Objeto "dac~" y la placa de sonido, estos se transducen en señal analógica continua. Si un sonido es débil en intensidad, esto significa que los números no se extienden en el rango completo de -1 a 1, sino más bien se encuentran restringidos a un rango más cercano a la posición cero, digamos por ejemplo, entre -0.5 y 0.5. Esto puede ser realizado en un patch multiplicando los números generados por el Objeto "osc~" por un determinado factor:



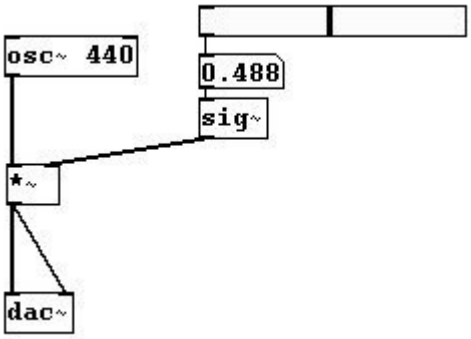
Ud. puede usar este método para configurar el volumen a cualquier nivel, desde el silencio absoluto hasta la mayor intensidad posible (la cual depende de los altoparlantes y del amplificador que esté utilizando).



También puede agregar un deslizador utilizando el Objeto de interfaz gráfica de usuario HSlider (**Put > HSlider**) y configurando su rango de 0 a 1 (cf. Capítulor 2.2.2.3.2 y 2.2.4.3.4):

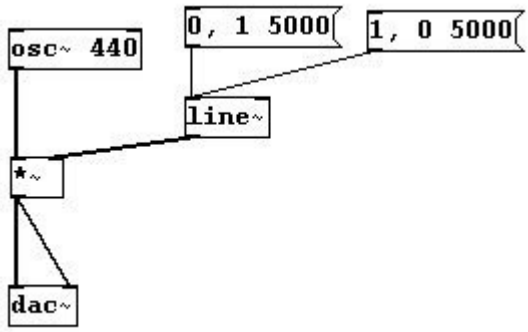


Sin embargo, si movemos rápidamente el deslizador obtendremos un sonido entrecortado. Esto se debe a que las señales (calculadas en muestras) colisionan copn los procesamientos de control (calculados en milisegundos). Si esto es realizado como describimos anteriormente, utilizando factores como 0, 0.1, 0.4, 0.7 o 1, se convierte en un asusnto irrelevante, pero más allá de cierta velocidad de cambio puede jugar un rol significativo. Para resolver este problema, debemos reemplazar la conexión de control con un Objeto de señal. Podemos usar el Objeto "sig~" para dicha conversión:

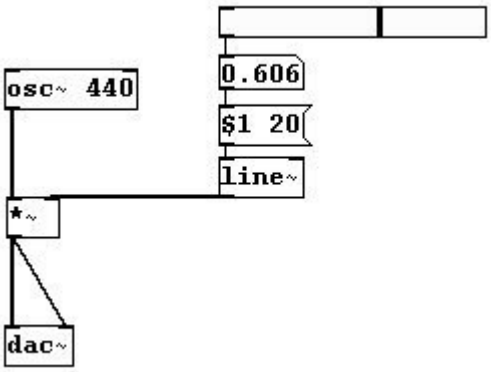


Esto asegura la sincronización entre los números generados por el oscilador (44100 números/segundo) y aquellos generados por el factor ("sig~" los convierte en 44100 números/segundo exactamente). Note bien: si una señal es conectada a la entrada derecha del Objeto "*~" , el mismo no debe llevar un argumento (como 0.5, usado anteriormente). El Objeto asumirá que en su entrada derecha ingresará datos de control.

Para crear un crescendo o un decrescendo, debe utilizar "line~":

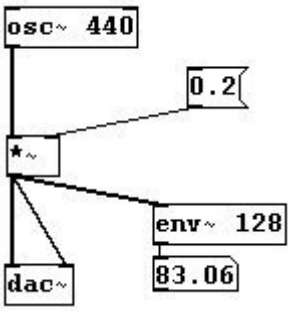


Aquí tiene una manera elegante de usar un deslizador como un regulador de volumen ("Potenciómetro"):

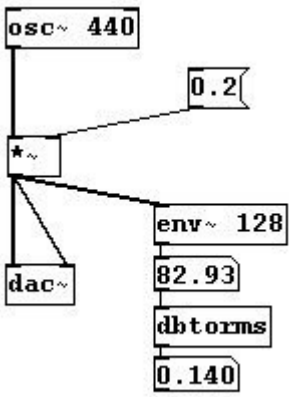


Este patch ejecuta un pequeño crescendo/decrescendo en cada paso. Llamamos "interpolación" al llenado de pasos con valores intermedios (como hemos visto con las alturas en el Capítulo 2.2.3.2.3).

Puede calcular el volumen de un sonido dado utilizando "env~", el cual devuelve como salida el volumen en dB. Siempre debe definir una duración de tiempo para poder calcular el valor promedio de volumen; el argumento será dado en muestras (se usa generalmente para este número alguna potencia de 2):

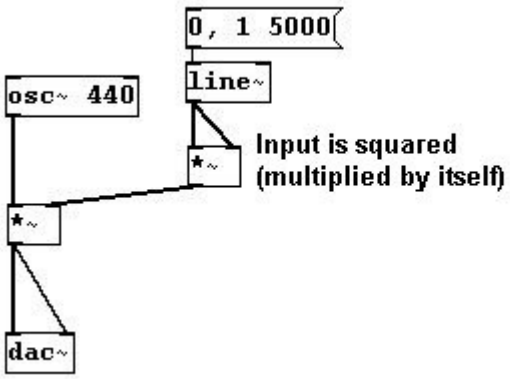


La conversión en RMS ...



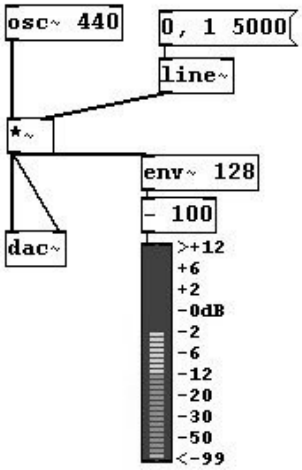
... deja en claro que los factores entre 0 y 1 no deben ser confundidos con los valores RMS entre 0 y 1.

Como hemos mencionado anteriormente, las percepciones aurales humanas de volumen y altura no corresponden con las mediciones llevadas a cabo por la física (observación hecha con los diagramas que vinculan alturas con frecuencia e intervalo). Un truco sencillo para crear crescendos o decrecendos más lineales es potenciando los valores:



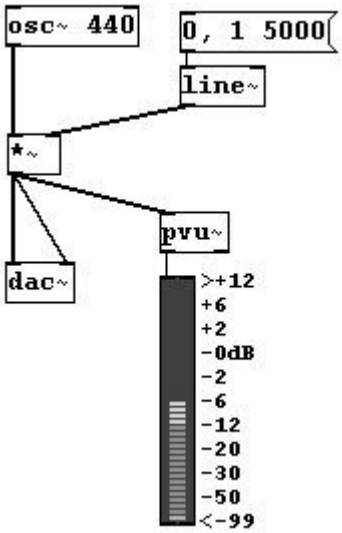
Sin embargo, uno debería intentar probar otras posibilidades. En definitiva, la manera en que aumentamos o disminuimos el volumen es una decisión composicional. Lo que constituye exactamente una "octava de volumen" no puede ser objetivado de la misma manera que la altura.

Existe en Pd un Objeto de Interfaz Gráfica de Usuario para visualizar la amplitud: el vúmetro (VU) (**Put** > **VU**). Toma como entrada un valor dB. Sin embargo, trabaja como una meza de mezcla tradicional: los 100 dB son mostrados como 0 dB y las variaciones por encima o debajo del mismo son presentados en el rango positivo o negativo respectivamente. Debe tener en cuenta esto cuando ingresa el valor de entrada. Simplemente reste de la salida del Objeto "env~":



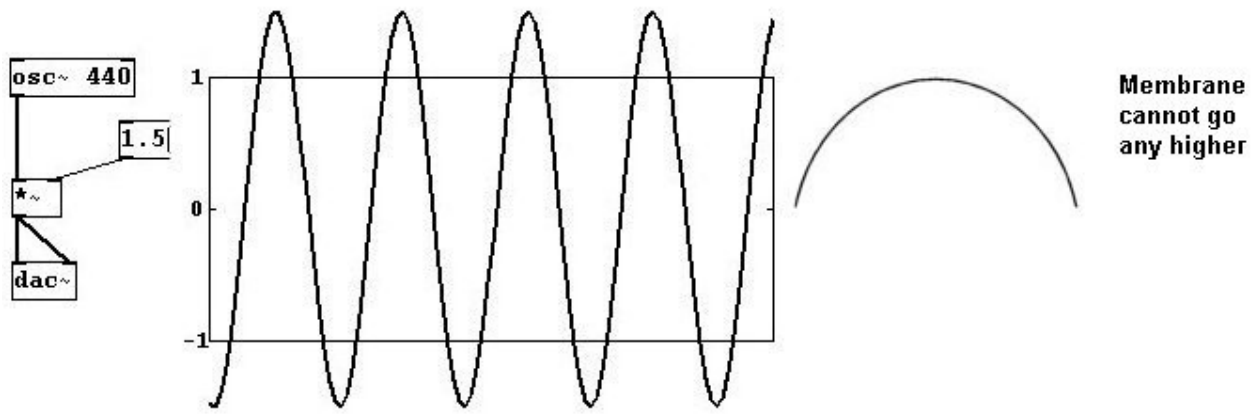
Entonces tendrá que el VU muestra los cambios en volumen de manera gráfica. (VU la abreviación de "volumen").

En Pd-extended, también puede usar el Objeto "pvu~" para la conversión del vúmetro:

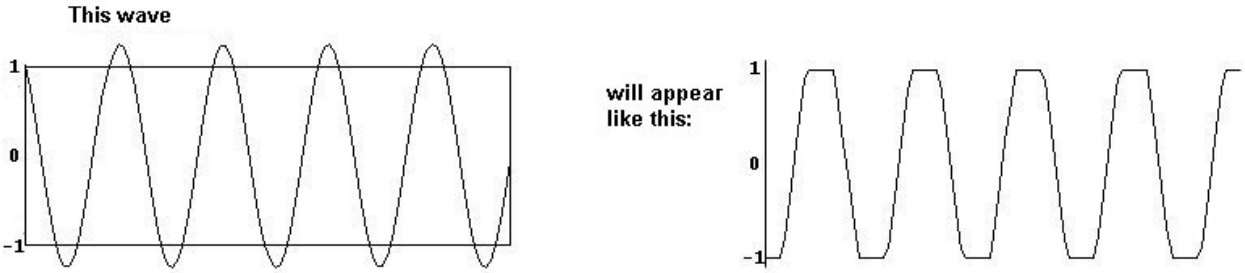


3.1.2.1.2 Problemas

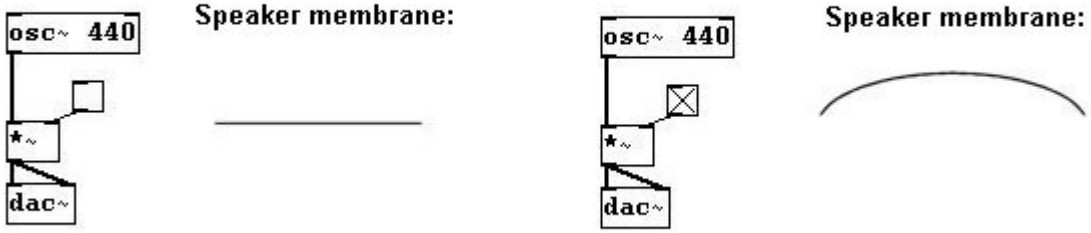
Otra cuestión importante: amplitudes superiores a 1 o inferiores a -1 serán cortadas ('clipped'). Si el "dac~" envía al altoparlante un valor más allá del rango de 1 a -1, la membrana simplemente se quedará en el extremo más lejano.



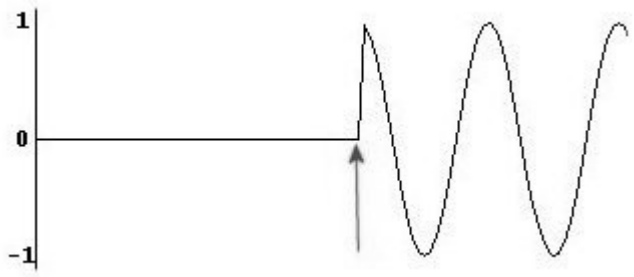
El incremento del volumen de un sonido al punto de recorte ('clipping') resulta en un efecto llamado *sobremarcha* (overdrive).



Tenemos otro problema cuando la membrana del altoparlante tiene que realizar un salto de gran distancia inmediatamente (por ejemplo, cuando se activa un sonido); el resultado es un *clic* ("click"):

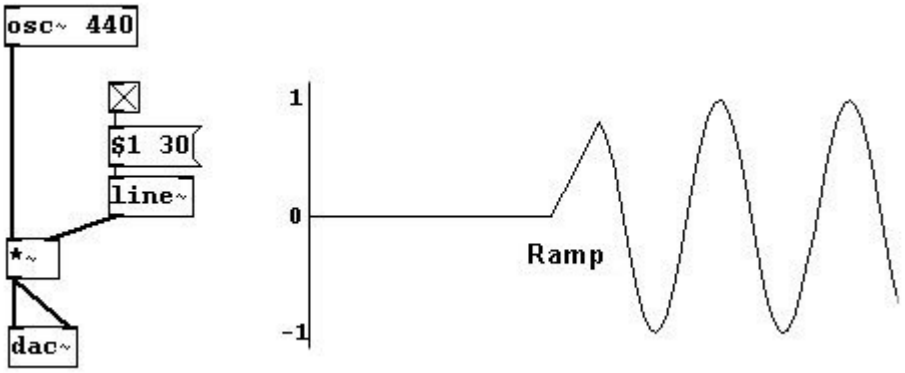


Este resultado es especialmente notable cuando el sonido mismo exhibe un movimiento de membrana muy suave, como por ejemplo con un tono sinusoidal. Dicho "tumbo" es fácil de ver en la siguiente ilustración:



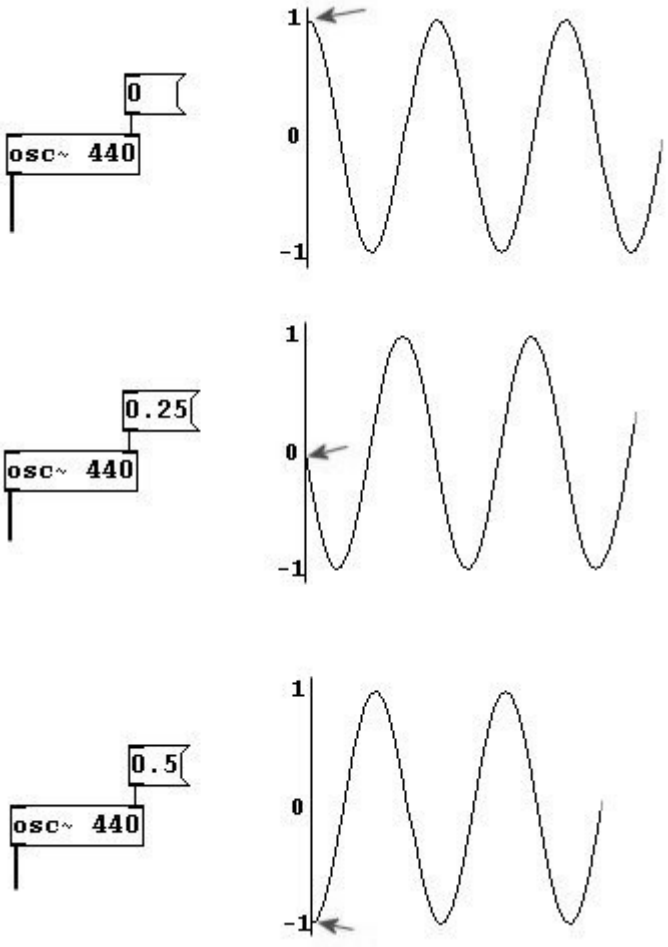
Un "tumbo" es generalmente un movimiento más rápido que 30 ms. Para evitar dicho clic, es

necesario contruir lo que se denomina "rampa", un crescendo muy rádido al principio y final:



3.1.2.1.3 Fase

En Pd, también podemos configurar la posición de la membrana para que comience (o donde debiera saltar) una onda sonora. Llamamos a esto fase de una onda. Puede configurar la fase en Pd mediante la entrada derecha del Objeto "osc~" con números cuyo rango se extiende desde 0 a 1:

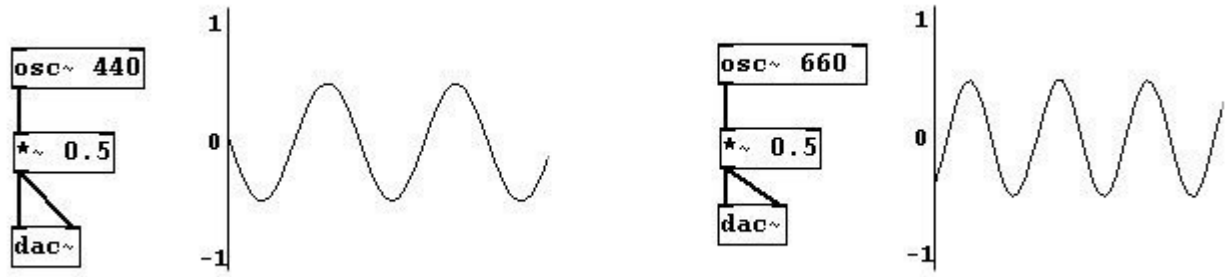


Un período entero de una onda abarca el rango de 0 a 1. Sin embargo, es frecuente hablar en terminos de grados, donde el período completo tiene 360 grados. Hablamos generalmente, por ejemplo, de un "cambio de fase de 90 grados". En Pd, la entrada para la fase sería de 0.25.

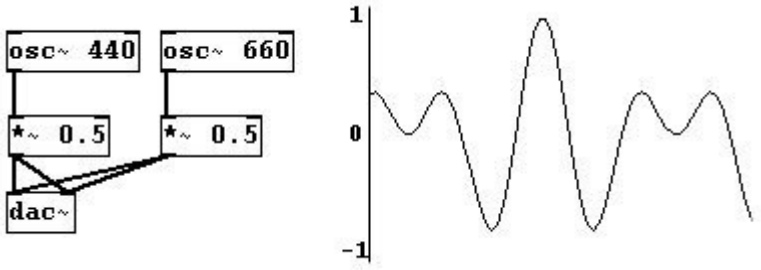
Un cambio de fase no tiene mucho efecto en la escucha. Sin embargo, regresaremos a este concepto más tarde.

3.1.2.1.4 Las ondas sonoras son aditivas

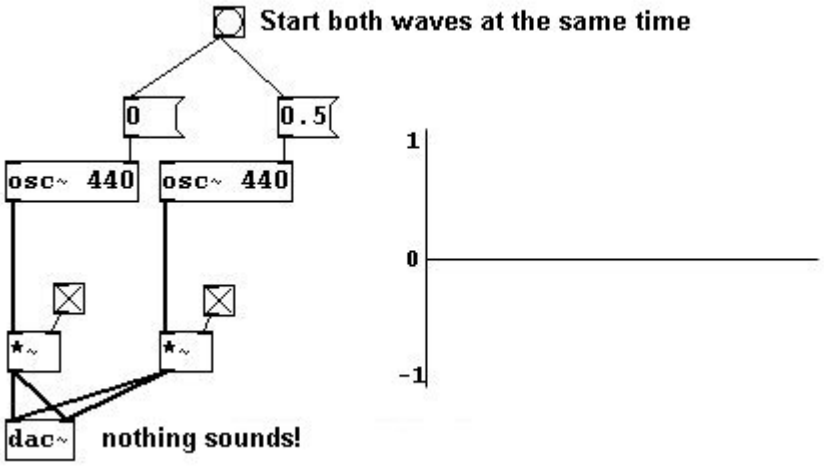
Digamos que tenemos dos osciladores:



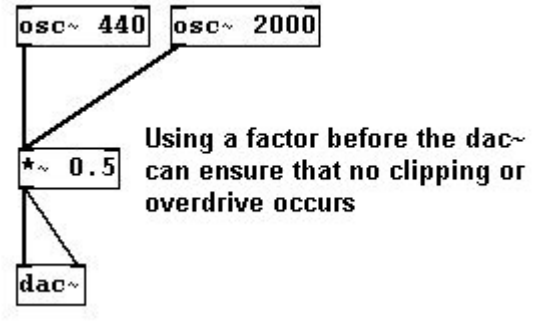
... y los conectamos a un "dac~". Obtendríamos lo siguiente:



Debido al factor multiplicativo, las ondas individuales de dichos osciladores sólo se extienden de -0.5 a 0.5. Sin embargo, juntas cubren un rango que se extiende de -1 a 1 y obtenemos como resultado una onda de forma más compleja. Esto se debe a que las ondas sonoras son *aditivas*. Simplemente: todas las vibraciones ocurren en el mismo medio, el aire. Esta cualidad aditiva también implica las cancelaciones. Ondas opuestas, donde una se "mueve hacia atrás" mientras que la otra se "mueve hacia adelante", se cancelan entre sí. Esto ocurre cuando las dos vibraciones de igual frecuencia se encuentran una respecto a la otra en una oposición de fase de 180 grados:



Cuando muchas fuentes sonoras se encuentran involucradas, generalmente tenemos que multiplicar el sonido total por un factor adecuado para evitar exceder los límites de -1 y 1:



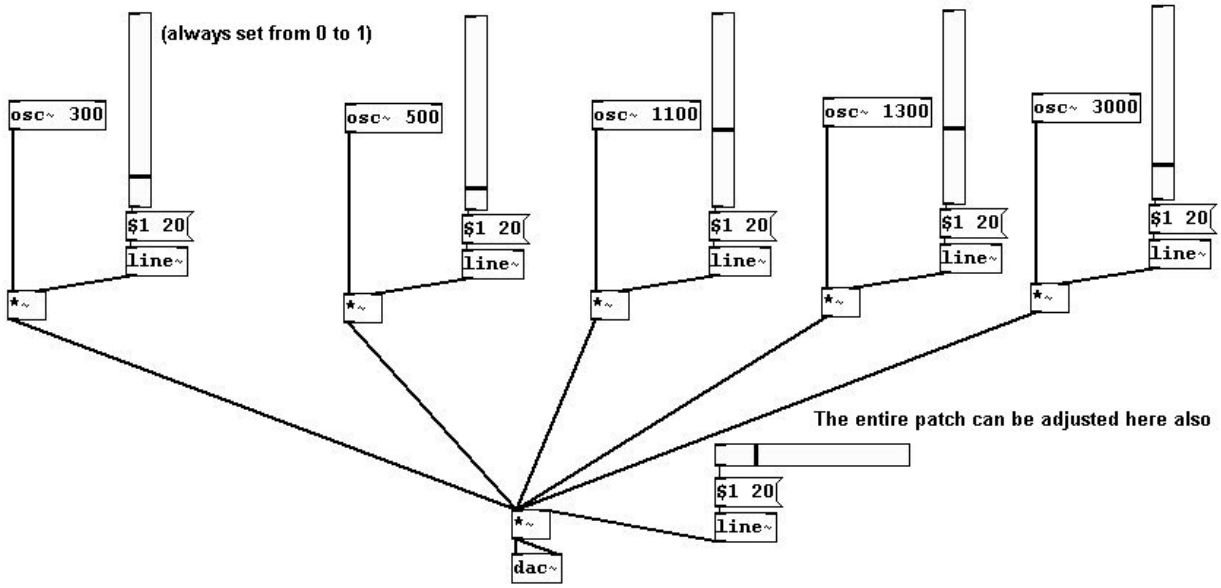
En este caso, ambos osciladores son simplemente conectados a un Objeto de multiplicación. Este automáticamente los suma (siempre que varias señales sean dadas a un objeto como entrada, primero son sumadas, luego procesadas de acuerdo al objeto) antes de llevar a cabo la multiplicación.

3.1.2.2 Aplicaciones

3.1.2.2.1 Acorde

Crear un acorde con volumen variable para cada tono del mismo:

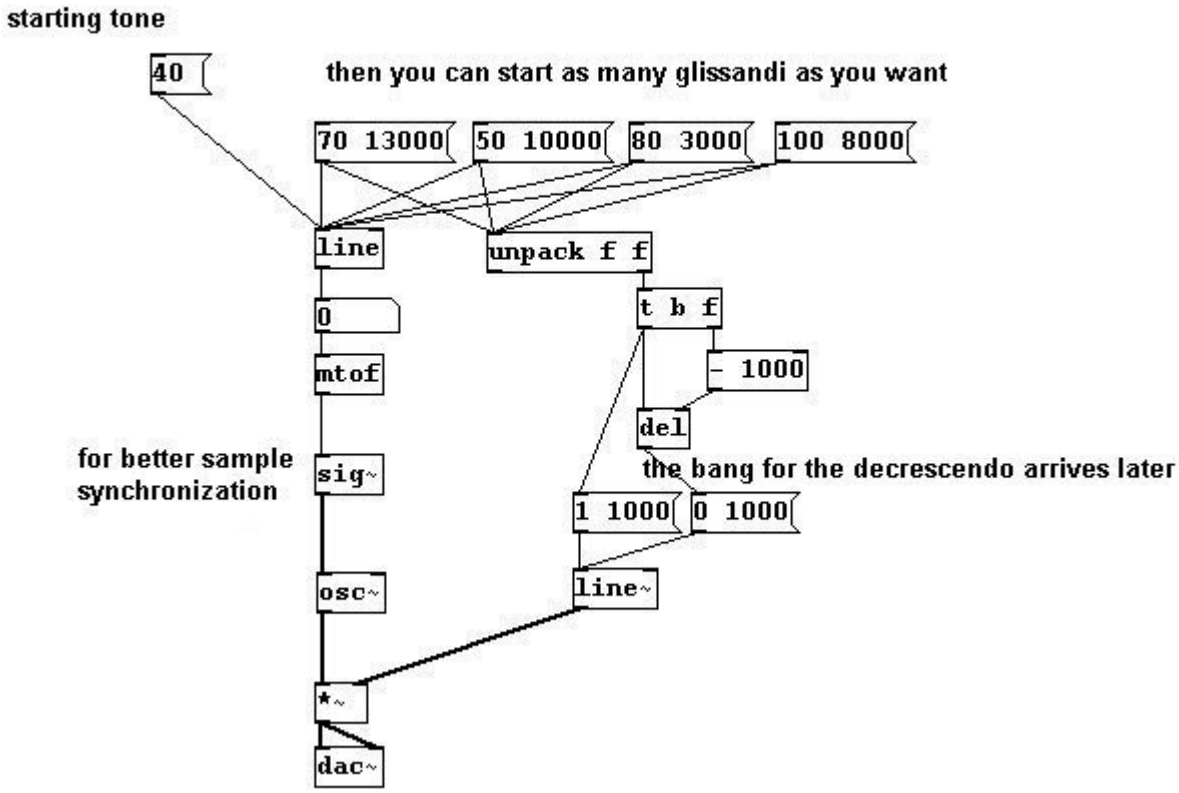
3-1-2-2-1-chord.pd



3.1.2.2.2 Glissando

Glissando con fundido inicial y final suave:

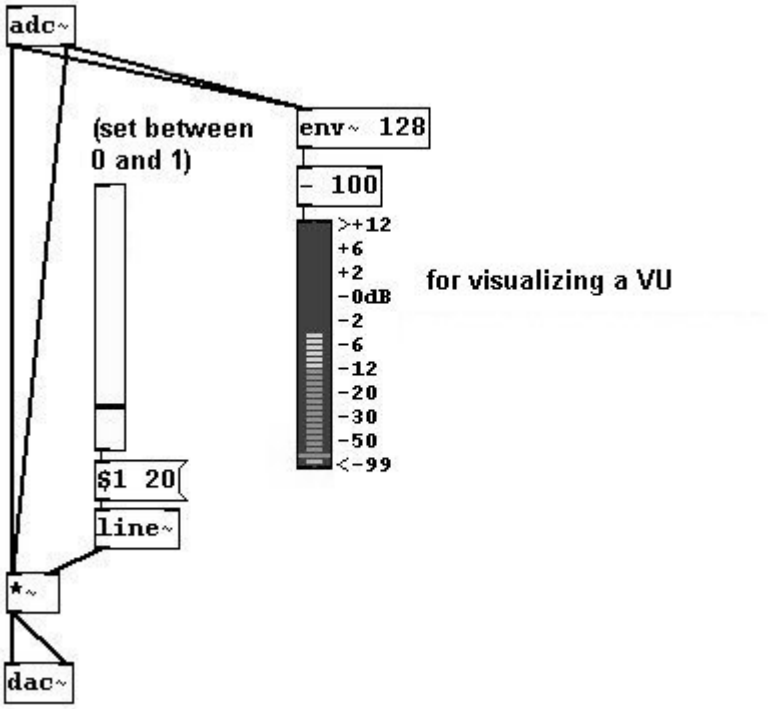
patches/3-1-2-2-2-glissandi-dim.pd



3.1.2.2.3 Procesamiento de entrada adc

Diga algo al micrófono y reproduzcalo con un volumen cambiado:

patches/3-1-2-2-3-edit-input.pd

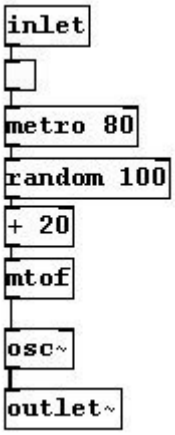


3.1.2.2.4 Concierto oscilador

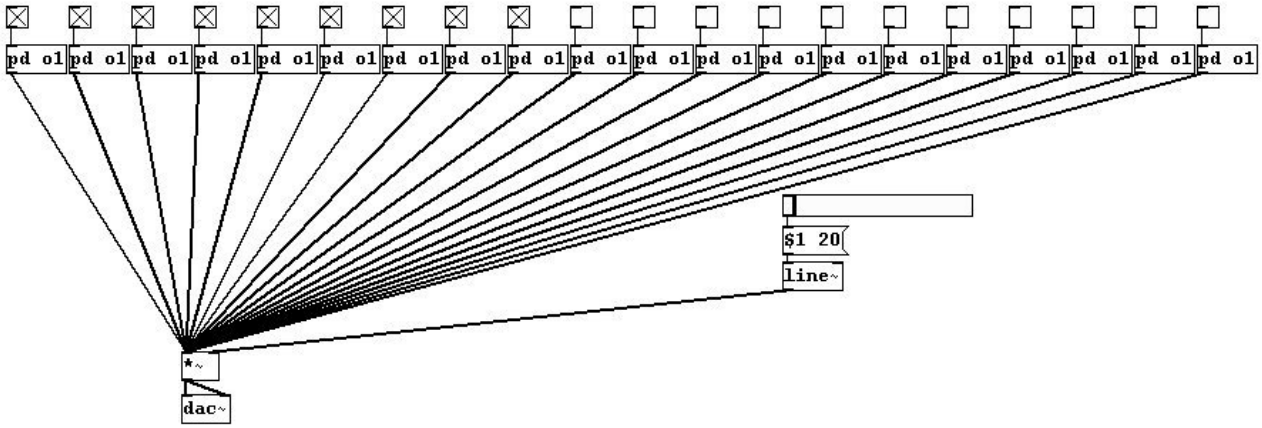
Volvamonos 'sinfónicos': ¿por qués no utilizar 20 osciladores a la vez?

patches/3-1-2-2-4-oscillatorconcert1.pd

Primero haga el subpatch "o1":



...luego realice múltiples copias...

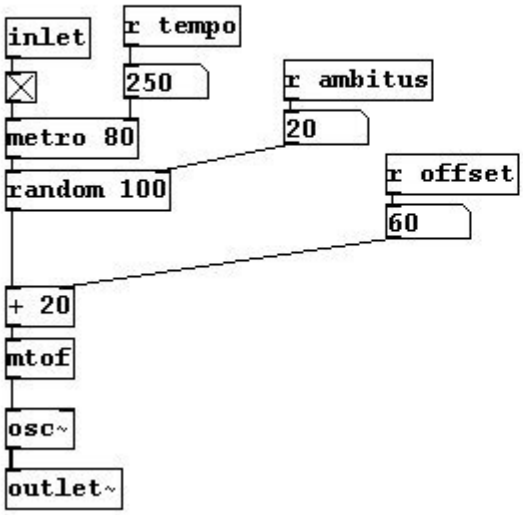


...¡por último activelas todas!

Por supuesto, también pueden se ajustados los parámetros para cada oscilador – de esa manera tiene algo con qué jugar:

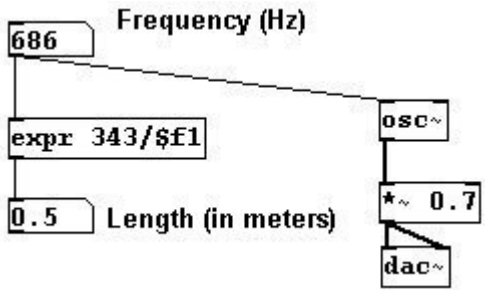
patches/3-1-2-2-4-oscillatorconcert2.pd

in every subpatch:



in the main patch:

La velocidad del sonido a 20 grados Celsius (68 grados Fahrenheit) es de aproximadamente 343 metros por segundo. Puede calcular el lardo de un periodo en el espacio y consultar el resultado inmediatamente...



...moviendo su cabeza medio metro hacia adelante y hacia atrás mientras escucha una frecuencia de 686 Hertz: puede escuchar claramente el pico de la onda y su valle.

3.1.2.2.5 Más ejercicios

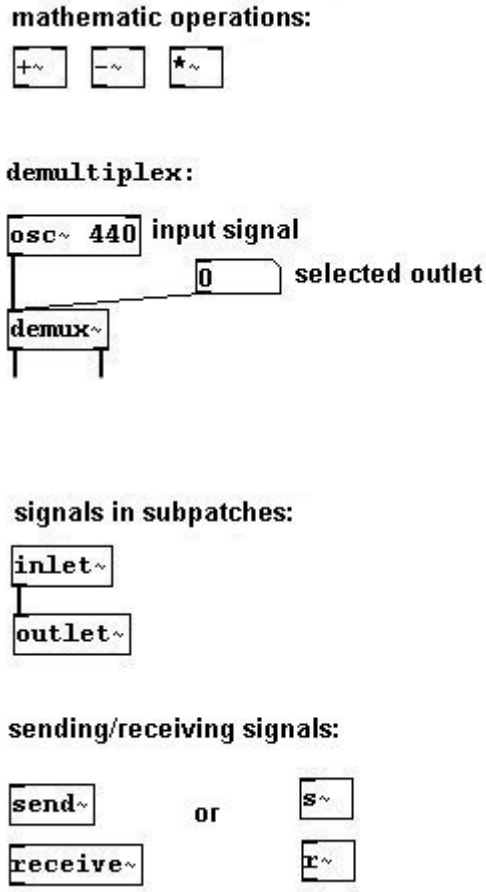
a) Crear acordes glissando (aleatorios) que también tengan cambios de volumen aleatorios en cada tono individual.

b) Crear un patch en el cual el volumen de la entrada de un micrófono controle la altura de un oscilador (¡luego utilice muchos, cada uno con una compensación (offset) distinta!)

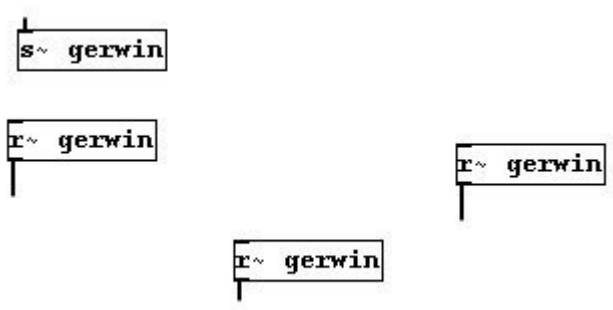
3.1.2.3 Apéndice

3.1.2.3.1 Otros Objetos "virgulilla"

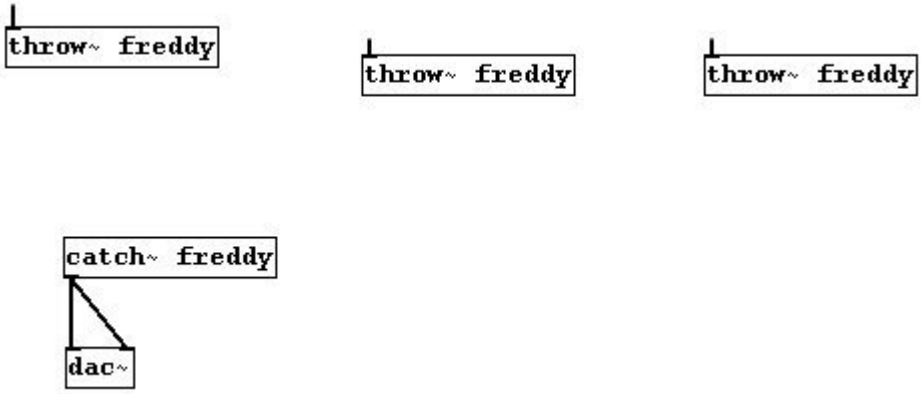
Muchos de los Objetos estudiados en el Capítulo 2 también tienen una versión con virgulilla. Trabajan de la misma manera, excepto que funcionan con datos de señales en vez datos de control:



Puede utilizar "send~" con tantos Objetos "receive~" que quiera; sin embargo, sólo puede emplear un sólo Objeto "send~":



También puede canalizar muchas señales diferentes a una localización central (por ejemplo, el "dac~") mediante el uso de "throw~" y "catch~":



3.1.2.3.2 Profundidad de bits

La profundidad de bits es también otro concepto importante en Pd. El procesador del ordenador sólo trabaja con código binario, con 0 y 1. El número de bit muestra cuantos lugares son usados para los ceros y unos. Si sólo tiene dos lugares, podría hacer 2^2, es decir, 4 combinaciones diferentes:

```
0 0
0 1
1 1
1 0
```

Mientras más lugares tenga, más detallado resultará el proceso que desea realizar. Para Pd, el cual usa números para calcular frecuencias, amplitudes, etc., significa que los números pueden ser procesados más precisamente, es decir, más lugares decimales pueden ser utilizados. Pd normalmente trabaja con 16 bit, el cual representa la misma calidad que un CD de audio. 16 bit significa $2^{16} = 65536$ valores posibles para cada muestra.

3.1.2.4 Para aquellos interesados especialmente

3.1.2.4.1 Presión sonora vs. Intensidad sonora

El volumen y, más importante aún, los incrementos de volumen -en términos tanto objetivos como subjetivos- son influenciados severamente por factores tales como cualidade acústica de la sala, edad del oyente, etc. No existe una única, precisa forma de medición del volumen, aunque existen teorías sobre la presión sonora y la intensidad sonora. Para más información, recomendamos la consulta de libros sobre acústica.

3.1.2.4.2 Datos de control vs. señales

Habrá notado que para partes significantes de la producción de sonido en Pd, utilizamos dos unidades diferentes: frecuencia y números MIDI para la altura, RMS y decibelios para la amplitud, y milisegundos y muestras para el tiempo.

Para el último de ellos, el ejemplo de la utilización de "línea ~" para crear un crescendo/decrescendo visto en la sección 3.1.2.1.1, debe explicarse con mayor detalle:

Si utilizase un Objeto "line" (sin virgulilla) para este propósito, podrían surgir sonidos indeseables

como "pops" o "clicks". Requeriría dos unidades de medida de tiempo diferentes para que se combinen; el problema es que no se sincronizan. Los diferentes intervalos numéricos no se igualarían, lo cual podría causar ciertas irregularidades sonoras como pequeños retardos o sonidos con clicks.

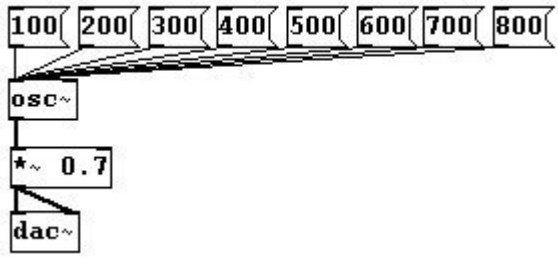
Como hemos descrito en el Capítulo 2.2.3.3.2, "line" genra un valor cada 20 milisegundos. Esto significa que podría no coincidir con las muestras. Aunque una nueva muestra aparezca cada 0.02 milisegundos, un valor de "line" podría no coincidir con una muestra más o menos simultanea, lo cual conduciría a complicaciones sonoras. Un Objeto "line~" (con virgulilla), sin embargo, genera una señal con 44100 valores por segundo. Estos 44100 valores son generados precisamente al mismo tiempo cualquier otro objeto virgulilla; siempre se encuentran sincronizados. El ordenador siempre procesa 44100 muestras por segundo sincrónicamente, independientemente de la posición en el patch.

3.2 Síntesis Aditiva

3.2.1 Teoría

3.2.1.1 La serie armónica

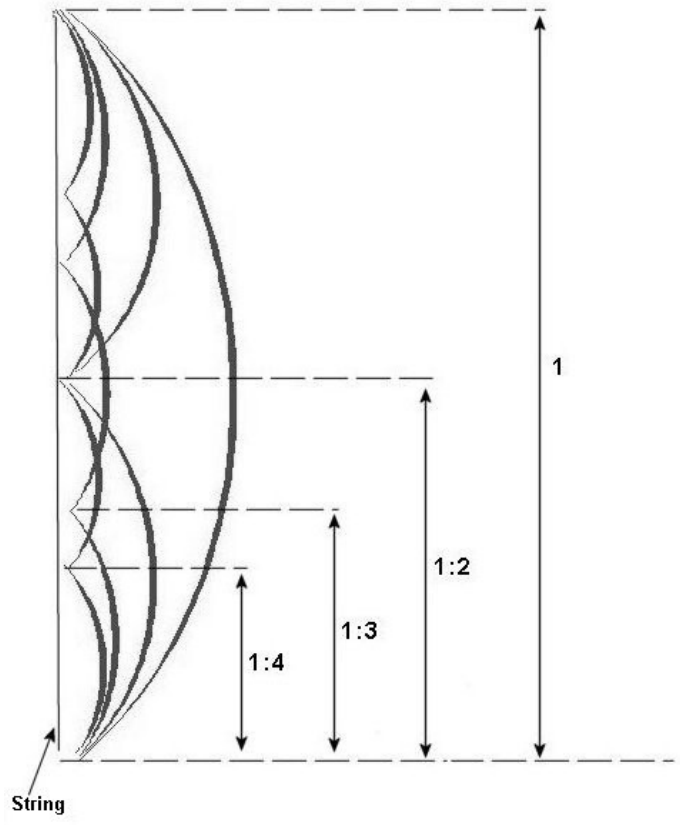
La serie aditiva de frecuencias (es decir, la serie que resulta de una simple adición del mismo valor en Hetz repetidamente), la cual produce una cadena de intervalos de tamaño decreciente, es llamada seie armónica:



También puede inferir la serie mediante la repetición de un experimento realizado por Pitágoras (ca. 570-510 a.C.) en el cual se divide a una cuerda en varias porporciones:

La proporción describe el largo de las dos partes de la cuerda en relación de la una con la otra.

Cuando tocamos una cuerda, no sólo vibra la misma como un entero, sino que también lo hace en cada proporción de número entero:

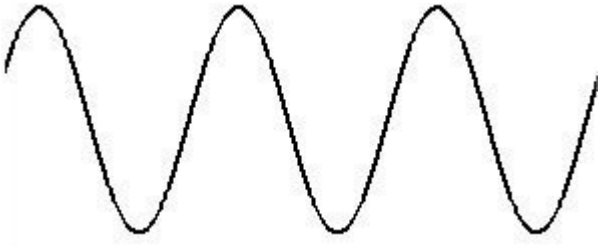


Las proporciones aquí describen la longitud de la sección vibrante en relación a la longitud de la cuerda completa.

¡Todas estas vibraciones parciales (llamados 'parciales' or 'armónicos') también suenan, por lo que cada sonido hecho en la cuerda podríamos entenderlo como un *acorde*!

Lo realmente hace especial este acorde es que todos sus componentes se fusionan. Cada sonido natural contiene hipertonos. Debido a características inherentes a la escucha humana, oimos todos estos componentes como un solo tono.

En cambio, los parciales superiores mismos (es decir, los parciales superiores al parcial fundamental) no contienen hipertonos. Un sonido aislado sin hipertonos no existe en la naturaleza, pero sí puede ser creado con medios electrónicos. Llamamos a estos tonos sinusoidales, un nombre proveniente del contorno de su forma de onda:

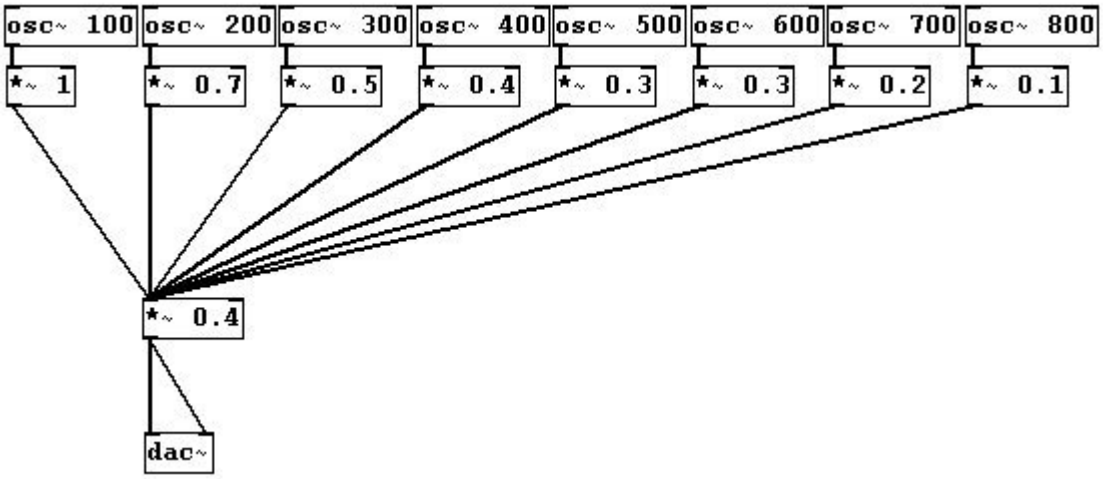


El físico Jean Baptiste Joseph Fourier (1768-1830) descubrió que cada sonido periódico puede ser representado utilizando sólo tonos sinusoidales (de diferente frecuencia, amplitud y fase), cuyas sumas resultan en el sonido original mismo. Dicho análisis y su correspondiente proceso matemático se llama análisis de Fourier y transformada de Fourier.

Utilizando este principio, es posible crear cualquier sonido periódico mediante la superposición de muchos tonos sinusoidales. Llamamos a este proceso "síntesis aditiva".

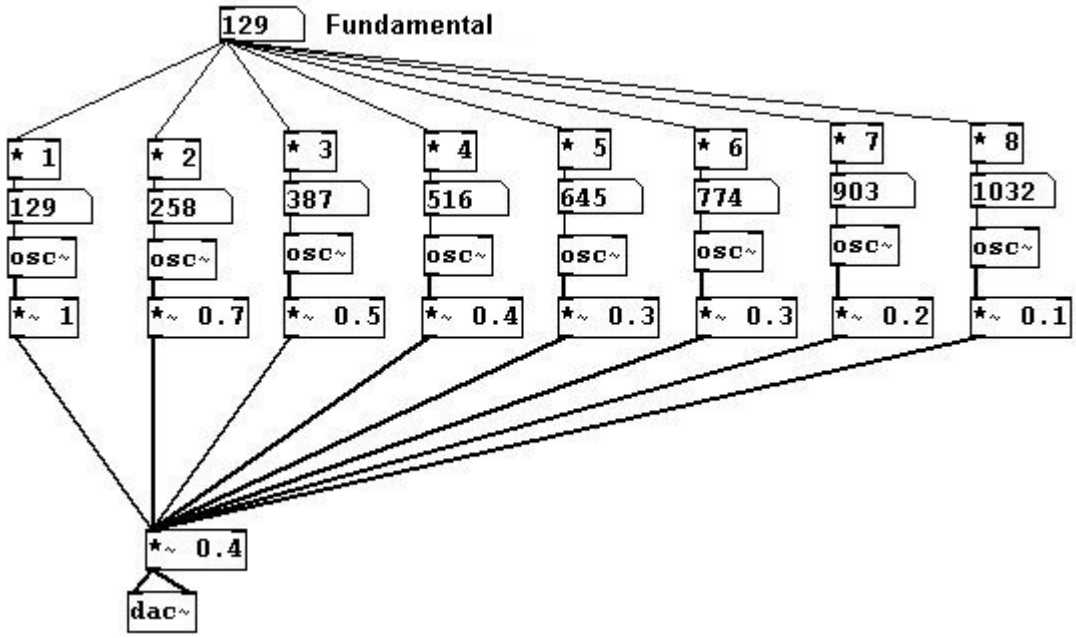
En Pd, como ya hemos mencionado, podemos utilizar "osc~" para generar un tono sinusoidal. Los tonos sinusoidales son sonidos muy característicos de la música electrónica, ya que son producidos y sólo pueden pueden ser producidos por medios electrónicos.

Puede crear un acorde basado en la serie de hipertonos usando un número de Objetos "osc~", cuyas frecuencias forman una serie aditiva:



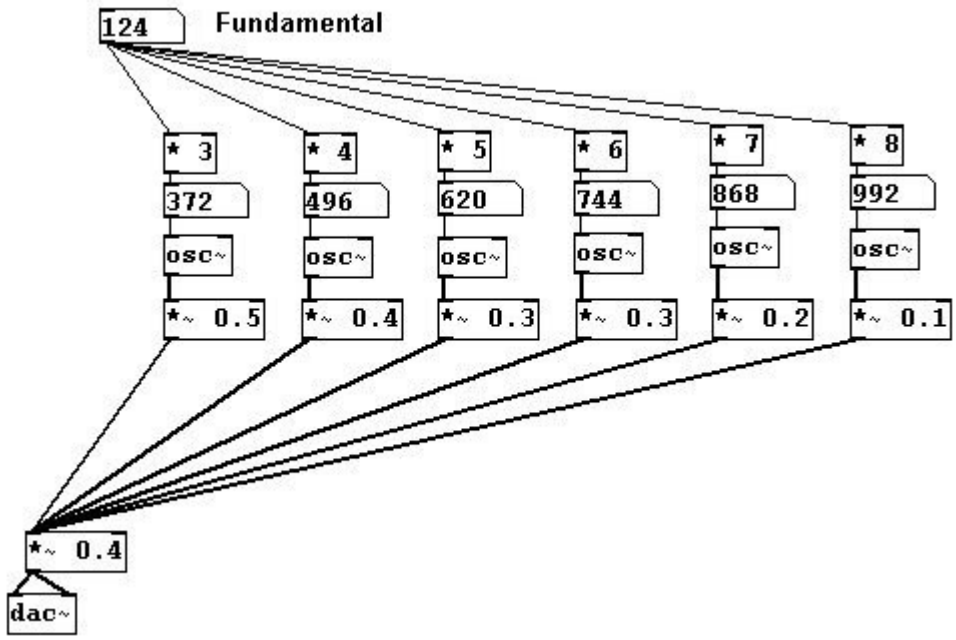
Generalmente, las amplitudes se reducen mientras que las frecuencias aumentan para que el acorde fusione mejor (aunque en algunos instrumentos, es característico que ciertos parciales sean más altos en intensidad que algunos otros, por ejemplo, el clarinete). La estructura y los volúmenes relativos de los hipertonos determinan el *color* del sonido. También llamado *espectro*.

El hecho de que nuestros oidos fusionen los hipertonos entre sí, se hace evidente cuando cambiamos la frecuencia fundamental:



Utilizaremos aquí los primeros ocho parciales. (Note Bien. El término 'parcial' incluye la fundamental mientras que el término 'hipertono' no lo hace. En otras palabras, el primer parcial es la frecuencia fundamental, el segundo parcial es el primer hipertono, el tercer parcial es el segundo hipertono, etc.)

Incluso si omitimos los parciales inferiores, escuchamos la frecuencia fundamental cuando cambiamos la fundamental:



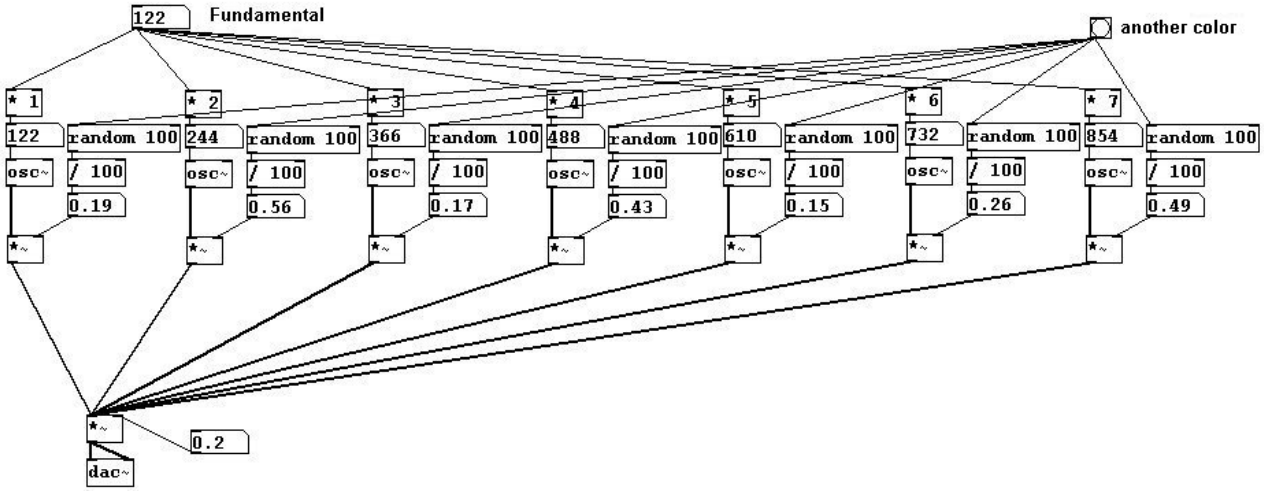
Nuestro cerebro calcula la fundamental basado en el espectro restante. Este tono no existente es llamado *tono residual*.

3.2.2 Aplicaciones

3.2.2.1 Un klangfarbe aleatorio (Alemán: color sonoro)

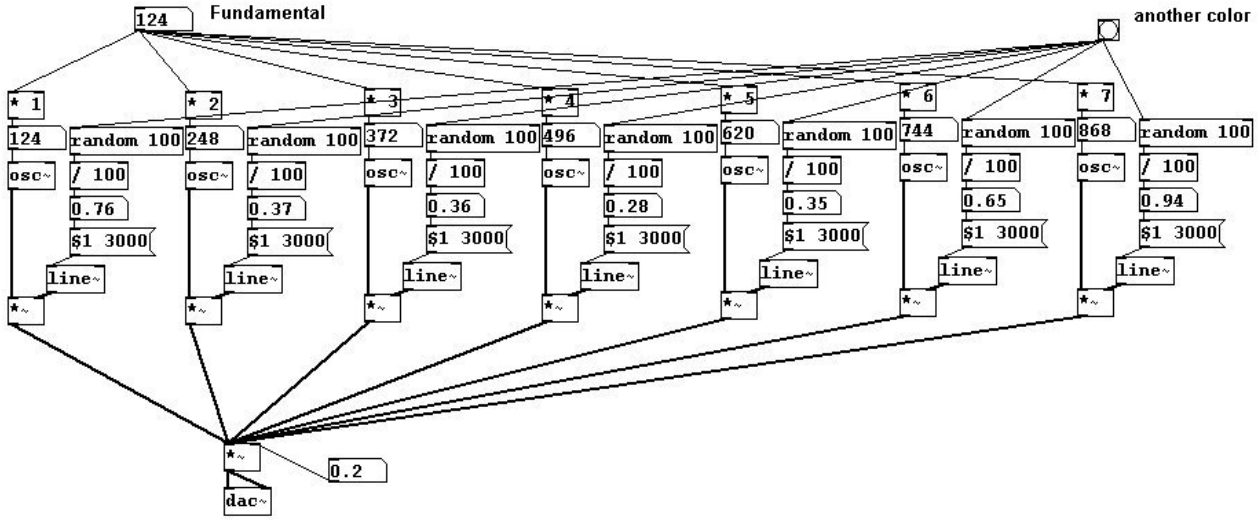
patches/3-2-2-1-random-color.pd

Por razones de espacio, hemos limitado este ejemplo a los primeros siete parciales:



3.2.2.2 Cambiando de un klangfarbe a otro

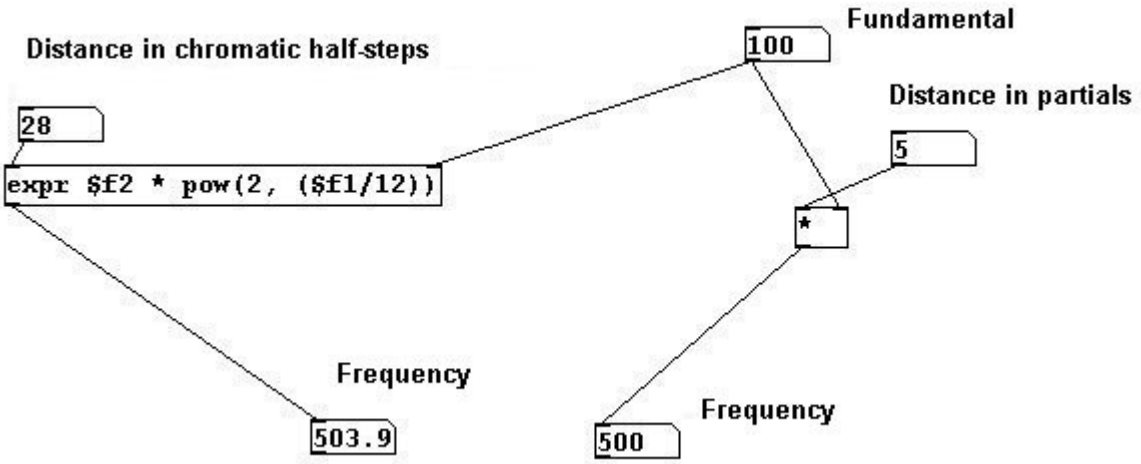
patches/3-2-2-2-colorchange.pd



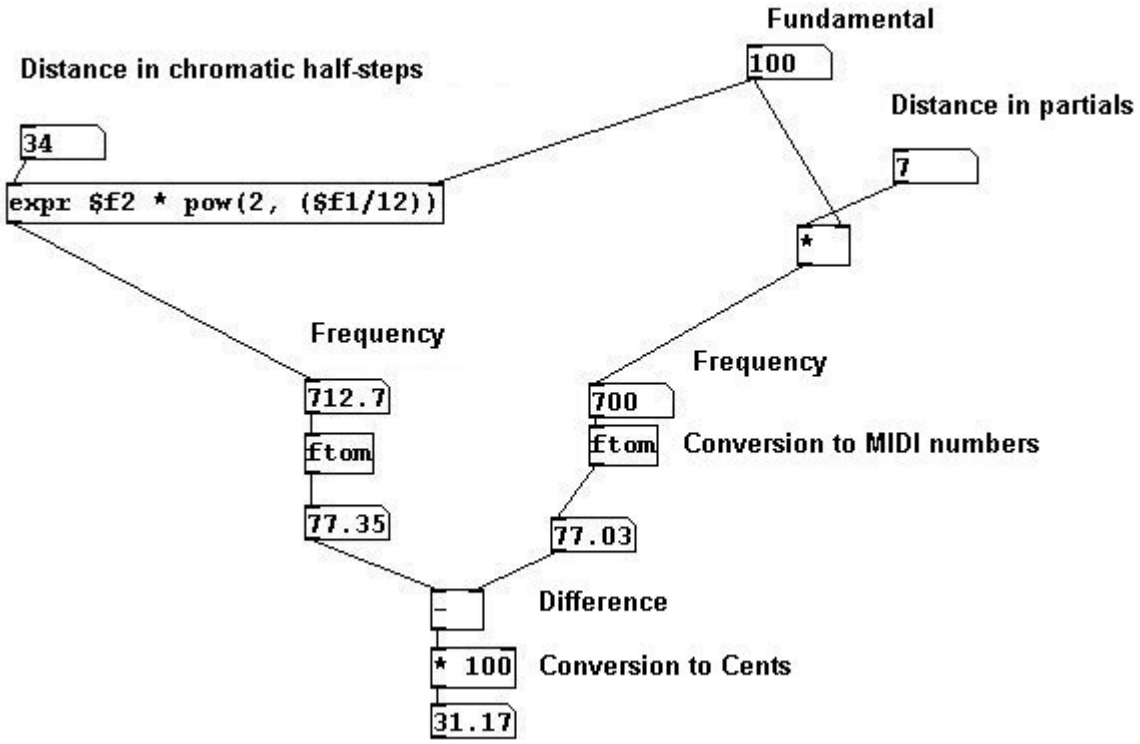
3.2.2.3 Afinación natural vs. Temperamento igual

Veamos la diferencia entre intervalos de afinación natural e intervalos de temperamento igual (¡primero ingresamos la frecuencia fundamental!):

patches/3-2-2-3-natural-tempered.pd



Muestra de la diferencia entre afinación natural y temperamento igual en cents (centésimas de un semitono):



Aquí puede ver: el séptimo parcial natural es 31 cents más bajo que una séptima de temperamento igual.

3.2.2.4 Más ejercicios

Cree un acorde de hipertonos con hipertonos manipulados, es decir, con hipertonos imprecisos.

3.2.3 Apéndice

3.2.3.1 Limitaciones de Pd

El ejemplo previo de klangfarbe aleatorio revela ona de las limitaciones de Pd: no podemos determinar aleatoriamente el número de osciladores. Debemos al menos determinar primero el

máximo.

3.2.4 Para aquello interesados especialmente

3.2.4.1 Studie II

Una de las piezas pioneras de la historia de la música electrónica es el 'Studie II' por Karlheinz Stockhausen, escrito en 1954. Este trabajo utiliza sólo tonos sinusoidales y mezclas de estos en intervalos no temperados. ¡El autor recomienda enfáticamente analizar esta pieza!

3.2.4.2 Composición con espectros

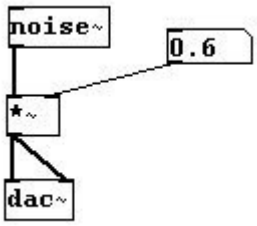
En el cuarto capítulo del libro "Audible Design", del compositor y teórico Trevor Wishart, se describe muchas posibilidades de composición con espectros.

3.3 Síntesis sustractiva

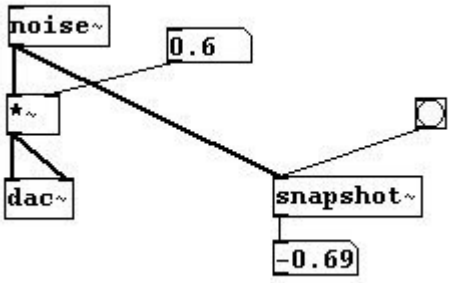
3.3.1 Teoría

3.3.1.1 Ruido blanco

Una vez Claude Debussy respondío a la pregunta de cómo componía diciendo que tomaba todas las alturas y luego suprimía las que no le gustaba. De esta manera vió la idea de filtrado. Opuesto a la síntesis aditiva -la cual usa como punto de partida lo que podría ser considerado como el 'átomo' del sonido, el tono sinusoidal-, la síntesis sustractiva comienza con todos los sonidos posibles y luego los reduce. Es realmente posible producir todos los sonidos. Hacer vibrar a la membrana de un altoparlante de manera completamente caótica y aleatoria produce como resultado todas las frecuencias audibles simultaneamente. El Objeto de Pd que se utiliza para realizar esto se llama "noise~":



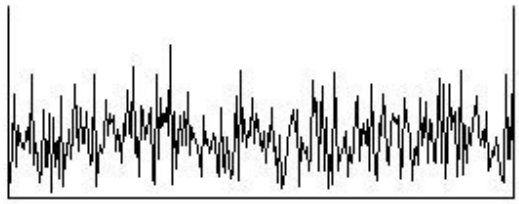
En términos técnicos, sería más preciso llamar al Objeto "noise~" como Objeto "random~", porque produce 44100 números aleatorios por segundo. Estos números cubren el rango de -1 a 1, es decir, las distintas posiciones probables de una membrana.



3.3.1.2 Filtros

Como la luz, llamamos "ruido blanco" al ruido que contiene todas las frecuencias audibles. La luz blanca ordinaria contiene todas las frecuencias de luz visible mientras que podemos derivar de la misma, digamos, luz roja o azul mediante la utilización de filtros.

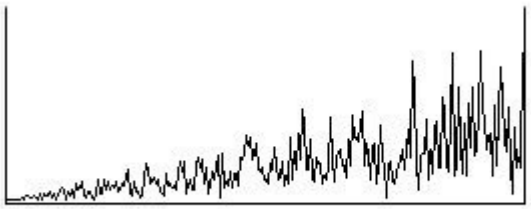
Pd incluye filtros tales como un pasa-bajos ("lowpass"), el cual sólo permite el paso de las frecuencias graves mientras suprime las altas frecuencias. Podemos ver esto representado en el siguiente diagrama; el eje x representa la frecuencia mientras que el eje y la amplitud:



white noise

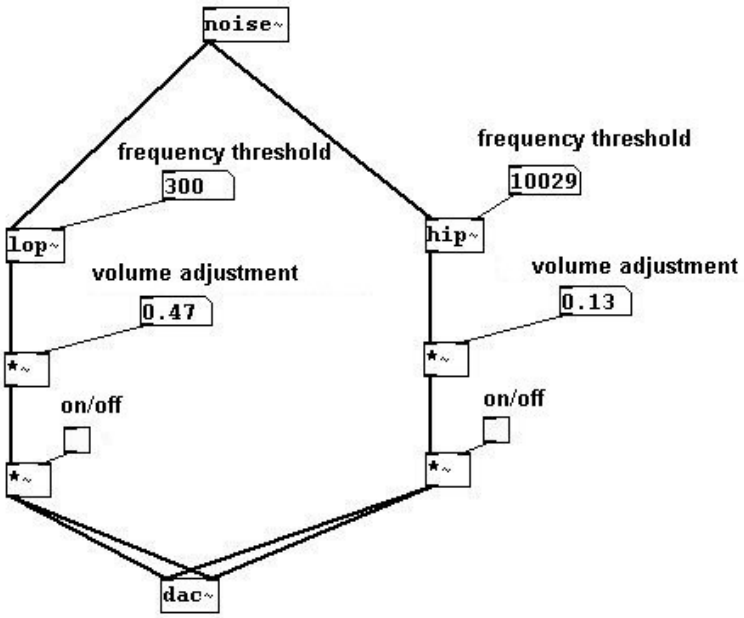
white noise with low-pass filter

También disponemos de un filtro pasa-altos ("highpass"), el cual permite sólo el paso de frecuencias altas:

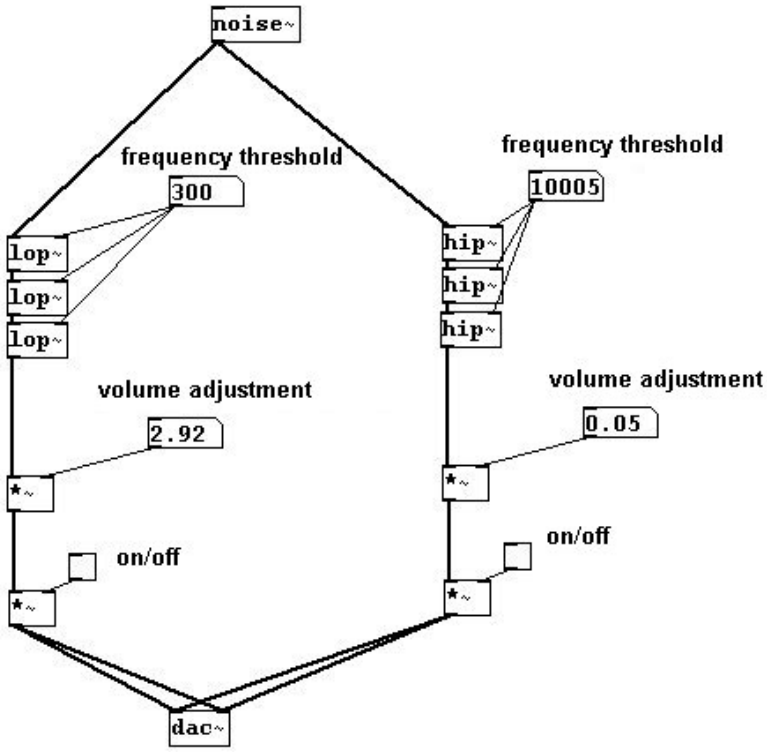


white noise with high-pass filter

Los Objetos de Pd para dichos filtros se llaman "lop~" y "hip~", respectivamente. Su argumento o entrada derecha, corresponde a la frecuencia a partir de la cual el sonido debe ser filtrado.

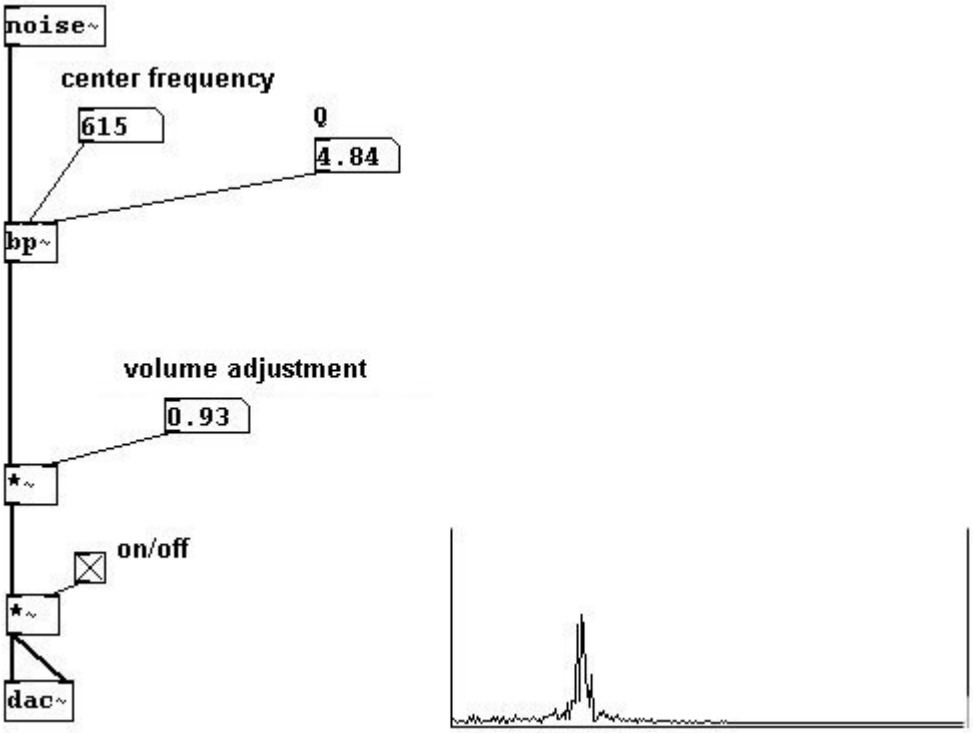


Como es posible ver en los diagramas precedentes, estos filtros no son muy 'escarpados'. Sin embargo, podemos intensificar el efecto de corte utilizando varios filtros uno luego del otro (filtros en cascada):

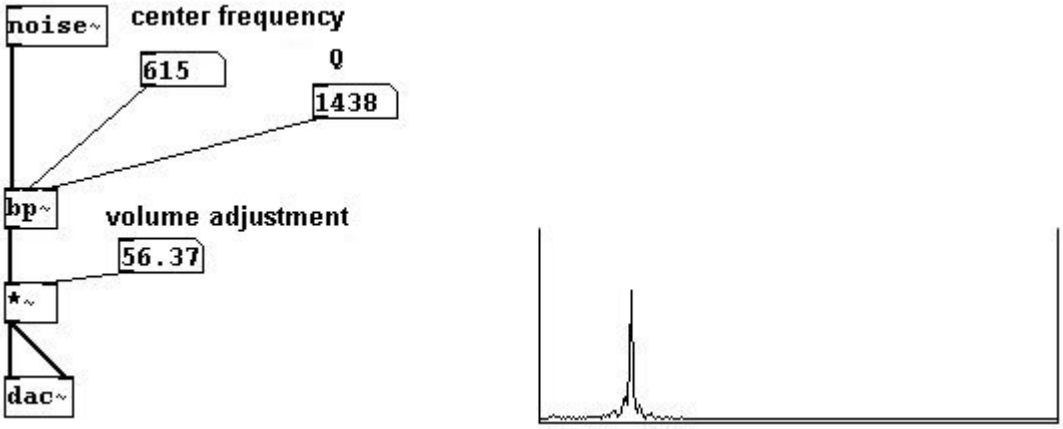


El volumen debe ser reajustado en cada filtro, ya que reducen la intensidad del sonido (aunque a veces fortalecen otras cosas).

Otro tipo de filtro disponible es el pasa-banda ("band-pass"). Este permite pasar sólo una pequeña porción de sonido que circunda una frecuencia central, una 'banda' de frecuencias. Como argumentos/entradas recibe la frecuencia central y el ancho de banda, llamado "q".



Teóricamente, si reducimos considerablemente la banda, deberíamos terminar con un sólo tono sinusoidal:



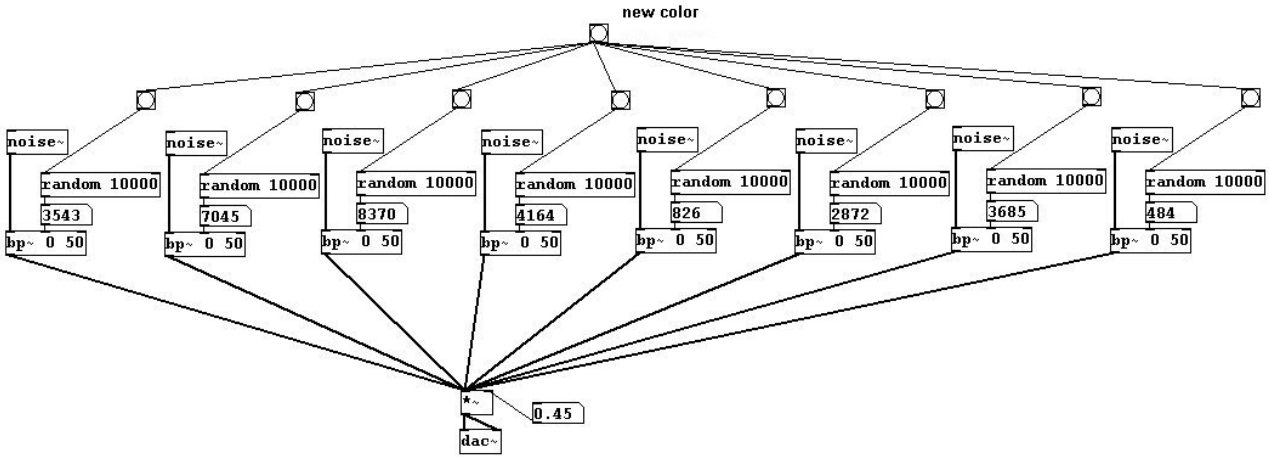
Sin embargo, como puede escuchar facilmente, este no es el caso. Siempre resta un pequeño componente de ruido al utilizar un filtro pasa-banda.

3.3.2 Aplicaciones

3.3.2.1 Colores de filtro

Para ejemplificar cómo podemos utilizar los filtros, aquí presentamos una distribución aleatoria de filtros pasa-banda:

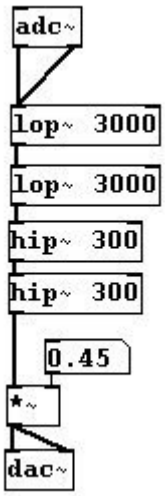
patches/3-3-2-1-filtercolors.pd



3.3.2.2 Telephone filters

Para la transmición de conversaciones telefónicas, se determinó el uso de un el rango de frecuencias que se extiende de los 300 a los 3000 Hz. Se considera que dicho rango es suficiente para comprender un parlamento. Podemos simular esto mismo:

patches/3-3-2-2-telephonefilter.pd



3.3.2.3 Más ejercicios

Experimento con filtrar la "orquesta glissando" (3.1.2.2.4).

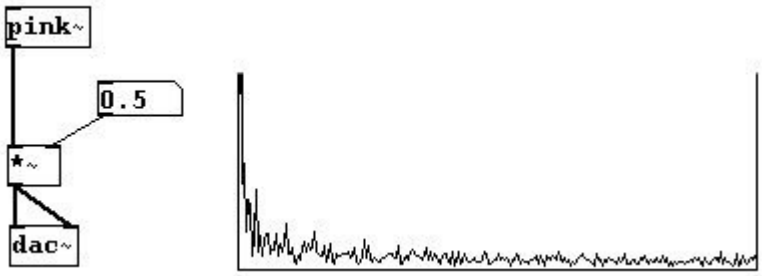
3.3.3 Apéndice

3.3.3.1 Ruido blanco y clicks

El Objeto "noise~" causa la vibración aleatoria de la membrana del altoparlante. Cuando activa y desactiva dicho sonido no escuchamos un click; esto ocurre porque el ruido está compuesto únicamente de clicks de variada intensidad. Entonces, no es necesario utilizar una "rampa" (cf. 3.1.2.1.2).

3.3.3.2 Ruido rosa

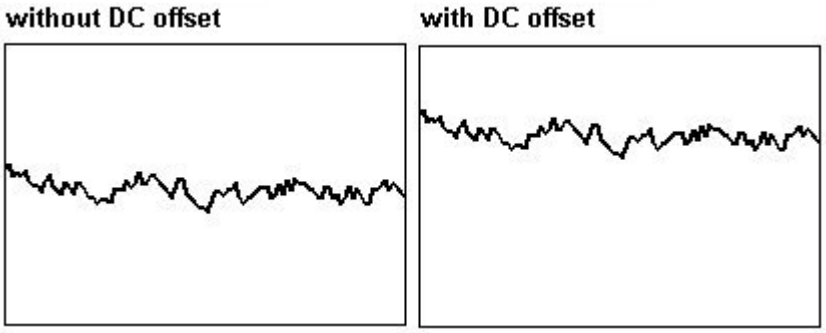
Además del ruido blanco, también existe el ruido "rosa". El oido humano no percibe todo el rango de frecuencias con el mismo volumen. Escucha mejor alrededor de los 2000 Hz, es por esto que el percibimos al ruido blanco como relativamente agudo. Escuchamos con menor intensidad los rangos graves y muy agudos. Si queremos crear un ruido en donde percibamos una distribución uniforme de todas sus frecuencias, debemos adaptarlo a la manera que oimos, es decir, las frecuencias graves tienen que ser más intensas que las frecuencias medias agudas. Esta distribución se llama ruido rosa y puede ser generada en Pd utilizando el Objeto "pink~":



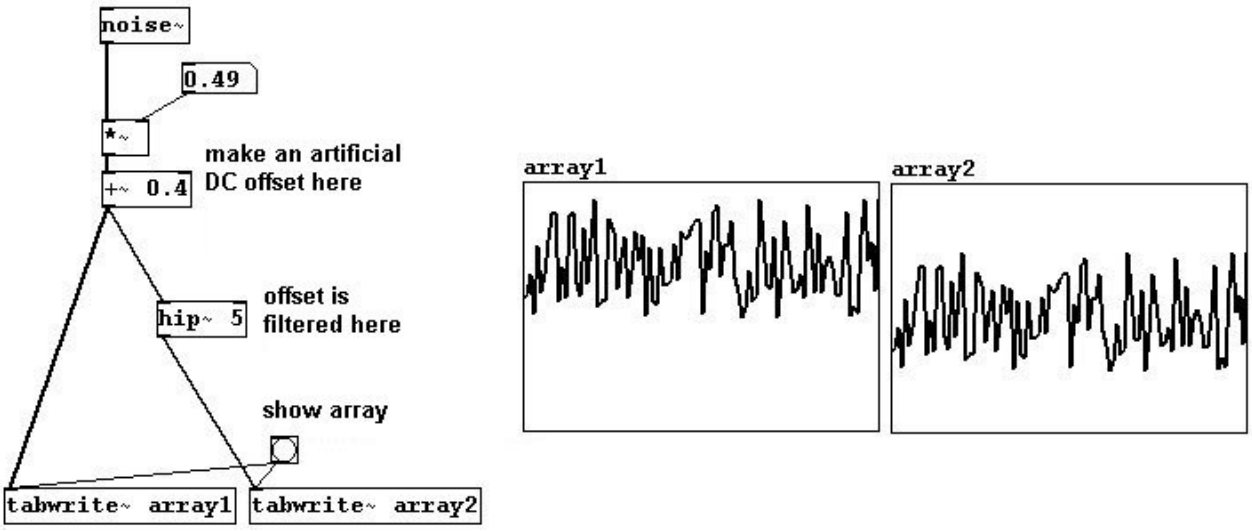
3.3.3.3 Desplazamiento del componente de continua (DC offset)

Cuando usamos un micrófono, la señal probablemente exhibirá una corriente de tierra desfazada.

