# A Graphical Interface for Real-Time Signal Routing

Jean-Marc Pelletier
International Academy of Media Arts & Sciences
3-94 Ryoke-cho, Ogaki-shi, Gifu
503-0014, Japan
+81-584-75-6600
jmp@iamas.ac.jp

## ABSTRACT
This paper describes DspMap, a graphical user interface (GUI) designed to assist the dynamic routing of signal generators and modifiers currently being developed at the International Academy of Media Arts & Sciences. Instead of relying on traditional box-and-line approaches, DspMap proposes a design paradigm where connections are determined by the relative positions of the various elements in a single virtual space.

## Keywords
Graphical user interface, real-time performance, map, dynamic routing

## 1.INTRODUCTION
The DspMap system described in this paper started out as part of the DspBox project [1]. The goal of this project was the development of a stand-alone device that incorporates all necessary hardware and software components for real-time audio production in a single unit. As such, the DspBox prototypes are equipped with a number of tangible elements such as faders and buttons that interface with the software. The main vector of interaction, however, is its rather large touch screen.

**Figure 1. The DspBox (prototype #1)**

The DspBox software allows users to develop their own modules in the form of Max/MSP [2] patches. Those modules, which can function either as signal generators or modifiers, are loaded and connected with each other prior to performance. In its original form, the DspBox software, owing in part to limitations inherent to MSP, could not allow objects to be added, removed and connections to be altered once audio production started. Furthermore, screen space limited the number of objects that could be accessible by the GUI.

In order to solve some of those problems, the DspMap system was proposed to make working with large numbers of audio generating and processing modules, as well as complex routings between those modules, simple, intuitive and seamless.

## 2.CONCEPT
The goals of the DspMap interface were that it should provide: interaction through simple and intuitive pointer (touch screen, mouse, etc.) gestures, simple and clear visual feedback, support for an arbitrarily large number of modules and the potentially complex connections between those modules.

In order to achieve the goals stated above, we looked at some of the problems of available GUIs. Perhaps the majority of modular synthesis software packages implement variations on the box-and-line approach to visual programing languages (VPL) [3]. Those include, not exclusively, Max/MSP, pd [4], CPS [5], Audiomulch [6], Reaktor [7] and Bidule [8]. While this approach to GUI design may seem natural for modular sound synthesis due to its analogy to analog modular synthesizers, it is in no way specific to audio applications, being implemented in data flow programming packages like LabVIEW [9] and video processing software like Isadora [10] and EyesWeb [11].

While the box-and-line approach has definite advantages, it is in our experience only marginally suited for on-the-fly design and performance. There are two reasons for this: 1) large numbers of modules and complex routings quickly lead to visual clutter and 2) since connections between modules must be made explicitly and individually, relatively complex gestures must be used to modify the structure of a "patch". The second problem was addressed by M. Kaltenbrunner et al. in their reacTable [12] whose software modules established connections between each other autonomously. While this approach does indeed solve part of the problems, we decided to work from a different paradigm altogether.

Instead of using the metaphor of electronic signal flow, where connections between modules must be made explicitly via real or virtual patch cords, we opted for a design that mirrored the behavior of sound producing objects in nature. This behavior has two characteristics that are relevant to us: 1) Mechanical sound sources are audible by default. The medium of transmission (air) does not need to be explicitly established between source and receiver. 2) The acoustic properties of the environment in which a sound source is placed will affect the signal before it reaches the


*Nime'05*, May 26-28, , 2005, Vancouver, BC, Canada.

receiver. From those two observations, we were able to design a user interface that effectively met the goals we set above.

As the name implies, audio modules in DspMap are organized in a "map" rather than a "patch". Here, "map" refers to the virtual space where objects are arranged. The user can scroll across the map or zoom in and out using simple gestures. There is no theoretical limit to the size of a map or to the degree of magnification, resulting in a seamless integration of macroscopic and microscopic structures as well as support for potentially large numbers of modules in limited screen space.

# 3.GUI OBJECT TYPES

Four types of objects can be placed on a map. They are:

*Labels*: Text comments that do not affect audio signals in any way. In the current implementation, they cannot be edited at run-time, though they can be moved around.

