Data Structures en Pd

por Gregorio Garcia Karman

I. Introducción

"La idea inicial era desarrollar Pd como un entorno de creación de música por ordenador en tiempo real como Max, pero de alguna manera también incluir la funcionalidad de realizar partituras de música por ordenador con representaciones gráficas determinadas por el usuario" [Puckette, Pd documentation]

"Pd ha sido diseñado para ofrecer un entorno fuertemente desestructurado para describir estructuras de datos y su apariencia gráfica. La idea subyacente es permitir al usuario mostrar cualquier tipo de datos, asociándolos arbitrariamente con una representación visual. Para llevar esta idea a cabo, Pd introduce la idea de una estructura de datos gráfica, similar a las estructuras de datos del lenguaje de programación C, con la prestación de poder asociar formas y colores a los datos, para que el usuario pueda visualizarlos y editarlos." [Puckette, Pd documentation]

Este tutorial es resultado de las experiencias de su autor con los objetos de manipulación de estructuras de datos de Pure Data. No constituye una documentación oficial del funcionamiento de estos objetos ni su autor se puede hacer responsable de lo que pueda ocurrir en caso de que obtenga resultados no deseados siguiendo lo que aquí se explica.

II. Códigos operativos básicos

struct

El objeto **stuct** sirve para definir una estructura de datos. Sus argumentos especifican el nombre de la estructura, y el tipo y nombre de los campos que contiene. En el ejemplo siguiente definimos una estructura de datos, de nombre *estructura_01*, con tres campos tipo *float*: x, y, k:

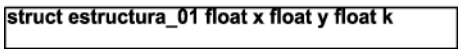


Fig. 1 Objeto **struct** con tres campos tipo *float.* **[tut_01.pd].**

La manera de utilizar el comando **struct** es colocándolo en el interior de un subpatch de pd. Un objeto **struct** dentro de un subpatch constituye un *template*, o plantilla de estructura de datos [Fig 2a]. Cada *template* debe contener un único **struct** en su interior. El nombre asignado a un **struct** debe ser único entre todos los patches abiertos en una misma sesión de pd [1].

[1] En caso contrario la ventana de pd muestra un aviso y las cosas suelen dejar de funcionar correctamente.

Para albergar los datos es necesario crear también una ventana de datos o *datawindow*, que es un subpatch de pd vacío y con nombre único [Fig 2.b]. (Cuando creo este segundo subpatch me resulta útil pensar en que estoy definiendo un "espacio de memoria" donde voy a almacenar los datos). Además la ventana *datawindow* servirá también para visualizar y manipular los datos gráficamente, como veremos más adelante[2].

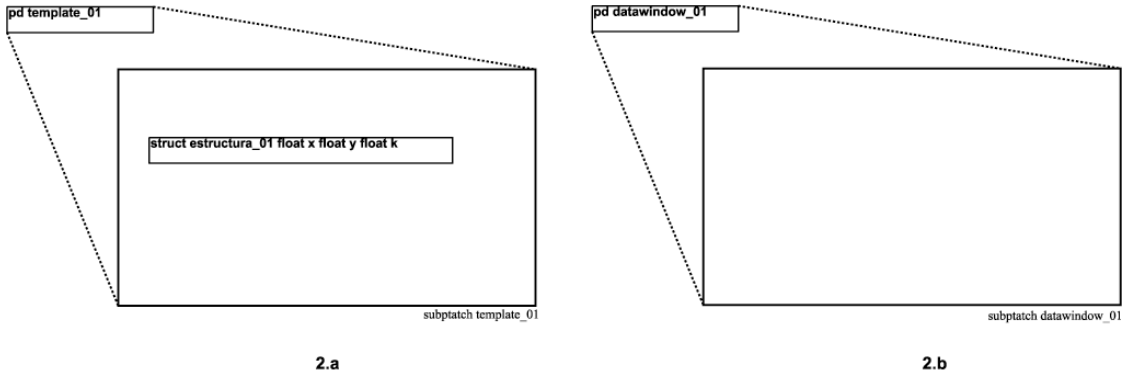


Fig. 2. Primeros pasos para crear una estructura de datos en Pd. a) Crear un objeto struct en el interior de un subpatch. b) Crear un segundo subpatch vacío para almacenar, visualizar y modificar gráficamente los datos.

Aunque de momento nuestra lista está vacía, los datos "almacenados" en un *datawindow* se organizan en forma de una lista ordenada de escalares (*scalars*). Como puede verse en la siguiente figura [Fig. 3.], cada uno de estos escalares consiste en varios campos de datos organizados según la definición realizada en **struct**. En este ejemplo el primer *scalar* de la lista almacenará los datos x_1, y_1, k_1, el segundo *scalar* de la lista x_2, y_2, k_2, etc...

