**Música por computadora**

Ernesto Romero y Hernani Villaseñor
Centro Multimedia 2012

# Sesión 20

### SuperCollider en la interdisciplina con nuevos medios

La interdisciplina podríamos entenderla como el entrecruce de diferentes disciplinas artísticas que enriquecen las diferentes prácticas de arte. Así, el código ha sido un detonador en este cruce constante que se extiende a la ciencia y la tecnología. Podemos mencionar el desarrollo de proyectos audiovisuales, el uso de sensores como extenciones y nodos entre artes escénicas, música y artes visuales.

### Análisis de datos

Existen innumerables maneras de analizar un grupo de datos. Lo interesante es determinar el algoritmo realiza el análisis que deseamos. Aquí veremos un ejemplo de un algoritmo que realiza un análisis a un grupo de 100 números enteros.

1. Determina que números son menores que 30 y los agrupa en un subconjunto A.
2. Determina que números son mayores que 30 y menores que 60 y los agrupa en un subconjunto B.
3. Determina que números son mayores que 60 y los agrupa en un subconjunto C.
4. Determina cuantas veces se repite cada elemento de cada subconjunto.

```
// Primero creamos un conjunto ~u con 100 números enteros
aleatorios entre 0 y 100

~u=Array.rand(100,0,100);

// Creamos los subconjuntos vacíos ~a, ~b y ~c.

~a=[];
~b=[];
~c=[];
```

Con el siguiente algoritmo agrupamos en ~a los números menores que 30, en ~b los mayores que 30 y menores que 60, y en ~c los mayores que 60.

```
(
~u.size.do{|i|
          if(~u[i]<=30,{~a=~a.add(~u[i])});
          if((~u[i]>30)&&(~u[i]<=60),{~b=~b.add(~u[i])});
          if(~u[i]>60,{~c=~c.add(~u[i])});
          }
)
```

Podemos ver cómo quedan agrupados los números en subconjuntos después del primer análisis.

```
~a;
~b;
~c;
```

También podemos visualizar una gráfica de cada subconjunto.

```
~a.plot:
~b.plot;
~c.plot;
```

A continuación tenemos un algoritmo que determina cuántas veces está presente cada número. Pero primero observemos los mensajes `.sort` y `.indicesOfEqual` que se usan en el algoritmo.

```
// .sort nos pone en orden ascendente los números de un array.
Si declaramos:

[2,1,3,1,0,3].sort

// Obtenemos

[0,1,1,2,3,3]
```

`.indicesOfEqual` nos dice en qué indices estan presentes un elemento dentro de un array.

Si queremos saber en qué indices se encuentra presente el número 1 dentro del array `[0,1,1,2,3,3]` declaramos:

```
[0,1,1,2,3,3].indicesOfEqual(1);
```

Y obtenemos:

```
[1,2]

// Para saber la cantidad de veces que el número 1 se encuentra
en el array [0,1,1,2,3,3] hacemos esto:

[0,1,1,2,3,3].indicesOfEqual(1).size;
```

Ahora sí, el siguiente algoritmo determina cuántas veces está cada número en al subconjunto ~a y nos genera 2 arreglos: el arreglo ~elementosA con los elementos de ~a en orden y sin repetir y otro arreglo ~incidenciasA con la cantidad de veces que esta cada elemento en ~a.

```
(
var array=~a, elemento_anterior=nil;
~elementosA=[];
~incidenciasA=[];
array.size.do{|i|
          if(array.sort[i]!=elemento_anterior,
               {
               ~elementosA=~elementosA.add(array[i]);


~incidenciasA=~incidenciasA.add(array.indicesOfEqual(array[i]).s
ize);
               });
          elemento_anterior=array[i];
          };

);
(
~elementosA.postln;
~incidenciasA
);

// Lo mismo para ~b


(
var array=~b, elemento_anterior=nil;
```

```
~elementosB=[];
~incidenciasB=[];
array.size.do{|i|
          if(array.sort[i]!=elemento_anterior,
               {
               ~elementosB=~elementosB.add(array[i]);


~incidenciasB=~incidenciasB.add(array.indicesOfEqual(array[i]).s
ize);
               });
          elemento_anterior=array[i];
          };

)

(
~elementosB.postln;
~incidenciasB
)

// Y para ~c

(
var array=~c, elemento_anterior=nil;
~elementosC=[];
~incidenciasC=[];
array.size.do{|i|
          if(array.sort[i]!=elemento_anterior,
               {
               ~elementosC=~elementosC.add(array[i]);

~incidenciasC=~incidenciasC.add(array.indicesOfEqual(array[i]).s
ize);
               });
          elemento_anterior=array[i];
          };
)
(
~elementosC.postln;
```

```
~incidenciasC
)
```

