Música por computadora

Ernesto Romero y Hernani Villaseñor
Centro Multimedia, México 2012

Sesión 12

Síntesis

Generadores

Los generadores son los encargados de producir sonido, en SC son llamados UGens o unidades generadoras de sonido. Podríamos dividirlos en dos grupos: Osciladores y Ruidos.

Osciladores

Los osciladores son objetos que generan una oscilación, es decir, la onda generada debe de hacer un recorrido pasando por los mismos puntos cada vez. También se les llama ondas periódicas. Un péndulo que oscila entre el punto *a* y el punto *b* es un ejemplo clásico de un oscilador.

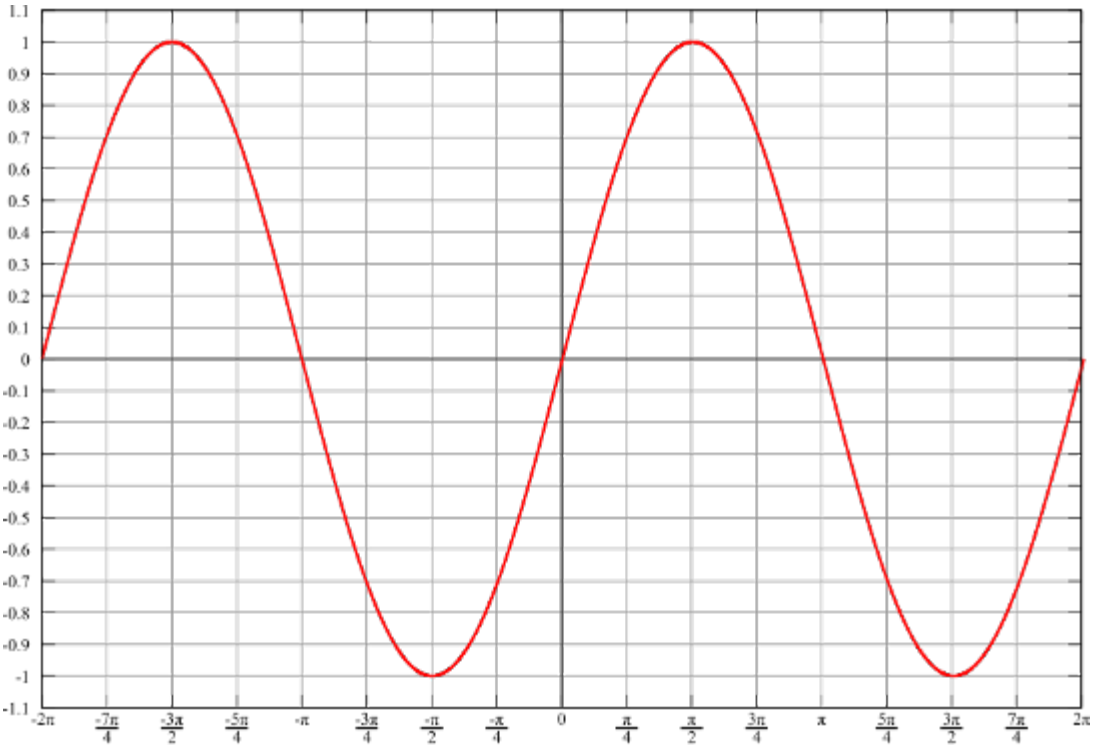
Formas de onda mediante voltaje: los primeros osciladores podían generar formas básicas como sinoidal, triangular o cuadrada, estos osciladores posteriormente fueron incorporados a los sintetizadores mediante un dispositivo llamado VCO (oscilador controlado por voltaje), con el tiempo dejaron de ser operados por voltaje y comenzaron a ser producidos de manera digital.

SC lo que hace es mandar al procesador una serie de funciónes matemáticas que le ordenan generar sonido. Así que parte de la programación de SC consiste en una serie de algoritmos diseñados para producir diferentes tipos de sonido electrónico mediante sus UGens.

Los UGens en SC se dividen en: fuentes: periódico y aperiódico, filtros, distorsión, paneo, reverbs, delays y ugens de buffer, síntesis granular, control: envolventes, triggers, contadores, compuertas, lapso, *decays*, espectrales.

Los osciladores en SC son considerados como fuentes periódicas de sonido.
Entre los osciladores más comunes están los de onda sinusoidal, cuadrada, triangular y de sierra.

Onda Sinusoidal
Aplica la función seno para generar una onda curva cuya amplitud oscila entre 1 y -1. Las fases de la onda se expresan en radianes y tienen un rango entre 0 y 2pi. La fase es un punto de la onda en particular.