Llamamos a esto desplazamiento del "componente de continua". El resultado es esta forma de onda:



Este desplazamiento se asemeja a una vibración infinitamente lenta con una frecuencia que se aproxima a 0. Dada su lentitud, puede ser removida con un filtro pasa-altos con una configuración extremadamente baja:

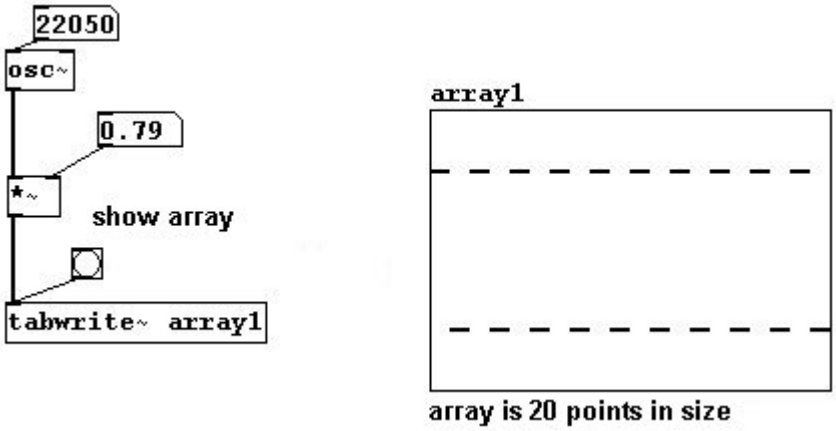


3.3.4 Para los interesados, especialmente

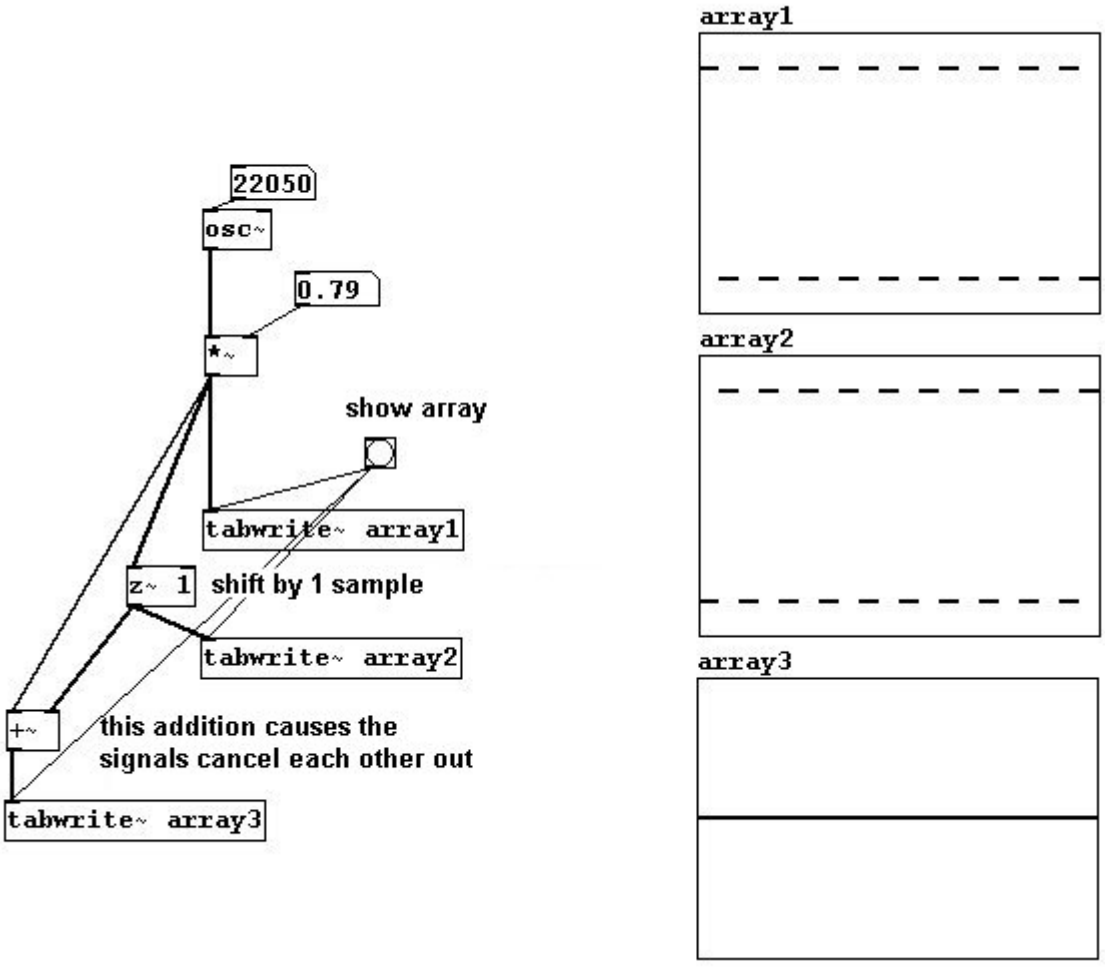
3.3.4.1 Cómo trabajan los filtros digitales

La 'vida interna' de los filtros digitales es compleja. Sin embargo, deberiamos hacernos una idea de cómo trabajan: como describimos en 3.1.1.3.1, una frecuencia de muestreo de 44100 Hz es capaz de representar una onda con un máximo de 22050 Hz. Esta onda tendría sólamente dos puntos por período:

patches/3-3-4-1-filterwork.pd



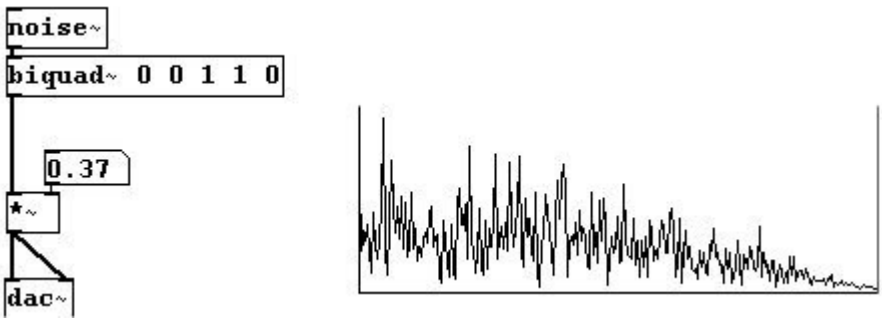
Si movemos esta onda sólo una posición, es decir, una sóla muestra adelante o atrás, y luego la sumamos a la onda original, el resultado será que ambas ondas se cancelan entre sí completamente. Este desplazamiento lateral puede ser llevado a cabo (en Pd-extended) mediante el Objeto "z~".



Los filtros digitales emplean este método de desplazamiento de una muestra de la onda y luego de sumatoria con la onda original para efectuar la cancelación. El Objeto "biquad~" puede ser utilizado para ajustarlo manualmente. Ejecuta la siguiente ecuación diferencial: y(n) = ff1 * w(n) + ff2 * w(n - 1) + ff3 * w(n - 2) with w[n] = x[n] + fb1 * x[n - 1] + fb2 * x[n - 2].

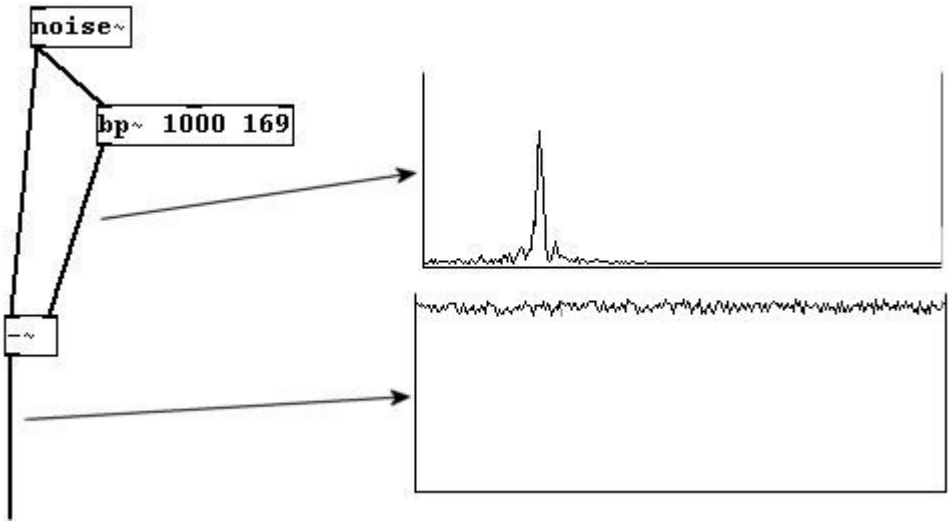
donde 'n' corresponde a la posición de la muestra, mientras que ff1, ff2, ff3, fb1, y fb2 corresponden a factores definidos libremente. En Pd, "biquad~" requiere cinco argumentos para ff1, ff2, ff3, fb1,

y fb2. La sintaxis es la siguiente: "biquad~" [fb1] [fb2] [ff1] [ff2] [ff3]. Para el caso de una onda de 22050 Hz como la mencionada anteriormente, podriamos también escribir "biquad~ 0 0 1 1 0". Esto suprime las frecuencias agudas, especialmente ondas con una frecuencia de 22050 Hz, las cuales se cancelan completamente. Aquí tenemos un filtro pasa-bajos utilizando "biquad~":



Puede utlizar la fórmula bicuadrática para crear mucho otros tipos de filtros. Por ejemplo, los argumentos 1.41407 -0.9998 1 -1.41421 1 corresponden a un filtro "rechazo de banda". Este es la inversa de un filtro pasa-banda; rechaza -es decir, bloquea- una banda de frecuencias en torno a una frecuencia central, en este caso 5512.5 Hz. La explicación para este tipo de cálculo nos llevaría a escribir un libro dedicado exclusivamente a esta temática. En Pd-extended existen Objetos ("band-pass", "equalizer", "highpass", "highshelf", "hlshelf", "lowpass", "lowshelf", "notch") que llevan a cabo estos cálculos. La ventaja del uso de un filtro biquad es que posibilita perfiles de filtros considerablemente más escarpados que los que presentan los Objetos "lop~", "hip~", o "bp~". La desventaja es que no sólo suprime ciertas frecuencias sino que también aumenta significantemente otras al punto de "explosión" (puede ver en la formula, que el filtro trabaja recursivamente).

Con el procesado bicuadrático también puede observar que los filtros emplean cambios de fase. Es por esta razón, por ejemplo, que un Objeto "bp~" no es simplemente una inversión de un filtro rechazo de banda:



3.4 Muestra

3.4.1 Teoría

Vamos a aclarar un tema potencialmente problemático a fin de evitar cualquier confusión: ya hemos aprendido que una muestra es la unidad más pequeña usada para medir y generar sonido en un

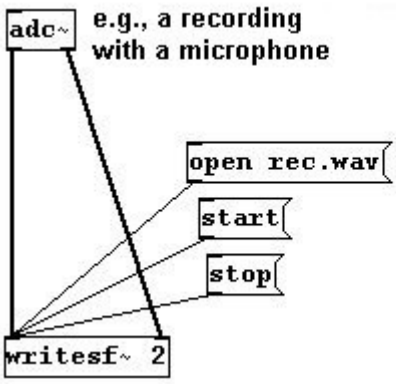
ordenador. En la música electrónica, la palabra 'muestra' también tiene desafortunadamente otro sentido muy diferente; significa una sección más bien pequeña (generalmente dura pocos segundos) de sonido grabado, registrado. Este capítulo trata sobre el procesamiento de pequeños fragmentos de sonido grabado. Primero deberá aprender cómo funciona en Pd una *matriz* o *vector de datos* ("array"), la cual requiere un poco de explicación.

3.4.1.1 Almacenamiento de sonido

3.4.1.1.1 Archivos de sonido

Existen varias locaciones en el ordenador donde los archivos pueden ser almacenados: la memoria principal o los discos duros. El acceso a la memoria principal es muy rápida en comparación a los discos duros; sin embargo, la primera cuenta con menor espacio disponible que los segundos.

El Pd podemos almacenas sonidos en ambas locaciones. El almacenamiento en disco duro implica el almacenamiento de un archivo de sonido fijo. Comunmente son usados los formatos WAV o AIFF. Se utiliza el Objeto "writesf~" para registrar un archivo de sonido a disco. El argumento corresponde al número de canales; este crea un número correspondiente de entradas las cuales se adjuntan con los sonidos que se quiera grabar. Primero debe utilizar el Mensaje "open [nombre]" para fijar el nombre del archivo que se desea crear. Para comenzar la grabación se usa el Mensaje "start" y para finalizarla el Mensaje "stop".

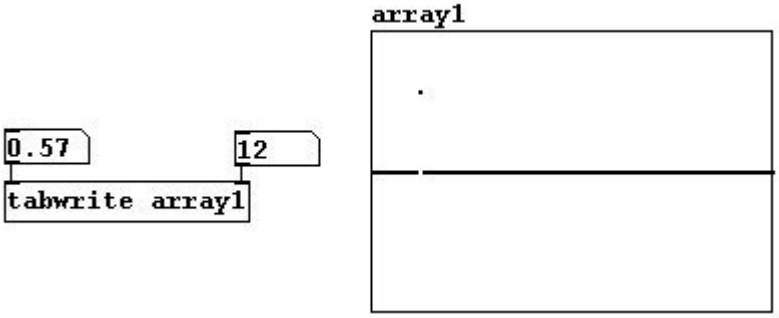


3.4.1.1.2 Reguladores (Buffers)

La otra posible locación es la memoria principal. Cree un lugar para un sonido utilizando una *matriz* o *vector de datos* ("array") (**Put Array**, luego un click en "ok"). También es usado como un espacio de visualización del sonido.

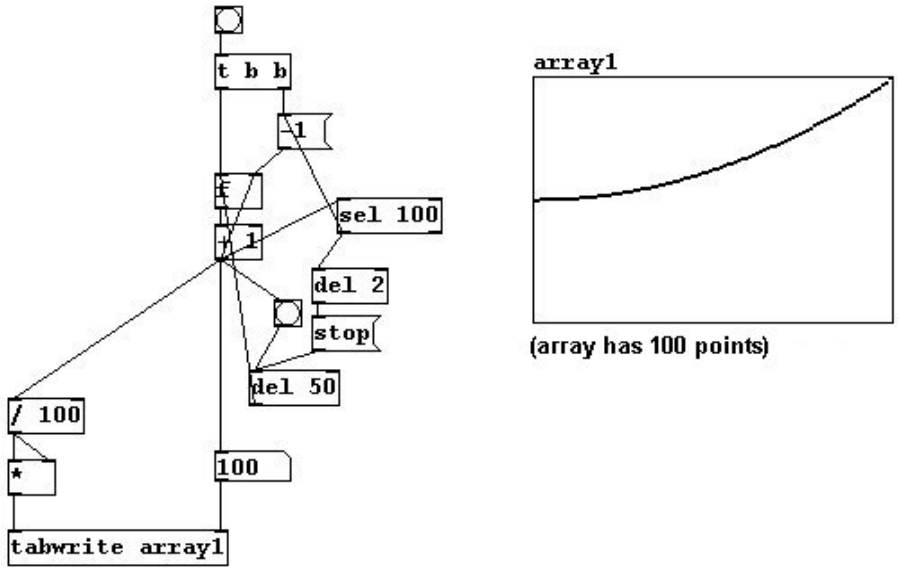
En principio, vamos a pensar a una *matriz* como un depósito de números. Una matriz tiene un número limitado de lugares de almacenamiento. Puede configurar este número clicando con el botón derecho sobre la matriz y luego seleccionando "Properties". Esto abre dos ventanas: una etiquetada como "array" y la otra como "canvas". En la ventana "array", puede configurar dicho número. Este número corresponde al número de lugares de almacenamiento. Un número puede ser almacenado en cada lugar de almacenamiento.

Puede asignar estos elementos usando el Objeto "tabwrite". El argumento corresponde al nombre de la matriz con la que se desea operar; la entrada derecha determina la posición; y la entrada izquierda determina el valor que desea guardar (como siempre: de derecha a izquierda). En la matriz, el eje x muestra la posición y el eje y muestra el valor:



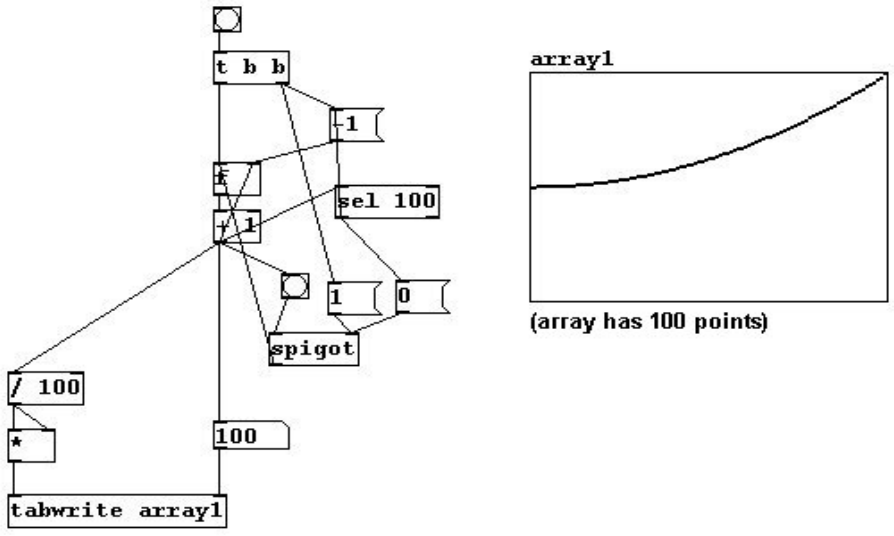
Puede utiliza un "array" para representar funciones:

patches/3-4-1-1-2-function1.pd



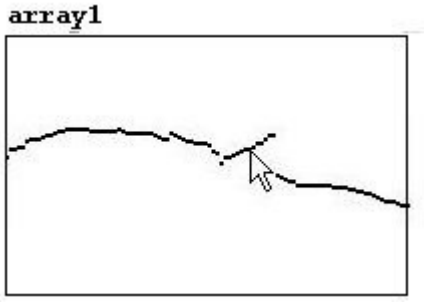
O sin extensión temporal (si el ejemplo mostrado provoca un desbordamiento de pila, será de gran ayuda insertar "del 0" entre el "spigot" y "f"):

patches/3-4-1-1-2-function2.pd

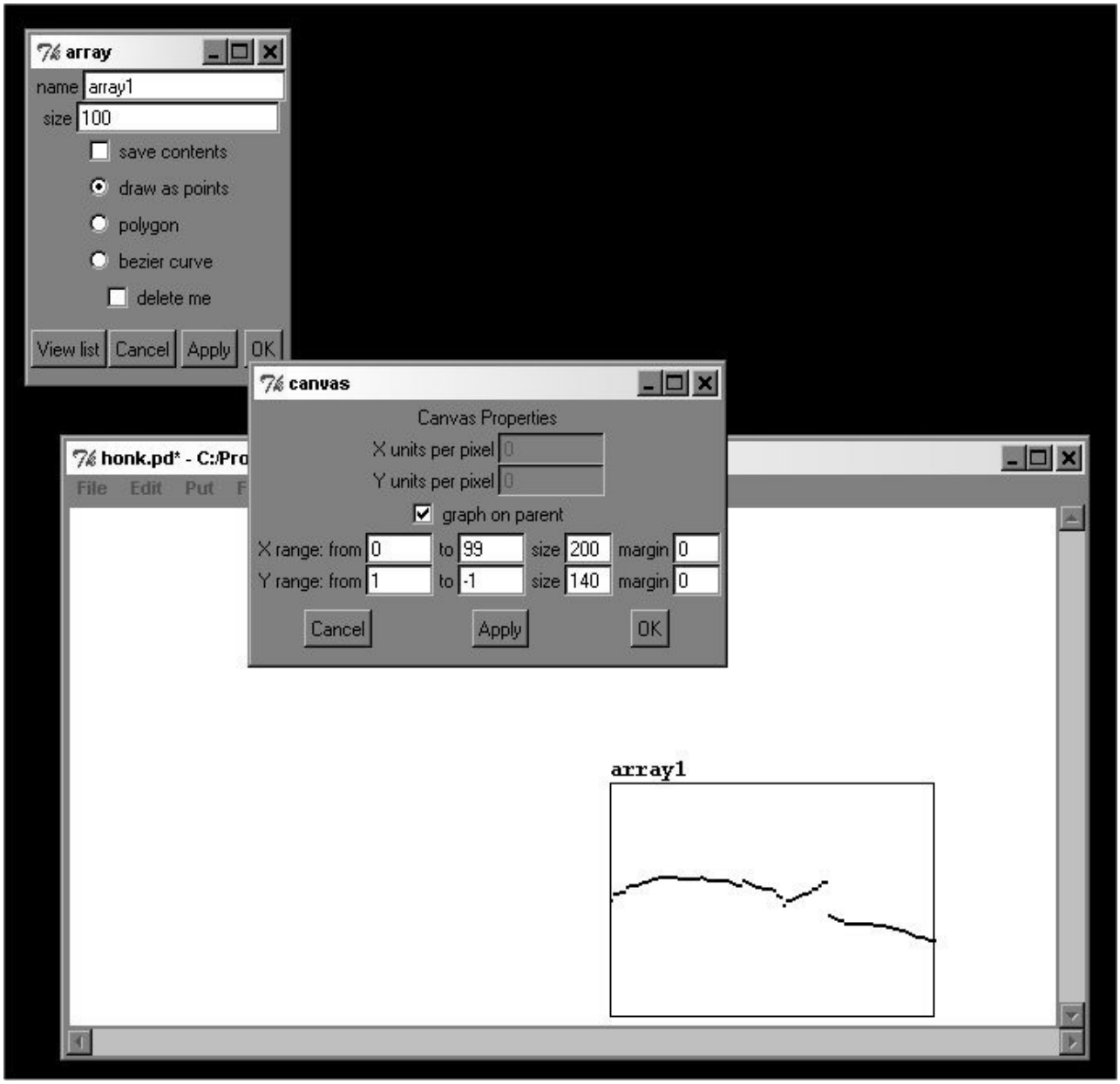


También podría dibujar en la matriz misma con ayuda del ratón, 'a mano' por llamarlo de alguna

manera. Si dirige el ratón a un valor de la matriz, el cursor (con forma de flecha) cambiará su dirección y podrá dibujar moviendo el ratón mientras mantiene presionado el botón derecho.



Bajo 'Properties' (botón derecho sobre la matriz), puede configurar lo siguiente:



En la ventana "array":

- name (nombre): Como todos los nombres en Pd, use caracteres alfanuméricos sin espacios. No utilice números sólamente.

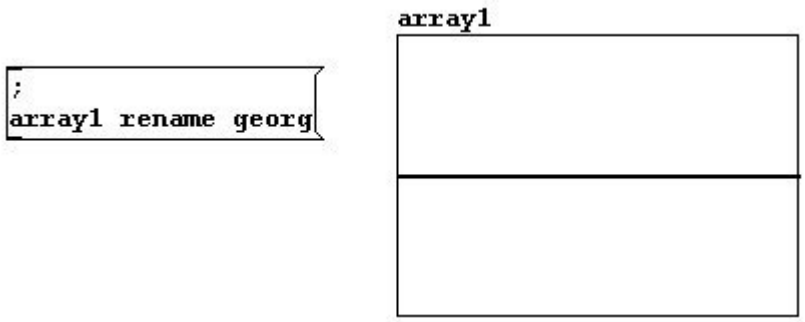
- size (tamaño): Descrito anteriormente.
- "save contents" (guardar contenidos): Cuando se encuentra tildado, todos los valores en la matriz serán guardados. Si utiliza matrices grandes o un gran cantidad de matrices, esto podría causar que su patch se cargue muy lentamente.
- "draw as points" (dibujar puntos) / "polygon" (polígono) / "bezier curve" (curva bezier): Diferentes formas de visualización.
- "delete me" (eliminarme): Cuando se encuentra tildado, ¡borra la matriz! Sin embargo permanece la caja vacía, por lo tanto también debe ser borrada.
- "view list" (ver lista): Muestra los valores en una lista.

En la ventana "canvas":

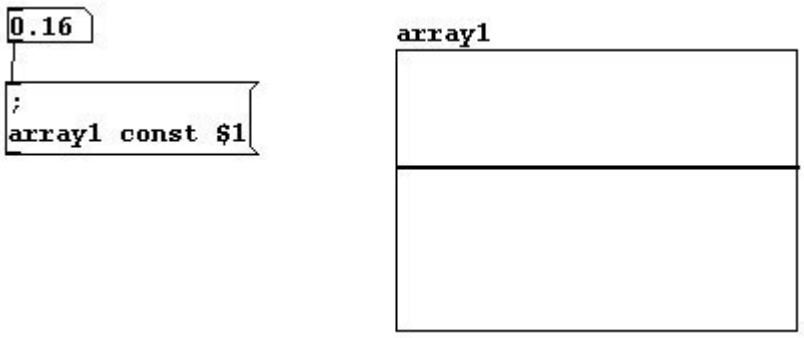
- "graph on parent" (graficar en el principal): Esto será examinado en 5.1.1.2.
- X range (rango X): Aquí puede configurar el rango para el eje x.
- Y range (rango Y): Aquí puede configurar el rango para el eje y. Los valores que excedan este rango también serán guardados, pero también ampliará toda la ventana para que estos valores puedan ser vistos.
- "size" (tamaño): Tamaño de visualización en el patch.

Una matriz también puede recibir mensajes y "envíos".

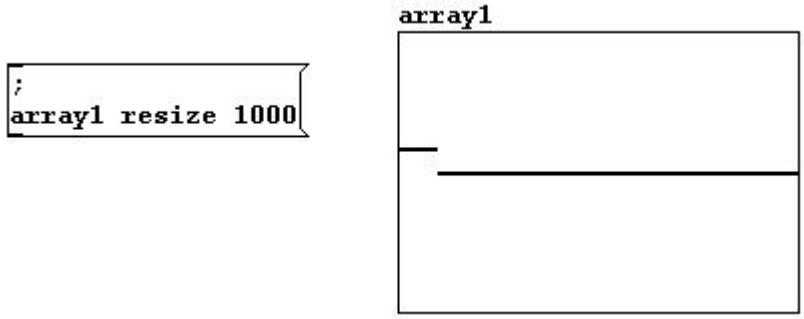
Renombrar:



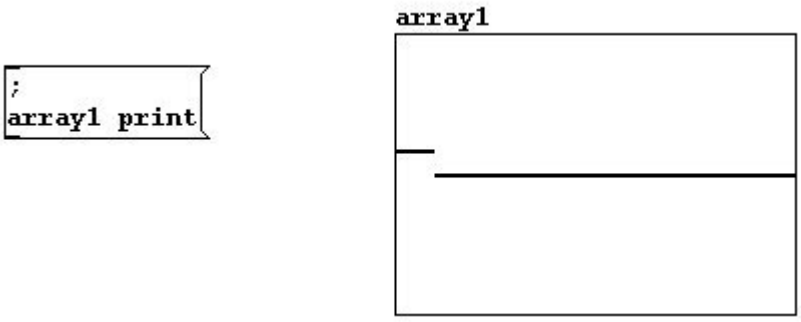
Igualar todos los valores:



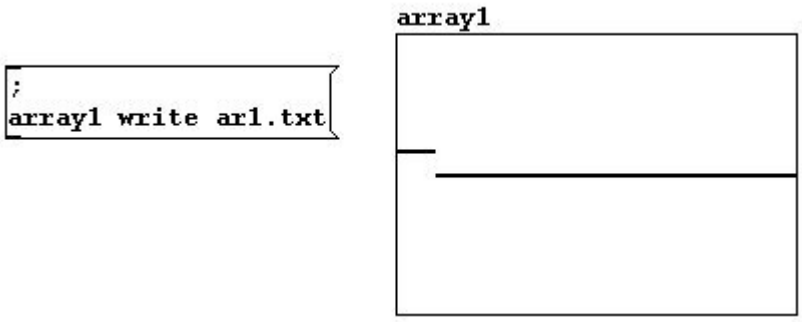
Cambiar el tamaño:



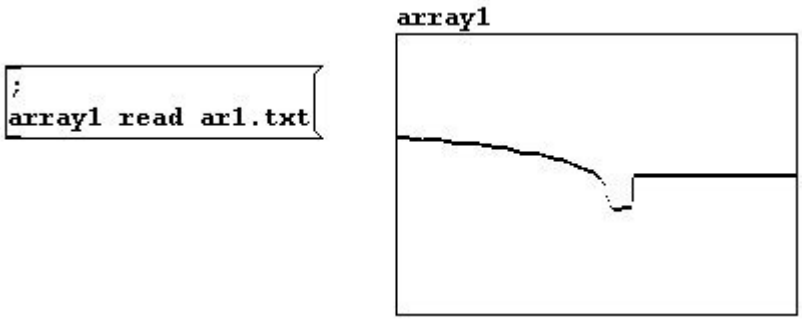
Imprimir el tamaño:



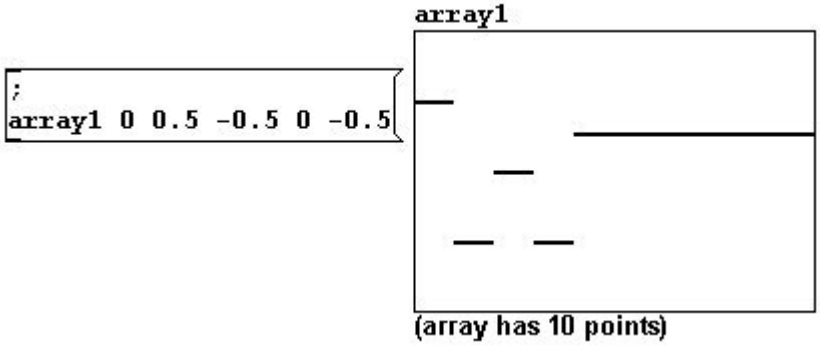
Escribir los contenidos a un archivo de texto:



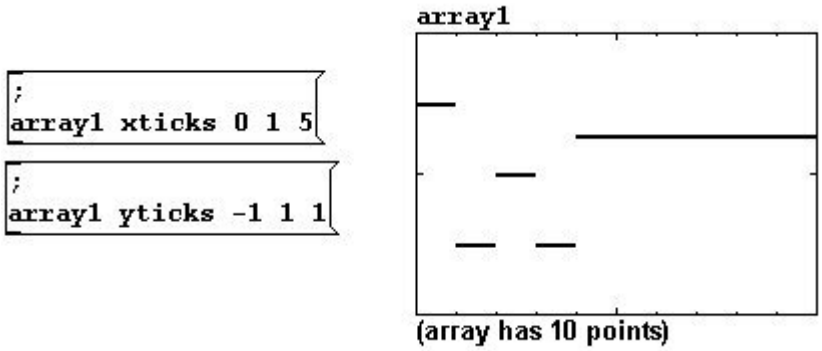
Leer un archivo de texto:



También puede escribir valores de la siguiente manera. El primer número determina el lugar de almacenamiento inicial, el segundo al valor del mismo; todos los demás valores corresponden a las posiciones que le siguen:

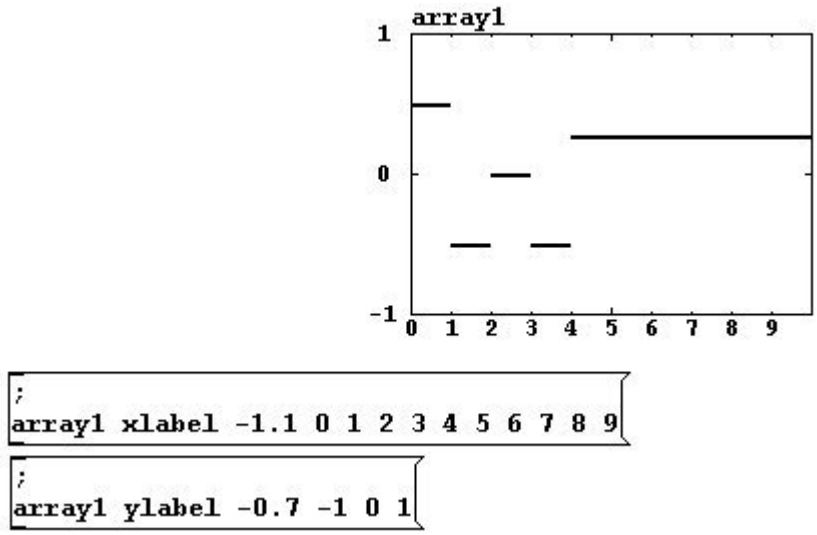


También puede crear signos en los ejes y etiquetarlos:



El primer argumento corresponde a la posición inicial; el segundo a la distancia entre lineas; el tercero es la distancia entre las lineas largas.

También los puede enumerar:



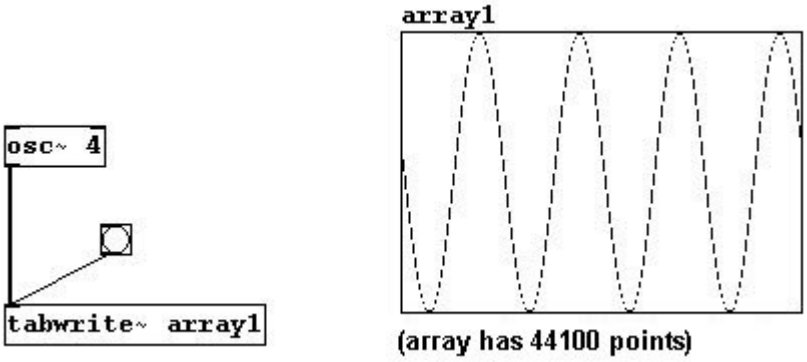
Primero, la posición de la etiqueta de números; segundo, los números que quiere mostrar de dicha etiqueta.

Note bien: Las lineas y las etiquetas no se guardan en el patch, entonces debe reingresarlas cada vez que abra el patch.

También puede, desde luego, almacenar sonido en una matriz. Para un ordenador, el sonido es -como hemos venido mencionando- sólo una cadena de números, 44100 para ser precisos. Puede guardar un segundo de sonido en una matriz; sólo necesitará 44100 lugares de almacenamiento.

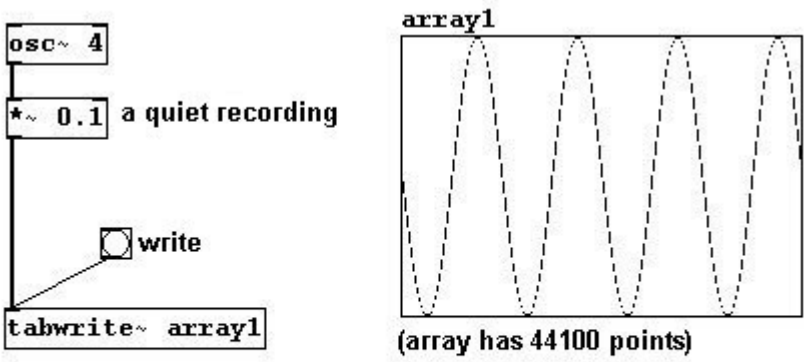
Aquí es donde entra en juego el Objeto "tabwrite~" . Recibe, o bien una entrada de sonido, o bien sólo un bang. Diferente al "tabwrite" (sin virgulilla) donde se ingresa el lugar de almacenamiento en la entrada derecha de forma 'manual' y su correspondiente valor en la izquierda, cuando "tabwrite~" recibe un "bang", comienza automáticamente con el primer lugar de almacenamiento y luego procede con los demás a la frecuencia de muestreo (44100 muestras/seg). Al mismo tiempo, a cada lugar de almacenamiento se le asigna un valor recibido por la entrada izquierda del sonido actual, resultando en total unos 44100 números guardados. Cualquier otra muestra que le siga a dicho número no será registrado. Si quiere deterner la asignación prematuramente, puede enviar un mensaje "stop".

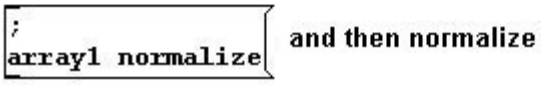
patches/3-4-1-1-2-normalize.pd



Si "tabwrite~" recibe un mensaje de punto flotante, este número será interpretado como un desplazamiento de muestra (sample offset). En otras palabras, la muestra que corresponde a este número flotante será el punto de inicio para la matriz.

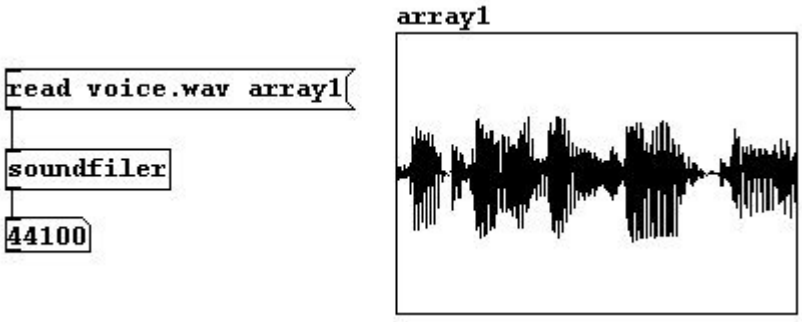
Una función util es que que podemos aumentar el volumen general. Si, por ejemplo, la grabación original es tiene poco volumen (es decir, la membrana del micrófono no vibró demasiado, lo cual resulta en muy pequeños valores), puede amplificarlo. Esto se llama "normalización" . Para realizarlo, puede utilizar el siguiente mensaje:





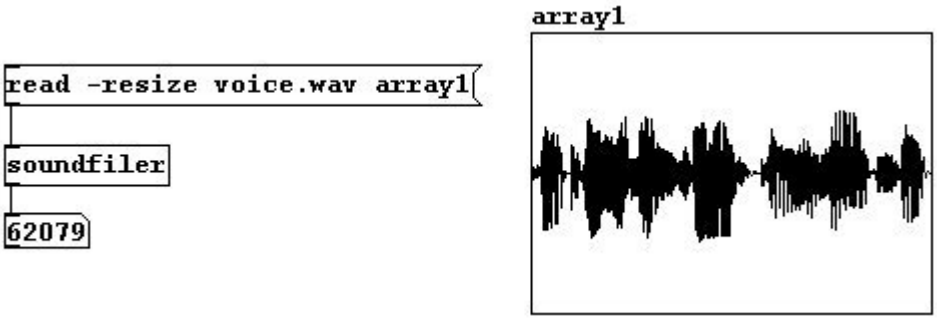
Tambien puede vincular un archivo de sonido sito en un disco duro a la memoria principal en una matriz. Esto se logra mediante el Objeto "soundfiler". Este Objeto permite cargar un archivo de sonido previamente guardado en un disco duro, dentro de una matriz, o salvar los contenidos de una matriz en un disco duro como un archivo de sonido. El comando "read" es usado para cargar un archivo de sonido. Los argumentos para el comando son el *nombre del archivo* (con el sendero si es necesario) y luego el *nombre de la matriz* en la cual se desea escribir.

patches/3-4-1-1-2-load-soundfile.pd



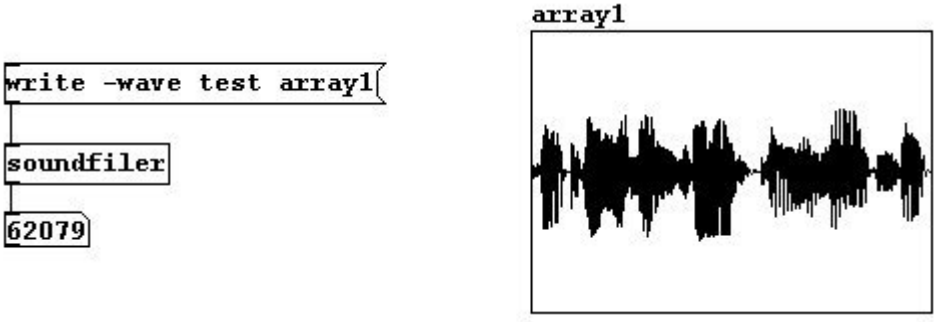
Una vez que haya cargado satisfactoriamente el archivo, se enviará el tamaño del archivo en muestras por la salido del Objeto "soundfiler".

También puede incluir otros comandos (llamados "flags") dentro del mensaje:



El comando "-resize" cambia el tamaño de la matriz para adecuarla a la del archivo de muestra (este se encuentra limitado a 4000000 muestras – casi 90 seconds, aunque puede ser cambiado mediante "maxsize").

A la inversa, el comando "write" guarda los contenidos de una matriz como un archivo de sonido a un disco duro. En este caso, el *formato* (WAV or AIFF) debe ser dado como flag, luego el *nombre del archivo* (con el sendero si es necesario), y luego el *nombre de la matriz*.

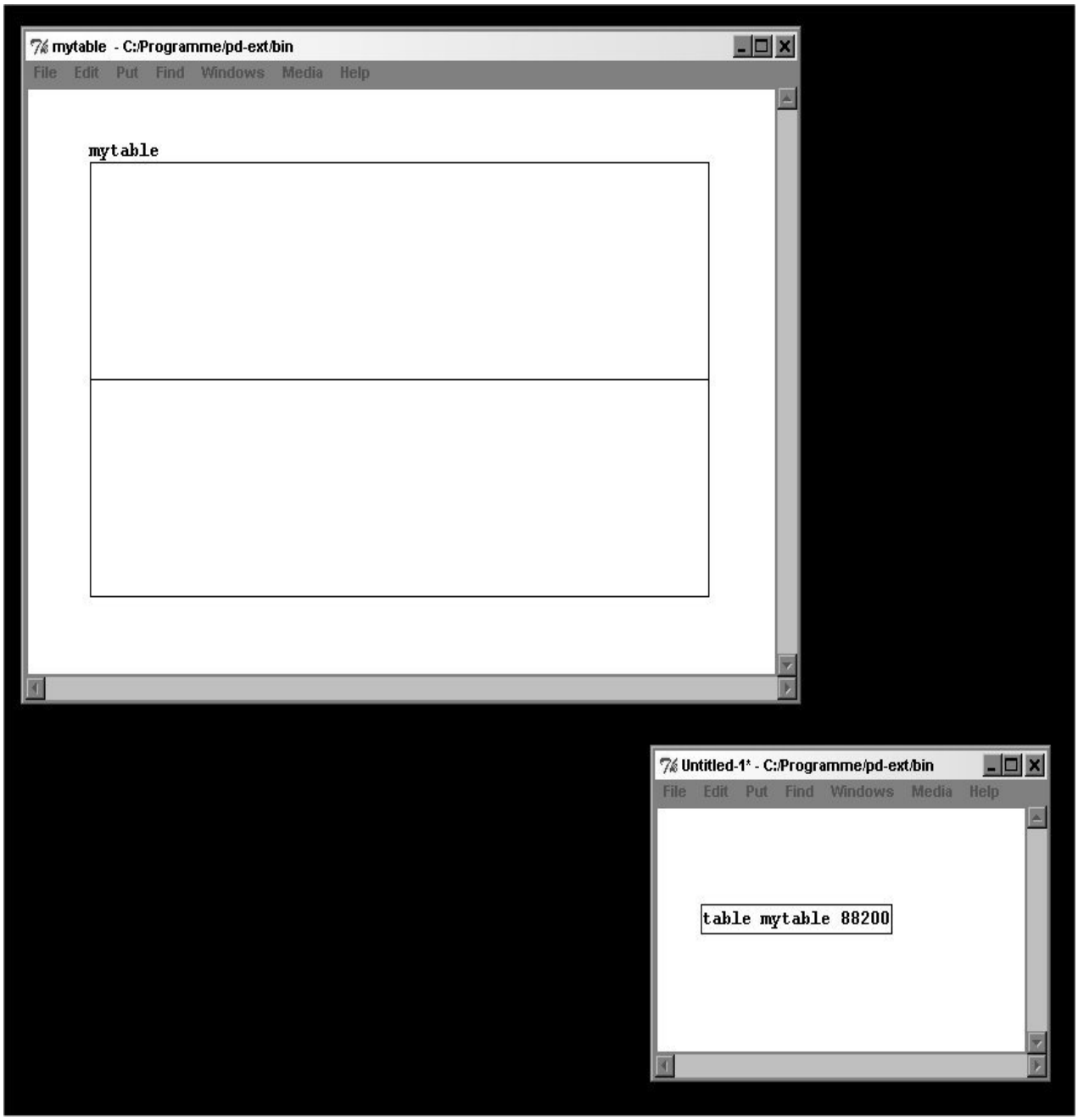


Otros "flags" importantes en este contexto son:

- normalize (normalizar): Optimiza los niveles de amplitud, como ha sido explicado anteriormente.
- rate (frecuencia de muestreo): Utilizado para configurar la frecuencia de muestreo para el archivo de sonido.

Como alternativa a la matriz, también puede utilizar una *tabla* ("table"). Cree un Objeto "table"; ingrese el *nombre* como primer argumento y el *tamaño en muestras* para el segundo. Esto creará una matriz en un subpatch (clic en el Objeto en el modo de ejecución) la cual será tratada como una matriz normal. Esta propuesta tiene la siguiente ventaja: los gráficos para una matriz normal pueden ser muy complejos. Esto puede ser verificado cuando transladamos una matriz grande sobre el

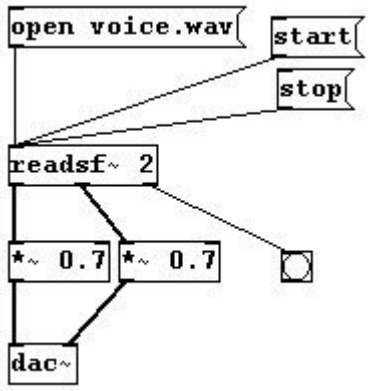
canvas: se mueve muy lentamente. Pero si la representación gráfica de una matriz se encuentra dentro de un subpatch, el mismo objeto puede ser transladado mucho más fácilmente.



3.4.1.2 Reprocucción de un sonido guardado

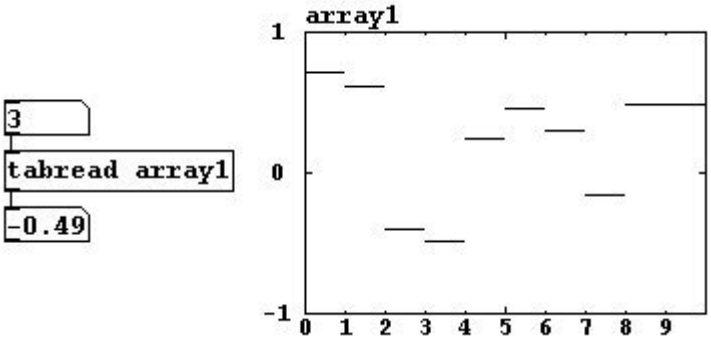
Los archivos de sonido que se encuentran alojados en un dispositivo de almacenamiento externo, como un disco duro, puede ser leido -es decir, reproducido- en Pd mediante el Objeto "readsf~". Como ocurre con "writesf~", utilice los mensajes "start" y "stop" (también puede utilizar"1" y "0"). Ingrese como argumento en número de canales. La salida extrema derecha envía un bang cuando el final del archivo es alcanzado.

patches/3-4-1-2-play-file.pd



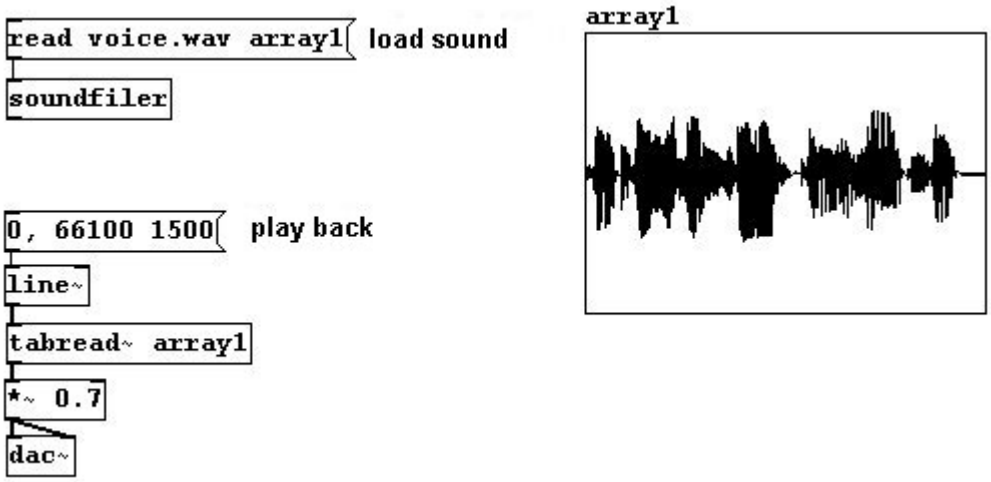
Veamos el nivel de control para una matriz: digamos que tiene una matriz con 10 lugares de almacenamiento. Puede utilizar "tabread" para leer cada uno de estos lugares:

patches/3-4-1-2-read-array1.pd



El principio es básicamente el mismo para las señales, excepto que tiene que recibir los valores guardados a una frecuencia de 44100 número por segundo. Es por esto que contamos con "tabread~". Si desea, digamos, reproducir un sonido almacenado en una matriz que dura 1.5 segundos (es decir, 66150 muestras), debe leer los valores de la matriz desde 0 a 66149 a una frecuencia de 44100 valores por segundo. Puede llevar a cabo esto con "line~":

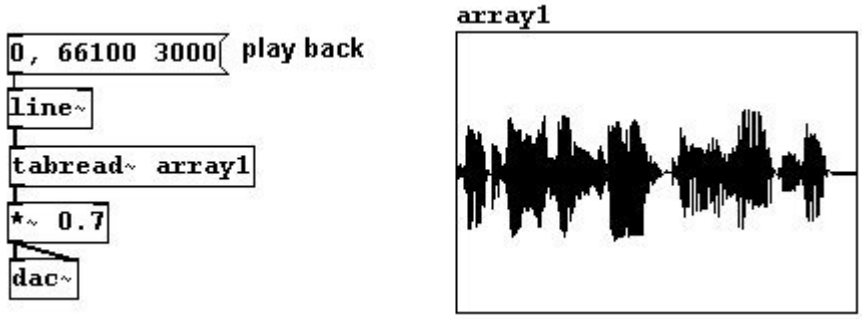
patches/3-4-1-2-read-array2.pd



"tabread~" recibe el *nombre de la matriz* como argumento. También podría optar por la matriz que quiera leer mediante el mensaje "set [nombreDeLaMatriz]".

Ahora puede comenzar a jugar con estos valores. Por ejemplo, puede reproducirlo doblando el

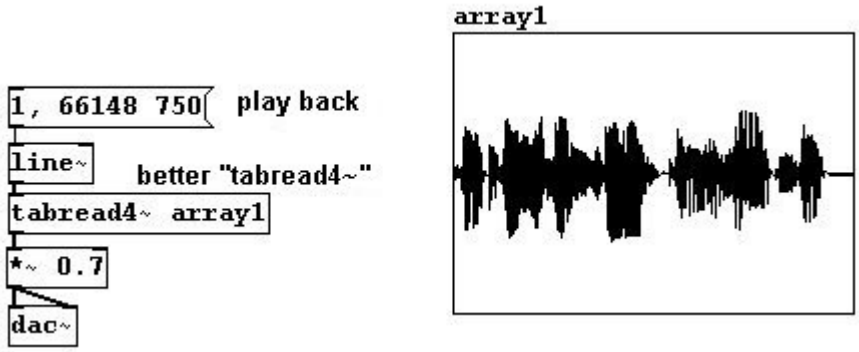
tiempo:



Esto disminuye la velocidad de reproducción a la mitad. Causa que todo suene una octava más grave porque la extensión del tiempo hace doblar todas las ondas sonoras, lo cual significa que las frecuencias serán reducidas a la mitad sonando así una octava más baja.

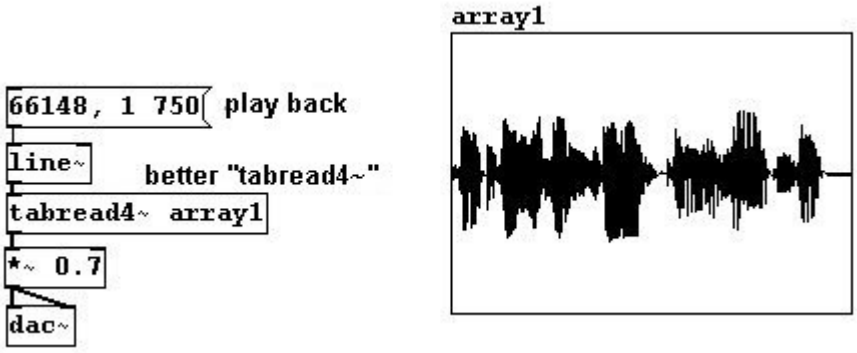
Esto conduce al problema que cada muestra individual que use necesita ser reproducida dos veces o de lo contrario existiran vacios. Para evitar este problema, existe una forma modificadad de este Objeto "tabread~" llamado "tabread4~", el cual interpola valores intermedios que genera utilizando información de los valores que en última instancia le precede y sigue (información más específica de esta función se encuentra disponible en 3.4.4.). En la mayoría de los casos, "tabread4~" es más apropiado para la lectura de matrices. Requiere un espectro de lectura de salida desde 1 a n - 2, donde 'n' corresponde al tamaño de la matriz que desea leer.

Por supuesto, puede reproducir algún sonido más rápidamente, lo cual aumentaría la frecuencia:

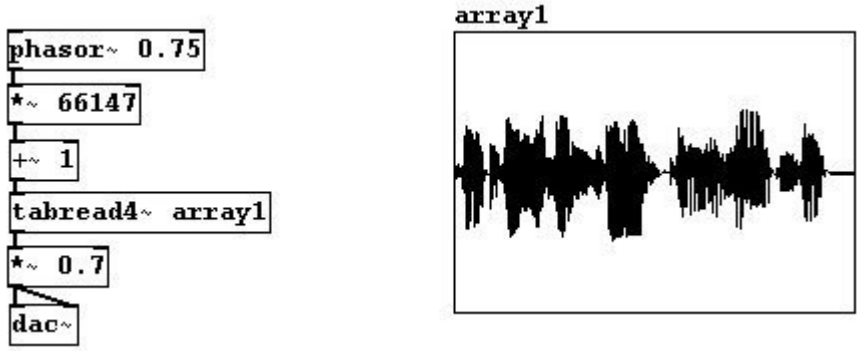


Más adelante, cuando se explique la síntesis granular, aprenderá a alterar el tempo y la altura independientemente.

Reproducir un archivo de muestra hacia atrás es naturalmente también una posibilidad:



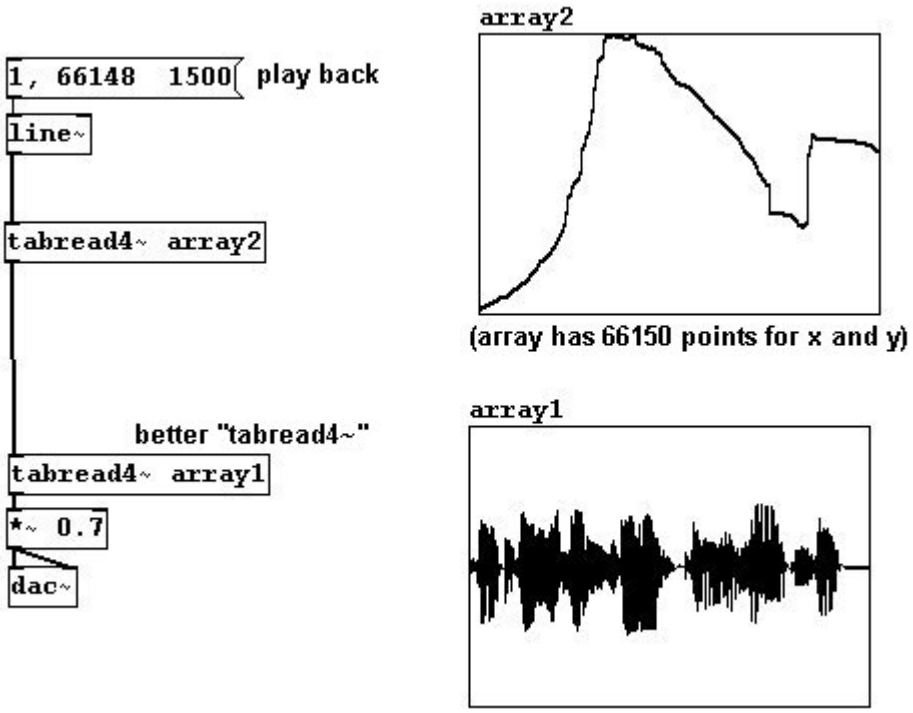
También podría usar el Objeto "phasor~":



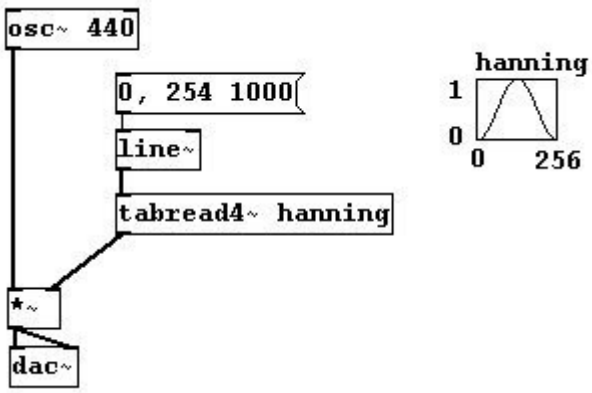
"phasor~" genera una serie de números entre 0 y 1 como señal. Si multiplica estos valores por 66148, obtiene una serie de números entre 0 y 66148. Ingrese 0.75 para la frecuencia de manera que la serie ocurra en exactamente 1.5 segundos.

Otra posibilidad sería crear su propia matriz para la estructura de la lectura de salida y usarlo para reproducir la primera matriz:

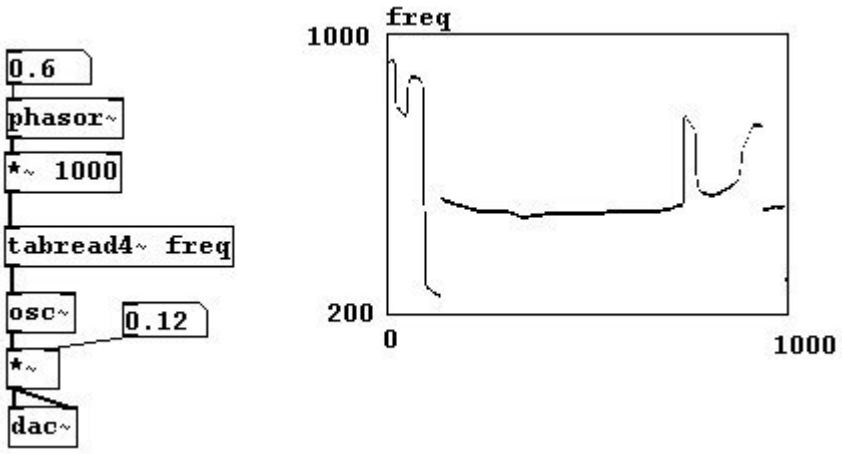
patches/3-4-1-2-read-array3.pd



O puede usar una matriz para controlar la amplitud:



O para controlar la frecuencia:



Nuevamente puede observar: sólo utilizamos números para controlar varios parámetros.

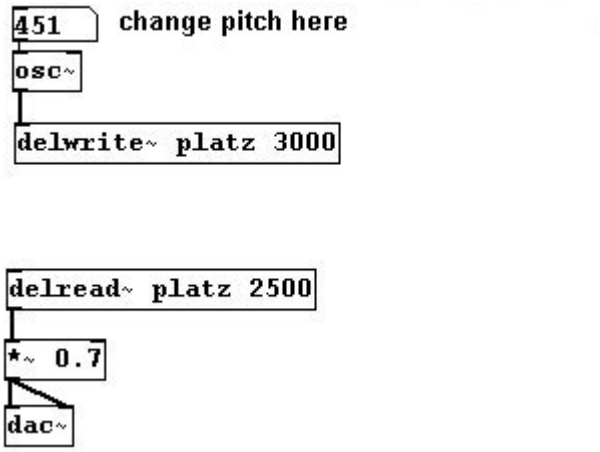
Puede utilizar tantos Objetos "tabread~" como quiera para leer la misma matriz. Sin embargo, nunca debe usar dos matrices con el mismo nombre, ya que esto seguramente lo conducirá a errores.

3.4.1.3 Retardo de audio

En el Capítulo 2.2.3.1.2 mencionamos cómo puden ser retardados los números o series de números. Tambien puede hacerlo con las señales. Esto se realiza con la creación de un buffer en el cual se escriben señales y del cual se leen señales luego de un cierto retardo. Para crear este buffer, utilice el Objeto "delwrite~". El primer argumento corresponde a un *nombre de fantasía* del buffer; el segundo al *tamaño* en milisegundos. A la entrada izquierda la alimentamos con la señal que deseamos escribir en el buffer. Una vez que el buffer se haya llenado, este se reescribe nuevamente desde el principio. Si el buffer almacena 1000 milisegundos, los últimos 1000 milisegundos de la señal entrante será guardados en el buffer.

Utilice "delread~" para leer los datos del buffer. El primer argumento corresponde al *nombre de fantasía* del buffer que deseamos leer; el segundo al retardo (en milisegundos; el cual puede ser cambiado usando un controlador de ingreso de datos en la entrada):

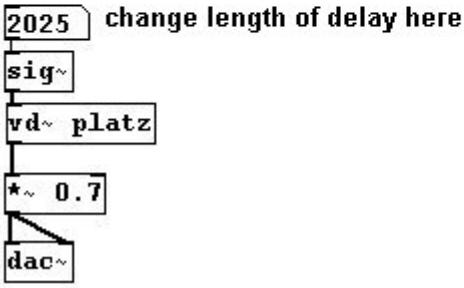
patches/3-4-1-3-delay.pd



Lógicamente la cantidad de retardo en el "delread~" debe ser menor o igual al tamaño del buffer. Si tiene un retardo de 2000 milisegundos pero el buffer contiene sólo 1000 milisegundos, naturalmente no funcionará. También resulta imposible utilizar un número negativo para el intervalo de retardo ya

que Pd no puede ver el futuro. Puede usar tantos Objetos "delread~" como desee para leer simultáneamente un retardo de buffer. Por último, no es posible visualizar los patrones de onda que se encuentran en un buffer.

Si bien es posible cambiar el intervalo de retardo en "delread~", para ello tiene que utilizar un controlador de ingreso de datos y existe una cierta probabilidad de error cuando se excede un límite arbitrario de velocidad (nuevamente estamos ante un conflicto entre datos de control y señales). Por esta razón, contamos con un Objeto especial para lecturas variables de retardo de buffers llamado "vd~" (abreviación para "variable delay", retardo variable). Se establece un intervalo de retardo (en milisegundos) a la entrada del mismo como señal , y luego lo puede cambiar de la manera que quiera (aunque, nuevamente, no puede usar números negativos ni exceder el tamaño del buffer):

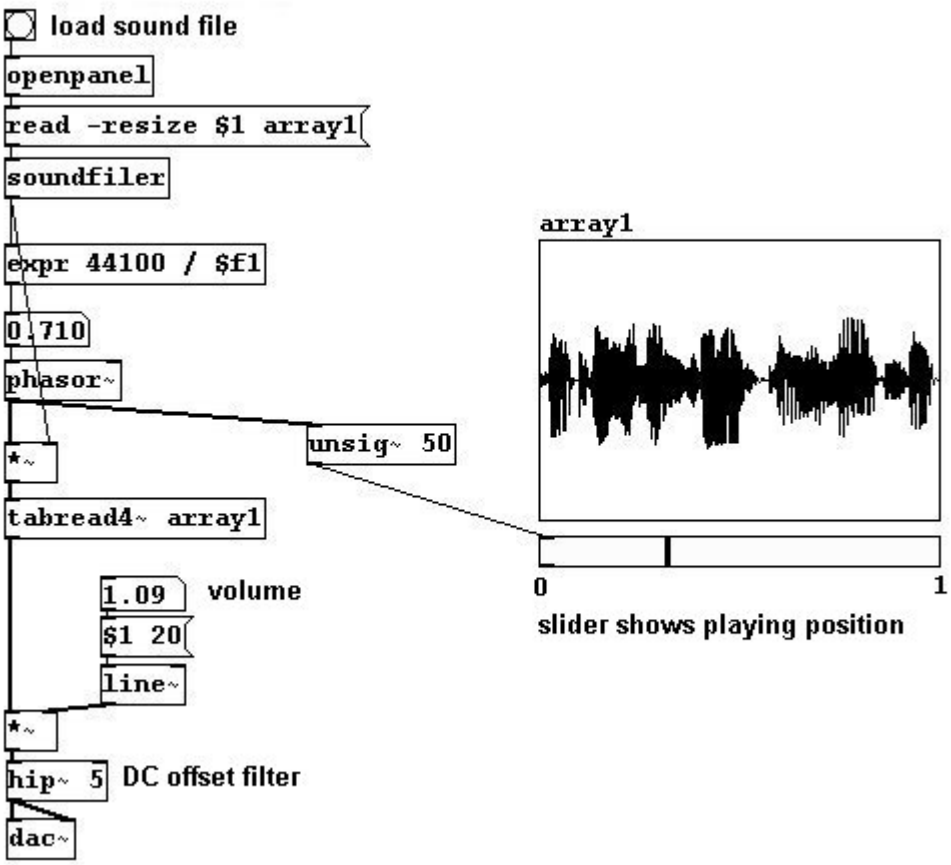


"vd~", tal como "readsf4~", crea interpolaciones.

3.4.2 Aplicaciones

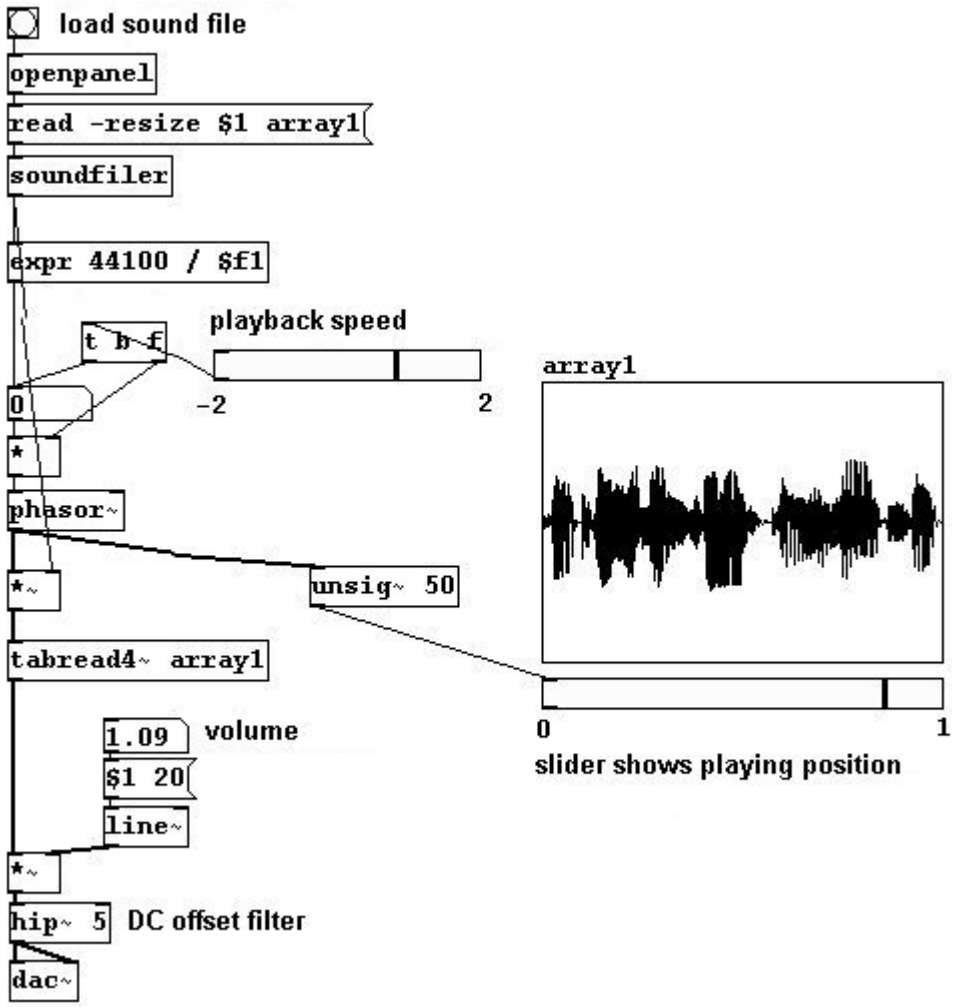
3.4.2.1 Un sampler sencillo

patches/3-4-2-1-simple-sampler.pd



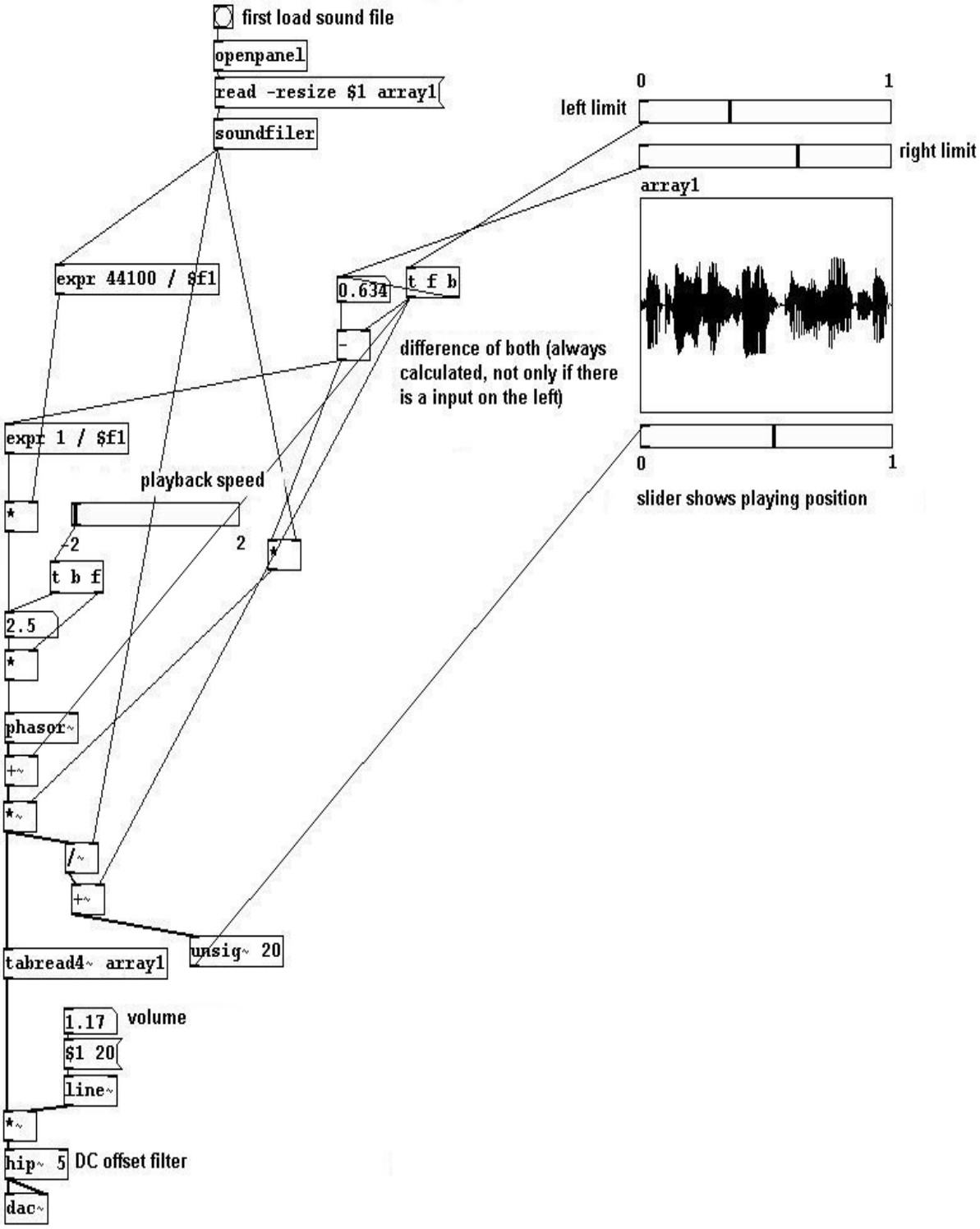
3.4.2.2 Con velocidad variable

patches/3-4-2-2-sampler2.pd



3.4.2.3 Cualquier posición

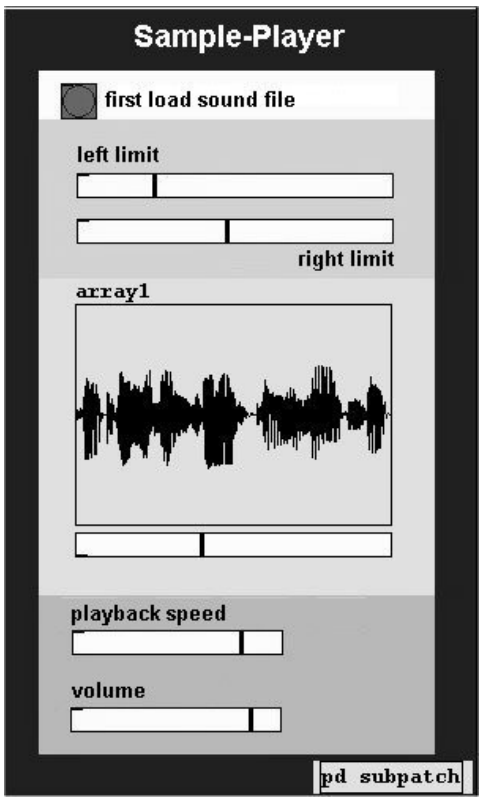
Aquí tiene una manera de escoger un fragmento de cualquier muestra que desee:
patches/3-4-2-3-sampler3.pd



3.4.2.4 Reproductor de muestras (Sample-Player)

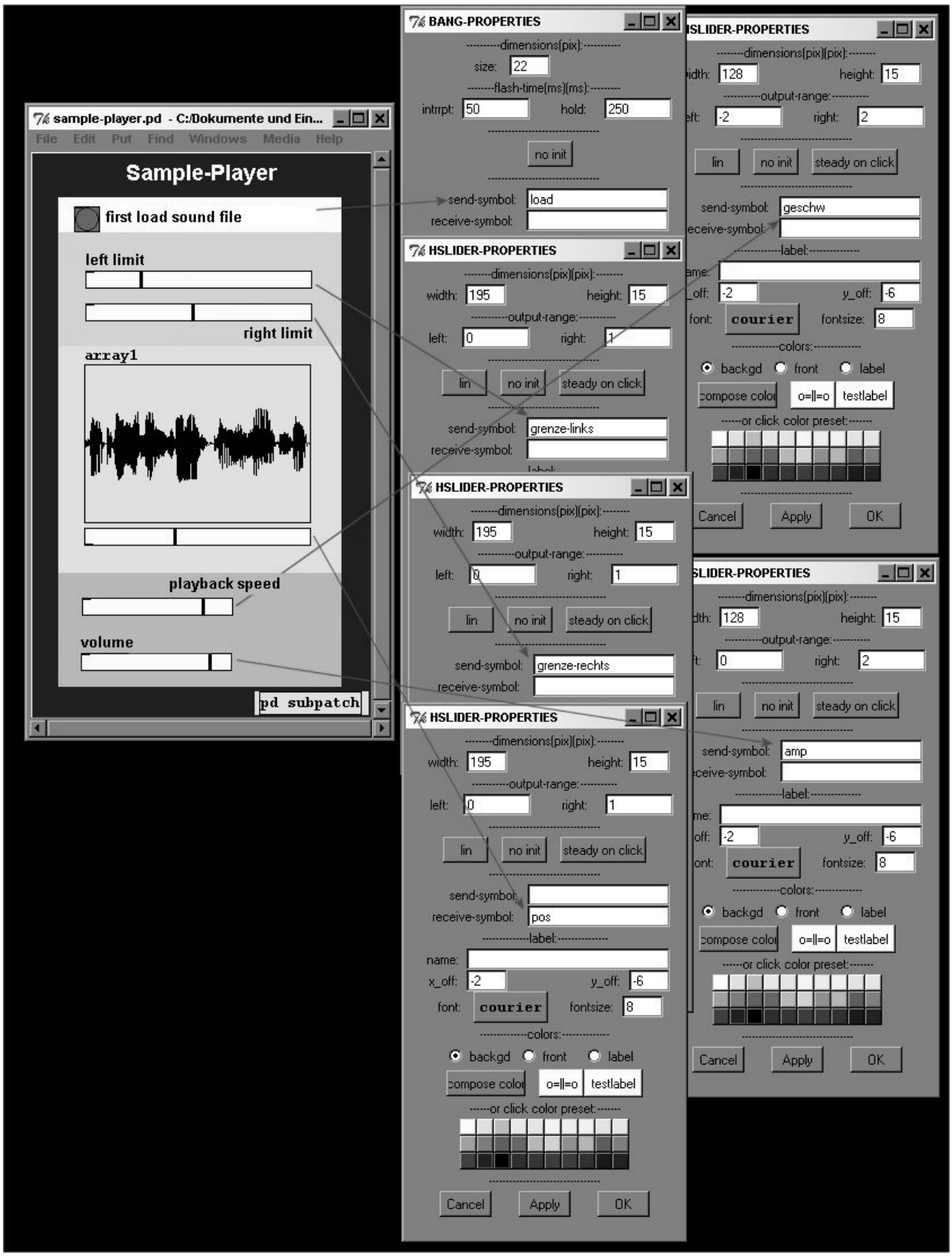
Si Ud. Cambia la representación gráfica, como hemos descrito previamente en 2.2.4, entonces su patch podría verse así:

patches/3-4-2-4-sampler-big.pd

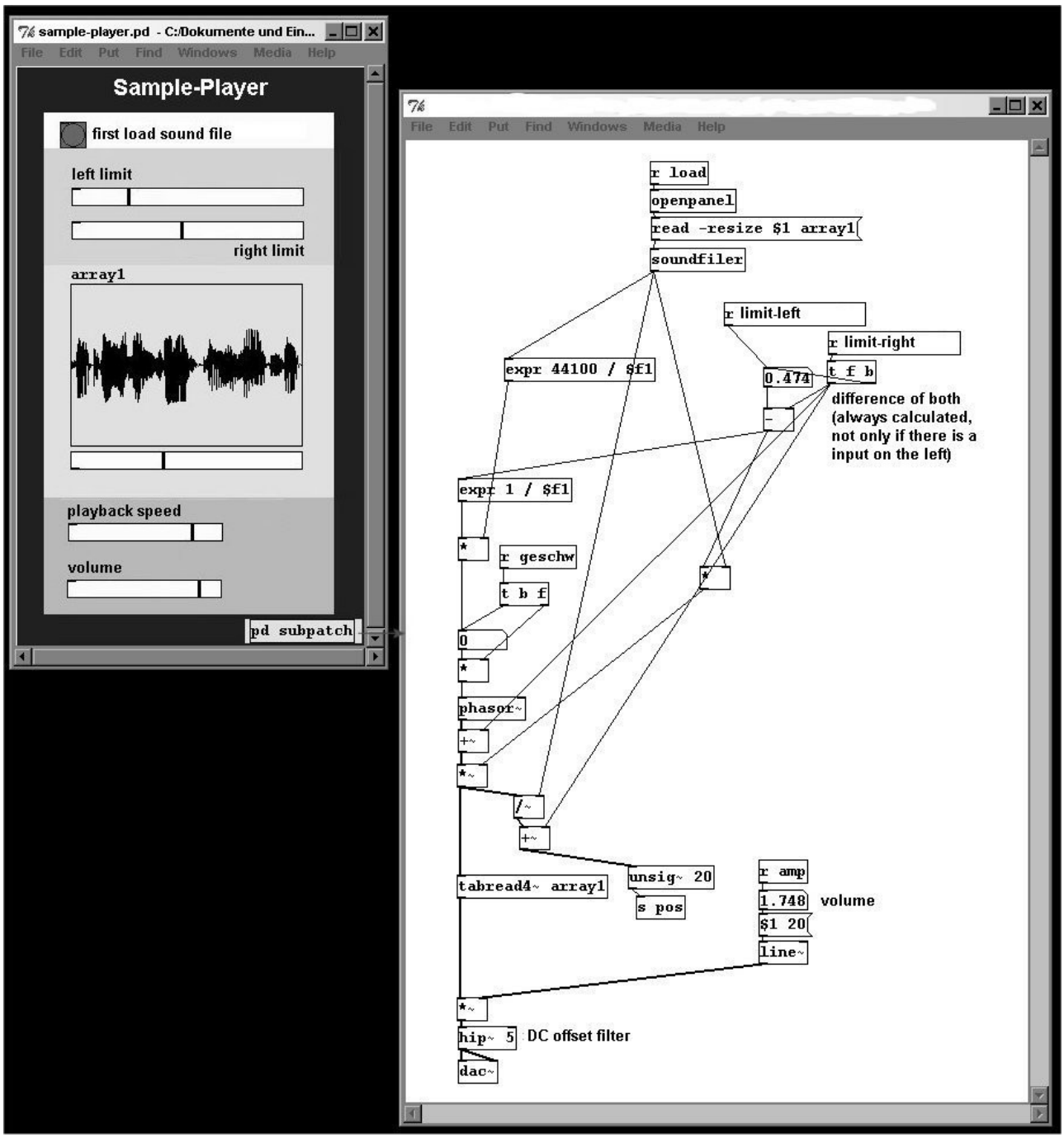


Cuatro Objetos "canvas" componen el atractivo fondo de los deslizadores y de la matriz. Note Bien: los gráficos que se crean en último término siempre aparecen por sobre los otros. Digamos que Ud. Ya cuenta con "array1" y luego crea un Objeto canvas coloreado; debe volver a crear "array1" (sólo copie, pegue y elimine el original) para que el nuevo "array1" aparezca por sobre el Objeto canvas.