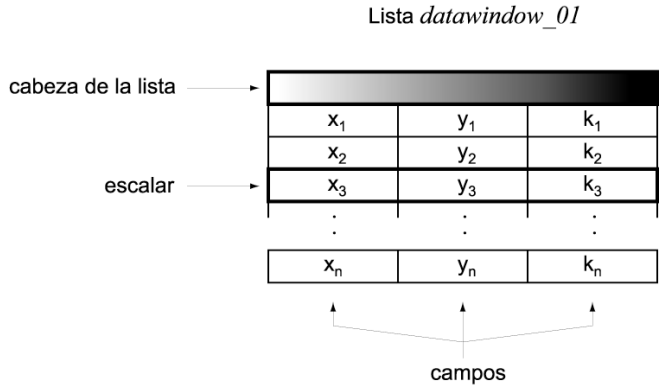


Fig. 3. Los datos almacenados en el subpatch *datawindow_01* se organizan en forma de lista ordenada de *scalars*.

El acceso a una posición determinada de la lista se realiza mediante punteros, un tipo de variable que permite localizar la posición de un *scalar* en una lista. El manejo de los punteros es una de las claves del funcionamiento de las estructuras de datos en Pd. Por ello, es necesario entender el funcionamiento del objeto **pointer** antes de poder introducir *scalars* en nuestra lista**.**

[2] Posteriormente veremos cómo incorporar instrucciones de dibujo, el otro elemento (opcional) de un template.

pointer

Pointer es un objeto de almacenamiento como **float,** pero en lugar de un número almacena la posición de un *escalar* en la lista. La posición almacenada puede ser la de un *escalar* existente en una lista pero **pointer** también puede apuntar hacia la "cabeza" de la lista, una posición especial situada antes del primer escalar de la lista y que no sirve para contener datos[3] [Fig. 3]. La posibilidad de apuntar hacia la cabeza de una lista nos es muy útil ahora ya que de otro modo no podríamos apuntar hacia nuestra lista vacía. Para obtener un puntero hacia la cabeza de una lista, debemos enviar al terminal izquierdo de **pointer** el mensaje *traverse* [*pd-datawindow*].

Al hacer esto el contenido de **pointer** se actualiza, apuntando ahora hacia la cabeza de la lista *datawindow*, sin generar aún ninguna salida. Para ello es necesario enviar a continuación un *bang* y **pointer** generará por su salida izquierda una variable tipo *pointer* (un nuevo tipo de PD del mismo orden que *float* o *symbol*[4]), el puntero hacia la cabeza de nuestra lista *datawindow* [5].

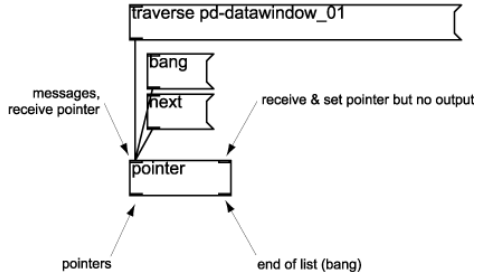


Fig. 4. Funcionamiento del objeto pointer

Si ahora enviamos el mensaje *next* al terminal izquierdo del objeto **pointer,** éste devuelve sucesivamente por su salida izquierda los punteros a los siguientes escalares de la lista. Sin embargo, como en este momento nuestra lista está aún vacía, si enviamos un *next* a **pointer,** el terminal derecho genera un *bang* indicando que se ha llegado al final de la lista y el terminal izquierdo no genera salida. Un nuevo *next* generará un error de 'puntero inexistente' en la ventana de Pd (`error: ptrobj_next: no current pointer`). Monitorice las dos salidas de **pointer** mientras prueba los mensajes.

Una vez que hemos definido los contenedores de datos y los punteros que proporcionan la herramienta para acceder a ellos, es el momento de añadir *escalares* a nuestra lista vacía con el objeto **append**.

append

Para añadir escalares a una lista se utiliza el objeto **append**. Sirve para insertar nuevos escalares en una determinada posición de la lista marcada por un *pointer*. Sus argumentos de creación son: el nombre del la estructura que define al escalar que vamos

[3] Su utilidad es poder apuntar hacia listas vacías así como poder añadir un escalar en la primera posición de una lista existente utilizando el comando **append**.

[4] puede ser utilizado en algunas funciones de encaminamiento como *trigger* o *pack*, pero su contenido no puede ser visualizado mediante un objeto <number> ó <symbol> ni tampoco impreso por el terminal de Pd.

[5] Cuando un puntero apunta a la cabeza de una lista en lugar de a un escalar, se habla de un 'empty pointer' (puntero vacío).

a añadir, y los nombres de los campos que queremos inicializar (debe definirse al menos uno).

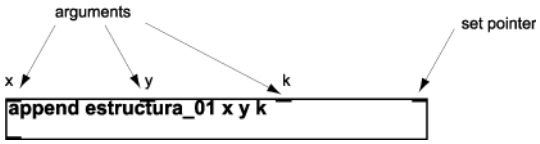


Fig. 5. El objeto **append** y sus *inlets*

Una vez inicializado el objeto y sus argumentos, la secuencia de comandos para utilizar el objeto **append** sería [figura 5]:

- Establecer el puntero: En primer lugar especificaremos la lista en la que se quiere insertar el *scalar* enviando un *pointer* al terminal derecho de append. En el ejemplo, al hacer clic sobre el mensaje *traverse datawindow1*, el objeto **pointer** genera un *pointer* que apunta hacia la cabeza de la lista *datawindow1*. Al recibir el *pointer* el objeto **append** queda preparado para insertar un escalar en la posición indicada (1).
- Enviar los valores: Se introducen a través del resto de terminales los datos. Habrá tantos terminales de entrada como campos del *scalar* se hayan inicializado como argumentos (en nuestro ejemplo *x*, *y*, *k*) (2). Cuando el terminal (caliente) del extremo izquierdo recibe un número, este dato, junto con los presentes en el resto de terminales, se añaden a la lista creando un nuevo *scalar*.