Jean Baptiste Fourier demostró que cualquier onda periódica se puede formar a partir de la suma de una o más ondas sinusoidales[1].

La onda sinusoidal no produce armónicos. Es una onda con un timbre simple. Este tipo de ondas no se encuentran en la naturaleza ya que los sonidos naturales siempre vienen acompañados de armónicos. La carencia de armónicos de las ondas sinusoidales hacen que no podamos oírlas en frecuencias menores de 20 Hz (Aunque en la práctica la frecuencia más baja audible es de alrededor de 38 Hz).

En SC generamos una onda sinusoidal con el objeto SinOsc y sus argumentos son los siguientes:

SinOsc.ar(frecuencia, fase, multiplicación, adición)

frecuencia - Todos los osciladores tienen como primer argumento la frecuencia. De igual modo el SinOsc. Se expresa en hertz y determina los ciclos por segundo que realizará la onda. Rango audible entre 20 y 20,000 hz (teóricamente).

fase - Se expresa en radianes (0, 2pi) y determina el punto en el que iniciará la onda.

[1] Computer Music - Synthesis, Composition and Performance. Charles Dodge, Thomas A. Jerse. Pag. 47

multiplicación - Un número que multiplica a la amplitud de la sinoidal. Normalmente la amplitud va de 1 a -1. Si , por ejemplo ponemos como argumento del mul el número cinco nuestra amplitud resultante será de -5 a 5.

adición - Un número que se suma a la amplitud de la sinoidal. Normalmente la amplitud va de 1 a -1. Si , por ejemplo ponemos como argumento del add el número cinco nuestra amplitud resultante será de 4 a 6.

Todos los UGens tienen como últimos argumentos mul y add. Para saber manejar bien estos argumentos podemos recurrir a la siguiente técnica.

Ponemos entre braquets el rango inicial de la amplitud de nuestro UGen, lo multiplicamos por un número que será nuestro mul y le sumamos otro que será nuestro add. Declaramos la línea y el arreglo que obtenemos será la amplitud resultante.

Ejemplo

[-1, 1] * 5 + 5

La amplitud resultante es de 0 a 10. Podemos decir que mul estira la amplitud y add la traslada en el eje Y. Por eso al multiplicar [-1, 1] * 5 estamos estirando la amplitud a [-5, 5]. Al sumarle 5 estamos trasladando la amplitud hhacia arriba 5 unidades obteniendo [0, 10].

Onda Cuadrada

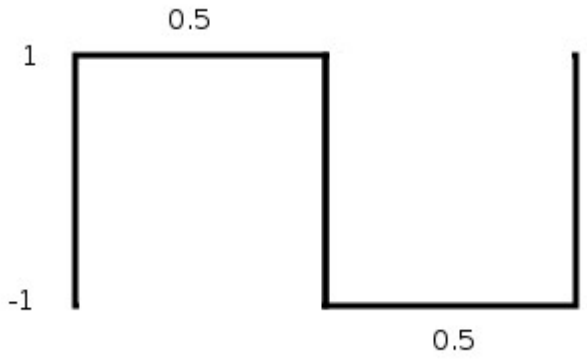
La onda cuadrada brinca de 1 a -1 en cada ciclo sin pasar por los números intermedios. La onda cuadrada arroja solo armónicos impares. Por ejemplo, si tenemos una onda cuadrada a 440 Hz estará compuesta por las siguientes frecuencias:

440 * 1 = 440
440 * 3 = 1320
440 * 5 = 2200
440 * 7 = 3080
440 * 9 = 3960
440 * 11 = 4840
.
.
.
etc

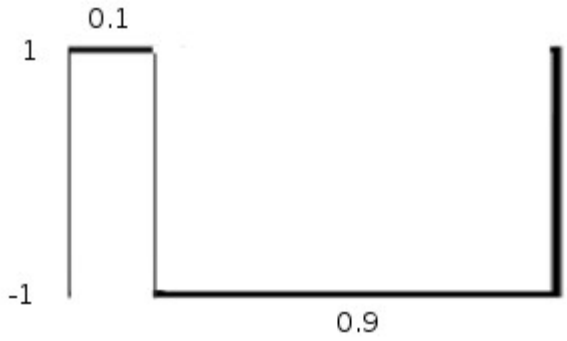
Mientras más definido sea el ángulo recto de la onda mas armónicos tendrá.

El hecho de que la onda cuadrada tenga armónicos nos permite que las escuchemos por debajo de los 20 hz. Atención: no estamos escuchando 20 Hz, sino sus armónicos.

