Keyboard-only Interface For Pure Data

Chun Lee
Goto10
12, rue Charles Gide
86 000 Poitiers
France
chun@goto10.org

Mathieu Bouchard
8257 Saint-Denis
Montrèal
matju@artengine.ca

ABSTRACT

This paper aims to present alternative means to operate Pd based on keyboard input. While the familiar "point-and-click" method serves most scenarios well, it has clear disadvantages for more experienced users. As software in the past have achieved high levels of user productivity using only the keyboard as control interface, this paper will attempt to translate the common programming tasks in Pd to the the domain of key commands. It will also look at how the keyboard driven model of interactivity can enhance the experience of using Pd.

Keywords

PureData DesireData keyboard-input

1. INTRODUCTION

Traditionally, Pure Data has been largely relied upon the mouse input for most of its operations. Although this offers an intuitive "point-and-click" interactivity that is user-friendly, after using Pd extensively, one can quickly notice its drawbacks. One obvious shortcoming is that users are required to constantly switch between keyboard and mouse to operate the program. This becomes especially problematic in the context where patches are developed rapidly. The additional time taken and extra attention required is arguably inconvenient and can distract users from the task of patching. Furthermore, in situations where the use of mouse are not permitted or limited, operating Pd becomes very difficult, if not impossible.

The aim of this paper, therefore, is to investigate alternative means of user-interaction in Pd. It will focus on use of computer keyboard as the main controlling apparatus. The keyboard is chosen simply because it is the most common input device and most software has successfully achieved high degree of user productivity with it being the only control interface.

The investigation will begin by identifying current operations requiring the mouse input; this should establish a methodical understanding on the scope of what is required if another interfacing method is employed. It will then demonstrate how keyboards can be used to achieve the equivalent functionalities. Lastly, it will examine the unique characteristics with keyboard based interaction and suggest how they might be incorporated so that Pd can take full advantage of such controlling methods.

The content of this paper, to an extent, is mainly speculative. In other words, it suggests, from experienced users' perspective, how computer keyboards may be incorporated to improve the existing controlling methods. Despite its empirical nature, the paper will hopefully lead to further research involving quantitative analysis[1] to evaluate the points made throughout this investigation.

2. MOUSE USAGE IN PD

Both the edit mode and run mode of Pd depends on mouse input for most of its fundamental operations. Table 1 summarizes these common tasks.

After observing how mouse is used in Pd, two distinct tasks becomes apparent if keyboard is to be used to control the program. These two tasks can be demonstrated in the following example. In editing a object's name, one has to move the mouse pointer to the target area on the canvas, then left click on the object box so that its name and arguments can be edited. Once the edit is complete, an additional left click on a blank canvas area will reinitialize the object. First, certain navigation mechanism is needed so that user can effectively target object or objects within the patch. Second, some user input is required so that functions such as editing the name of a object can be invoked.

As Pd patches are based on the arrangement of objects on two-dimensional canvases, the mouse is therefore an intuitive device for navigation due to its positional nature. The computer keyboard, on the other hand, is not designed to resolve coordinate based events. Although it is possible to emulate the mouse pointer controlled by simple bindings using the arrow keys, such implementation would be impractical to use. For this reason, it demands a differnt kind of design to suit the nature of keyboard so that effective navigation can be achieved.

[1] Jef Raskin suggested several methods of analysis in his book entitled 'The humane interface'[2], which could potentially be adopted in evaluating the efficiencies between different types of user interaction. In particular, the GOMS keystroke-level model originally proposed by Card, Moran and Newell[1] could provide a comparative test between the use of the mouse and the keyboard in controlling Pd.

Pd Convention 2007. Montréal, Québec, Canada.