Internamente **append** mantiene un puntero hacia una posición en la lista, que, como hemos visto en el paso anterior, se establece y actualiza enviando un *pointer* a su terminal derecho. Cuando el nuevo escalar se añade a la lista, el puntero que mantiene **append** se incrementa en una posición (apuntando al escalar añadido), *pointer* del que se hace eco en su salida (3). Una vez que se ha establecido internamente el puntero de **append** no es necesario volver a enviarle el mensaje de *traverse* para insertar un nuevo escalar salvo que se quiera volver a la cabeza de la lista.

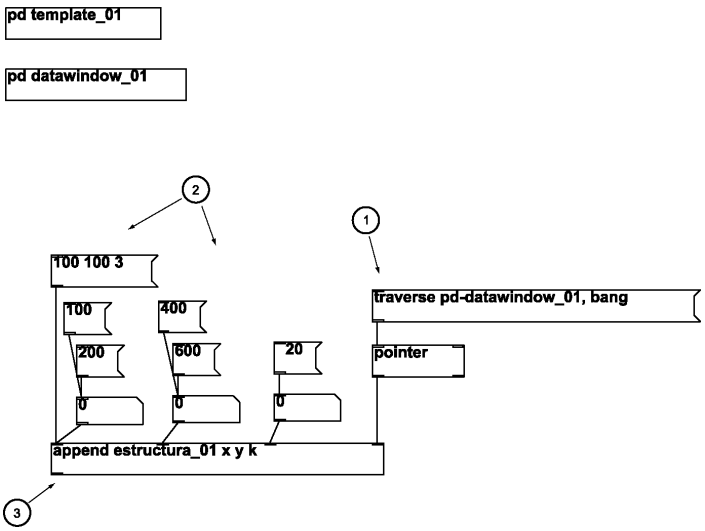
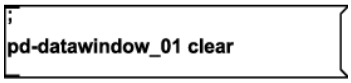


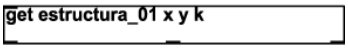
Fig. 6. Aspecto del patch. Secuencia de acciones al añadir el primer escalar a la lista utilizando **append. [tut_06.pd]**

get

Para ver los escalares que hemos añadido a la lista utilizaremos el objeto **get,** que permite recuperar el contenido almacenado en los campos de cada escalar que hemos introducido. Los argumentos de creación de **get** son: el nombre de la estructura donde está definido el tipo de escalar que vamos a consultar (en nuestro caso *estructura_01*); y a continuación el nombre de los campos que se desea recuperar. De esta manera se crea un objeto con un terminal de entrada y tantos terminales de salida como campos se hayan invocado[6].

Si desea INICIALIZAR EL CONTENIDO DE LA LISTA, borrando todo su contenido (p. ej. para empezar a añadir escalares de nuevo) puede enviar el mensaje *clear* a la ventana de datos:





Al recibir en su *inlet* un puntero a un escalar, el objeto **get** presenta en sus salidas los valores de los campos del escalar apuntado. Como en el ejemplo de **append,** para acceder a un determinado escalar, primero debemos colocarnos en la cabeza de la lista mediante un mensaje de *traverse* a **pointer**, para luego movernos por ella mediante mensajes *next*.

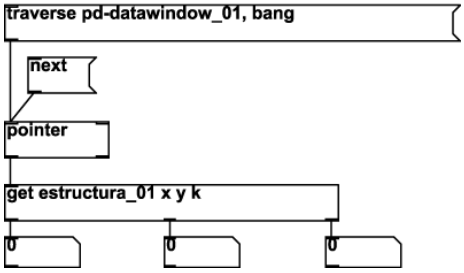
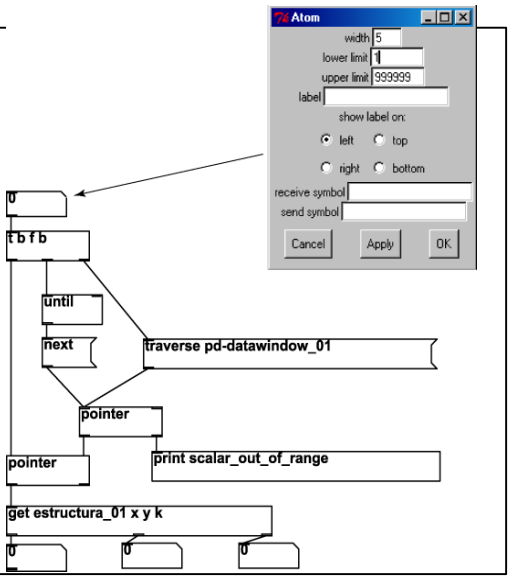


Fig. 08. Conexiones básicas para usar el objeto **get [tut_08.pd]**

EJEMPLO tut_09.pd Un pequeño patch para acceder a cualquier posición de la lista utilizando el comando **until**. Esta estrategia es muy eficaz si conocemos la posición numérica en la lista del escalar que queremos consultar. En los motajes con **until** es necesario tener cuidado para evitar los bucles sin fin, así que por precaución establezca antes el límite inferior en las propiedades de objeto numérico a un valor mayor que 0 (*lower limit = 0, upper limit =* 99999*)*.