*Points*: Points correspond to modules that offer some form of audio output but accept no signal input. Points are thus typically signal generators. At run-time, points can be dragged freely to a new location.

*Listener*: The listener is a unique object of central importance. The amplitude of each point is scaled according to the distance from that point to the listener. The farther a point is to the listener, the weaker its output signal will be. If a point is situated beyond a certain distance from the listener, it is deemed to be out of "hearing range" and its signal processing should be temporarily turned off. This allows a very large number of points to be used in a single map without overly taxing the CPU. Any point within hearing range of the listener will be audible. Like points, the listener object can be dragged to different locations. Unlike points, it can also jump from any two locations on the map, allowing for abrupt changes.

*Boxes*: Boxes correspond to modules that accept signal input and provide output. They are used for signal modifiers and should not output any signal without input from a point. Boxes, which appear as semi-opaque rectangles, can be dragged around the map like points. Individual edges can also be dragged , allowing the shape and size of a box to be modified at run-time.

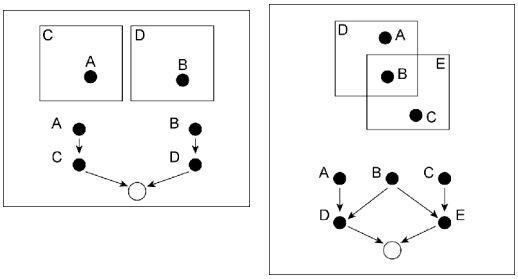
# 4.RULES FOR CONNECTIONS

The connections between points, boxes and the listener are determined according to the following rules:

(1) The scaled output of any point that is enclosed by a box is routed to that box.

(2) In the situation where two or more boxes overlap, the scaled output of any point situated within an overlapping region is routed to each of the boxes concerned, in parallel fashion.

**Figure 2. Point and box vs. flow chart for rules #1 and #2**




(3) Boxes that are connected to one point or more are connected to the listener, provided at least one point is within hearing range.

(4) However, the output of any box that is completely enclosed by another box is routed to the larger box rather than the listener.

**Figure 3. Point and box vs. flow chart for rule #4 and listener distance**

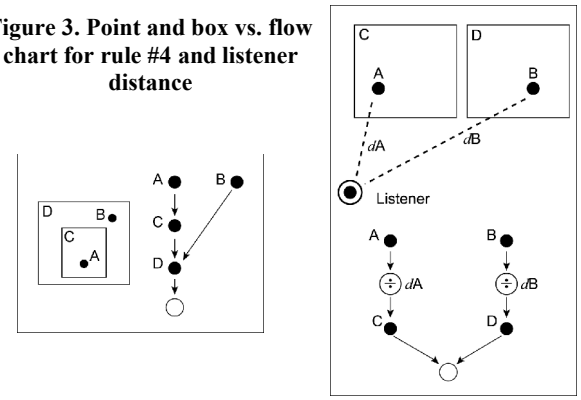
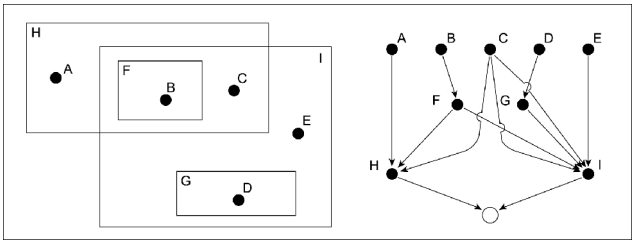



**Figure 4. Point and box vs. flow chart for complex routing**




# 5.SYSTEM ARCHITECTURE

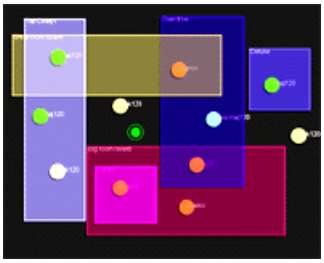
The current version of DspMap runs from within Max/MSP and the graphics are created using OpenGL [13]. The choice of Max/MSP as a development platform was entirely motivated by a need for compatibility with existing DspBox modules.