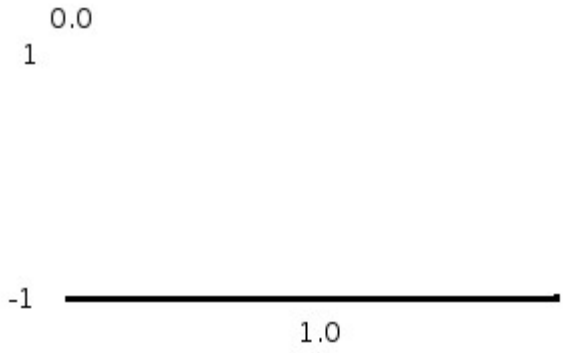
El tiempo que la onda está en 1 o en -1 en cada ciclo se le llama ancho del pulso y determina cambios en el timbre que percibimos. El ancho total de la onda es de 1 y representa la suma del tiempo que está en 1 más el que esta en -1. Por defecto SuperCollider establece el ancho de la onda en 0.5 se refiere a la parte de la onda que está en en -1. Esto hace que la parte de la onda que está en 1 quede en 0.5 también teniendo una proporción de 0.5/0.5.



Ancho de pulso = 0.5



Ancho de pulso = 0.9



Ancho de pulso = 1.0

El UGen que SuperCollider emplea para hacer ondas cuadradas es Pulse.ar y esta es su sintaxis.

Pulse.ar(frecuencia, ancho de pulso, multiplicación, adición)

frecuencia - Expresada en hertz. Podemos usar valores menores de 20hz.

ancho de pulso - rango entre 0 y 1. Si ponemos el ancho en 1 la parte positiva quedara en 0 por lo que no habrá oscilación y no sonara nada. Lo mismo ocurre si ponemos 0 en ancho de pulso. El oído aprecia igual los valores de ancho de forma simétrica a partir del 0.5, de esta forma oiremos igual un ancho de 0.25 que de 0.75.

multiplicación - Un numero que multiplica a la amplitud de la onda.

adición - Un numero que se suma a la amplitud de la onda y la traslada en el eje de las y.

Podemos escuchar las variaciones tímbricas que se obtienen al cambiar el ancho de banda de un Pulse insertando un MouseX en el argumento correspondiente.

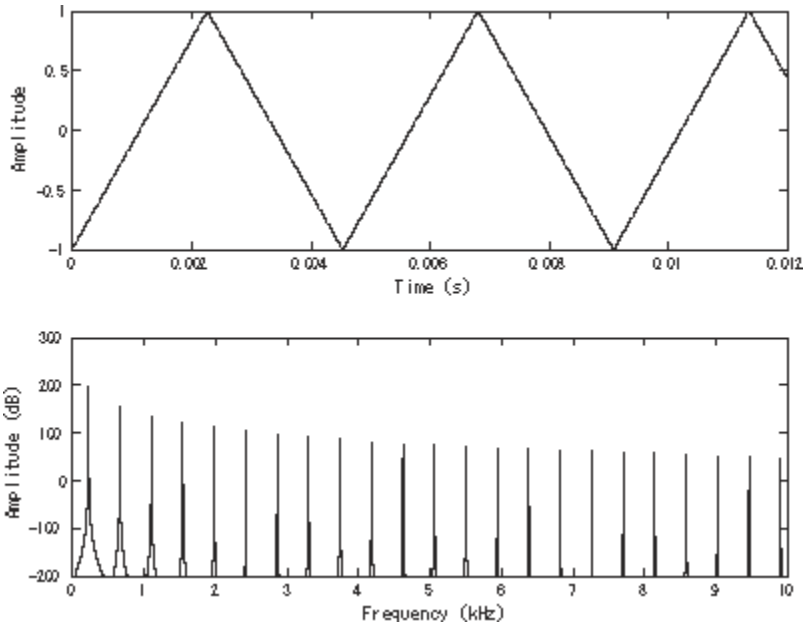
```
{Pulse.ar(440, MouseX.kr(0,1).poll)}.scope
```

Observen como el timbre es el mismo cuando estamos de un lado o del otro del 0.5.
El mensaje .poll nos ayuda a imprimir en la post window los valores de un UGen, en este caso el MouseX

Los Ugens en SC son aproximadamente 250, utiliza la ayuda para estudiar los UGens de tu interés.

Onda Triangular

La onda triangular presenta armónicos impares también pero decrece su amplitud con el inverso del cuadrado del numero armónico correspondiente. así la amplitud del armónico 5 será de 1/25. Es por esto que su espectro es mucho mas simple que el de la onda cuadrada.



