

Virtual Patch-Cords for the Katosizer

by
David Blythe, John Kitamura, David Galloway, and Martin Snelgrove

Computer Systems Research Institute
University of Toronto
Toronto, Ontario
M5S 1A4

Abstract

An object-oriented graphics system is presented which allows a musician to program a very versatile signal processor/synthesizer by drawing a block diagram of the desired 'patch'.

Introduction

The *Katosizer* [Kita85, Kita86], is a highly programmable digital synthesizer and signal processor. Its architecture is shown in Figure 1: a high-speed 'pipelined bus' (p-bus, [Rose85]) interconnecting specialized signal and control processors. We have augmented it by adding an Atari 520ST personal computer to handle graphics and disks. This paper describes our implementation of a sophisticated human interface and control program that runs on this system. It provides a natural way for musicians to exploit the power of a highly programmable and rather complex machine in real time.

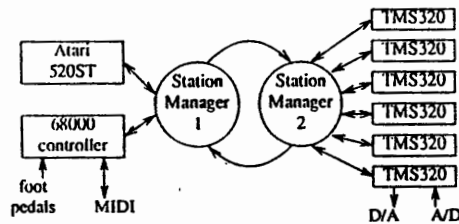


Figure 1 - Katosizer Architecture

Related Work

Our human interface is an example of a 'direct manipulation system' [Shne83], of which the Pinball Construction Set [Budg83] is another example. A similar, but single-machine, interface is provided in the Soundscape [Mime86] program. The diagrams we use are a little like those in OEDIT [Verc75] which attempted to generate (non real-time) MUSIC V [Math69] programs. Another relative of the virtual patchcord may be found in the work of Haerberli [Haeb86], in which a graphically-driven connection manager is used to design computer graphics algorithms for IRIS workstations. Graphical editors for parameters, envelopes and so on are relatively well-known [Buxt82, Gres85, Prus85].

The User's View

Figure 2 is a screen dump showing the user's view of the Katosizer system. It shows a 'patch' in which icons representing physical objects and programs have been connected together by 'virtual patchcords'.

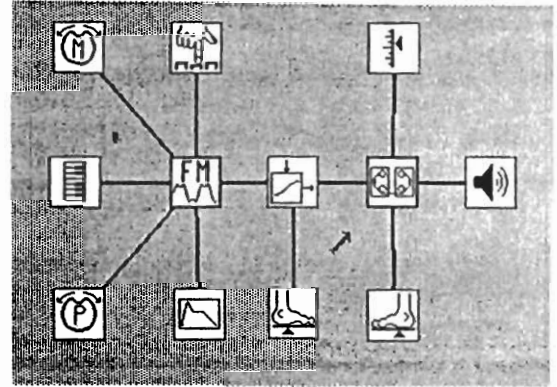













Figure 2 - A Patch

A MIDI keyboard  is shown driving an FM synthesizer  (actually consisting of Katosizer software). Its carrier envelope may be changed with an envelope editor . Preset envelopes and other parameters may be selected by using a MIDI program-select button . The keyboard's pitch-bend  and modulation wheel  are also connected to the synthesizer.

The synthesizer output is fed into a distortion box  controlled by a foot pedal . This in turn enters a 'tape-loop'  whose loop gain is controlled by a slider , and whose input gain is set by another pedal. The result goes to the speaker .

Automatic Patching

Our system doesn't just draw these pictures, it implements them. Figure 3 shows how the patch is implemented on a Katosizer. The connection manager (conman) within the Atari workstation interprets the graphical representation and sends the appropriate code to the 68000 controller, which in turn communicates with the signal processors.

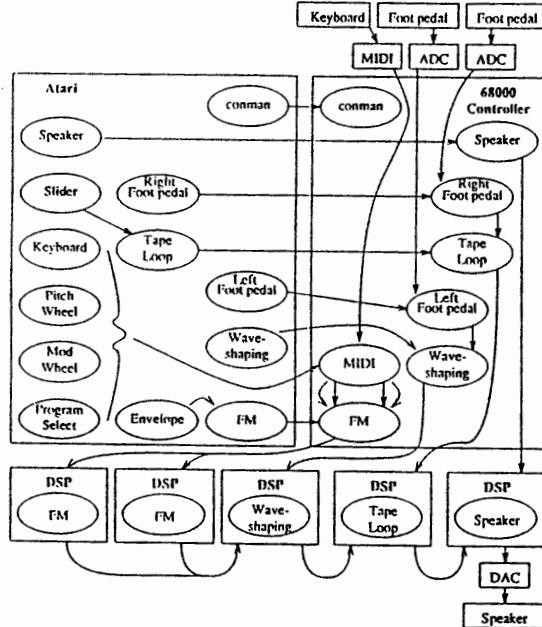


Figure 3 - Patch Implementation

The MIDI keyboards and foot pedals are off-the-shelf hardware. They are plugged into MIDI receivers and low-speed 8-bit analog-to-digital converters on the 68000-based real-time controller. The synthesizer, distortion module and tape loop are assembly code running on high-performance TMS320 [TI83] digital signal processors (DSPs). The output (speaker) icon refers to a 16-bit audio-quality digital-to-analog converter driven by a DSP (at a 35-45KHz sampling rate). The slider is a piece of graphics software running on the Atari, whose value can be changed with the mouse.

This system boasts three different types of machine, running code written in three different languages. The DSPs run hand-tuned assembly code for high speed. The controller code is written in Concurrent Euclid [Cord81], which is a Pascal-like language featuring modules, processes and Hoare monitors [Hoar74]. This was a convenient choice because our 'objects' correspond neatly to processes and modules, and monitors offer a good way to synchronize processes. The Atari software was written in C because we had good graphics code available (our port of Bell Labs' BLIT system [Pike84]) and because the dirtiness of C allows faster graphics response.