[6] Los nombres de los campos deben existir en la estructura invocada.

set

Como en **get**, los argumentos de creación del objeto **set** son el nombre de la estructura que define al escalar, y los nombres de los campos que queremos modificar. El terminal de entrada del extremo derecho, espera recibir un *pointer* hacia el escalar que queremos modificar (*traverse, next,...*). El resto de terminales sirve para introducir los nuevos valores de los campos. Como ocurría con **append** sólo el terminal del extremo izquierdo es "caliente" y actualiza el contenido del escalar. A diferencia de **append,** el comando **set** no añade nuevos escalares a una lista sino que modifica el valor de los campos de escalares preexistentes.

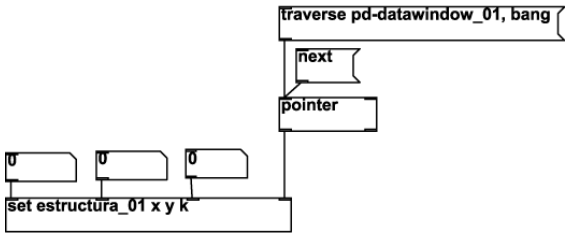


Fig 10. Montaje para el uso del objeto **set**

Hemos completado el mecanismo básico para crear, almacenar, modificar y consultar una estructura de datos simple. En el siguiente apartado se explica cómo vincular los datos contenidos en la lista creada con objetos gráficos.

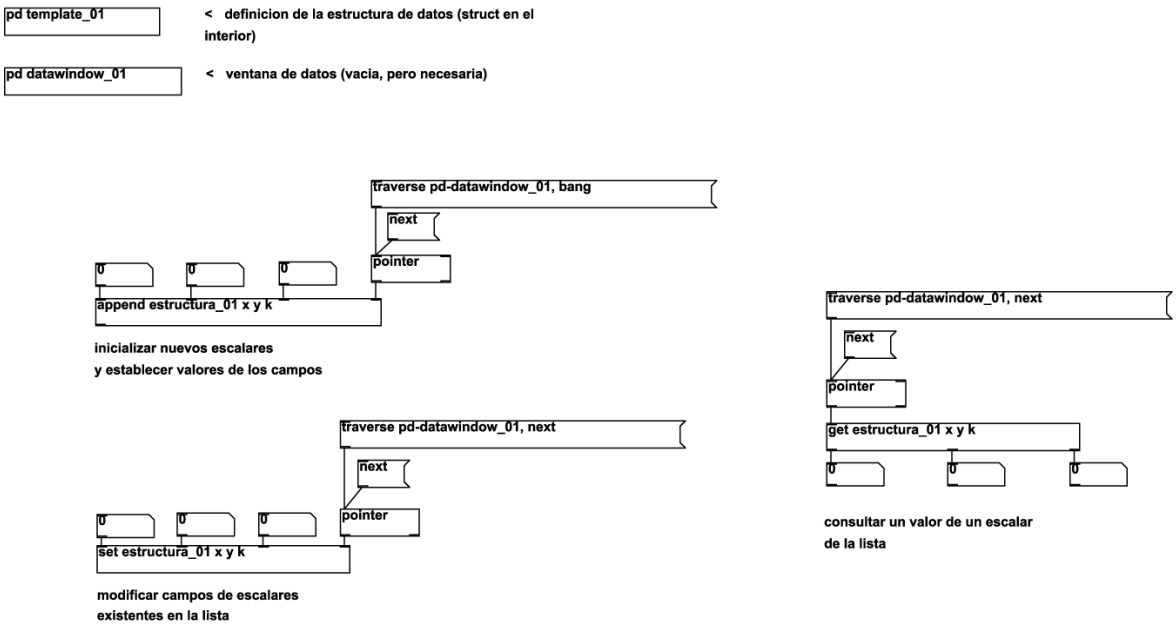


Fig. 11. Vista del patch completo con todos los objetos básicos para manejar una estructura de datos simple. **[tut_11.pd]**

Instrucciones de dibujo

Los comandos de dibujo **drawnumber, drawpolygon**, **filledpolygon, drawcurve** y **filledcurve** establecen vínculos entre los campos numéricos (*floats*) de un *scalar* y objetos gráficos, permitiendo la visualización de sus valores y el control gráfico de las listas. Estas instrucciones de dibujo se colocan en el interior de un *template* acompañando al comando **struct.** Cada instrucción de dibujo crea un *objeto* (un dibujo en el subpatch *datawindow*) por cada *scalar* existente en la lista *datawindow*. Un mismo *template* puede contener varias instrucciones de dibujo.