En SUperCollider la onda triangular se genera con el objeto LFTri y esta es su sintaxis:

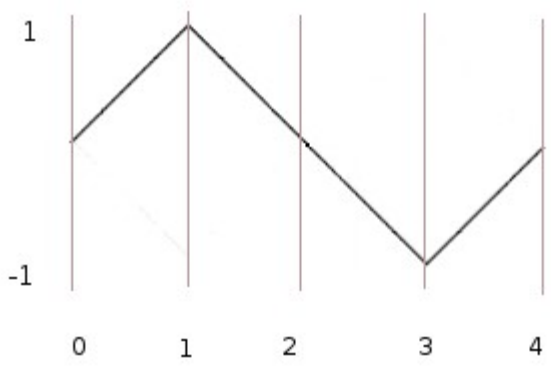
LFTri.ar(frecuencia, fase, multiplicación, adición)

frecuencia - Expresada en hertz. Podemos usar valores menores de 20hz.

fase - El punto en el que iniciara la onda. Los valores pueden ser entre 0 y 4.

multiplicación - Un numero que multiplica a la amplitud de la onda.

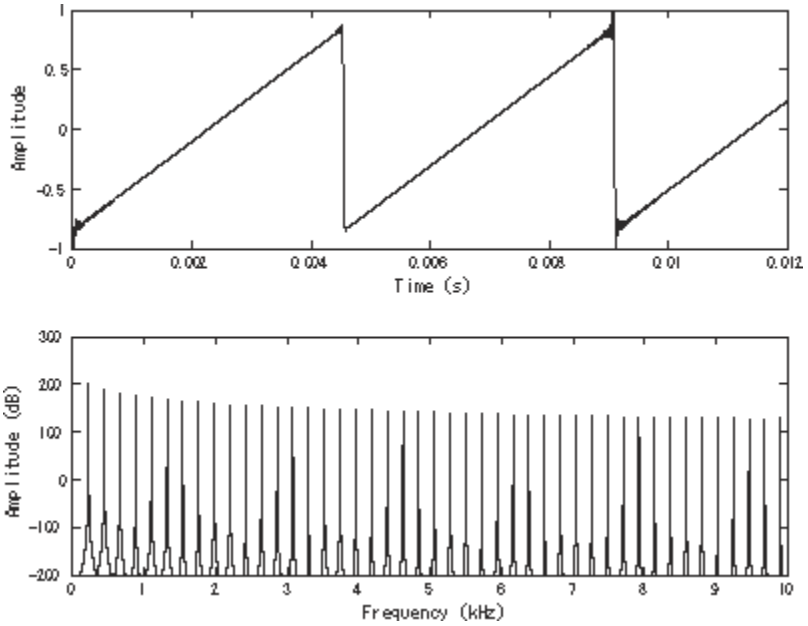
adición - Un numero que se suma a la amplitud de la onda y la traslada en el eje de las y.



fases de la onda triangular

Onda de Sierra

La onda de sierra contiene en su espectro a todos los armónicos. Su forma es parecida a los dientes de una sierra y por eso se le llama así.
Por tener esta riqueza tímbrica la onda de sierra es usada a veces para hacer sintesis sustractiva y obtener sonidos como el de las cuerdas.



SuperCollider utiliza la clase Saw para generar esta onda. Esta es su sintaxis:

Saw.ar(frecuencia, multiplicación, adición)

frecuencia - Expresada en hertz. Podemos usar valores menores de 20hz.

multiplicación - Un numero que multiplica a la amplitud de la onda.

adición - Un numero que se suma a la amplitud de la onda y la traslada en el eje de las y.

Notar que SuperCollider no posee un argumento para la fase de la onda de sierra

Ruidos

Los ruidos generan amplitudes de manera no oscilatoria. Son muy usados para hacer mediciones o como fuente primaria para cierto tipo de síntesis, en especial la síntesis sustractiva. Algunos ruidos se forman al generar todas las frecuencias al mismo tiempo. La diferencia de fuerzas entre las frecuencias nos da diferentes tipos de ruido. Uno de los ruidos más comunes es el ruido blanco o *White Noise* y no es otra cosa que todas la frecuencias sonando al mismo tiempo con la misma fuerza. El ruido rosa o *Pink Noise* disminuye la fuerza de sus frecuencias 3 decibeles por octava. El ruido café o *Brown Noise* lo hace 6 decibeles por octava.