Explicaremos cómo fue hecho exactamente:

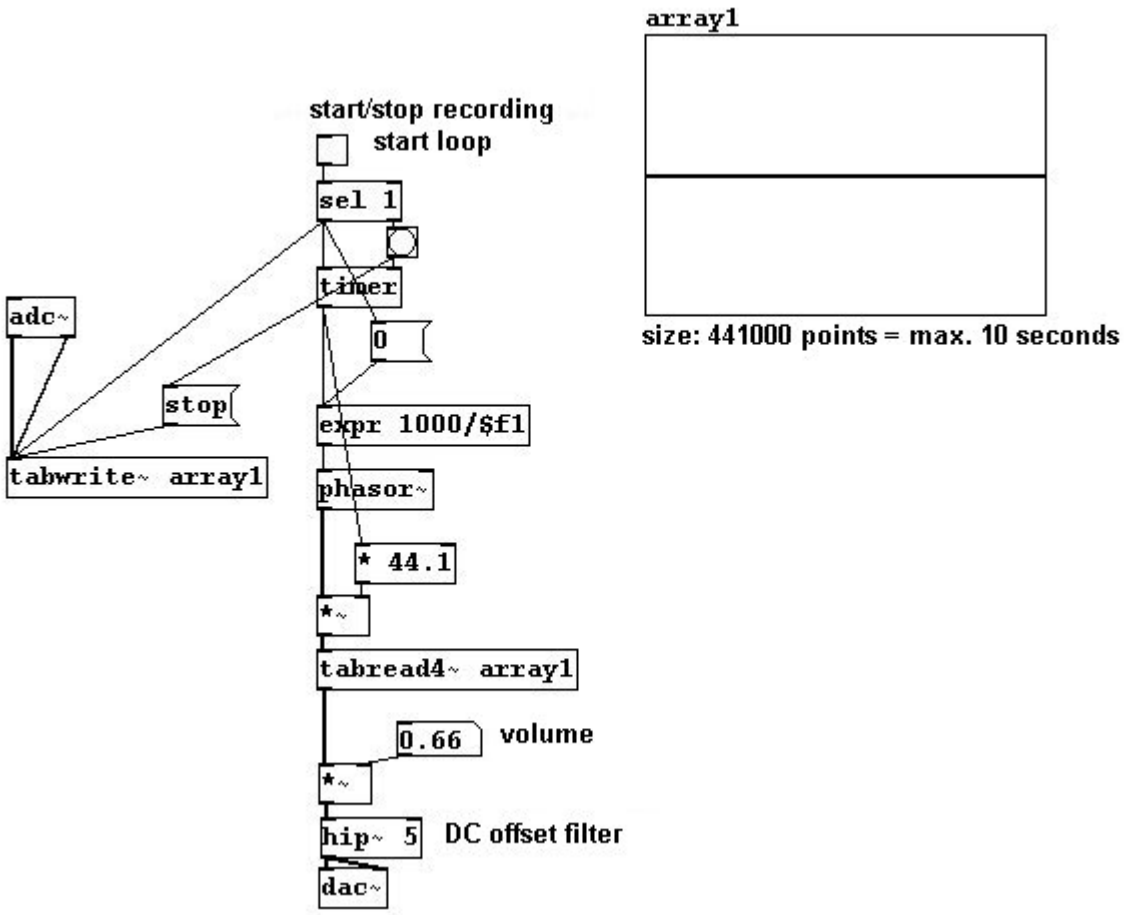


Y a continuación el subpatch:

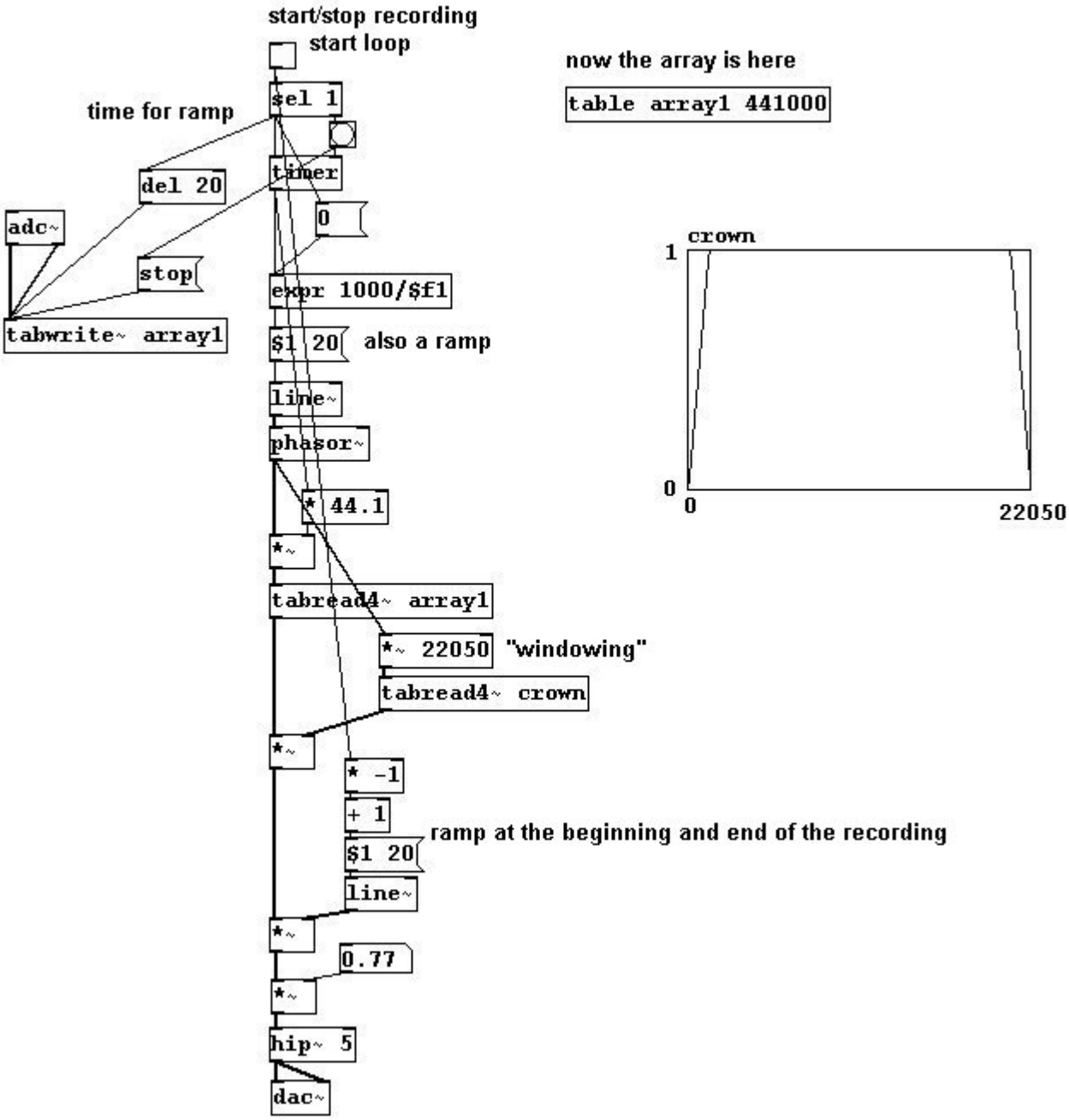


3.4.2.5 Generador de bucles (Loop generator)

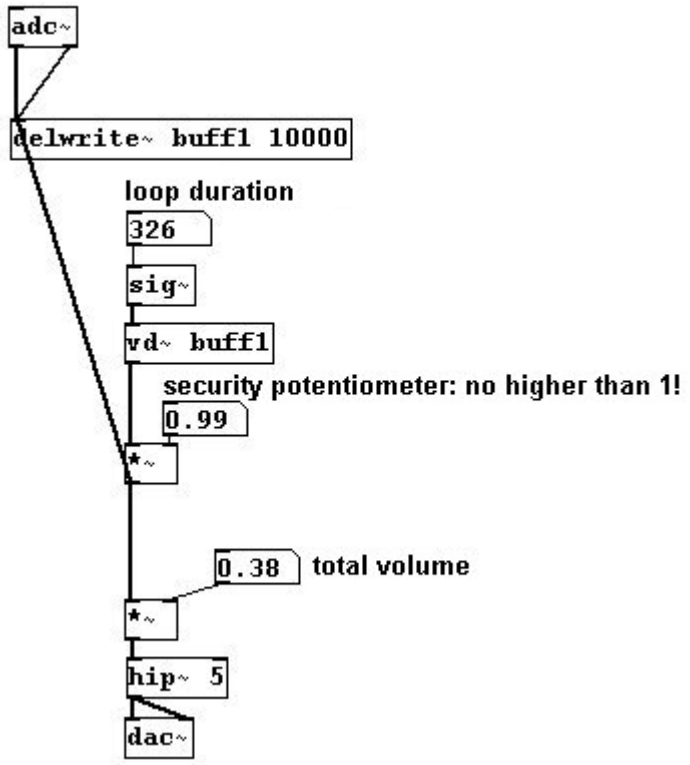
patches/3-4-2-5-loop-generator1.pd



Lamentablemente, en este generador de bucles ocurren clicks. Primero, es altamente recomendable colocar la matriz en un subpatch porque el gráfico requiere muchos recursos del procesador. Segundo, al final del bucle debemos ir brevemente a 0 para que no se genere un súbito salto de valor (cuyo efecto sería un click). Para esto debemos programar un "ventaneo" (windowing). En sincronización con la salida de lectura de la matriz, es determinado por la amplitud de otra matriz (en este caso por "crown") la cual controla la encolvente dinámica. Esta envolvente tiene un valor de 0 al inicio y al final para asegurar que no haya un cambio repentino de valor cuando el bucle se repita. En lugar de una ventana "crown", también puede utilizar una ventana "Hanning", la cual usa una parte de la función seno (esto será visto más tarde). Por supuesto que la matriz "crown" debe encontrarse también en otra ventana, pero lo hemos dejado así para resultar más claros.



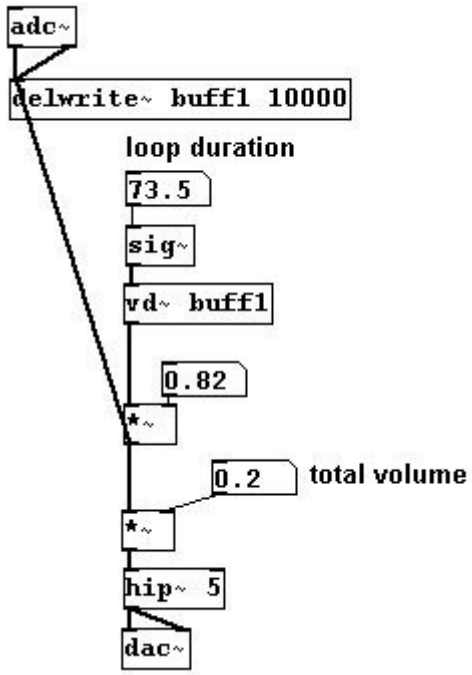
Una versión más simple de bucle puede ser creada utilizando retroalimentación (feedback): patches/3-4-2-5-loop-generator2.pd



La desventaja que tiene este enfoque es que conlleva un máximo de duración de bucle; en este caso nos atenemos a 10000 milisegundos.

3.4.2.6 Reverberador

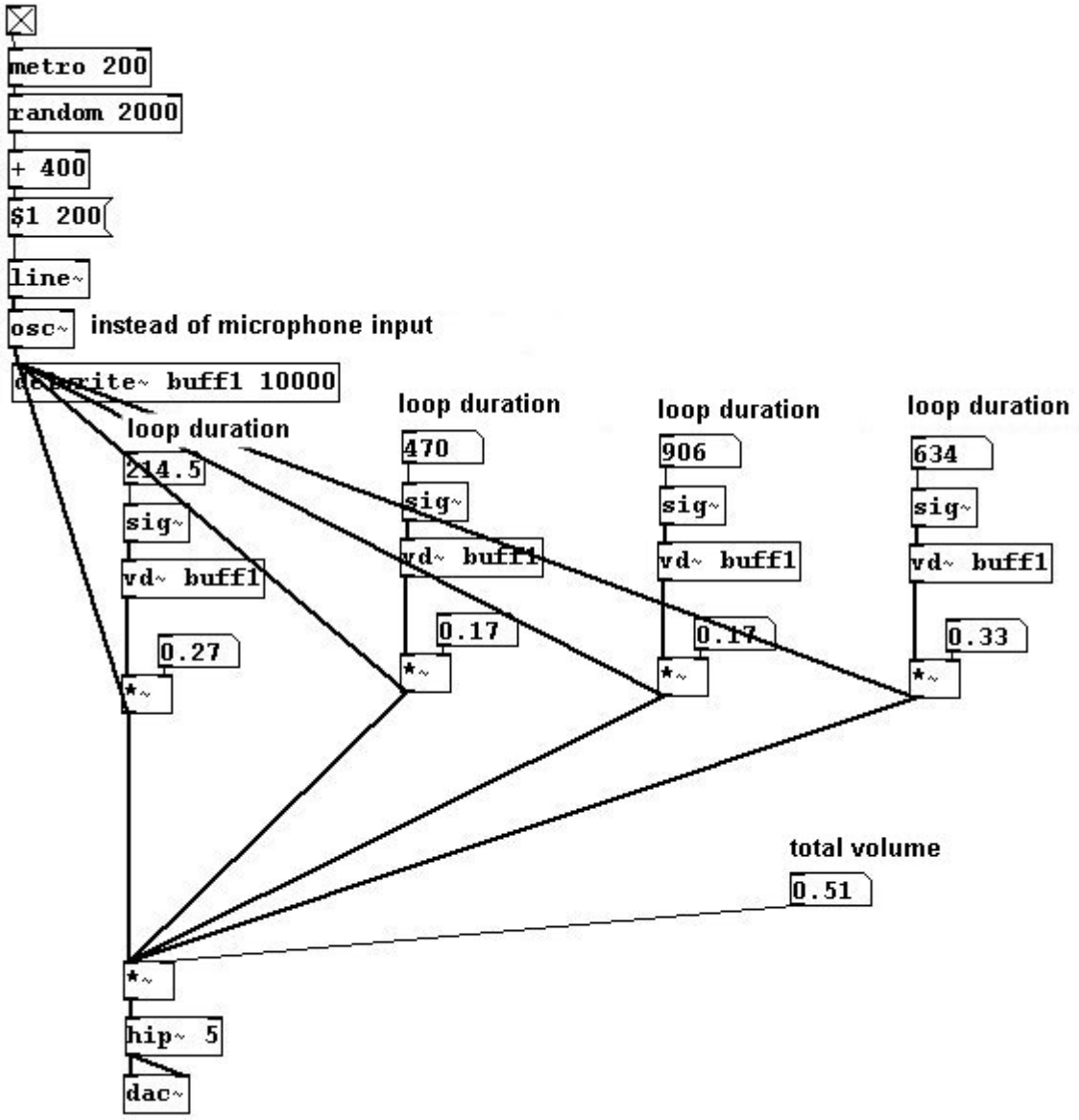
Puede simular un efecto de "reverberación" si la retroalimentación de la señal disminuye continuamente:



3.4.2.7 Textura

O puede crear una textura:

patches/3-4-2-7-texture.pd

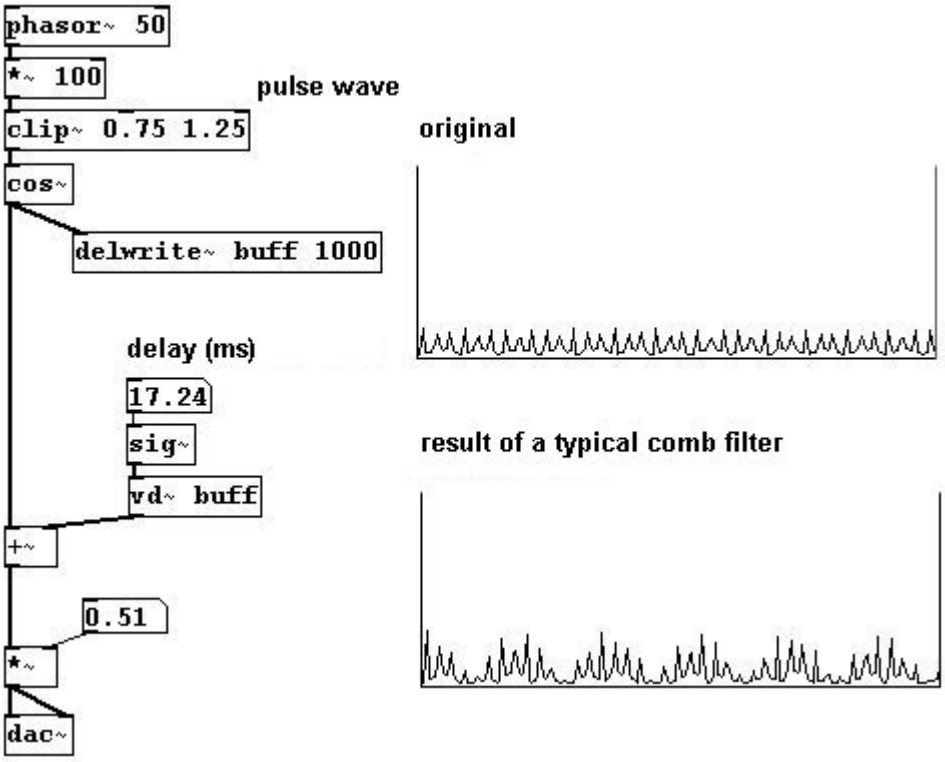


Debe tener cuidado que la retroalimentación no 'explote', es decir, que su volumen no se incremente exponencialmente.

3.4.2.8 Filtro peine (Comb filter)

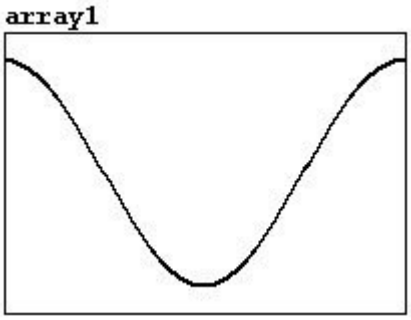
Puede construir un filtro peine usando retardo de audio. La idea fundamental es que a una señal original se le agrega un retardo de la misma. Como resultante obtenemos amplificaciones y cancelaciones a intervalos regulares, las cuales le dan al espectro la apariencia de un peine:

patches/3-4-2-8-combfilter.pd

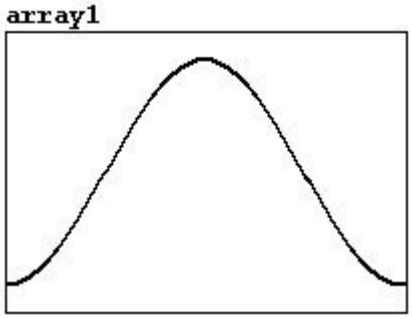


3.4.2.9 Octavador

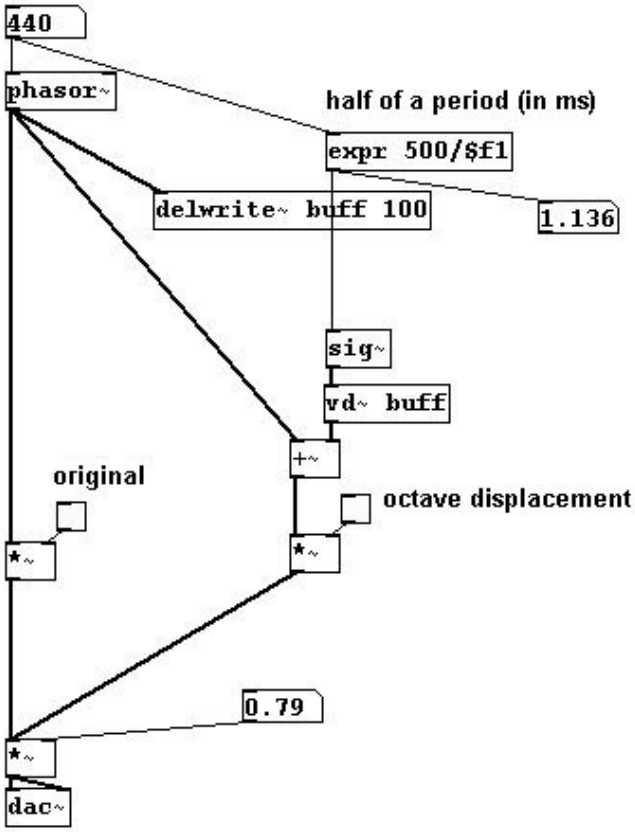
Si conoce la frecuencia fundamental de una señal, puede construir un octavador de la siguiente manera: Tomemos una onda...



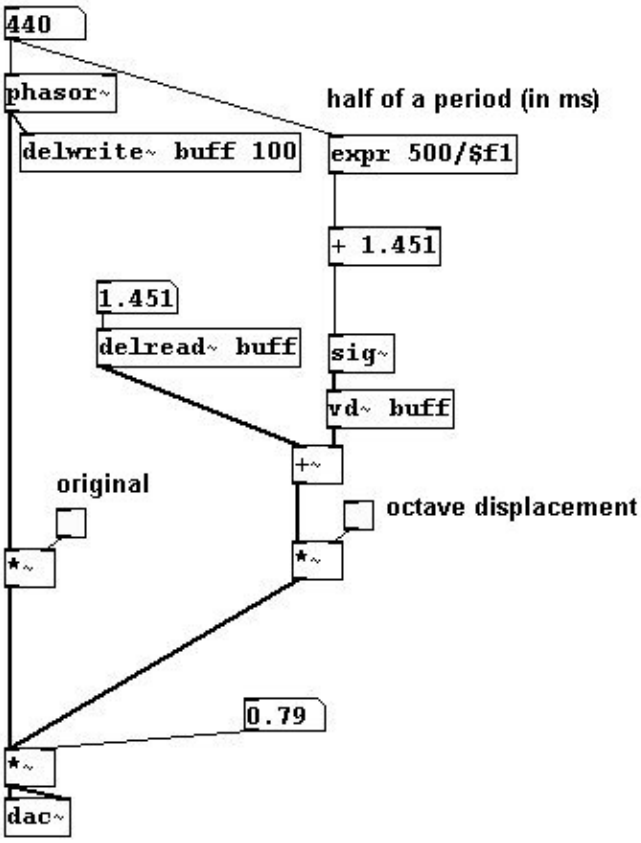
...y a esta misma señal retardada medio periodo...



...sumandolas nos da como resultado 0 (= cancelación). Si retarda una señal periódica medio periodo y lo suma a la señal original, el tono fundamental (y todos los parciales impares) son cancelados. Esto se vería de la siguiente forma:



Pero no funciona así. Debe recordar que Pd procesa todos los datos de audio en bloques de 64 muestras (a menos que cambie la configuración), porque es más eficiente que el procesar individualmente cada muestra (cf. 3.1.1.3.2). Con el patch citado anterior, obtenemos un retardo de 1136 milisegundos, o 50 muestras. Podemos mitigar este problema mediante el uso de un buffer con un retardo de un bloque (64 muestras = 1451 ms) para leer el original; lo mismo vale para el desplazamiento (offset) del retardo:

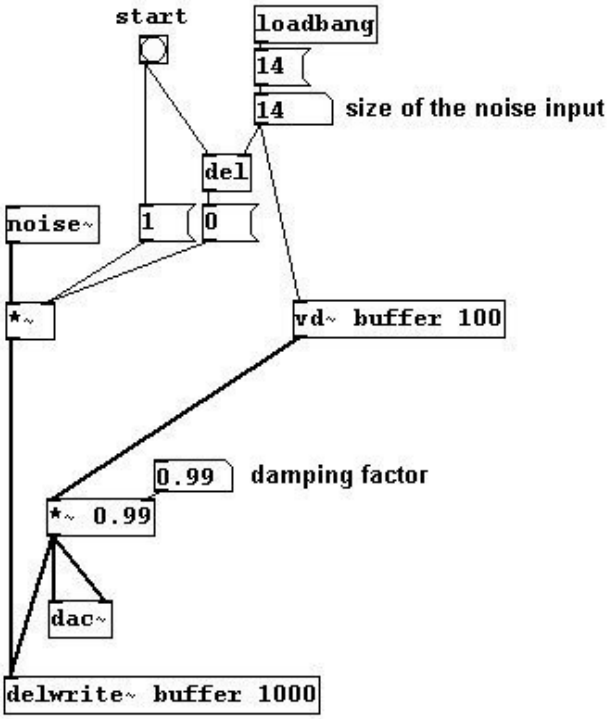


patches/3-4-2-9-oktavedoubler.pd

3.4.2.10 Algoritmo Karplus-Strong

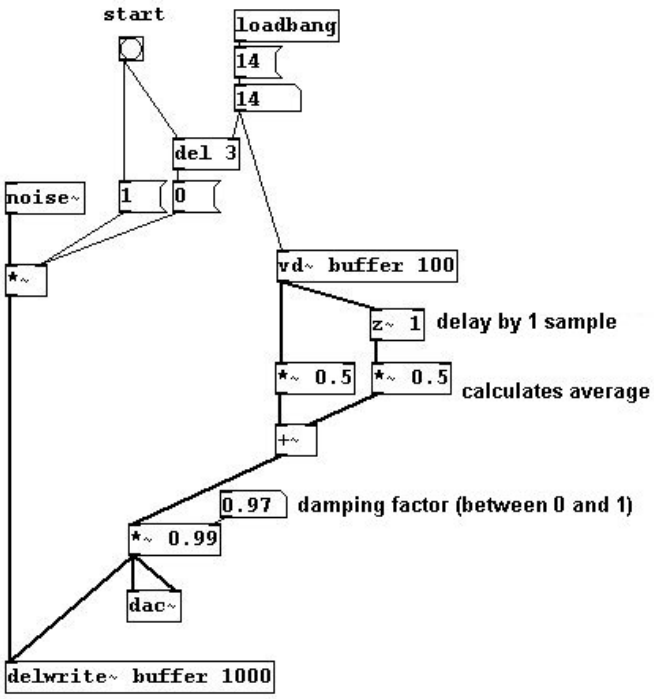
Un uso especial de bucle es el algoritmo Karplus-Strong. Es uno de los primeros ejemplos de síntesis por modelado físico, un proceso que intenta reproducir exactamente lo que sucede cuando un material físico vibra. En nuestro ejemplo, nuestro modelo físico es una cuerda punteada. Cuando se puntea una cuerda, esta en principio vibra caóticamente y luego se ajusta a sí misma al largo de la cuerda. También pierde energía, es decir, se desvanece gradualmente. Este proceso puede ser reconstruido matemáticamente tomando una porción de ruido blanco y reproduciéndolo periódicamente una y otra vez mediante la escritura a y lectura desde un buffer:

patches/3-4-2-10-karplus-strong1.pd

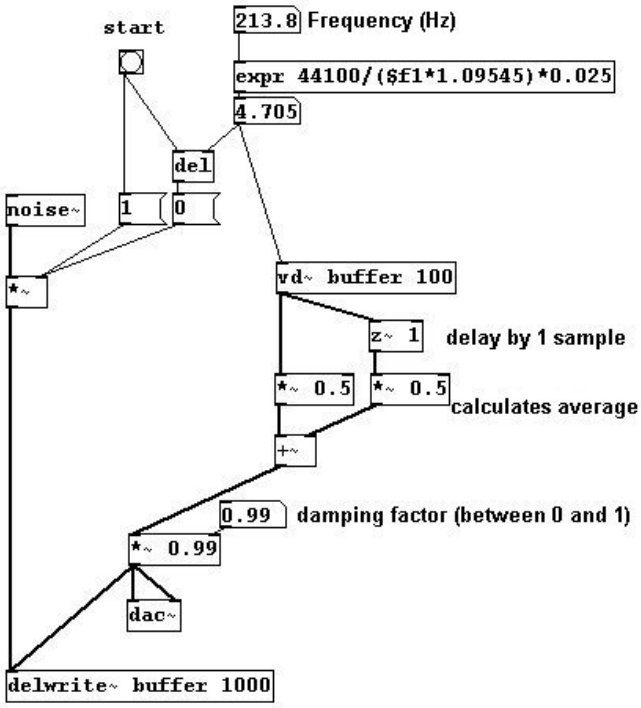


El efecto de la cuerda puede ser mejorado si el material con el cual comenzamos vibra más y más 'suavemente'. Esto se realiza mediante el cálculo del promedio: se toma el promedio de cada dos y se escribe este resultado dentro de un buffer en lugar de los valores originales. La vibración se vuelve menos y menos 'angular'. Utilice el Objeto "z~" (Pd-extended) para configurar el retardo a una muestra; ingrese el número de muestras como argumento:

patches/3-4-2-10-karplus-strong2.pd



El tono es diferente cada vez. Esto se debe a que "noise~" produce números aleatorios, los cuales, naturalmente, son diferentes todo el tiempo. Podríamos agregar el cálculo para las frecuencias resultantes:



patches/3-4-2-10-karplus-strong3.pd

3.4.2.11 Más ejercicios

a) Construya una función de grabación dentro del Sample-Player.

b) Cree un patch para reverberar o una textura con diferentes tiempos de retardo para la señal entrante, por ejemplo, con múltiplos de la serie de Fibonacci (en donde el siguiente número es siempre la suma de los dos previos: 0 1 1 2 3 5 8 13).

c) Utilice distintos sonidos Karplus-Strong para hacer texturas de distintas densidades.

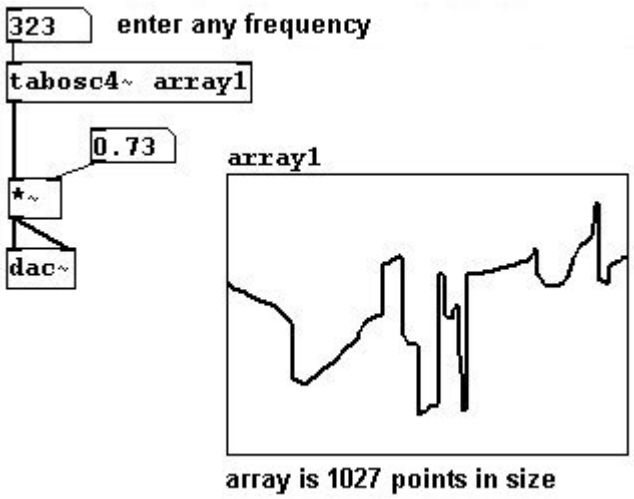
d) Aplique un filtro peine a patches presentados en secciones previas.

3.4.3 Apéndice

3.4.3.1 Oscilador de matriz

Una manera de simplificar la combinación de "tabread~" y una señal "phasor~" multiplicada, es utilizar el Objeto "tabosc4~". Este lee una matriz por la frecuencia que usted indique. Una limitación de este método es que el tamaño de la matriz debe ser una potencia de dos (ej.: 128, 512, 1024) más tres puntos (aquí, 1027 = 1024 + 3).

patches/3-4-3-1-arrayoscillator.pd

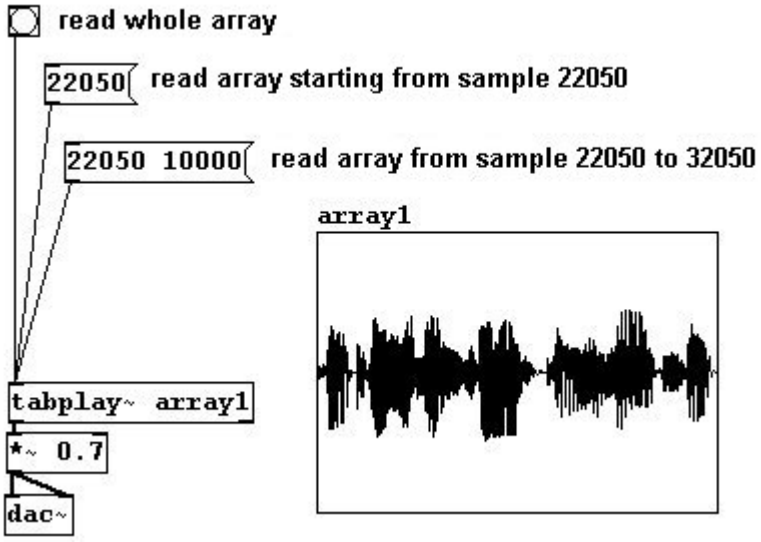


Esto ya hace una mención al "modelado de onda" (wave shaping), un tema que será explicado en el Capítulo 3.5. Mientras tanto, puede entretenerse dibujando cualquier onda dentro de la matriz usando el mouse.

3.4.3.2 Reproducción de una matriz

Otra simplificación es "tabplay~"; simplemente reproduce una matriz a la velocidad original (cuando registra en su entrada un bang). it simply plays an array back at the original speed (when banged). De acuerdo a su conveniencia, puede configurar los puntos de inicio y finalización para la reproducción (punto de inicio y duración en muestras):

patches/3-4-3-2-simply-play-array.pd

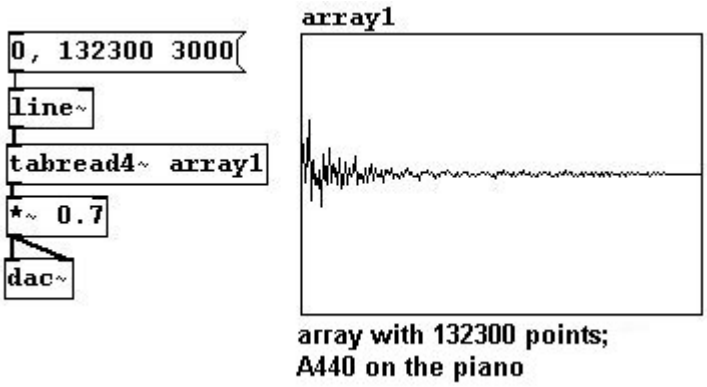


3.4.3.3 Reproducción de una matriz en un bloque

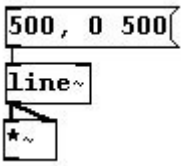
Los Objetos "tabwrite~" y "tabread~" también tienen otra forma especial: "tabsend~" y "tabreceive~". Estos escriben/leen una matriz en sincronización con los "bloques" (3.1.1.3.2). "tabsend~" escribe cada bloque en una matriz (por supuesto que estos deben ser del tamaño de un bloque, que por defecto en Pd es de 64 muestras). "tabreceive~" lee cada bloque de la matriz. Regresaremos a este punto más tarde en el Capítulo sobre FFT (3.8).

3.4.3.4 Glissando de muestras

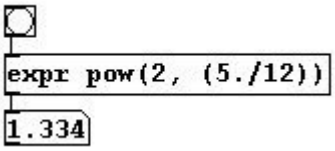
Sabemos que podemos reproducir una matriz a la altura normal o una octava más alta, etc. ¿Pero que ocurre si queremos un glissando desde la octava a la altura original? Para lograrlo, necesitamos subdividir el problema en "indicador principal" y "adicción". El "indicador principal" corre a velocidad normal en la matriz. Usemos una matriz con 132300 puntos como ejemplo, la cual equivale a 3000 milisegundos:



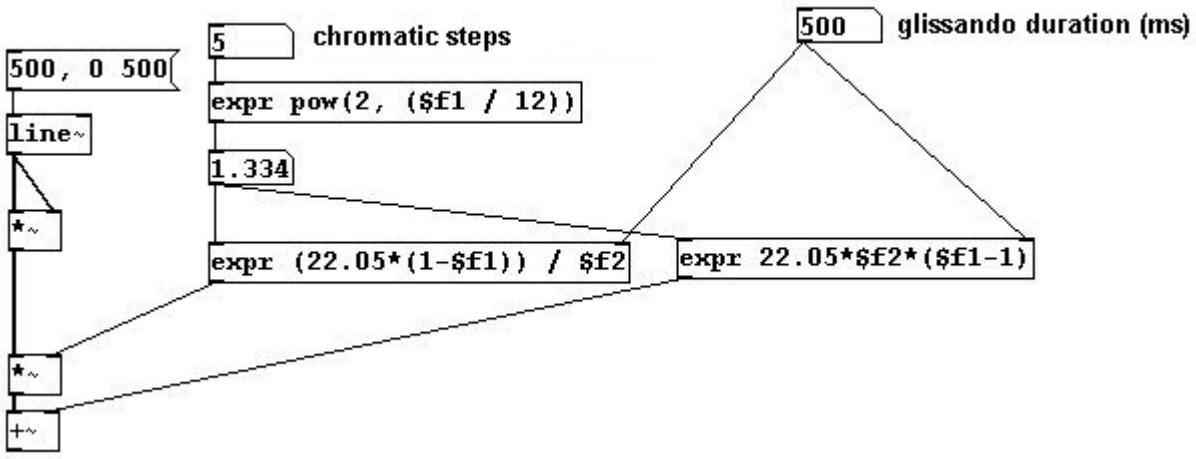
Luego viene la "adicción", la cual realiza el glissando. Utilicemos un glissando que comience 5 semitonos por encima de la altura original y regrese a la altura original en 500 milisegundos. Necesita hacer un "line~" de esto en reverso, y luego potenciar los valores:



Por encima de este, debe determinar el factor para las frecuencias de los cinco semitonos (cf. 3.1.1.4.3)...

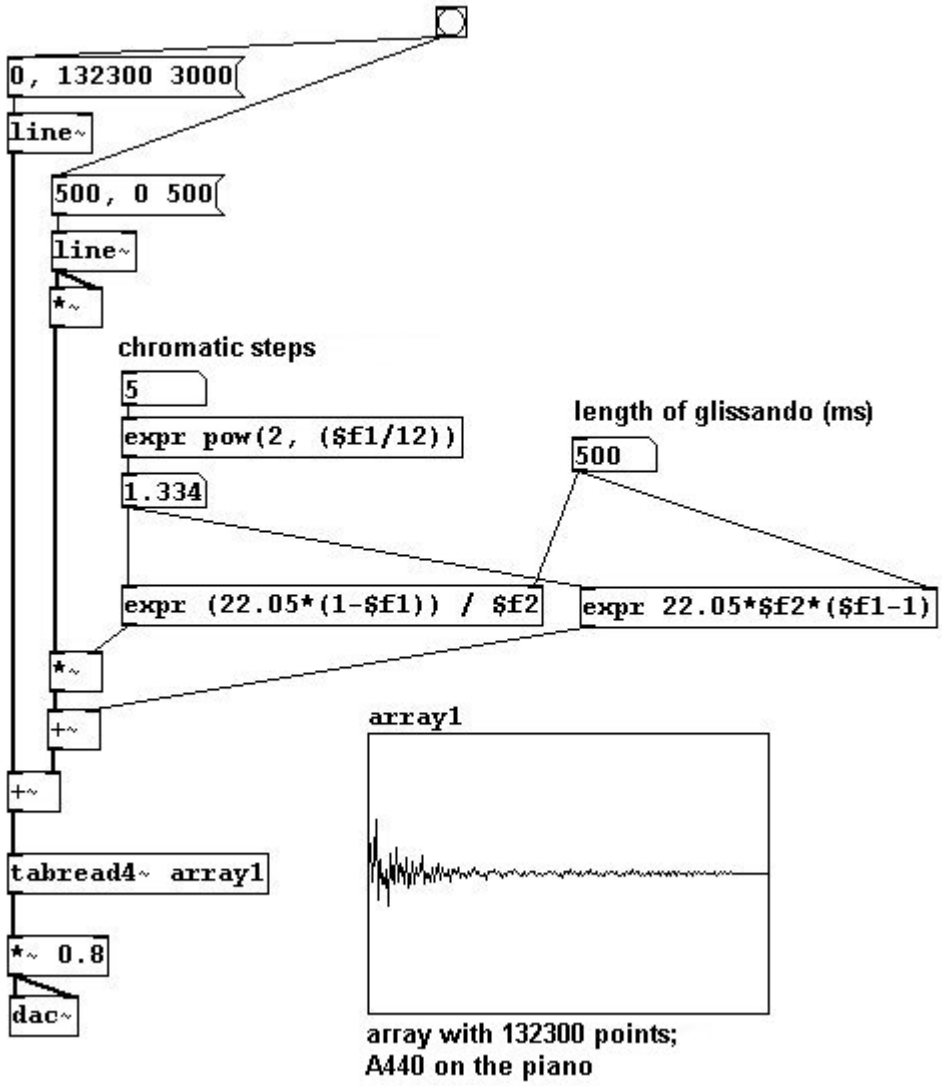


...y finalmente realizar el siguiente cálculo:



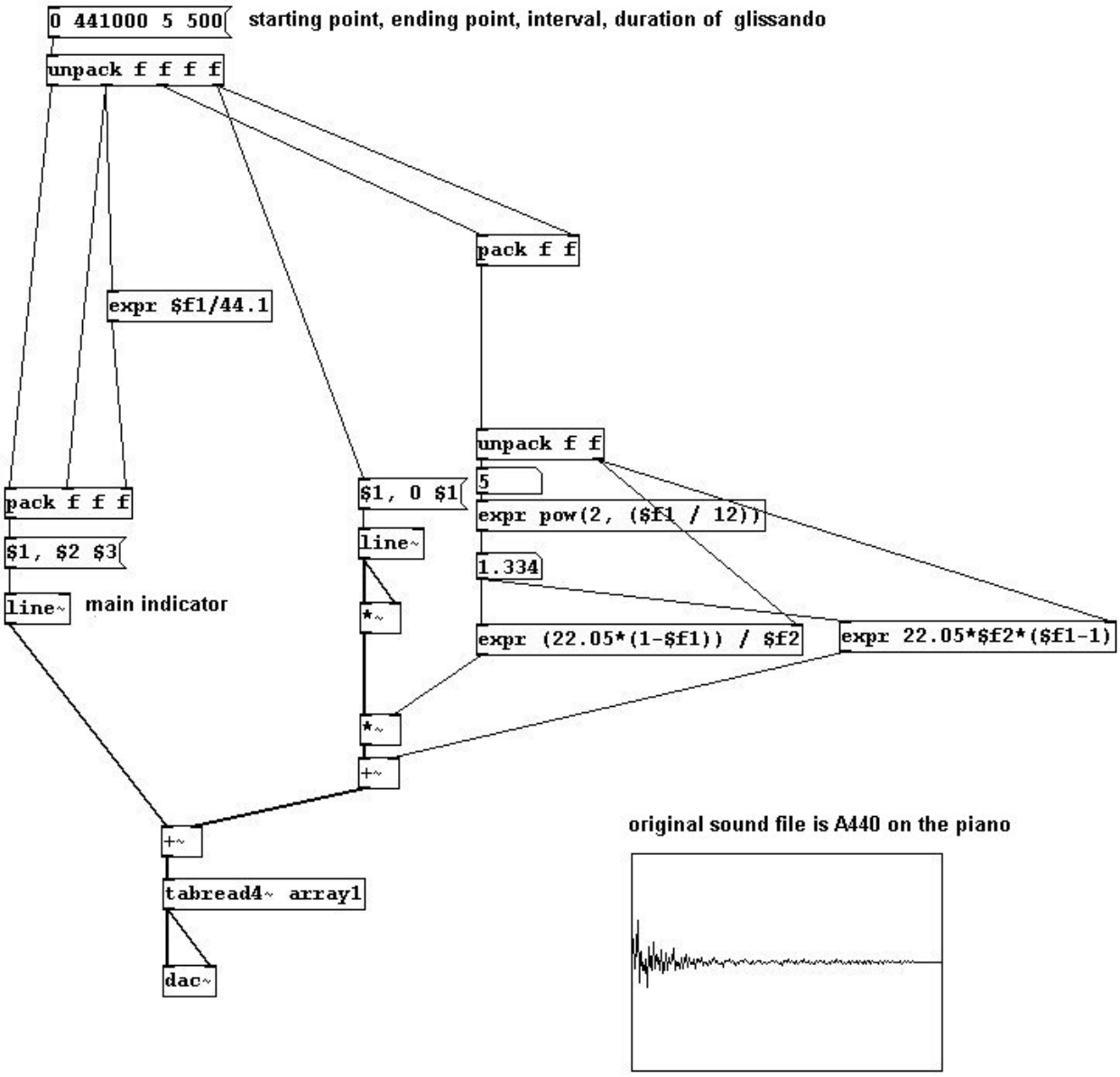
Esta es la "adición" que es sumada al "indicador principal":

patches/3-4-3-4-sample-glissando1.pd



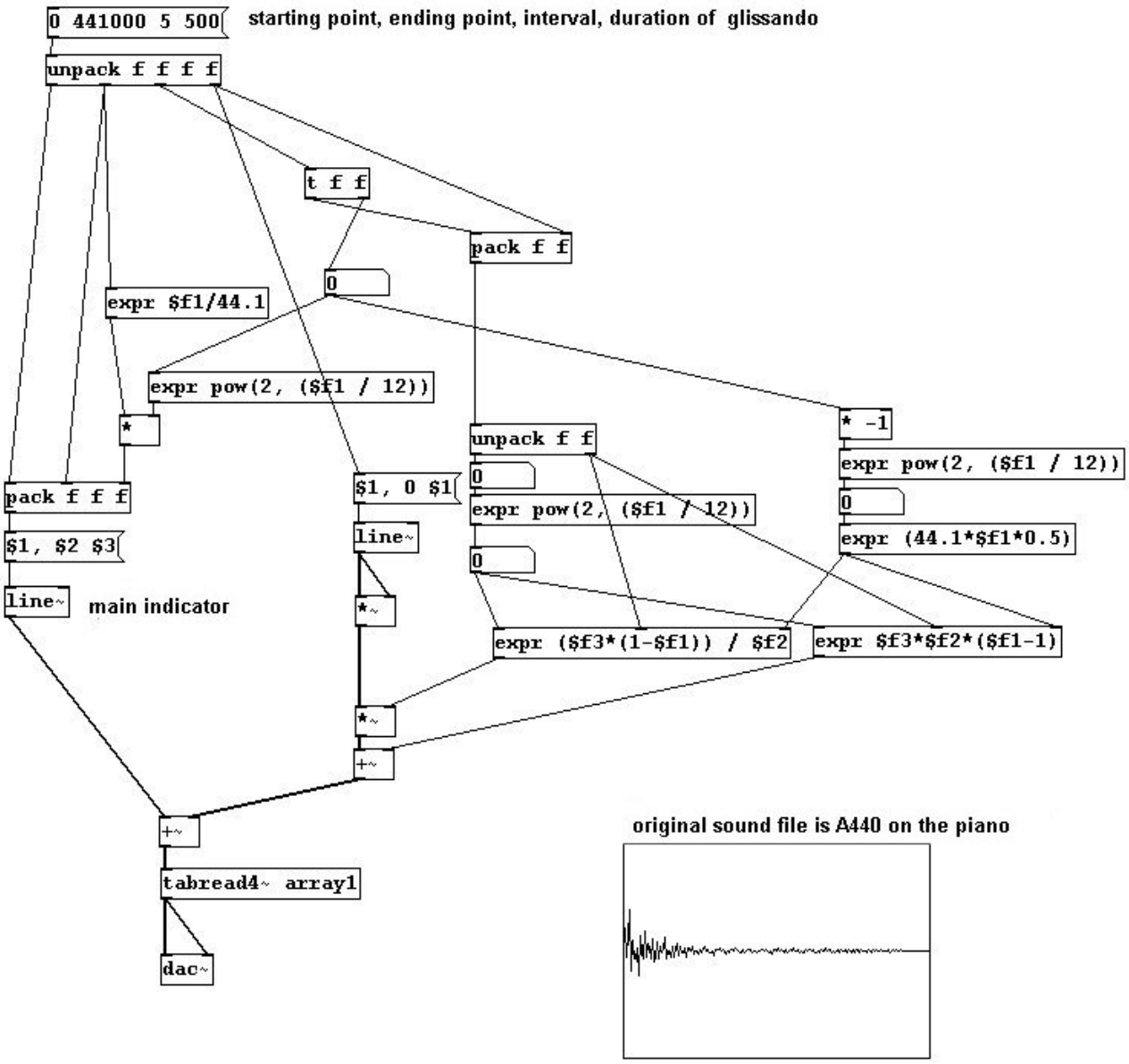
Puede usar cualquier glissando en la altura en cuestión que desee; incluso son posibles los valores negativos:

patches/3-4-3-4-sample-glissando2.pd



Inversamente, para alejarnos de la altura original:

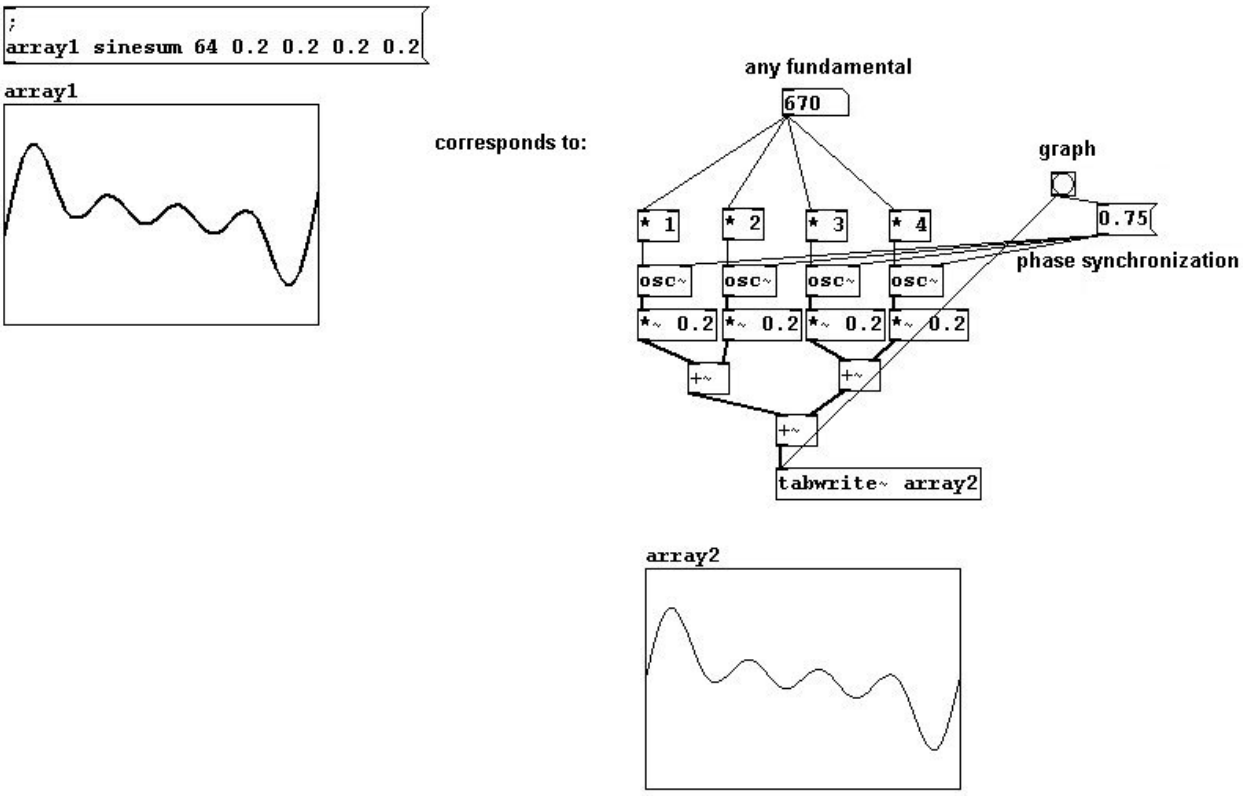
patches/3-4-3-4-sample-glissando3.pd



3.4.3.5 Síntesis aditiva con matriz

Como función especial en Pd, podemos crear una suma de tonos sinusoidales dentro de una matriz - es decir, síntesis aditiva con explicamos en el Capítulo 3.2. Esto se logra utilizando el mensaje "sinesum". El primer argumento corresponde al (nuevo) tamaño de la matriz (debe ser una potencia de 2; tres puntos se le añadirán automáticamente para asegurar una óptima conección del comienzo y finalización de una fase) y también los factores de volumen para cualquier número de parcial:

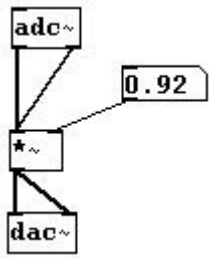
patches/3-4-3-5-sinesum.pd



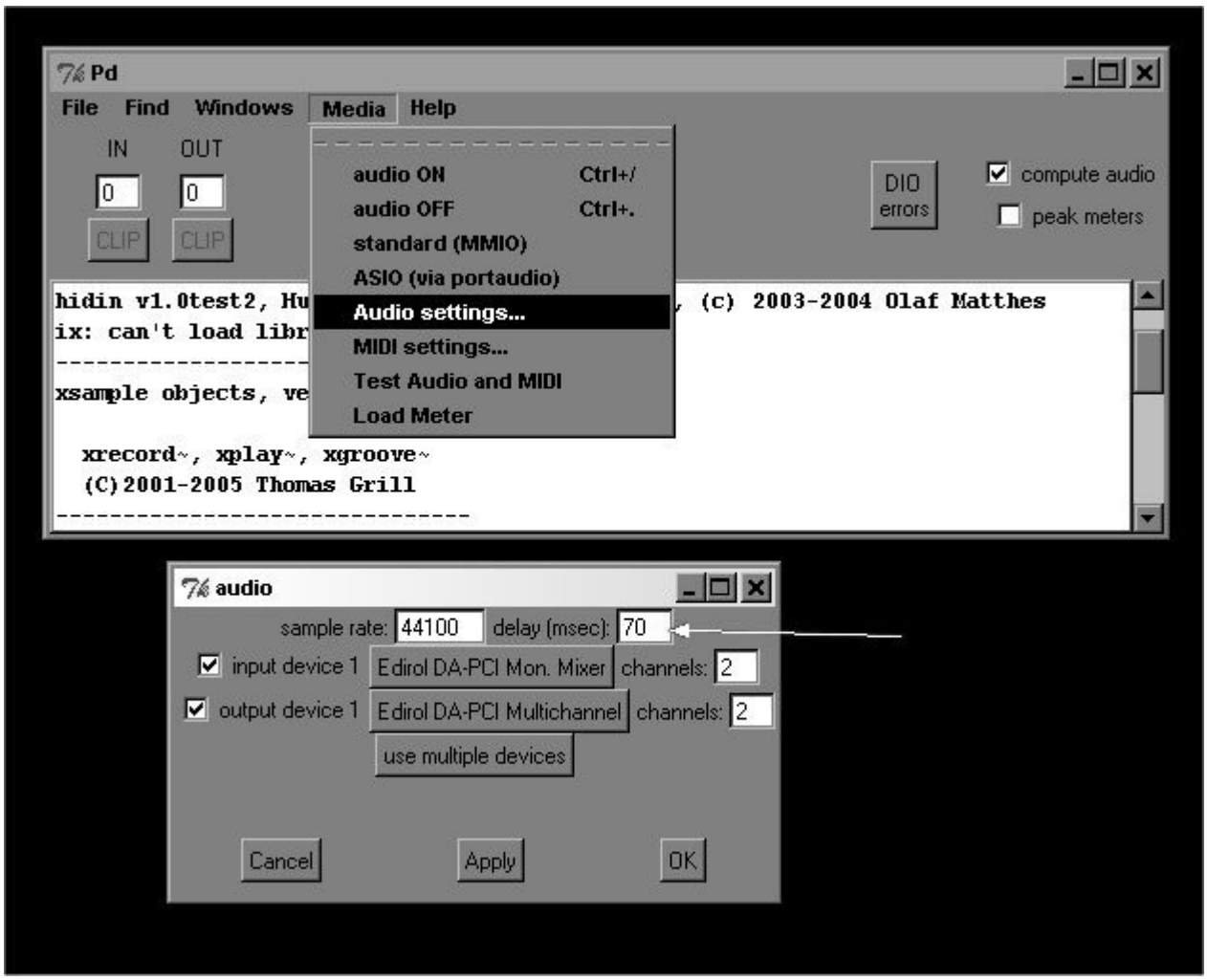
En vez de ondas sinusoidales, también puede utilizar "cosinesum" para trabajar con ondas coseno.

3.4.3.6 Latencia

A veces ocurren retardos de audio cuando no lo queremos. Incluso podemos escucharlo cuando tenemos conectado un micrófono a los altoparlantes y hacemos un ruido extremadamente breve en frente del mismo, como por ejemplo chasquear los dedos:



La tarjeta de sonido y especialmente el sistema operativo determinan la longitud de esta "latencia". Idealmente la latencia es tan pequeña (por debajo de los 5 ms) que el oido humano no puede percibir el retardo. Esto requiere un procesador de ordenador muy rápido, una buena tarjeta de sonido, y un sistema operativo apropiado. Puede configurar la latencia bajo **Media** > **Audio settings**:



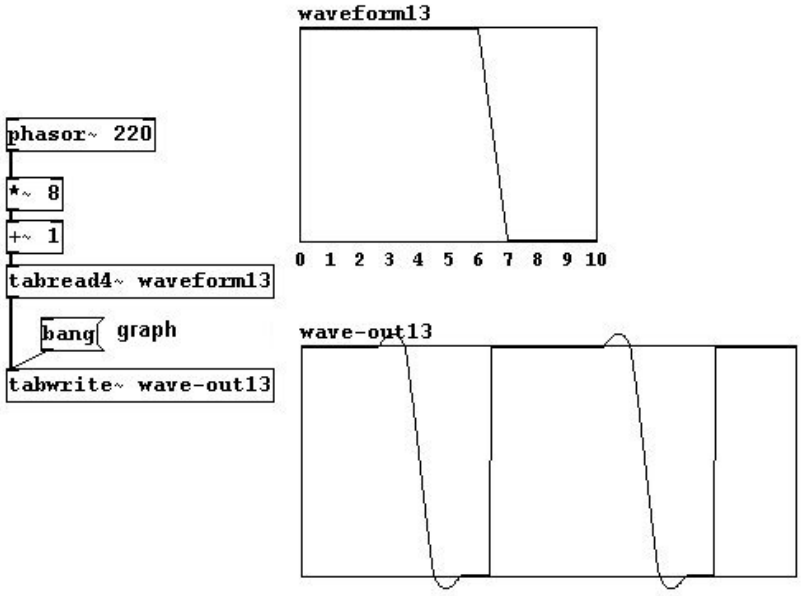
Bajo Microsoft Windows, no puede actualmente (Junio 2008) lograr una latencia menor a 50 ms sin causar errores.

3.4.4 Para los interesados, especialmente

3.4.4.1 Interpolación de 4-puntos

En este ejemplo puede ver cómo funciona la interpolación en "tabread4~":

patches/3-4-4-1-four-point-interpolation.pd

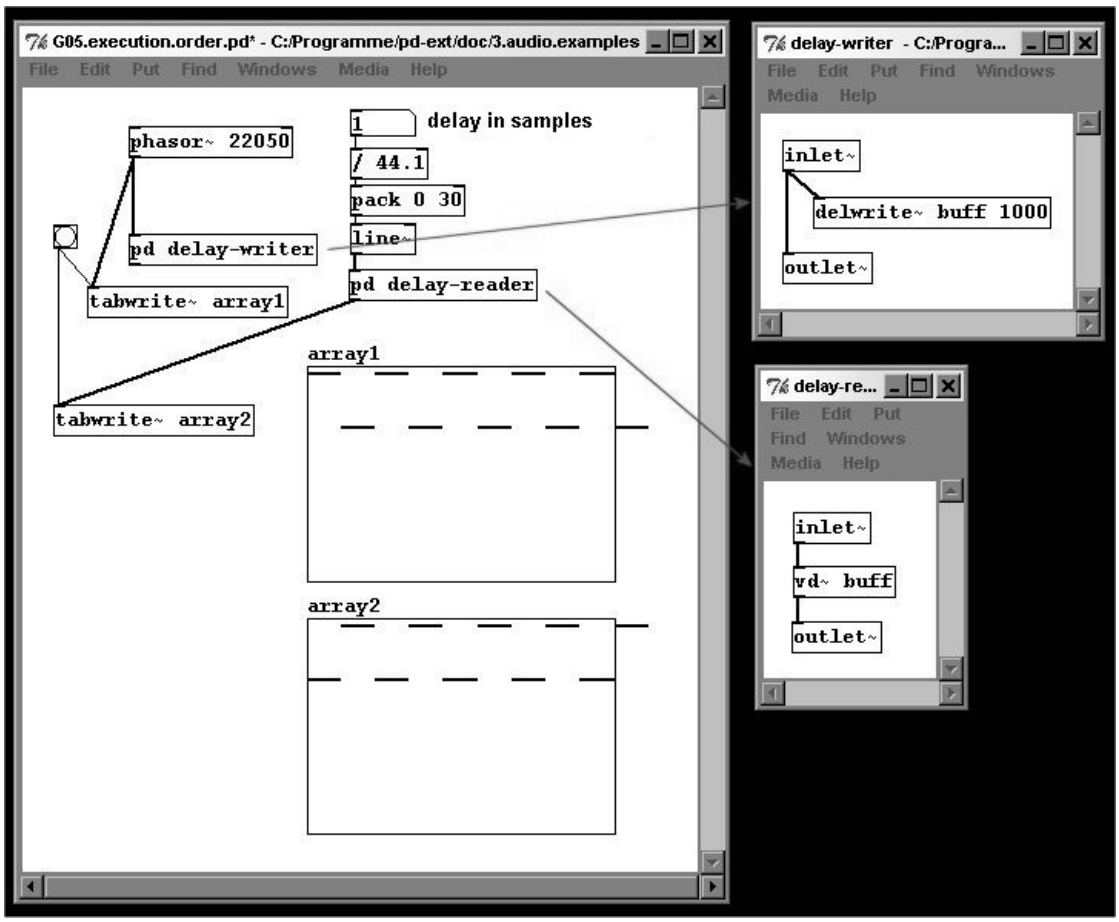


El salto de 1 a -1 es 'suavizado' por un tipo de interpolación sinusoidal. Como el nombre lo indica, se usan y alteran *cuatro puntos*: los *dos directamente enfrente* y los *dos directamente detrás* del intervalo que quiere interpolar.

3.4.4.2 Retardo prudente de muestras

Una manera de retardar algo por un cierto número de muestras con "delread~" y "vd~" es usando un subpatch (de otra manera tendremos el problema del tamaño de bloque, como describimos anteriormente en relación con el desplazamiento de octava):

patches/3-4-4-2-samplewise-delay.pd



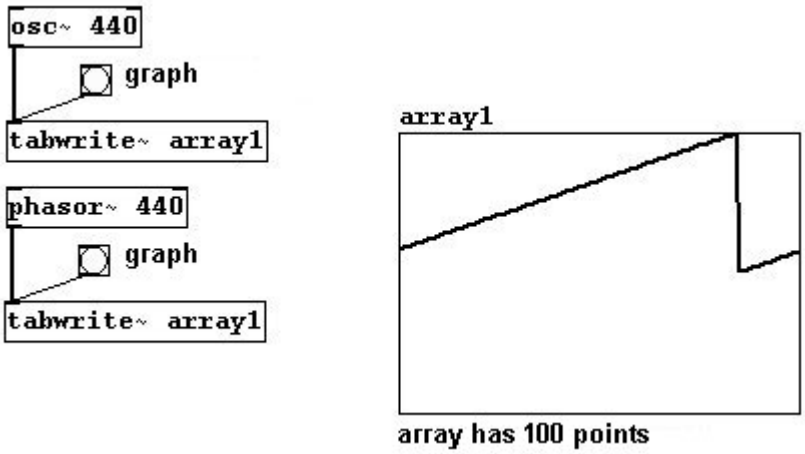
3.5 Modelado de onda

3.5.1 Teoría

3.5.1.1 Formas de onda

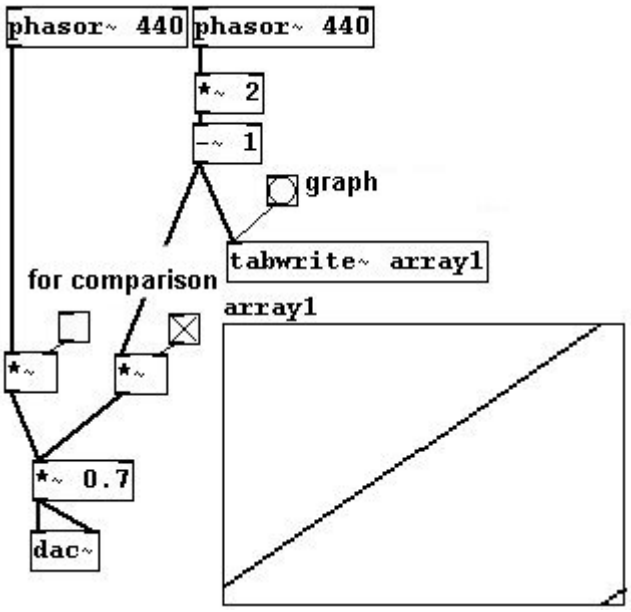
En 3.1.1.1.2 usted aprendió diferentes formas de onda (seno, dientes de sierra, triángulo, cuadrada e impulso). Pd contiene Objetos para dos de estos, a saber "osc~" para ondas sinusoidales y "phasor~" para dientes de sierra. Puede usar una matriz para visualizar las formas de onda:

patches/3-5-1-1-waveform-graph.pd



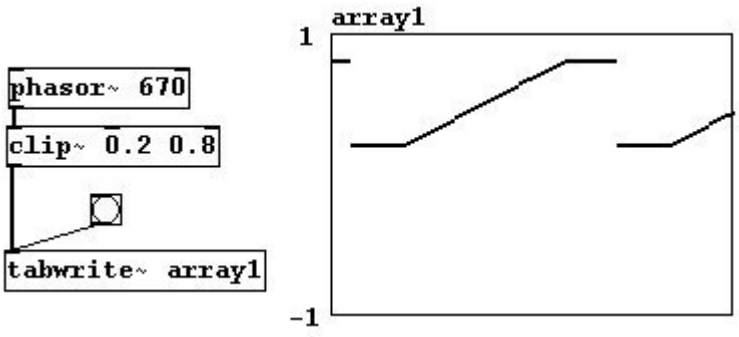
Note Bien: El diente de sierra del Objeto "phasor~" siempre cubre el rango de 0 a 1; nunca llega a valores negativos. Sin embargo puede hacerlo más fuerte realizando un pequeño cálculo:

patches/3-5-1-1-strong-phasor.pd



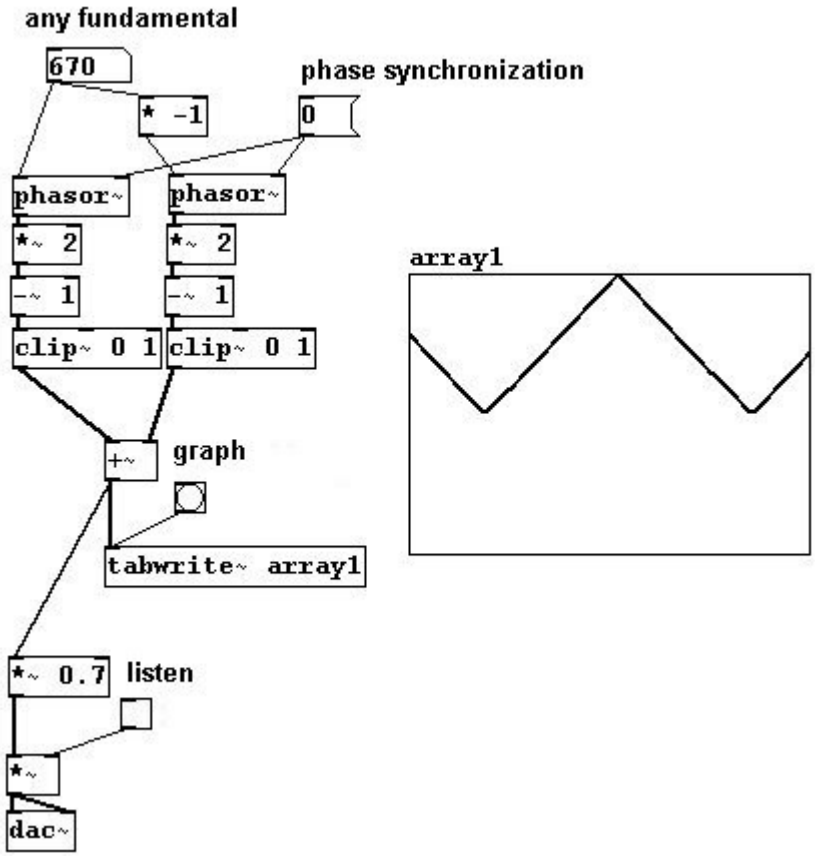
También puede crear otras formas de onda mediante el agregado de algunas operaciones al "phasor~". Para lograr esto, necesita un nuevo Objeto: "clip~", el cual recorta todo lo que exceda el rango indicado. Como argumentos ingrese dos números para los límites inferior y superior; los números fuera de dicho rango serán 'recortados':

patches/3-5-1-1-other-waveforms.pd



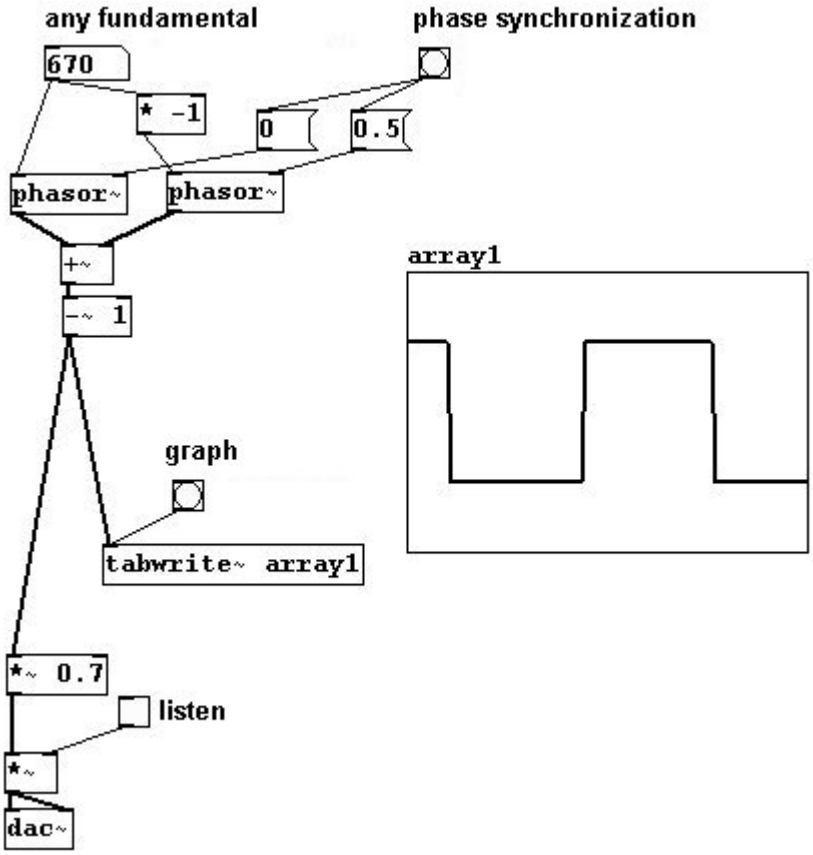
Ahora sobre la onda triangular:

patches/3-5-1-1-triangel.pd



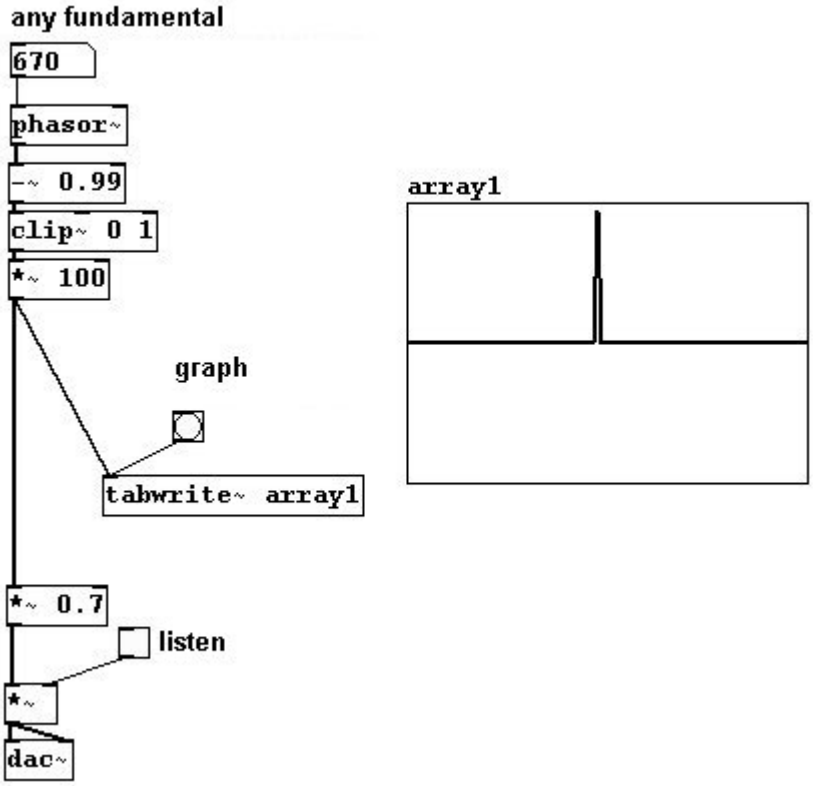
Onda cuadrada:

patches/3-5-1-1-square.pd



Impulso:

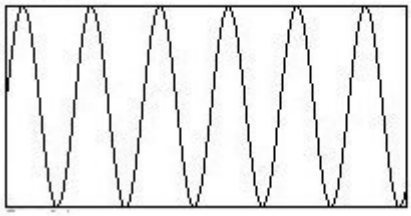
patches/3-5-1-1-pulse.pd



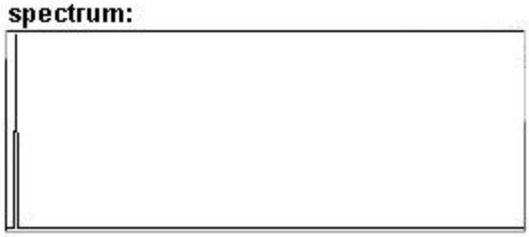
Estas son todas las formas de onda estandard que exciben ciertas características:

Seno: un tono simple sin ningún hipertono

wave:

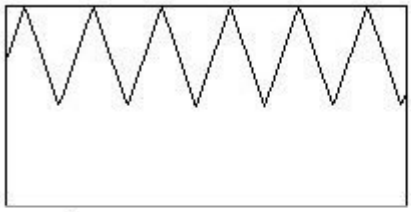


spectrum:



Triángulo: Como la onda seno, pero con los parciales impares también

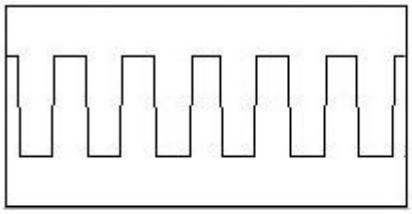
wave:



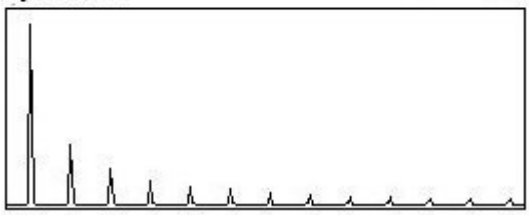
spectrum:

Cuadrada: sólo los parciales impares

wave:

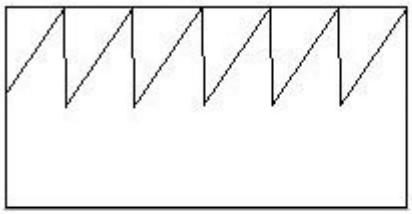


spectrum:

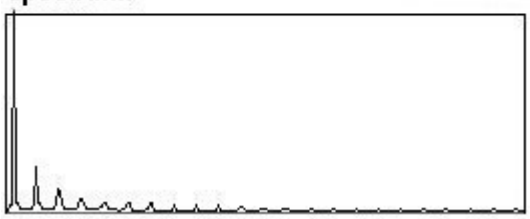


Diente de sierra: todos los parciales

wave:

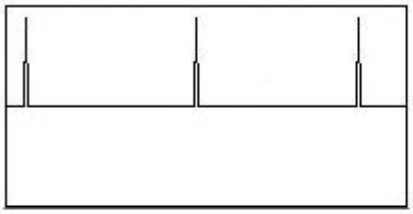


spectrum:

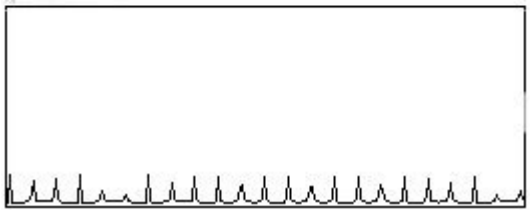


impulso: todos los parciales presentes con intensidades casi iguales

wave:



spectrum:



Puede observar que las formas de onda simátricas exhiben sólo los parciales impares (dentro de cada periodo, exactamente dos periodos de cada acceso posterior al parcial impar; a esto llamé simétrico), mientras que las formas de onda asimétricas exhiben también los parciales pares.

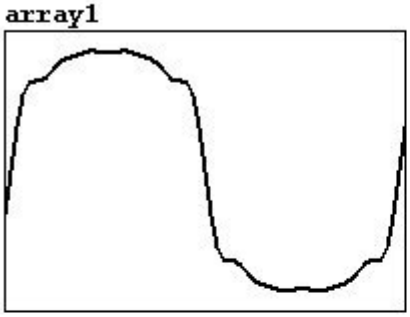
Estas formas de onda también pueden ser realizadas usando síntesis aditiva:

patches/3-5-1-1-waveform-fourier.pd

becoming a square wave:

```
;
array1 sinesum 64 1 0 0.2 0 0.1 0 0.08 0 0.05 0 0.03
```

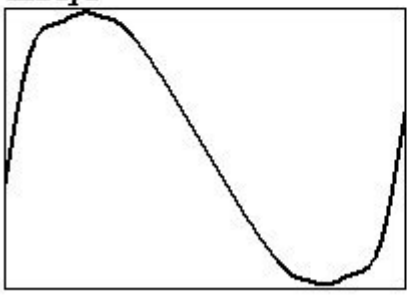
array1



becoming a sawtooth:

```
;
array2 sinesum 64 1 0.2 0.1 0.08 0.05 0.03 0.02 0.01
```

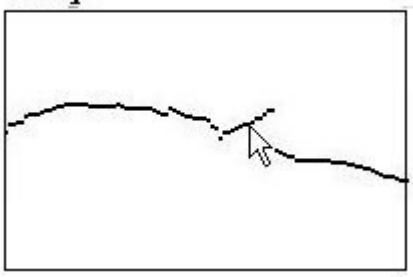
array2



etc.

Existe aquí un espacio ilimitado para la experimentación y es esta una de las vias para lograr nuevos sonidos. También puede dibujar sus propias formas de onda directamente sobre la matriz. Para hacerlo, se debe encontrar en el modo de ejecución y mover el ratón a una linea de la matriz. La flecha del cursor cambia de dirección:

array1



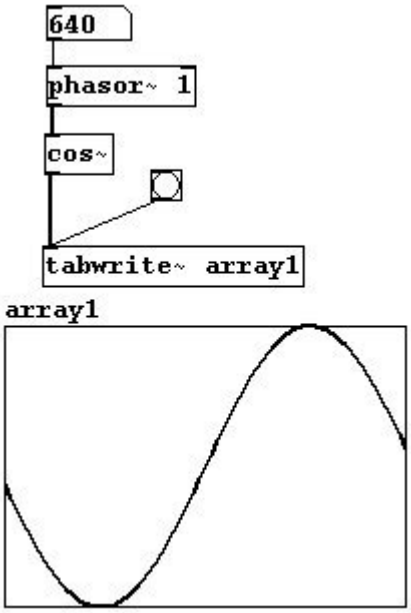
Mientras mantiene presionado el botón izquierdo del ratón, mueva el mismo para dibujar su forma de onda.