**Mapeo**

Por mapeo entendemos la traducción de información de parámetros distintos, que por su cualidad informática están representados en números, pero que en su salida representan diferentes campos de acción. Es decir una frecuencia es representada en un número que puede ser traducido a un canal de color RGB, ambos son representados en números pero sus rangos son distintos. Aplicando una técnica de mapeo podemos transponer el rango de frecuencia del oído humano -20 a 20,000 Hz- a un canal de color del sistema RGB -0 a 255-. Esto es sencillo dado que dentro de la computadora estos números tienen el mismo comportamiento y son fácilmente escalables, toman sentido de ser frecuencia o color solo en su salida.

**MouseX.kr / MouseY.kr**(valor min, valor max, envolvimiento, intervalo)
Ugen del cursor, en plano vertical y horizontal de la pantalla.

valMin, valMax: rango de valores entre izquierda y derecha de la pantalla.
intervalo: factor del intervalo.
envolvimiento: curva del mapeo. 0 es lineal, 1 es exponencial.
intervalo: factor de intervalo, sirve para .. el movimiento del cursor.

**MouseButton.kr**(valMin, ValMax, intervalo)
Ugen del botón del ratón.
valMin: valor cuando el botón no está presionado.
valMax: valor cuando el botón está presionado.
intervalo: factor del intervalo.

```
(
SynthDef(\mapeo,{|out=0|
     var sen, env, mouseX, mouseY, mouseB;
     mouseX=MouseX.kr(300,3000,1);
     mouseY=MouseY.kr(1.0,0.001,1);
     mouseB=MouseButton.kr(0,1,0.1);
     sen=FSinOsc.ar(mouseX,0,mouseY);
     env=EnvGen.kr(Env.perc(0.1,3),mouseB,doneAction:0);
     Out.ar(out,sen*env)
     }).send(s)
)
a=Synth(\mapeo)
a.free
```

Otros recursos son los mensajes .linlin y .linexp que interpolan datos de manera lineal y exponencial.

**.linlin**(entradaMin, entradaMax, salidaMin, salidaMax, clip)
Envuelve los datos de tal manera que un rango lineal de entrada es mapeado a un rango lineal de salida.

clip: puede ser uno de estos 4: nil, no hay clip, \minmax hay clip en el mínomo y máximo de salida, \min hay clip en el mínimo de salida, \max hay clip en el máximo de salida.

**.linexp**(entradaMin, entradaMax, salidaMin, salidaMax, clip)
Envuelve los datos de tal manera que un rango lineal de entrada es mapeado a un rango exponencial de salida. Hay que tener cuidado que el mínimo y máximo de salida no sean menores de cero y que sean del mismo signo. El clip funciona igual en en .linlin.

**OSC**

Open Sound Control es un protocolo de comunicación en red que utilizan tanto computadoras como sintetizadores y otros aparatos multimedia. Asimismo este protocolo comunica diferentes programas. SC tiene implementado OSC para su comunicación interna y de manera externa para comunicarse con otros programas como pueden ser Processing, OpenFrameworks o PureData. Este protocolo también es útil para comunicar un *smartphone* o para hacer colaboraciones en red a través de Internet o de manera local mediante TCP/IP a través de los puertos UDP.

La comunicación mediante OSC podemos pensarla en de dos maneras, una es el envío de datos y otra es la recepción de datos.

**Envío de datos**

**NetAddr**(IP, puerto)
Dirección de red

IP: es una cadena de números: “127.0.0.1” representa comunicación interna.
puerto: es un número como este 57120, es el número de puerto exclusivo de SC.

```
// envío de datos de SC a SC dentro de la misma computadora

NetAddr("127.0.0.1", 57120)
```

**Recepción de datos**

OSCresponder(dirección, nombreComando, acción)
Cliente que responde a la red, registra una función para ser llamada con un comando específico de una dirección OSC específica.

dirección: la dirección desde donde recibe el *responder,* puede ser nil en cuyo caso responde a cualquier dirección
nombre de comando: un comando OSC del tipo: /algo
acción: una función que será evaluada cuando un comando de ese nombre es recibido por la dirección

```
// recepción de datos
OSCresponder (n, "/buenas/tardes", {arg tiempo, mensaje,
respuesta; [mensaje, tiempo]}).add
```