Algunos de los UGens que tiene SC para generar estos ruidos son los siguientes. Observar que sus argumentos son solo de amplitud ya que no existen otros parámetros.

```
{WhiteNoise.ar(1)}.scope;
{PinkNoise.ar(1)}.scope;
{BrownNoise.ar(1)}.scope
```

Podemos observar el espectrograma de los UGens con la siguiente linea:

```
FreqScope.new
```

Ruidos aleatorios

Otros formas de hacer ruido es generando amplitudes de manera aleatoria. Algunas clases de SC que hacen esto son las de tipo LFNoise las cuales generan amplitudes aleatorias entre -1 y 1 con cierta frecuencia. Los cambios entre las amplitudes pueden darse de brinco, es decir de manera discreta, (LFNoise0), con una interpolación lineal (LFNoise1), o con interpolación cuadratica curva (LFNoise2).

```
{LFNoise0.ar}.plot // ruido a escalón, genera cambios discretos
{LFNoise1.ar}.plot// ruido en rampa, genera cambios continuos lineales
{LFNoise2ar}.plot// ruido cuadrático, genera cambios continuos curvos
```

La sintaxis de estos UGens es igual para todos, Veamos la de LFNoise0

LFNoise0.ar(frecuencia, multiplicación, adición)

frecuencia - La cantidad de amplitudes aleatorias por segundo entre 1 y -1. Expresada en hertz. Podemos usar valores menores de 20hz. No es una frecuencia de oscilación

multiplicación - Un numero que multiplica a la amplitud de la onda.

adición - Un numero que se suma a la amplitud de la onda y la traslada en el eje de las y.

Otro UGen que tiene SuperCollider para hacer ruido es el Dust. Genera pulsos aleatorios entre 1 y 0. A diferencia de los LFNoise, que generan amplitudes con una frecuencia definida, el Dust lo hace con una densidad aleatoria, obteniendo una frecuencia estadística. Es decir que si Dust tiene una densidad de 1 obtendremos aproximadamente 1 pulso por segundo. A veces mas, a veces menos.

Dust.ar(densidad, multiplicación, adición)

también es posible generar ruidos por medios distintos. Por ejemplo este código genera un ruido Browniano, de tal forma que se mueve una décima de amplitud cada vez hacia arriba o hhacia abajo aleatoriamente.

```
(
SynthDef(\browniano, {|amp=0|
                var sig;
                sig=SinOsc.ar(0,0,1,amp);
                Out.ar(0,sig);
                }).send(s);
)

~browniano=Synth(\browniano)

(
Tdef(\browniano, {var amp=1;
                inf.do{
                amp=((amp+([1,0, -1]*0.1).choose)%3) -1;
                ~browniano.set(\amp, amp.postln);
                0.001.wait;
                }
                }).quant_(0)
)
```

```
Tdef(\browniano).play
Tdef(\browniano).stop

s.scope
```

también podemos generar ruido usando arreglos Demands:

aquí generamos amplitudes aleatorias de 1, 0 o -1 periódicamente usando un Impulse a 1000 hz

{Demand.ar(Impulse.ar(1000),0,Drand([1,0, -1],inf))}.scope

aquí generamos las mismas amplitudes pero en secuencia. Lo que hace que suene como ruido es el Dust que esta pidiendo estas amplitudes sin una periodicidad.

{Demand.ar(Dust.ar(1000),0,Dseq([1,0, -1],inf))}.scope

Síntesis Aditiva

La síntesis aditiva no es más que una técnica donde se suman tonos sinoidales. prácticamente cualquier sonido puede ser descompuesto en tonos puros (sinoidales). Así que, de manera inversa, agregando tonos sinoidales se puede construir un timbre.

Hay tres parámetros que afectan la relación de los sonidos que vamos a sumar y estos son: la frecuencia, la fase y la amplitud. Anteriormente vimos un ejemplo donde sumamos una serie de armónicos, este ejemplo es síntesis aditiva. También podemos sumar tonos que no tengan relación de múltiplos, como en el caso de las campanas cuyos armónicos no guardan relación matemática de múltiplos.

sine1 + sine2 + sine 3 + sine 4 + sine5 + sineN...

```
// suma de sinoidales con relación armónica
(
{var n=5; SinOsc.ar(220,0,1/n)
+ SinOsc.ar(440,0,1/n)
+ SinOsc.ar(660,0,1/n)
+ SinOsc.ar(880,0,1/n)
+ SinOsc.ar(1100,0,1/n)
}.play
)
```

```
// suma de sinoidales sin relación armónica, ni de fase
(
{var n=5; SinOsc.ar(220,0,1/n)
+ SinOsc.ar(475,0,1/n)
+ SinOsc.ar(527,pi,1/n)
+ SinOsc.ar(679,pi/2,1/n)
+ SinOsc.ar(735,3pi/2,1/n)
}.play
)
```

Hay objetos en SC que nos ayudan a sumar la señal como Mix.ar que mezcla un Array de UGens.

