PWGL EDITORS: 2D-EDITOR AS A CASE STUDY

Mikael Laurson
CMT
Sibelius Academy
laurson@siba.fi

Mika Kuuskankare
DocMus
Sibelius Academy
mkuuskan@siba.fi

ABSTRACT

This paper presents some of the main concepts behind PWGL editors. PWGL is an OpenGL based visual programming language specialized in computer aided composition and sound synthesis. Editors have a central role in PWGL as they allow to investigate and manipulate complex objects. We first describe some of the general design issues behind PWGL editors. We will use one of the main editors, the 2D-editor, as a case study. The 2D-editor allows to combine and synchronize visually various 2D-objects within one editor.

1. INTRODUCTION

Recent development in computer hardware and operation systems has made it possible to reconsider the basic concepts behind music related software development for computer aided composition and sound synthesis. Typically this kind of software has to deal with complex and demanding problems such as real-time sound synthesis, complex representation of musical data, large databases, and search problems. One of the key issues is how these various tools and approaches can interact with each other. Previous successful efforts in this direction, such as PatchWork (PW, [2]) and OpenMusic [1], have proved that combining a high-level programming language (such as Common Lisp and CLOS) with visual programming results in an environment that can be used to solve a wide and challenging area of compositional and analytical problems.

Our research team at Sibelius Academy has recently introduced a new visual programming environment, called PWGL [3]. It is a new version of PW and aims to modernize and improve many of the useful concepts behind PW. PWGL is completely redesigned and is currently based on LispWorks and OpenGL which allows to develop platform independent code that runs in all major operating systems.

PWGL provides a direct interface to its base languages Common Lisp and CLOS. Besides a large library of kernel boxes, the system includes also complex editors, such as a 2D-editor, Score-editor and Chord-editor, which allow to present, inspect and modify musical structures. PWGL editors have their roots in PW editors [2].

PW contains several editors for music notation and break-point functions. A PW editor consists of two main components. A PW editor is represented in a patch by a specialized (1) editor-box that in turn contains an (2) editor-window. Editor-boxes typically have one or several inputs for incoming data and one output that returns an object contained in the editor-box. The inputs are used to feed either objects (generated elsewhere in the patch) or textual formats that allow to build chord-sequences, rhythmical structures, break-point functions, etc. The user can open the editor-box with a double-click or a keyboard shortcut. This operation opens an editor-window which allows the user to edit visually various aspects of the object such as rhythm, timing, pitch, position, etc. Typically the music notation related PW editor-boxes do not show the contents contained in the box (the chord-editor having a visual iconic representation of the chord-object is an exception). If the user wants to inspect the contents of a PW music notation editor-box the respective editor-window must be opened.

Although the PW editor concept has proven to be useful in many practical applications, there are several limitations in the system. PW has suffered from several handicaps both from the user and from the programmers point of view. In the next sections we will discuss some of the problems related to PW editors and how they are solved in PWGL: in the case of music notation related editors PW contains a large number of different box options that leads to patches that are difficult to understand and to maintain; the PW box design scheme makes it difficult (or impossible) to combine different editor types within one box; the PW BPF-editor is limited to only one object type; a PW editor-box can have only one output; the contents of several of the editor-boxes cannot be seen nor manipulated directly in the patch, and so on.

The rest of this paper is organized as follows. We start by discussing in general terms the main PWGL editors, the Score-editor and the 2D-editor. We will compare the former PW editors with the new ones, show how their visual outlook differ, how the new box design affects the use of PWGL editors in complex application boxes, etc. After this we will focus on the case study of this paper, the 2D-editor. We first discuss in more detail the components of the 2D-editor. We describe the various 2D-objects that are currently available in PWGL. We also discuss the grid and patch-level editing options. We finish with some concluding remarks.

2. PWGL EDITOR OVERVIEW

2.1. Integration of editors

One of the main design issues in PWGL is to develop and to improve the former PW editors. A typical example of this trend is to reduce the number of editors that existed in PW. For instance, PW music notation editors contained at least five options to present musical notation data: non-mensural monophonic MN-editor, non-mensural polyphonic MN-editor, mensural monophonic RTM-editor, mensural polyphonic RTM-editor and Chord-editor.