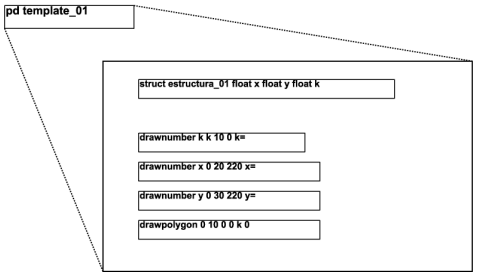
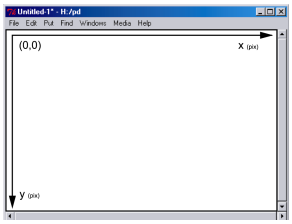
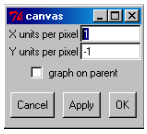


Fig 12. Plantilla de datos con un comando **struct** y varias instrucciones de dibujo asociadas.

GEOMETRÍA DEL CANVAS
El número de unidades que representa cada píxel de la ventana se ajusta en las propiedades de la ventana (click botón derecho sobre el patch). Por defecto los valores son Xunits = 1, Y units= -1, que significa que el origen de coordenadas para las instrucciones de dibujo se va a situar en el extremo superior izquierdo de la ventana como se muestra en la figura.

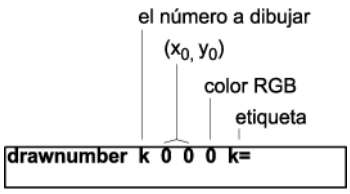




Estos valores pueden utilizarse para hacer zoom sobre el patch o para cambiar la posición de los ejes. Para utilizar la (problemática) opción *graph on parent* (GOP) véase el tutorial de Frank Barnecht en <footils.org/cms/show/31>

Drawnumber. Dibuja un número en una posición de la ventana. Sus argumentos son:

- el número a dibujar
- un par (x_0, y_0) indicando coordenadas relativas.
- color RGB[7]
- etiqueta (opcional)



Si incluye la instrucción anterior en el subpatch *template_01*, Pd dibujará inmediatamente el valor de los campos k_i en las posiciones (x_i, y_i) dentro del subpatch

[7] Los colores RGB se especifican con un número de tres cifras. Cada cifra corresponde a un canal de color, con 0 el valor mínimo y 9 el máximo:

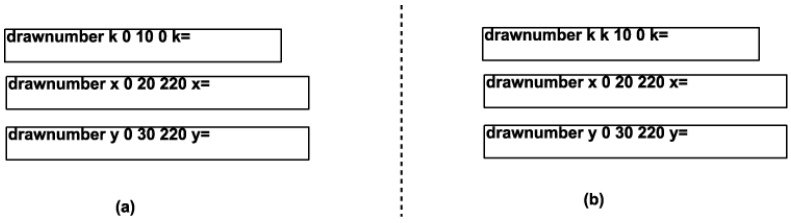
000 – negro
900 – rojo
090 – verde
009 – azul
999 – blanco

datawindow_01 por cada escalar (x_i y_i k_i) existente [Fig. 13]. Esto quiere decir que sin haberlo hecho explícito, las variables *x* e *y* del escalar controlan la posición donde se muestra la variable *k*. Esto ocurre porque las variables *x* e *y* definidas en nuestro ejemplo son nombres de variable reservadas, que cuando se encuentran definidas como campos de un escalar controlan la posición absoluta de los objetos gráficos asociados.

Fig. 13. Captura de pantalla de la ventana de datos utilizando la instrucción de dibujo <**drawnumber k 0 0 0 k=**>.

Situando el ratón sobre el valor numérico k_i puede modificar gráficamente el valor de la variable *k* de cada escalar creado. Compruebe cómo ha cambiado el valor de *k* en los escalares manipulados utilizando el objeto **get**.

Observe el resultado de usar las siguientes instrucciones de dibujo con la estructura *estructura_01*.



a) Con el primer set de instrucciones, además de <*k*> se dibujarán dos nuevos objetos gráficos (<*x*> e <*y*>) que indican las coordenadas de la variable *k* [Fig 14]. Fíjese cómo la posición de cada número dibujado se obtiene como resutado de sumar la posición absoluta especificada en los campos (x_i, y_i) del escalar, con el valor de los campos (x_0, y_0) de la instrucción de dibujo: <k=k_i> aparece dibujado en la posición (x_i, y_i + 10), <x=x_i> en la posición (x_i, y_i + 20), <y=y_i> en la posición (x_i , y_i + 30). De manera general la posición en que se dibuja un objeto se puede expresar como:

$$(x_i + x_0, y_i + y_0)$$

x_i, y_i son los valores de los campos x e y del escalar *i* (coordenadas absolutas)

x_0, y_0 son las coordenadas de la instrucción de dibujo (coordenadas relativas)