Sin embargo esto es algo tedioso y poco 'elegante'. Examinemos entonces la teoría del 'modelado de onda":

3.5.1.2 Función de transferencia

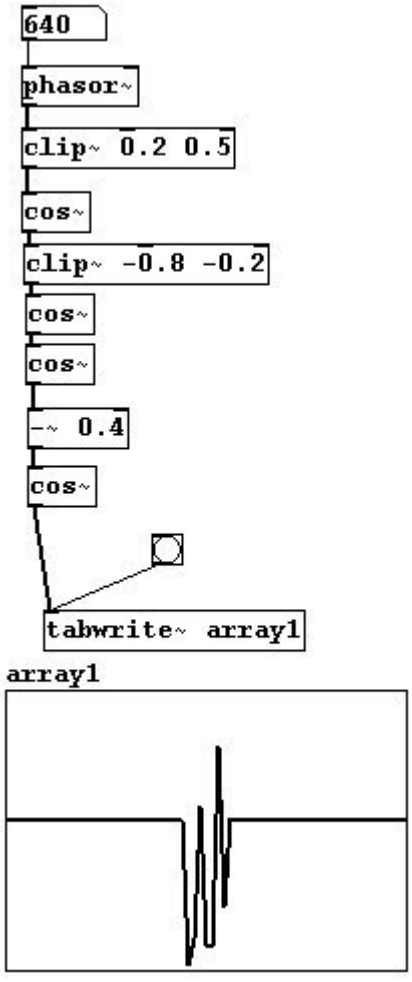
Una función lineal experimenta lo que se llama "función de transferencia". Para funciones lineales

puede usar "phasor~", el cual siempre se dirige de 0 a 1. Por ejemplo, puede realizar una onda coseno usando el Objeto "cos~" el cual calcula una función coseno:



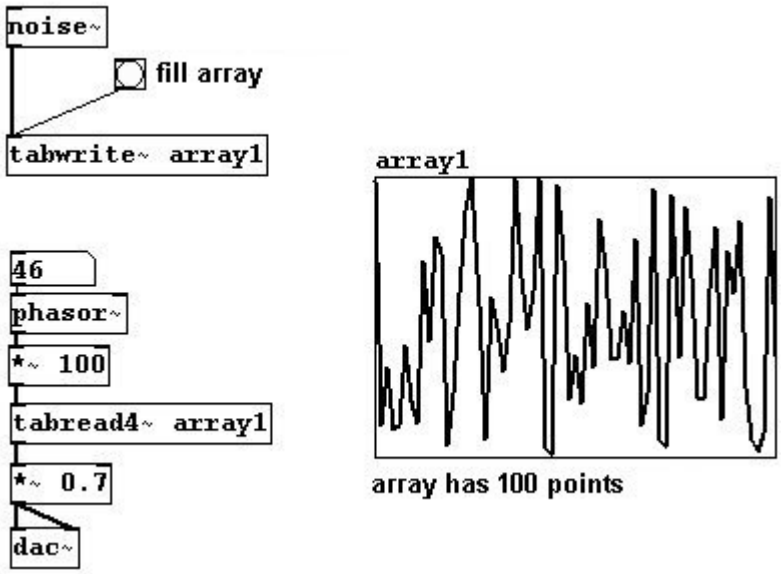
Puede realizar todo tipo de transferencias de esta manera; aquí tiene otro ejemplo:

patches/3-5-1-2-transferfunction.pd



3.5.1.3 Formas de onda aleatorias (controladas)

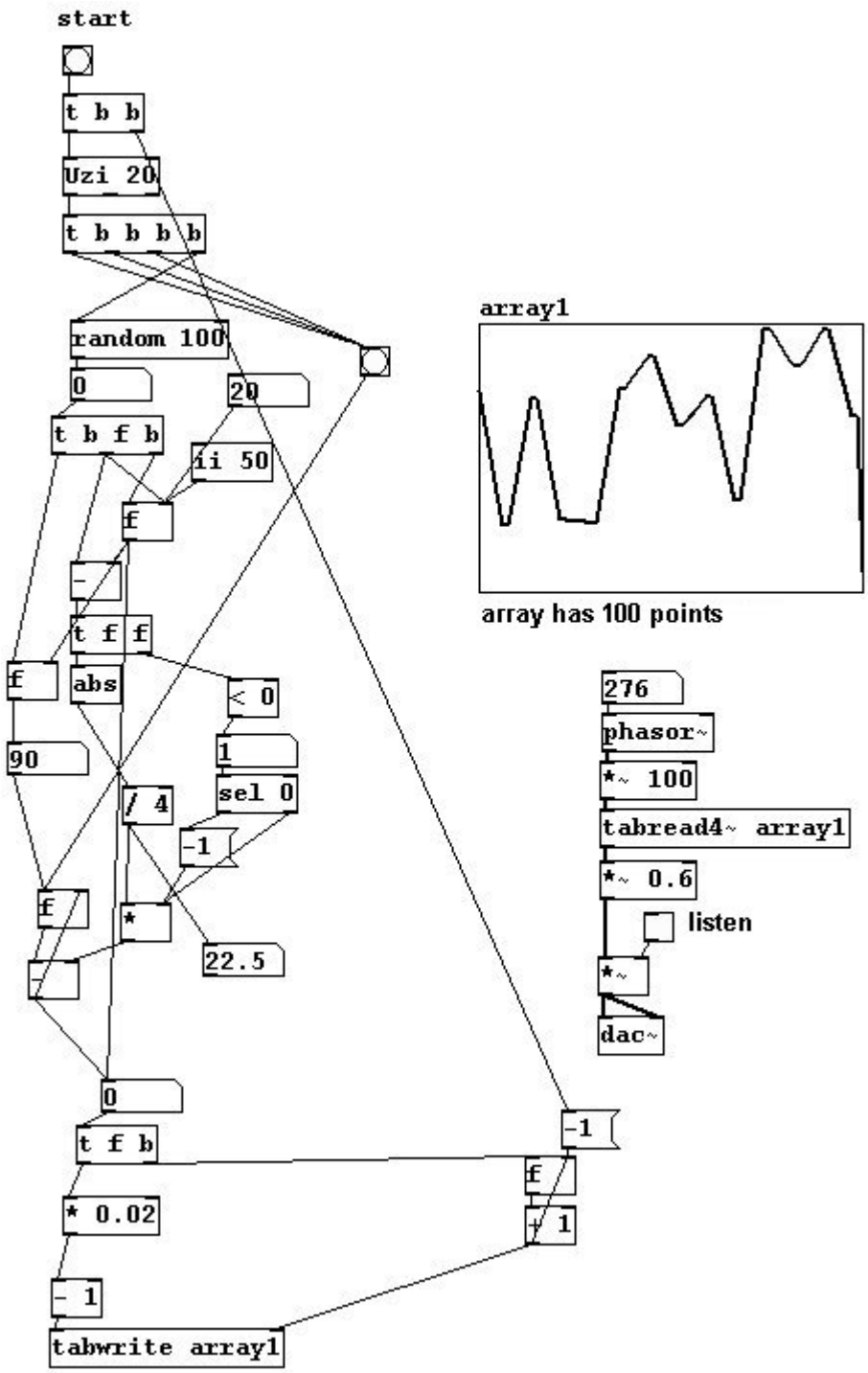
También puede grabar ruido dentro de una matriz y luego leerla periódicamente -es decir, leer el mismo contenido una y otra vez (cf. Karplus-Strong). Si hace esto más de 20 veces por segundo, escuchará un tono:



El espectro de este sonido es naturalmente bastante impredecibleCada vez que llene la matriz con números aleatorios con el Objeto "noise~", obtendrá una nueva onda con nuevas características.

Pero todavía hay algo sistémico trabajando. Puede, por ejemplo, interpolar todos los clicks (grandes saltos en la forma de onda) para conseguir una forma de onda resultante más suavizada. Generemos ahora puntos aleatorios usando interpolación lineal ("Uzi" (Pd-extended) genera el número de bangs especificado en su argumento y los envía a la salida lo más rápido posible):

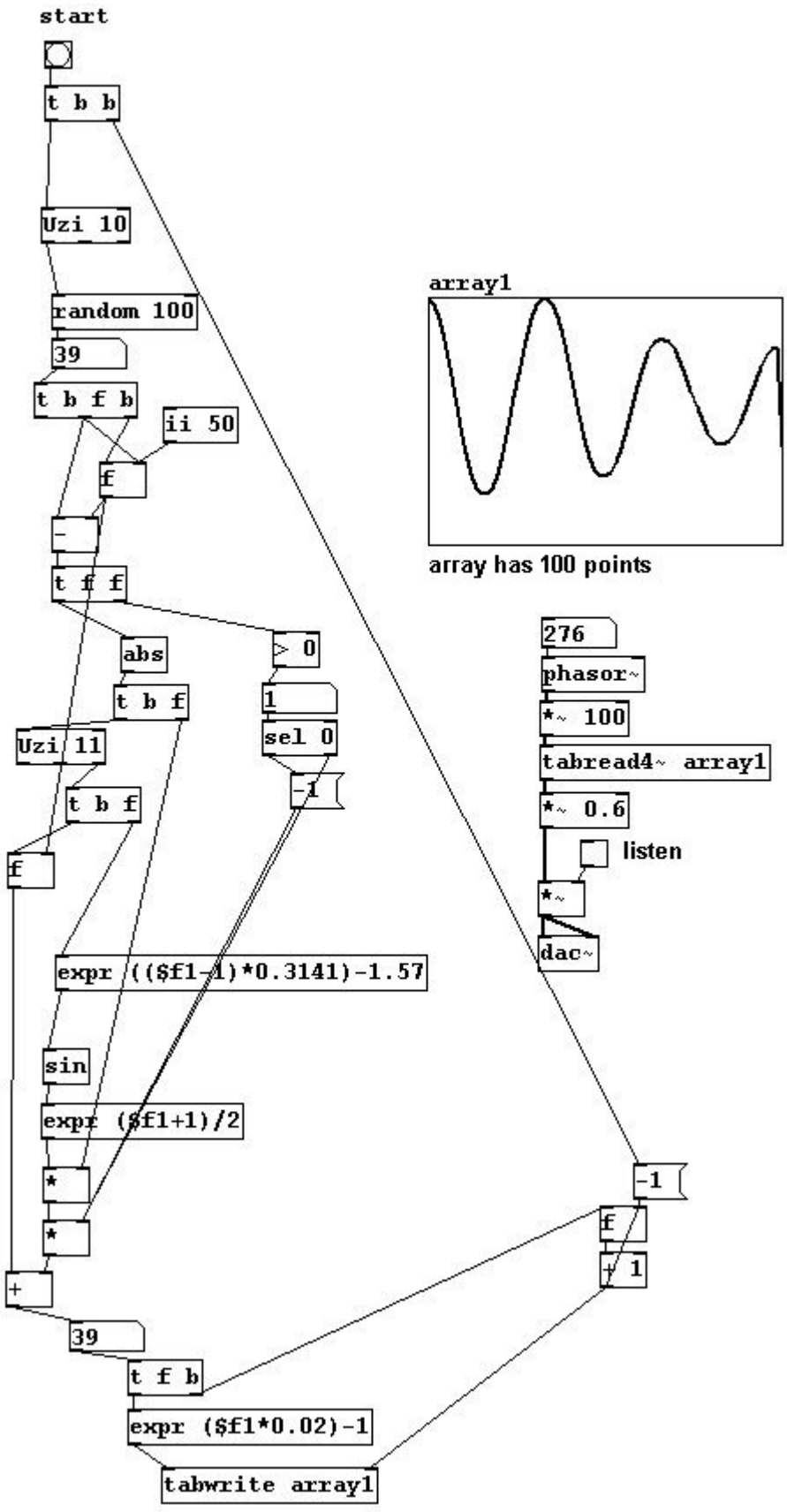
patches/3-5-1-3-wavgorithm.pd



En este ejemplo siempre se usan cuatro puntos para la interpolaciones.

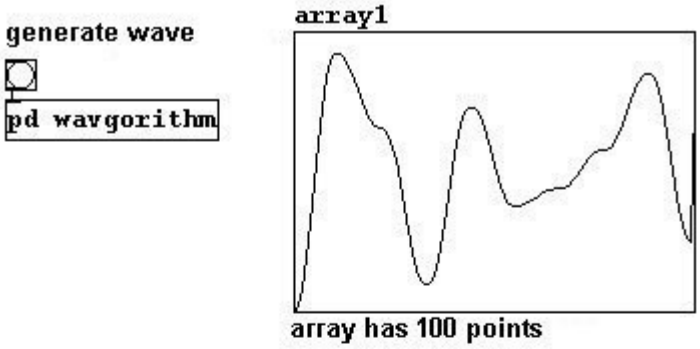
El resultado será mucho más suave si usa una interpolación sinusoidal a cambio de la lineal. Aquí son usados diez puntos para las interpolaciones:

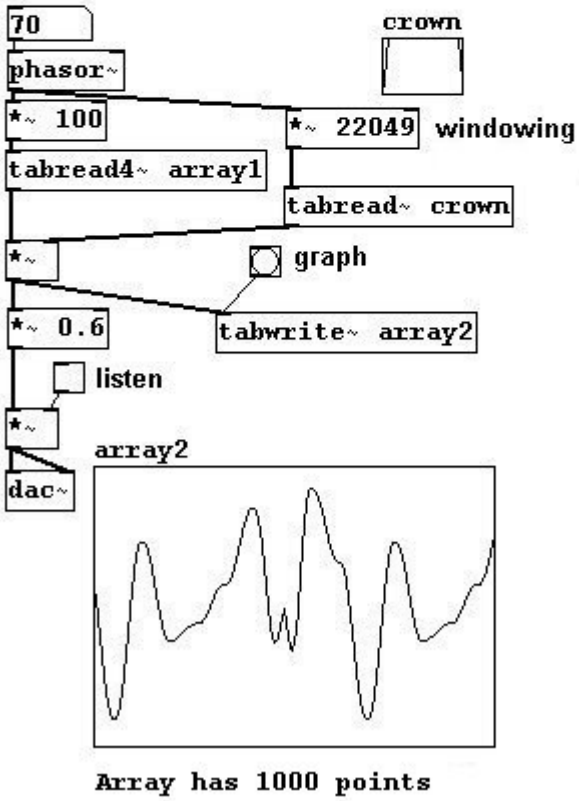
patches/3-5-1-3-wavgorithm+sin.pd



Ahora debemos interpolar también la conección del final del periodo con su principio. Usaremos un ventaneo para esto y programaremos el cálculo dentro de un subpatch.

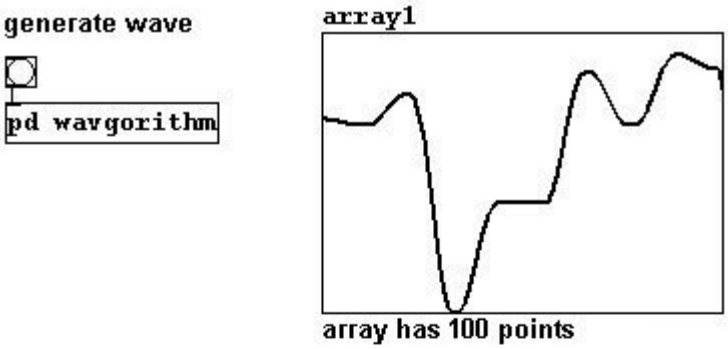
patches/3-5-1-3-wavgorithm+sin+fenster.pd

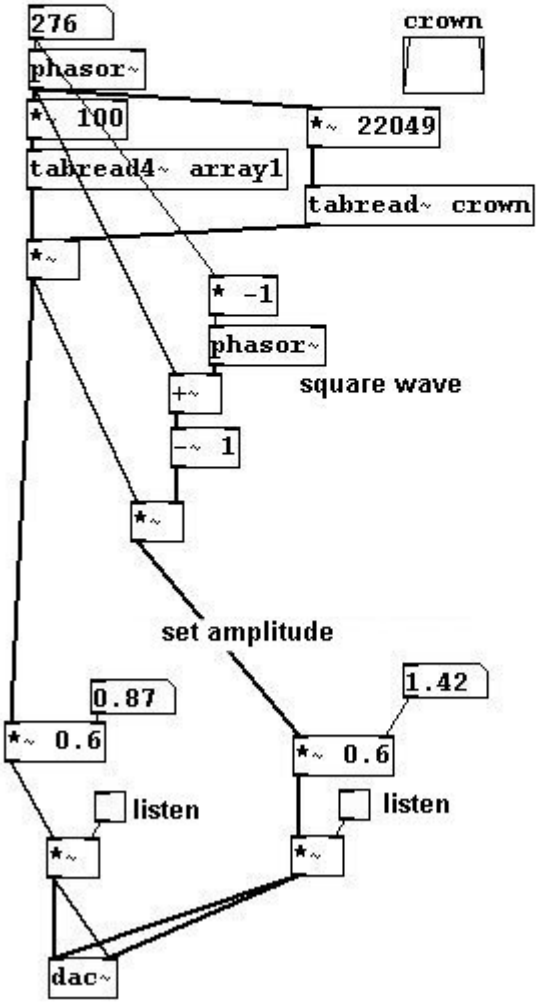




De esta manera la membrana se encuentra en 0 al principio y al final de cada periodo. El resultado podría ser mucho más suave si fuera "enventanado" con una ventana Hanning (3.9.4.1).

Además, también son posibles las funciones de transferencia que utilicen formas de onda conocidas -por ejemplo, una onda cuadrada, la cual intensificará los parciales impares.



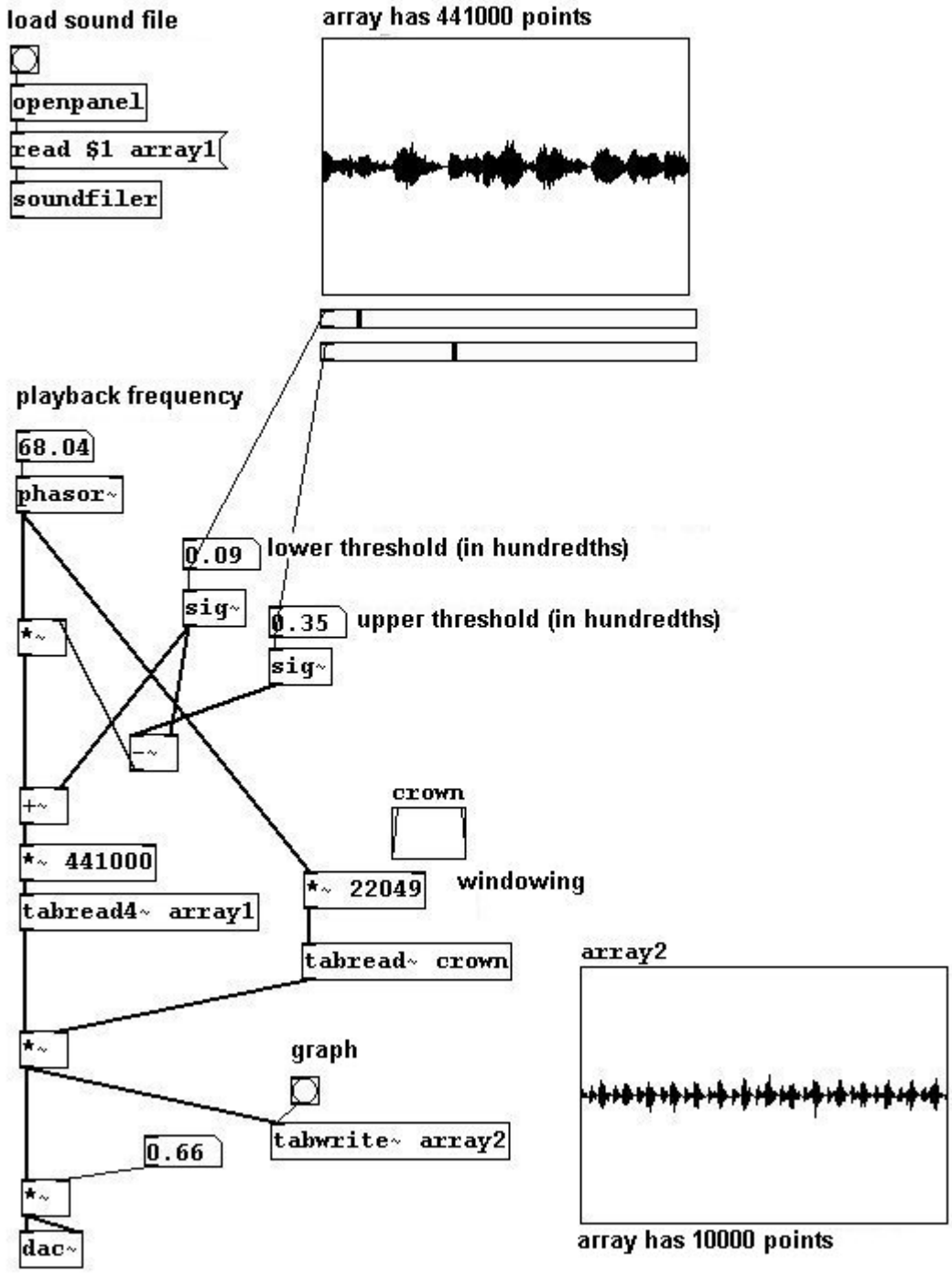


Y así sucesivamente. En estos últimos ejemplos, la onda tiene que ser creada a nivel de control; al contrario de los primeros ejemplos que usaron funciones de transferencia, no podemos cambiarlos en "vivo".

3.5.1.4 Robado de onda (wave stealing)

Una técnica final que puede ser considerada como síntesis de modelado de onda es el "rodado de onda". Este implica tomar pequeñas secciones de piezas de música conocidas...

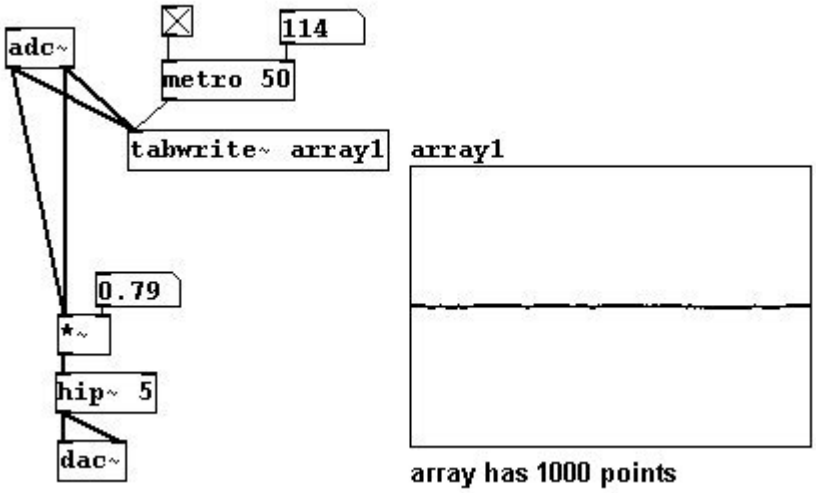
patches/3-5-1-4-wavestealing.pd



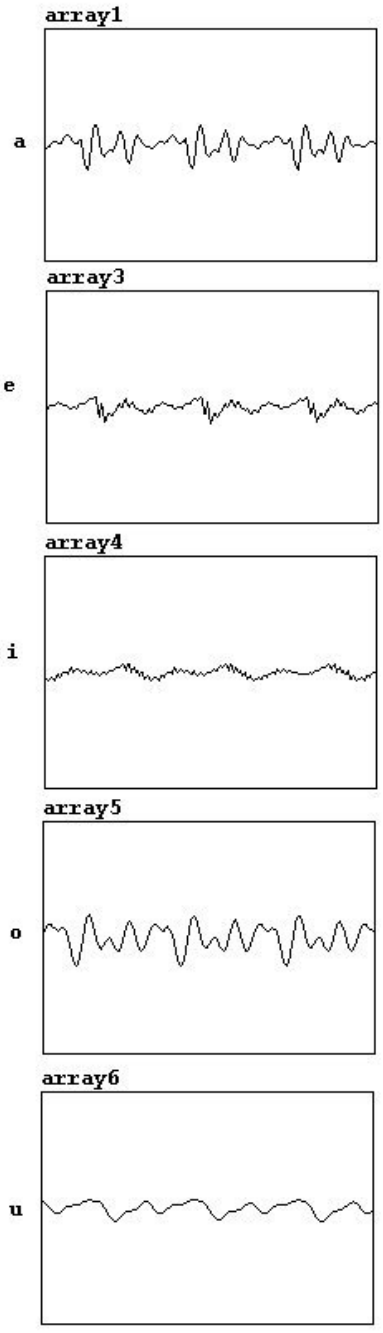
3.5.2 Aplicaciones

3.5.2.1 Canto de formas de onda

Con el siguiente patch, es posible grabar formas de onda con un micrófono para cantar formas de onda.

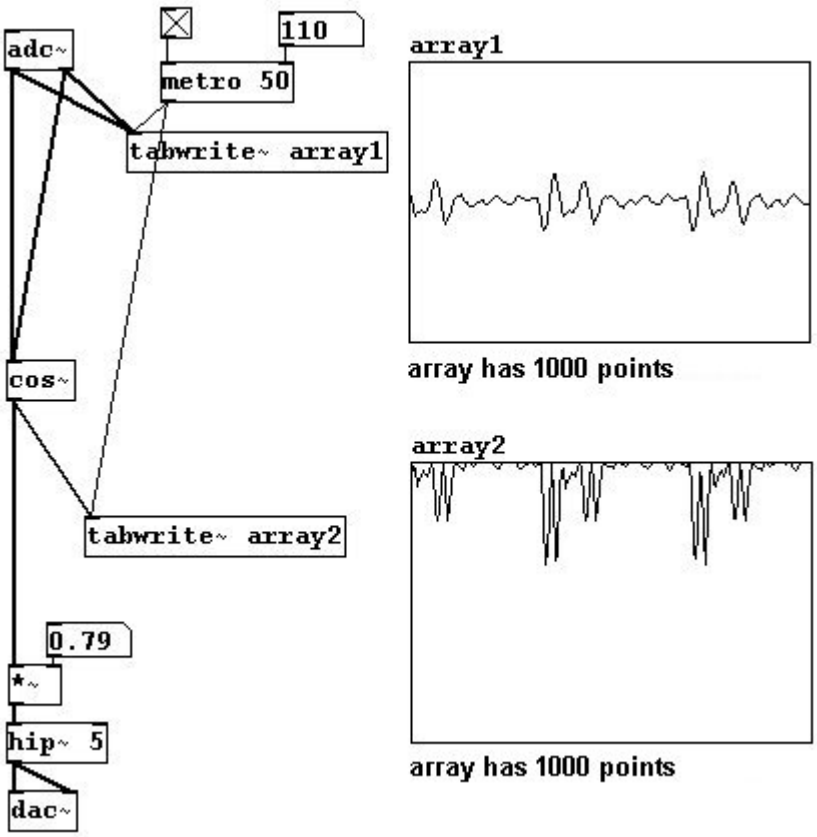


Las vocales (pronunciación en Alemán) se asemejan a esto:



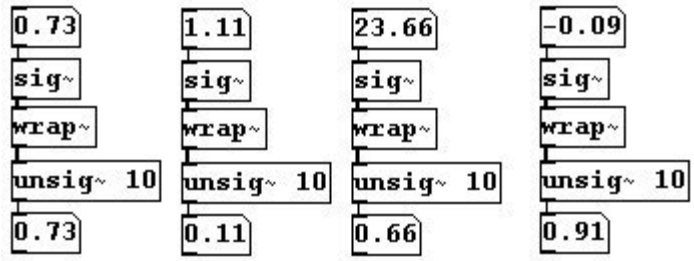
3.5.2.2 Transferencias

Y esta entrada de señal podría, naturalmente, también ser enviada a una función de transferencia:



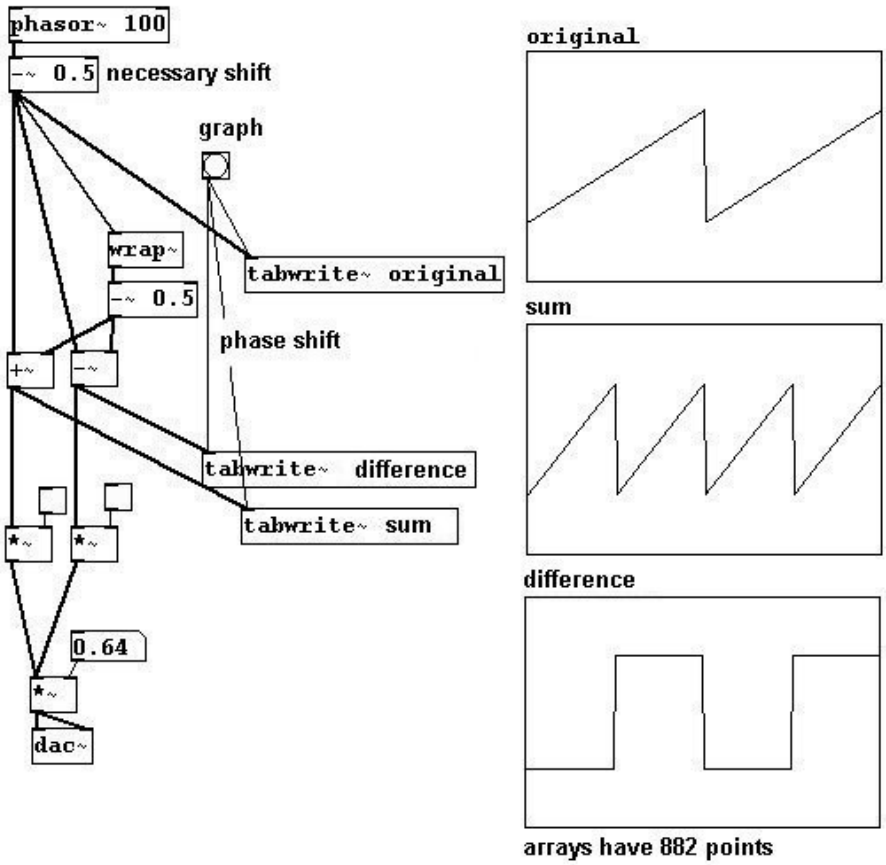
3.5.2.3 Parciales pares / impares

También puede dividir una onda diente de sierra en parciales pares e impares. Para lograr esto, necesita usar "wrap~". Este calcula la diferencia entre el número de entrada y el entero más cercano debajo de ete (es tomado el valor absoluto, para que el resultado siempre sea positivo). Algunos ejemplos:



Y ahora para la división de la onda de diente de sierra: usamos el Objeto "wrap~" para desplazar la fase del mismo; luego es sumado a y restado desde la señal original, resultado en una onda de diente de sierra con el doble de frecuencia y una onda cuadrada.

patches/3-5-2-3-even-odd-partials.pd



3.5.2.4 Más ejercicios

a) Crear una onda que cambia constantemente.

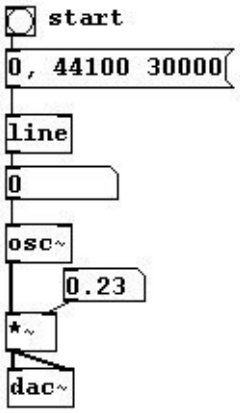
b) Crear un patch en el cual la interpolación y los puntos de la matriz usados para formas de onda aleatorias (controladas) sean variables en número.

3.5.3 Apéndice

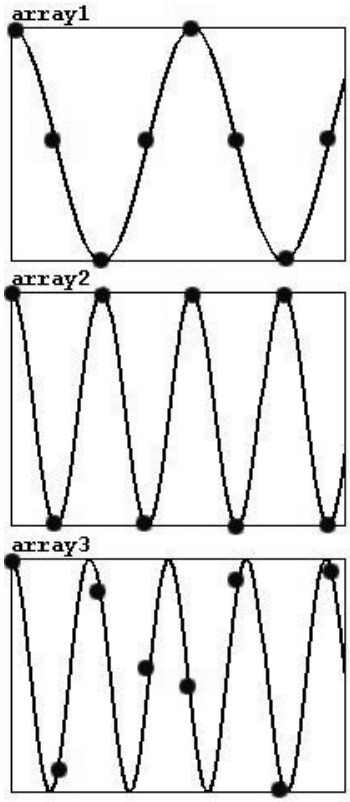
3.5.3.1 Foldover

En este punto debemos atender un problema espinoso en el campo del procesamiento de sonido digital: el foldover. Examinemos la siguiente situación:

patches/3-5-3-1-foldover1.pd



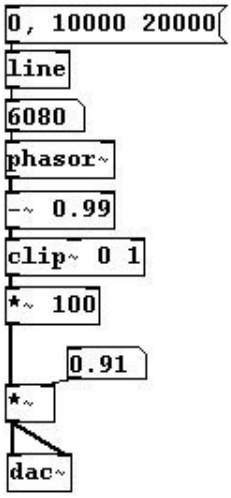
¿Qué ocurre? Luego de los 22050 Hz, cambia la dirección hasta que se alcanza los 44100 Hz, punto en el cual obtenemos una altura con una frecuencia de 0 Hz (luego de esto la altura volvería a subir). La razón de esto es que una frecuencia de muestreo de 44100 Hz puede producir una onda de hasta 22050 Hz como máximo (cf. 3.1.1.3.1). Por otra parte, existen algunos errores de lectura típicos. Veamos tres ondas con diferentes frecuancias: la superior tiene una frecuencia de 11025 Hz, la del medio 22050, y la inferior un poco más de 22050. Las marcas sirven como puntos de referencia (para las muestras), las cuales tienen una velocidad constante de 44100 por segundo.



Cada periodo en una onda de 11025 Hz puede ser representada con cuatro puntos (por supuesto que se pierde la forma característica de la onda sinusoidal). 22050 Hz es la frecuencia más alta que puede ser representada, dado que el teorema de Nyquist requiere al menos dos puntos por periodo. Los errores ocurrirán con frecuencias más altas que esta; no todos los periodos seran capturados y los puntos de medición realmente registran una frecuencia más baja en lugar de una más alta.

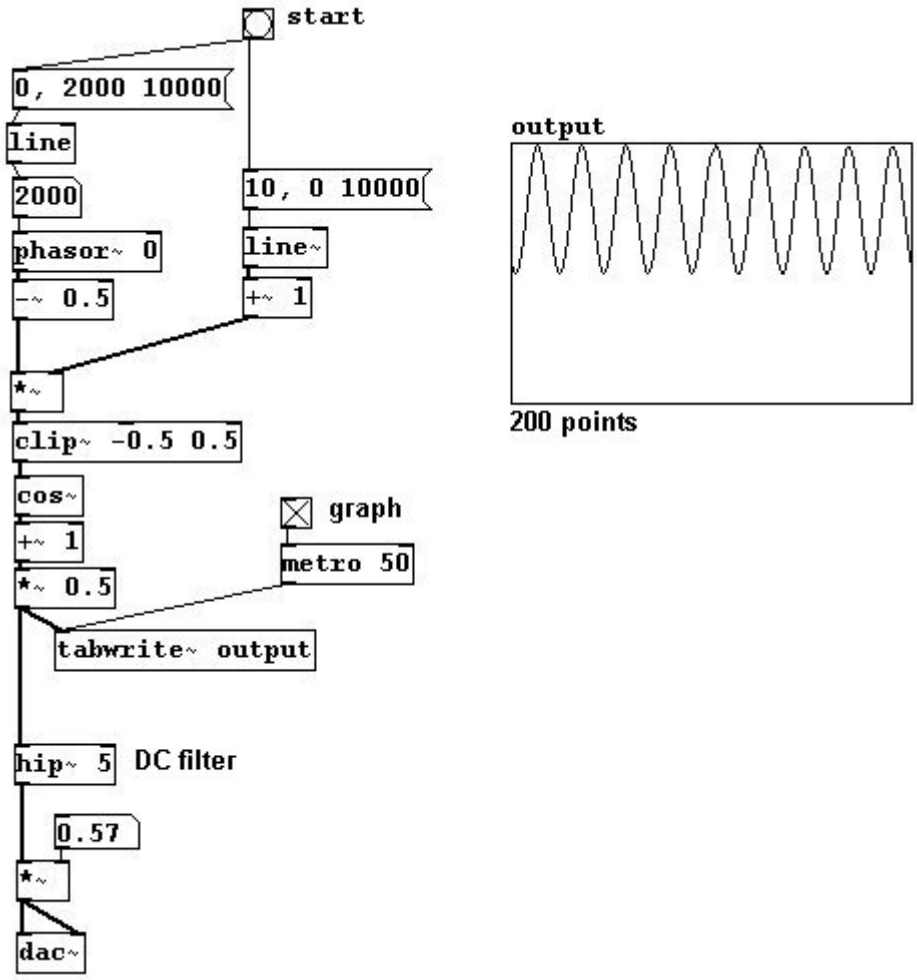
El problema es más pronunciado para formas de onda que exhiben hipertonos, por ejemplo, conun impulso:

patches/3-5-3-1-foldover2.pd



Aquí podemos notar el efecto más temprano: luego de los 700 Hz algunos hipertonos son capturados de forma inadecuada. Esto es porque la forma de onda de impulso prácticamente consiste en una sóla linea, la cual es rápidamente 'perdida'. Una solución a este problema es comenzar con un impulso y ampliar la onda a medida que la frecuencia se incrementa, de manera que termine con una onda sinusoidal al final:

patches/3-5-3-1-foldover3.pd



3.5.4 Para los interesados, especialmente

3.5.4.1 GENDY

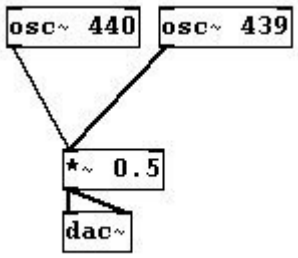
Iannis Xenakis en sus últimos anños desarrolló un proceso de generación de ondas llamado GENDY. Él 'compuso' sólo formas de ondas y derivados de las mismas, por ejemplo, en su pieza "Gendy 3".

3.6 Síntesis por modulación

3.6.1 Teoría

3.6.1.1 Modulación en anillo

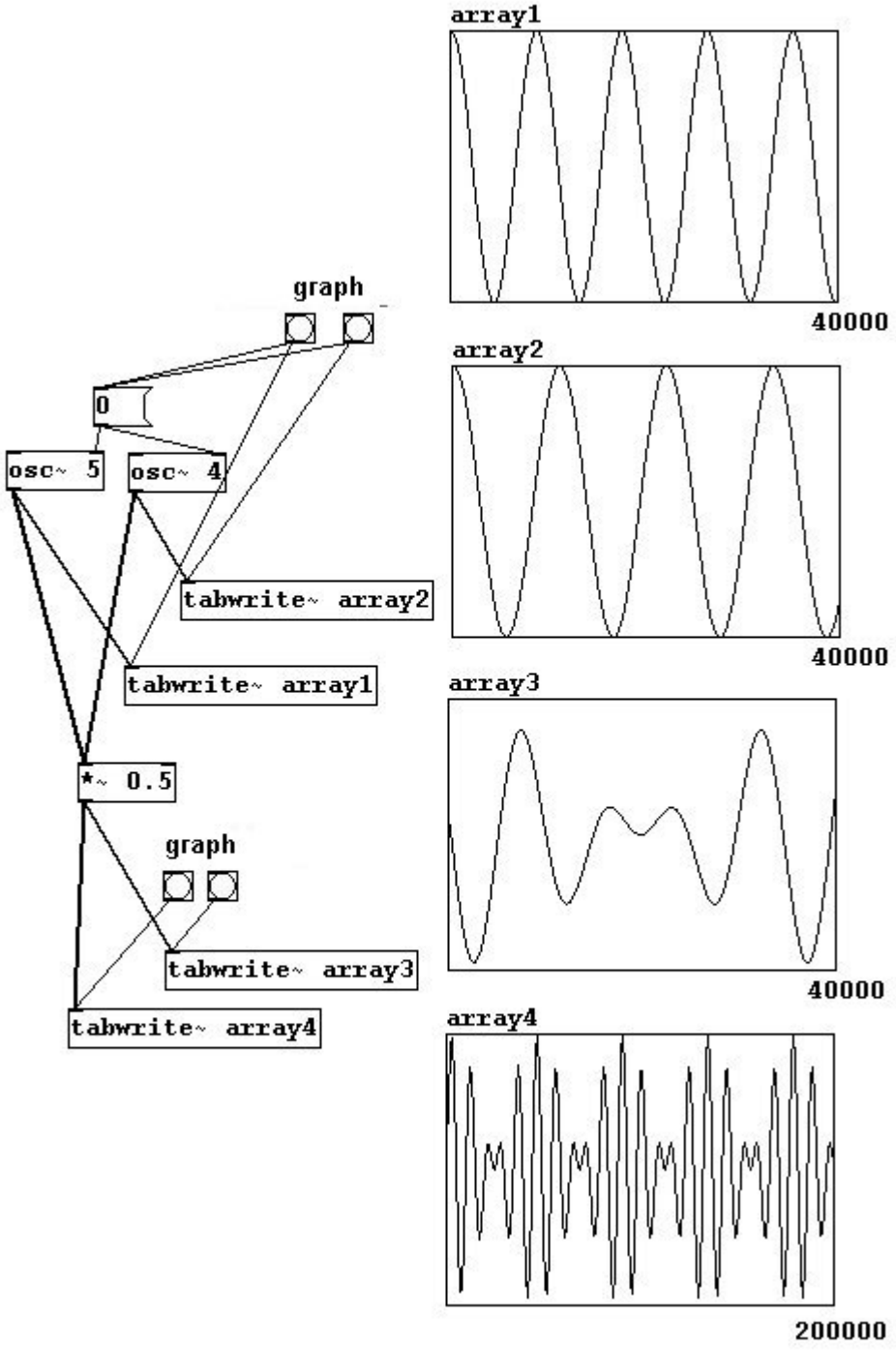
Primero observemos el siguiente fenómeno:



Si escuchamos dos tonos sinusoidales muy cercanos en frecuencia el uno del otro, escuchará cancelaciones de onda fluctuantes. Esto se debe a la interrelación de dos ondas casi (¡pero sólo casi!) idénticas en frecuencia. Se llama a este fenómeno *batido*. La velocidad, o ritmo, del batido es exactamente igual a la diferencia entre las dos frecuencias -en el ejemplo anterior, 440 – 439 = 1 Hz.

Veamos un ejemplo simplificado utilizando osciladores con 4 y 5 Hz::

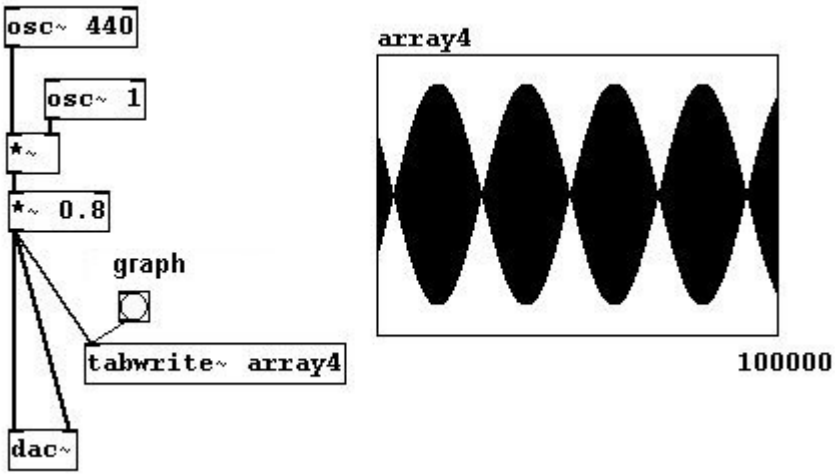
patches/3-6-1-1-ringmodulation1.pd



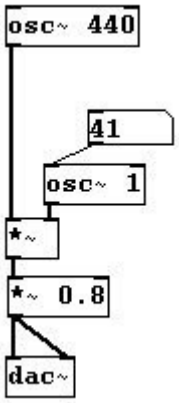
Las ondas alternan entre sumatorias (nunca lo olvide: ¡las ondas se suman entre ellas!) y cancelaciones. En array1 y array2 podemos ver una parte de las dos sinusoides originales, en array3 la onda resultante de la suma, y en array4 la misma onda resultante pero en un periodo más amplio de tiempo. El resultado es una pulsación que aumenta y disminuye en volumen (para que el oido humano perciva dos tonos diferentes, sus frecuencias deben diferir en aproximadamente 5 cents).

También puede determinar, de manera precisa, el ritmo de estas fluctuaciones de amplitud. Como pudo observar en array4 del diagrama anterior, la amplitud posee una forma sinusoidal. Por lo tanto puede simplemente utilizar un oscilador para determinar dicha amplitud:

patches/3-6-1-1-ringmodulation2.pd



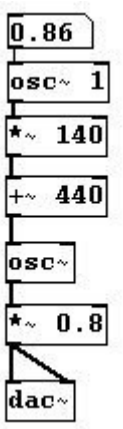
(La razón por la cualesta matriz es tan oscura en comparación con la anterior se debe a que esta utiliza frecuencias mucho más altas.) La onda resultante corresponde a la sumatoria de dos ondas. Si aumenta progresivamente la frecuencia de la *modulación de amplitud*...



... escuchará dos frecuencias que se alejan una de otra simétricamente -una hacia los agudos, la otra hacia los graves- por un intervalo igual a la distancia de la amplitud del eje medio, es decir, la amplitud de frecuencia inicial. Este proceso es un ejemplo de *modulación de amplitud (síntesis AM)*, llamada 'modulación en anillo' por su naturaleza simétrica. Si la frecuencia inicial es 440 Hz y la frecuencia de amplitud es 100 Hz, escuchará dos tonos: uno a 340 Hz y el otro a 540 Hz.

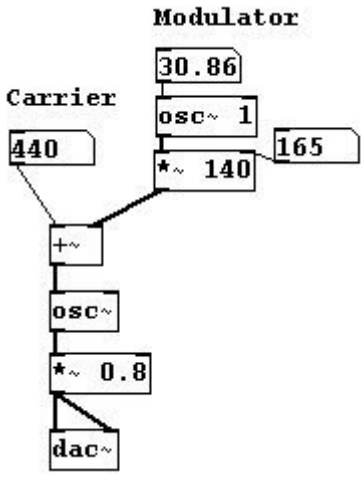
3.6.1.2 Modulación de frecuencia

También puede usar un oscilador para modular la frecuencia de una onda sinusoidal. Llamamos a esto *modulación de frecuencia*:

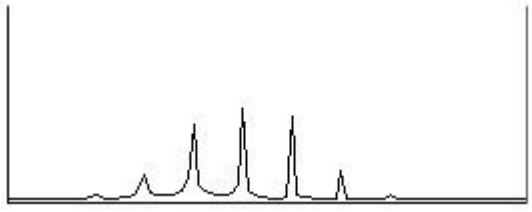


Un opscilador es la "portadora" (carrier) y el otro la "moduladora" (modulator). Si utilizamos una frecuencia baja en la "moduladora", tendremos como resultado un vibrato. Si aumentamos la dicha frecuencia a partir de los 20 Hz, obtendremos progresivamente un resultado multifónico cada vez más complejo:

patches/3-6-1-2-frequencymodulation.pd



La onda resultante consiste en una sumatoria de muchas ondas sinusoidales diferentes; la *frecuencia portadora* se encuentra en el medio de este complejo, mientras que los otros tonos se distribuyen a ambos lados de esta, a un intervalo de distancia determinado por la *frecuencia moduladora*.



Cuando aumentamos la amplitud de modulación, las amplitudes de la frecuencias adicionales tambien aumentan. Sin embargo, este incremento es difícil de formularlo mateméticamente.

Llegamos a una situación especial cuando la *frecuencia moduladora* corresponde a un número entero múltiplo de la *frecuencia portadora* (es decir, 1x, 2x, 3x, 4x, 5x, 6x, etc.). Los tonos adicionales superiores a la frecuencia portadora también serán números enteros múltiplos de la frecuencia portadora -es decir, sus hipertonos.

Además, las frecuencias negativas son reflejadas en el rango positivo del eje de las frecuencias. En la situación especial mencionada en el párrafo previo, estas son cubiertas por frecuencias

"normales". Digamos que tenemos una frecuencia portadora de 200 Hz y una frecuencia moduladora de 100 Hz; el "recubrimiento" comienza a partir de tercer tono con dirección negativa (el cual también corresponde, una vez reflejado, a los 100 Hz, y los siguientes a 200, 300, etc.), resultando entonces amplificaciones y supresiones de acuerdo a la longitud de fase.

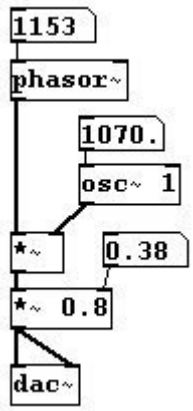
La ventaja de la *modulación de frecuencia (síntesis FM)* sobre la síntesis aditiva (la adición simple de ondas sinusoidales) consiste en que sólo necesita dos osciladores para obtener sonidos complejos y ricos en espectro (¡sólo debe cambiar la frecuencia y en especial la amplitud de la moduladora!). Un sonido de síntesis FM característico posee un 'espectro inarmónico', es decir un espectro cuyos hipertonos no son múltiplos de números enteros de la fundamental. Algunos instrumentos metálicos, como campanas o gongs, exhiben un espectro similar; por esta misma razón, los sonidos construidos mediante síntesis FM son frecuentemente asociados con timbres 'metálicos'.

3.6.2 Aplicaciones

3.6.2.1 Modulación en anillo con espectro más rico

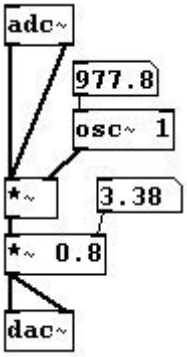
Las modulaciones en anillo que utilizan sonidos con mayor cantidad de hipertonos son naturalmente más complejos y ricos en espectro:

patches/3-6-2-1-ringmodulation3.pd



3.6.2.2 Modulación en anillo en vivo

patches/3-6-2-2-ringmodulation-live.pd

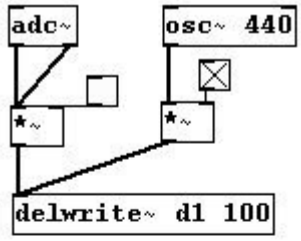


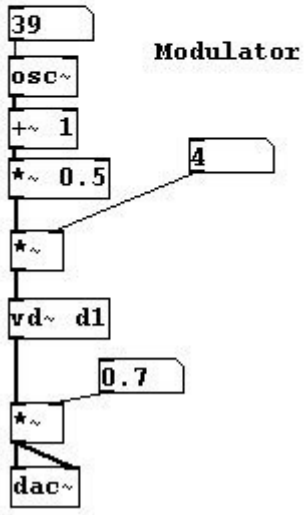
3.6.2.3 Modulación de frecuencia en vivo

Para usar la modulación de frecuencia en contextos 'en vivo', debe utilizar un *retardo variable* para

poder cambiar la frecuencia:

patches/3-6-2-3-frequencymodulation-live.pd





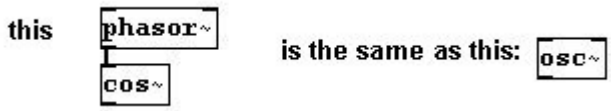
3.6.2.4 Más ejercicios

Combine todo lo que aprendió hasta ahora.

3.6.3 Apéndice

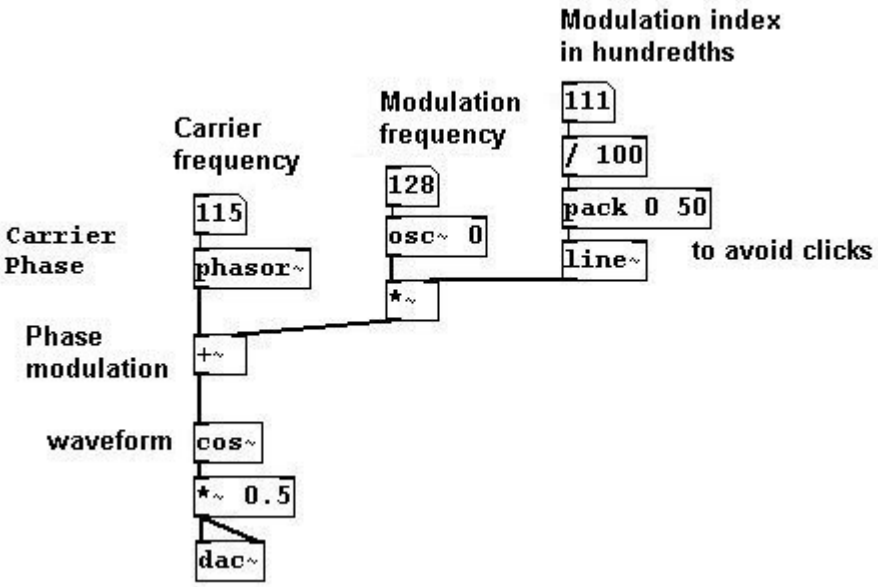
3.6.3.1 Modulación de fase

La modulación de frecuencia también es llamada modulación de fase y también puede ser programada de esta manera. Para lograr esto, la portadora debe estar dividida en *procesamiento de fase* y *procesamiento de forma de onda*. Así es cómo trabaja:



Y la modulación de fase se ve de la siguiente forma:

patches/3-6-3-1-phasemodulation.pd



3.7 Síntesis granular

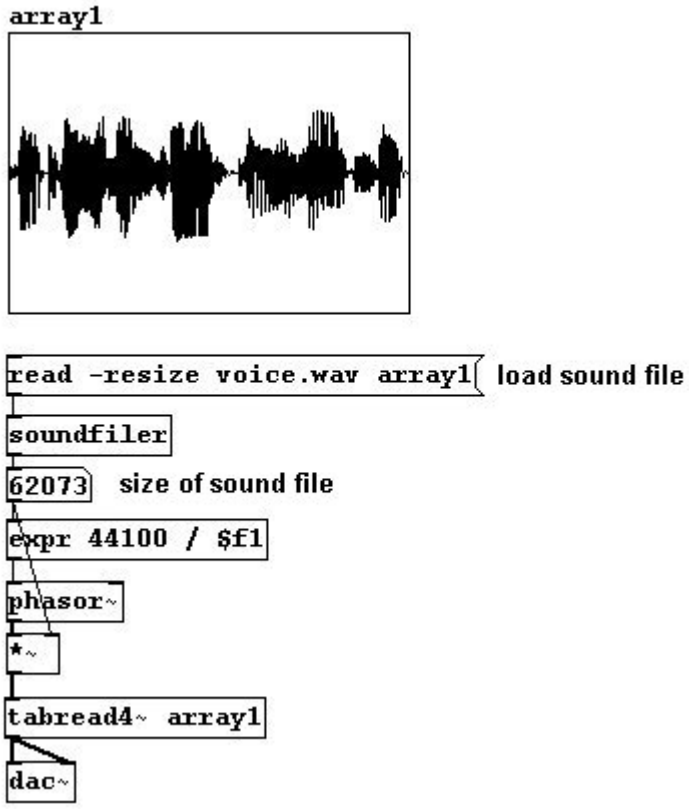
3.7.1 Teoría

3.7.1.1 Teoría de la síntesis granular

En el Capítulo 3.4 (muestra) aprendimos cómo cambiar la velocidad de un sonido existente dentro de una matriz, pero a la vez se afecto por un cambio en la altura. Una manera de disociar estos parámetros es utilizando la *síntesis granular*. La idea de esta reside en que el sonido es muestreado a la velocidad original, pero es reproducido a una velocidad diferente para cada punto de muestra.

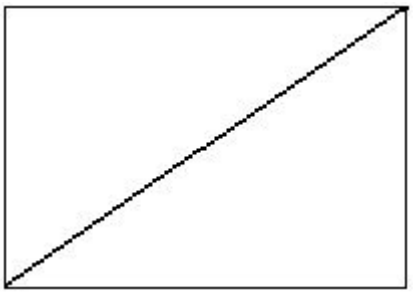
Tenemos un "indicador" que se mueve a lo largo de la matriz a velocidad normal:

patches/3-7-1-1-granular-theory1.pd

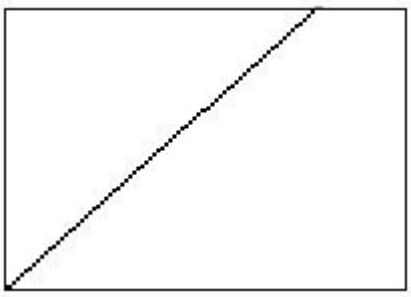


Sólo en ciertos intervalos obtenemos indormación sobre la posición actual del indicador; cuando esta información es recibida, la matriz es ejecutada desde ese punto, aunque a una velocidad diferente.

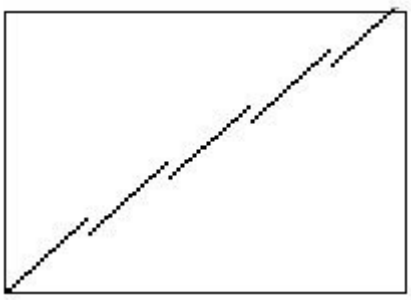
Para una mejor comprensión, digamos que este gráfico corresponde a la velocidad normal:



...y este pertenece a una velocidad 'muy rápida':

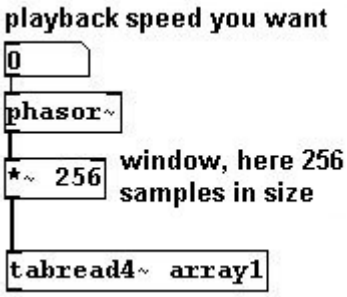


...entonces la síntesis granular corresponde al siguiente gráfico:



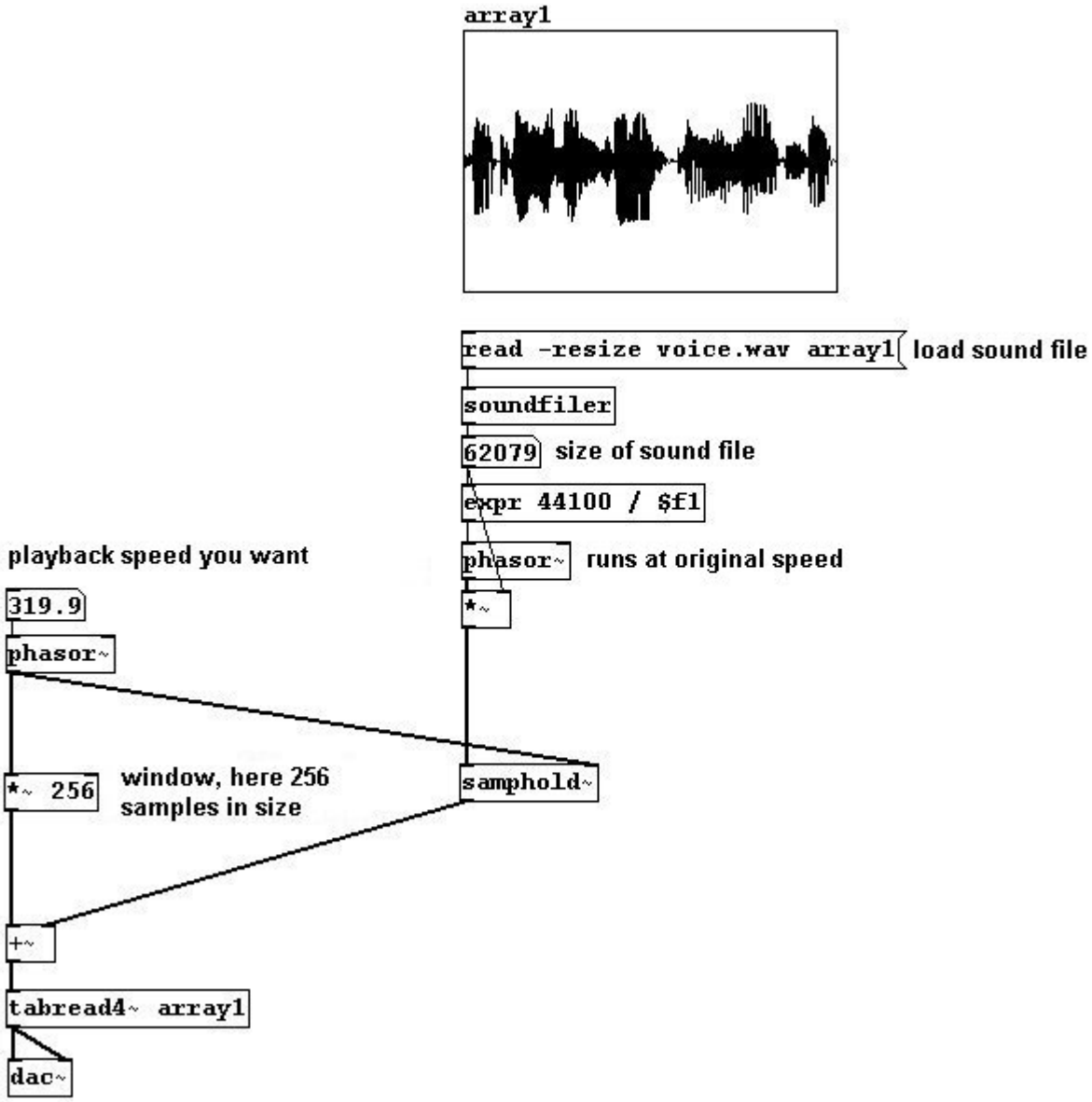
Aunque la reproducción sea 'muy rápida' ('o muy lenta'), siempre comienza en un punto que corresponde a la velocidad inicial. Estos fragmentos individuales son llamados "granos"; el tamaño de los mismos es llamado "tamaño de grano" o "tamaño de ventana". Estos "granos" son tan pequeños y usados en tan grandes cantidades que no pueden ser escuchados individualmente, sino como una entidad contínua. Esta es la magia detrás de la síntesis granular

Cada "grano" individual es reproducido de la siguiente manera:



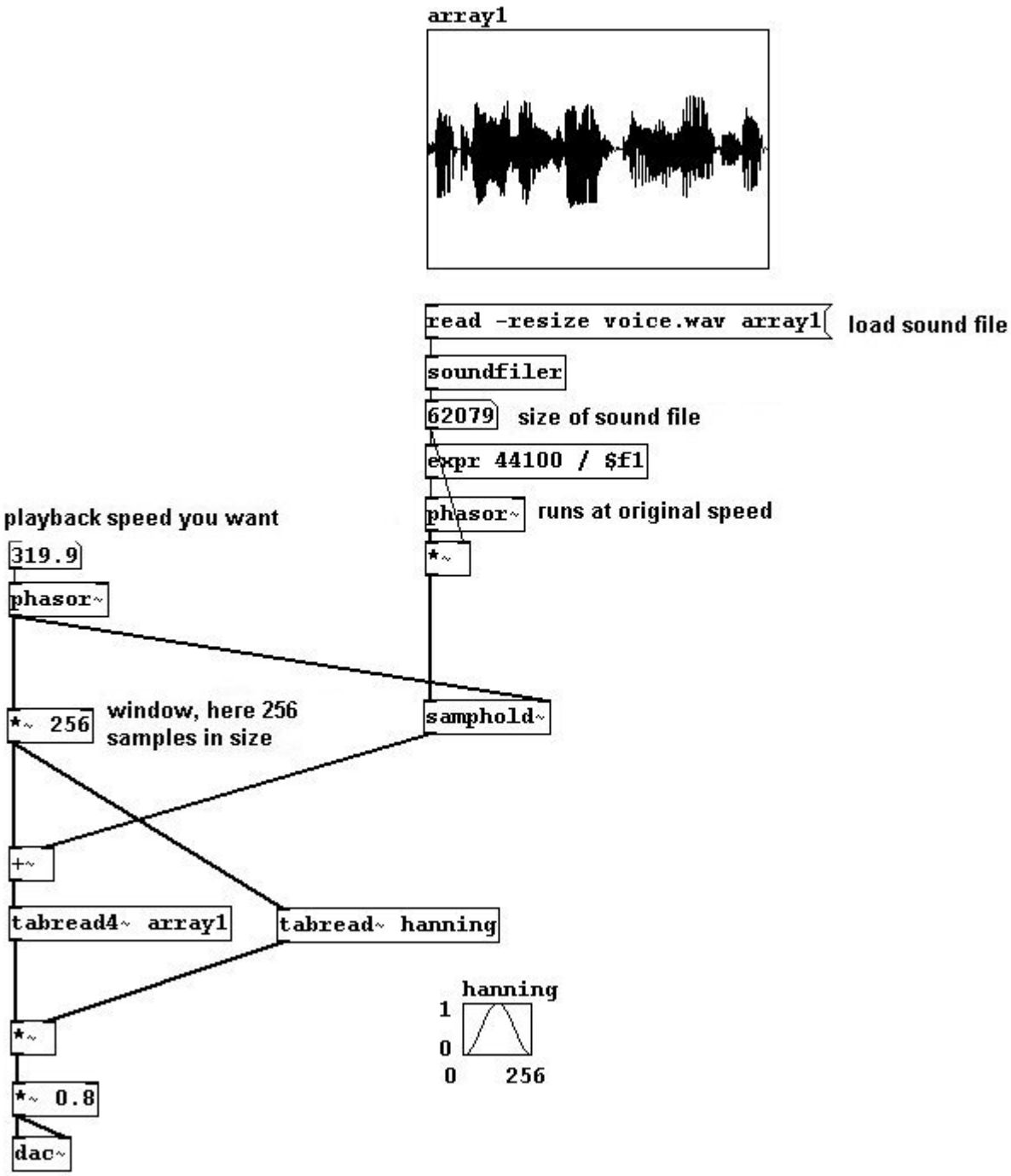
Luego de que un grano es ejecutado, se produce un salto a la siguiente posición; esta posición es tomada desde la posición es tomada de la posición actual del "indicador principal". Existe un Objeto especial para realizar esto: "samphold~". Cuando tenemos un paso descendente en la entrada derecha, "samphold~" envía inmediatamente la muestra actual en la entrada izquierda y repite esto hasta que el valor de la entrada derecha es menor que el valor anterior. Este extraño proceso tiene sentido si en la entrada derecha se encuentra un "phasor~". Recibe sólo una vez un paso descendente, justo al final de un periodo. Un grano puede leerse de esta manera y el desplazamiento puede agregarse al final del mismo:

patches/3-7-1-1-granular-theory2.pd



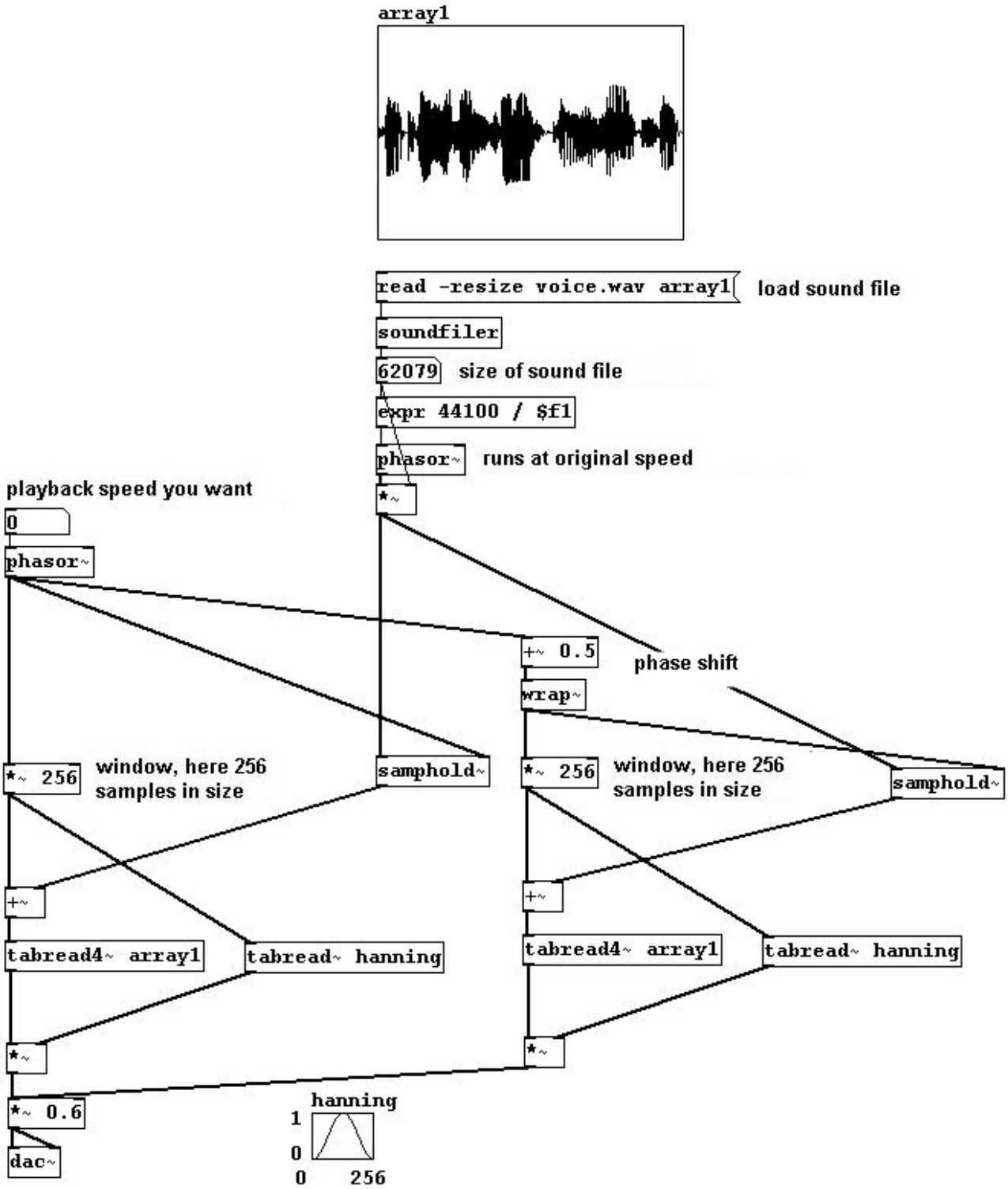
De esta manera, la resultante suena más aguda, pero con la misma duración que el original. Si observamos atentamente, es claro que esto puede llevarnos a complicaciones. Si la reproducción de una muestra a la otra es más rápida que la velocidad del indicador (la cual corre a la velocidad original), entonces se volverá a tomar dicha muestra (repitiendola) la próxima vez que "samphold~" sea disparado. A la inversa, si los "grains" son reproducidos más lentamente que el movimiento del indicador, entonces se omitirás algunas partes. Pero mientras que las velocidades del original y de la reproducción no difieran tan dramáticamente, esto no se notará (demasiado). Para rectificar esto, se pueden llevar a cabo algunas mejoras. Primero, utilice una ventana Hanning para suprimir los clicks que resultan con cada salto a un nuevo valor:

patches/3-7-1-1-granular-theory3.pd



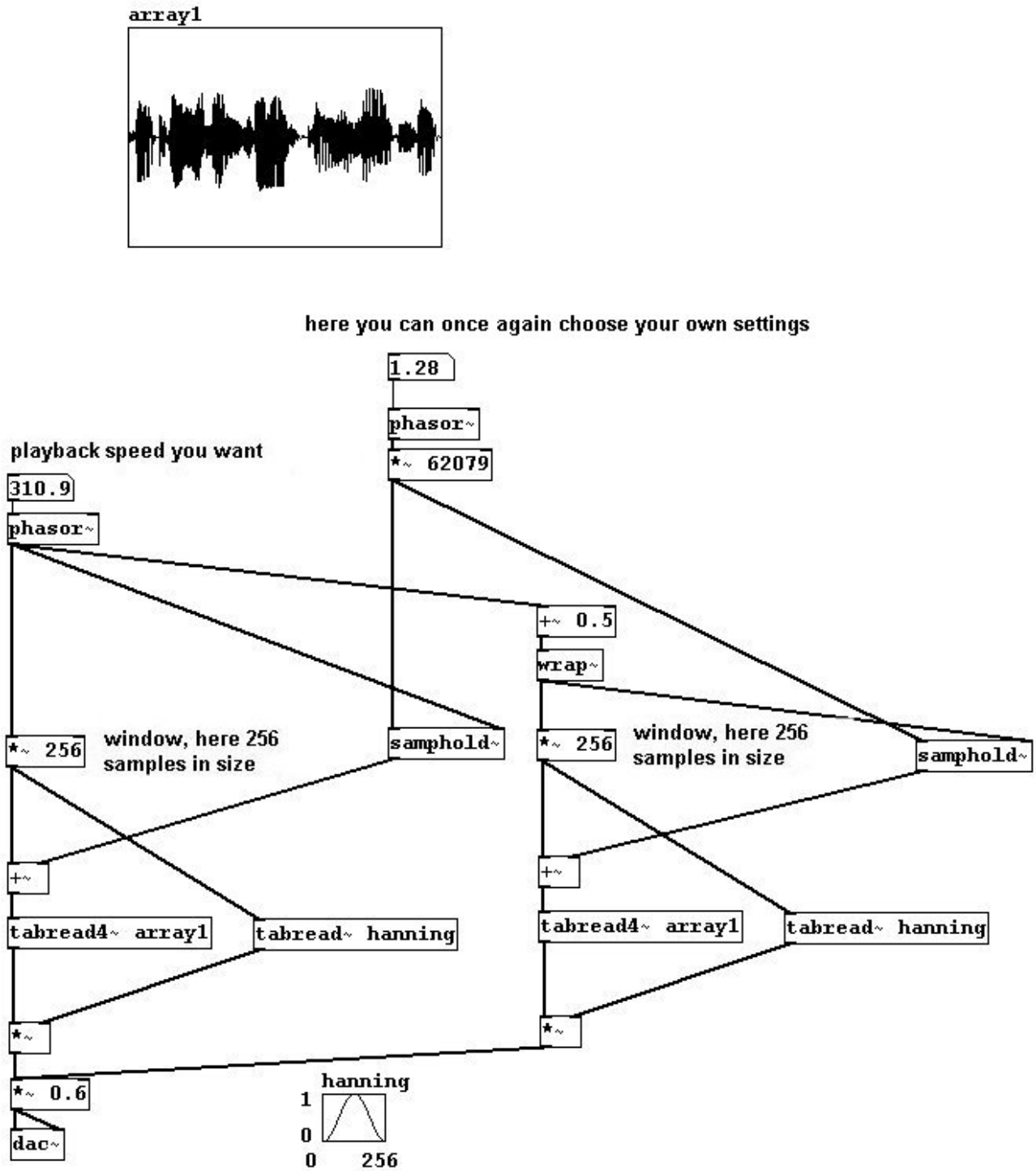
Los huecos resultante pueden ser llenados utilizando un segundo lector de granos, desplazado medio periodo:

patches/3-7-1-1-granular-theory4.pd



Lo bueno de la síntesis granular es que, además de la posibilidad de cambiar la altura sin afectar la velocidad, también puede cambiar la velocidad sin afectar la altura:

patches/3-7-1-1-granular-theory5.pd

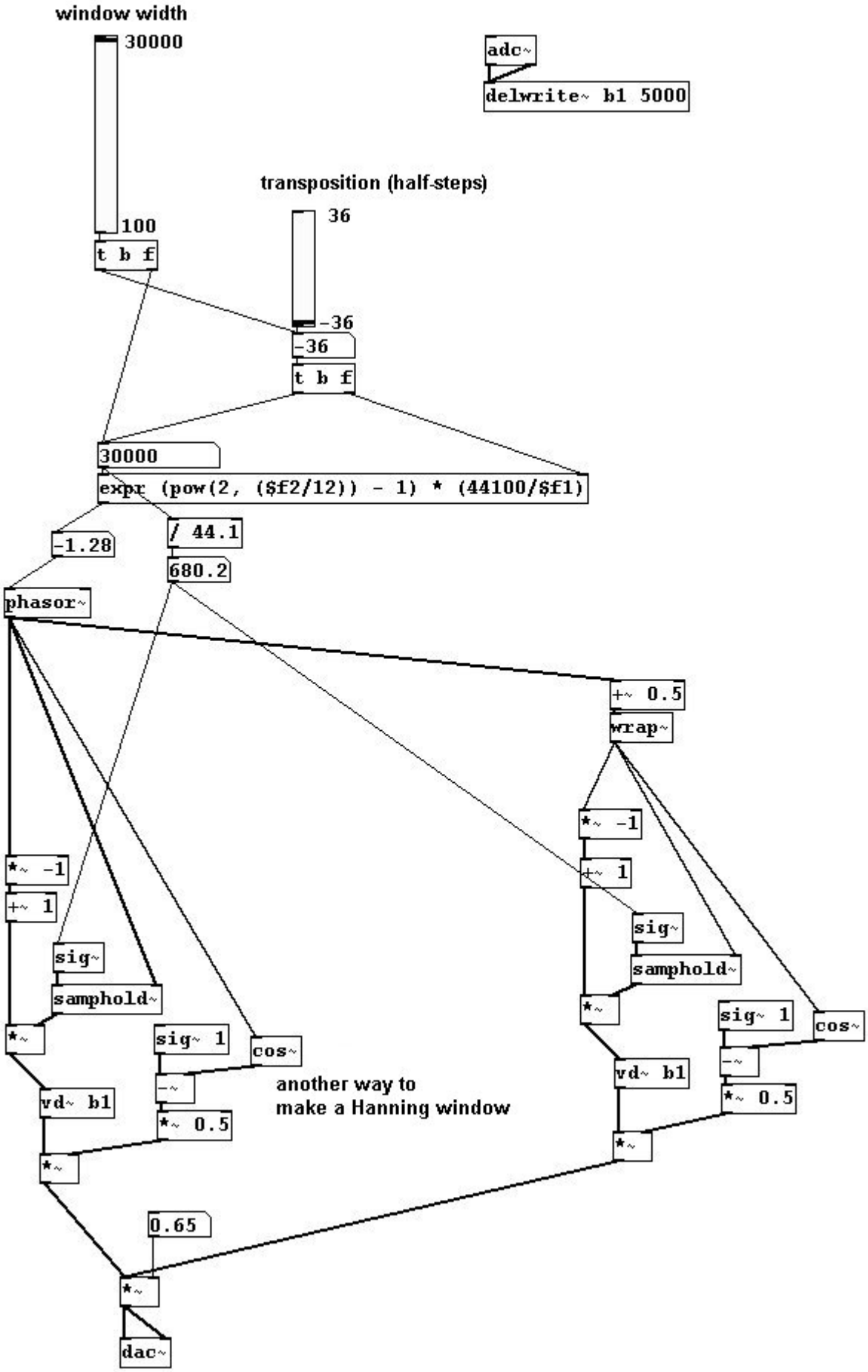


3.7.2 Aplicaciones

3.7.2.1 Síntesis granular en vivo

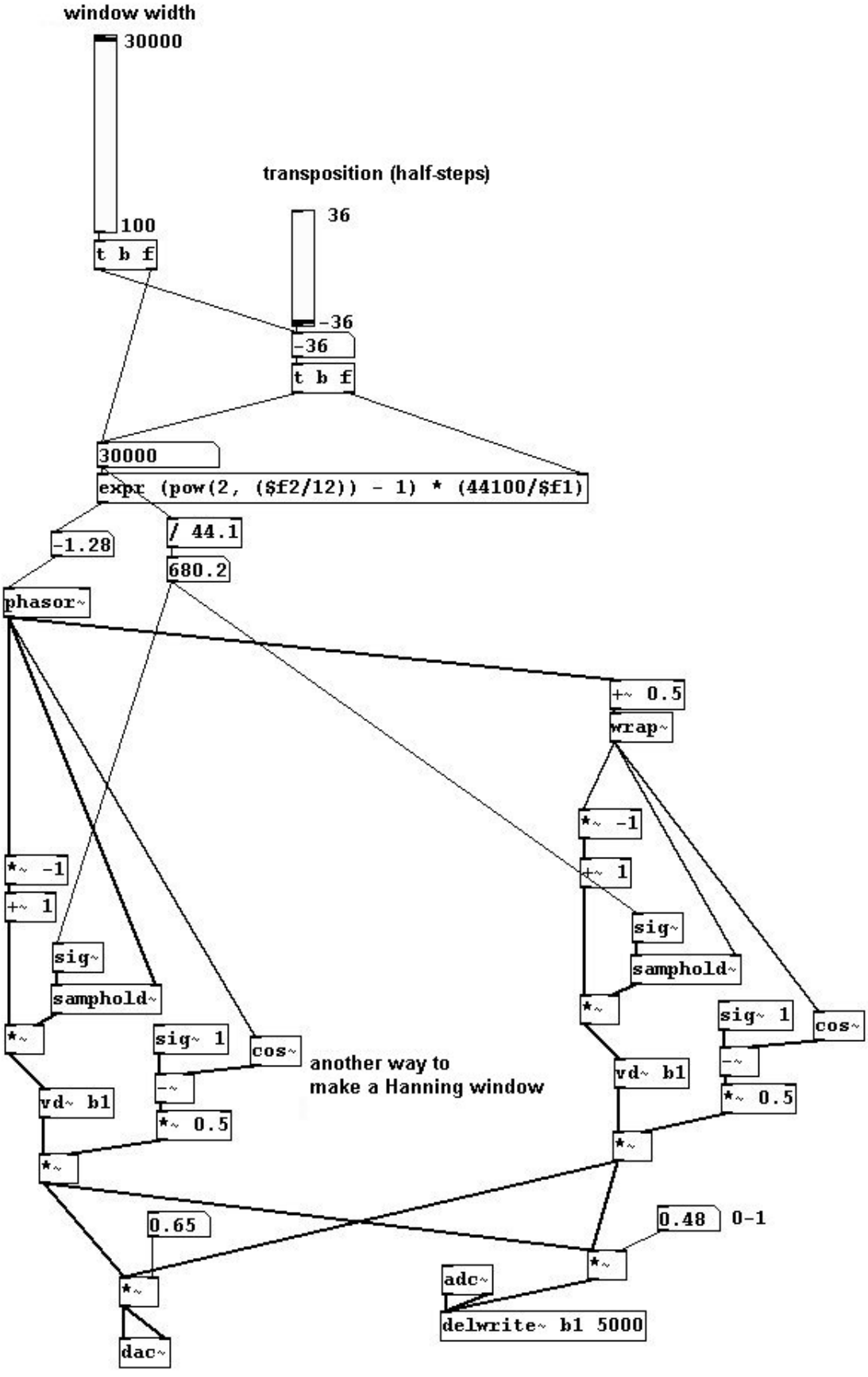
Para usarlo en una performance en vivo, nuevamente necesitará incluir *retardos variables*:

patches/3-7-2-1-granular-live.pd


window width
30000
100
t b f
adc~
delwrite~ b1 5000
transposition (half-steps)
36
-36
-36
t b f
30000
expr (pow(2, ($f2/12)) - 1) * (44100/$f1)
-1.28
/ 44.1
680.2
phasor~
+~ 0.5
wrap~
*~ -1
+~ 1
sig~
samphold~
*~
sig~ 1
cos~
-~
*~ 0.5
vd~ b1
*~
0.65
*~
dac~
another way to
make a Hanning window

3.7.2.2 En vivo con retroalimentación

patches/3-7-2-2-granular-live-feedback.pd



3.7.2.3 Más ejercicios

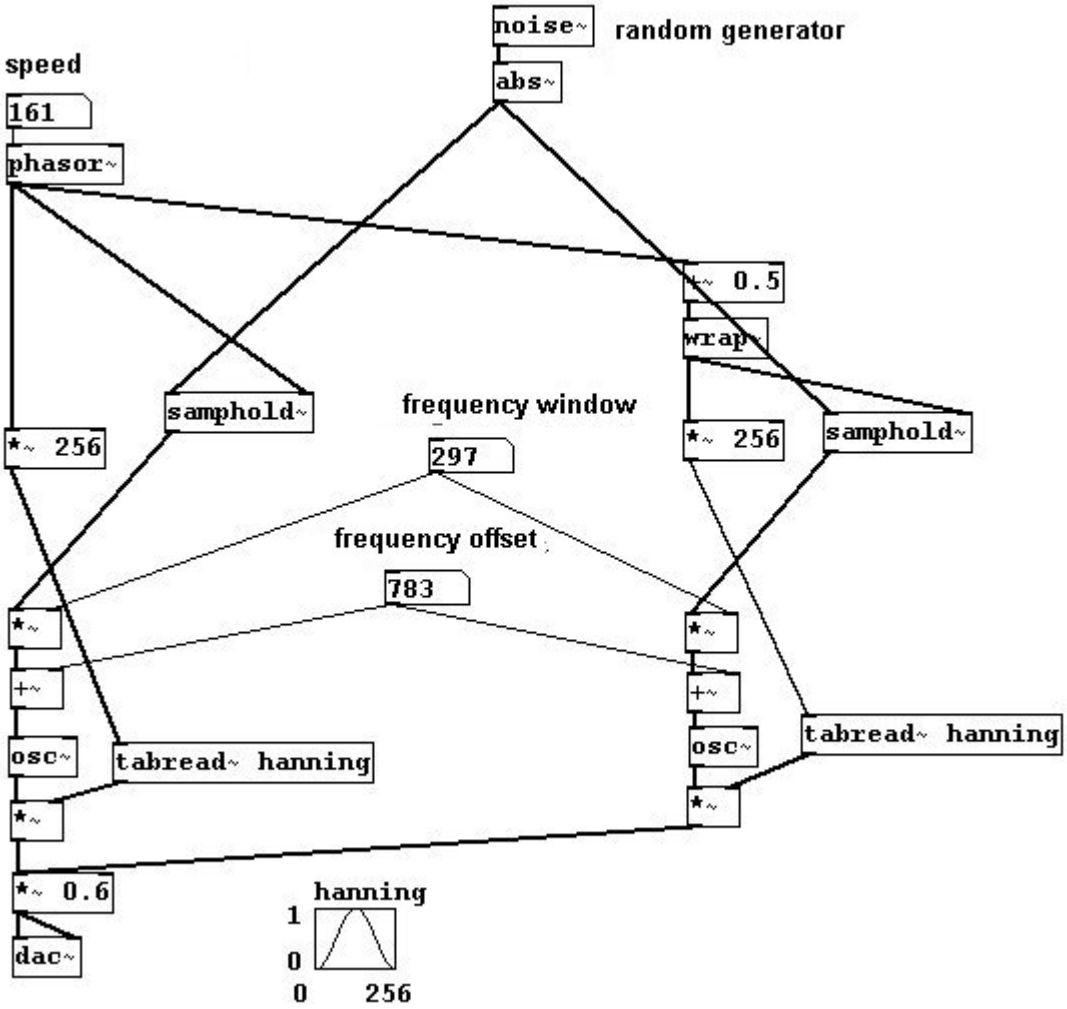
Cree cuatro lectores cada uno con un tamaño de ventana variable. ¡Experimente!