Mix.ar([sine1, sine2, sine3, sine4, sine5])

```
// suma de ondas sinoidales usando Mix, guardando una relación
armónica y de fase
(
{var n=5; Mix.ar([SinOsc.ar(220,0,1/n),
SinOsc.ar(440,0,1/n),
SinOsc.ar(660,0,1/n),
SinOsc.ar(880,0,1/n),
SinOsc.ar(1100,0,1/n)])
}.play
)

// suma de ondas sinoidales usando Mix, sin relación armónica ni
de fase
(
{var n=5; Mix.ar([SinOsc.ar(220,0,1/n),
SinOsc.ar(460,pi,1/n),
SinOsc.ar(665,0,1/n),
SinOsc.ar(840,0,1/n),
SinOsc.ar(1110,3pi/2,1/n)])
}.play
)
```

Podemos generar las ondas de los osciladores primarios usando exclusivamente la suma de ondas sinusoidales.

Onda cuadrada

Generamos sinoides con frecuencias de armónicos impares de una fundamental.

Si declaramos esta línea obtenemos un array del 1 al 40
(1..40)

Al multiplicar los elementos por dos obtenemos pares del 2 al 80
(1..40)*2

Y al restar uno convertimos los pares en impares del 1 al 79
(1..40)*2 -1

Usaremos este arreglo para los de armónicos de la frecuencia fundamental 440. Es necesario disminuir la amplitud de cada armónico para que no se sature la señal.

{var arm=(1..40)*2 -1; Mix(SinOsc.ar(440*arm,0,1/arm))}.scope

El resultado es muy similar a

{Pulse.ar}.scope

Onda Triangular

A partir de sinusoidales con frecuencias de los armónicos impares y cambiando cada (4n -1) armónicos la fase a pi, donde n es el número de armónico. Las amplitudes deben de ser el inverso del cuadrado del número de armónico, igual que con la onda cuadrada

```
(
~arm=(1..80)*2 -1;
~armFase=(4*~arm -1);
~arm.size.do{|i|
                if(~armFase.find([~arm[i]])==nil, {
            {SinOsc.ar(440*~arm[i],0,1/(~arm[i]**2))}.play;
                                    },
                                    {
            {SinOsc.ar(440*~arm[i],pi,1/(~arm[i]**2))}.play;
                                    }
                )}
)
```

Onda de Sierra

Se puede generar una onda de sierra sumando sinusoidales con las frecuencias de todos los armónicos de una frecuencia fundamental y disminuyendo la amplitud de cada una por el inverso del numero de armónico

{var arm=(1..100); Mix(SinOsc.ar(440*arm,0,0.5/arm))}.scope

Filtros

Un filtro es un sistema que, dependiendo de algunos parámetros, realiza un proceso de discriminación de una señal de entrada obteniendo variaciones en su salida.

Filtro Pasa Bajas

Es aquel que permite el paso de frecuencias bajas, desde la frecuencia 0 hasta una frecuencia determinada. Esta frecuencia determinada es la frecuencia de corte. Por ejemplo, si determinamos que sea 200 Hz la frecuencia de corte dejaremos pasar todas las frecuencias mas bajas que 200 Hz. Recordemos que no hay frecuencias menores a 0 Hz.

Utilicemos el UGen LPF - Low Pass Filter-- Filtro Pasa Bajas.

Sintaxis : LPF.ar(entrada, frecuencia de corte, multiplicación, adición)

entrada : La señal que queremos filtrar. Tiene que tener .ar.