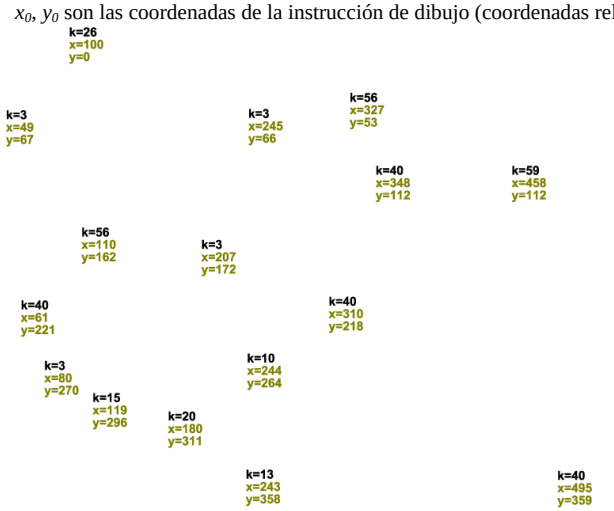


Fig. 14. Captura de pantalla de la ventana de datos del ejemplo a).
Las variables x, y reflejan las coordenadas del objeto k.

b) En el segundo ejemplo, el valor de x_0 es sustituido por la variable *k*. Ahora la posición del objeto <k=k_i> depende del valor variable k_i, y puede verse cómo se modifica su posición cuando movemos el valor numérico de *k* en la ventana de datos. En este caso la posición de <k=k_i> en la ventana de datos aparece en las coordenadas (x_i+ k_i, y_i).

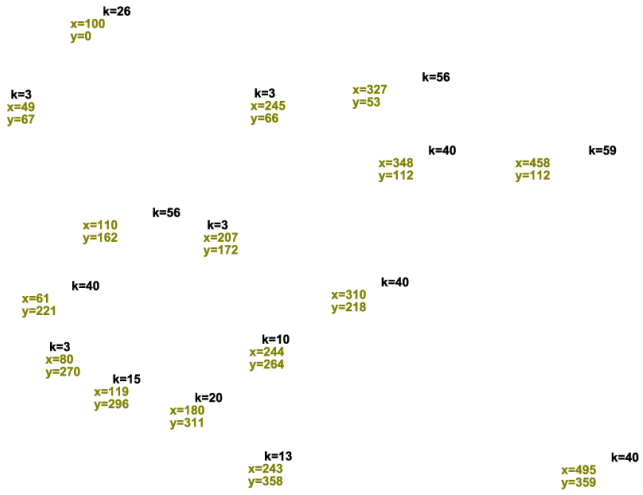


Fig. 15. Captura de pantalla de la ventana de datos con las instrucciones de dibujo del ejemplo b). Nótese el desplazamiento adicional del objeto <k> = (x_i+ k_i, y_i).

Como puede verse, los campos numéricos definidos en **struct** pueden utilizarse como argumentos de las instrucciones de dibujo, vinculando los datos de la lista con la representación gráfica.

Si pasa al modo edición (CTRL+E) en la ventana de datos, podrá mover, cortar, pegar, duplicar y eliminar escalares de la lista gráficamente. Compruebe cómo cambian los datos numéricos cuando desplaza los objetos por la pantalla.

drawpolygon. Dibuja una sucesión de segmentos de recta unidos entre sí.

- color RGB
- ancho de línea
- dos o más pares (a, b) indicando coordenadas.

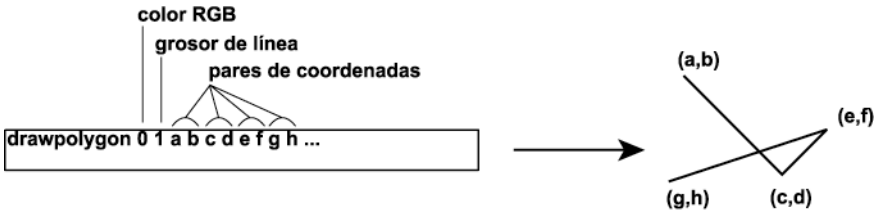


Fig. 16. El comando de dibujo **drawpolygon** con sus argumentos. El ejemplo de la derecha representa el objeto gráfico correspondiente a un escalar resultante de la instrucción de dibujo.

Como con **drawnumber** todos los argumentos numéricos de las instrucciones de dibujo pueden ser sustituidos por variables de la estructura:

- Si todos los campos numéricos son constantes, el dibujo resultante no estará vinculado a ningún campo del escalar.

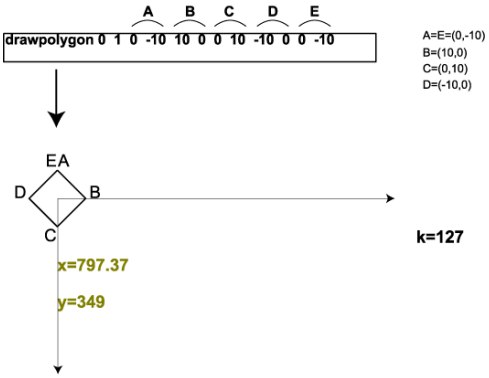


Fig 17. Dibujo de un rombo estático (no está vinculado a ningún campo del escalar).

- Cuando una variable controla un argumento de dibujo podemos cambiar el valor de la variable modificando ese aspecto del objeto gráfico.