3.7.3 Apéndice

3.7.3.1 Técnica granular como sintetizador

La síntesis granular también puede ser usada como un sintetizador de nubes de alturas, usando, más convenientemenre, un generador aleatorio:

patches/3-7-3-1-granularsynthesizer.pd



3.8 Análisis de Fourier

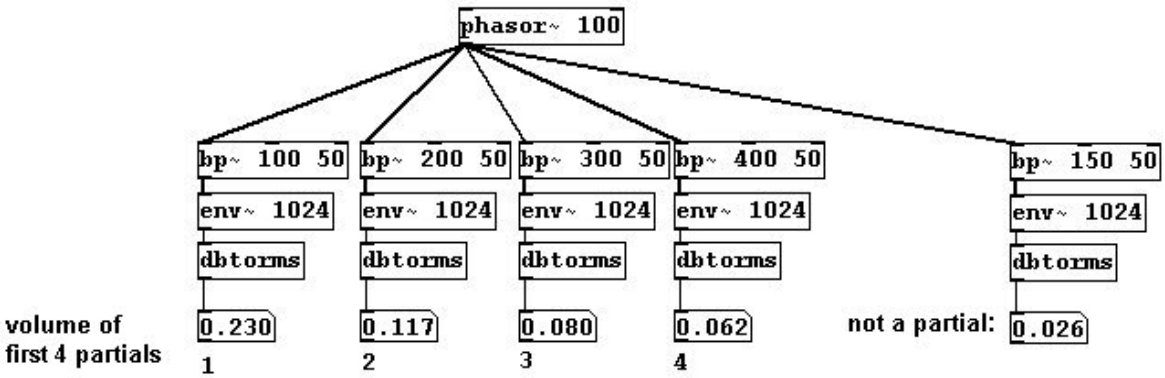
3.8.1 Teoría

3.8.1.1 Analizar parciales

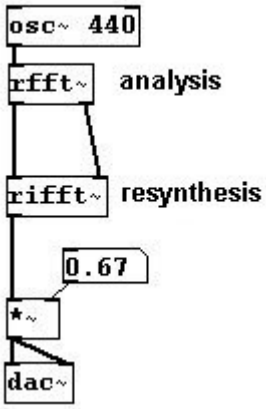
Volvamos a un concepto básico de síntesis aditiva: un sonido comprende parciales. Si quiere saber

que componentes contiene un sonido, puede emplear un conjunto de filtros pasa-banda para cada parcial:

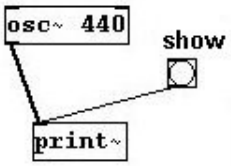
patches/3-8-1-1-analyze-partials.pd



Este proceso realiza lo que se llama *transformada de Fourier*. Divide todo el espectro de frecuencias en partes de igual tamaño y determina la amplitud y fase para cada uno de ellos.Uno podría a su vez reconstruir la señal original a partir de estos valores. La derivada de las partes de componentes se llama análisis, la reconstrucción se llama resíntesis. Puede llevar a cabo ambas usando los Objetos "rfft~" y "irfft~":



el tamaño de las secciones individuales, llamadas *recipientes*, se determina por el tamaño del bloque (por defecto 64 muestras). El uso de "print~" nos muestra los valores en un bloque dado:

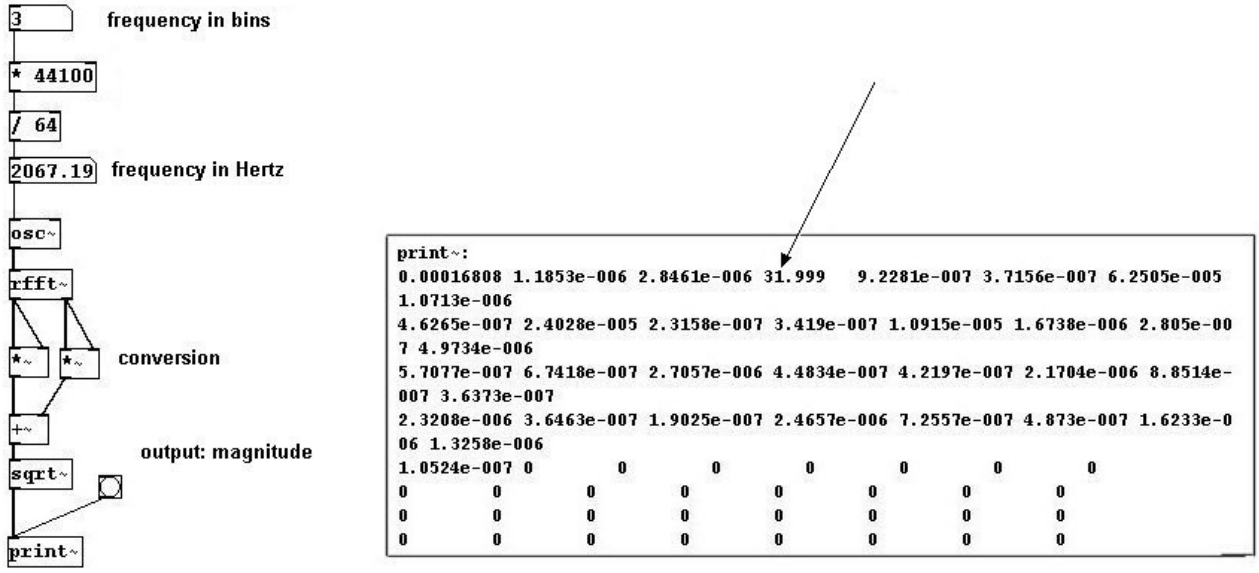



```
print~:
-0.22677 -0.28734 -0.34679 -0.40487 -0.46135 -0.51603 -0.56867 -0.61909
-0.66707 -0.71243 -0.75499 -0.79459 -0.83107 -0.86428 -0.89409 -0.92039
-0.94307 -0.96205 -0.97725 -0.98862 -0.9961  -0.99967 -0.99931 -0.99501
-0.98681 -0.97473 -0.95882 -0.93915 -0.91579 -0.88884 -0.85839 -0.82457
-0.78751 -0.74734 -0.70425 -0.65839 -0.60994 -0.5591  -0.50606 -0.45104
-0.39424 -0.33589 -0.27622 -0.21547 -0.15387 -0.091666 -0.029103 0.033576
0.096122 0.15829  0.21984  0.28052  0.3401   0.39835  0.45502  0.50992
0.56281  0.61349  0.66176  0.70743  0.75031  0.79025  0.82708  0.86067
```

Como ocurre con "snapshot~" o "unsig", podemos observar los valores de amplitud producidos. Con "print~" podemos ver TODOS los valores generados, limitado a un bloque de DSP. Atengámonos primero a las 64 muestra; es decir, todo el espectro -hasta 44100 Hz- es dividido en

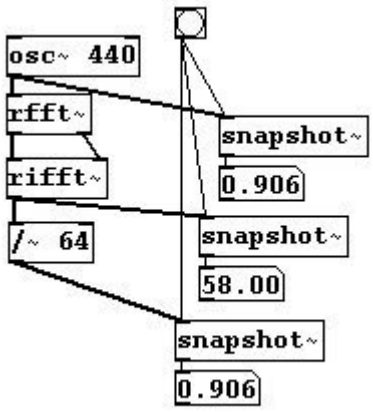
recipientes de un tamaño de 44100/64 = 689 Hz. Lo siguiente que debemos considerar es que, con "FFT", los datos de amplitud y fase no son representados en el formato habitual; aparecen como valores de seno y coseno. Por ahora, no vamos a avanzar en este aspecto con mayor detalle; podemos transformar los datos de una forma más comprensible de la siguiente manera:

patches/3-8-1-1-rfft1.pd



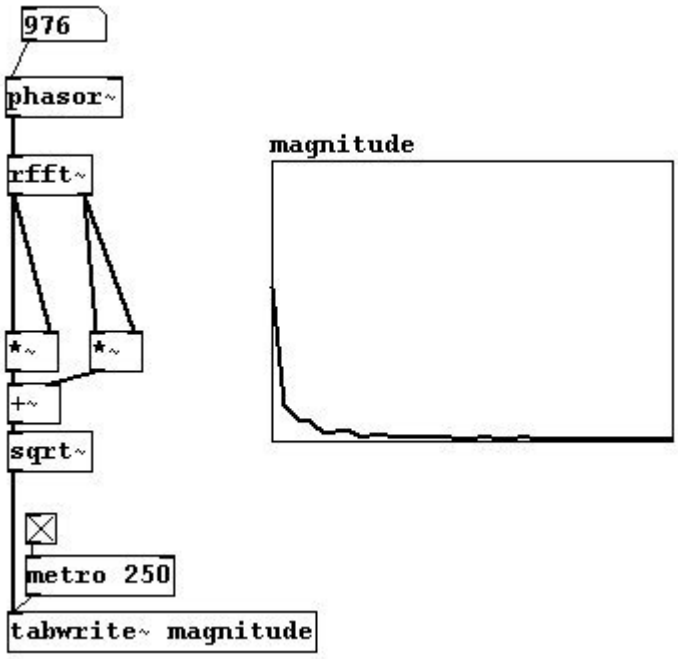
Como podemos ver, "print~" genera 64 valores de amplitud. La amplitud es dada aquí como magnitud, siempre como valor positivo (porque fue cuadrado). Observemos más atentamente: exceptuando el tercer recipiente, el cual tiene un valor de (ca.) 32, no tenemos más que valores pequeños. No disponemos de cálculo para números superiores a la frecuencia de Nyquist.

Generalmente después de un proceso de FFT se realiza un proceso de normalización, debido a que los valores de amplitud resultantes son bastante altos. Primero, este es el tamaño del bloque:



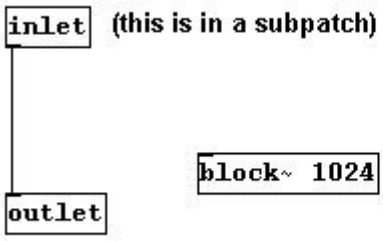
Podriamos presentar el análisis de FFT en una matriz:

patches/3-8-1-1-rfft-array.pd



De esta manera podemos observar el espectro de una señal. Note Bien: FFT transforma la información que transcurre en el tiempo en información de frecuencias; estas son actualizadas a partir de cada bloque nuevo. Hablamos generalmente de *dominio de tiempo* y *dominio de frecuencia*.

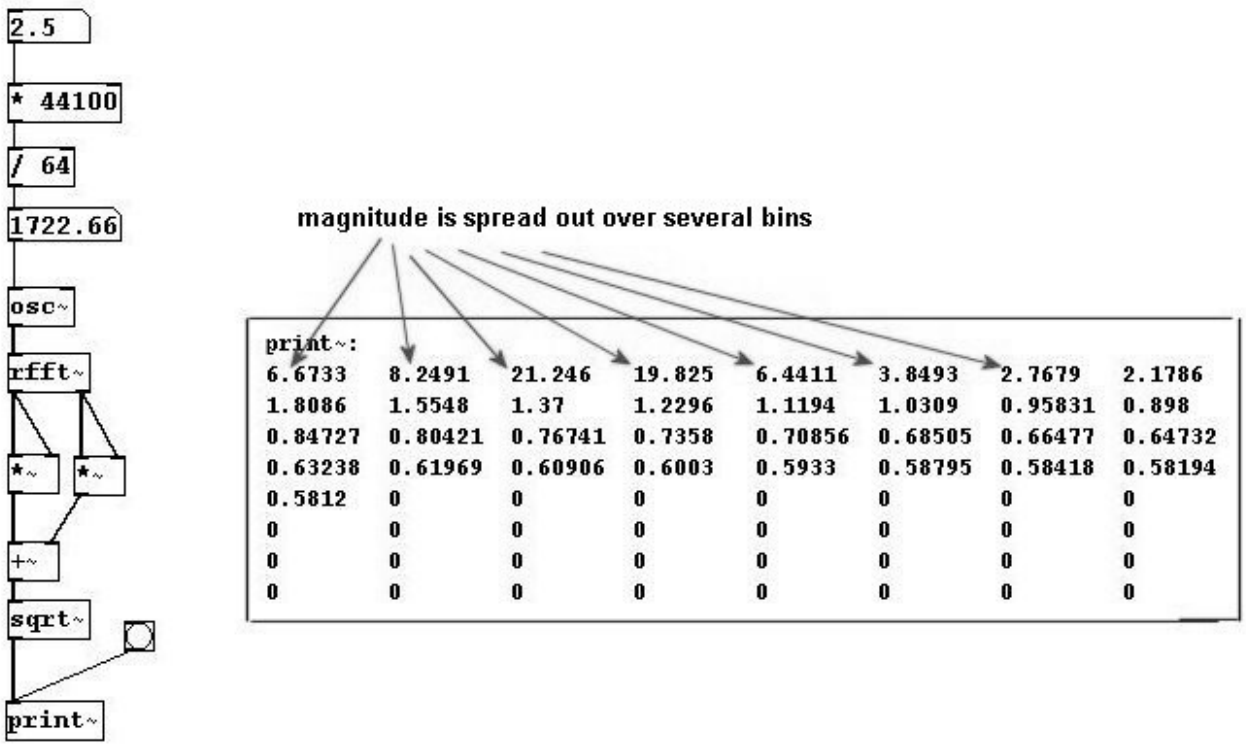
En Pd, el tamaño del bloque sólo puede ser cambiado en un subpatch. Esto lo logramos mediante el uso de "block~":



Cuando determine el tamaño del bloque, asegurese de considerar que un tamaño de bloque mayor le permite trabajar con frecuencias más graves. Por ejemplo: con un tamaño de 1024 muestras, cada recipiente tiene un tamaño de 44100/1024 = ~43 Hz, por lo tanto cuenta con una resolución más fina. La desventaja es que el proceso toma más tiempo.

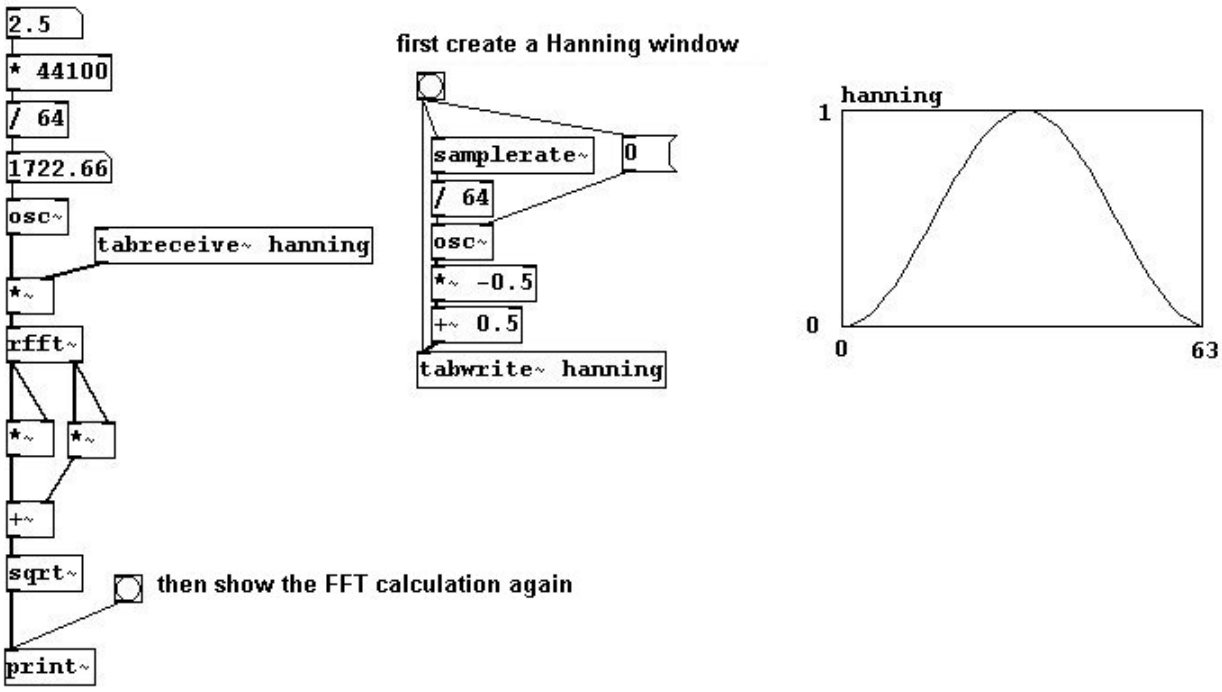
3.8.1.2 Analizando cualquier señal

Atengámonos a un tamaño de bloque de 64 muestras, el cual puede ser usado para analizar el espectro de una frecuencia fundamental de 689 Hz. ¿Pero que pasa si se producen otras frecuencias?

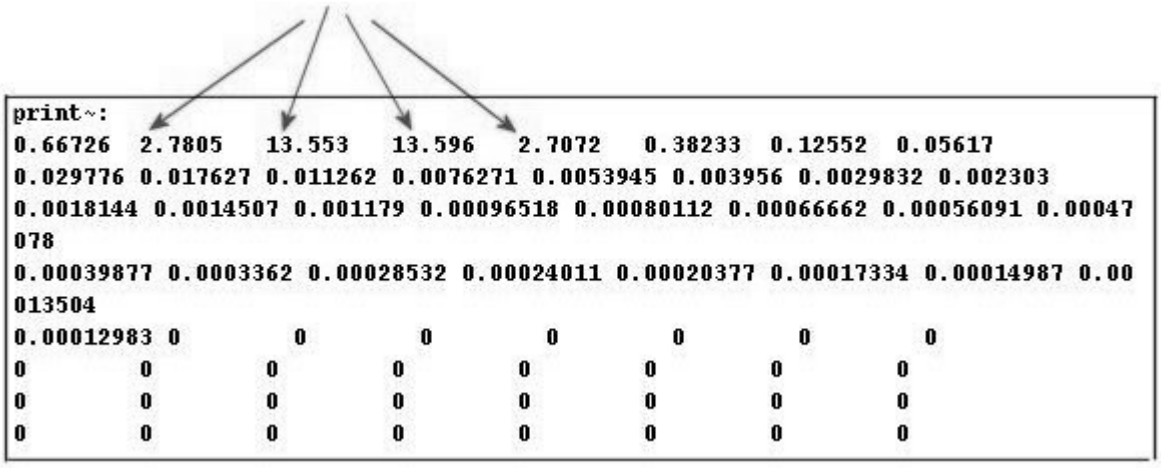


Entonces, la información es dividida entre muchos recipientes y la fase cambia con cada análisis. Este problema no puede ser resuelto completamente; debemos aplicar un "truco". La manera usual de resolver el problema es superponer ventanas como se hace en la síntesis granular; crear una versión enventanada de la original. Para llevarlo a cabo, podemos usar "tabreceive~" , un Objeto que siempre lee la información de una matriz con el tamaño de un bloque a travez de una ventaña Hanning -aquí con 64 muestras.

patches/3-8-1-2-rfft3.pd

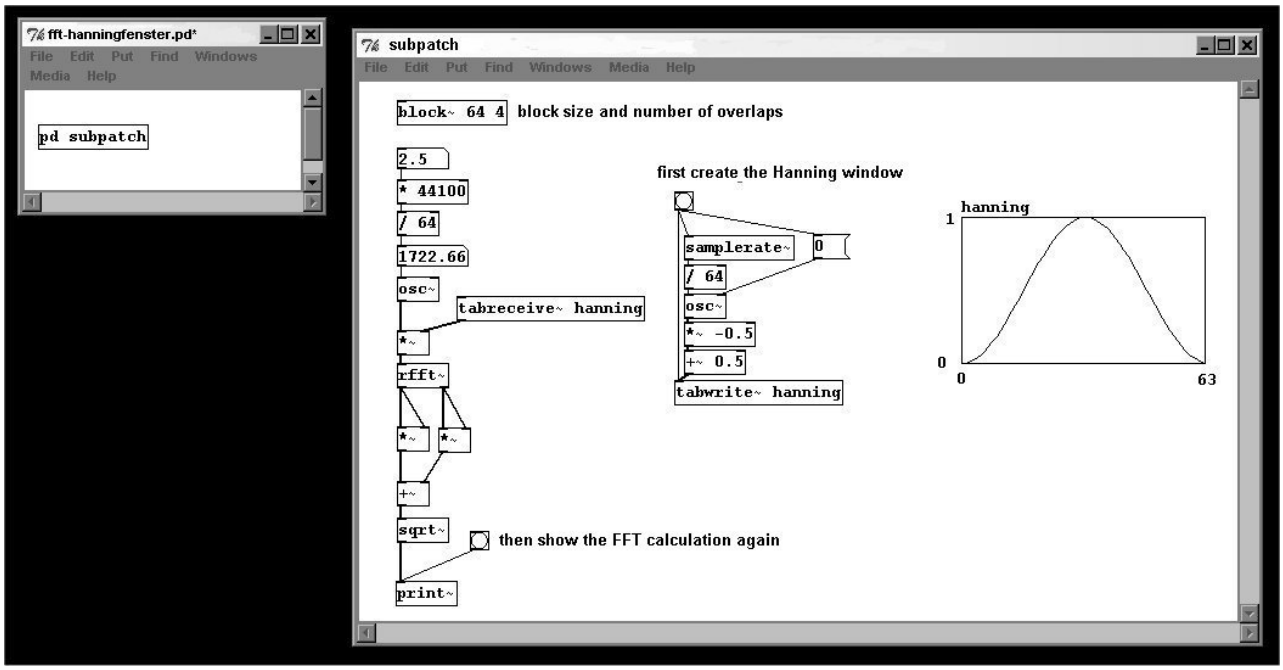


De esta manera, los valores de magnitud no se "extienden" demasiado.



Para agregar en relación al enventanado, las ventanas necesitan solaparse una a la otra. Esto es muy sencillo de llevar a cabo con Pd: sólo ingrese el número de ventanas (comúnmente 4) como segundo argumento de "block~". El resultado final también debe ser enventanado. La normalización adecuada para cuatro ventanas superpuestas es (3 * tamaño de bloque) / 2. Debido a que está usando "block~", debe colocar todo esto en un subpatch.

patches/3-8-1-2-fft-subpatch.pd



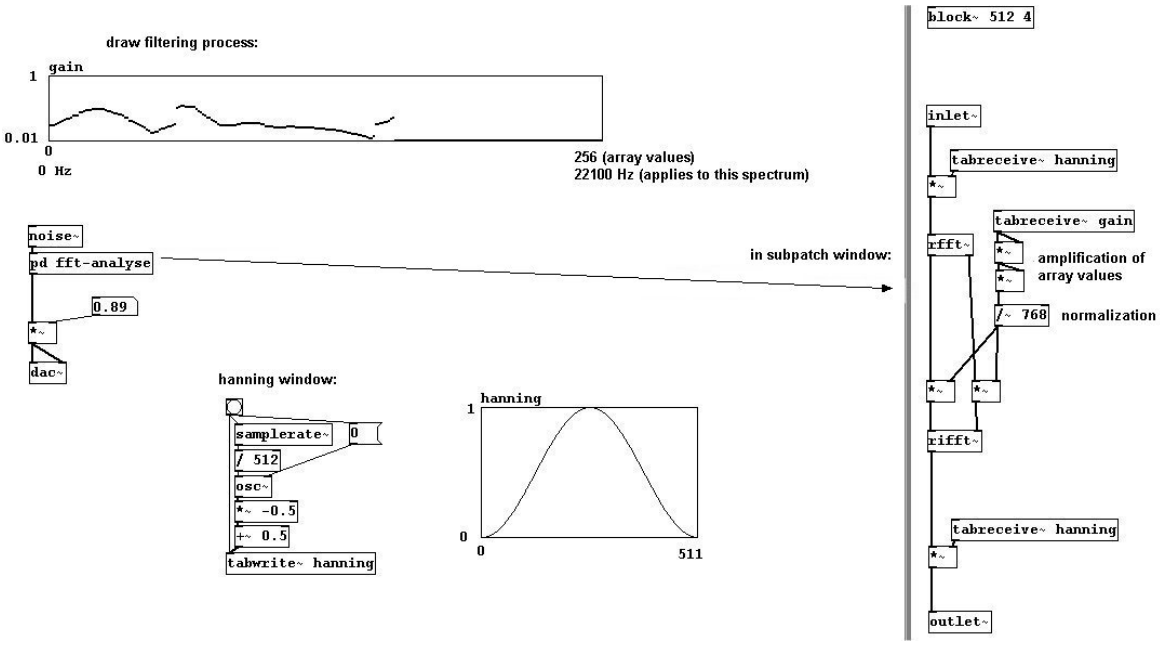
Mediante las técnicas de superposición y enventanado, tenemos muy buenas probabilidades de analizar correctamente una señal.

3.8.2 Aplicaciones

3.8.2.1 Filtros

Lo que resulta realmente útil al trabajar con FFT es que los valores determinados por el mismo pueden ser cambiados antes de resintetizar los componentes en un sonido resultante. Por ejemplo, podriamos modificar el volumen de ciertos recipientes; podriamos construir filtros pasa-altos, pasa-bajos, etc., o 'dibujar' alguno propio.

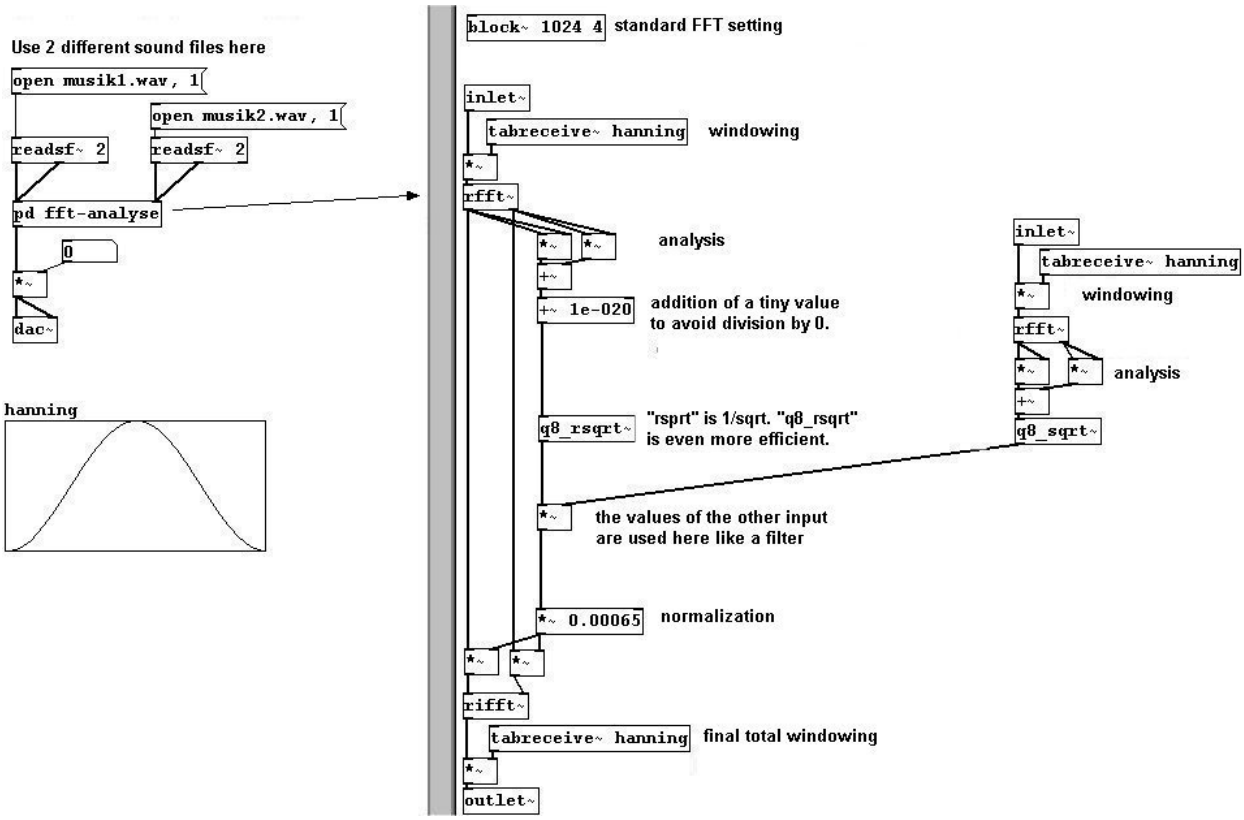
patches/3-8-2-1-fft-filter.pd



3.8.2.2 Plegado

La *convolucion* es un efecto muy apreciado -plegar una señal con otra; es decir, reproducir la media de sus amplitudes. El cálculo Hanning debería ser ya familiar para usted. Un tamaño de bloque de 1024 muestras y cuatro superposiciones corresponde al estandard.

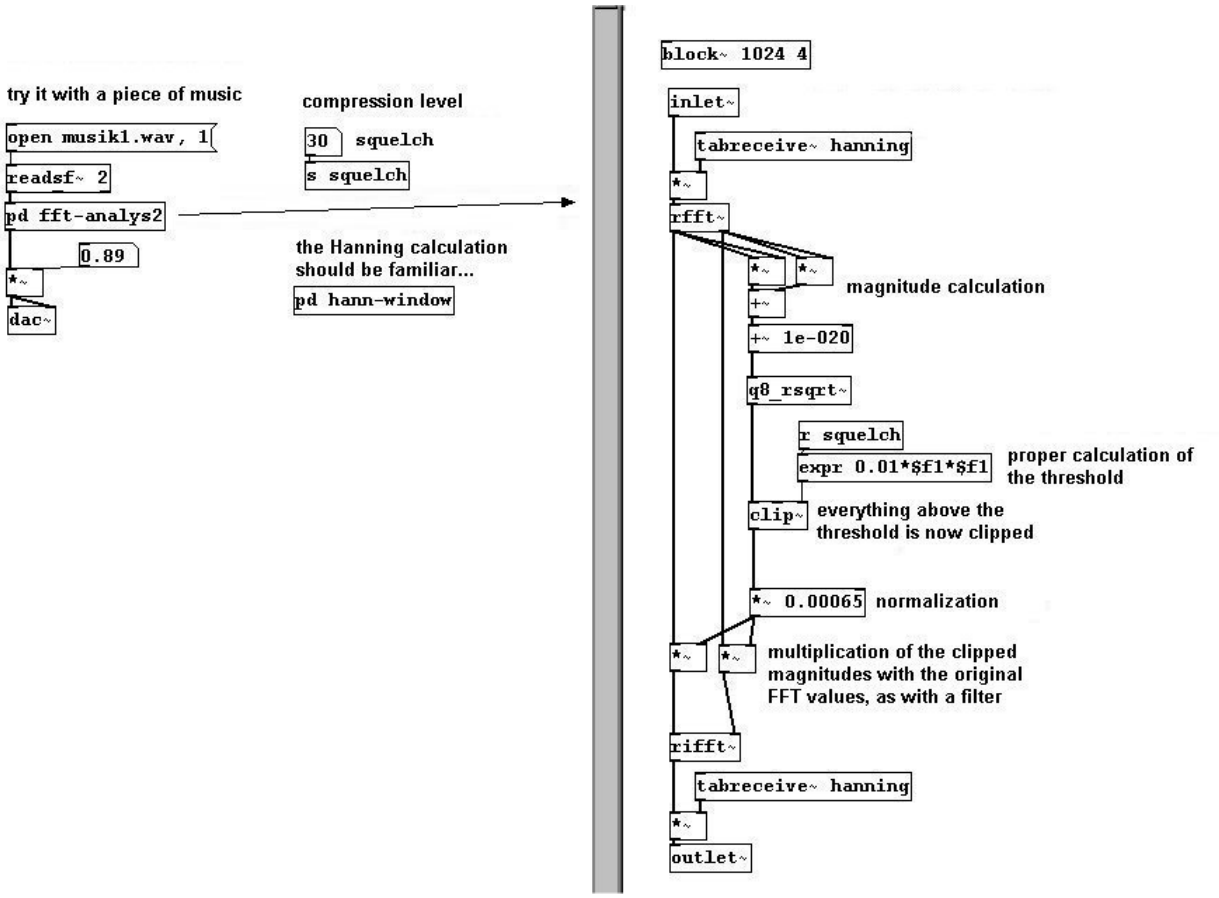
patches/3-8-2-2-convolution.pd



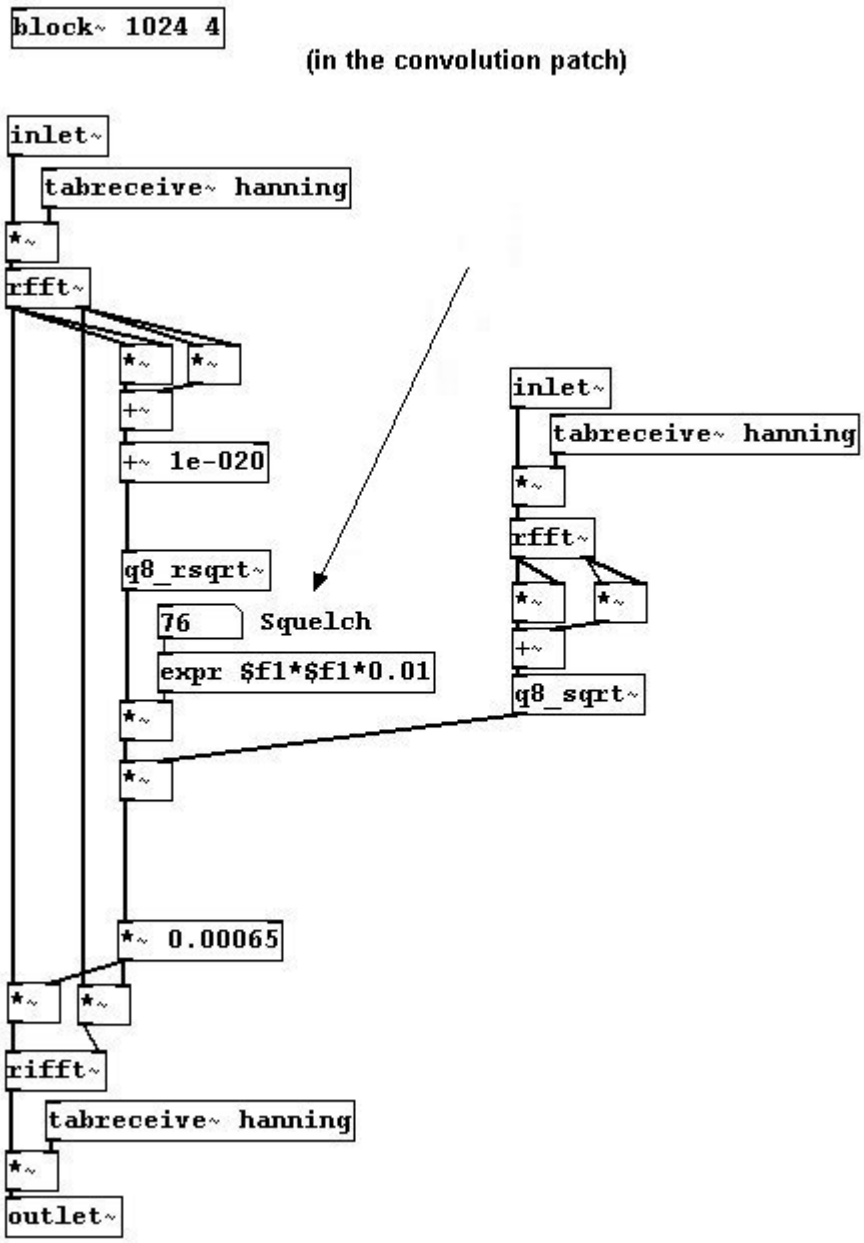
3.8.2.3 Compresor

También podría construir un compresor. Esto significa que podria amplificar, en un sonido, la parte del espectro que posee un volumen débil a fin de acercarlo a la parte del espectro con volumen más alto. Simplemente use los valores de magnitud como factores para las salidas de "rfft", aunque advierta que los valores que excedan un cierto umbral ("squelch" en la ilustración) serán simplemente recortados en dicho punto:

patches/3-8-2-3-compressor.pd



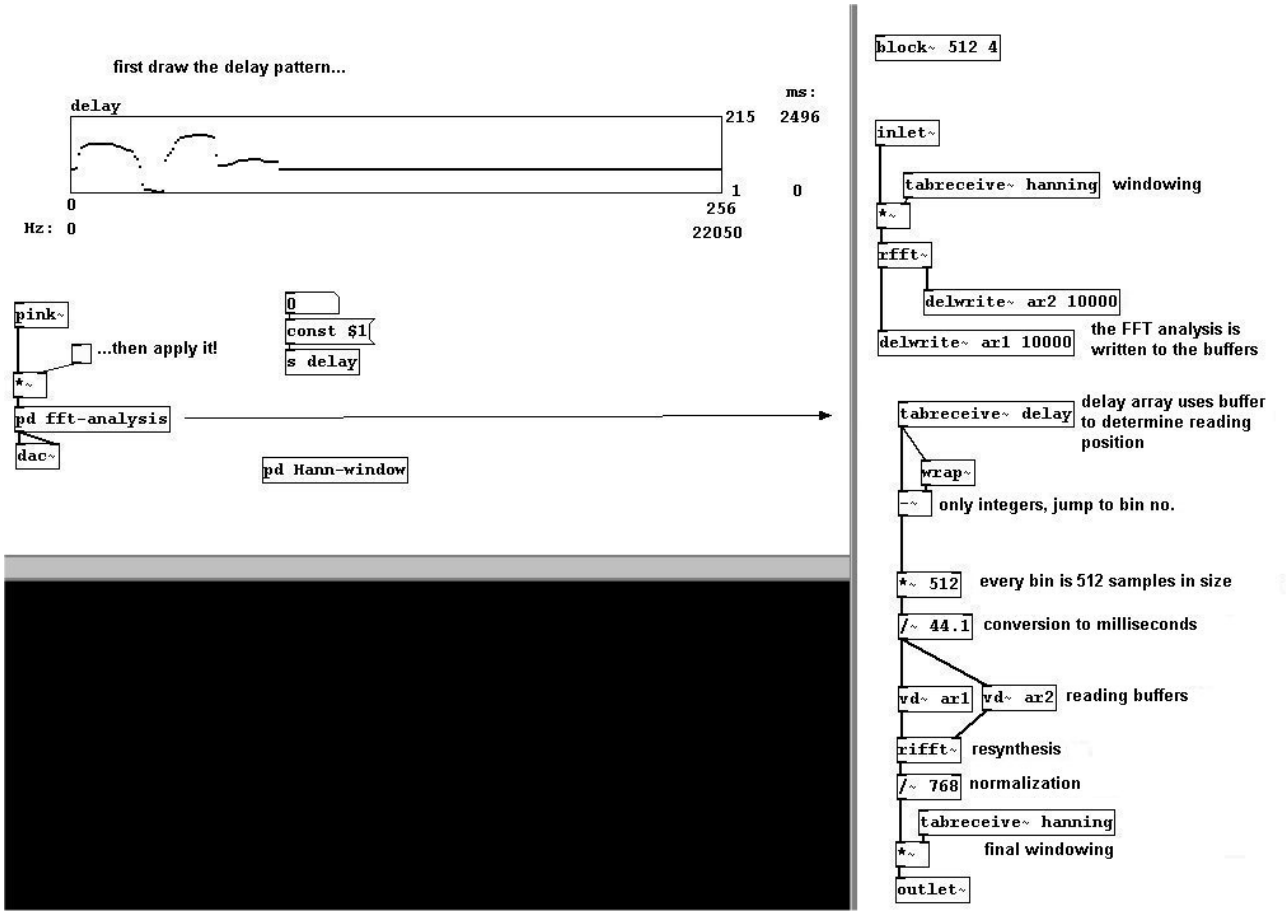
Si implementa esto en la conexión de uno de los dos análisis, obtiene un efecto de convolución más rico:



3.8.2.4 Retardo espectral

También puede reproducir ciertos recipientes con diferentes cantidades de retardo para realizar lo que es llamado "retardo espectral". El análisis FFT es escrito en dos buffers diferentes. Usando una matriz, determina el retardo para cada recipiente. El retardo máximo es de ca. 2500 ms, ya que cuenta con un buffer de 10000 ms pero con cuatro superposiciones ("block~"), lo que significa 10000/4 = 2500. Para ser precisos, es en realidad 2496 ms: 2496 * 44.1 = ca. 110080 muestras, lo cual resulta en 110090 / 512 = 215 posibles posiciones de recipiente. Ya que generalmente la señal entrante no encaja en el tamaño del recipiente, los valores del análisis son divididos entre recipientes colindantes (cf. 3.8.1.2). Si dichos recipientes colindantes ocurren en distintos tiempos, puede haber reducciones de volumen.

patches/3-8-2-4-spectral-delay.pd

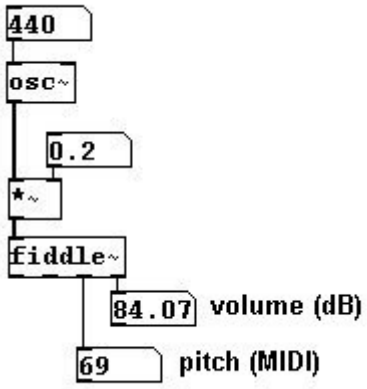


¡Intentelo con una pieza de música memorable!

3.8.3 Apendice

3.8.3.1 fiddle~

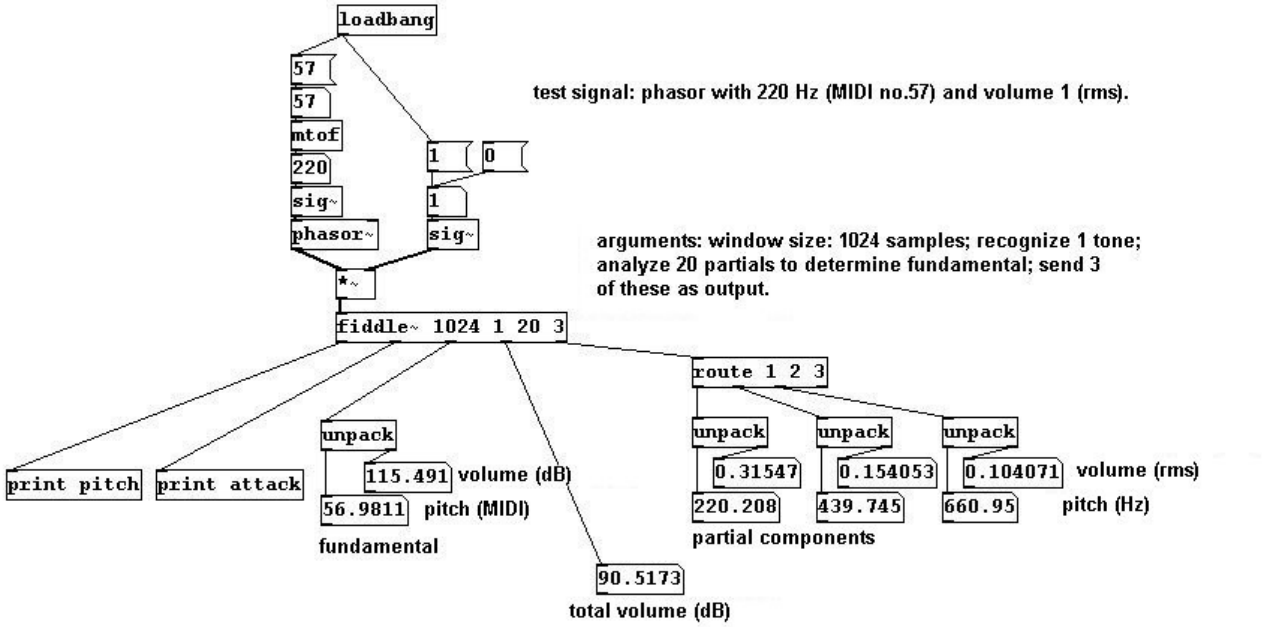
Disponemos en Pd de un Objeto basado en el algoritmo FFT que realiza un análisis de volumen Y altura a la vez. Se llama "fiddle~". También determina los volumenes y parciales de la señal entrante.



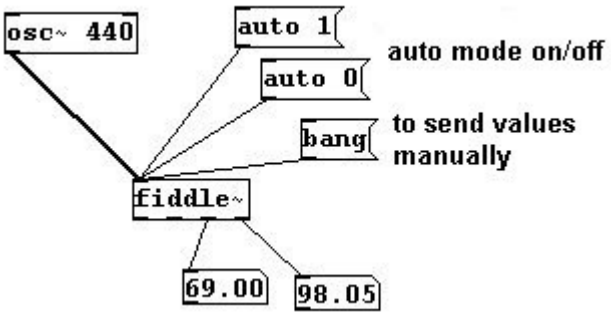
Los argumentos que recibe son: 1. Tamaño de ventana (en muestras), 2. Número de tonos simultáneos a ser reconocidos (máx. Tres tonos diferentes), 3. Número de picos a encontrar, y 4. Número de picos de salida. La configuración por defecto es: 1: 1024, 2: 1, 3: 20, 4: 0. Como salidas

(de izquierda a derecha): 1. Altura en MIDI (sólo cuando encuentra un cambio), 2. Volumen en dB (sólo cuando encuentra un cambio extremo ("ataque")), 3. Altura y volumen de la fundamental (como una lista), 4. Volumen total, y 5. Parciales individuales con sus respectivos volumenes (¡en Hertz / RMS! - también como una lista).

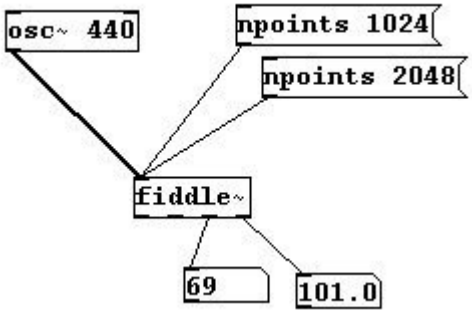
patches/3-8-3-1-fiddle.pd



Mensajees para "fiddle~": para evitar un procesamiento constante de datos, puede desactivar el "auto mode" (modo automático) y activar a cambio el "poll mode" (modo encuesta); sólo expide datos cuando recibe un mensaje 'bang':

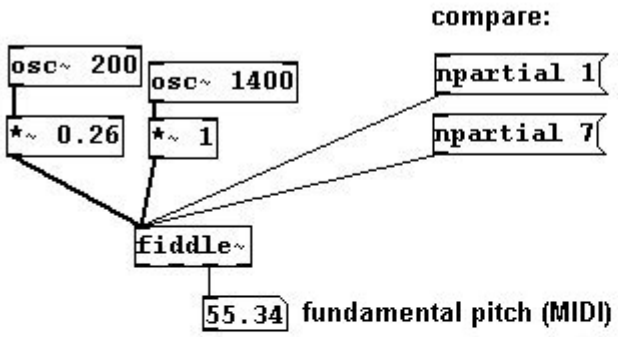


Puede determinar un tamaño de ventana (multiplos de dos):



Los parciales más agudos no son analizads tan intensamente para determinar la fundamental- Sin

embargo puede cambiar esto instruyendo al Objeto que analice un determinado parcial al menos con la mitad de intensidad que la aplicada a la fundamental:



We use 200 Hz as the fundamental. The fundamental is played quietly, the 7th partial (1400 Hz) loud. Fiddle first identifies the louder tone as the fundamental until we tell it that the 7th partial is especially loud. It then concludes that the louder tone must be the 7th partial and correctly identifies the fundamental.

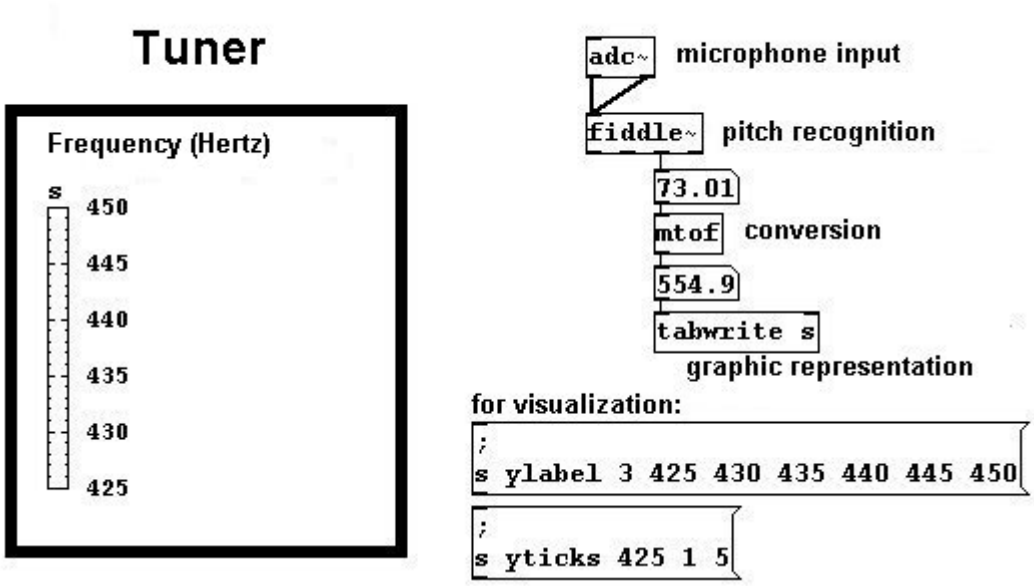
Esto es útil en los casos donde ciertos parciales de la señal entrante son especialmente fuertes (por ejemplo, el tercer parcial de un clarinete).

Note Bien: La señal de entrada es analizada cada medio tamaño de ventana, es decir, si el tamaño de ventana es de 1024, entonces corresponde a 512 muestras, lo cual equivale a 11.6 ms. La frecuencia más pequeña que "fiddle~" puede reconocer es (44100 / tamaño de ventana) * 2.5; para un tamaño de ventana de 1024 muestras, resulta en ca. 108 Hz..

3.8.3.2 Afinador

Aquí presentamos un aforma de construir un afinador:

patches/3-8-3-2-tuner.pd

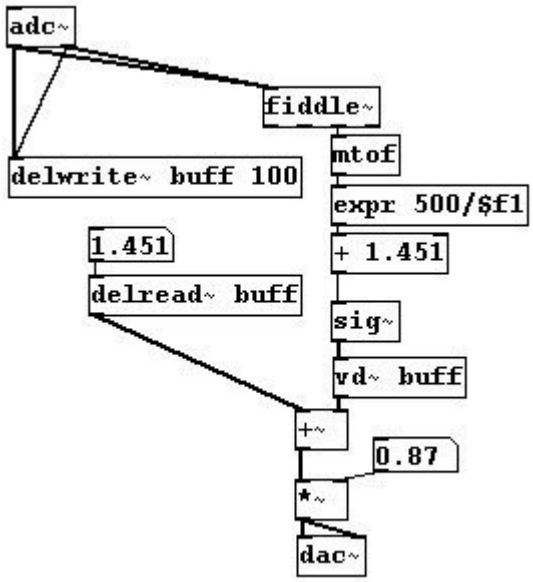


Para esta visualización fue utilizada una matriz con un sólo lugar de almacenamiento.

3.8.3.3 Octavador #2

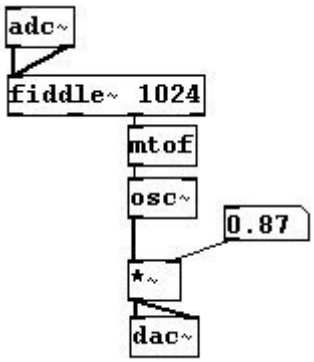
Para el octavador descrito en 3.4.2.9, puede ahora usar una entrada de micrófono siempre que la fundamental pueda ser usada para conducir los cálculos (es decir, siempre que la entrada de señal sea periódica y pueda ser entendida por "fiddle~"):

patches/3-8-3-3-oktavedoubler-fiddle.pd



3.8.3.4 Seguidor de altura

Podemos imaginar muchas aplicaciones interesantes usando el Objeto "fiddle~" de esta manera. Un ejemplo prototipo sería usar una entrada de micrófono, como una voz cantante, y 'seguir' el contorno melódico de la voz como un puntero laser:



Se plantea el siguiente dilema: existe siempre un retardo en el uso de "fiddle~". Mientras más pequeño sea el tamaño de ventana, más pequeño resulta el retardo. Sin embargo, mientras más pequeño sea el tamaño de ventana, más alto resultará el rango más bajo de altura que puede ser reconocido. Aun más, el resultado de "fiddle~" es siempre un poco caótico. Puede aprender cómo minimizarlo en 4.3.1.3

3.8.3.5 Más ejercicios

En vez de sólo 'seguir' la entrada de micrófono, puede crear una voz paralela a una distancia de quinta perfecta o incluso un acorde paralelo.

3.9 Correciones de amplitud

3.9.1 Teoría

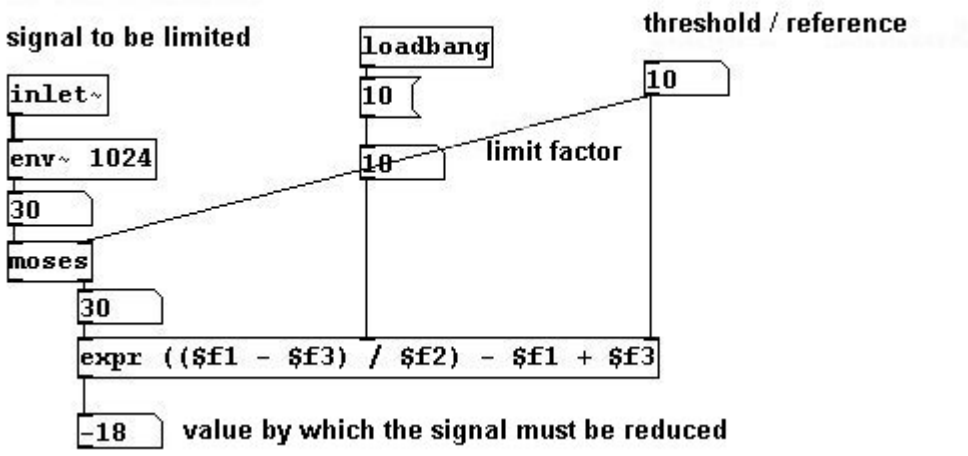
Al final de este largo capítulo dedicado a las técnicas de audio en Pd, nos concentraremos en los procesamientos de amplitud.

3.9.1.1 Limitador

Como aprendío en 3.1.2.1.2, la membrana del altoparlante puede vibrar hasta cierto punto; a partir del mismo ocurre un "recorte". Sin embargo puede construir un dispositivo automático que reduzca partes muy intensas de una señal antes que sean recortadas. En la ingeniería acústica, llamamos alñ dispositivo que realiza dicha tarea "limitador".

En un limitador tenemos una sóla entrada de señal cuyo volumen debe ser medido. Si se excede el límite superior, su volumen será reducido de acuerdo a un factor ya determinado hasta que alcance el punto de referencia. (En la siguiente sección, el volumen es calculado en dB.)

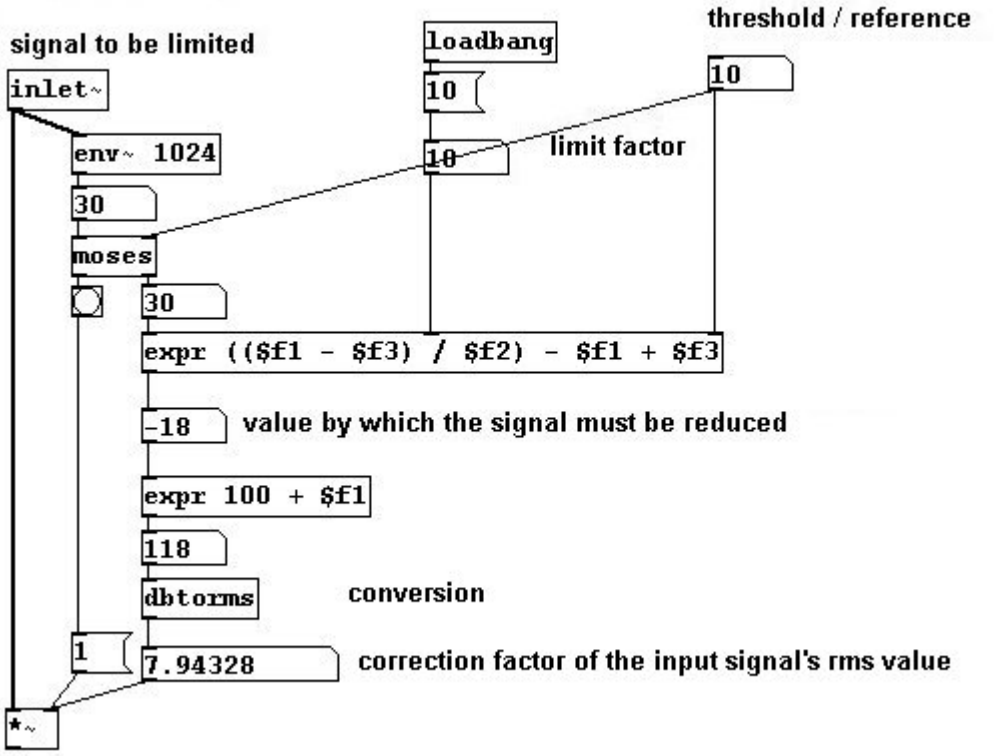
patches/3-9-1-1-limiter1.pd



Es este primer ejemplo, el umbral (threshold) es de 10 dB, el factor 10, y la señal de entrada es de 30 dB. La diferencia entre la señal de entrada y el punto de referencia es de 20 dB. El factor determina que debería ser reducido (por un factor de 10) a 2 dB; es decir, la señal de entrada debe ser reducida por -18 dB a un valor de 12 dB.

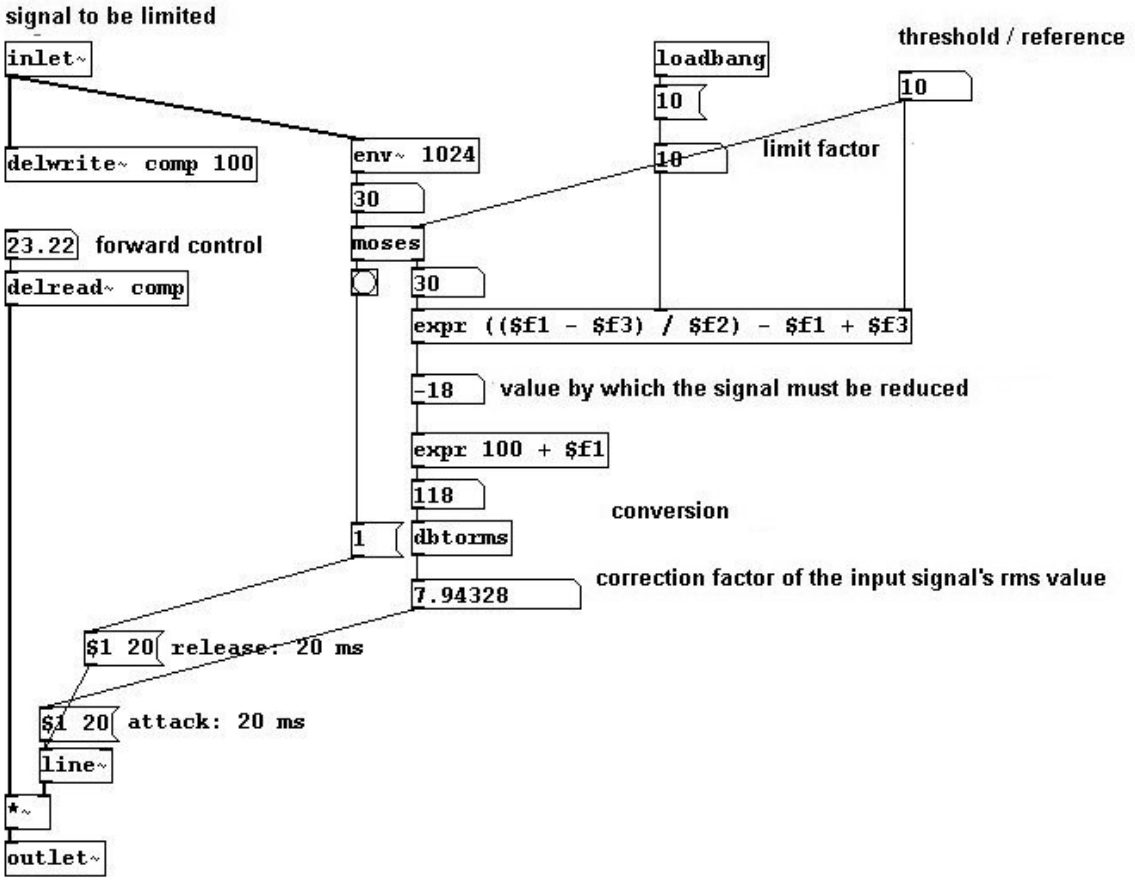
Ahora aplicaremos el factor de limitación a la señal original cuando exceda el umbral; si permanece por debajo del umbral, el factor simplemente se mantiene en 1:

patches/3-9-1-1-limiter2.pd



Hay dos aspectos a observar: "env~" toma el promedio de la muetra dada, aquí seimpre es una ventana de 1024 muestras. Esto resulta en un retardo de of 1024 / 44.1 = 23.22 ms que podemos configurar. Pero también podría realizar la reducción antes de que la señal exceda el umbral, lo cual retardaría aún más la señal original. En los procesamientos de señal, el término para esto es "acción de control de anticipación" (feedforward control action). Si el retardo de la señal original es más corta y el retardo de la corrección más larga, hablamos de "acción de control reversa" (reverse control action). En este caso, el umbral es excedido brevemente antes de que se realice corrección alguna. Algunas variables que deben ser definidas aquí son la velocidad en la cual la corresción toma lugar y también la velocidad con la cual la señal regresa al volumen original una vez que caiga nuevamente por debajo del umbral, estos son llamados tiempos de "ataque" y "liberación".

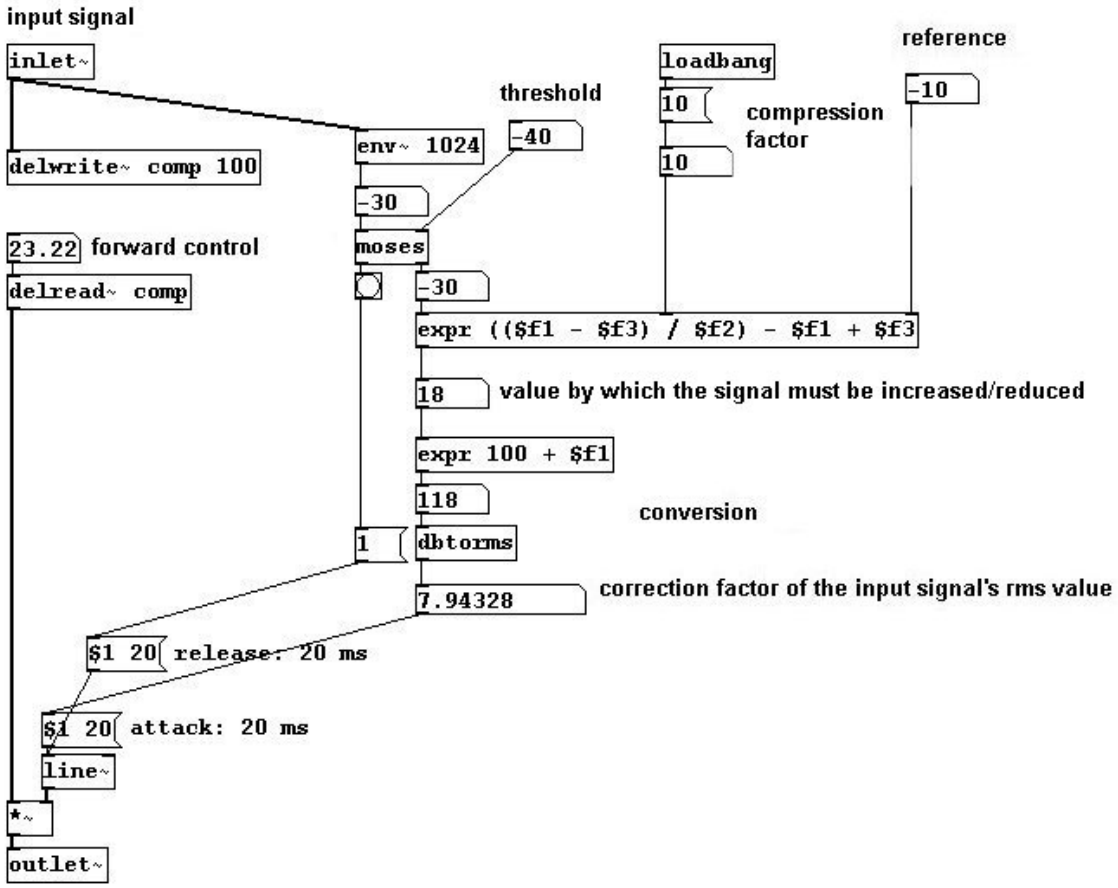
patches/3-9-1-1-limiter3.pd



3.9.1.2 Compresor

Una vez que haya establecido el umbral superior (luego del cual la correción ocurre) y el punto de referencia (el punto al cual la corrección se aproxima), puede también ajustar las configuraciones para que el volumen que se encuentra por debajo del punto de referencia sea amplificado y aquel por encima del mismo sea disminuido. El dispositivo que realiza esta tarea se llama *compresor*:

patches/3-9-1-2-compressor.pd



3.9.2 Aplicaciones

3.9.2.1 Tonos Larsen

Si acerca un micrófono de entrada a un altoparlante -con el Objeto "adc~" directamente conectado al "dac~"-, pronto escuchará un tono (¡y aleje rápidamente el micrófono del altoparlante!). Esto ocurre porque el aire siempre contiene un poco de ruido; el microfono lo capta y lo envía al altoparlante; el altoparlante lo reproduce y el micrófono toma la señal amplificada; etc.

Dependiendo de la distancia entre micrófono y parlante, la señal es amplificada en cada momento. Dependiendo también de la habitación, el largo del cable, y la latencia del ordenador, obtenemos una resultante de tonos periodicos muy agudos, también llamados "tonos Larsen". Al mismo tiempo, el volumen se incrementa dramáticamente, como resultado de la señal constantemente amplificada. Este representa el clásico caso de feedback -un circuito, un sistema recursivo. Siempre que se utilicen micrófonos y altoparlantes, corremos el peligro de generar feedback. Un limitador usado entre el micrófono y el parlante ayudaría a aliviar este peligro.

3.9.2.2 Más ejercicios

a) Crear un "expansor": ¡convierta pequeñas diferencias en amplitud en grandes diferencias!

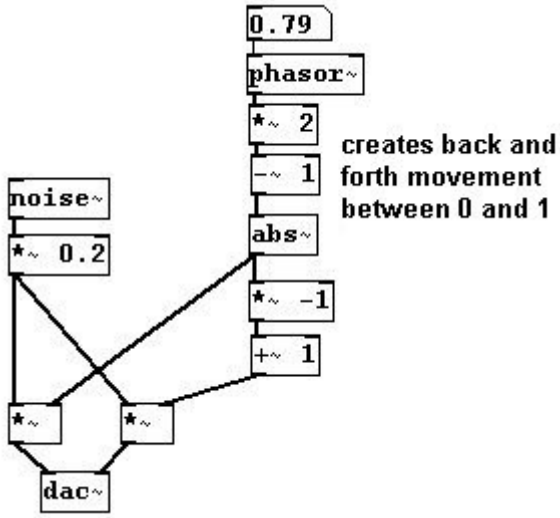
b) Un inversor de volumen: cambie volumen bajo por alto y viceversa.

3.9.3 Apéndice

3.9.3.1 Movimiento en el espacio

El volumen puede ser utilizado para simular movimiento en el espacio. Normalmente contamos con un par de altoparlantes stereo y por lo tanto dos entradas para el "dac~". Si cambia los volumenes relativos de los parlantes gradualmente -siempre que usted se encuentre directamente en el medio de ambos-, puede experimentar cómo 'viaja' el sonido entre los dos parlantes. Llamamos a esto 'fuente de sonido fantasma'.

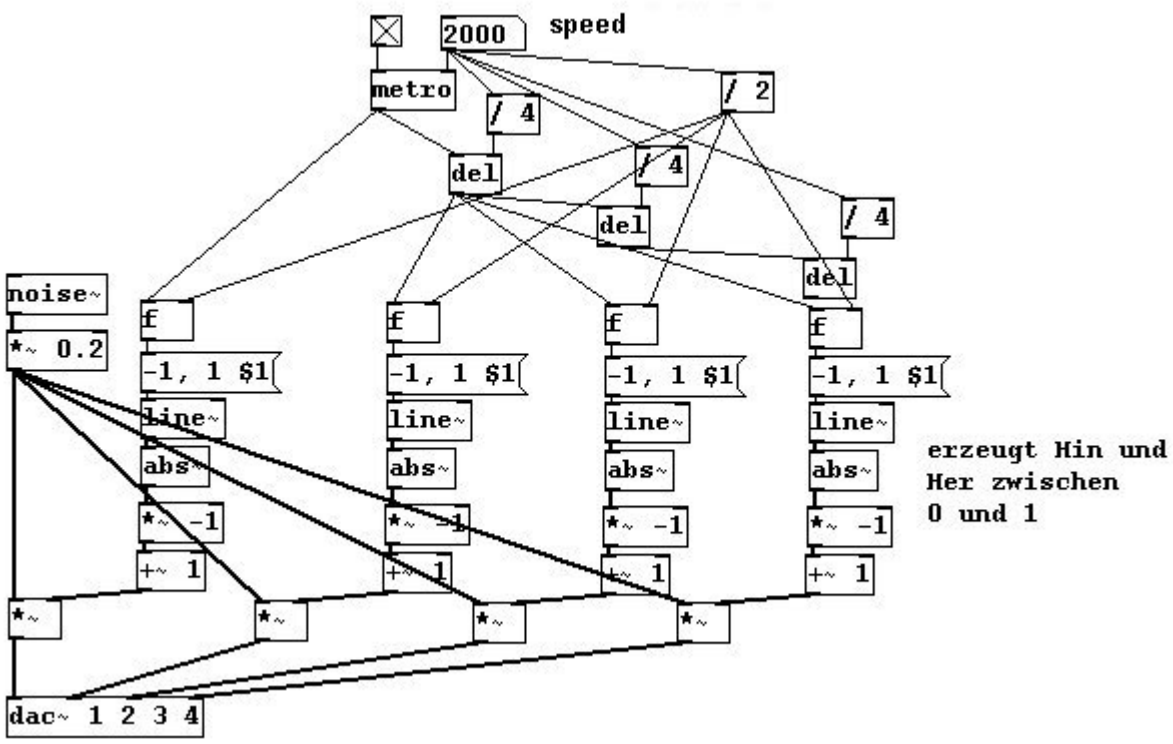
patches/3-9-3-1-spatial-stereo.pd



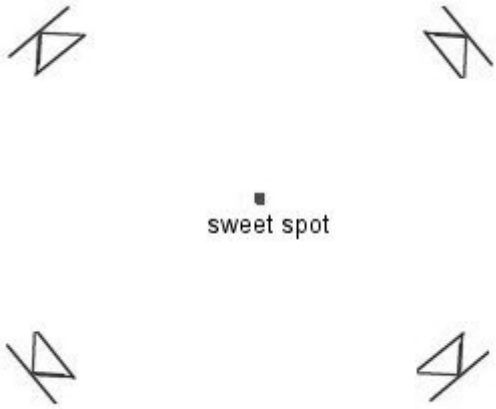
Configuración de altoparlantes:

Si tiene, digamos, cuatro parlantes en una configuración cuadrafónica, puede crear movimiento circular a través del espacio (naturalmente, esto requiero una placa de sonido con cuatro salidas independientes; como argumentos, puede ingresar la las entradas al "dac~"):

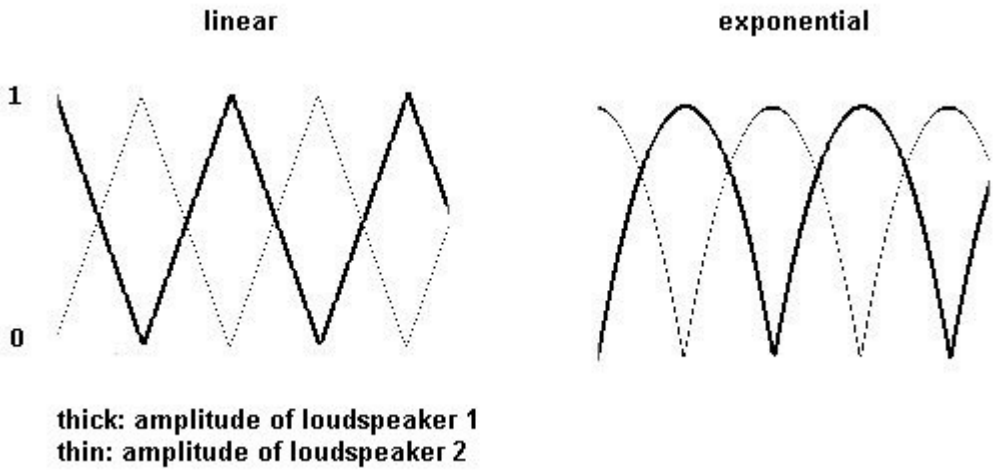
patches/3-9-3-1-spatial-quadro.pd



Configuración:



En este caso, el effecto sólo trabaja correctamente si usted se encuentra precisamente en el medio (en el "sweet spot"). En el ejemplo anterior, tambien puede escuchar una 'brecha' en volumen entre los dos niveles de volumen más alto para cada parlante. Debe experimentar si la superposición de volumen debiera ser lineal o exponencial (cf. los tipos de ventana en 3.9.4). El compositor tiene que usar su oido y decidir por si mismo.

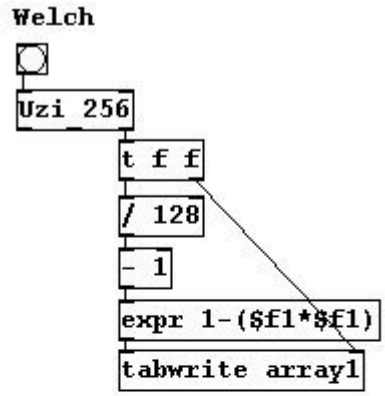


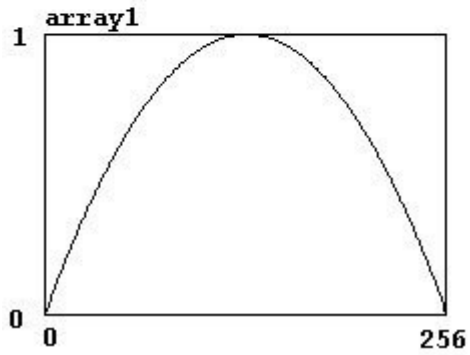
3.9.4 Para los interesados, especialmente

3.9.4.1 Otras ventanas

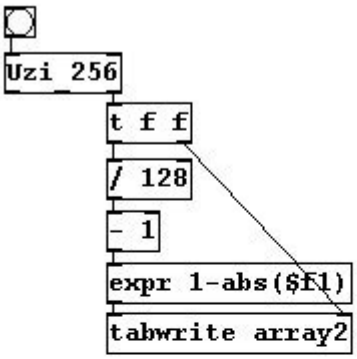
En capítulos previso, fueron usadas frecuentemente las ventanas "Hanning" (la cual corresponde a parte de una función coseno) para evitar clicks. Pero también disponemos de otros tipos de ventanas que podrian ayudarnos para experimentar:

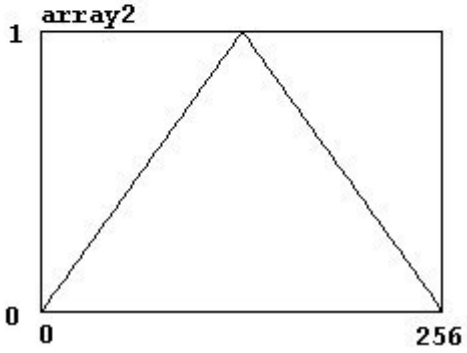
patches/3-9-4-1-windowing.pd



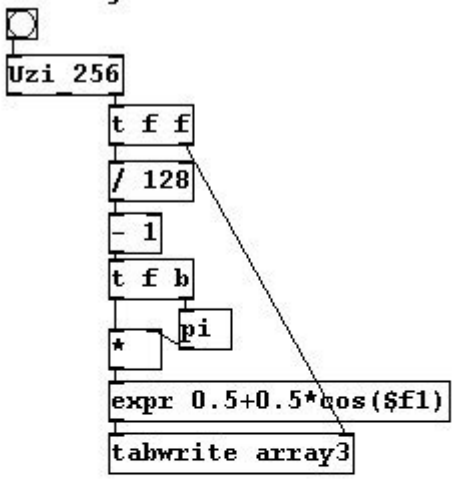


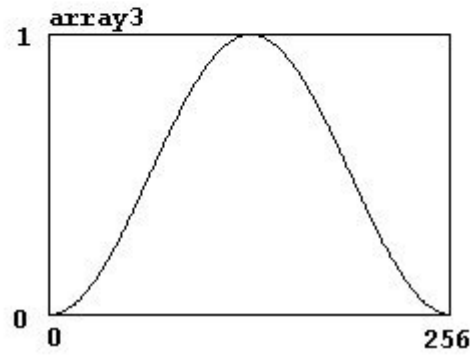
Bartlett

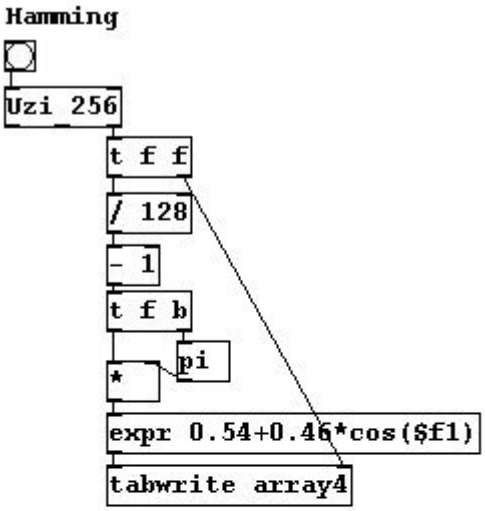


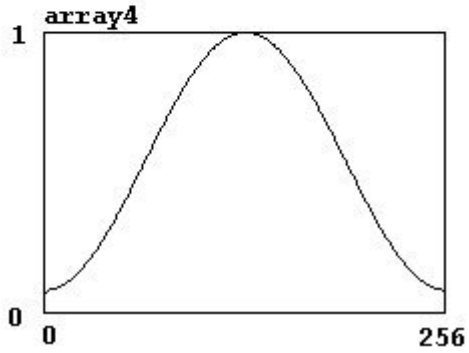


Hanning

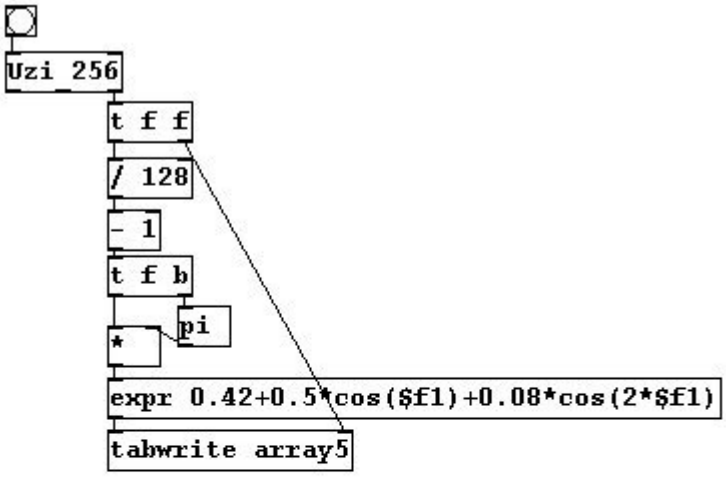


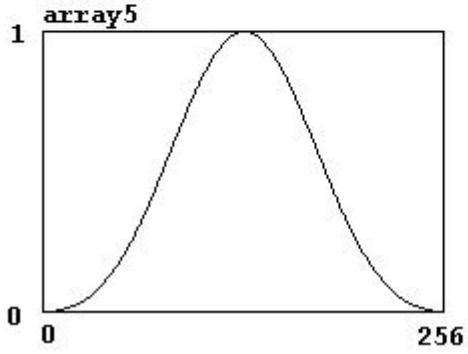



Hamming
Uzi 256
t f f
/ 128
- 1
t f b
pi
*
expr 0.54+0.46*cos($f1)
tabwrite array4

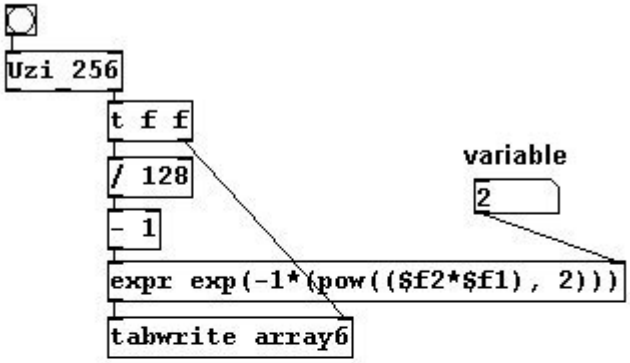

array4
1
0
0
256

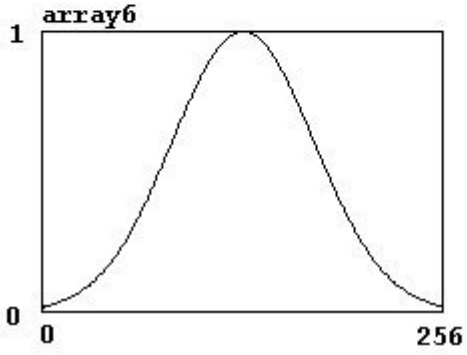
Blackman





Gaussian





Capítulo 4. Control de sonido

Table of Contents

La música se desarrolla en el tiempo y el compositor desea naturalmente que la misma cambie en el tiempo. En el capítulo anterior, revisamos los conceptos básicos de la generación de sonido. Ahora veremos cómo usar Pd para controlar estos sonidos generados -o controlar los controles de estos sonidos- en el tiempo

4.1 Algoritmos

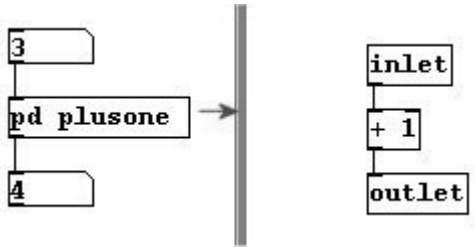
4.1.1 Teoría

4.1.1.1 ¿Qué entendemos por algoritmo?