The current PWGL score-editor, by contrast, aims to simplify this scheme by combining the first four options into one editor, called Score-editor. The Score-editor allows to present a large number of different musical material types within one editor, such as chord sequences, melodic lines, time notation (timing information is given in seconds), frame notation (used in both mensural and non-mensural context and is indicated in the score by enclosing a group of pitches or gestures inside a rectangle) and traditional western metric notation. Different material types can even be mixed (for instance frame notation with metric notation). The reduced number of music notation editors simplifies greatly the complex issue of how to process, present and manage musical data within a visual programming language.

A similar approach has been taken with the PWGL 2D-editor which is based on the former PW BPF-editor. The BPF-editor allowed the user to generate, inspect and edit only one type of objects, i.e. break-point functions (bpfs). It was used for various purposes such as defining abstract musical shapes and gestures and to control sound synthesis. The main difference between the BPF-editor and the current 2D-editor is that the latter one supports any collection of objects that have two dimensions, such as bpfs, sound samples, chord-sequences, Bezier functions, markers, scrap-objects, spectrographs, and so on. This scheme allows to combine and synchronize visually various 2D-objects within one editor. The 2D-editor can be used for a wide area of tasks such as sound synthesis control, visualization of mathematical functions, compositional sketches, musical processes, and analysis results.

Occasionally the 2D-editor applications resemble somewhat the OpenMusic tool called maquette, which allows to organize the musical materials computed in patches or built in the editors into higher level, time oriented structure [1]. This similarity is quite obvious for example in the bottom part of Figure 7 (see Appendix), where the user has applied a user-grid that allows to synchronize various materials with a complex metric time structure. There are, however, cases (see for instance Figures 4, 5 and 8) where the OpenMusic maquette and the PWGL 2D-editor approaches are quite different: maquette is more specialized in handling time oriented tasks, while the 2D-editor has a more abstract flavor as it can represent any data with two dimensions.

PWGL provides a programming template that allows the user/programmer to define new 2D-objects. This programming interface is discussed shortly in Section 3.2. Example 2D-editors containing several 2D-object types are given below in Figures 2, 4, 5, 7 and 8.

2.2. Editor input-boxes

In PW, an editor-box typically can contain only one editor-window as the editor-window is associated directly with the editor-box. PWGL, by contrast, is based on a more uniform and flexible box design scheme where all boxes consists of a main-box. A main-box in turn can include one or several input-boxes. Even recursive boxes are possible (i.e. a box can contain instances of itself). PWGL supports several types of input-boxes: input-boxes for textual Lisp expressions (numbers, lists, strings, etc.), menu-input-boxes (both simple and hierarchical menus are supported), buttons, sliders, and editor-input-boxes. The latter input-box type typically owns a specialized editor-window which can be opened and manipulated by the user. Figures 1, 2 and 3 show several PWGL box examples with different input-box types:

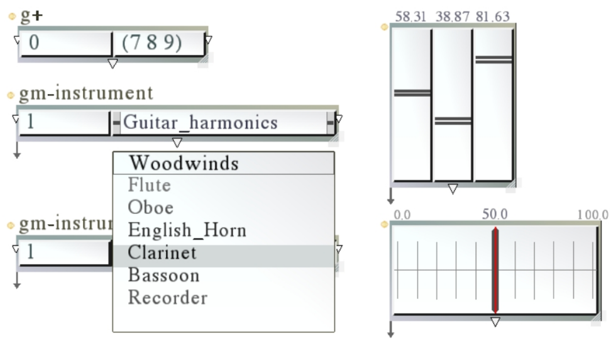


Figure 1. PWGL box examples, to the left: textual input-boxes and menu-input-boxes; to the right: slider-bank boxes with slider-input-boxes.

Figures 2 and 3, in turn, show the three main editors of PWGL. Each of them is a PWGL box containing one editor-input-box.

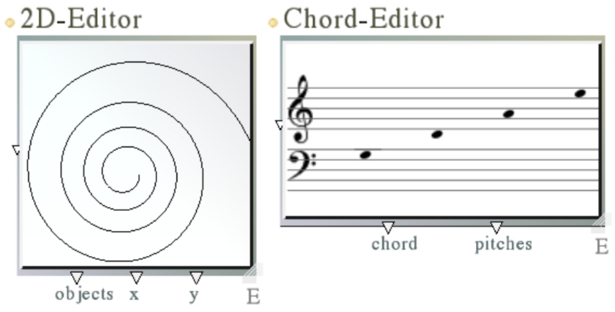


Figure 2. PWGL editor box examples, left: a 2D-editor; right: Chord-editor.