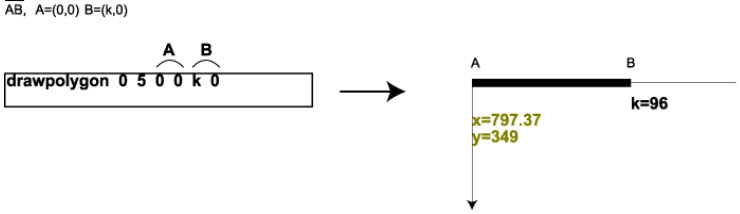


Fig 18. Dibujo de un slider horizontal vinculado al campo *k* del escalar.

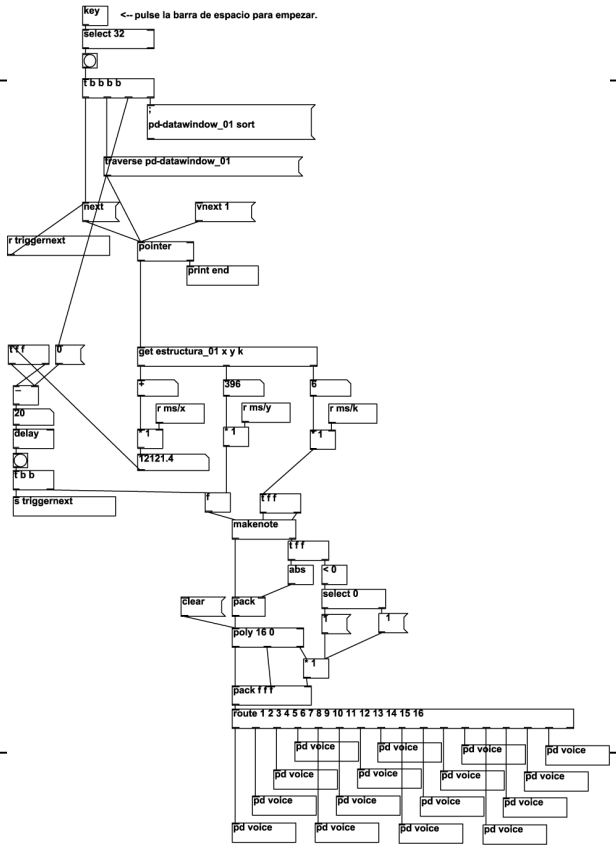
De esta manera tendremos un control del valor de *k* a modo de 'slider' [Fig. 18].

Fig 19. Captura de panatalla de la ventana de datos utilizando la instrucción **drawpolygon 0 5 0 0 k 0**. Hemos utilizado la instrucción para dibujar un segmento de recta horizontal cuya longitud está controlado por *k*, como si fuera un slider.

- Cuando una misma variable controla dos ó más campos, podremos cambiar el valor de la variable modificando cualquiera de los controles gráficos; el resto de atributos vinculados a esa variable se modifican simultaneamente (pruébese por ejemplo con la instrucción **drawpolygon 0 2 k 0 0 0 0 k**).

Siempre debe recordar que las coordenadas efectivas de un dibujo son resultado de sumar las coordenadas relativas (controladas por los argumentos de la instrucción de dibujo: a_i, b_i, c_i, d_i,...) y absolutas (la posición del escalar controlada por los argumentos *x* e *y*).

Como **EJEMPLO** de aplicación utilizando la estructura de datos con el interfaz gráfico de la figura 19, se ha construido un secuenciador para un granulador de 16 voces que actúa sobre un fichero de audio cargado en una tabla. Los objetos gráficos controlan la posición y duración del grano y el punto de lectura dentro del fichero[ejemplo_struct.pd].
Analice el mecanismo de secuenciación basado en un objeto **delay** y estudie el sistema de asignación de polifonía (voces). **[tut_019.pd]**



drawcurve. Dibuja curvas controladas por pares de coordenadas. Similar en su manejo a drawnumber. Sus parámetros son:

- color RGB
- ancho de línea
- dos o más pares (a, b) indicando coordenadas.

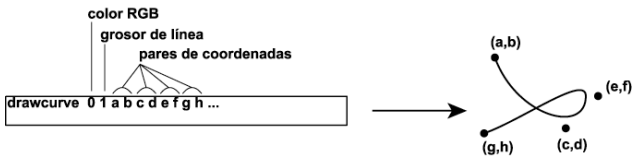


Fig 20. **drawcurve** y sus argumentos

Enviar el mensaje sort a la ventana de datos: ordena la lista de escalares en orden creciente según el argumento x. En nuestro ejemplo anterior antes de reproducir la partitura se reordena la lista de escalares, secuenciando los escalares en orden creciente según el instante de disparo (en el ejemplo hemos utilizado la variable x como instante de disparo del grano).

```
;
pd-datawindow_01 sort
```

filledpolygon & **filledcurve.** Dibujan figuras geométricas cerradas. Se añade un nuevo argumento con respecto a las instrucciones anteriores (drawpolygon, drawcurve): el color interior del objeto gráfico.