Un algoritmo es un término técnico para la descripción de una secuencia de pasos en un procedimiento que un programa de ordenador ejecuta.

Si cuenta con un subpatch que adiciona el valor 1 a un número ingresado, puede considerar a este como un algoritmo (muy simple): el algoritmo de este subpatch consiste en la adición del valor de 1.

patches/4-1-1-1-plus-one-algorithm.pd



En esencia, cada Objeto en Pd ejecuta un algoritmo. Lo que antes se requería como dispositivo llamado generador de ruido, hoy lo realiza el algoritmo contenido en el Objeto "noise~".

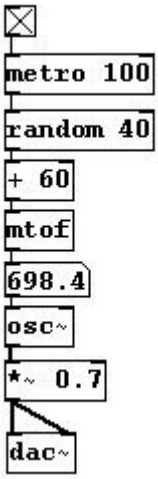
En este capítulo, estamos particularmente interesados en el desarrollo de algoritmos que, una vez iniciado el proceso, el ordenador pueda ejecutar completamente por sí mismo y que cumpla el objetivo de cambiar al sonido en el tiempo. Ya hemos visto algunos ejemplos de esto, como en 2.2.3.2.7

4.1.2 Aplicaciones

4.1.2.1 Estocástica

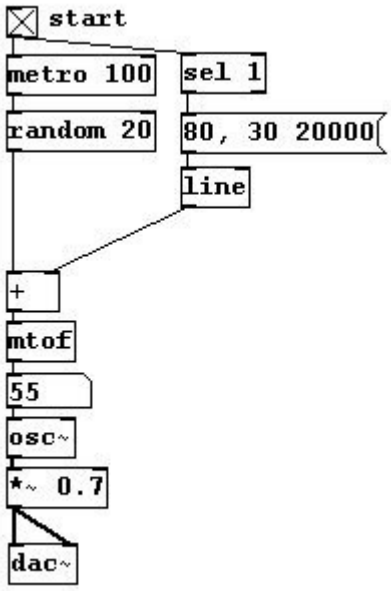
Una forma simple pero muy usual de hacer que el ordenador opere por sí mismo es utilizar un generador aleatorio...

patches/4-1-2-1-random.pd



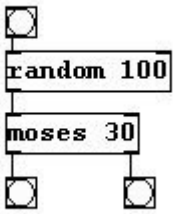
Puede limitar esta selección aleatoria para que cambie asímisma:

patches/4-1-2-1-random-limits.pd



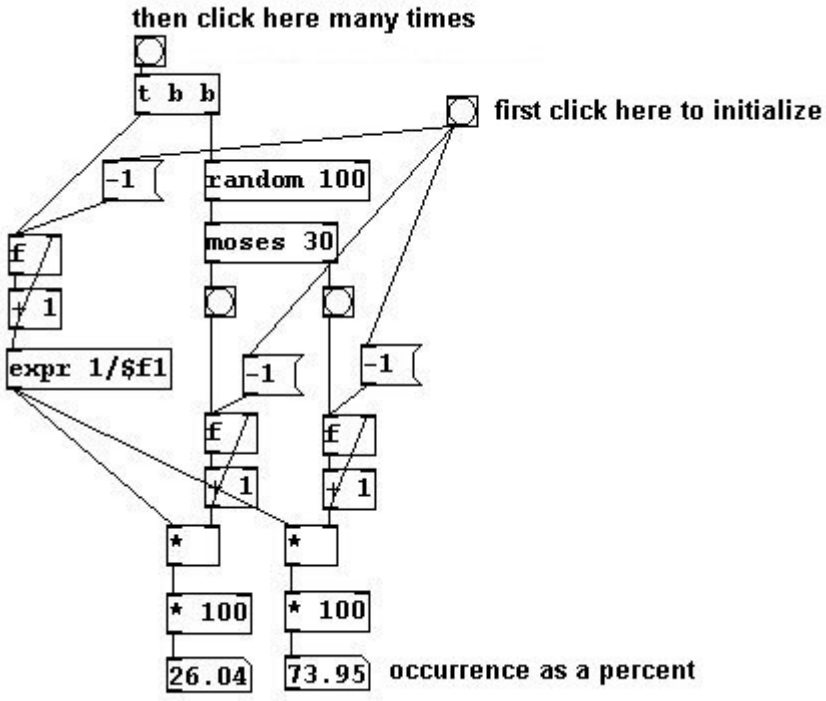
El resultado de un generador aleatorio sigue las leyes de la estocástica, de la probabilidad. Con "random 6", cada número de 0 a 5 tiene una probabilidad de 1/6. Aunque es altamente improbable, es posible que uno de los números no aparezca jamás o que no aparezca por un tiempo muy

prolongado. Esta probabilidad también puede ser directamente controlada:



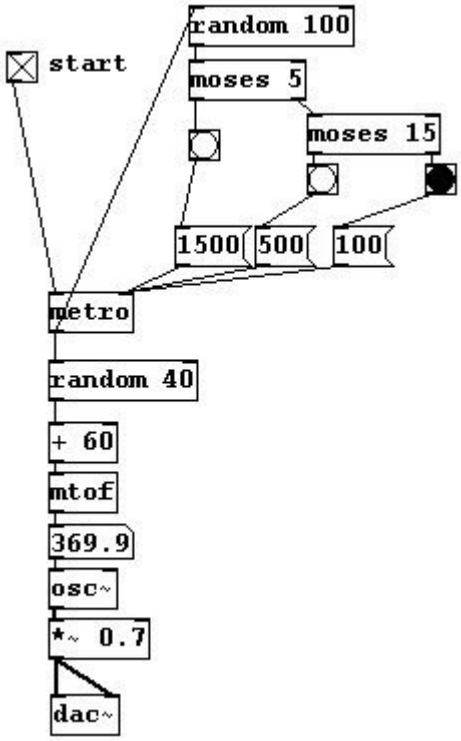
Como podemos ver aquí, la probabilidad de que obtengamos un bang en la salida izquierda es de un 30 % mientras que en la salida derecha es de un 70% . Puede examinarlo como puede ver a continuación:

patches/4-1-2-1-probability.pd

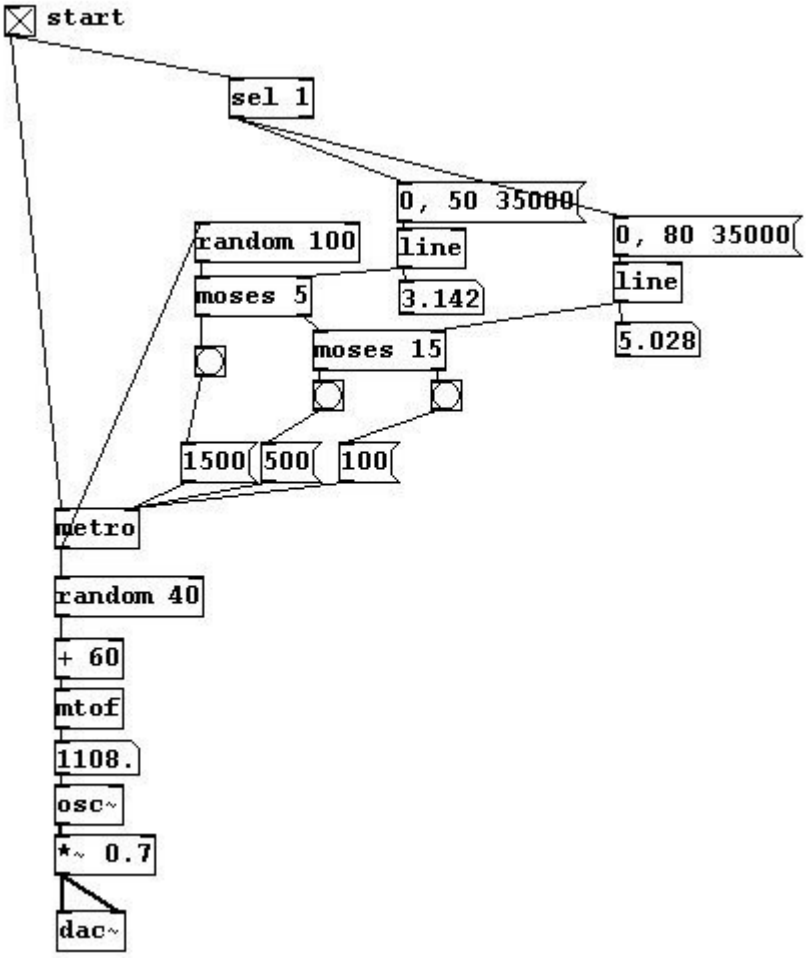


Puede usar este principio para seleccionar diferentes duraciones para ciertos eventos sonoros: los cortos se dan muy frecuentemente, los medios ocasionalmente y los largos rara vez.

patches/4-1-2-1-probability-examples.pd

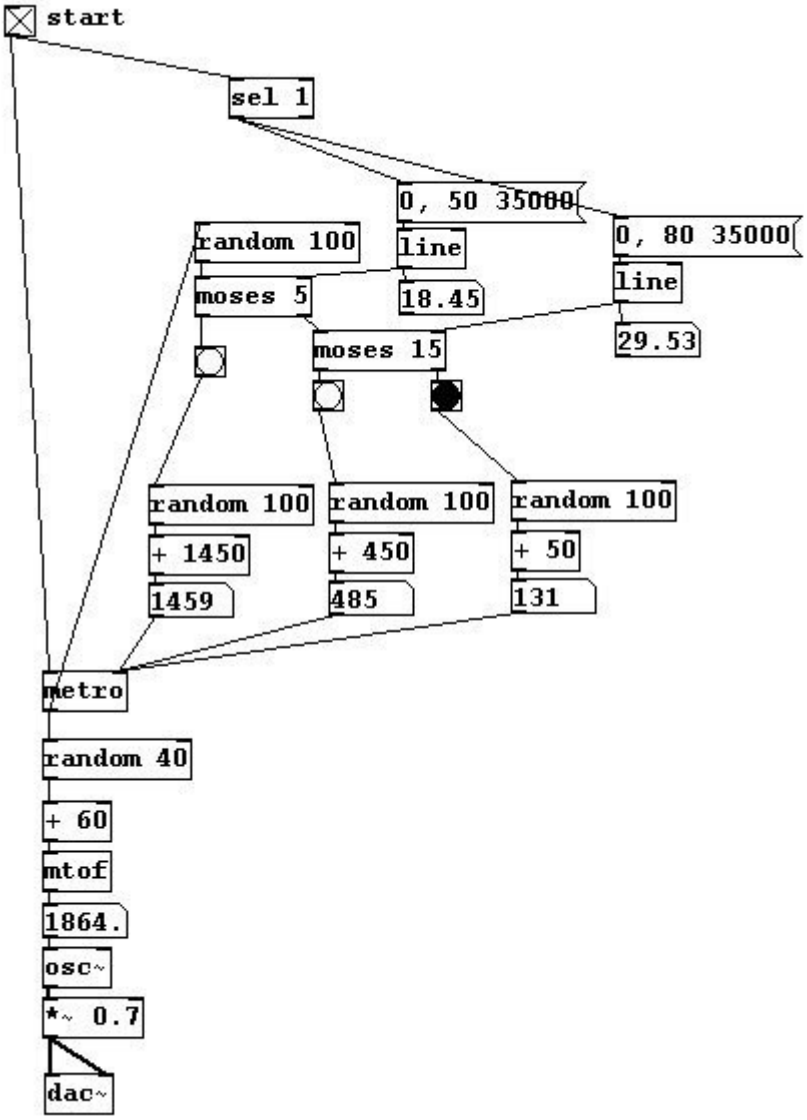


También podría variar esta distribución en el tiempo...



Al principio obtenemos sólo duraciones cortas, al final principalmente duraciones largas (las cuales por supuestos requieren mucho tiempo).

También se puede introducir un poco de variación a las diferentes duraciones:



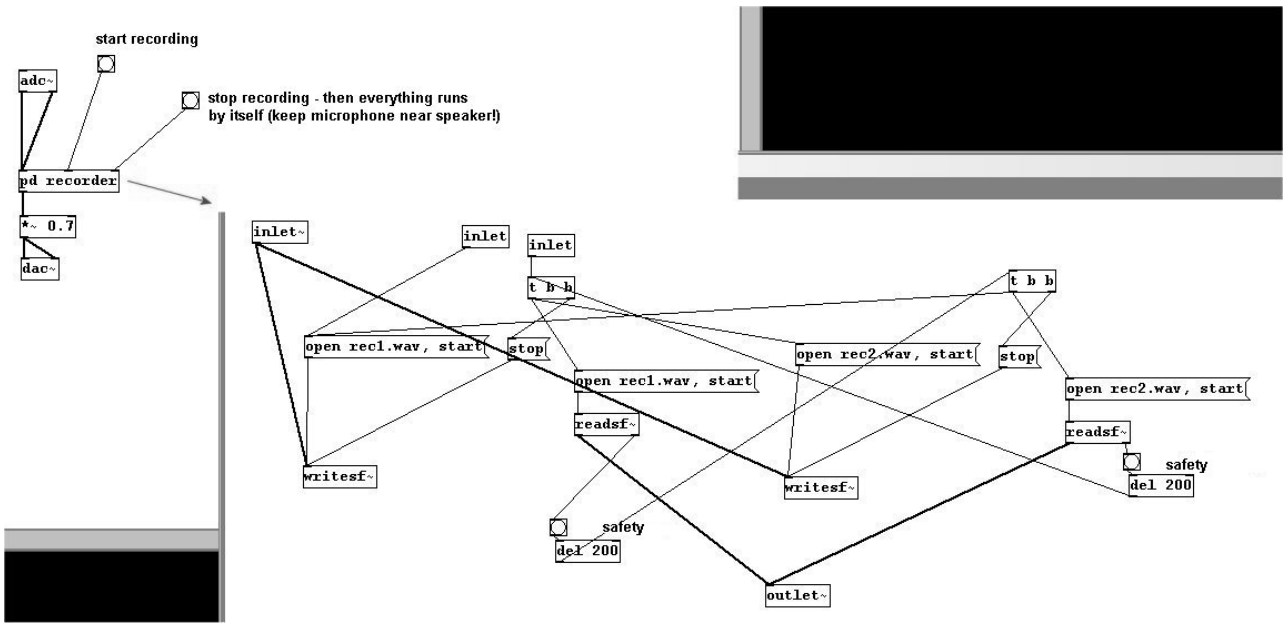
Si así lo desea, puede continuar con esto, configurando en su patch cada vez más parámetros aleatorios, hasta que quede satisfecho.

4.1.2.2 Sistemas recursivos

Existe una pieza de Alvin Lucier en una idea relativamente simple: alguien se sienta en una habitación y habla con dirección a un micrófono. Esto entrada en un momento es grabada, reproducida, regrabada, y así sucesivamente. En cada ciclo, la calidad de la grabación se va deteriorando, cada vez se pierde más información. Más específicamente, las frecuencias que el altavoz, el micrófono y la habitación pueden representar correctamente son propagadas, mientras que las otras resultan gradualmente filtradas.

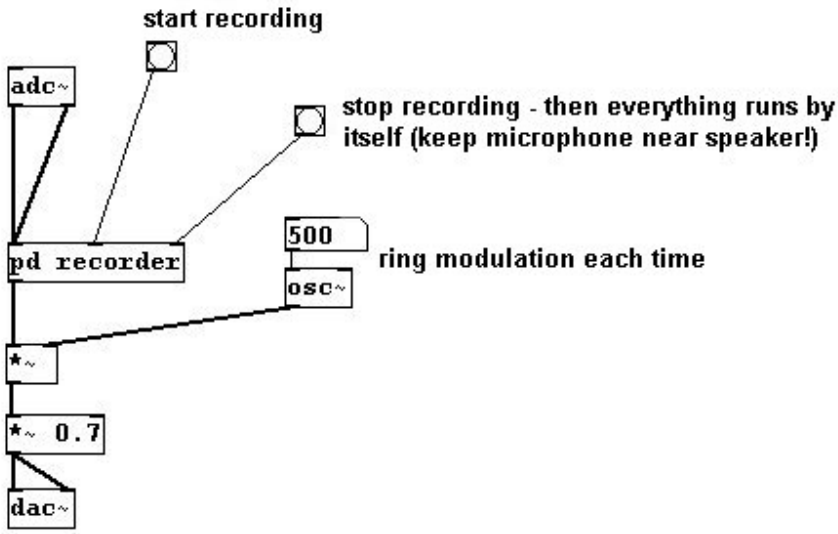
Esto puede ser fácilmente programado en Pd:

patches/4-1-2-2-lucier.pd



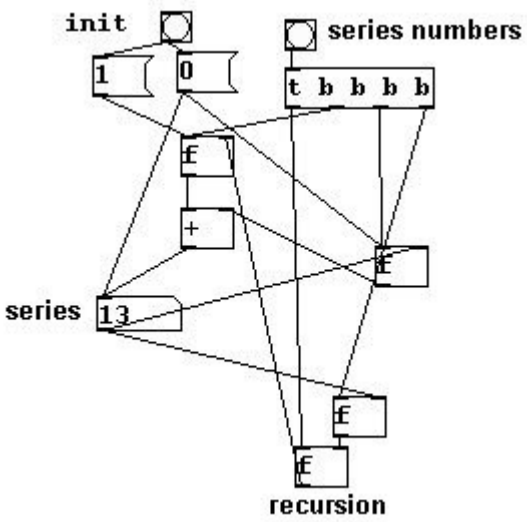
Podríamos decir que el algoritmo consiste en grabación y reproducción. En este caso, el resultado es retroalimentado en el algoritmo una y otra vez. Un proceso que funciona automáticamente de esta manera es llamada una *recursión*. Las recursiones ya han sido mencionadas en 3.4.2.9 y 3.4.2.10.

En el próximo ejemplo, también usaremos altreración y regrabación. Una modulación en anillo recursiva:



Las recursiones que trabajan puramente con números también pueden resultar interesantes. Uno de los más conocidos ejemplo de esto que se da frecuentemente en la música es la serie de Fibonacci. El algoritmo consiste en que los últimos dos números de una lista son sumados entre sí para para producir un nuevo resultado final en la lista.

patches/4-1-2-3-fibonacci.pd



4.1.2.3 Más ejercicios

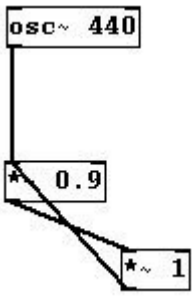
a) Grabe una muestra y reproduzcala a una velocidad erronea. Grabe esta reproducción 'erronea', reproduzcala, grabe y así sucesivamente. Pruebe esto mientras (1) reproduce la muestra con la misma velocidad 'erronea' y (2) con una velocidad 'erronea' diferente.

b) Cree un algoritmo de modelado de onda recursivo utilizando retardo (es decir, un algoritmo en el cual la salida sea retroalimentada como entrada.

4.1.3 Apéndice

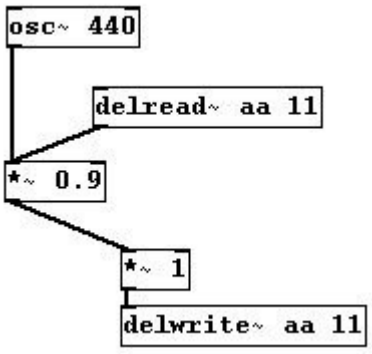
4.1.3.1 Bucle de DSP

Los algoritmos recursivos usados para distorcionar el sonido tienen limitaciones técnicas. Si hace lo siguiente...



...aparece en mensaje de error "DSP loop detected" y el audio no será enviado nuevamente al bucle. Sin usar un retardo temporal en la señal, no puede crear ninguna recursión (de audio).

Aquí puede ver cómo evitar errores:



4.1.4 Para aquellos especialmente interesados

4.1.4.1 Composición algorítmica

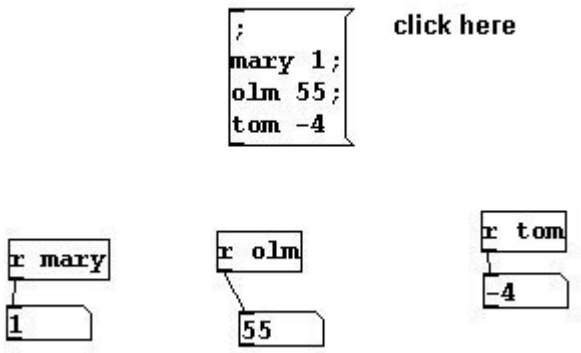
El uso de algoritmos en la composición musical es un campo muy amplio. Principios algorítmicos pueden ser observados en el trabajo de compositores medievales y ha sido un área ampliamente utilizada en la música desde el siglo XX en adelante. Las composiciones algorítmicas pueden ser fascinantes incluso desde una perspectiva púramente matemática. La naturaleza es rica en ejemplo de algoritmos. Para más información:

http://en.wikipedia.org/wiki/Algorithmic_composition

4.2 Sequencer

4.2.1 Teoría

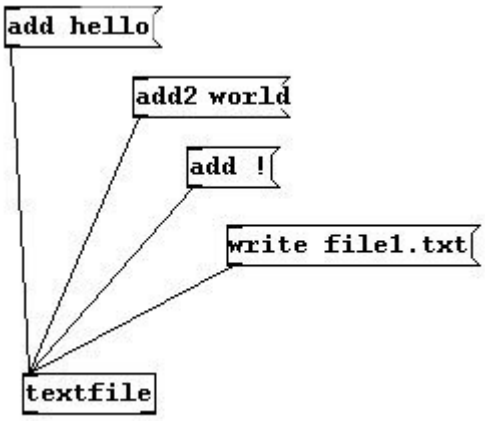
En lugar de procesos automáticos, también podemos escribir verdaderas "partituras" en un patch de Pd. Un ejemplo simple sería el uso de muchos comandos "send", como los descritos en 2.2.4.1.3:



Pero para poder incluir mucha más información y determinar la secuencia cronológica, la siguiente sección cubrirá las formas de realizar 'partituras' en Pd.

4.2.1.1 Archivo de texto

Puede recuperar números y símbolos de un archivo de texto normal o, a la inversa, guardar números y símbolos usando "textfile". Primero veamos la función de almacenamiento. Haga clic en los Mensajes de arriba hacia abajo:

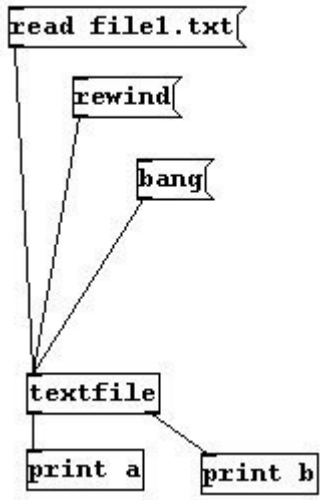


Ahora Pd ha creado un archivo de texto llamado "file1.txt" en el mismo directorio que el patch. El mismo contiene:

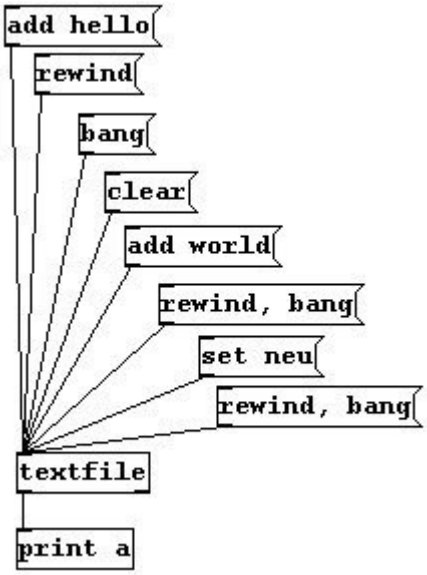
```
hello;
world!;
```

"add" crea un símbolo o un número y lo cierra con un punto y coma. "add2" no cierra con un punto y coma.

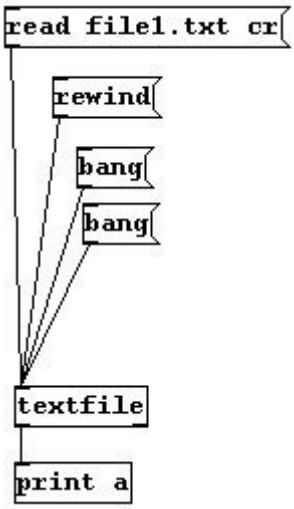
Si quiere leer lo que salvó, cargue el archivo y use "rewind" para comenzar desde el inicio. Ahora, cada vez que pulse 'bang', se enviará una línea (hasta el punto y coma) a la salida izquierda. Luego de la última línea, se envía un bang a la salida derecha.



También puede escribir algo y leerlo con el Objeto sin siquiera salvar un sólo archivo. También puede usar "clear" para borrar todo el contenido. "set" primero borra todo el contenido y luego comienza una nueva línea. Haga clic desde arriba hacia abajo:



También puede cargar un archivo de forma que no aparezcan los punto y coma:

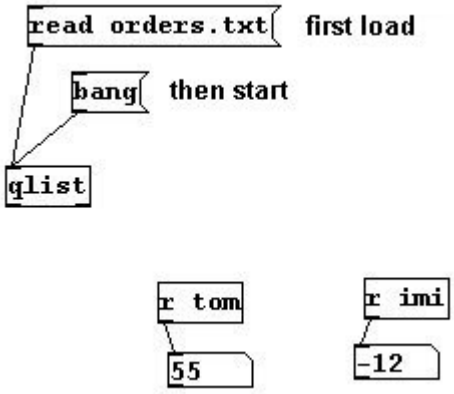


"write name.txt cr" también trabaja de la misma manera.

4.2.1.2 Qlist

El Objeto "qlist" puede entenderse como una expansión muy práctica a "textfile". Puede ser usado para leer Mensajes ordenados cronológicamente y enviarlos a Objetos "receive". El archivo "orders.txt" tiene estos contenidos:

```
0 tom 55;
1000 imi -12;
4000 tom 3;
2000 imi -2;
```

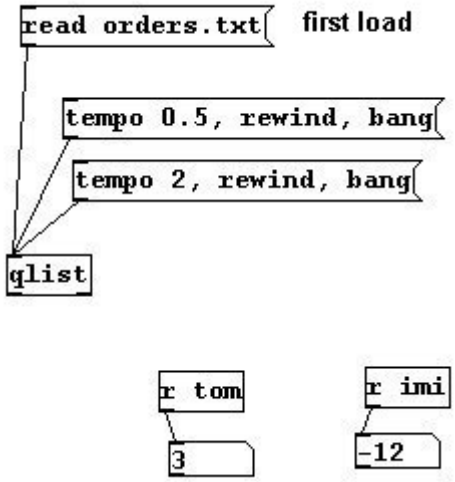


Am principio "tom" recibe el número 55; un segundo más tarde, "imi" recibe -12; cuatro segundos más tarde, "tom" recibe 3; dos segundos después, "imi" recibe -2. Trabaja de la misma manera con símbolos

At the beginning, "tom" receives the number 55; one second later, "imi" receives -12; four seconds later "tom" receives 3; two seconds later "imi" receives -2. It works the same way with symbols.

Por lo demás, "qlist" tiene las mismas funciones que "textfile": add, add2, rewind, clear.

También puede modificar el tempo usando "tempo" y un factor:

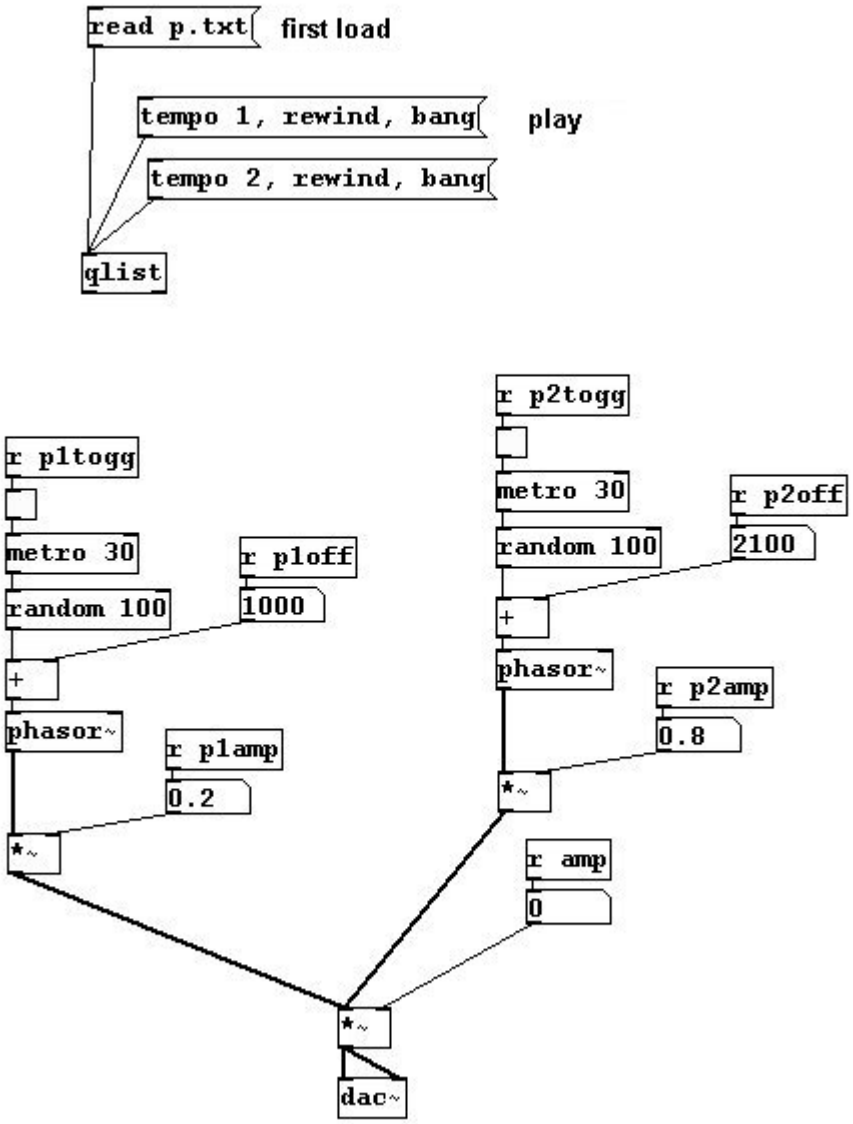


4.2.2 Aplicaciones

4.2.2.1 Partitura para un patch

Una vez que haya armado los generadores de sonidos, puede escribir una pieza de música como un archivo de texto. Digamos que tiene este patch...

patches/4-2-2-1-score.pd

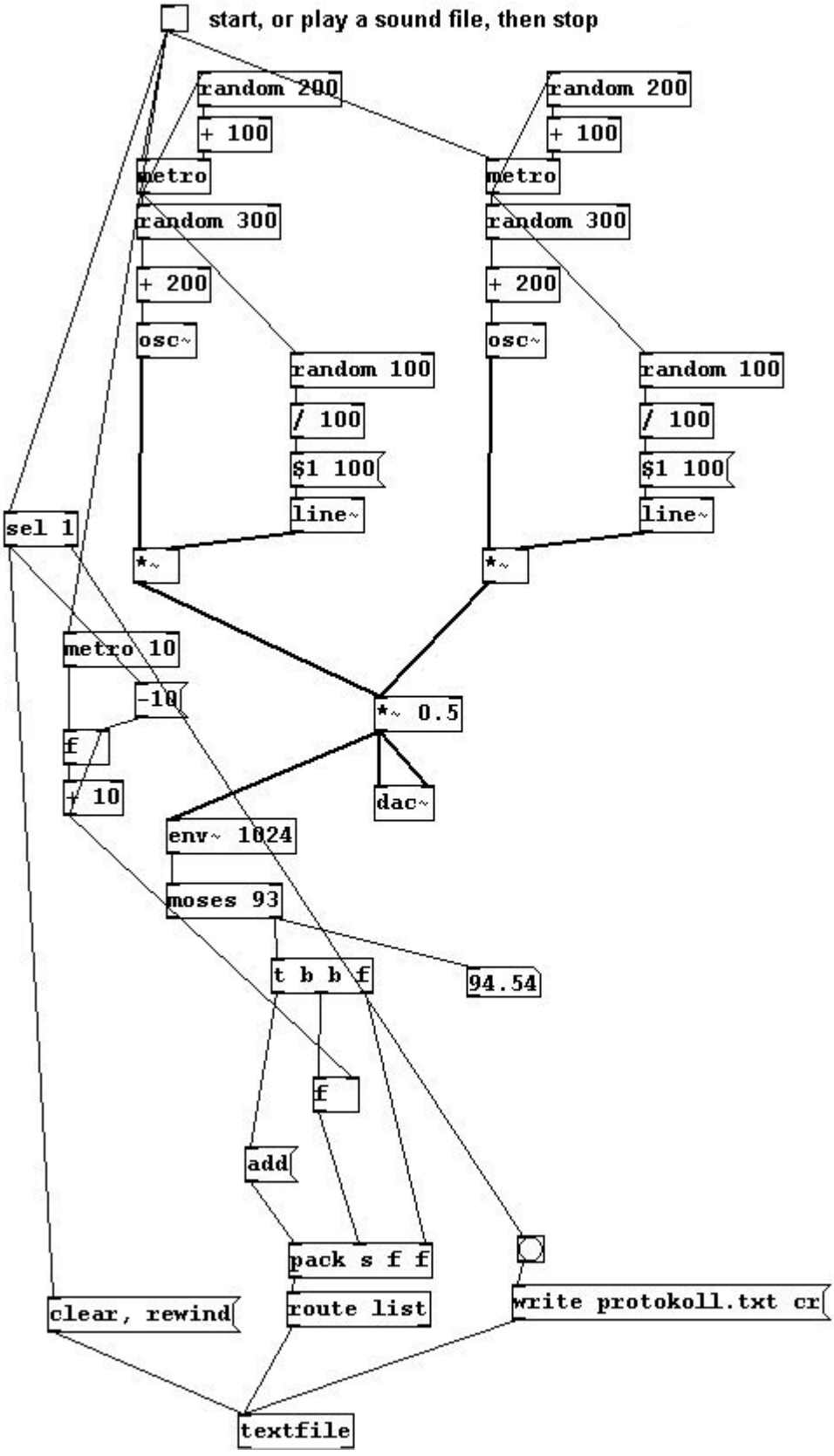


...y esta "partitura" (patches/p.txt):

```
0 p1off 1000;
0 p1togg 1;
0 p1amp 1;
0 amp 0.5;

3000 p2off 100;
0 p2togg 1;
0 p2amp 1;

2000 p2off 400;

3000 p1amp 0.2;

3000 p2amp 0;

1000 p2off 2100;
0 p2amp 0.8;


5000 amp 0;
```

```
0 p1togg 0;
0 p2togg 0;
```

También podría escribir información tomando como fuente a los sonidos mismos:
patches/4-2-2-1-write-score.pd



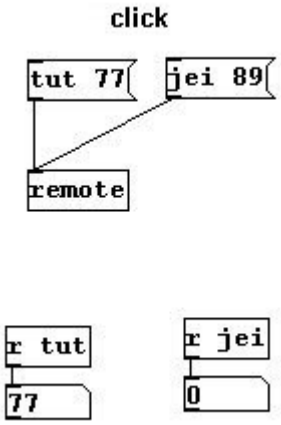
4.2.2.2 Más ejercicios

Escriba algoritmos estocásticos dentro de un archivo de texto que use un "qlist" para reproducir el patch de 4.2.2.1 a diferentes velocidades.

4.2.3 Apéndice

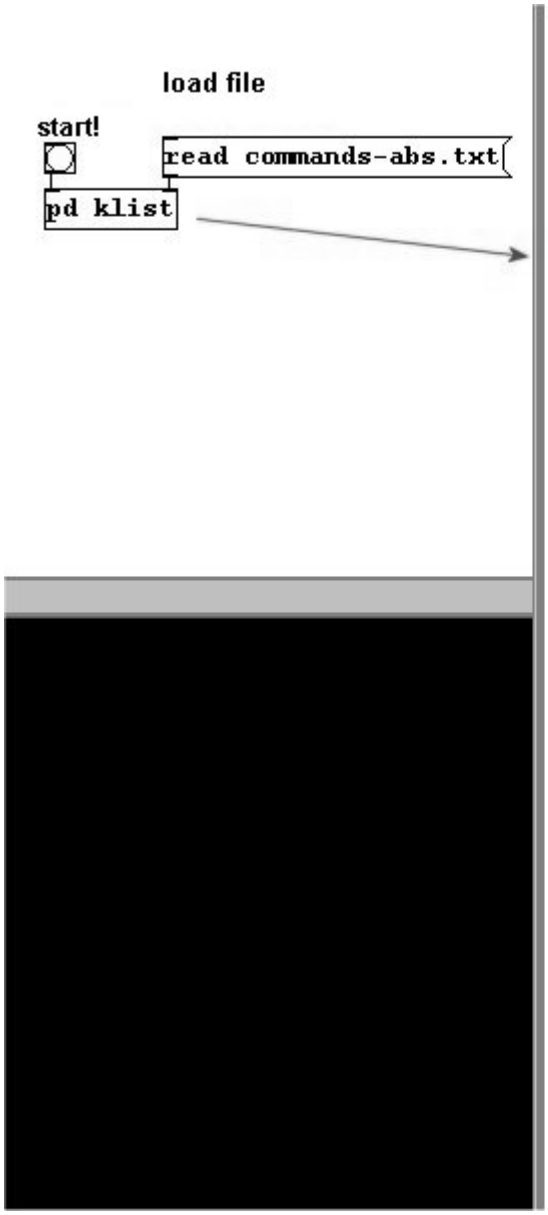
4.2.3.1 Modificando qlist

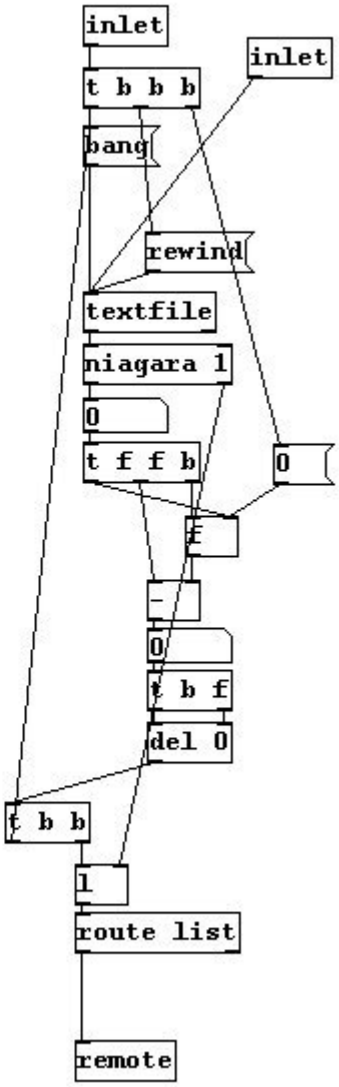
El tiempo para qlist se expresa en valores delta, esto es, siempre se describe el intervalo de tiempo de un evento al siguiente. En ocaciones, en cambio, podría resultar más práctico escribir el archivo de texto con intervalos de tiempo absolutos. Para lograr esto podemos usar el Objeto "remote" (disponible en Pd-extended). Este recibe el nombre de un Objeto "receive" como lista seguido por el valor que se desea inviar. Esto nos ahorra el trabajo de tener que usar muchos "sends":



Entonces puede construir su propio qlist usando valores absolutos:

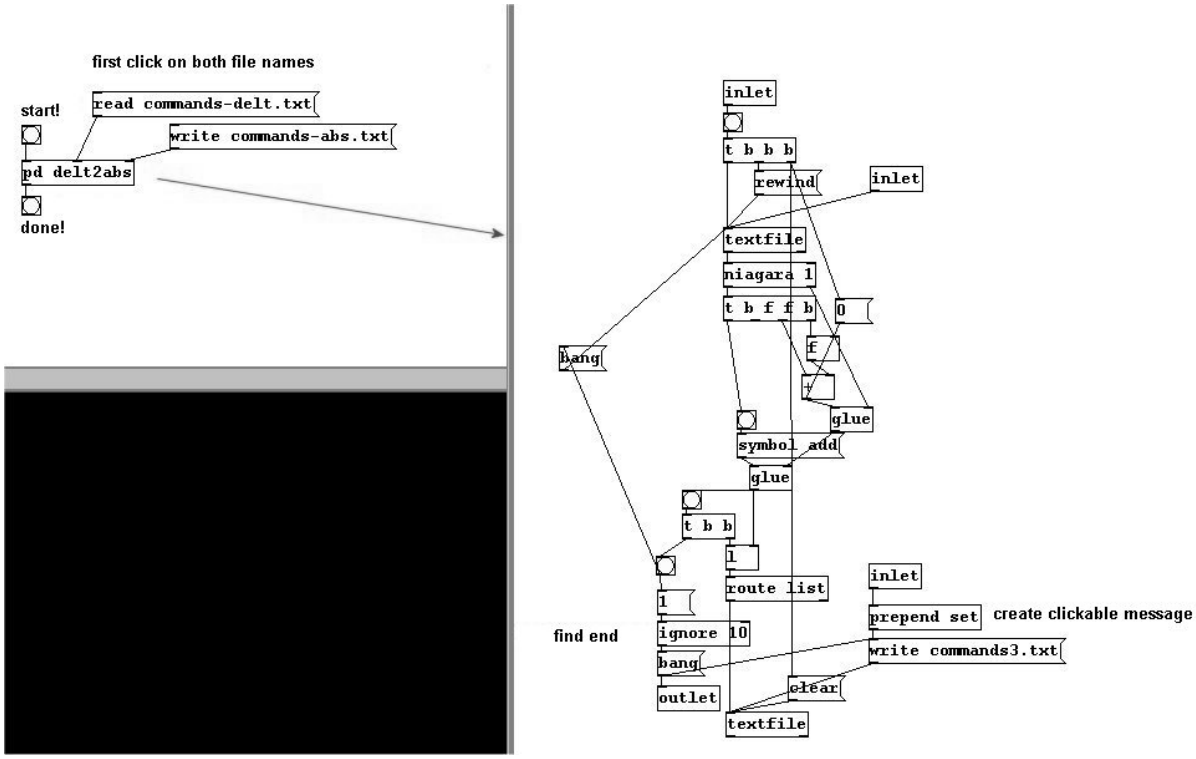
patches/4-2-3-1-klist.pd





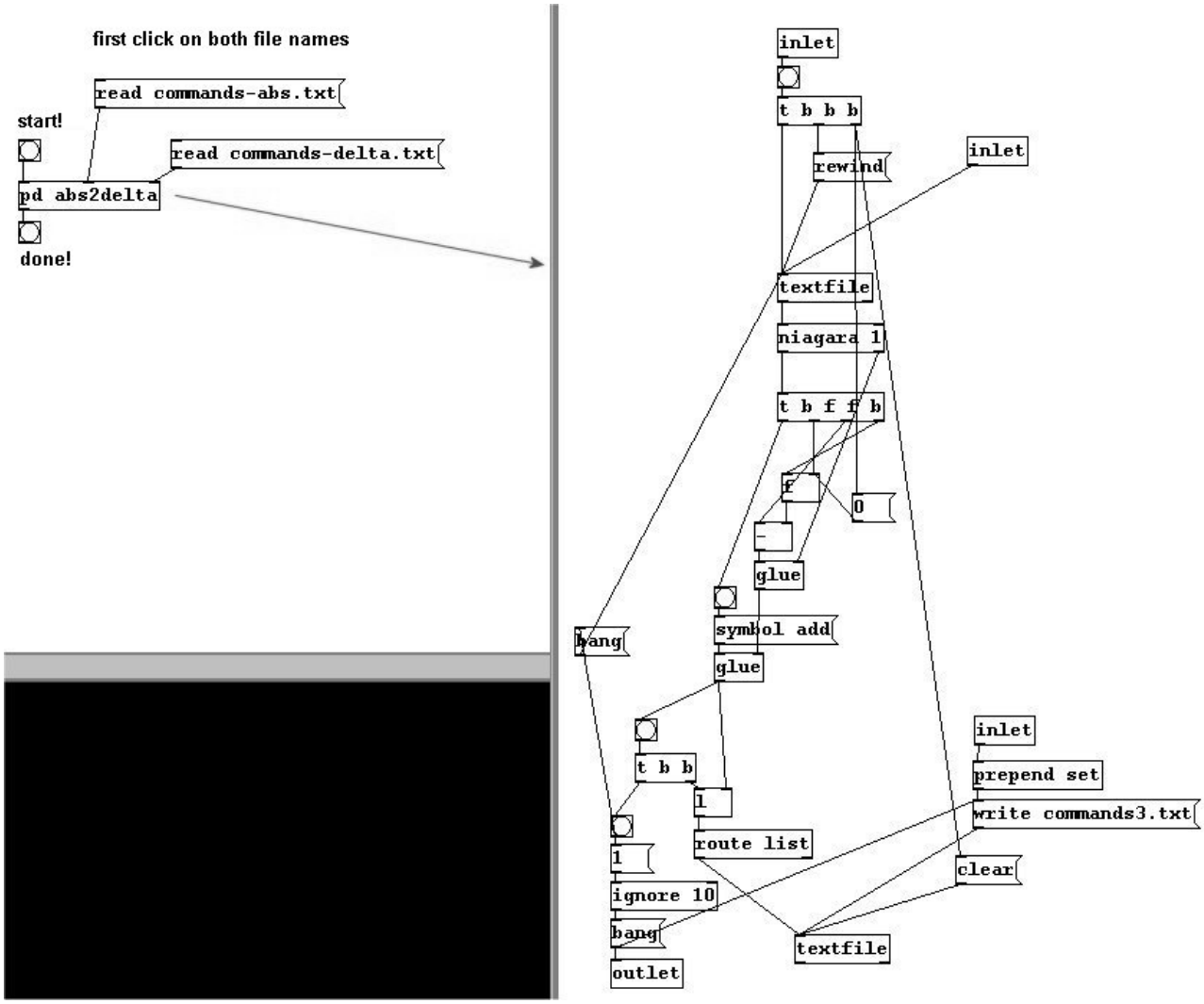
y aquí tiene cómo cambiar una lista de valores delta en valores absolutos:

patches/4-2-3-1-klist-convert1.pd



y a la inversa:

patches/4-2-3-1-klist-convert2.pd



4.2.4 Para los interesados, especialmente

4.2.4.1 Creando listas externamente: Lisp

También puede tomar "textfile" para usar archivos de texto que contengan algoritmos previamente realizados. Existen lenguajes de programación especiales que pueden llevar a cabo esto. Uno de estos lenguajes es LISP, el cual es especialmente adecuado para la creación y procesamiento de listas:

http://en.wikipedia.org/wiki/Lisp_(programming_language)

4.3 Dispositivos de Interfaz Humana (HIDs - Human Interface Devices)

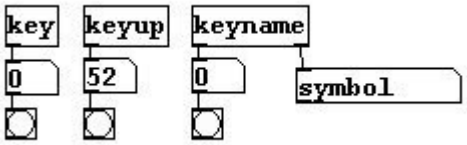
4.3.1 Teoría

Es posible ejecutar un patch en vivo, de la misma manera que puede tocar un instrumento en vivo sobre un escenario. Mientras tenga un patch abierto y ejecutándose puede hacer clic sobre los Objetos GUI, aunque el uso del ratón termina siendo muy poco práctico si quiere ser muy preciso

en relación al tiempo. Por esta razón existen Dispositivos de Interfaz Humana (HIDs – Human Inteface Devices), interfaces entre hombre y ordenador. El ratón y el teclado son técnicamente HIDs, pero existen muchos otros más, algunos de los cuales están especialmente diseñados para ser usados en el campo musical, por ejemplo para controlar un patch de Pd.

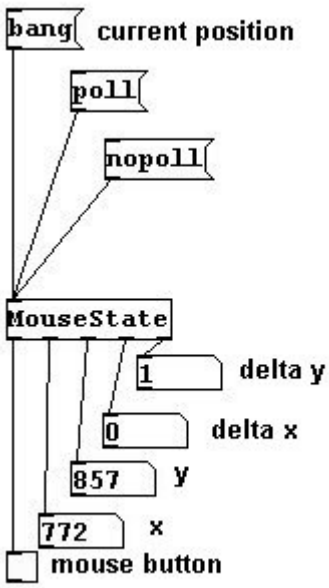
4.3.1.1 Teclado y ratón

Si hace un clic en una caja Número, puede ingresar un valor con el teclado. También podría usar el teclado para directamente transmitir otra información:



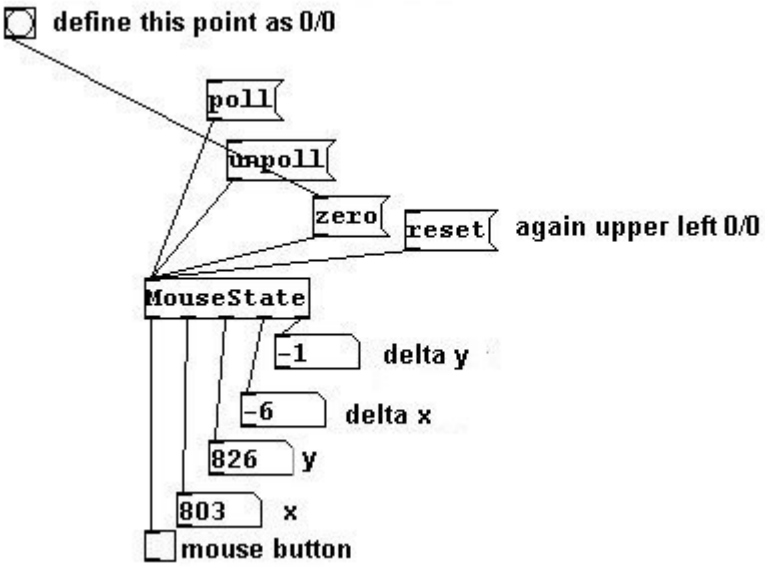
Las teclas están numeradas (con algunas excepciones, por ejemplo las teclas F1 y F12). "key" registra la tecla que ha sido presionada, "keyup" la que ha sido liberada. Siempre se envía el número de tecla. "keyname" muestra el nombre normal de la tecla.

Si usa el externo "MouseState" (¡sensible a mayúsculas y minúsculas!) de Pd-extended, también puede usar los datos del ratón:



Con "poll" y "unpoll" puede iniciar/detener la visualización (en mi versión tiene que primero hacer clic en "unpoll" y luego en "poll" para que funcione). Este Objeto muestra las coordenadas absolutas x/y, los valores delta, y si se encuentra presionado en botón izquierdo del ratón.

Normanlmente las coordenadas 0/0 aparecen en la esquina superior izquierda del monitor; con "zero" puede seleccionar otro punto como referencia:



4.3.1.2 MIDI

A principios de los 80s, grandes fabricantes de instrumentos electrónicos establecieron un protocolo de transferencia de datos standard para usarlo con un conjunto de dispositivos de entrada llamado *Interfaz Digital de Instrumentos Musicales (MIDI - Musical Instrument Digital Interface)*. Ahora existen teclados MIDI, mesas de mezcla MIDI, guantes de datos MIDI, etc. En Pd existen Objetos para recibir y enviar datos MIDI. Puede enviar estos datos a un dispositivo o un patch donde los convierte en sonido. Sin embargo, la mayoría de los ordenadores contiene también sonidos MIDI, por lo tanto los datos MIDI pueden ser también convertidos a sonido allí.

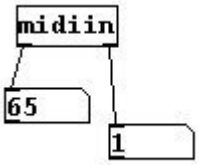
El protocolo MIDI mismo no contiene ningún sonido, pero comprende comandos para el control del patch o de otros instrumentos, por ejemplo, "note-on", "velocity", y "note-off". Además de estos comandos básicos, MIDI usa otros comandos más especializados que pueden ser usados para, digamos, cargar otros sonidos o modificar sonidos cargados usando datos de control producidos mediante interruptores, botones o potenciómetros.

Cada comando MIDI standard (excepto el de datos de sistema exclusivo, abreviado como SysEx) conlleva un número de canal además de su comando de identificación y su comando de datos. El número de canal tiene un tamaño de 4 bits, lo que implica que 2^4=16 canales pueden ser controlados. Dependiendo del software los canales spueden ser numerados del 0 al 15 ó del 1 al 16, aunque el último es el más común.

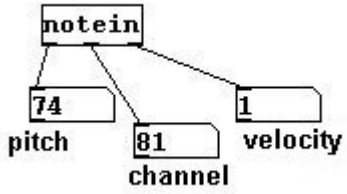
Ya que MIDI es un protocolo serializado y la velocidad de transmisión de datos de las interfaces MIDI es bastante baja en comparación con los estándares de hoy en día, pueden surgir problemas de sincronización con frecuencia cuando muchas notas son tocadas a la vez, especialmente cuando se utiliza junto a un programa de secuanciación. Incluso tocar un acorde con muchas notas puede conducir a un retardo audible porque MIDI nunca puede enviar notas simultáneas, sólo una a la vez.

Para los siguientes ejemplos es necesario contar con equipamiento físico MIDI. En Pd puede configurar estos dispositivos bajo **Media** > **MIDI settings**.

El Objeto más básico es "midiin". Cada ingreso MIDI es presentado aquí, el *valor* en la salida izquierda y el *número de canal* en la derecha.

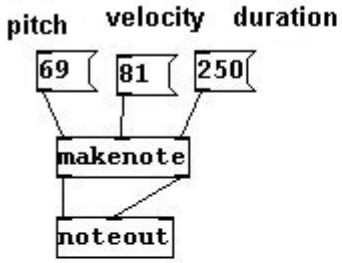


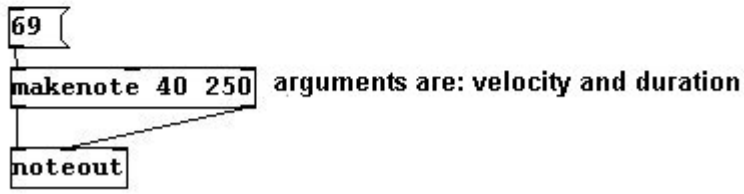
Si dispone de un teclado MIDI u otro dispositivo de entrada con alturas definidas, con "notein" obtiene los siguientes valores:



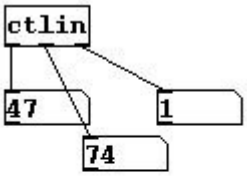
El número MIDI para la *altura* aparece en la salida izquierda, la *velocidad* (fuerza de ataque) en la central, y el número de *canal* en la derecha.

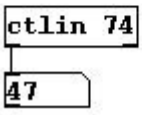
A la inversa, podría enviar esta información a un instrumento. Si un sólo instrumento se encuentra conectado, no tiene que ingresar un número de canal. Necesitará usar "noteout" y luego "makenote", el último de los cuales combina las entradas de una manera similar al Objeto "pack":





También existen valores de *cambio de control*, los cuales son ingresados con "ctlin" y "ctlout". Veamos "ctlin":



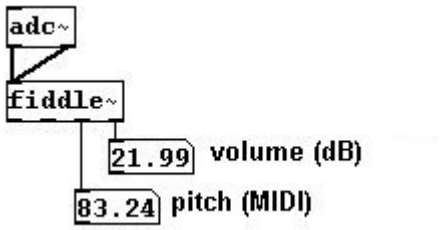


La salida izquierda corresponde al *valor*, la central al número de *control*, y en la derecha el *canal*. También podría ingresar directamente el número de control como argumento.

Todos los demás transmisores y receptores MIDI funcionan así también. Entre ellos están "pgmin", "bendin", "touchin", y "sysexin".

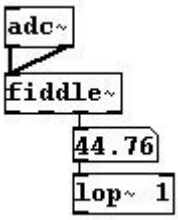
4.3.1.3 Usar señales para controlar un sonido

La entrada sonora recibida a través de un micrófono puede ser no solamente usada estríctamente como sonido sino también como control de datos. Como aprendió en 3.8.3.1, puede usar "fiddle~" para determinar información relacionada a la amplitud y frecuencia:



Estos números –nuevamente, Pd trabaja sólo con números– pueden ser usados en conjunción con parámetros en un patch.

Un problema que se presenta aquí es el relacionada con que los datos que se obtienen con el Objeto "fiddle~" suelen ser muy caóticos. Existe un truco que se puede usar para filtrarlo:



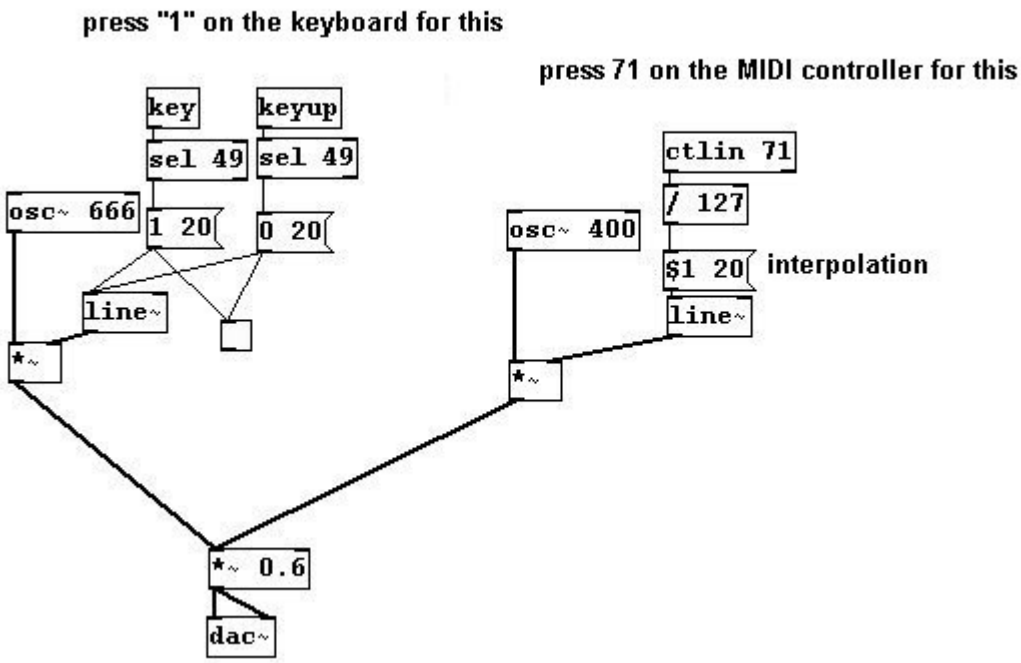
Un filtro pasa bajos puede recibir también un ingreso de control de datos. En este caso, sólo cambios relativamente lentos pueden pasar.

4.3.2 Aplicaciones

4.3.2.1 Ejecutando patches en vivo

Usando los dispositivos de entrada y métodos descritos anteriormente, podemos cambiar externamente los parámetros en un patch:

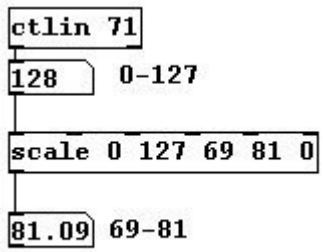
patches/4-3-2-1-patch-play.pd



Queda claro que diferentes dispositivos poseen diferentes funciones: un botón representa encendido o apagado, un potenciómetro significa cambios graduales.

Para controladores que manejan un rango de números, como potenciómetros o faders, se recomienda el uso de interpolación (especialmente porque en MIDI sólo obtienen 128 valores).

En Pd-extended (pero sólo si GEM no se carga en memoria) existe un externo muy útil que puede usar para este caso: "scale". Por ejemplo, si quiere obtener alturas entre 69 y 81 (números MIDI), y quiere usar un controlador MIDI para esto (el cual genera números de 0 a 127) podría escribir:



El último argumento significa: lineal (0) o exponencial (1).

4.3.2.2 Más ejercicios

a) Use sonidos MIDI externos en vez de osciladores para los algoritmos en 4.1.2.1.

b) Use parametros de cualquier patch del Capítulo con dispositivos de entrada.

4.3.3 Apéndice

4.3.3.1 Otros HIDs

Además del teclado normal, ratón, y dispositivos MIDI, el número de otros dispositivos de entrada continúa creciendo: desde joysticks para juegos de ordenador, hasta tabletas para dibujo, o sensores

de movimiento. Aunque hasta el día de hoy (Junio de 2009) no existe en Pd un sólo Objeto que trabaje con todos estos dispositivos a la vez, debemos mencionar los siguientes externos: "joystick", el cual puede recibir información de joysticks; "wintablet" para tabletas Wacon en Windows; y "hid" en Linux y MacOSX, el cual trabaja con muchos dispositivos de entrada diferentes. También existe hace un tiempo un dispositivo llamado placa Arduino la cual digitaliza la información que proviene de instrumentos analógicos que se encuentran conectados a la misma. Con software adicional, esta información también puede ser recibida por Pd.

4.3.3.2 Entrada de video

También encontramos en Pd un componente de video llamado GEM, el cual puede ser usado para extraer números de una señal de video –grabada previamente o en vivo– los cuales pueden ser usados para controlar parámetros sonoros.

4.3.4 Para los interesados, especialmente

4.3.4.1 Diseño de instrumento

También he dado una conferencia (en alemán) en el *Serie Conferencias Lagerhaus* en Freiburg, Alemania, sobre el arte de usar dispositivos de entrada externos en una composición. Puede ser vista online en la siguiente dirección:

http://www.kreidler-net.de/theorie/instrument-design.htm

4.4 Redes

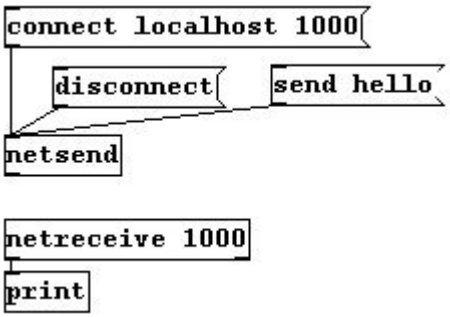
La siguiente sección trata sobre la creación de redes entre varios ordenadores que no sólo son usados para el procesamiento de sonido sino también para establecer una comunicación entre otras personas con la finalidad de ejecutar una pieza musical de manera colaborativa.

4.4.1 Netsend / Netreceive

Un patch de Pd en un ordenador puede intercambiar datos con un patch de Pd que se encuentre en otro ordenador.

Primero conecta los ordenadores –los cuales tienen instalado Pd– entre sí mediante un cable de red. Use "netsend" para conectar un ordenador al otro. Ingrese el Mensaje "connect [nombre] [número de puerto]"; en vez del nombre, podría ingresar la *dirección IP* del otro ordenador. "disconnect" finaliza la conexión.

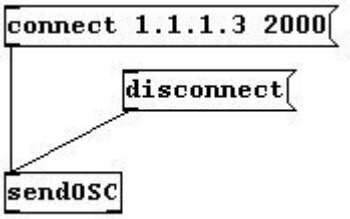
Una vez conectado, use "send" seguido de los símbolos que quiera enviar, para comenzar a dirigir los mensajes al otro ordenador. Este recibe estos datos usando "netreceive [número de puerto]".



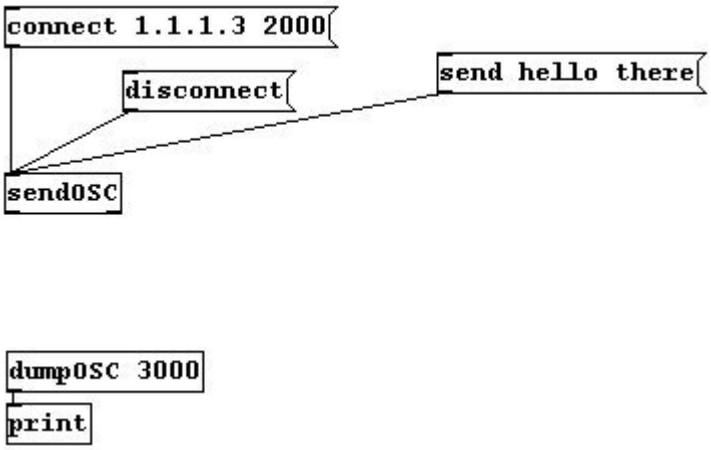
4.4.2 OSC

En Pd-extended, también disponemos de los Objetos OSC. Estos son usados por muchos otras aplicaciones de ordenador para el intercambio de datos. OSC es la sigla correspondiente a *OpenSound Control*. Trabaja de manera similar a "netsend" y "netreceive".

Tiene que estar conectado a otro ordenador mediante un cable de red y ambas deben estar ejecutando OSC. Use "sendOSC" (¡sensible a mayúsculas y minúsculas!) para establecer la conexión al otro ordenador. Ingrese en Mensaje "connect" como así también la *dirección IP* y el *número de puerto* del otro ordenador. Finalice la conexión usando "disconnect".



Una vez establecida la conexión use "send" seguido de los símbolos que quiera enviar para iniciar el envío de Mensajes al otro ordenador (el cual también debe estar ejecutando OSC). El otro ordenador recibe los datos usando "dumpOSC" con el *número de puerto* del transmisor como argumento.



Capítulo 5. Misceláneos

Tabla de contenidos

5.1 Racionalización eficiente

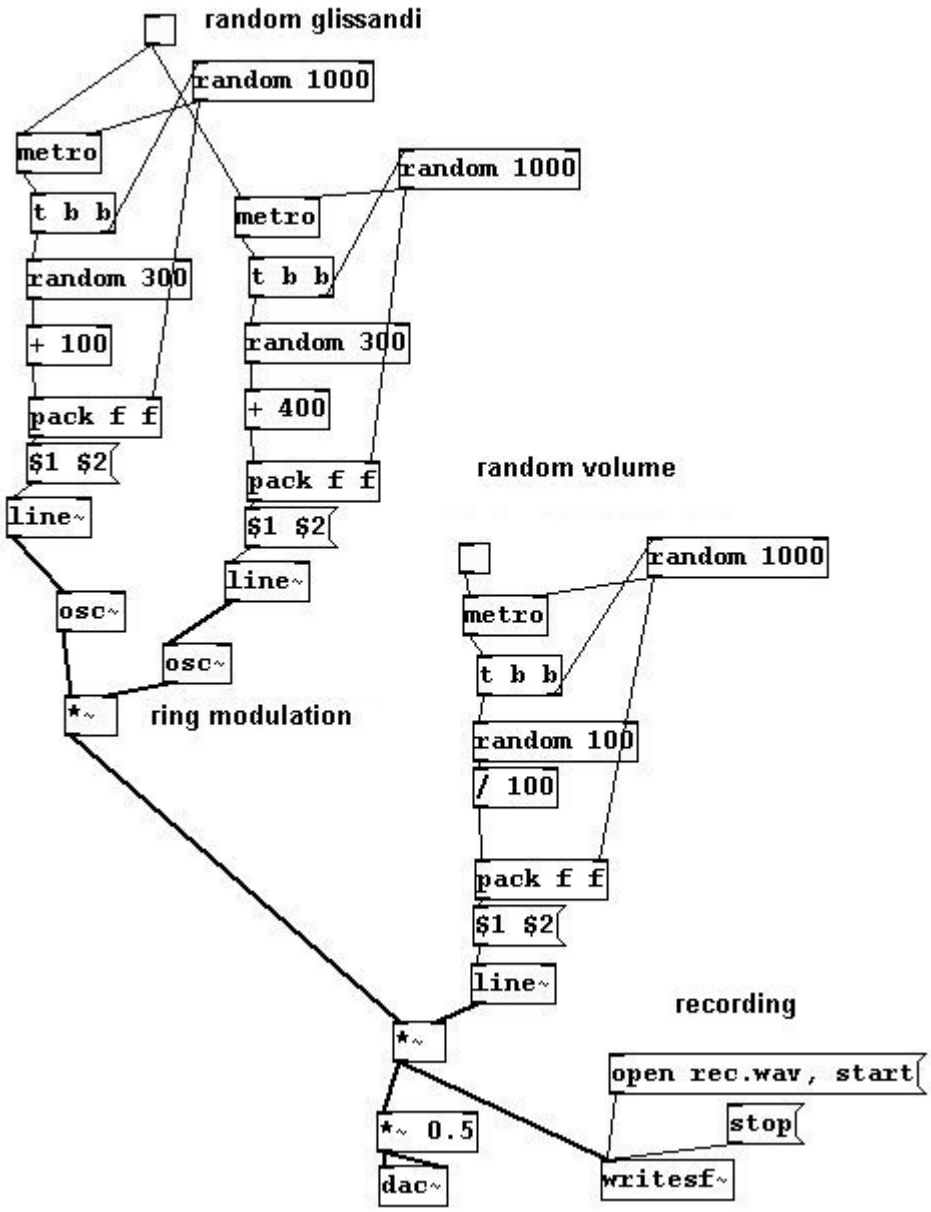
5.1.1 Teoría

5.1.1.1 Subpatches

Usted ya aprendió cómo crear subpatches en Pd en 2.2.4.4 . Ahora aprenderá cómo usarlos eficientemente.

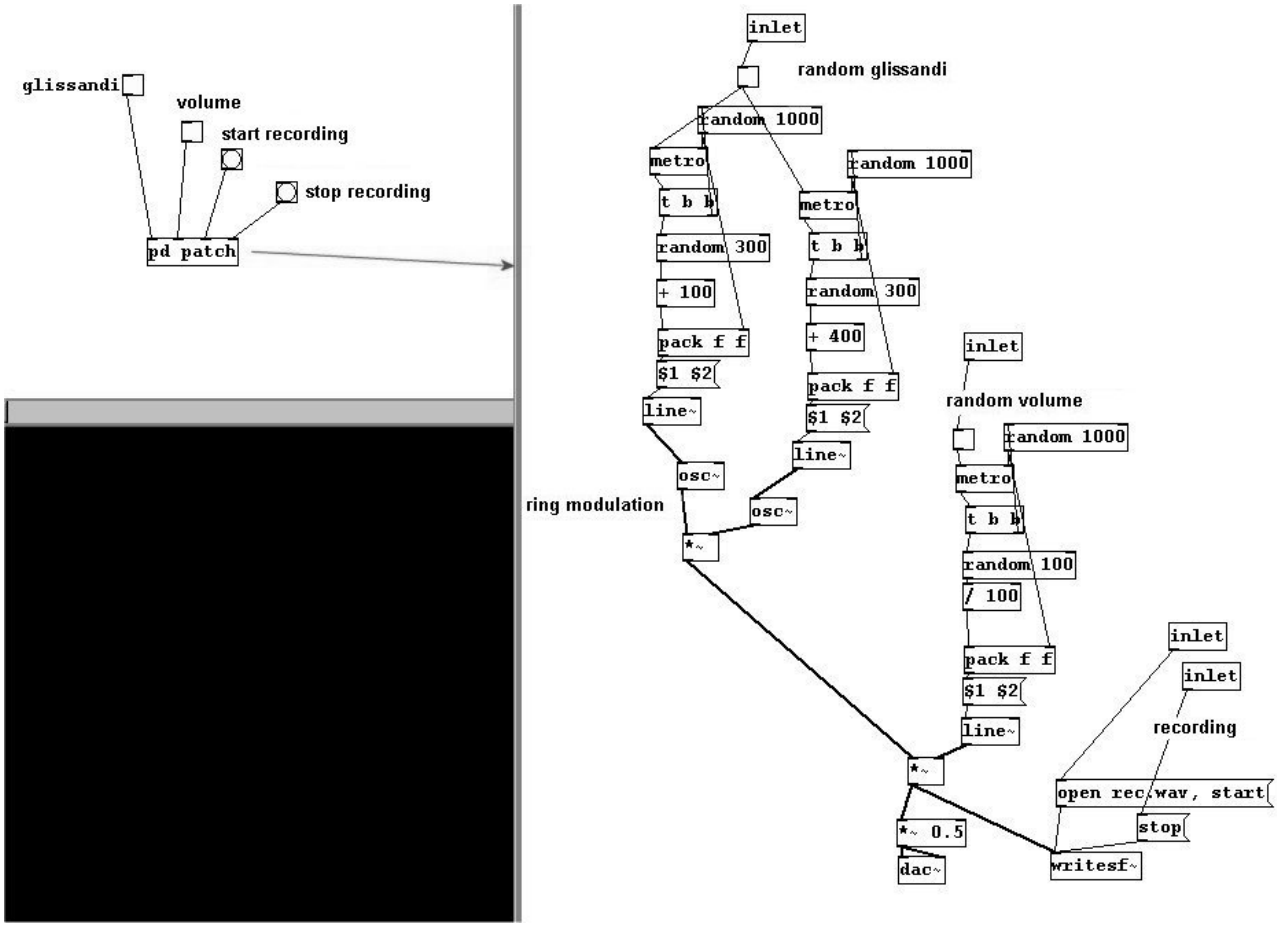
Digamos que tiene este patch:

patches/5-1-1-1-subpatch1.pd



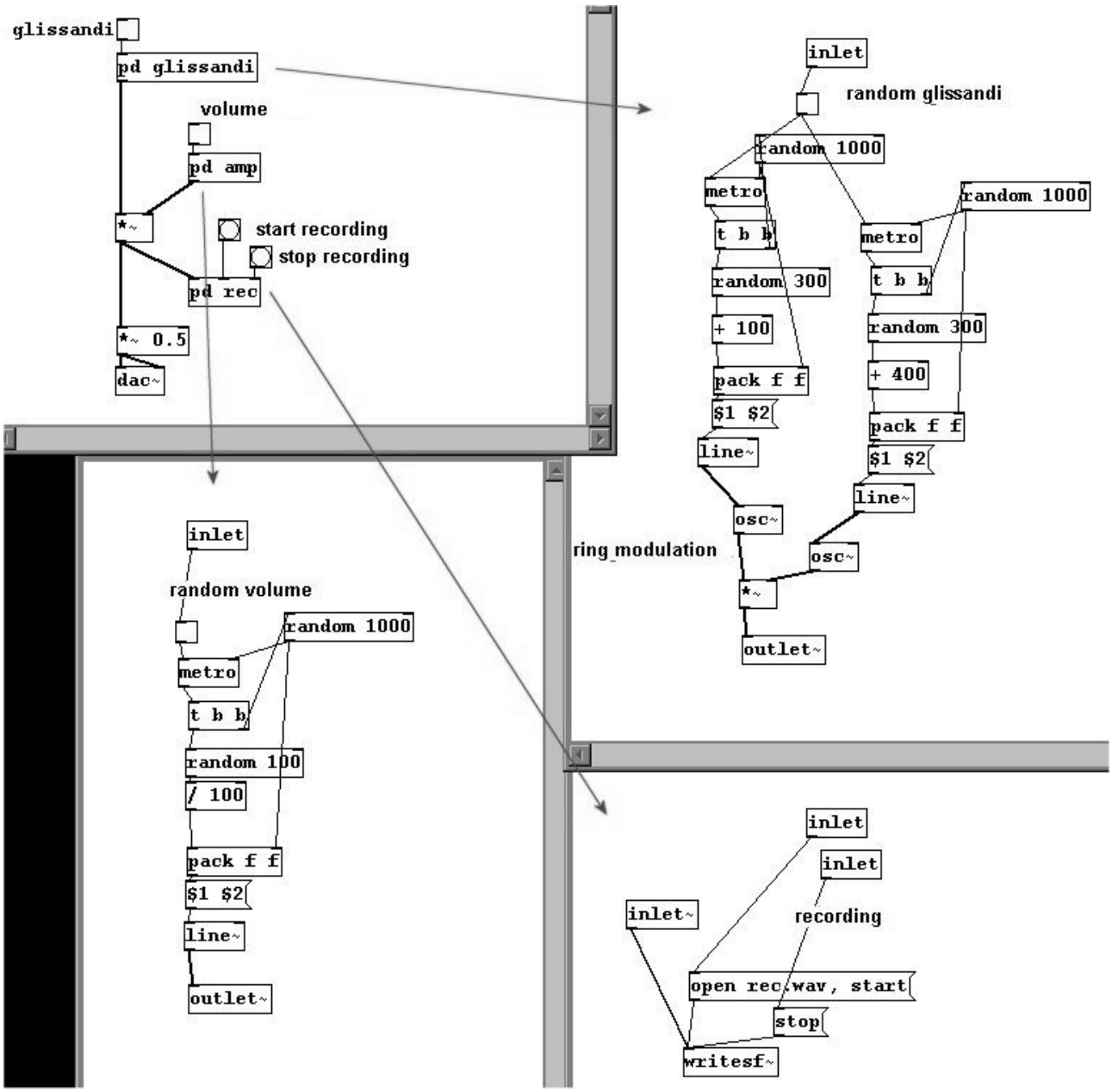
Podría arreglar el patch guardando todo lo que no necesita un acceso inmediato dentro de un subpatch:

patches/5-1-1-1-subpatch2.pd



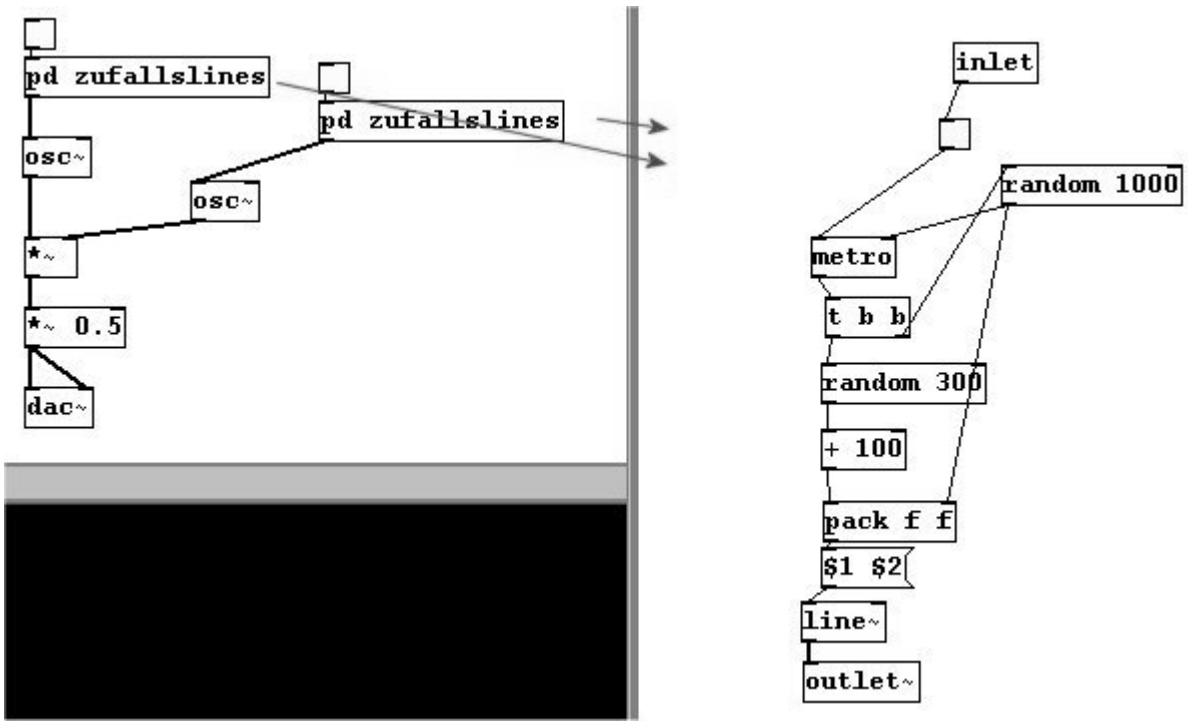
También tiene sentido guardar los contenidos en varios subpatches en caso de querer editar más tarde una parte específica.

patches/5-1-1-1-subpatch3.pd



Puede construir algún algoritmo (aquí: lineas aleatorias) para un patch que podría ser usado en diferentes lugares:

patches/5-1-1-1-module.pd

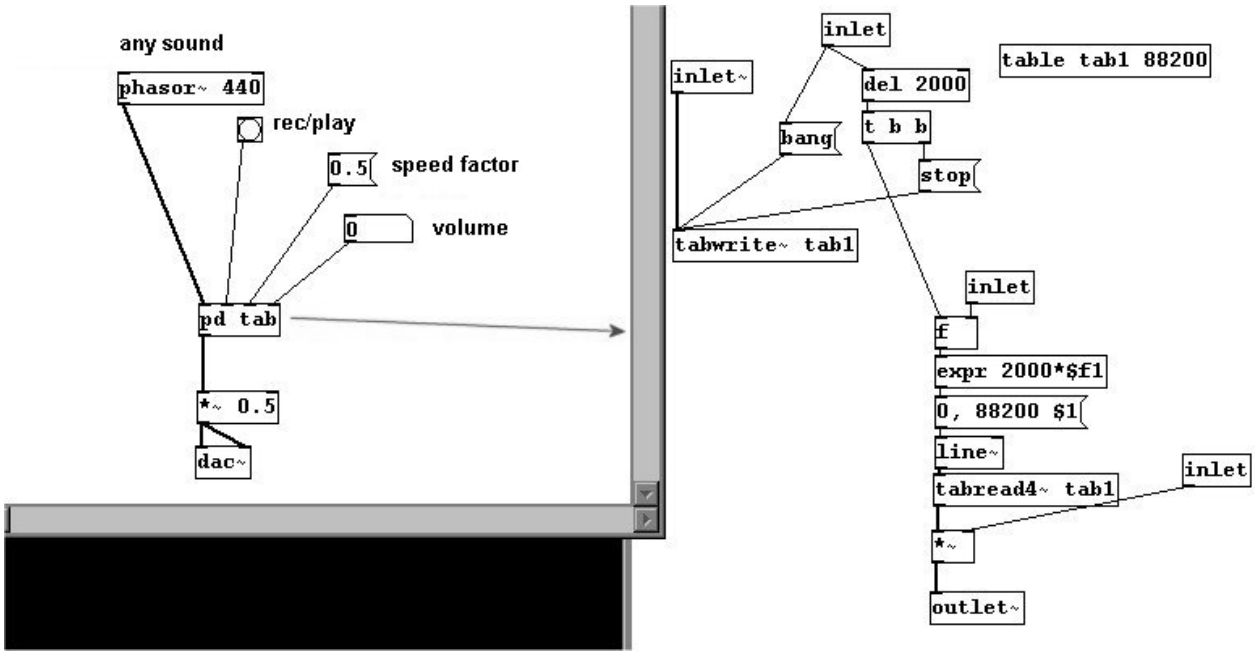


Un subpatch como este es llamado un "módulo".