Table 1: Common mouse input

Edit mode		
Action	**Description**	**Mouse events**
Object placements	placing new objects, reposition existing objects	move mouse pointer, click and drag
Object high-lighting	single and multiple selection of objects	move mouse pointer, click, shift+click
Object edition	Editing and completing objects's name or argument	move mouse pointer, click
Patch core creation	Connecting and disconnecting object's patch core	click, drag
Additional functions	accessing extra functions such as the properties dialog and the help file	right click, select from pop-up menu
Run mode		
Action	**Description**	**Mouse events**
GUI interactions	Using the GUI elements of Pd such as bang, toggle, number box	move mouse pointer, click and drag

The user input to invoke additional functions for targeted objects can easily be substituted by keyboard shortcuts and commands. In fact, some of these shortcuts already exists in current Pd[2]. Because any generic input events can be used to associate with the execution of arbitrary functions, it would only require binding the necessary commands to recognize specific key strokes so they can be accessed without the mouse.

This notion consequently forms the objectives in devising keyboard alternatives to achieve the equivalent functionality found with the use of mouse. First, a efficient method of navigation needs to be designed to allow user to target objects effectively. Secondly, key bindings will be used to execute and perform the required tasks once a selection is made.

3. KEYBOARD NAVIGATION

Various methods can be conceived to navigate within an existing patch using key controls. One of the simplest way is to divide the search space into quadrants with 45 degrees offset, with the center of the quadrants being the object currently targeted. Four key commands[3] can then be assigned to the four directions of each quadrant. According to the key pressed, the search routine will return the closest object from the center in a given direction. The "target" may then be switched to the returned object. Following this method, users can navigate from object to object by indicating the directions.

This strategy highlights some key issues in keyboard based patch navigation. Assuming the center of the quadrants is always the object currently targeted, the search routines have to take care of the starting point of each navigation. This could simply be solved by storing the end point of the previous selection, thus each navigation always resumes from the last known coordinate. The routines would also perhaps need to distinguish between objects and connections, so that they can be selected differently. Furthermore, the object that currently is targeted is not the same as selected object. In other words, being targeted is the same effect as having the mouse pointer "hovering" over an object box. Although the targeted object would most likely be selected next, it is nevertheless still important to point out the differences[4]. To address this notion clearly, the object that is currently targeted will be referred to as "target" object in this paper. Objects in the selection are naturally referred to as "selected" objects.

Instead of a simple search in the quadrant, other methods could include following certain tolologies of the patch. For instance, the target object is chosen according to the connections or send/receive pair in the patch. Furthermore, different types of navigation methods could be used in combination to increase the speed in reaching the target object.

The main difficulty in devising efficient methods of navigation mostly lies in the arbitrary placement of objects. As the position of objects on the canvas are freely chosen by the user, it is therefore somewhat problematic to construct generic methods in which the navigation is completely reliable and optimized. For instance, if the position of objects are limited to a grid fashion, navigating between objects will be much simplified and more effective. For this reason, it is also conceivable to first quantize the position of objects, then implementing a visual cursor which steps across the grid to put objects in target. The advantage of such quantized visual cursor is that there is nothing to distinguish between existing objects and blank space on the canvas. This allows a generic navigation technique to emerge, which can be applied to a wider range of tasks.

4. KEYBOARD SHORTCUTS AND COMMANDS

Besides patch navigation, establishing effective keyboard shortcuts to execute commands and functions is also vital in the efficient controls of the program. These keyboard shortcuts can be invoked after objects have been targeted so that operations such as selection, highlighting inlet/outlet, loading help files and editing object's name may be accomplished.

Key shortcuts may also be implemeneted to select item from exisiting program menus. For instance, each manu can be associated with a particular key binding and once activated, arrow keys can then be used to assist in selecting the desired item. This method of accessing menu items has

[2]For example, the ctrl+c, ctrl+v, ctrl+d to copy/paste and duplicat object. and the arrow keys to reposition selected objects

[3]For example, Ctrl+arrow keys

[4]In implementation, currently targeted object can be selected by default. however, as the "target" moves to the next object, the current object must also be deselected first unless specified.

Table 2: Summary of keyboard based control

Area of work	Discription
Patch Navigation	To effectively put objects in target
Key shortcuts	To invoke additional funtions To access program menus
Command evaluator	To invoke commands using a prompt

long been a standard practice in many software to increase usability. The lack of these type of shortcuts in current Pd would also be a main area for improvement.