There are two main components to the DspMap system. The first manages the GUI state, while the second carries out the actual signal processing. The first component, currently implemented as an external, passes messages to the second, native, component. Should DspMap be ported to other environments, the first component would remain largely unchanged, while the second would be entirely dependent on the structure of the host system.

The first component is made up of the following parts:

- *The map window*
  The map window is the main performance environment described above. While, currently, only operations that pertain to object positioning can be performed within the map window, we plan to implement all program features as map window operations.

**Figure 5. The map window**



- *The edit window*
  The edit window is a standard tool bar-type of window that is used to create and delete objects as well as access file operations. It can only be used when designing maps, being disabled at run-time. Eventually, its functionality will be incorporated into the map window.
- *File I/O*
  Maps created with the previous units can be stored in a file and recalled for later use. The file format is text-based and is human-readable. This allows the files to be edited manually. The simple text format also allows files to be created with software other than DspMap itself, which may be of use for certain types of algorithmic composition.
- *The map manager*
  This is the main software unit that maintains the current map state and computes routings. It outputs messages to the host application for it to update connections.

The second, host-specific component can be thought of having:

- *Audio modules*
  Those are the actual modules that correspond to points and boxes on the map. In the Max/MSP implementation, they are abstractions that only require inlets and outlets for boxes and outlets only for points. Everything else is left to the user to design. Audio modules show up in the edit window simply by being placed in folders labeled “points” and “boxes”.
- *The router*
  The router receives three main kinds of messages from the map manager. The first type of message is construction commands that instruct the router to create or delete audio modules. The second type consists of position messages, which the router must use to scale the outputs of points. The last type is connection messages, that the router uses to send data to and from the various modules. The router implementation is dependent on the host application.

## 6.PERFORMANCE APPROACHES

The DspMap allows certain types of performance techniques and strategies that would be difficult on other graphical user interfaces.

Composition for live performance and improvisation can be approached through the use of “clusters”. By clustering groups of points and boxes in certain areas, the performer can easily switch from different musical sections by moving the listener to and from various clusters.

On a micro scale, even rather complex routings can be rapidly made sense of through the overlap and enclosing of boxes. Visually, boxes being semi-opaque, various combinations will appear as different colors – effectively mirroring the way various combinations of signal modifiers will “color” the output of a sound source differently.

Amplitude effects such as fades and tremolos are easily achieved by moving either individual points or the listener object. By adjusting the zooming scale of the map, the user can control the precision of those effects; by zooming in very close, very minute modulations can be achieved.

The performance technique most specific to the DspMap is something we call “rapid routing”. By dragging points across boxes, or combinations of boxes, the routing of this point's outputs to various effect can be changed very rapidly. This can be used to create various interesting effects. Similar results can also be achieved by moving/resizing boxes themselves to connect or disconnect several points at more or less the same time.

## 7.FUTURE WORK

The DspMap is still very much in its infancy and much work remains to be done to improve usability. It is, however, in its current state, already a very expressive tool that meets its objectives. In the future, we plan to improve the DspMap in the following ways:

*Porting the software to various platforms.* In its prototypical form, the DspMap software was written in Javascript to run from within Max/MSP. However, the program has been rewritten in C++ to improve efficiency. This rewriting should also allow the DspMap to be used from within other environments, such as pd or SuperCollider [14].

*Allowing maps to be edited at run-time.* Currently, points and boxes cannot be created or deleted once a performance has started. This can somewhat be circumvented by moving objects to and from a “waiting area”. However, performance possibilities would be improved if the user was able to dynamically create and delete objects at any time. This may be problematic on platforms like Max/MSP that do not really allow modification of the DSP chain at run time, but seems possible in environments like SuperCollider or ChucK [15].

*Improving automation.* The current system allows any object to send messages either to itself or other objects. This feature can be used to build objects that move by themselves. The general interface for automation needs to be improved, however, to maximize its possibilities.

*Access to parameter control.* This is perhaps the greatest current limitation of DspMap: object parameters cannot be controlled from within the map interface. The workaround, bringing up control panels for individual objects, is rather clumsy. In the future, we must find a mechanism by which parameter control information can be accessed and routed intuitively from within the map interface.