Thus at the patch level all PWGL editors are represented as input-boxes. This scheme has several advantages as it allows to build boxes with arbitrary number

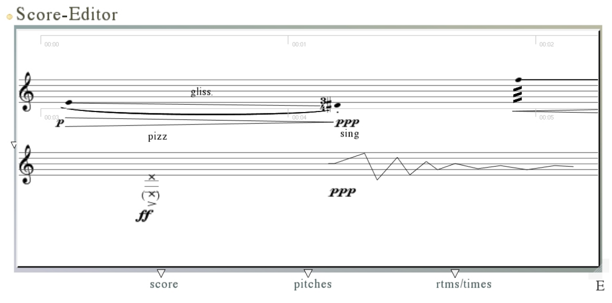


Figure 3. PWGL editor box examples: Score-editor.

of input-boxes that can vary by type and layout options (layout of PWGL boxes is discussed in more detail in [5]). Figure 4 shows a complex PWGL box with several input-boxes. Of special interest are the two music notation related editors (the box contains a Score-editor input-box and a Chord-editor input-box), and the two 2D-editor input-boxes found at the bottom part of the box. The first one contains a 2D-bpf and a 2D-marker, while the second one contains a 2D-sound-sample.

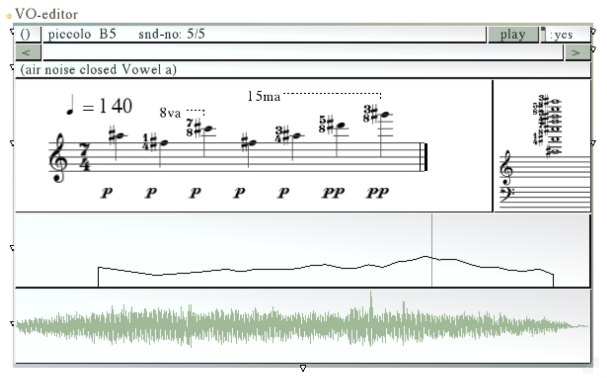


Figure 4. A complex application box with textual input-boxes, buttons, menu-input-boxes, a Score-editor input-box, a Chord-editor input-box, and two 2D-editor input-boxes.

2.3. Visual appearance of editor input-boxes

All editor input-boxes are able to draw directly the contents of their editor-windows on a patch level (see for instance Figures 2, 3, 4 and 5). This means that the user can have a very precise overview of the patch and the contents of the editors contained in the main window without having to open the respective editor-windows. If necessary, this information can be zoomed and/or hidden either by resizing the box or by using a special minimize-button in the top-left corner of the box.

2.4. Local pan and zoom of editor-input-boxes

PWGL provides a uniform way of handling pan and zoom operations using a 3-button mouse. Pan is accomplished by dragging a mouse while pressing the scroll-wheel (second) button, and zoom is achieved with the scroll-wheel. These operations are typically global, i.e. they affect the complete contents of a patch window or an editor-window. PWGL editor input-boxes also support local pan and zoom. This is accomplished by moving the cursor above an editor-input-box. The visual appearance can then be locally manipulated either by dragging the scroll-wheel button (for pan) and/or with the scroll-wheel (for zoom). Figure 5 shows two examples where the user has used both the pan and zoom operations to achieve close-up views of the contents of editor-input-boxes.