5. CANVAS COMMAND EVALUATOR

So far the discussion has focused in modeling the control methods found in the use of mouse via the keyboard. The keyboard navigation followed by key commands reflects how objects are manipulated with moving mouse pointer plus additional mouse events.

A prime example is the use of the command prompt. A command prompt consists of a text field in which commands can be entered via the keyboard and is then evaluated. The benefit of such a command evaluator is somehow more subtle. As the functionalities of given software expand, it has to effectively present the features to its users. Both direct key bindings and menu selection via the mouse falls short of being efficient solutions. This is due to the fact that they are not scalable. With key bindings, convenient key sequences quickly run out without resorting to prefixing additional key strokes. This makes command invocation lengthier and harder to memorize. Menu selection, on the other hand, makes accessing features a cumbersome process consisting of long mouse movements and repetitive clicks. Even with key shortcut based menu selection, reaching the desired item can still be a fastidious process, especially if the item belongs to nested menus. Command evaluation, armed with auto completion and history recall, exposes all methods and functions in a unified and accessible manner. Furthermore, as features increase, it does not produce additional elements to the existing graphical user interface. This is beneficial in keeping the program interface simple and streamlined.

The advantage of command evaluator becomes more apparent when methods can be defined externally. In such a scenario, the extended commands can still be invoked in just the same way as standard functions. Moreover, if the evaluator supports defining customized macros, one can quickly group standard sequences of commands as macros and use it with the prompt[5].

As a result, it is foreseeable to have an extensive set of commands that are accessible through the evaluator to facilitate the productivity and improve the experience of using Pd. Moreover, one can also adjust the internal settings of Pd in runtime using the prompt. Through such implementation, the first impression of Pd can still be kept minimal and user friendly to newcomers, whilst experienced users can gain greater control and flexibility through the command evaluator.

6. CONCLUSION

This paper has briefly discussed some methods and directions in which keyboards may be used as the sole control interface of Pd. Table 2 summarizes the areas to be explored and improved.these There are certainly other approaches utilizing keyboard input which remain to be further examined[6]. Techniques mentioned in this paper nevertheless serve as a starting point towards preliminary experimentations on the subject.

Due of the current design and architecture of Pd, implementing such features would involve significant undertakings. Furthermore, without initial testing and evaluation, it still remains largely uncertain as to both the usefulness and practicality of such feature. For this reason, most of the ideas described in this paper have been implemented in DesireData as proof of the concept[7]. This is because the design of DesireData allows additional client features to be quickly developed and tested.

Although the proposal of the keyboard-only interface for Pd appears to encourage productivity from the user's perspective, it can be argued that overall it adds additional components and complexity to the program, thus making it less stable and harder to maintain. Moreover, since the patching process in Pd is inherently graphical, the computer keyboard might not be the most suitable device to control the general behavior of the program. Because editing a patch isn't a generic task like text editing, for instance, Pd's programming paradigm may nor significantly benefit from these keyboard based control methods. Such arguments are entirely valid and need to be further investigated. However, the current lack of keyboard controllable functions in Pd makes its usage suboptimal at times. In particular, it is not uncommon for artists to spend extensive periods of time developing patches. Any minor inconveniences would thus exaggerate and affect the overall programming experience. For this reason, any efforts to improve the usability of Pd should still be tested and carried out. Consequently, this is the main goal and motivation of this paper.

7. REFERENCES

[1] S. Card, T. Moran, and A. Newell. *The psychology of human-computer interaction*. Hillsdale, 1983.

[2] J. Raskin. *The Human Interface*. Addison Wesley, 2000.

[3] J. Tidwell. *Designing Interface*. O'REILLY, 2005.

[5]Keyboard macros can be created by recording the key shortcuts and upon invocation, it simply plays back the recorded commands in sequence

[6]Although not focused entirely on keyboard based interaction, Jenifer Tidwell offered an wealth of patterns towards interface design in her 2005 publication[3], which can be used as further guild lines on the subject.

[7]Severl key features described in this paper are already implemented in DesireData. see http://desiredata.goto10.org for further detail