5.1.1.2 Abstracciones

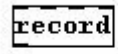
Si tiene un subpatch el cual puede ser usado universalmente, como el que sigue, el cual graba dos segundos de audio en una matriz para luego reproducirlo a una cierta velocidad con amplitud variable:

patches/5-1-1-2-abstraction1.pd



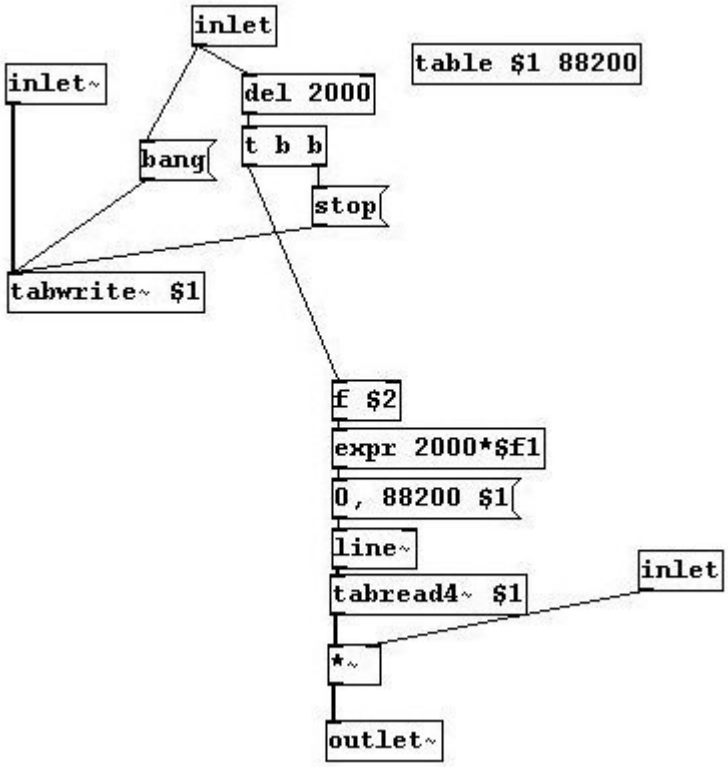
...y quiere usarlo en diferentes lugares, es entonces que surge un problema: una vez que el patch haya sido duplicado, debe renombrar la matriz e ingresar una nueva entrada (para la velocidad). En vez de todo esto podría hacer una *abstracción*, es decir un patch que es guardado como un archivo separado y puede trabajar con múltiples variables.

Tome el subpatch anterior y guardelo como "record.pd". Luego abra un nuevo patch y guardelo como "main.pd" en el mismo directorio en donde salvó "record.pd". Después de esto cree un objeto llamado "record":

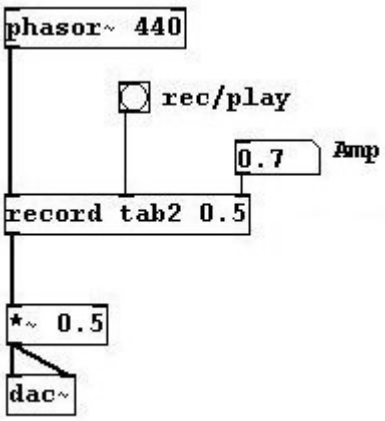


"record" no existe realmente como Objeto en Pd. Pero si un patch con dicho nombre (más el sufijo ".pd") existe en el mismo directorio, se crea un Objeto que trata al patch como si fuera un subpatch.

Una ventaja del uso de las *abstracciones* reside en el manejo de las variables. Vuelva a cargar el patch "record.pd" (**File** > **Open**). Puede escribir variables dentro de los Objetos en la forma "$1", "$2", etc. Definamos las variables para el *nombre de la matriz* y la *velocidad*:

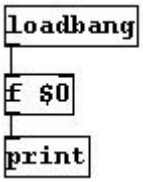


Ahora puede escribir nuevamente el Objeto "record" en el patch "main.pd", esta vez con dos argumentos para las dos variables anteriormente ingresadas; el primer argumento corresponde al *nombre de la matriz* y el segundo a la *velocidad de reproducción*.

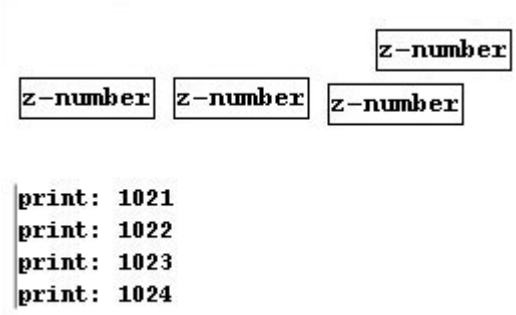


El volumen, el cual queremos modificar también desde el patch principal, está configurado como entrada tal como en un subpatch.

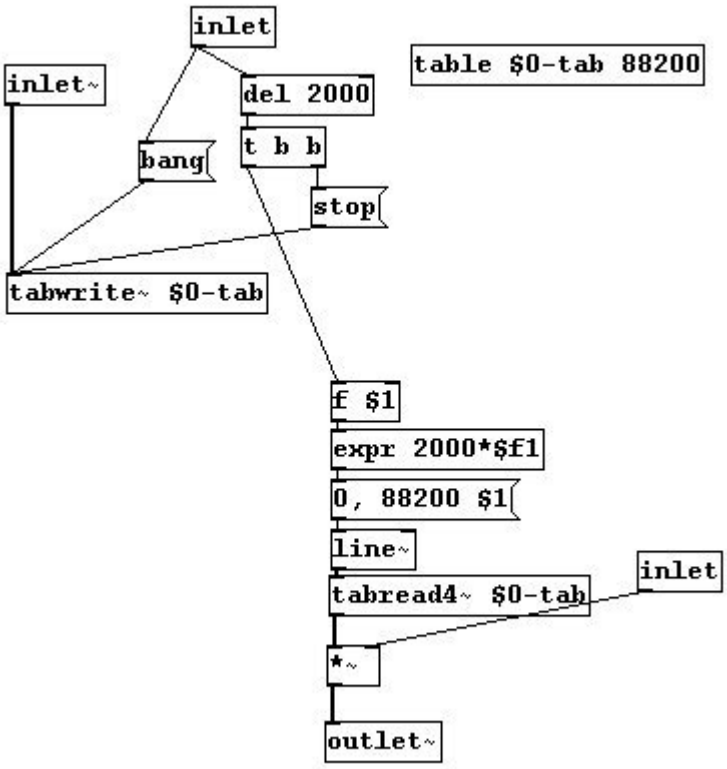
Pero todavía tenemos el problema por el cual la matriz tiene que ser definida con un nombre único cada vez que llamamos a "record" en el patch principal. Existe una opción especial en Pd para resolver esto que genera un número aleatorio individual el cual puede ser usado como nombre para un patch determinado. Esto es generado por $0. Podría realizar la abstración "z-number.pd":



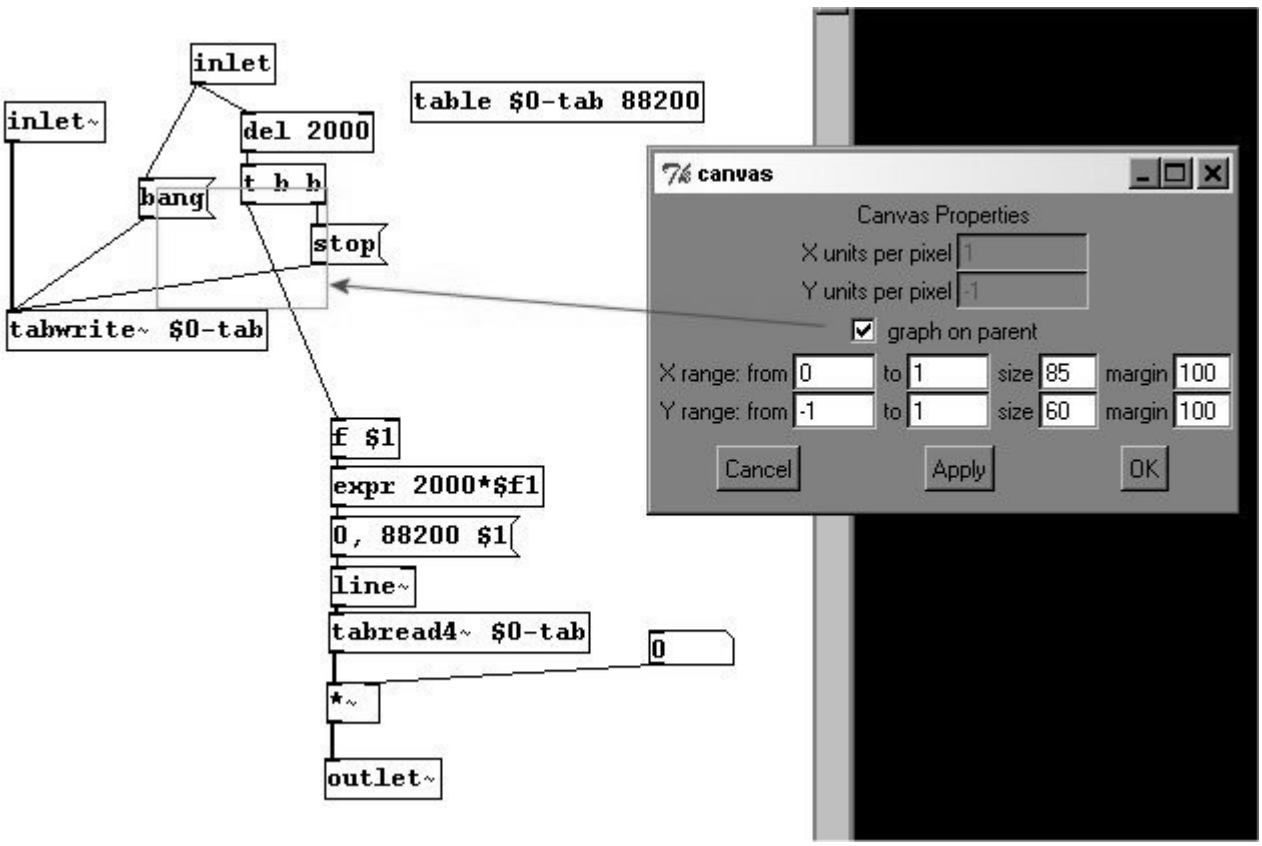
Un número aleatorio es generado con un desplazamiento de 1000, y cuenta de allí hacia arriba. Ahora, cada vez que llama a la abstracción otro número será generado:



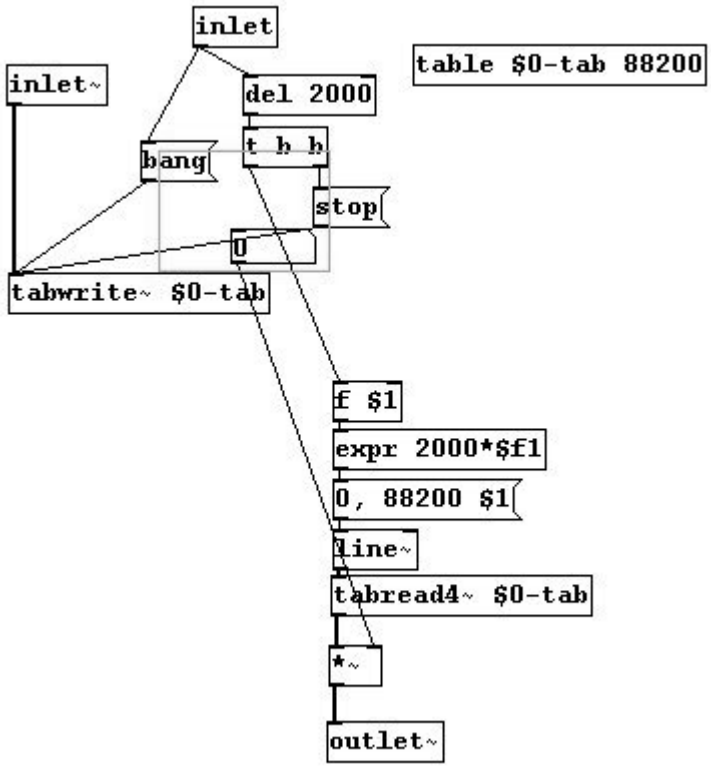
Esto se usa para dar un nombre único a la matriz. La forma convencional es: $0-[arrayname]. Tendrá que configurar una variable ($1) para la velocidad. El volumen debería mantenerse variable.



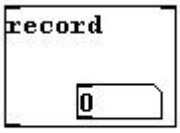
Finalmente, puede llamar elementos gráficos desde una abstracción usando la función "graph on parent". En el patch "record" puede crear una caja Número en vez de una entrada para el volumen, luego clic-derecho en la superficie blanca y allí **Properties**. Tilde la opción "graph on parent" y luego clic en "apply":



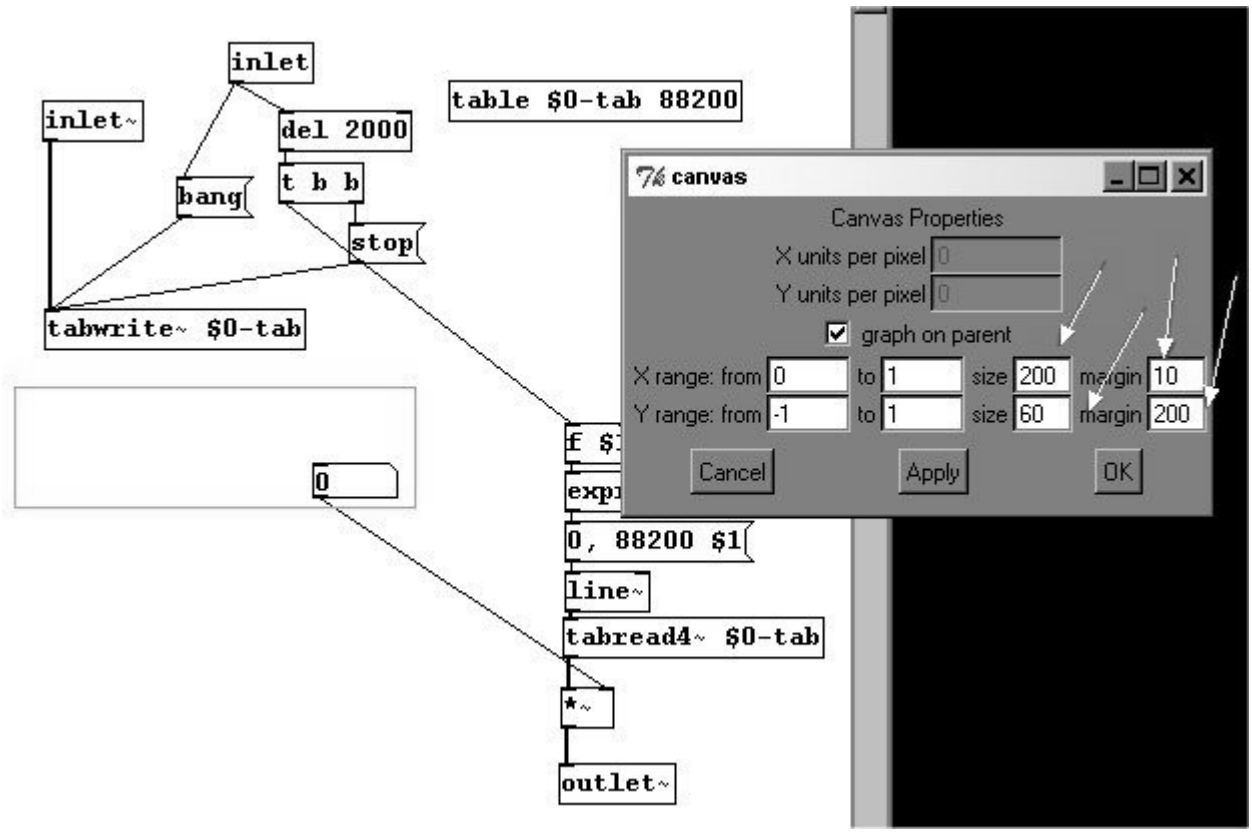
Debería ver un rectángulo rojo. Todos los Objetos GUI que se encuentran encerrados dentro de este rectángulo rojo aparecerán luego en el Objeto. Mueva la caja Número para el volumen dentro del reactángulo rojo, guarde "record.pd", y luego cree el Objeto en el nuevo patch:



En el nuevo patch:

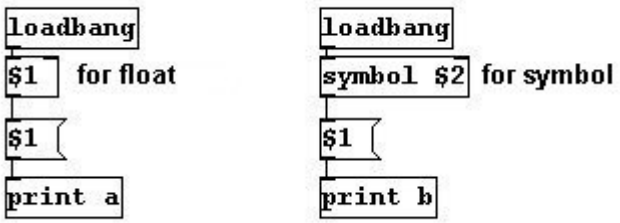


Puede cambiar el tamaño y la posición del espacio rojo aquí:



"Graph on Parent" también funciona con los subpatches.

Dos puntos importantes: puede ver y cambiar los contenidos de una abstracción en el patch principal (sólo basta con un clic sobre el Objeto en el patch principal). ¡Sin embargo, los cambios que realice aquí no serán guardados! Sólo si abre en Pd la abstracción misma como un patch normal y realiza los cambios necesarios, sólo de esta manera serán guardados. El segundo punto consiste en que las variables $ sólo pueden ser escritas en Objetos dentro de una abstracción; S1 en una caja Mensaje tendrá el mismo sentido que una entrada en la caja Mensaje (cf. 2.2.2.1.4). Por ejemplo:



Y, por supuesto, si copia el patch principal y quiere usarlo en cualquier lado (por ej., en otro directorio o en otro ordenador) tiene que también copiar el patch "record". Puede hacer que las abstracciones que usted cree se encuentren disponibles para cualquier patch si las guarda en el directorio "extra" sito en el directorio principal de Pd. Todos los patches que se encuentran allí se encuentran siempre disponibles como abstracciones. De manera similar, puede crear su propio directorio conteniendo abstracciones y configurarlo para que Pd los cargue automáticamente. Para lograrlo puede configurar una ruta de acceso al mismo bajo **File** > **Path**.

5.1.1.3 Expandiendo Pd

Muchos Objetos útiles no incluidos en la versión original de Pd han sido desarrollados por programadores de todo el mundo desde su concepción. Estos Objetos son llamados "externos".

Colecciones de *externos* son llamados "librerías", por ej., la librería zexy o la librería MaxLib. Debe ubicarlos dentro de la carpeta llamada "Extra" y también integrarlas dentro del proceso de inicialización (**File** > **Startup**). Una nueva versión de Pd llamada "Pd-extended" ya incluye librerías que expanden la versión original.

Pd también puede ser expandido a través del uso de programas adicionales. GEM es el más conocido y es usado para procesar datos de video y archivos de video de manera similar a como Pd procesa el sonido y los archivos de sonido. Adicionalmente, pueden ser usados en Pd los datos de OpenSound Control (OSC) de otros programas que también disponen de una conexión OSC. Las versiones de Pd de algunos programadores han sido expandidas de tal manera que se hace difícil reconocerlas como versiones de Pd.

5.1.2 Aplicaciones

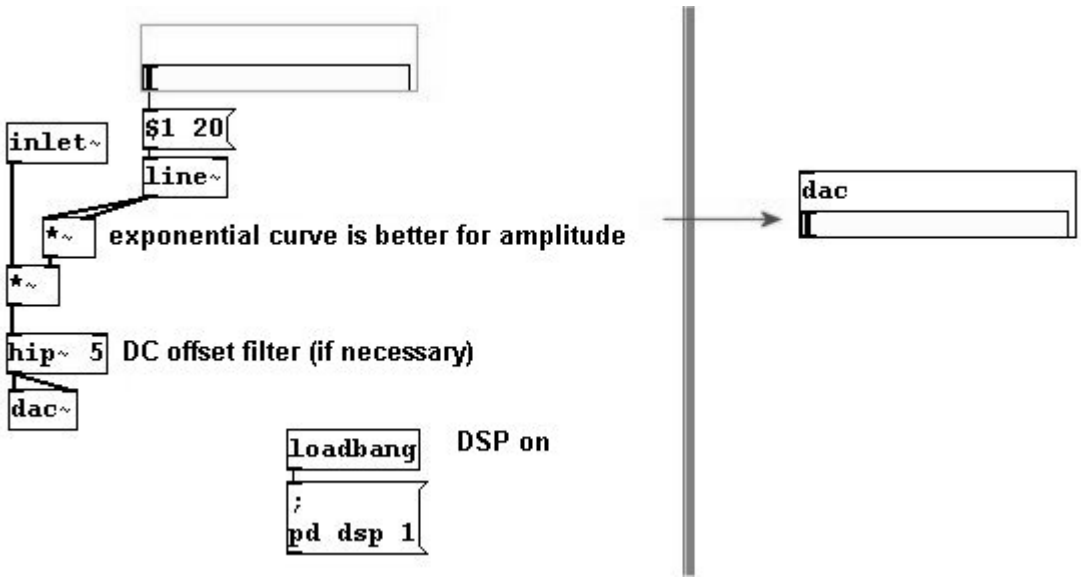
5.1.2.1 Personalizar su Pd

Como mencionamos en el capítulo anterior, Pd puede ser fuertemente personalizado. Crearemos a continuación algunas abstracciones útiles.

As mentioned in the previous chapter, Pd can be customized to a large extent. Let's create some useful abstractions.

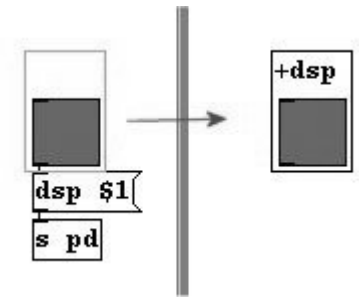
En una programación normal generalmente necesitamos un Objeto "dac~" con un slider de volumen incorporado. Llamaremos a esta abstracción "dac":

patches/dac.pd



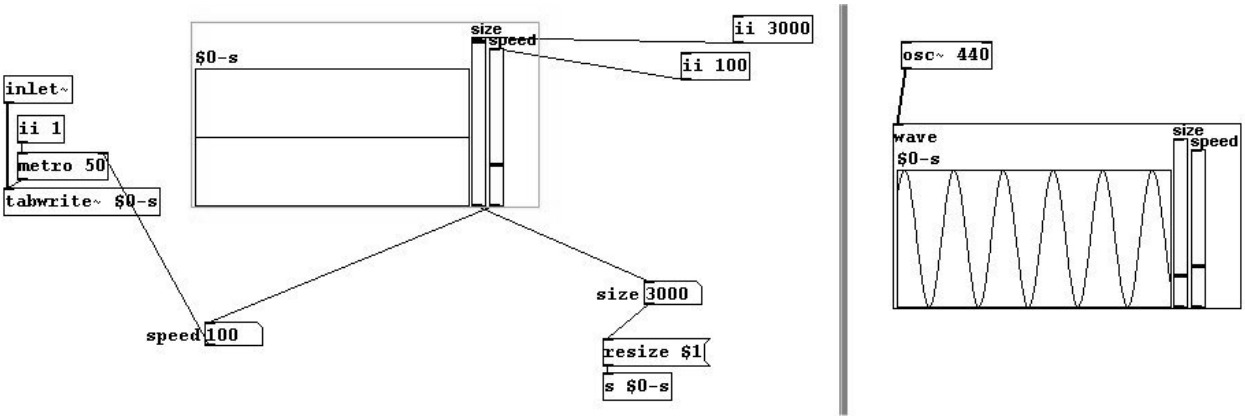
Un interruptor de DSP, llamado "+dsp":

patches/+dsp.pd



Una representación gráfica de una forma de onda:

patches/wave.pd



5.1.2.2 Más ejercicios

Como abstracciones:

a) Integre el Interruptor de DSP dentro de la abstracción "dac" así como un botón *silencio* encendido/apagado

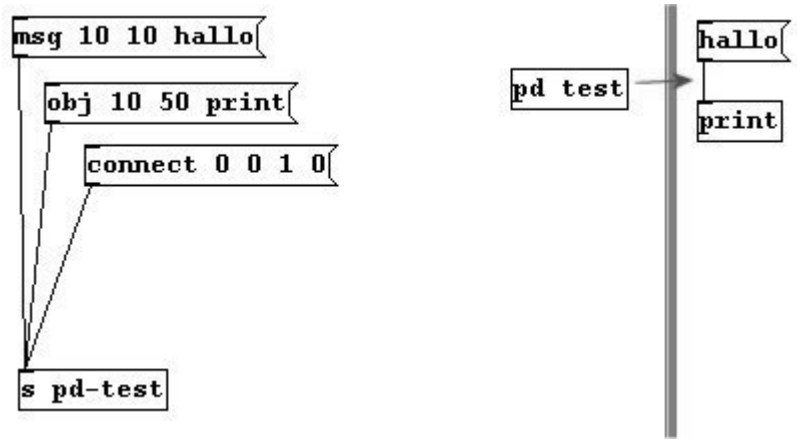
b) Construya un compresor.

c) Contruya una representación gráfica de un espectro entrante.

5.1.3 Apéndice

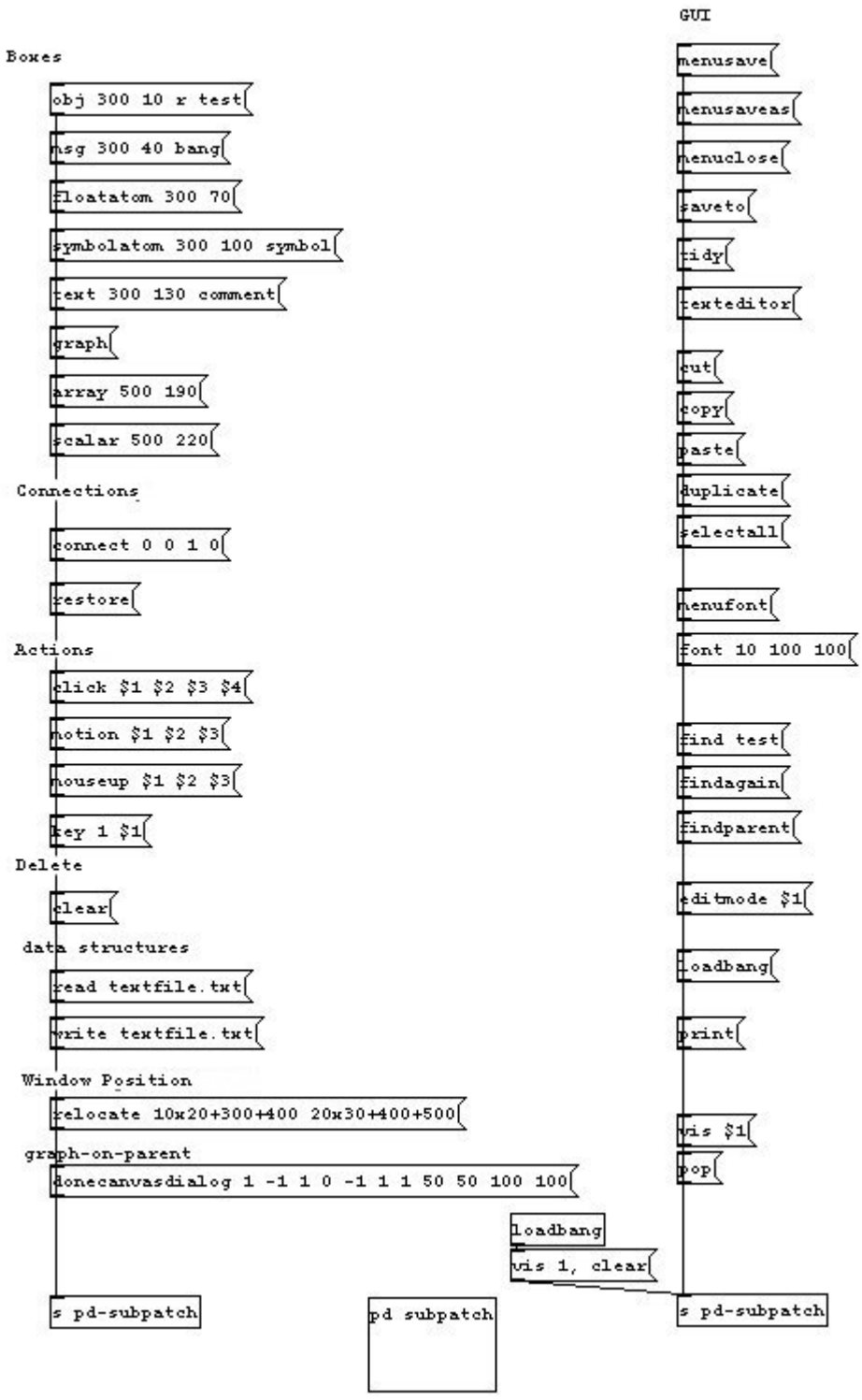
5.1.3.1 Crear un patch automáticamente

Tenemos un patch con un subpatch dentro llamado "test". En el patch principal enviamos los siguientes Mensajes (de arriba hacia abajo) a el subpatch usando "s pd-test":



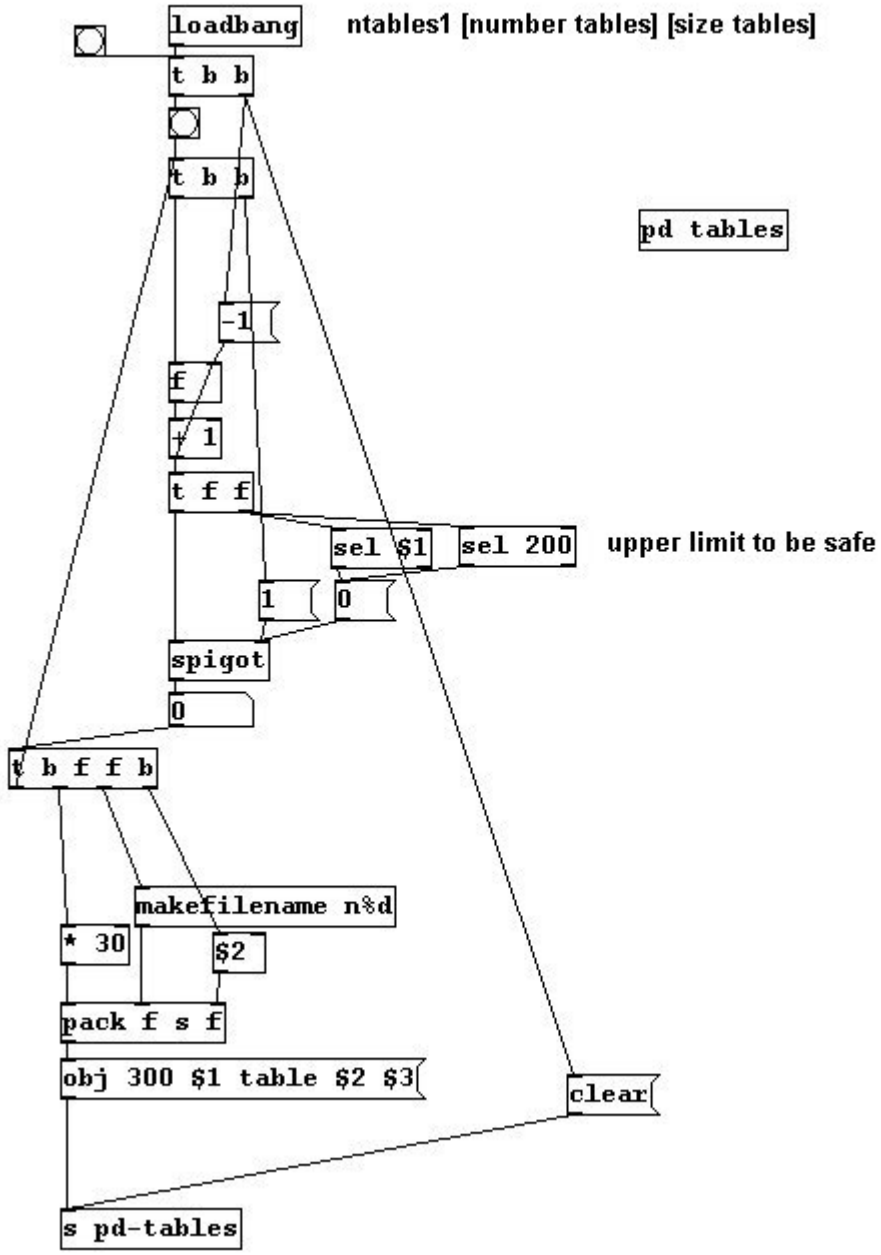
"obj" crea un Objeto; ingresa las *coordenadas x/y* y el *nombre del Objeto* con sus argumentos. Lo mismo cuenta para las cajas Mensaje. La coneción se realiza de la siguiente manera: "connect [número de Objeto] [número de salida] [número de Objeto] [número de entrada]". Los Objetos son numerados de acuerdo al orden en que son creados, las salidas y las entradas de izquierda a derecha. Todo es configurado inicialmente a 0. El Mensaje "clear" borra el contenido de un subpatch.

Aquí presentamos una visión de conjunto de todos los comandos. Hasta el presente (Junio de 2008) algunos de estos todavía no funcionan:



Esto nos permite –aunque de una manera un tanto complicada–, por ejemplo, crear un cierto número de matrices con una abstracción. Este es el único método que Pd nos permite para llevar a cabo esta operación (cf. 3.2.3.1):

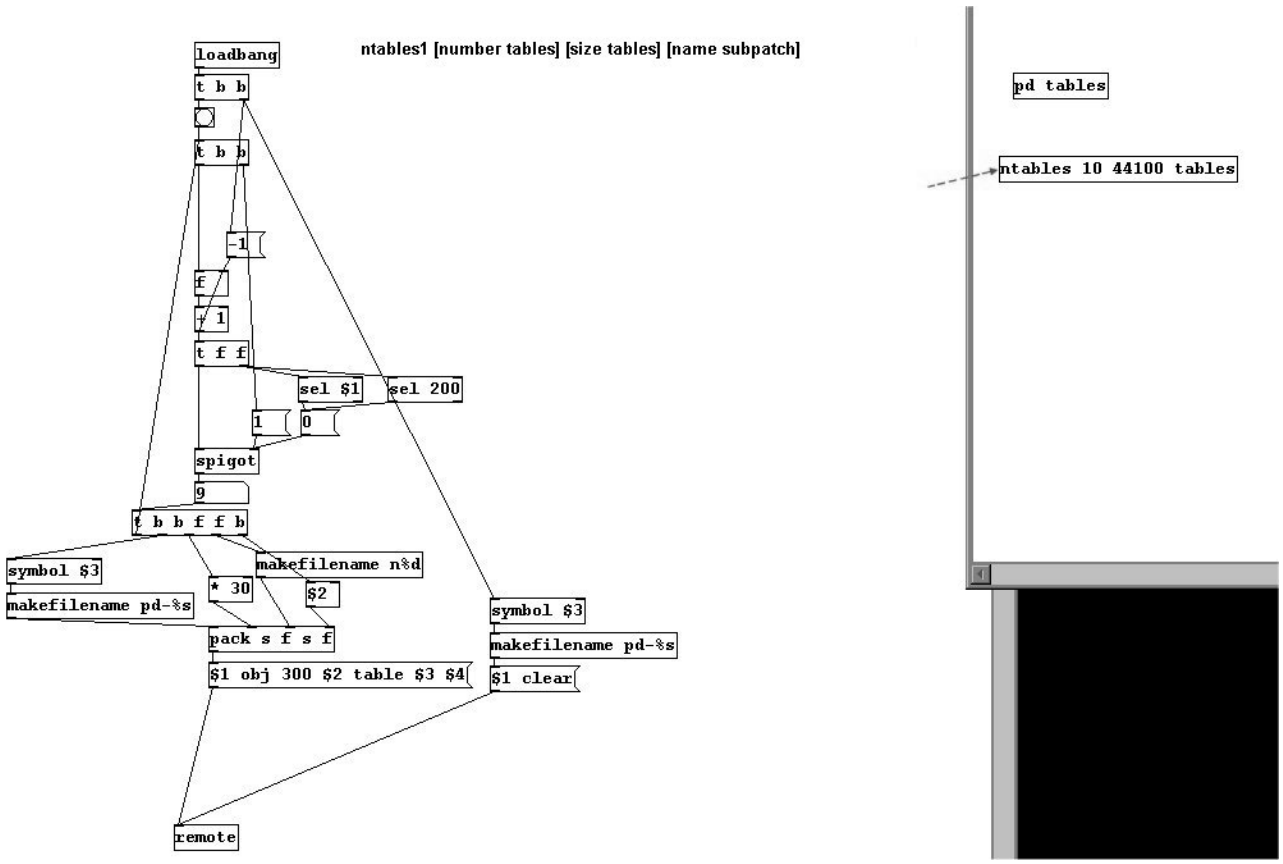
patches/ntables.pd



Con esta abstracción puede crear diez matrices de tamaño 44100 dentro del subpatch:

```
ntables1 10 44100
```

De esta manera, puede también crear el subpatch en su patch principal:



5.1.4 Para los interesados, especialmente

5.1.4.1 Escribir sus propios Objetos

Es posible escribir sus propios Objetos, naturalmente. Pd es sólo una superficie para un programa en el lenguaje de programación C, el cual permite escribir sus propios Objetos ("externos"). Aquí dispone de una guía para ayudarlo a llevar a cabo esto:

http://iem.at/pd/externals-HOWTO/

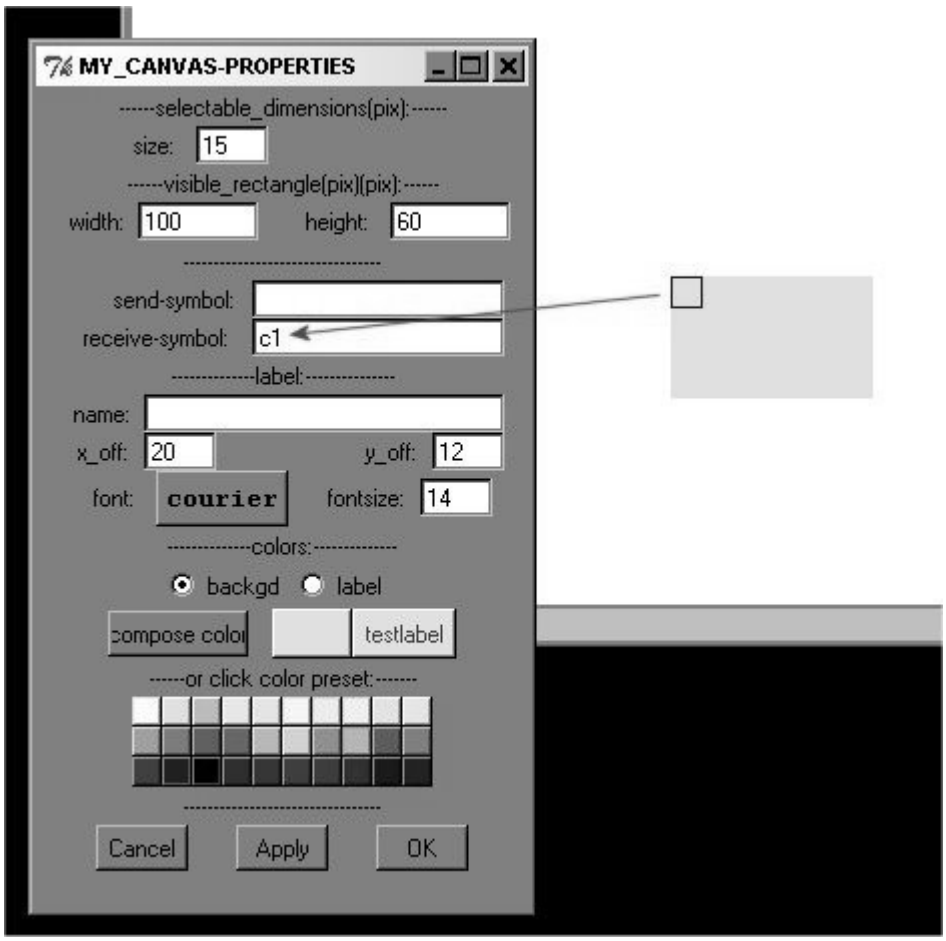
5.2 Visuales

5.2.1 Teoría

5.2.1.1 Pd es visual y esto puede ser programado

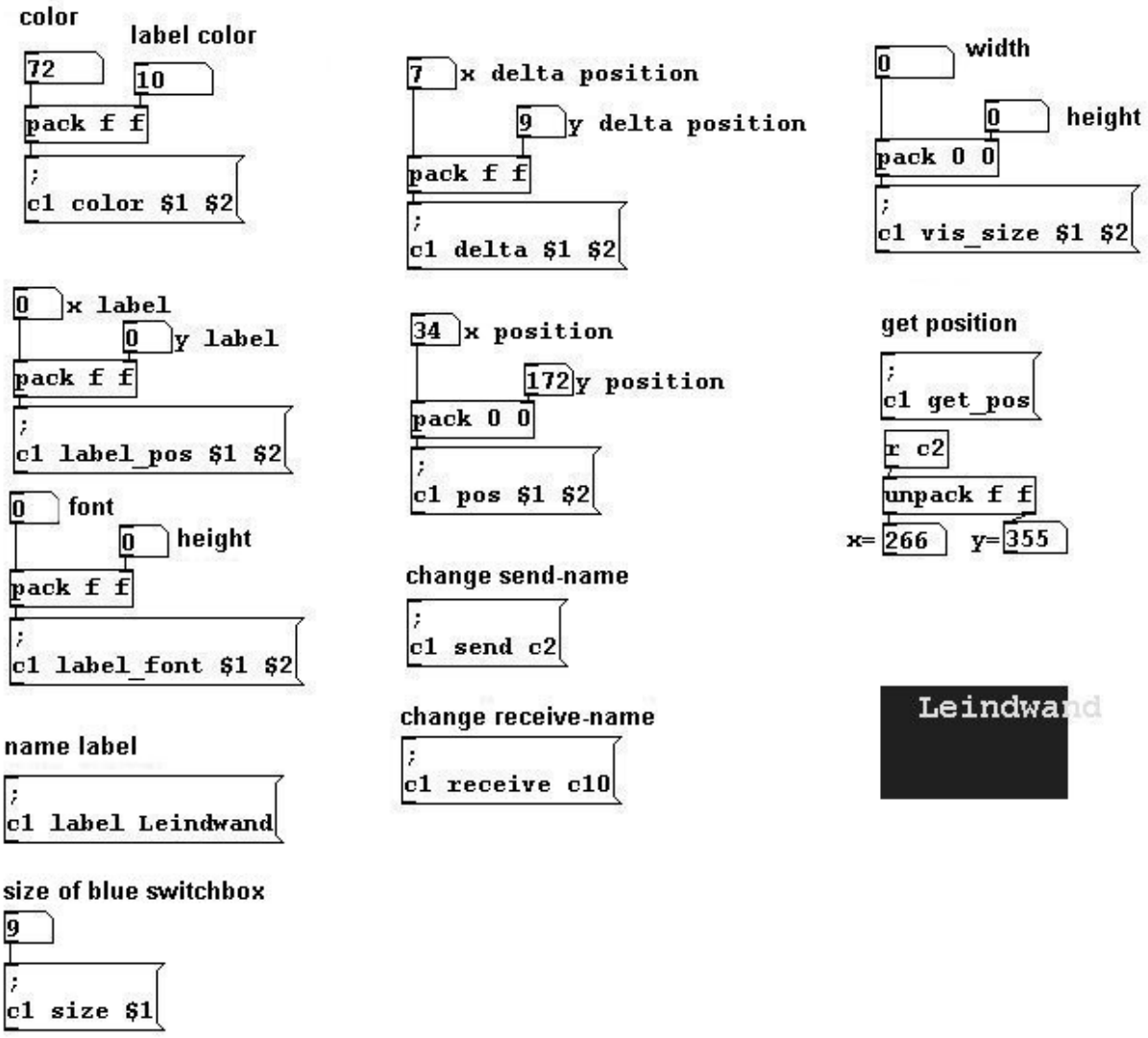
Las herramientas para la representación visual fueron introducidas en 2.2.4.3. Aquí examinaremos cómo controlar un lienzo.

Cree un lienzo (**Put** > **Canvas**) e ingrese en el mismo el *símbolo de recepción* "c1" bajo **Properties** (clic-derecho sobre la caja superior-izquierda del lienzo):



Ahora puede enviar Mensajes al lienzo:

patches/5-2-1-1-canvas.pd



5.2.2 Aplicaciones

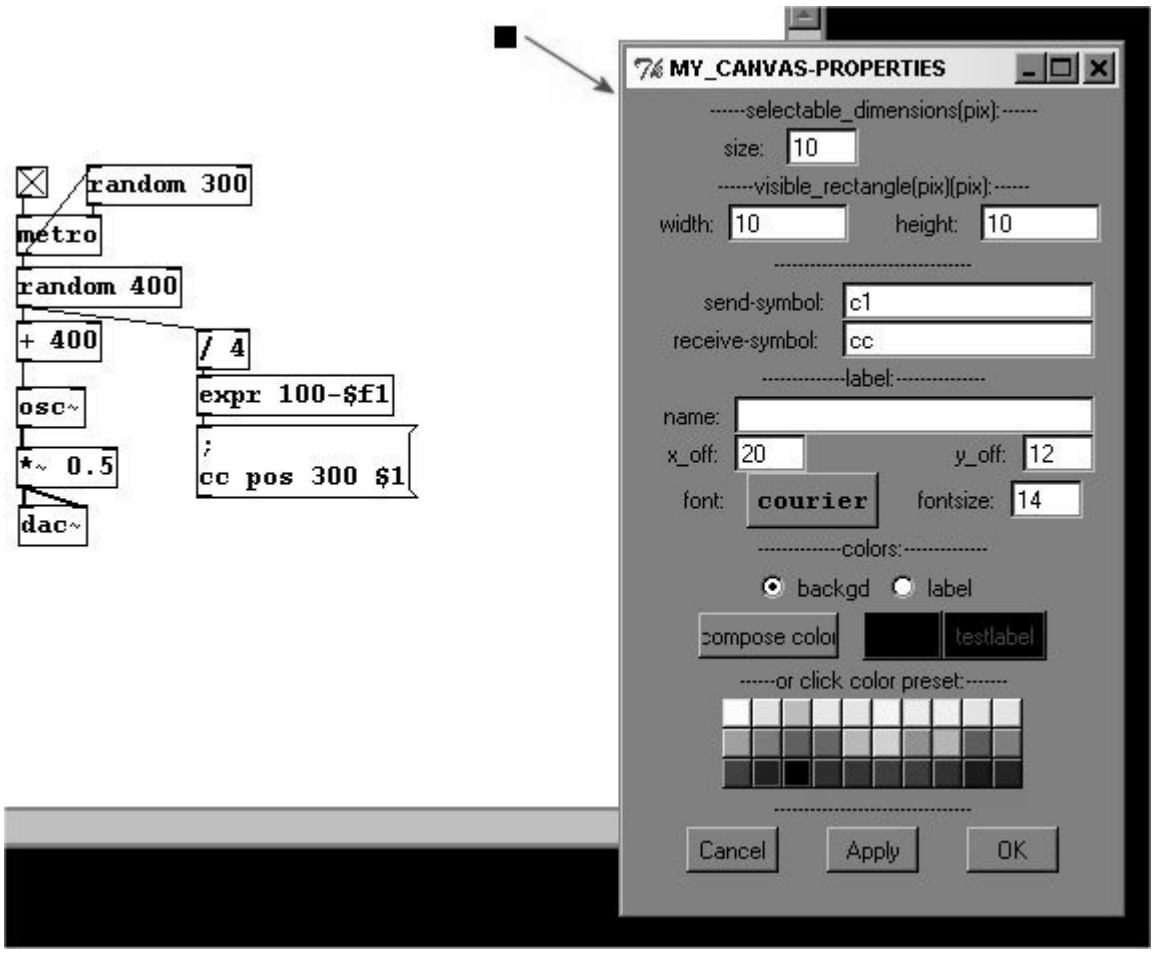
5.2.2.1 Ventana principal del patch y subpatches

Debemos dejar en claro algo de lo dicho anteriormente: un patch ordenado se comprende de una ventana principal y los subpatches (cf. 3.4.2.4 and 5.1.1.1). En este manual hemos insistido constantemente en este tipo de organización en aras de la claridad (resulta más fácil comprender un gráfico cuando toda ña información requerida se encuentra en una sóla ventana.

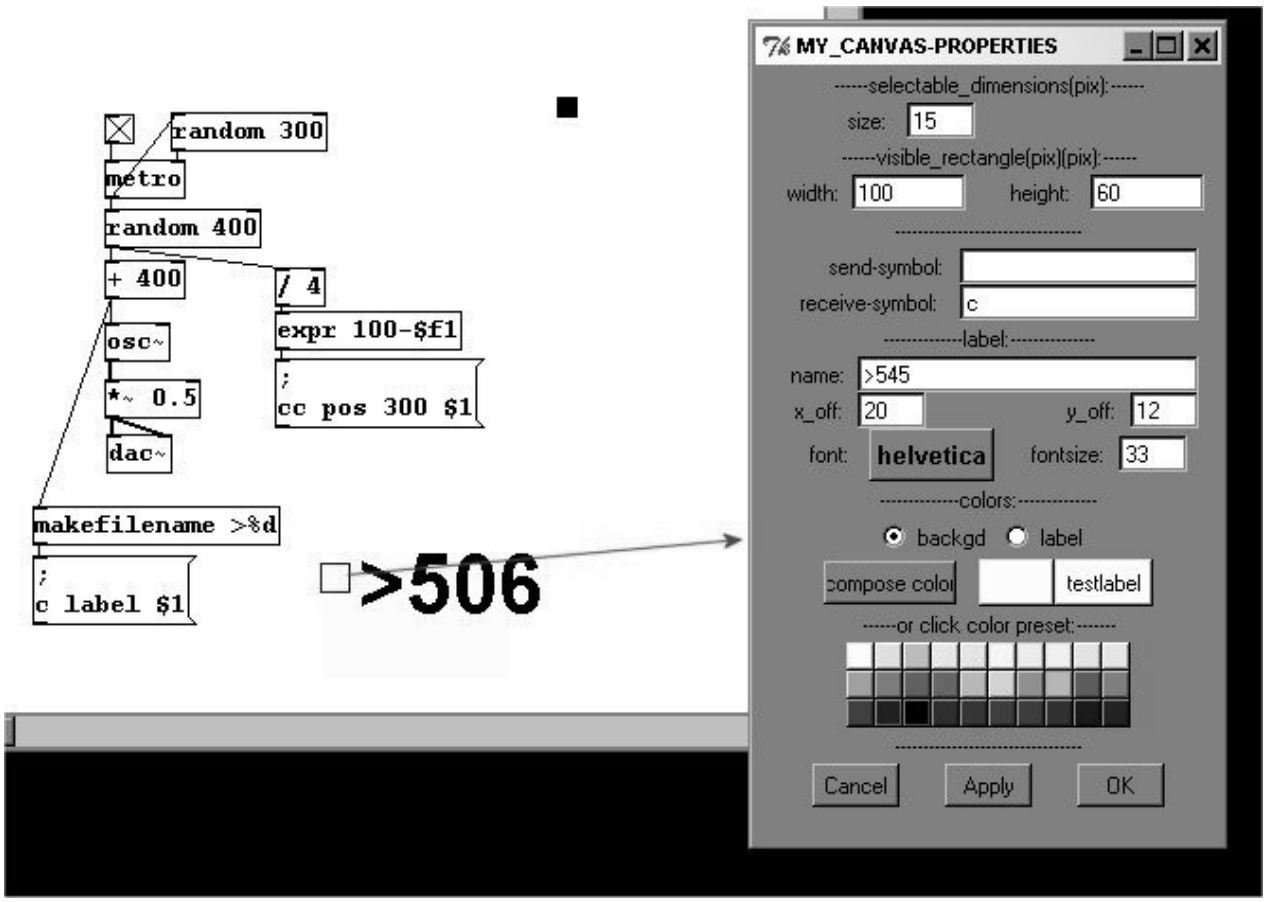
5.2.2.2 Lienzos como pantalla

Los elementos de color en los lienzos pueden maximizar de gran manera la claridad de sus patches (cf. 3.4.2.4). Por cuestiones de visualización, también pueden ser usados de manera variable. Pore ejemplo, podría mostrar los resultados de un generador aleatorio de la siguiente manera:

patches/5-2-2-2-canvas-display.pd



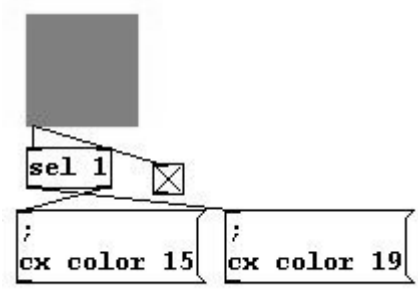
También podría mostrar un número en un tamaño mayor (sin embargo, un símbolo siempre debe acompañar al número; aquí se utiliza un ">"):



Las posibilidades son prácticamente infinitas; podría incluso producir partituras gráficas completas utilizando lienzos.

5.2.2.3 Lienzos como GUI expandido

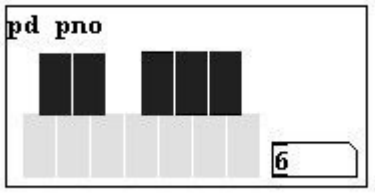
Los lienzos pueden ser impuestos sobre otros Objetos GUI; el orden en el que se crean los mismos puede ser crítico. Podría realizar sus propios toggles:



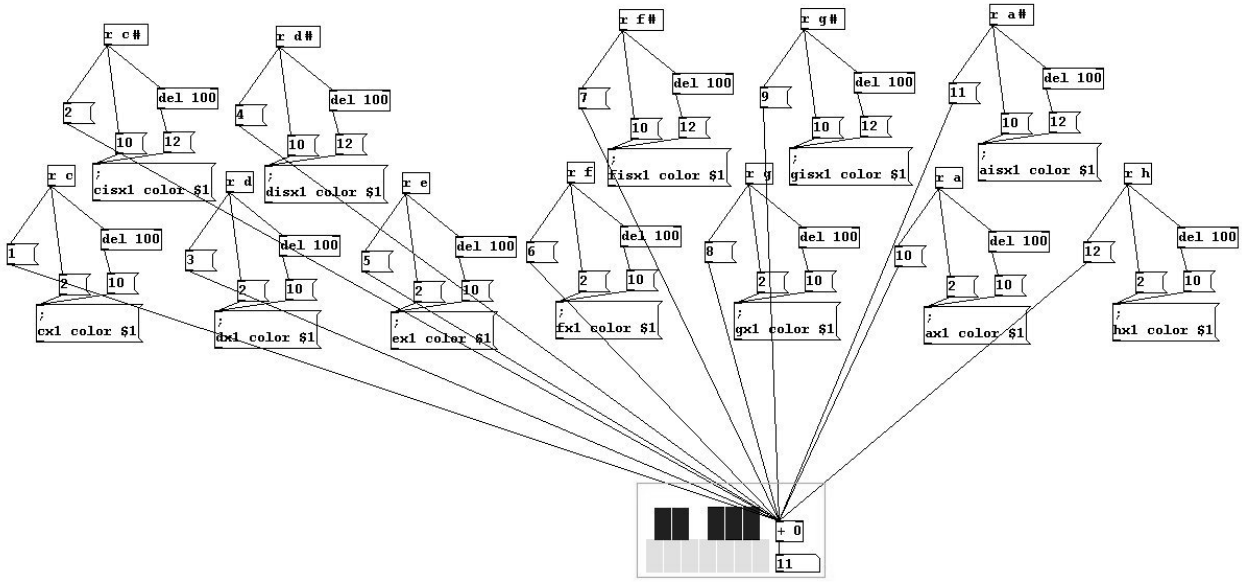
Primero fue creado un toggle con un tamaño de 55 y luego fue conectado a "sel 1". Luego se creó un lienzo con un tamaño 55 y un símbolo de recepción "cx" el cual fue superpuesto directamente encima del toggle (el orden de la creación de los Objetos es esencial; de lo contrario, el lienzo desaparece detrás del toggle). El Objeto "sel 1" y todos los demás restantes podrían ser separados dentro de un subpatch y finalmente usar un Objeto "send" interno para el toggle.

Aquí también encontrará que las posibilidades no tienen fin. Para nombrar un sólo ejemplo, puede reemplazar una octava de teclas de piano como la siguiente (queda claro que yo no soy el diseñador gráfico de Pd más talentoso):

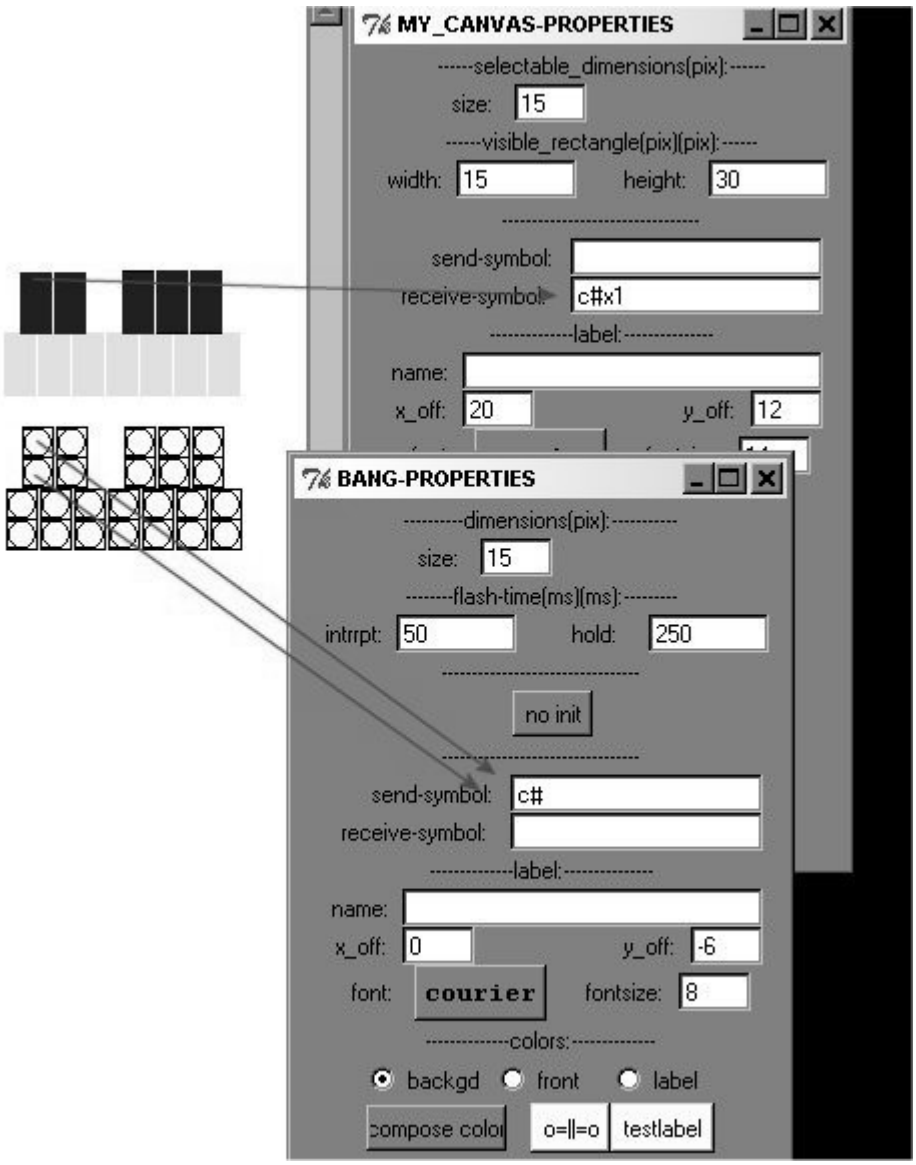
patches/5-2-2-3-piano-display.pd



El subpatch contiene:



Detras de las 'teclas de piano' existen una serie de bangs:



5.2.2.4 Más ejercicios

a) Cree un cronógrafo.

b) Un gráfico para un metro de 5/8, es decir, una guía visual que parpadea.

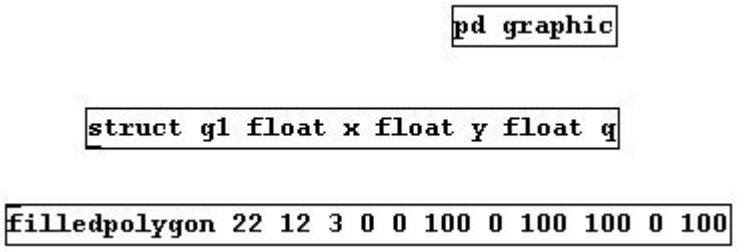
5.2.3 Apéndice

5.2.3.1 Estructuras de datos

Las estructuras de datos son una familia de gráficos entera en Pd.

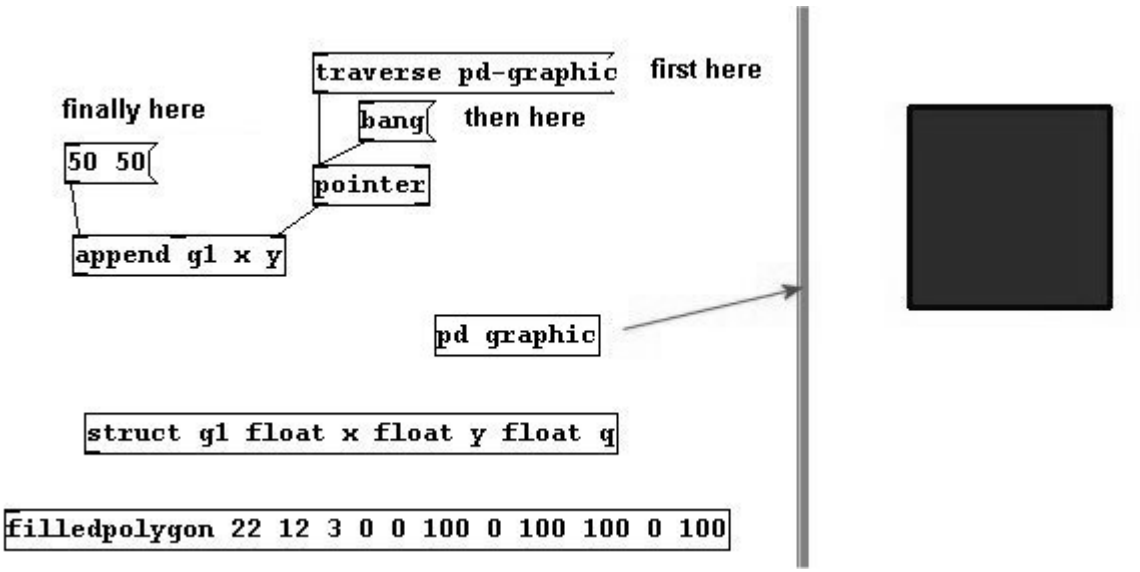
Usted puede ubicar elementos gráficos en un subpatch. Primero debe crear el subpatch "graphic" y definir las variables y un gráfico para este subpatch. A esto se lo llama "template". Contiene las variables en el Objeto "struct"; como argumentos ingrese el *nombre del template* (aquí "g1") y luego incluya el *tipo de datos* y el *nombre* –en este caso, float x float y float q. "float" corresponde al *tipo* (un número decimal); x, y & q corresponden a *nombres* de elección libre.

El gráfico puede ser definido con los Objetos "drawcurve", "drawpolygon", "filledcurve", y "filledpolygon". Para nuestro primer ejemplo usemos "filledpolygon". Un polígono es una figura geométrica que posee muchos lados. Los argumentos de izquierda a derecha corresponden a *color interior*, *color de perímetro*, *ancho de perímetro* y *pares de puntos de coordenada* (comenzando por la parte superior izquierda y procediendo en función de las agujas del reloj).



Para crear un gráfico necesitamos el Objeto "append". Su primer argumento corresponde al nombre del template y luego variables posibles –x e y deberían ser utilizadas siempre. La entrada derecha corresponde al lugar en el subpatch donde el gráfico va a ser localizado. Tiene que imaginar que los elementos gráficos en un subpatch son ordenados uno luego del otro como en una lista. "append" primero debe saber el lugar en la lista que el Objeto "pointer" ha llamado. Ingrese a "pointer" el Mensaje "traverse pd-graphic"; esto provocará que "pointer" se dirija al principio de la lista. Un Mensaje "bang" enviará este lugar a la lista (a "append"). Luego ingrese datos para las variables dentro de la entrada izquierda del Objeto "append".

patches/5-2-3-1-data-structures1.pd



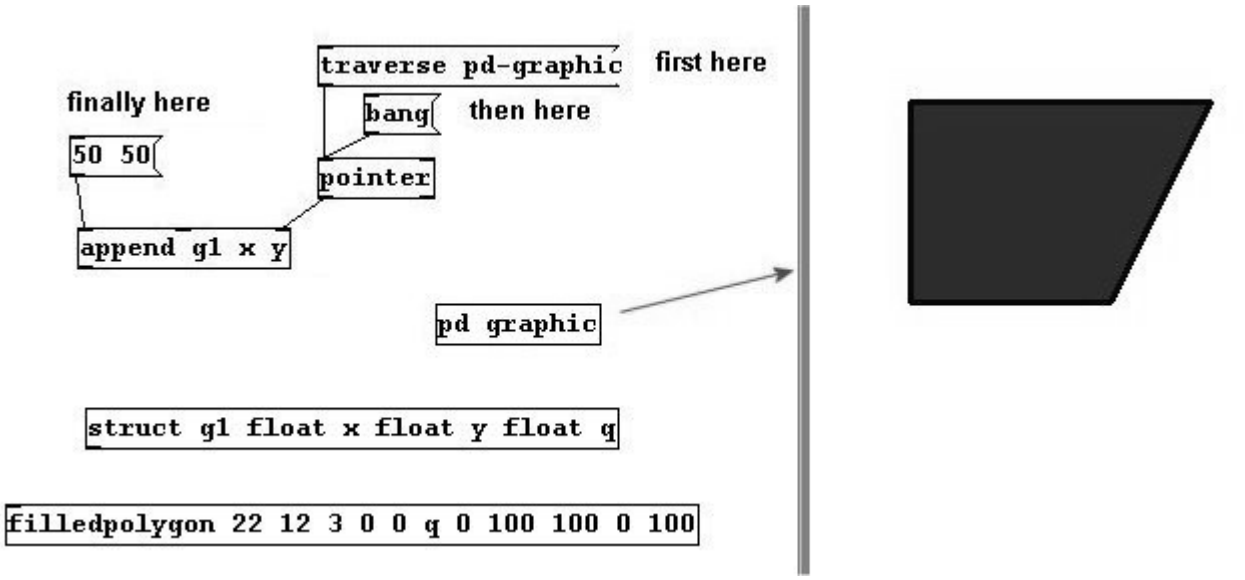
Sólo en aras de claridad: "x" e "y" son nombres especiales para un elemento gráfico usado en las estructuras de datos. Definen una posición absoluta, 50/50. Ahora ha creado un "filledpolygon" con el color interior 22, el perímetro de color 12, el ancho de perímetro 3, y cuatro puntos: superior izquierdo a una distancia de 0/0 desde la posición absoluta, superior derecho a una distancia de 100/0 desde la posición absoluta, inferior derecha a 100/100, e inferior izquierda a 0/100. Si

simplemente quita los últimos dos números de "filledpolygon", obtendriamos como resultado un triángulo. Puede aplicar este mismo método para crear un polígono con cualquier número de lados.

"traverse pd-graphic" simplemente le lleva al principio de subpatch "graphic"; un "bang" envía esta posición a "append" y, cuando le enviemos datos con las variables necesarias, "append" creará un gráfico en esta posición.

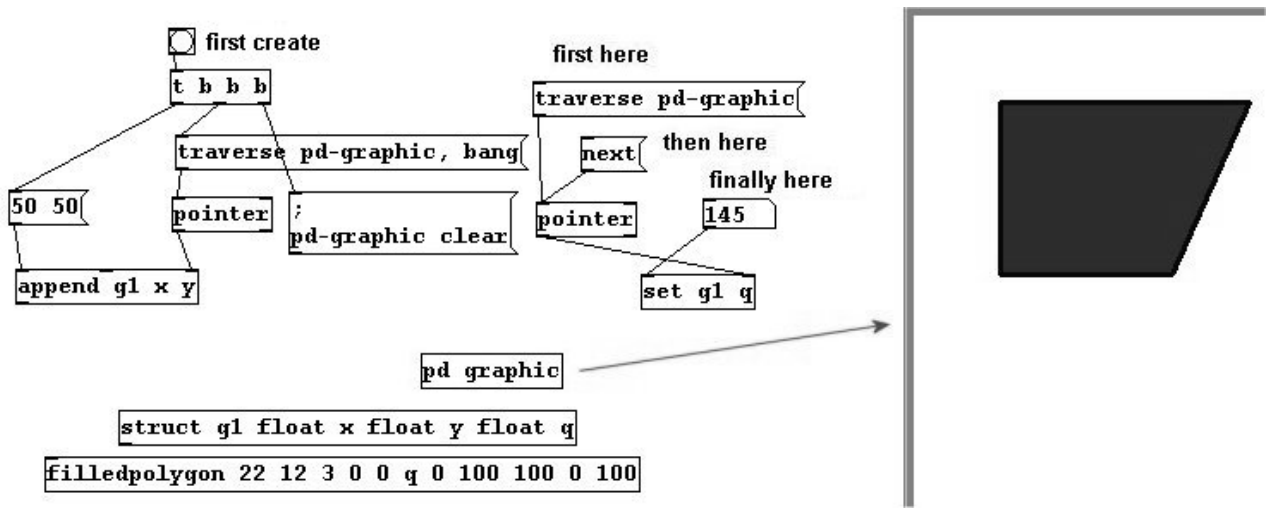
Pero también podría cambiar algo en este gráfico. Ha definido la variable "q" (un número decimal) para el template "g1". Ahora ingrese lo siguiente dentro de "filledpolygon": "filledpolygon 22 12 3 0 0 q 0 100 100 0 100". Un punto tiene sólo una variable. El punto desaparece y ahora obtiene un triángulo. Esta variable puede ser configurada usando "append":

patches/5-2-3-1-data-structures2.pd



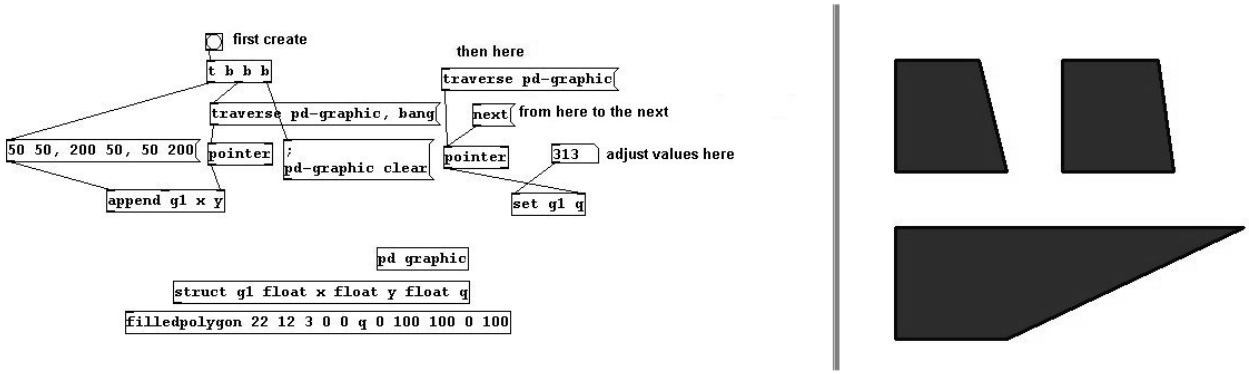
...pero también usando "set". Este también debe recibir la información del *pointer* con respecto a la posición donde algo será cambiado en el subpatch. En este caso, es un paso después del inicial. Use "traverse pd-graphic" para dirigirse al principio. El inicio se encuentra vacío; use "next" para ir al siguiente gráfico. Ahora puede ingresar un valor para *q* en la entrada izquierda.

patches/5-2-3-1-data-structures3.pd



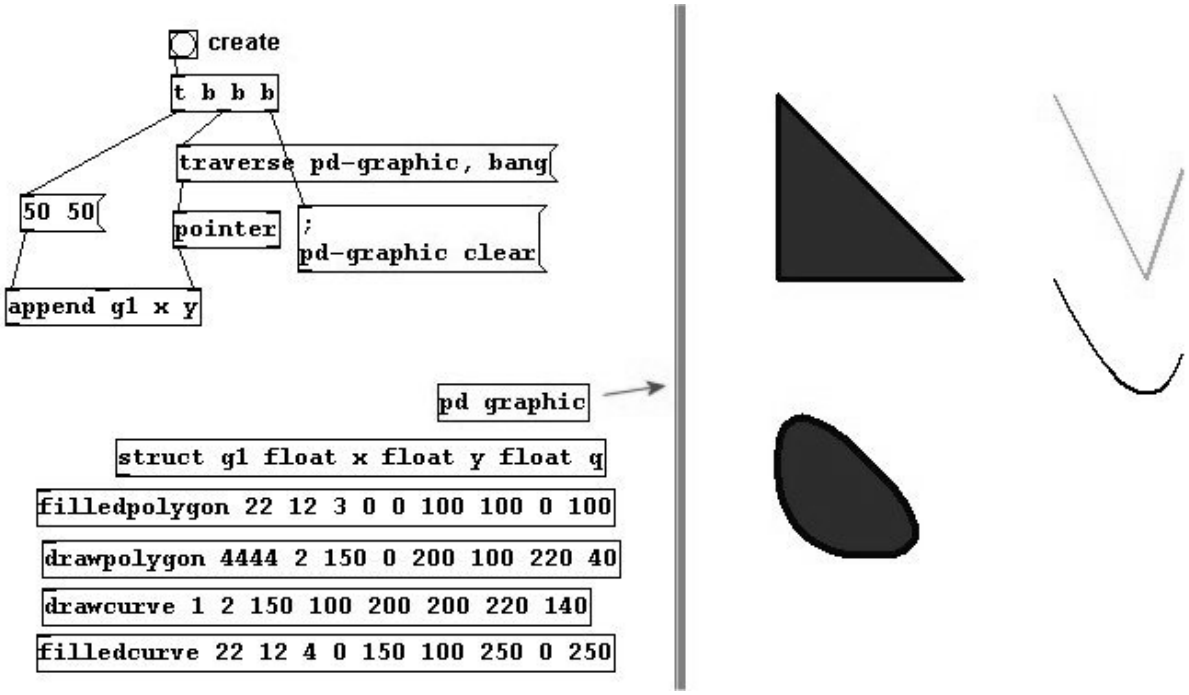
Se pueden realizar muchos gráficos simplemente enviando Mensajes a "append". Use "next" para moverse de un gráfico al siguiente.

patches/5-2-3-1-data-structures4.pd



Con respecto a los otros elementos gráficos: "drawpolygon" corresponde sólo a una línea con ángulos; por lo tanto, el primer argumento del *color interior* es omitido. Lo mismo ocurre con "drawcurve", excepto que sus ángulos son redondeados. Sin embargo, "filledcurve" resulta en una figura cerrada y el primer argumento nuevamente es reservado para el *color interior*.

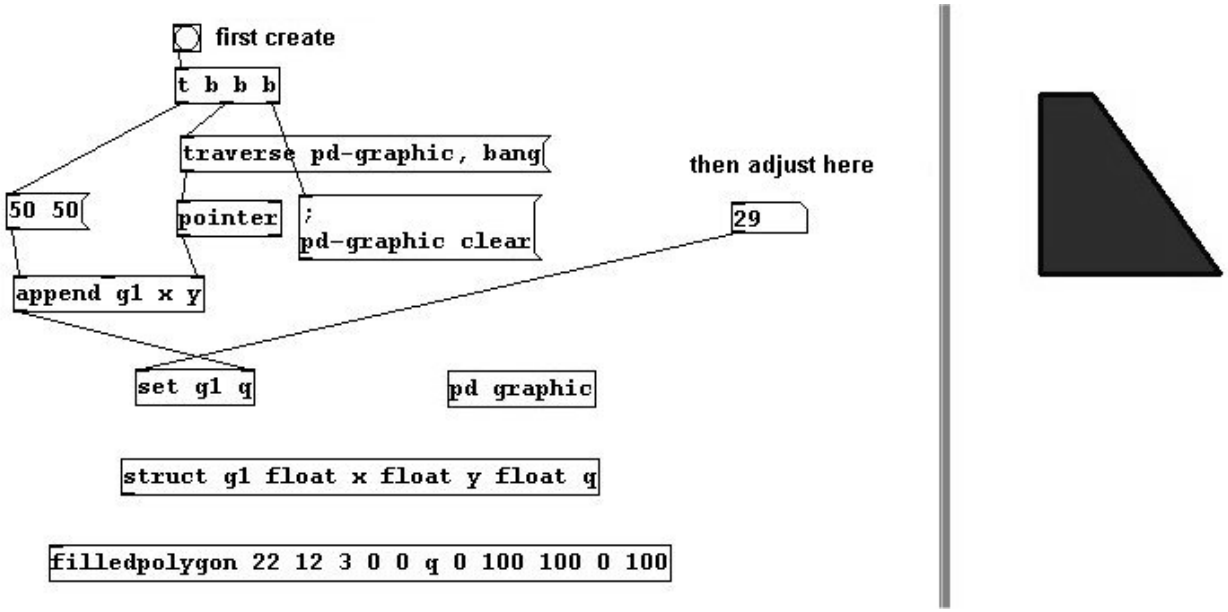
patches/5-2-3-1-data-structures5.pd



Otro punto importante: los puntos (vértices) de un gráfico que poseen variables pueden ser cambiados con el ratón; el puntero cambia su forma en el punto, significando que en ese lugar puede hacer clic y arrastrarlo para modificar la forma del gráfico.

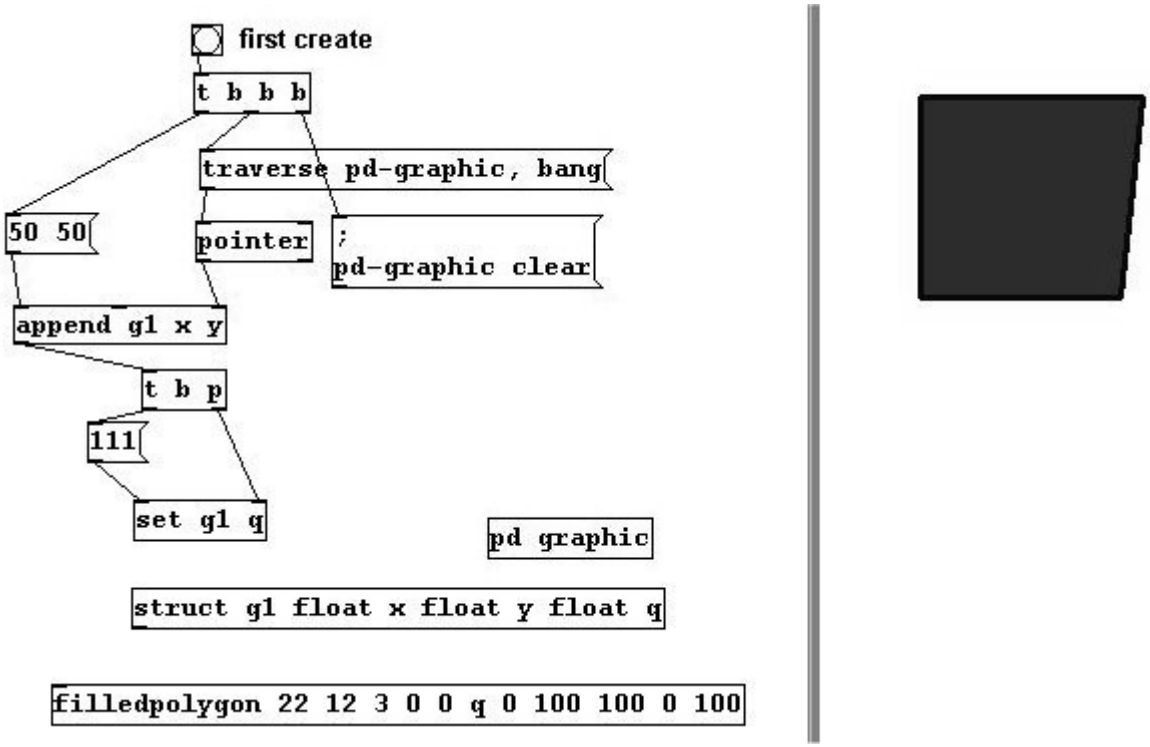
Si utiliza "append" para diseñar un objeto gráfico, la salida corresponde a un nuevo *pointer* para este nuevo objeto (como con "next"):

patches/5-2-3-1-data-structures6.pd



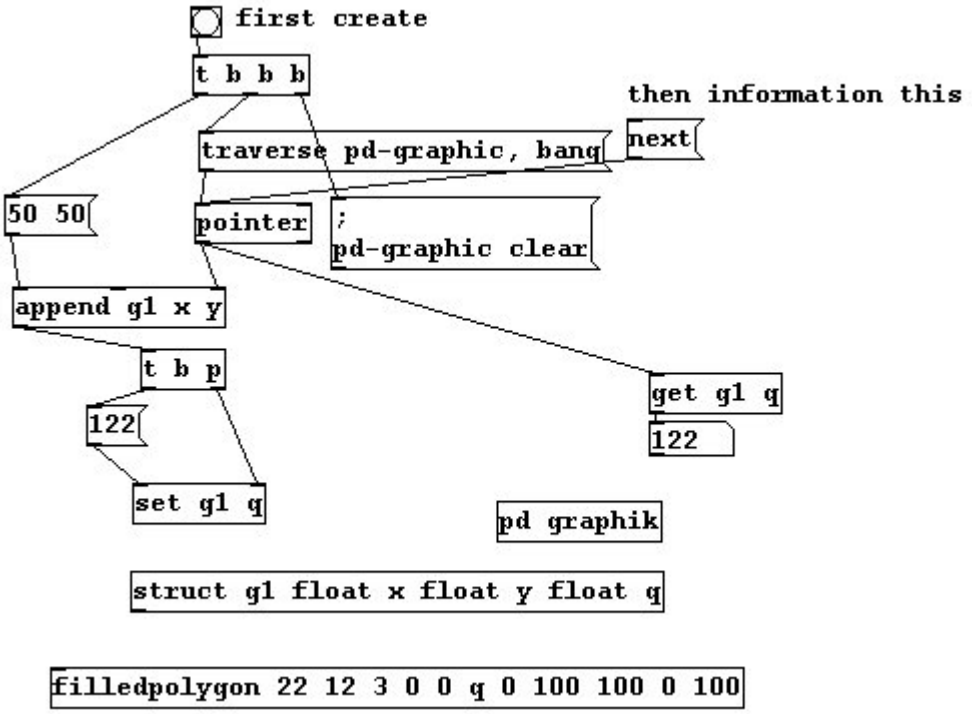
Un *pointer* corresponde a un tipo de información, por ej., para "trigger":

patches/5-2-3-1-data-structures7.pd

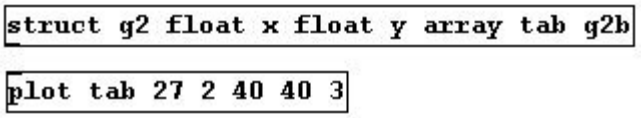


Puede usar "get" para recibir información de los elementos gráficos que se encuentran ligados a los *pointers*:

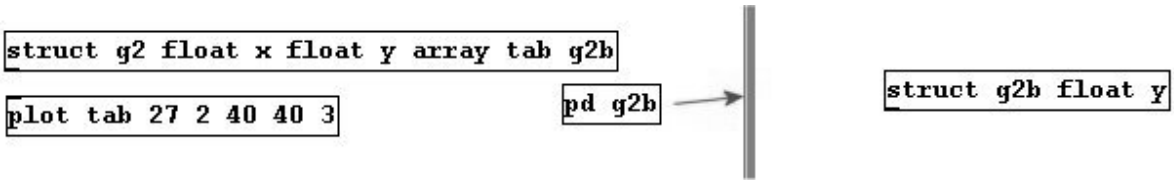
patches/5-2-3-1-data-structures8.pd



Por último, puede crear una matriz (gráfica) usando estructuras de datos. La matriz es definida en "struct" con un *nombre* y otro *template* asignado. Use "plot" para definir el *color*, *ancho*, *punto de inicio* (x/y), y la *distancia entre puntos* para esta matriz.