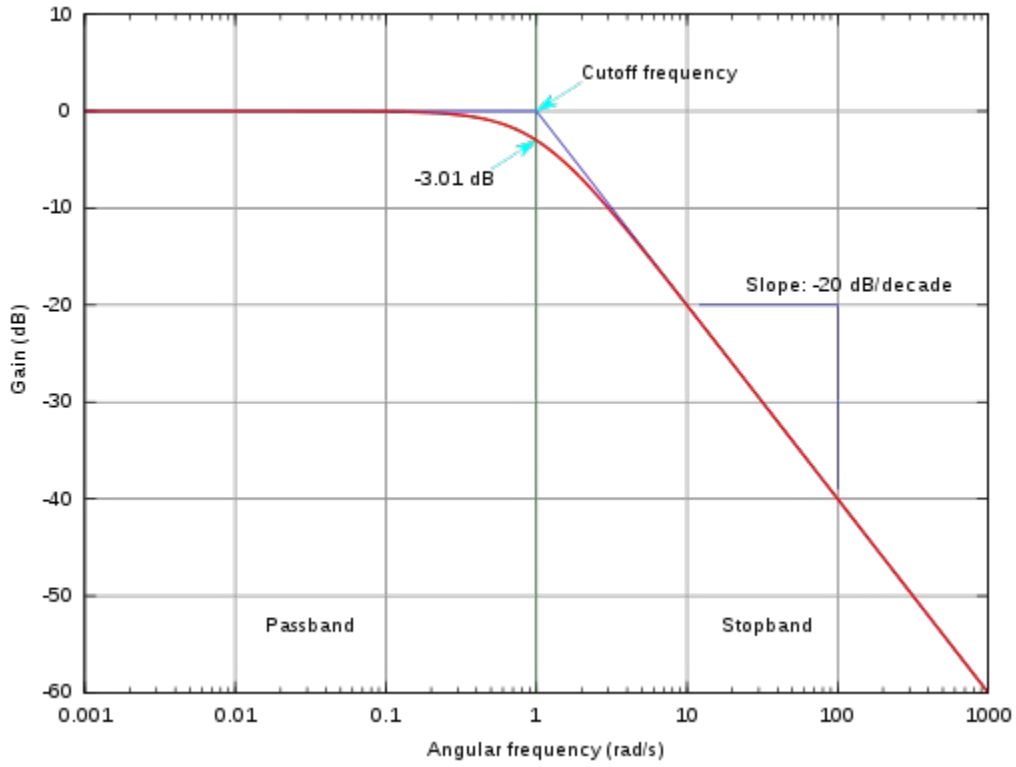
frecuencia de corte : Frecuencia en Hertz a partir de la cual se permitirá el paso de frecuencias mas bajas.
multiplicación : Número por el cual multiplicamos la señal del filtro. Generalmente se identifica con el volumen o amplitud del sonido siendo 0 el mínimo y 1 el máximo recomendado. El default es 1 dejando la señal sin alterar.
adición : Número que se le suma a la señal del filtro. Observar que a la señal se le aplica primero el mul y luego el add. El default es 0 dejando la señal sin alterar.

Ejemplos:

Pasan las frecuencias debajo de los 7030 Hz
{ LPF.ar(WhiteNoise.ar(0.7), 7030)}.scope //

Con los movimientos del mouse en y podemos variar la frecuencia de corte:
{ LPF.ar(WhiteNoise.ar(0.7), MouseY.kr(17000,200))}.scope

Un filtro ideal impediría completamente el paso de las frecuencias menores que la frecuencia de corte usando algo así como una recta vertical o un muro. Pero lamentablemente no vivimos en un mundo ideal y los filtros no pueden hacer esta función. Lo que hacen es un aproximado que resulta en una curva que va disminuyendo la amplitud de las frecuencias. Esto sucede con los filtros Pasa altas y Pasa bandas que veremos a continuacion



Filtro Pasa Altas

Es aquel que permite el paso de frecuencias desde una frecuencia determinada hhacia arriba, sin que exista un límite superior especificado. Esta frecuencia determinada es la frecuencia de corte. Por ejemplo, si determinamos que sea 700 Hz la frecuencia de corte dejaremos pasar todas las frecuencias mas altas que 700 Hz.

SuperCollider usa la clase HPF (High Pass Filter) para esto.

HPF.ar(entrada, frecuencia de corte, multiplicación y adición)

entrada : La señal que queremos filtrar. Tiene que tener .ar.

frecuencia de corte : Frecuencia en Hertz a partir de la cual se permitirá el paso de frecuencias mas altas.

multiplicación : Número por el cual multiplicamos la señal del filtro. Generalmente se identifica con el volumen o amplitud del sonido siendo 0 el mínimo y 1 el máximo recomendado. El default es 1 dejando la señal sin alterar.
adición : Número que se le suma a la señal del filtro. Observar que a la señal se le aplica primero el mul y luego el add. El default es 0 dejando la señal sin alterar.