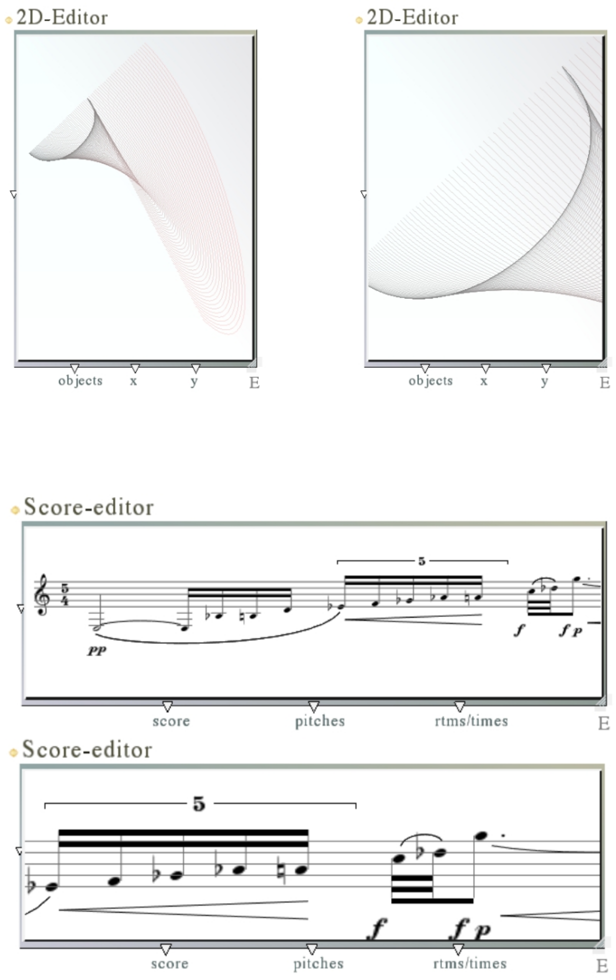


Figure 5. Two local zoom/pan examples: top: a 2D-editor input-box containing 50 2D-bezier functions before and after pan and zoom; bottom: a Score-editor input-box before and after pan and zoom.

3. 2D-EDITOR

3.1. 2D-editor components

This section presents the main components of a PWGL patch containing a 2D-editor. Figure 6 shows a patch window (see the arrow with the label 1) containing a box, called 2D-editor, which in turn contains one 2D-editor input-box (arrow 3). The user has opened the 2D-editor-window (arrow 2), which shows the actual contents of the editor. It consists of two rulers, y-ruler (arrow 5) and x-ruler (arrow 6), allowing a constrained pan or zoom either along the x-axis (x-ruler) or y-axis (y-ruler) utilizing the scroll-wheel as described in Section 2.4. Unconstrained pan/zoom operations (i.e. the operation occurs both along the x- and y-axis simultaneously) are accomplished by utilizing the scroll-wheel in the editor-view area (arrow 4). When the user moves the mouse above the editor-view area the mouse co-ordinates are shown in a text display area (arrow 7). While the editor can contain several 2D-objects simultaneously only one of them can be active at a time. The active object is drawn with a darker color while the inactive ones are drawn with lighter colors. Furthermore, any edition operations, such as clicks, double clicks, keyboard events, and menu actions, affect typically only the active object. Arrow 8 points to a text display area that contains information of the 2D-type of the active object (in this case BPF) and its number plus the total number of 2D-objects present in the editor (these numbers are given as a ratio, in this case 1/2). The editor view can have a grid (arrow 9) which can either be automatic or defined by the user. The example contains two 2D-bpf objects (see arrow 10 and arrow 11).

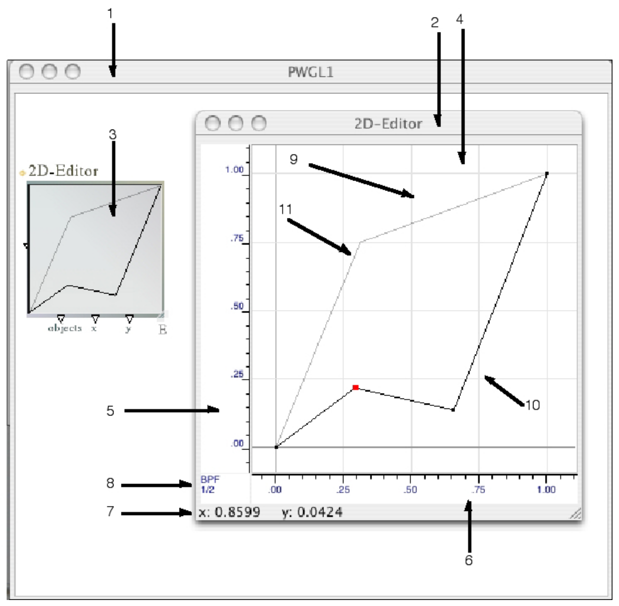


Figure 6. A 2D-editor patch with its main components 1) PWGL patch window, 2) 2D-editor window, 3) 2D-editor input-box containing two bpf-objects, 4) 2D-editor-view, 5) x-ruler, 6) y-ruler, 7) mouse co-ordinates, 8) active 2D-object type, index, name 9) grid, 10) 2D-bpf object no 1 (the active 2D-object), 11) 2D-bpf object no 2.