- color RGB interior
- color RGB línea
- ancho de línea
- dos o más pares de coordenadas (a, b) indicando posición de los vértices.

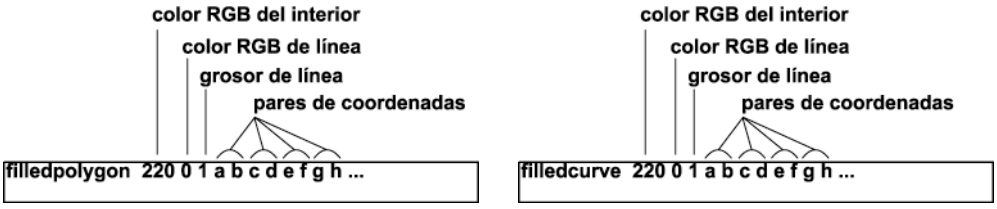


Fig 21. Argumentos de **filledcurve** & **filledpolygon**

Guardar y recuperar partituras: El contenido de la ventana de datos se guarda cuando salvamos el patch. También puede enviar los comandos *write* y *read* a la ventana de datos para salvar y cargar las partituras:

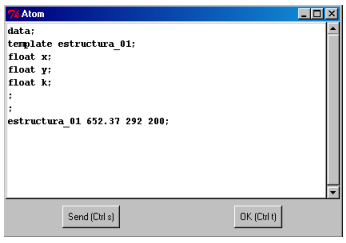
- Enviar un msg *write* [nombrefichero.txt] a la ventana *pd-datawindow*:

```
;
pd-datawindow_01 write partitura.txt
```

- Enviar un msg *read* [nombrefichero.txt] a la ventana *pd-datawindow*:

```
;
pd-datawindow_01 read otrapartitura.txt
```

Acceso a los datos de los objetos gráficos: Además de los modos de edición gráfica de la partitura ya utilizados*, si examina las propiedades de un objeto gráfico en la ventana de datos (click dcho. / Properties), Pd muestra un cuadro de diálogo en el que se puede ver la estructura del escalar así como editar numéricamente los valores de sus campos.



*Edición con el mouse de los campos variables definidos en las instrucciones de dibujo: click y arrastrar sobre el elemento de control.
*Modo de edición (CTRL-E): mover, copiar, cortar, pegar, borrar.

Proyecto propuesto

En el patch de ejemplo, hacer que sea controlable la velocidad de reproducción de cada grano, y en consecuencia su altura y duración, desde la partitura. Una partitura posible adaptada a esta modificación podría ser:

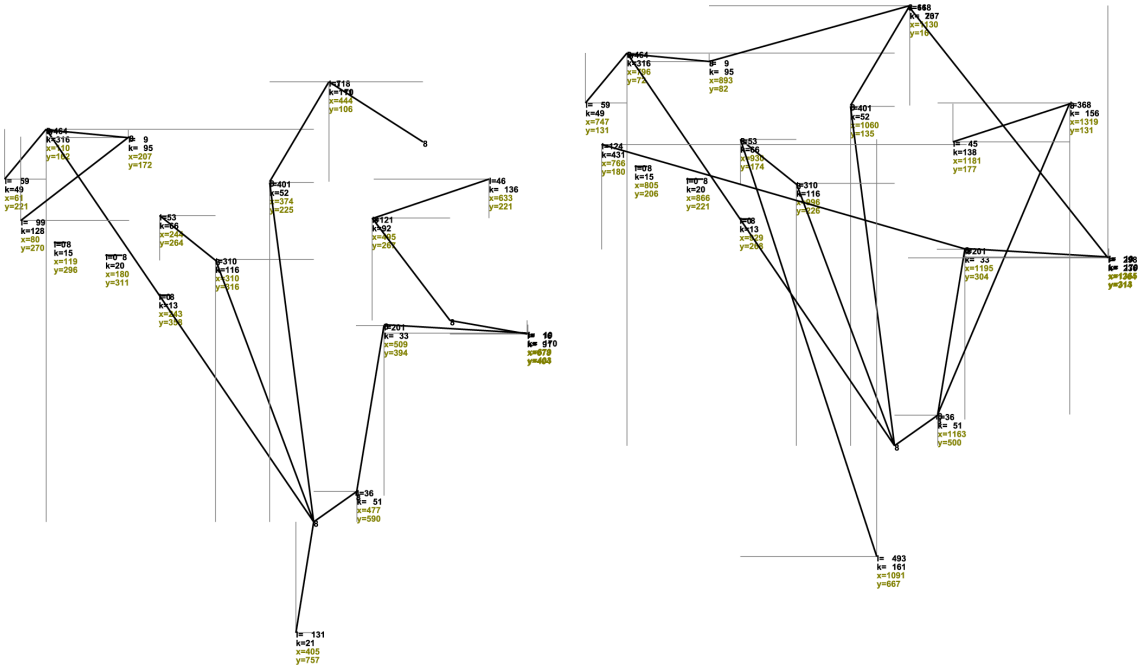


Fig 22. Posible partitura para el proyecto propuesto

Por hacer

Estructuras anidadas (campos de tipo array), plot, vnext, etc...