Comenzamos enviando datos internamente en SuperCollider.

```
// declaro el código para enviar datos a SC dentro de mi pc
n=NetAddr("127.0.0.1", 57120)

// delcaro el código para recibir datos de SC dentro de mi pc
r=OSCresponder(nil,"/recibo",{|tiempo,respuesta,mensaje,comando|
[tiempo,respuesta,mensaje,comando].postln}).add

// mando un mensaje con esta línea, en la post aparece el
resultado
n.sendMsg("/recibo",1)
```

La post me imprime una serie de valores encasillados dentro de un Array, de los cuales me interesa el tercero que es el mensaje, el cual a su vez es un Array que contiene el comando OSC y un valor útil. Para filtrar ese valor usaremos un OSCresponder que indica que información quiero leer.

**Aplicaciones del mapeo y OSC**

**Sonificación de datos**

La sonificacion aborda el manejo de un conjunto con una gran cantidad de elementos o de un flujo constante de datos variables para convertirlos en sonido.
Ahora vamos a sonificar el conjunto ~u después de haber sido analizado. Creamos primero un SynthDef que acepte cambios en argumentos de frecuencia y duración.

```
(
```

```
SynthDef(\sonido, {|freq=200, amp=0.5, dur=0.15|
     var sig, env;
     sig=Saw.ar(freq, amp);
     env=EnvGen.kr(Env.sine(dur,1),doneAction:2);
     Out.ar(0,sig*env);
     }).send(s);
);

Synth(\sonido)

// Reestructuramos los subconjuntos ~a, ~b y ~c.

(
~a=[];
~b=[];
~c=[];
~u.size.do{|i|
     if(~u[i]<=30,{~a=~a.add(~u[i])});
     if((~u[i]>30)&&(~u[i]<=60),{~b=~b.add(~u[i])});
     if(~u[i]>60,{~c=~c.add(~u[i])});
     }
)

// Declaramos una frecuancia fundamental.

~fund=30;
```

El siguiente algoritmo de sonificación le manda el mensaje .midiratio a los números enteros del subconjunto ~a para leerlos como semitonos sobre la fundamental. La duración es el inverso de la cantidad de veces que se repite el número entero. Además repite la nota que representa ese número entero la cantidad de veces que se repite dentro de ~a.

```
(
(
Tdef(\sonificacionA, {var dur;
               ~a.size.do{|i|
               dur=1/
(~incidenciasA[~elementosA.find([~a[i]])]).postln;
     ~incidenciasA[~elementosA.find([~a[i]])].do{
     Synth(\sonido, [\freq, ~a[i].midiratio*~fund, \dur, dur]);
```

```
     dur.wait;
     }
     }
     }).quant_(0);
);
/*
Tdef(\sonificacionA).play;
Tdef(\sonificacionA).stop;
*/

// Lo mismo para ~b.

(
Tdef(\sonificacionB, {var dur;
                    ~b.size.do{|i|
                              dur=1/
(~incidenciasB[~elementosB.find([~b[i]])]).postln;

     ~incidenciasB[~elementosB.find([~b[i]])].do{
                              Synth(\sonido, [\freq,
~b[i].midiratio*~fund, \dur, dur]);
                              dur.wait;
                              }
                              }
                    }).quant_(0);
);
/*
Tdef(\sonificacionB).play;
Tdef(\sonificacionB).stop;
*/

// Y para ~c

(
Tdef(\sonificacionC, {var dur;
                    ~c.size.do{|i|
                              dur=1/
(~incidenciasC[~elementosC.find([~c[i]])]).postln;
```

```
     ~incidenciasC[~elementosC.find([~c[i]])].do{
                                Synth(\sonido, [\freq,
~c[i].midiratio*~fund, \dur, dur]);
                                dur.wait;
                                }
                                }
                        }).quant_(0);
);
/*
Tdef(\sonificacionC).play;
Tdef(\sonificacionC).stop;
*/
)

// Aquí hechamos a andar los tresubconjuntos sonificados.

/*
(
Tdef(\sonificacionA).play;
Tdef(\sonificacionB).play;
Tdef(\sonificacionC).play;
);
(
Tdef(\sonificacionA).stop;
Tdef(\sonificacionB).stop;
Tdef(\sonificacionC).stop;
)
*/
```

**Bibliografía**

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