3.2. 2D-objects

PWGL contains currently the following library of 2D-objects: bpf (break-point function, a collection of time-ordered points connected with linear line segments), bezier (a collection of Bezier functions with two end points and two control points), scrap (a collection of labeled polygons), marker (a collection of labeled vertical marker-lines) and sound sample. In the near future the list will be augmented at least with the following 2D-objects: envelope, spectrogram and point collection.

The system includes a programming protocol that allows the user/programmer to add new 2D-objects. Due to space limitations we can only give here a very brief and simplified description of the protocol. First, the generic class definition, called 2D-object, is to be subclassed. Then at least the following methods should to be specialized for the new class:

- drawing methods: draw-2D-object, draw-active-2D-object
- drag method: drag-selected-2D-subobjects
- editing methods: remove-subobject, add-subobject
- miscellaneous: get-class-name-label, min-x, min-y, max-x, max-y, add-to-menu

3.3. Grids

Besides the abililty to combine various 2D-objects, the 2D-editor also has a grid option that allows to utilize either automatic grids or to define user-grids. Automatic grids are simple Cartesian grids that give a visual clue of the position of the objects in the co-ordinate system. This grid type is automatic as the system attempts to fill the editor view and rulers with evenly spaced help lines and co-ordinate values dependent on the current pan and zoom state of the editor. Figure 6 gives an example of an automatic grid. User-grids or structured grids, by contrast, are used to define grids that have a set of help lines and co-ordinate values that are fixed. User-grids are useful as they allow to synchronize various 2D-objects with the help of a so called snap to grid option (i.e. a 2D-object, when dragged, is moved to the nearest grid x-y-position). The user-grid can be either regular or irregular and can be defined independently both for grid x-values and y-values. These values can either be delta values (such as 0.1) - defining a constant distance value - or a list of absolute co-ordinate values, for example (0 100 300 550). Furthermore, the grid values for the x-values can be given as an ENP-score-notation expression. This notation is an enhanced version of the PW RTM-notation and it can be used among others to define complex metric rhythms (for more detailed discussion see [4]). The latter option allows to use user-grids that are calculated out of the start-times of the attack points of an arbitrary metric rhythm. Figure 7 (given in the last page of this article) shows a score (top) with its rhythmic ENP-score-notation expression (middle). At the bottom of Figure 7 we give a 2D-editor having a user-grid where the y-values are defined as a constant delta value (1.0) and the x-values are calculated from the start-times of an ENP-score-notation expression. The measure start-times are indicated with a darker vertical line and labeled with the respective time signature (5/4) . The attack points are shown with light-gray vertical lines. The 2D-editor contains two 2D-object types (a 2D-scrap object and a 2D-bpf object) that are synchronized to the rhythmic structure given in the top part of Figure 7.

3.4. Patch-level editing

The contents of a 2D-editor input-box can be edited directly on the patch level (i.e. the user operates directly with the editor-box) without having to open the actual editor-window. This kind of editing allows typically to make some rudimentary editing operations such moving an object in the co-ordinate system. While offering only a subset of the editing capabilities of the editor-window this feature can be useful in many situations. For instance, in real-time synthesis control 2D-objects can give valuable visual information of parameter changes. Our final example (see Figure 8 in the last page) shows a 2D-editor-box that contains a sound sample and a 2D-marker object (i.e. a collection of vertical marker-lines). The user is moving one marker-line horizontally above the underlying sound sample. To the right of the cursor there is also a small help window displaying the horizontal position of the marker-line in samples.

4. CONCLUSIONS

We have presented some important design efforts dealing with PWGL editors. These improvements include: integration of available editor-boxes, complex box design, multi-object 2D-editor, visual control of editor-boxes, 2D-object programming protocol, grid options, and patch-level editing.

5. ACKNOWLEDGEMENTS

The work of Mikael Laurson has been financed by the Academy of Finland within the project “CLARA–Control Language for Audio Related Applications”.