*Surround audio.* While the distance between points and the listener is used to modify the output of those points, the relative

direction has no value. This has proved, in practice, to be a useful feature. However, because of its topographical aspects, a natural use of the DspMap would be in the production of surround, or multi-channel audio. This will likely be implemented as a separate operation mode, keeping the current model – individual objects are entirely responsible for their spatial placement – for normal operation.

*Improving visual feedback.* The use of different colors to tell objects apart has limitations when there is a large number of objects. In future versions, we plan to introduce more elaborate visual elements, such as icons, and further make use of OpenGL's 3D capabilities.

*Multi-user environments.* While the DspMap currently supports only a single listener and user, in subsequent versions it may be possible for several users to access and modify a single map at the same time. This may be done over a network, in a fashion similar to on-line gaming.

## 8.CONCLUSIONS

We have found the visual interface described in this paper to be especially well suited for the performance of live computer music. Even as a rough prototype, the DspMap proved to be an expressive, intuitive and easy to use system that made new techniques for musical expression possible. We believe that DspMap's flexibility and flow fits within a general trend towards less rigid programming paradigms and may possibly be used to fully express the possibilities of future research in this domain.

## 9.ACKNOWLEDGMENTS

I would like to thank members of the DspBox project team, Masayuki Akamatsu and Shigeru Kobayashi for their support. I would also like to thank International Academy of Media Arts & Sciences student Yosuke Hayashi for helping out in the realization of the DspMap prototype.

## 10.REFERENCES

[1] Akamatsu, M. DspBox http://www.iamas.ac.jp/project/dspbox/

[2] Puckette, M. Combining Event and Signal Processing in the MAX Graphical Programming Environment. *Computer Music Journal* 15(3), fall 1991, 68-77

[3] Burnett, M., Baker, M. A Classification System for Visual Programming Languages. *Journal of Visual Languages and Computing,* September 1994, 287-300

[4] Puckette, M. Pure Data: another integrated computer music environment. In *Proceedings, Second Intercollege Computer Music Concerts*, Tachikawa, Japan, 1996, 37-41

[5] *CPS* http://cps.bonneville.nl/

[6] Bencina, R. Oasis Rose the Composition - Real-Time DSP with AudioMulch. In *Proceedings of the Australasian Computer Music Conference -ANU,* Canberra, Australia, 1998, 85-92

[7] Reaktor http://www.nativeinstruments.de/index.php?reaktor_us

[8] Bidule http://www.plogue.com/bidule/

[9] Johnson, W. Hanna, P. Millar, R. Advances in Dataflow Programming Languages. In *ACM Computing Surveys, v.36 n.1,* March, 2004, 1-34

[10] Farley, K. Digital Dance Theatre: The Marriage of Computers, Choreography and Techno/Human Reactivity. In *Body, Space and Technology, 3(1)*, 2002, 39-46

[11] Camurri, A. Hashimoto, S. Ricchetti, M. Ricci, A. Suzuki, K. Trocca, R. and Volpe, G. EyesWeb: Toward Gesture and Affect Recognition in Interactive Dance and Music Systems. In *Computer Music Journal*, Vol.24 No.1, 2000, 57-69

[12] Kaltenbrunner, M., Geiger, G., Jordà, S. Dynamic Patches for Live Musical Performance. In *Proceedings of the 4th Conference on New Instruments for Musical Expression (NIME 04)*, Hamamatsu, Japan, 2004, 19-22

[13] Renate, K. Frazier, C. OpenGL Reference Manual, 2nd Edition, Addison-Wesley, 1992

[14] McCartney, J. SuperCollider: a New Real Time Synthesis Language. In *Proceedings of the International Computer Music Conference 1996 (ICMC96)*, Hong Kong, 1996, 257-258

[15] Wang, G., Cook, P. On-the-fly Programming: Using Code as an Expressive Musical Instrument. In *Proceedings of the 4th Conference on New Instruments for Musical Expression (NIME 04)*, Hamamatsu, Japan, 2004, 138-143