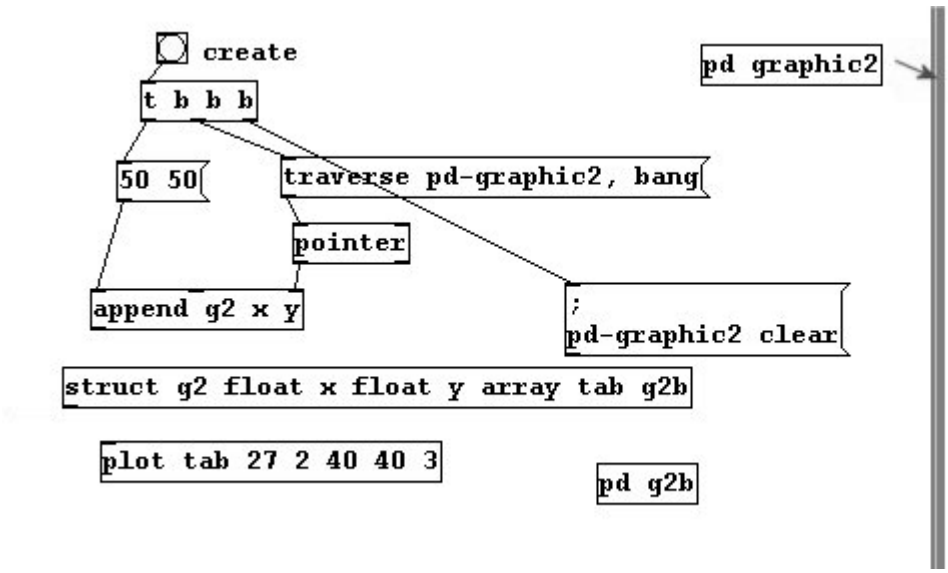
Otro subpatch debe contener el otro *template* que determina las variables de la matriz:



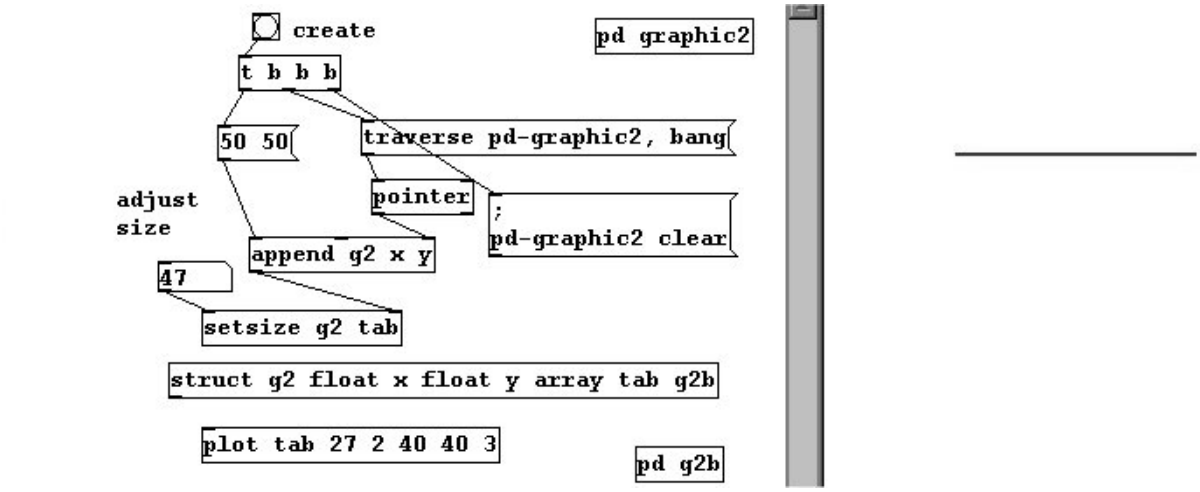
La variable "y" es automáticamente sobreentendida como la altura de la matriz. Esta varible es necesaria para crear la matriz correctamente.

Aquí tenemos la matriz creada:

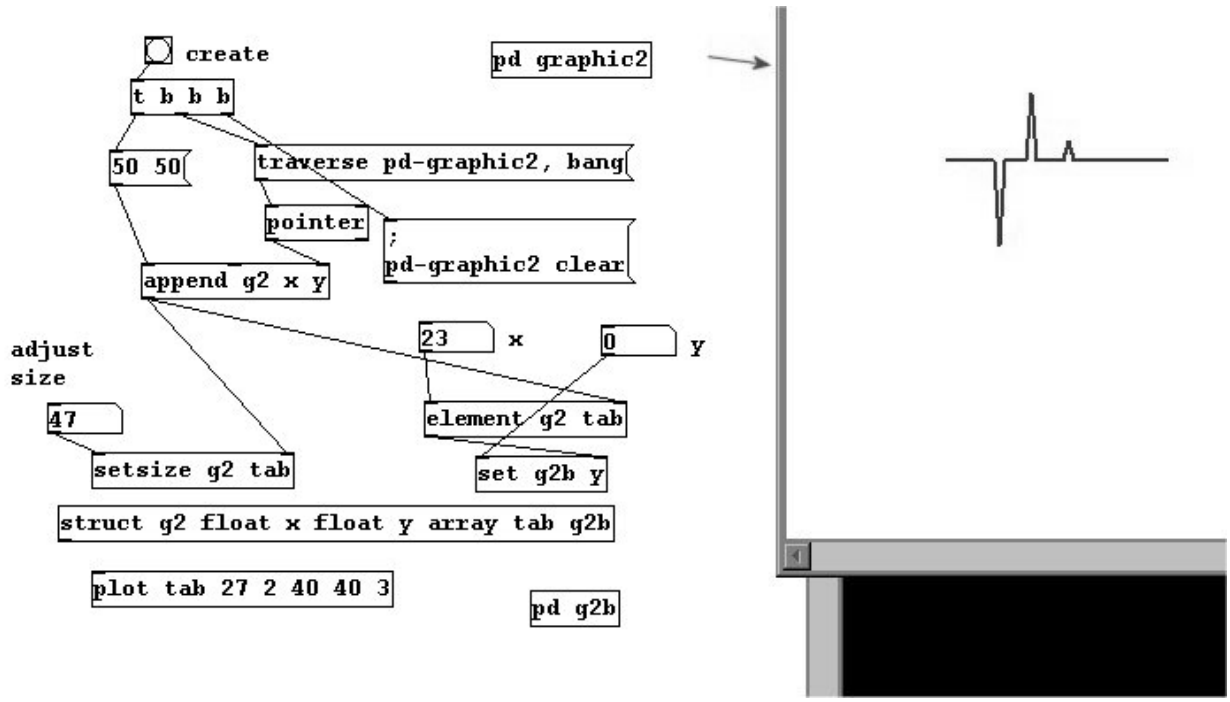
patches/5-2-3-1-data-structures9.pd



Puede cambiar el tamaño con "setsize":



La variable "y" tiene que cambiar algo indirectamente; use "element" para acceder a ella.



La forma de las estructuras de datos pueden cambiar en el futuro (fecha de escritura: Junio 2008). Además, en la documentación original de Pd pueden ser encontradas otras funciones especiales para las estructuras de datos.

5.2.4 Para los interesados, especialmente

5.2.4.1 GEM

Como mencionamos previamente, existe un programa adicional para Pd llamado GEM que es usado para la generación y edición de video.

http://gem.iem.at

Epílogo

Ahora que ha finalizado este tutorial y ha probado todas las técnicas presentadas, esperamos que se encuentre impaciente para combinar todos los Objetos entre sí y hacer sus propios descubrimientos –después de todo, es esta la intención detrás del diseño de Pd.

Por supuesto que no existe un final virtual para el estudio posterior de los conceptos aquí presentados. Muchos otros libros sobre Procesamiento de sonido digital aguardan su lectura – especialmente "Theory and Techniques of Electronic Music" escrito por Miller Puckette, el diseñador principal de Pd. Sería ciertamente recomensable incrementar aún más sus competencias y conocimientos en acústica, técnicas de grabación de estudio, y programación. También debería considerar adquirir fluidez en los fundamentos de algún programa de secuenciación para la programación sonora. Sin embargo, el entusiasmo, la satisfacción artística y la reflexión estética siguen siendo los componentes más importantes.

Por cualquier otro cuestionamiento sobre Pd, aliento el contacto a la comunidad "Pd-list". Este libro fue escrito de acuerdo a la versión 0.39 de Pd de fines de 2007. Esperemos que la información contenida aquí no se desactualice demasiado pronto.

Johannes Kreidler

Apéndice A. Soluciones

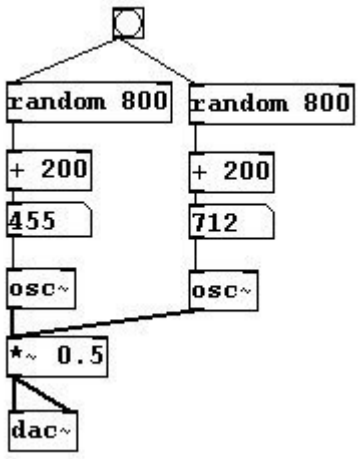
Tabla de Contenidos

Aquí encontrará soluciones sugeridas para las secciones "Más ejercicios" encontrados al final de cada capítulo. Estas no se presentan como las únicas maneras de solucionarloa; en la mayoría de los casos, otras soluciones correnctas son posibles.

2.2.1.2.8

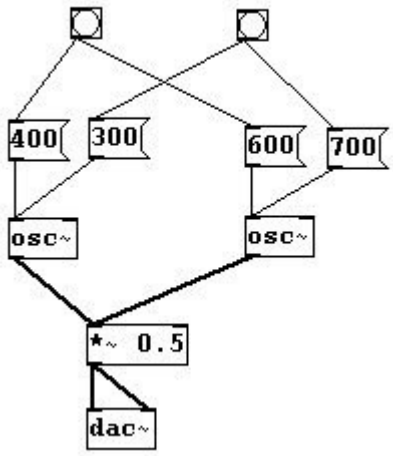
a) Dos melodías aleatorias simultáneas:

patches/a-1-two-randommelodies.pd

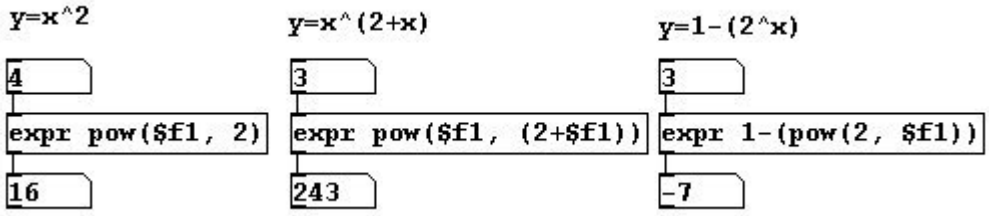


b) Seleccione dos intervalos (cualquiera) diferentes usando dos bangs:

patches/a-2-two-intervals.pd

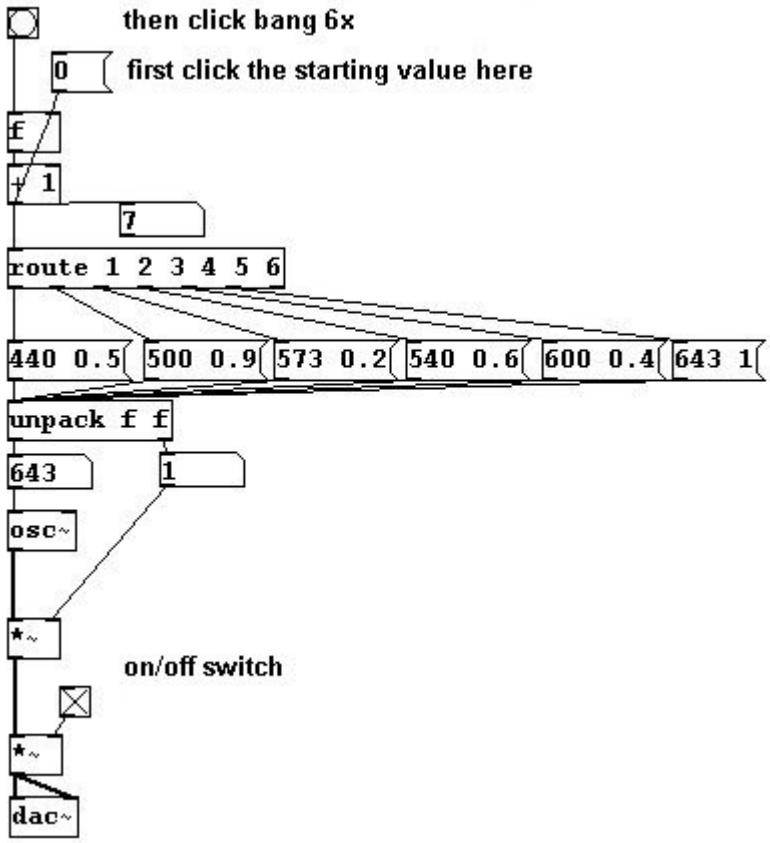


c) Use "expr" para calcular funciones exponenciales, por ej., $y = x^2$ or $y = x^{(2 + x)}$ or $y = 1 - (2^x)$:
patches/a-3-exponentialfunctions.pd



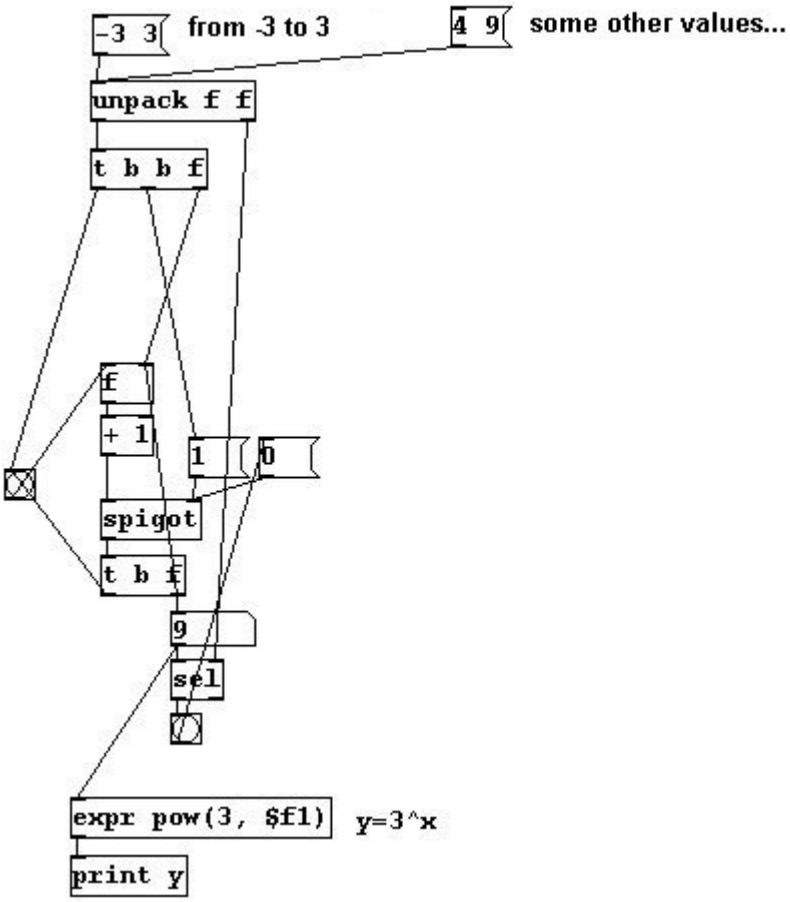
2.2.2.2.6

a) Una secuencia de listas con altura y volumen:
patches/a-4-listsequence.pd



b) Una función que, dada una lista de dos números (el primero y último valor pertenecen al dominio *x*) calcule los valores de *y* -por ejemplo, calcular los valores de la función $y = 3^x$ para el rango $x = -2$ a $x = 4$:

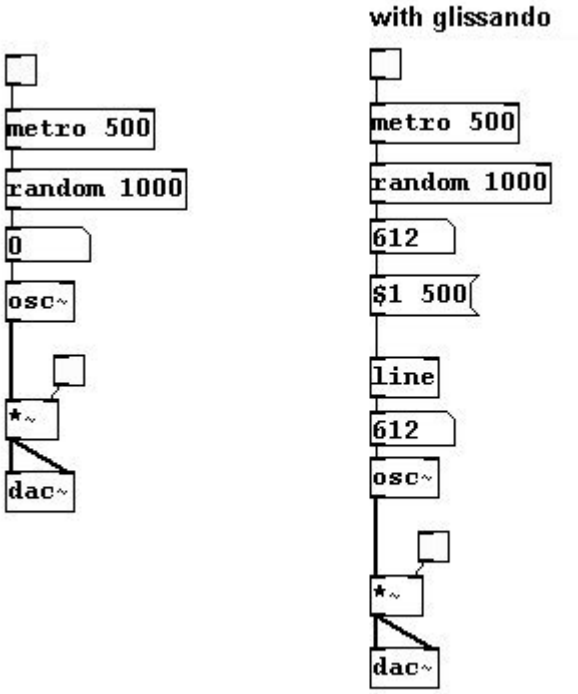
patches/a-5-functionpart.pd



2.2.3.2.9

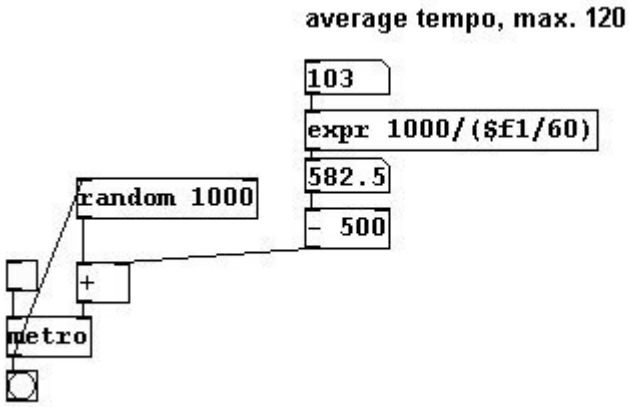
a) Una melodía aleatoria que salte directamente al próximo tono cada 0.5 seconds (alternativamente: como glissando en vez de salto):

patches/a-6-randommelody500.pd



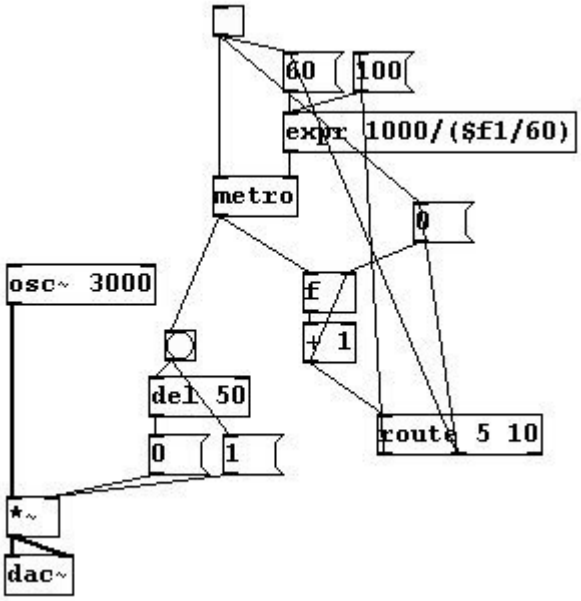
b) Un metrónomo con ritmos irregulares aleatorios (en el cual pueda configurar el tempo promedio):

patches/a-7-irregmetro.pd

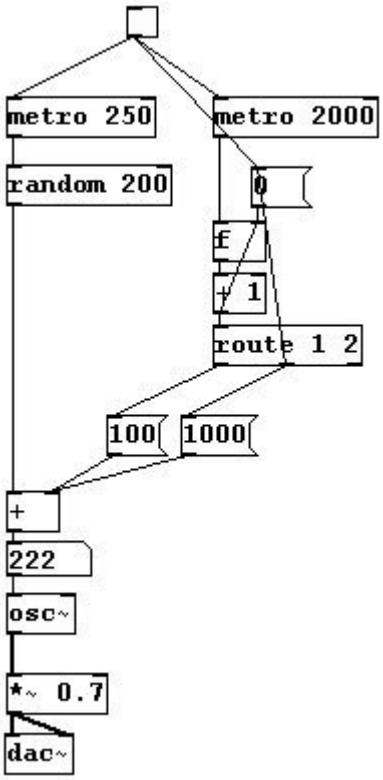


c) Un metrónomo que pulse cinco tiempos a un tempo de negra = 60 y cinco tiempos a negra = 100:

patches/a-8-twometro.pd



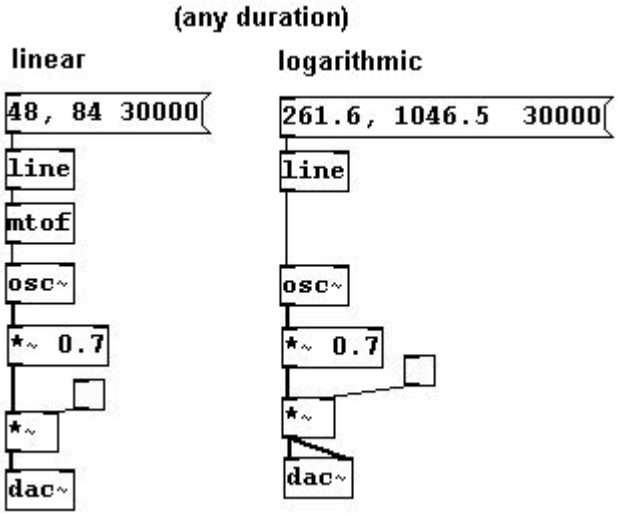
d) Una melodía que alterne entre registros muy agudos y muy graves cada dos segundos:
patches/a-9-highlowmelody.pd



3.1.1.2.2

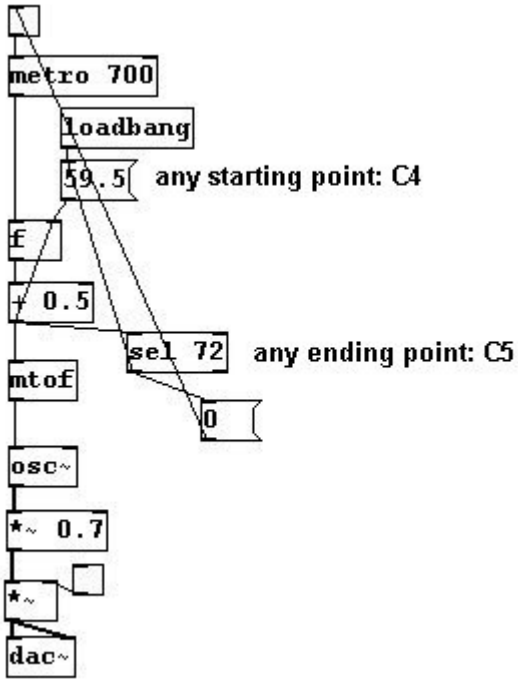
a) Dos glissandos: uno lineal y el otro logarítmico (al oido humano) desde C3 hasta C6.

patches/a-10-linloggliss.pd



b) Una escala de cuartos de tono:

patches/a-11-quarterton.pd

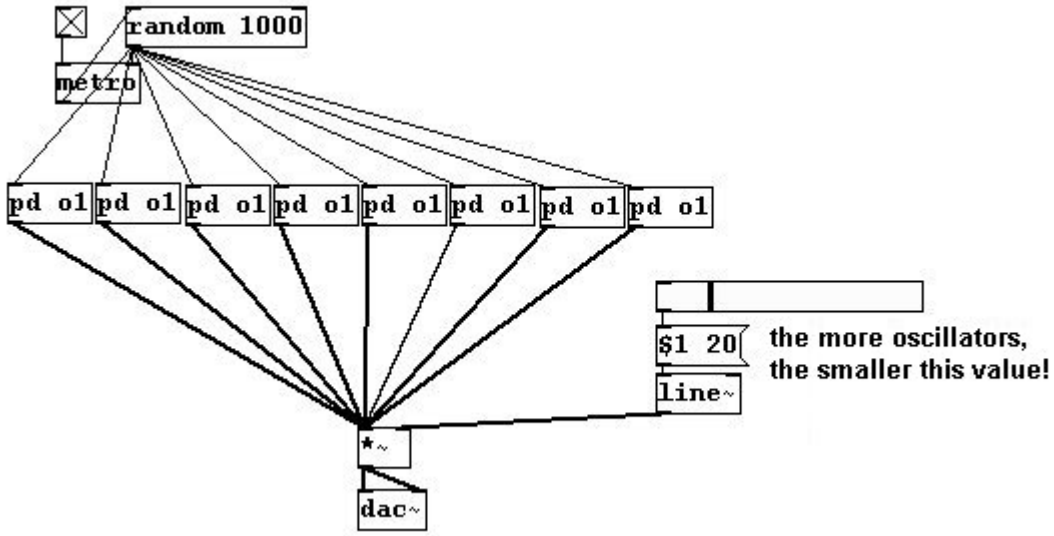


3.1.2.2.5

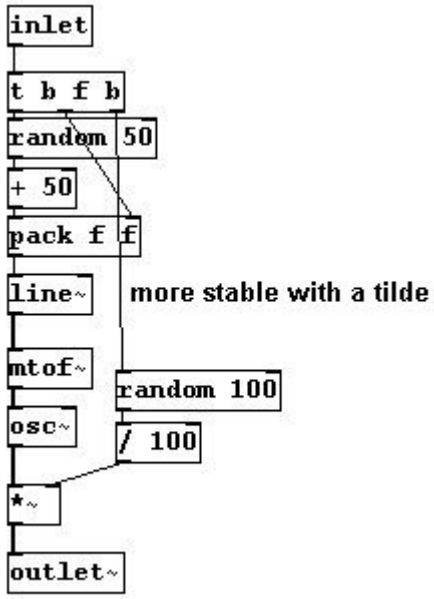
a) Acordes glissando (aleatorios) que también posean cambios de volumen aleatorios para cada tono individual:

patches/a-12-randchord.pd

El patch principal...:

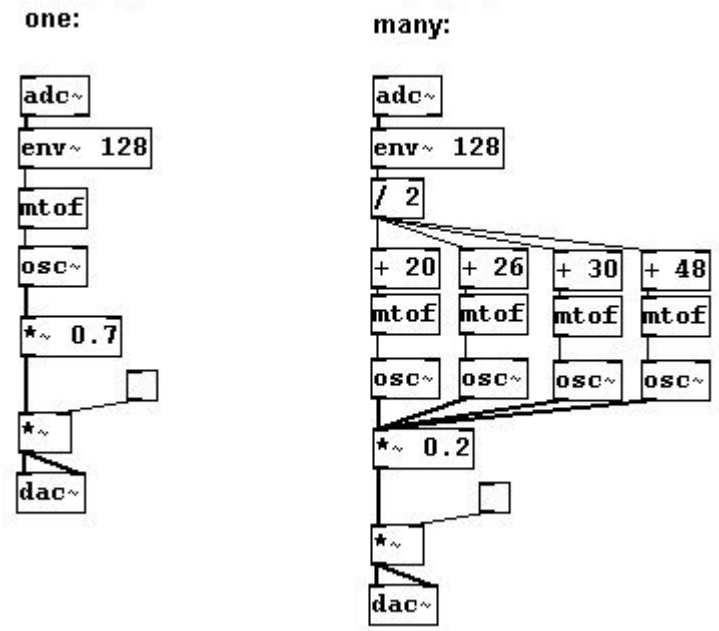


...y el subpatch "o1":



b) El volumen de la entrada de un micrófono controla la altura de un oscilador (alternativamente: muchas alturas de osciladores teniendo diferentes offsets de frecuencia):

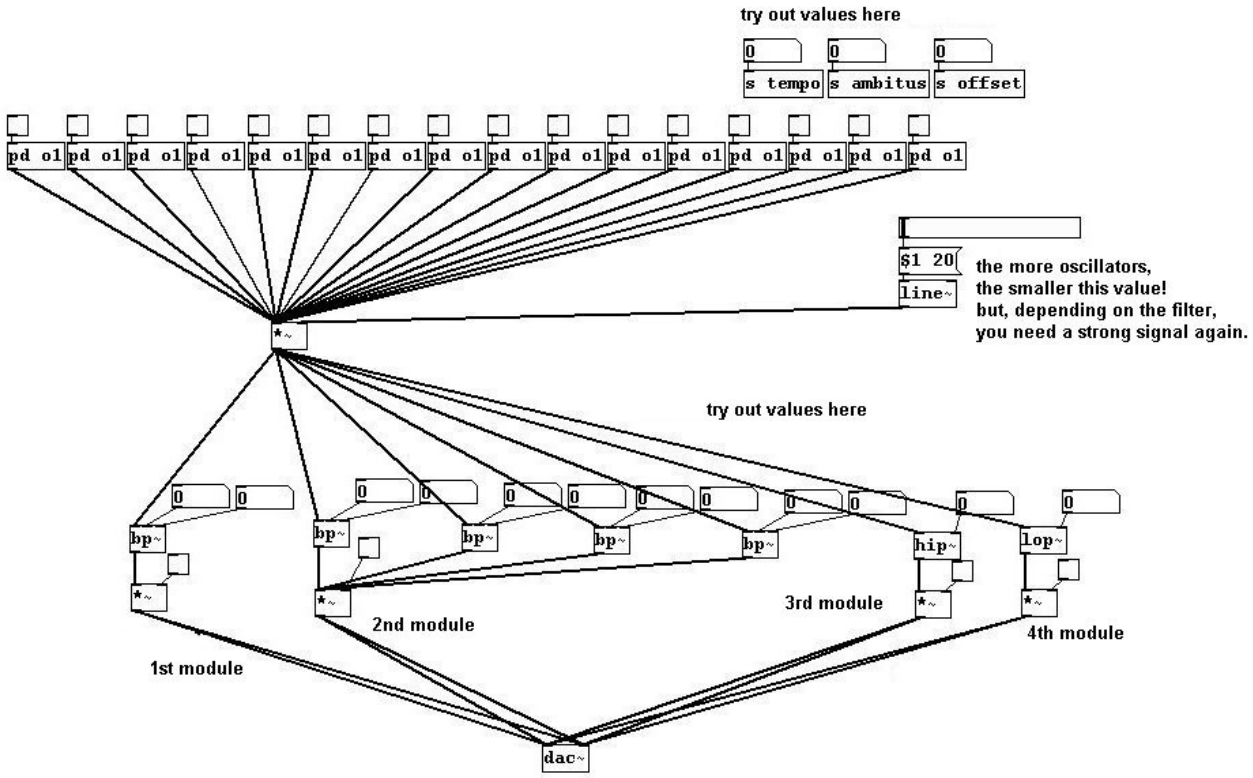
patches/a-13-adcampcont.pd



3.3.2.3

Experimente filtrando la "orquesta de glissando" (3.1.2.2.4):

patches/a-14-orchestrafilter.pd

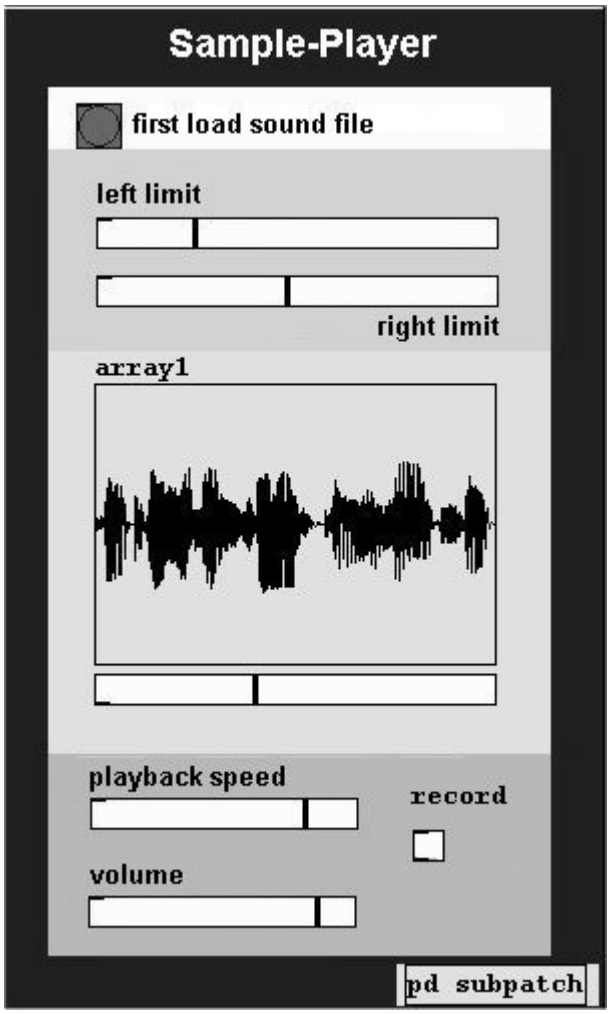


3.4.2.11

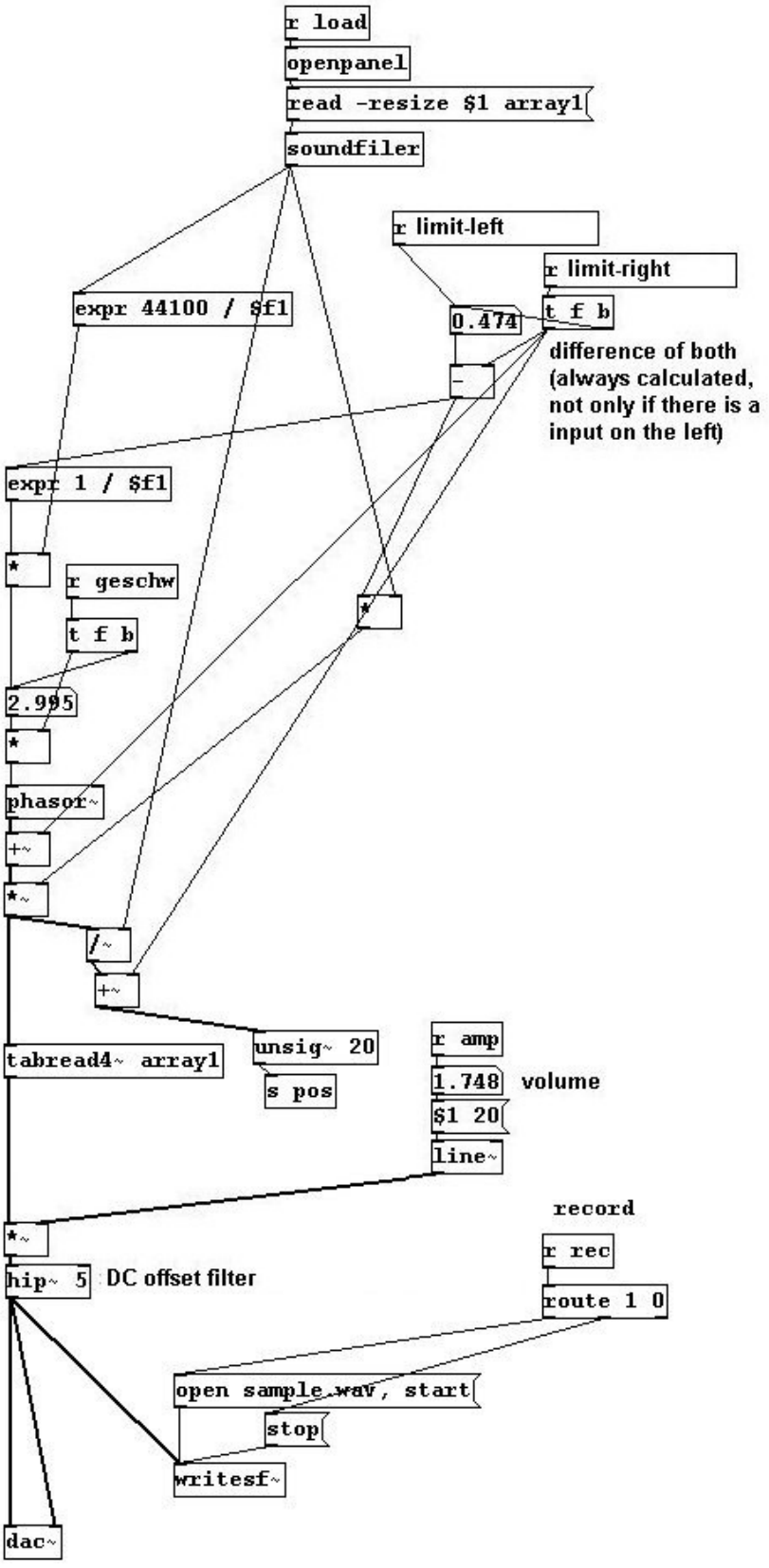
a) Construya una función de grabación dentro del reproductor de muestras:

patches/a-15-recsampler.pd

Un toggle es añadido al patch principal de 3.4.2.4, el cual envía a "rec".

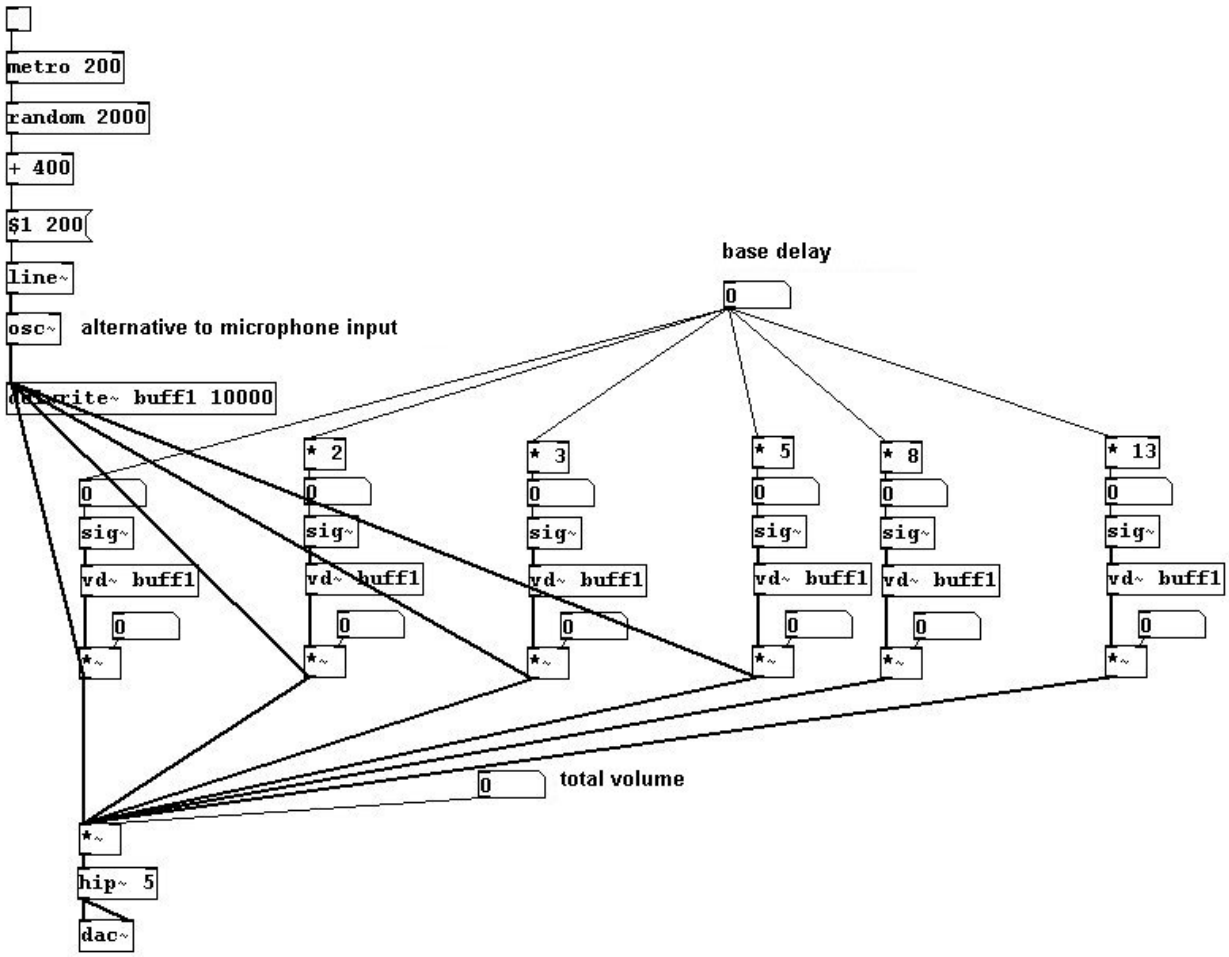


En el subpatch:

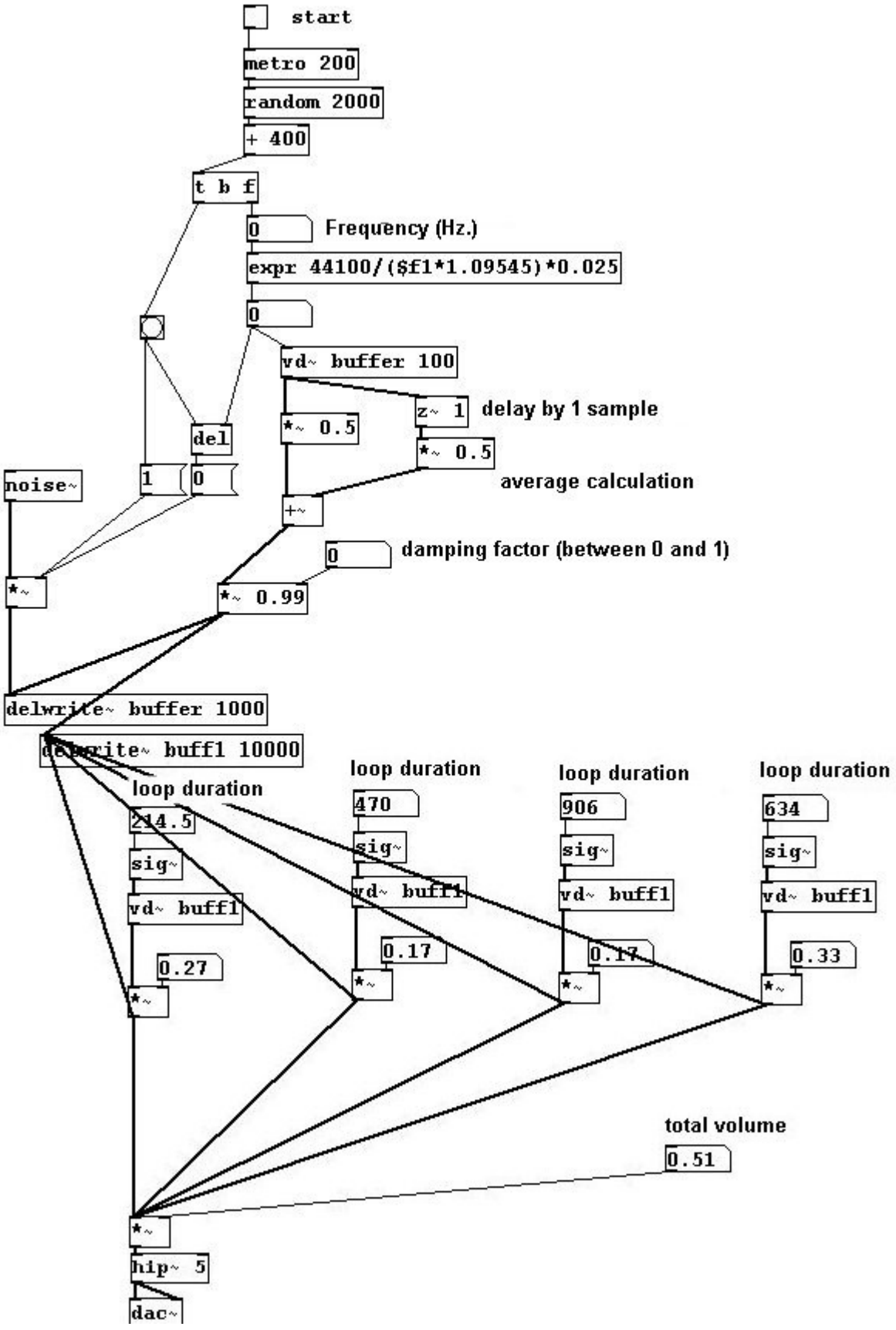


b) Cree un patch para reverberar o una textura con tiempos de retardo diferentes para la señal entrante, por ejemplo, con múltiplos de la serie de Fibonacci (en la cual el próximo número resulta siempre de la suma de los dos anteriores: 0 1 1 2 3 5 8 13):

patches/a-16-fibodelay.pd

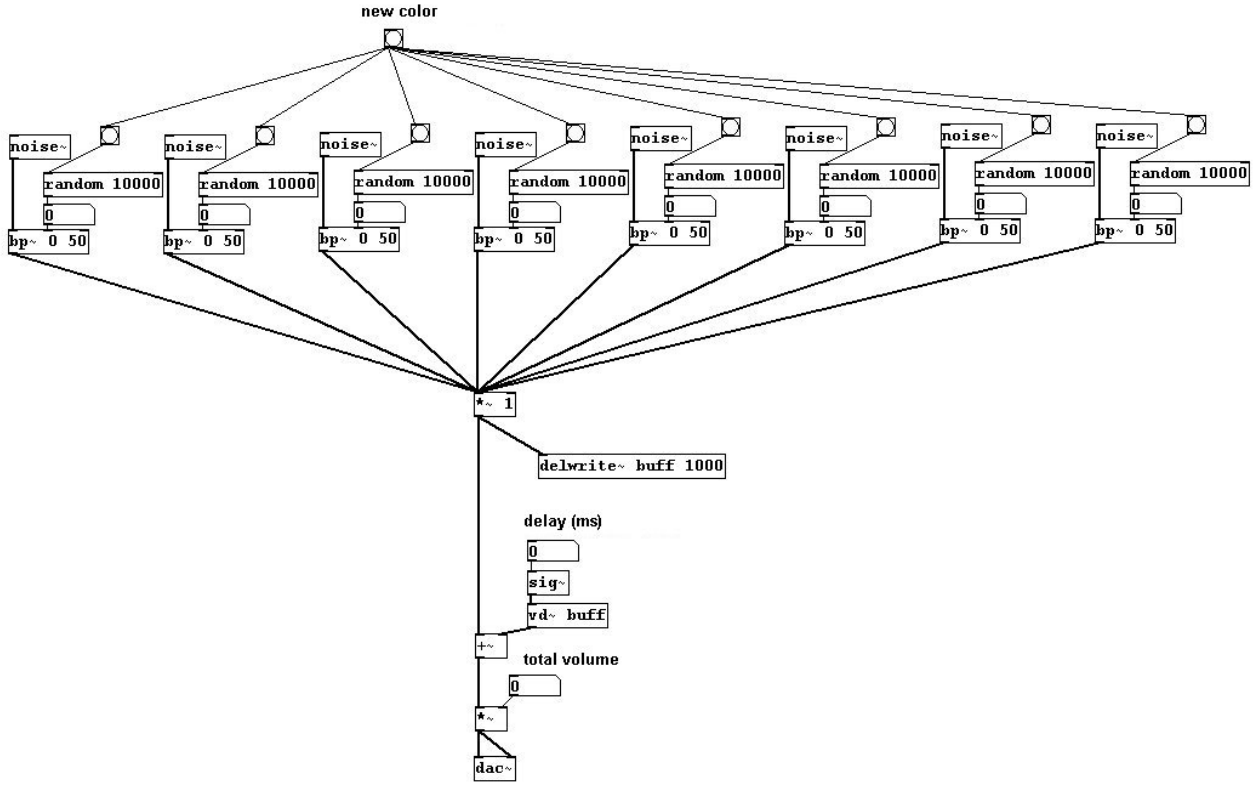


c) Use diferentes sonidos Karplus-Strong para hacer texturas de densidades variables:
patches/a-17-karplus-text.pd



d) Aplique un filtro peine a los patches presentados en las secciones previas:
patches/a-18-combfilteruse.pd

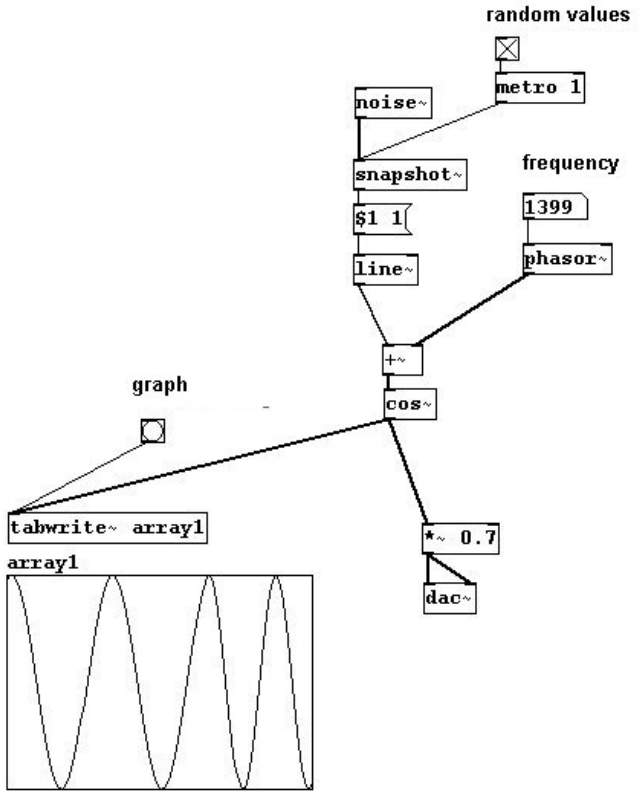
Por ejemplo, usando el "filtro colores" (3.3.2.1) (use un retardo muy corto, por ej., 15 ms):



3.5.2.4

Una onda que cambia constantemente:

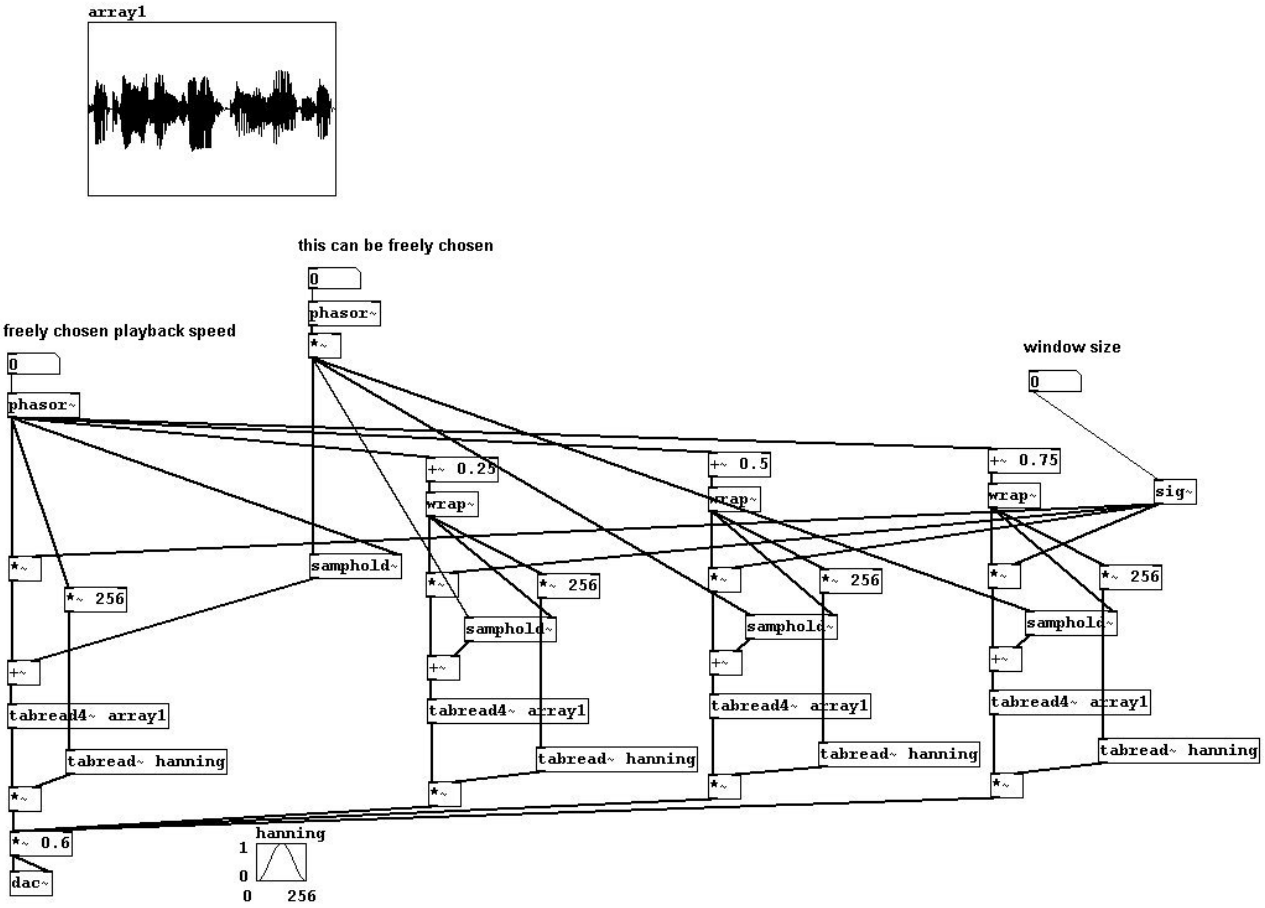
patches/a-19-wavechange.pd



3.7.2.3

Cuatro lectores, cada uno con un tamaño de ventana variable. ¡Experimente!

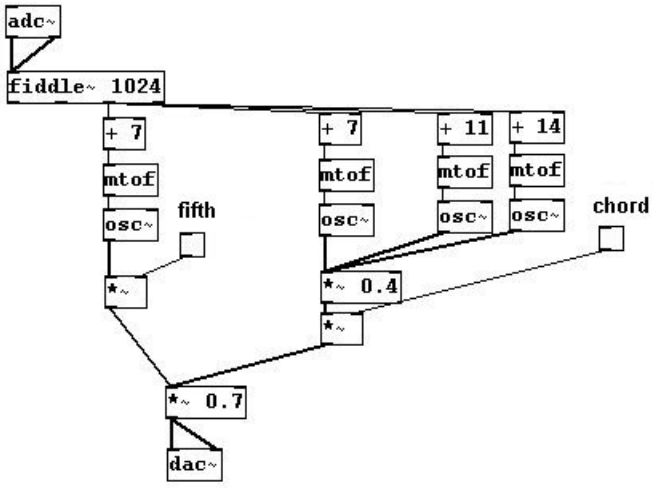
patches/a-20-four-reader.pd



3.8.3.5

En vez de sólo 'seguir' la entrada de micrófono, puede crear una voz paralela a una distancia de quinta perfecta o incluso un acorde paralelo:

patches/a-21-followers.pd



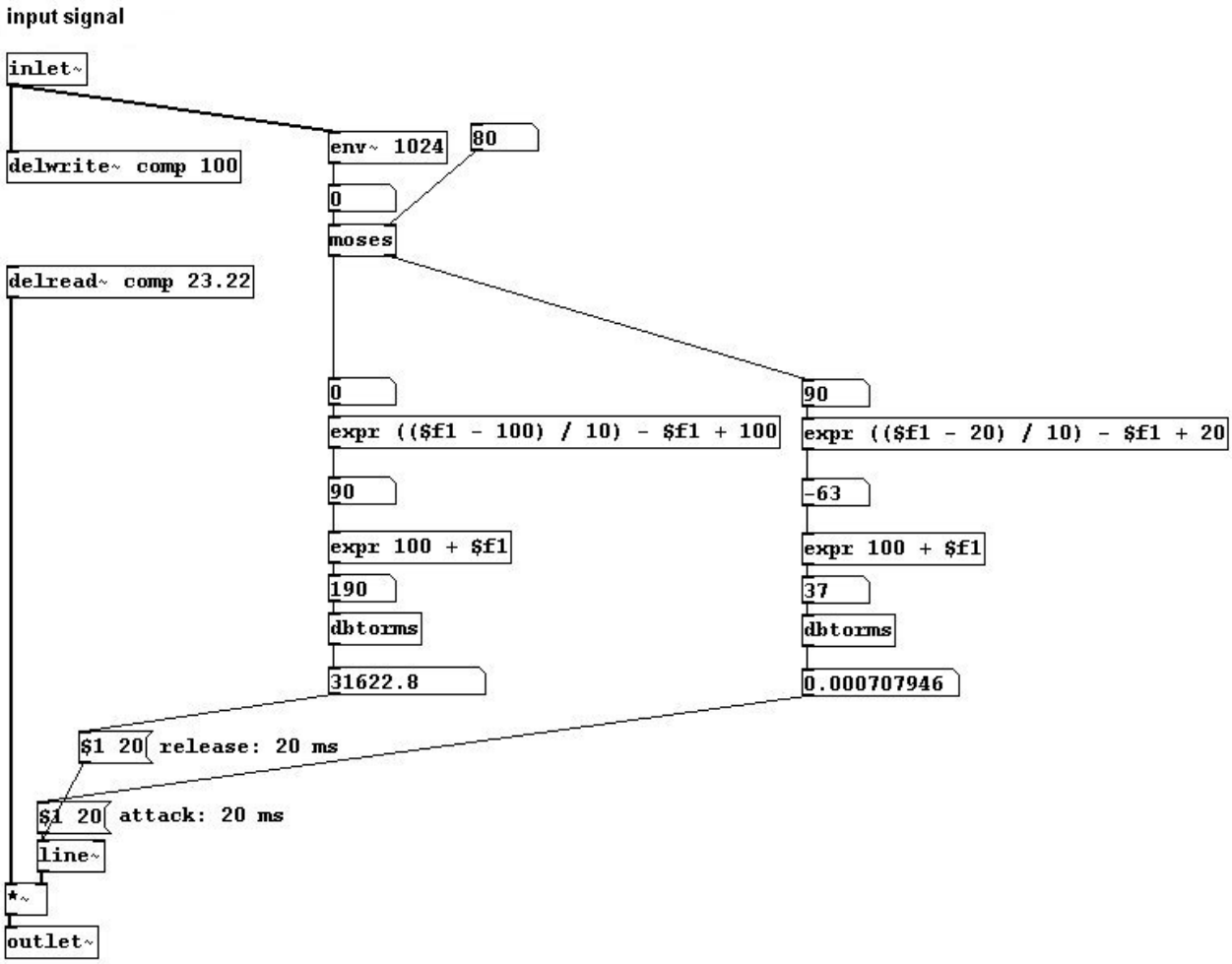
3.9.2.2

a) Un "expansor" – convertir pequeñas diferencias en amplitud en grandes diferencias:

Para esto sólo necesita usar un valor de referencia mayor que la señal de entrada, aplicado al patch de compresor (3.9.1.2).

b) Un inversor de volumen: cambie volumen bajo por alto y viceversa.

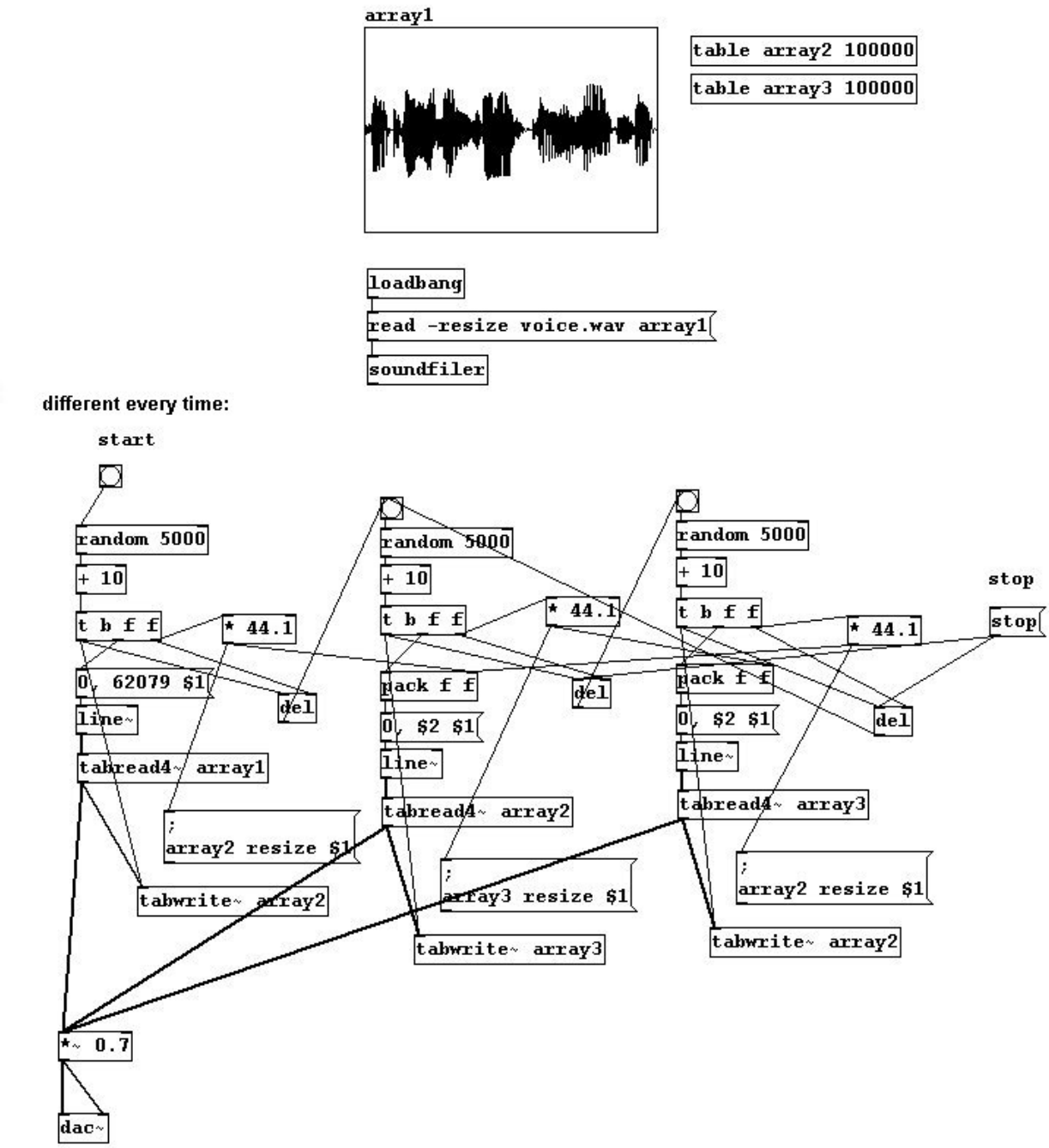
patches/a-22-ampinvert.pd



4.1.2.3

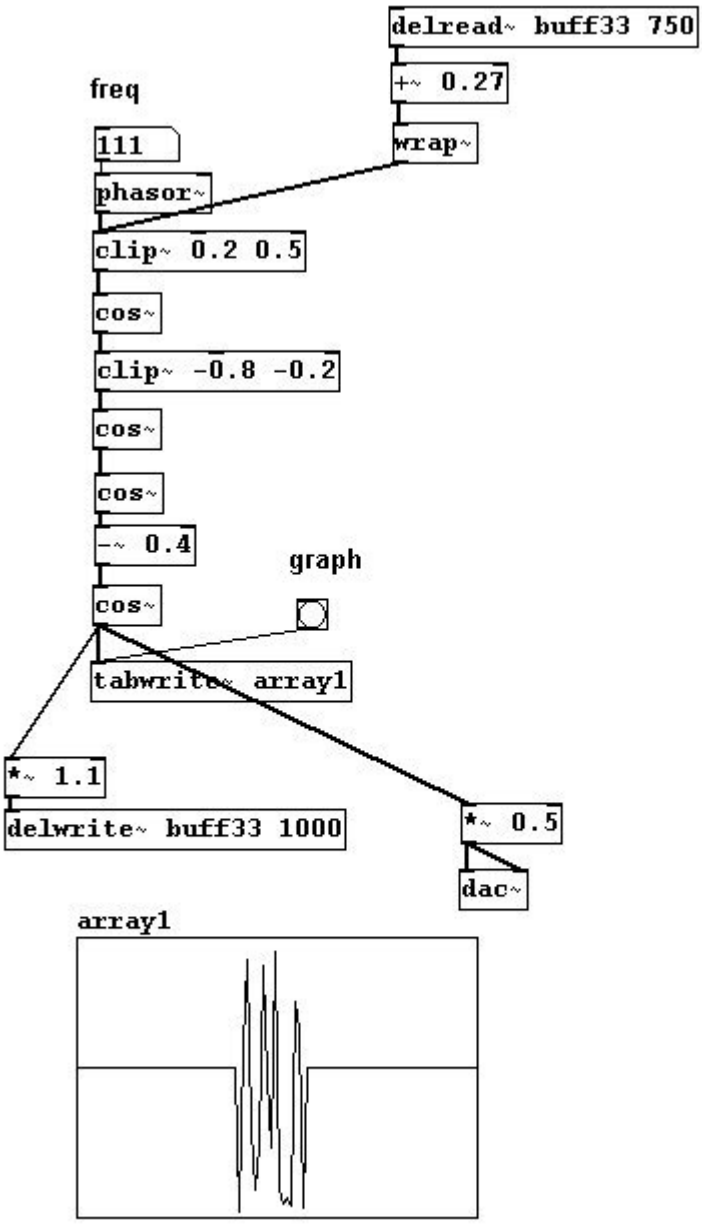
a) Grabe una muestra y reproduzcala a una velocidad erronea. Grabe esta reproducción 'erronea', reproduzcala, grabe y así sucesivamente. Pruebe esto mientras (1) reproduce la muestra con la misma velocidad 'erronea' y (2) con una velocidad 'erronea' diferente.

patches/a-23-sample-false.pd



b) Cree un algoritmo de modelado de onda recursivo utilizando retardo (es decir, un algoritmo en el cual la salida sea retroalimentada como entrada:

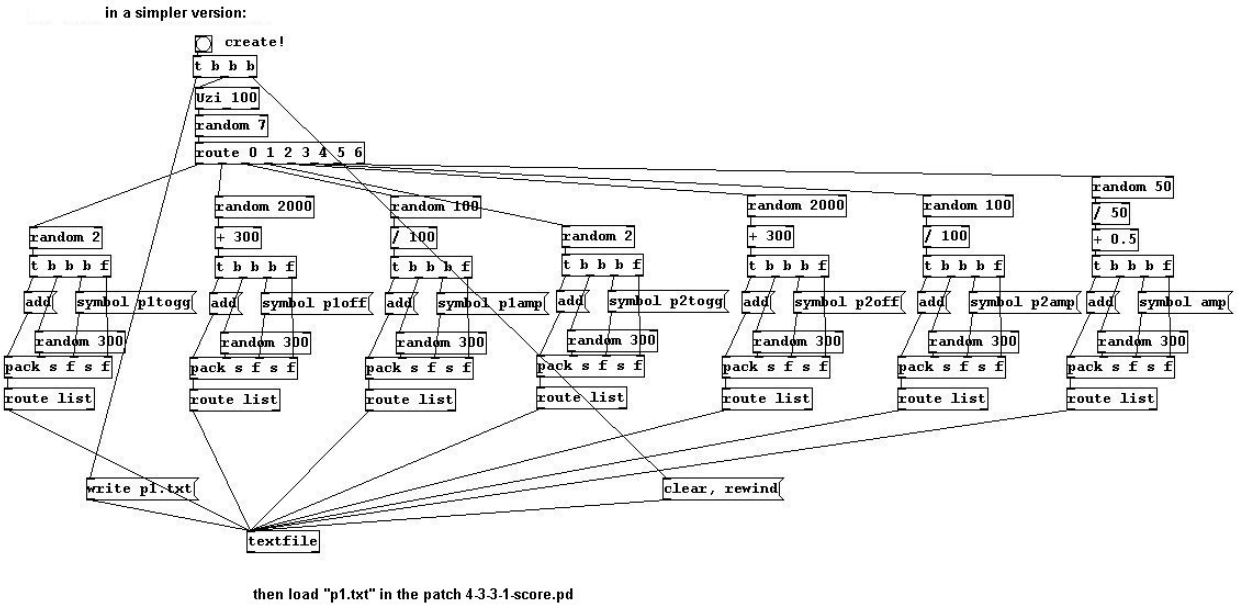
patches/a-24-waveshape-feedback.pd



4.2.2.2

Escriba algoritmos estocásticos dentro de un archivo de texto que use un "qlist" para reproducir el patch de 4.2.2.1 a diferentes velocidades:

patches/a-25-textcreate.pd

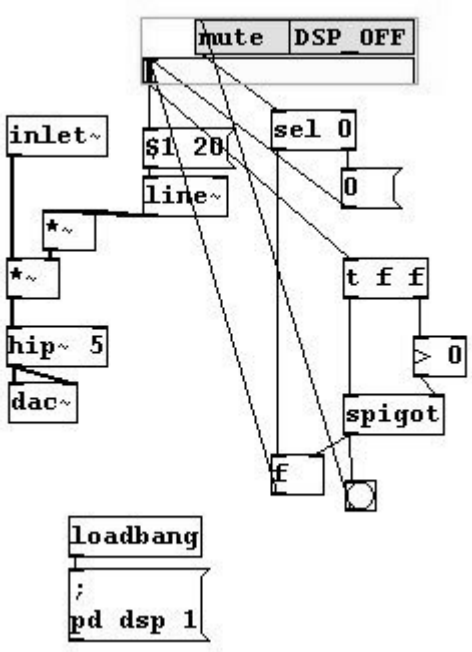


5.1.2.2

Como abstracciones:

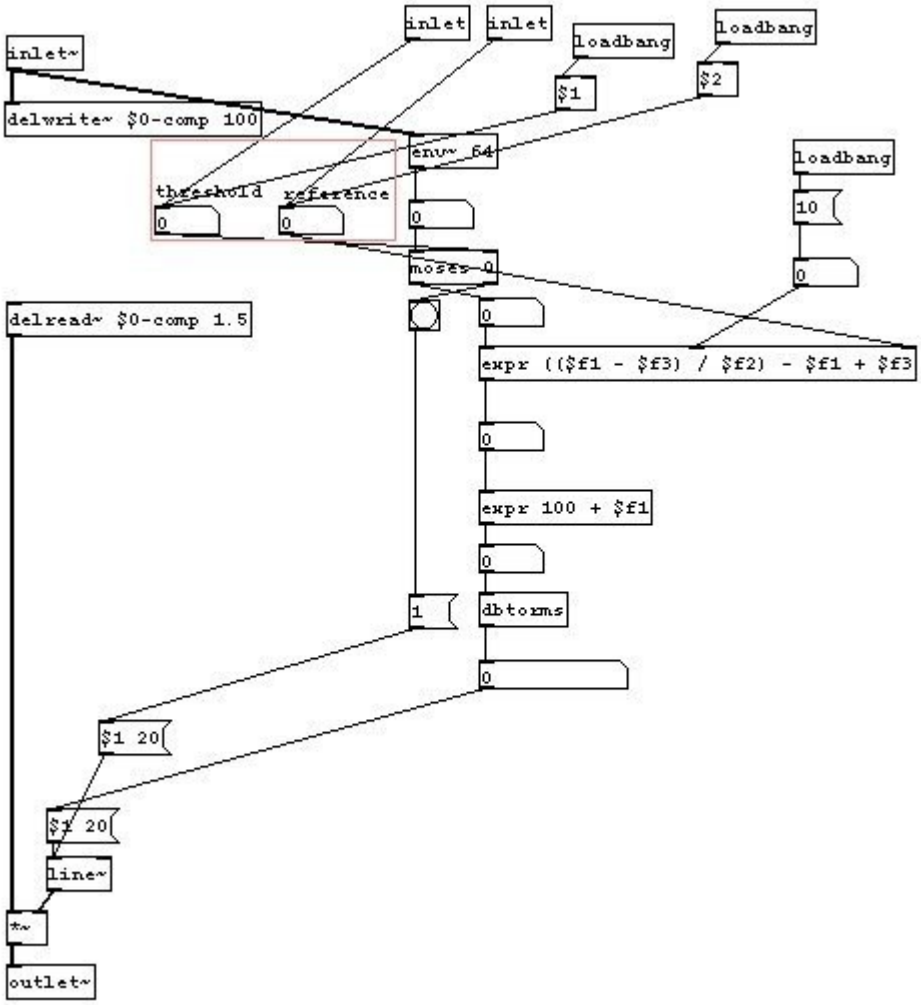
a) Integre el Interruptor de DSP dentro de la abstracción "dac" así como un botón *silencio* encendido/apagado:

patches/a-26-dac-extended.pd



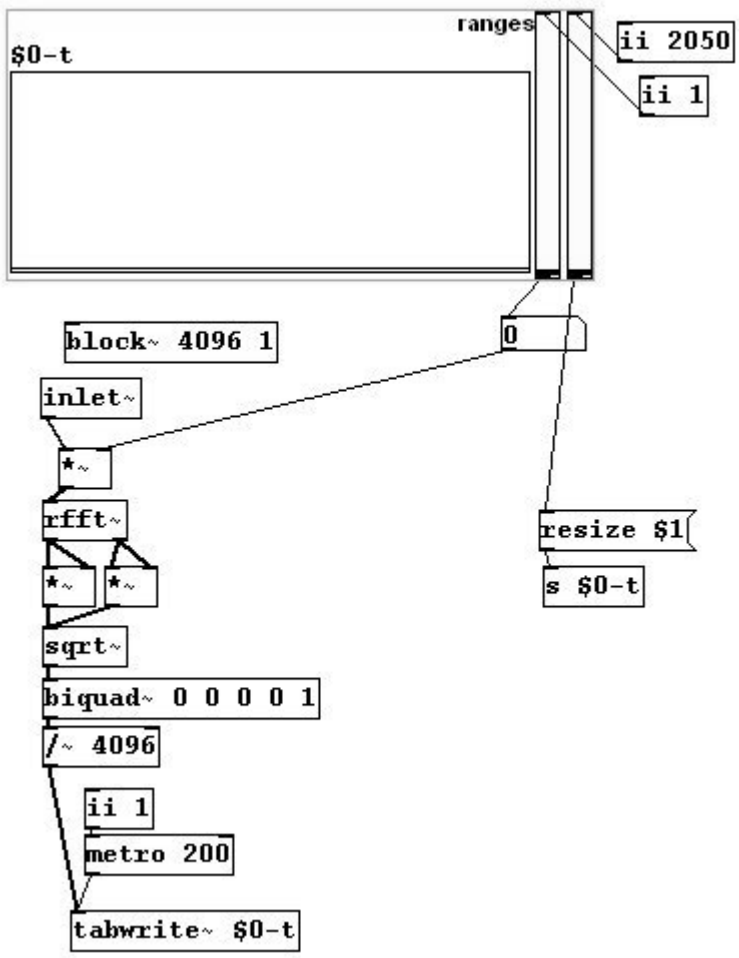
b) Un compresor:

patches/a-27-compress~.pd



c) Una representación gráfica de un espectro entrante:

patches/a-28-spectrum.pd

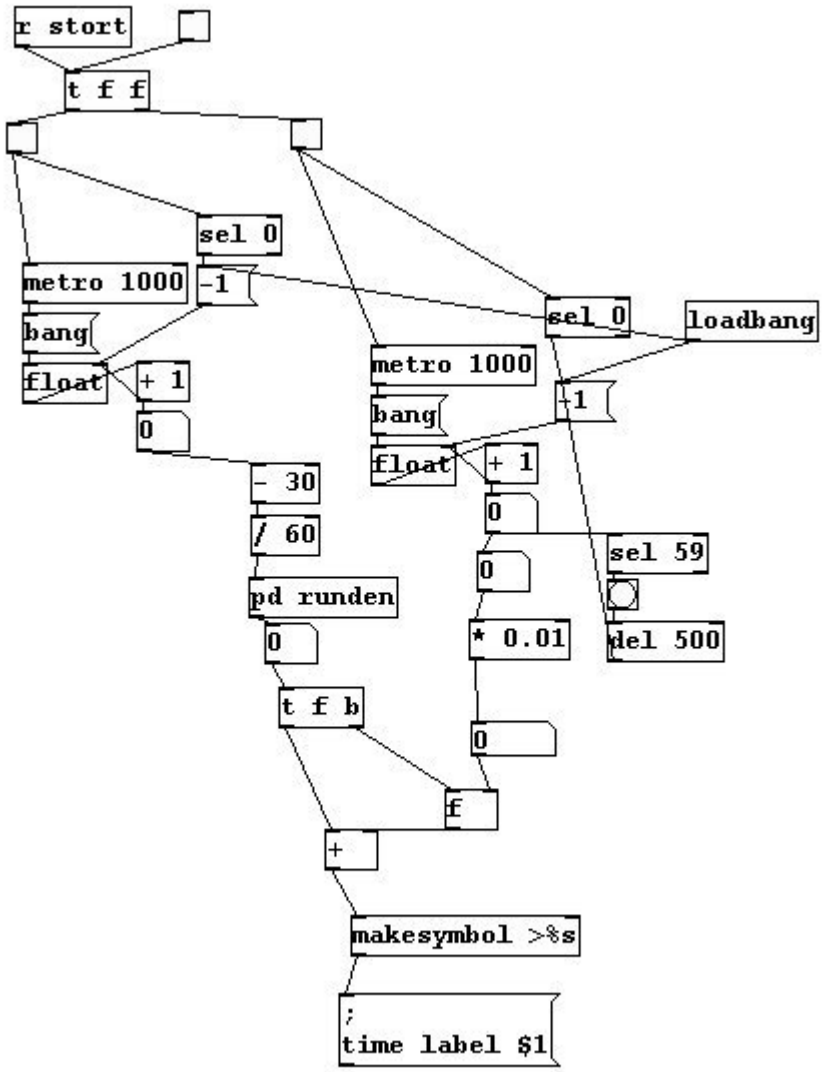


5.2.2.4

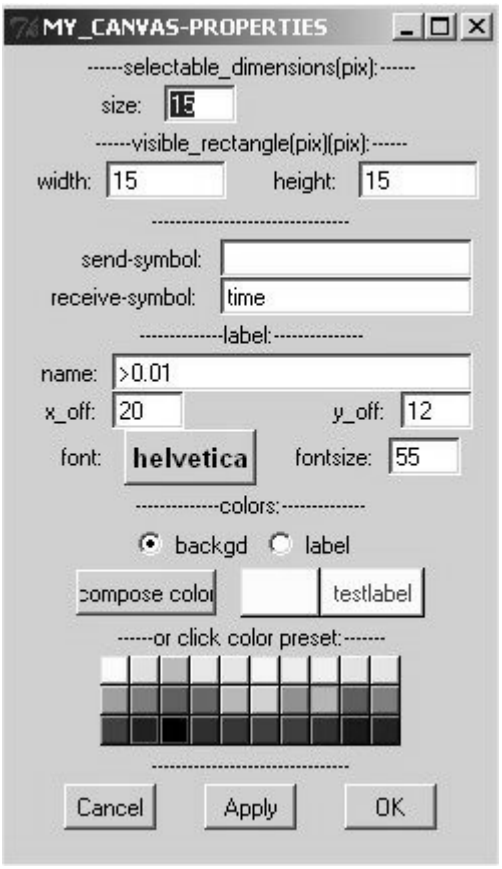
a) Una representación gráfica de un cronógrafo:

patches/a-29-stop.pd

...con el subpatch "stop":



...y la siguiente configuración de lienzo:



El toggle, se encuentra configurado en envío a "stort".

b) Un gráfico para un metro de 5/8, es decir, una guía visual que parpadea. Vea el archivo del patch. patches/a-30-opt-track.pd