6. REFERENCES

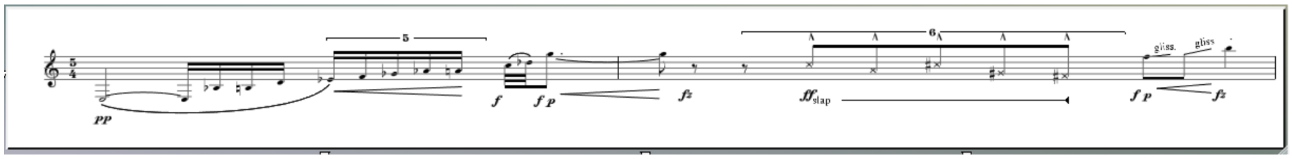
[1] Gerard Assayag, Camillo Rueda, Mikael Laurson, Carlos Agon, and Olivier Delerue. Computer Assisted Composition at IRCAM: From PatchWork to OpenMusic. *Computer Music Journal*, 23(3):59–72, Fall 1999.

[2] Mikael Laurson. *PATCHWORK: A Visual Programming Language and some Musical Applications*. Studia musica no.6, Sibelius Academy, Helsinki, 1996.

[3] Mikael Laurson and Mika Kuuskankare. PWGL: A Novel Visual Language based on Common Lisp, CLOS and OpenGL. In *Proceedings of International Computer Music Conference*, pages 142–145, Gothenburg, Sweden, September 2002.

[4] Mikael Laurson and Mika Kuuskankare. From RTM-notation to ENP-score-notation. In *Journées d’Informatique Musicale*, Montbéliard, France, 2003.

[5] Mikael Laurson and Mika Kuuskankare. Some Box Design Issues in PWGL. In *Proceedings of International Computer Music Conference*, pages 271–274, Singapore, 2003.



```
-> (((2 (1)) (1 (1.0 1 1 1)) (1 (1 1 1 1 1)) (1 (1 1 6))) ((1 (1.0 -1)) (2 (-1 1 1 1 1 1)) (1 (1 1)) (1 (1))))
```

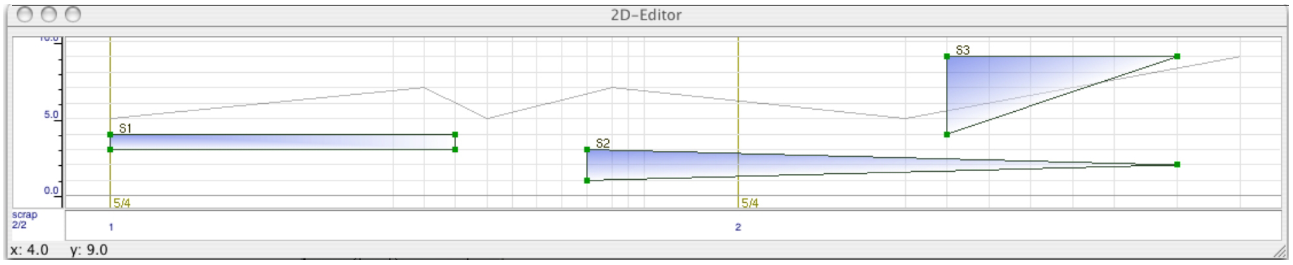


Figure 7. A user-grid example where a metric structure (top and middle) defines the grid x-values of a 2D-editor (bottom).

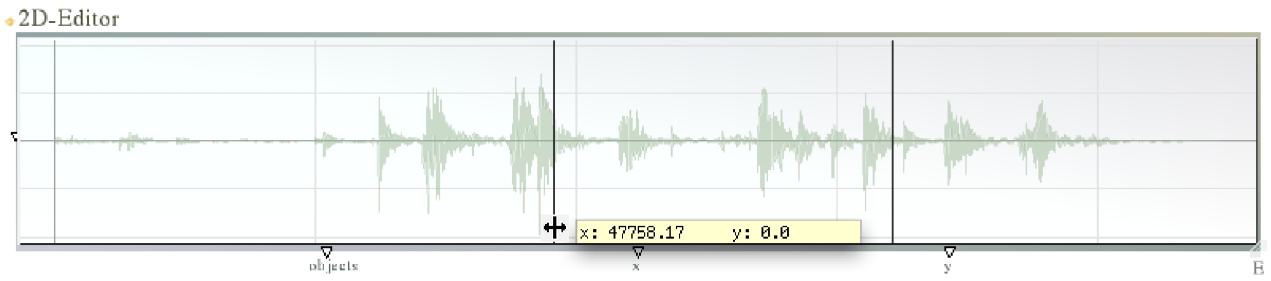


Figure 8. An example with real-time synthesis control using a marker-object in combination with a sound sample.