Ejemplos

```
{ HPF.ar(WhiteNoise.ar(0.7), 7030)}.scope
{ LPF.ar(WhiteNoise.ar(0.7), MouseY.kr(17000,200))}.scope
```

Filtro Pasa Banda

Es aquel que permite el paso de frecuencias contenidas dentro de un determinado rango o banda, comprendido entre una frecuencia inferior y otra superior. La distancia entre estas frecuencias determina el ancho de banda. La frecuencia que está en el centro de esta distancia es la frecuencia de corte. Por ejemplo, si determinamos que sea 1000 Hz la frecuencia de corte y 200 Hz el ancho de banda podemos saber cual es el rango de frecuencias que dejaremos pasar usando la siguiente fórmula:

cota inferior = frecuencia de corte - ancho de banda/2

cota superior = frecuencia de corte + ancho de banda/2

La cota superior es la frecuencia límite superior y la cota inferior es la frecuencia límite inferior.

Entonces

cota inferior = 1000 - 200/2 = 900
cota superior = 1000 + 200/2 = 1100

Si sabemos cuáles son las cotas inferior y superior que queremos entonces podemos obtener el ancho de banda y la frecuencia de corte con la siguiente fórmula:

ancho de banda = cota superior - cota inferior

frecuencia de corte = cota inferior + ancho de banda/2

Entonces, siguiendo el ejemplo anterior

ancho de banda = 1100 - 900 = 200
frecuencia de corte = 900 + 200/2 = 1000

SuperCollider tiene el UGen BPF (Band Pass Filter)

BPF no es el único filtro que utiliza un ancho de banda. En este filtro y en todos los demás de este tipo el ancho de banda no se puede escribir directamente como un argumento. En vez de ancho de banda estos filtros tienen como argumento rq.

q = frecuencia de corte / ancho de banda.
Por lo tanto el recíproco de q = 1/q = ancho de banda / frecuencia de corte = rq

El máximo de rq es 2 y el mínimo es 0. El rq no puede ser mayor que 2 por que la cota inferior se iría por debajo de los 0hz.

Ejemplo

frecuencia de corte = 1000
rq = 2.5 = ancho de banda/1000

Despejando tenemos que

ancho de banda = 2.5 * 1000 = 2500

Entonces

cota inferior = 1000 - 2500/2 = -250

Sintaxis : BPF.ar(entrada, frecuencia de corte, rq, multiplicación, adición)

entrada : La señal que queremos filtrar.

frecuencia de corte : Frecuencia en Hertz que determina el centro de la banda de nuestro filtro.

rq : recíproco de q, es decir, ancho de banda / frecuencia de corte.

multiplicación : Número por el cual multiplicamos la señal del filtro. Generalmente se identifica con el volumen o amplitud del sonido siendo 0 el mínimo y 1 el máximo recomendado. El default es 1 dejando la señal sin alterar.

adición : Número que se le suma a la señal del filtro. Observar que a la señal se le aplica primero el mul y luego el add. El default es 0 dejando la señal sin alterar.

Algunos ejemplos.

```
// El tercer argumento del filtro pasa banda es el reciproco de
Q.
```

```
{ BPF.ar(WhiteNoise.ar(0.1), 7030, 700/7000)}.scope
```

En este ejemplo tenemos ancho de banda=700 y frecuencia de corte=7000. O sea, 700/7000=0.1 Por lo tanto rq=0.1

A veces es más rápido escribir el número décimal que el quebrado. Veamos entonces como queda sustituyendo del ejemplo anterior:

```
{ BPF.ar(WhiteNoise.ar(0.1), 7030, 0.1)}.scope

// aquí tenemos otro valor para el rq.
{ BPF.ar(WhiteNoise.ar(0.1), 7030, 1)}.scope
```

Si tenemos que 1 = ancho de banda / frecuencia de corte = rq, entonces sabemos que ancho de banda=frecuencia de corte = rq.
Y si frecuencia de corte = 7030,
entonces rq = 7030 / 7030.

Por lo tanto sabemos que la linea de código anterior se puede escribir también de la siguiente forma:

```
{BPF.ar(WhiteNoise.ar(0.1), 7030, 7030/7030)}.scope
Por último usemos el control del mouse en y para la frecuencia de
corte y en x para el rq

{ BPF.ar(WhiteNoise.ar(0.7), MouseY.kr(17000,200),
MouseX,kr(0,2))}.scope
```

Bibliografía

Hutchins, C. (2005). *SuperCollider Tutorial*.

Netri, E. y Romero, E. (2008). *Curso de SuperCollider: principiantes.* México DF: Centro Multimedia.

SuperCollider Help.

Wikipedia. Filter (signal processing). Recuperado de: http://en.wikipedia.org/w/index.php?title=Filter_%28signal_processing%29&oldid=493194971