The DSPs and controller communicate through the p-bus. It is a high speed pipelined synchronous bus that allows processors to send 16-bit data packets together with 8-bit 'opcodes' to any of 64 destination processors. We use opcodes to specify the type of a message.

Our software protocol is based on UNIX 'sockets' [Leff83]. A socket is a mechanism allowing processes to read and write to each other as if they were files. We chose this as a scheme of proven power that has the incidental advantage that we will also be able to reuse a lot of this code as a UNIX graphical 'shell'. The controller and Atari have UNIX-like socket procedures, while the DSPs run a faster but less-general version of the same protocol. This process makes the exact location of a process inconsequential to the processes with which it communicates.

Examples of Objects

Waveshaping

Controlled distortion effects are obtained by a DSP program which takes an audio input u and produces an audio output y according to:

$$y = kf(u) + (1-k)u$$

where the value k sets the amount of distortion, from a clean signal at $k=0$ to a fully distorted signal at $k=1$. The function (f) is implemented with a 2K-sample table lookup with interpolation.

This program has three kinds of inputs. It can accept sample values (u), distortion levels (k) or new distortion tables (f).

The standard way to specify the distortion function is through Chebychev polynomial components [Road79]. This can be done in our system by touching the waveshaping icon with button 1. This transforms the icon from its default dormant state to an editing window where the user can drag sliders representing magnitudes of 10 Chebychev components and see the resulting function. Figure 4 shows the transformed version of the waveshaping icon.

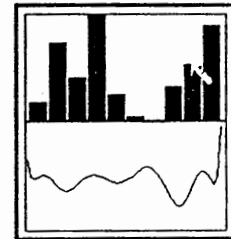


Figure 4 - Edit Mode of Waveshaping Icon

Reverb/Tape Loop

The reverb/tape loop is similar in implementation to waveshaping. A circular buffer of up to 64K samples is used to store and loop audio samples. The software implements the equations:

$$x(t) = k_1 x(t-nT) + k_2 u(t)$$

in which x is the signal in the reverberation loop recycled every n samples (up to a couple of seconds) with a gain k_1 (typically between 0.9 and 0.990), and the input is fed in with gain k_2 . The output is then formed by adding the reverb signal to the input by

$$y = x(t-nT) + u$$

This piece of code accepts inputs u , k_1 and k_2 .

Foot Pedal

Foot pedals are standard volume pedals connected to a 2KHz 8-bit A/D converter on the controller. Each of the eight A/D channels corresponds to a software process on the controller. These processes monitor their pedals for changes, and send appropriate messages. They can also receive messages that set their minimum and maximum values.

Note that, while DSPs run a single process each at audio rates, the controller runs a dozen or more processes at control rates.

MIDI Input

The controller has two MIDI input ports. Each port receives and transmits keyboard, button-push, thumbwheel and other data from up to 16 channels. In order to simplify interaction with MIDI, our software demultiplexes these streams of data and the user can individually manipulate each type of MIDI event, each with a separate icon.

The MIDI keyboard objects produce 'note-on', 'velocity' and 'note-off' events. It accepts inputs allowing key-transpose and velocity scaling.

Graphical Slider

The sliders are implemented in C code running on the Atari. They provide the user with a virtual 'valuator' which can be used just like a foot pedal. It also has inputs allowing 'min' and 'max' values to be set. By default they are 0 and 1023, and are interpreted as fixed point numbers between 0 and .999.

Recorder

This is a multitrack event recorder running in the control processor. Any of the control devices (MIDI, foot pedals, sliders, etc.) may be connected to the recorder, and any value changes recorded for later playback. The recorded output can be reconnected to any object in the same way as any of the input devices. The recorder also possesses some simple editing features such as joining or mixing of tracks and deletion of events. These features are activated by transforming the recorder icon into its edit window form.

Envelope Editor

The envelope object is an Atari object used for creating new envelopes, functions, or waves. The user may store or retrieve an existing function or draw a new one on the screen using the mouse. Each time the envelope associated with the object is changed it is transmitted out its port to all objects connected to it. Figure 5 shows an envelope being created.

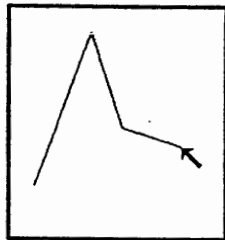


Figure 5 - Creation of Envelope

Frequency Modulation

The FM [Chow73] object has one part running in the controller and other parts running in DSPs. Each DSP implements two two-oscillator FM voices with arbitrary wavetables and envelopes. It produces a single (mixed) audio output and accepts 6 input parameters: $c:m$ ratio, carrier envelope, modulator envelope, frequency, key velocity and pitch bend.

Allocation of voices to note events is done by the portion of the object resident in the controller. This code can control an arbitrary number of voices.

Similar objects do Karplus-Strong [Karp83] string synthesis, percussion simulation [Kita85], sampling, and wavetable synthesis.

Microphone

One of the DSPs controls a pair of 50KHz 16-bit A/D converters. This allows the system to process externally generated signals.

The Connection Manager

Figure 6 shows the appearance of the Atari screen just as a connection is about to be made. The user has already dragged icons for two objects from the menu, connected one end of a 'cord' to the first object, and run the cord to the second object (pegging it down in a couple of places so as not to trip over it later). He/she is just about to plug it into the second object. This was started by pushing button 2 at which point the pop-up menu of jacks appeared, selecting the appropriate jack by dragging the mouse up and down (current jack has black background), and the connection will be completed when button 2 is released.

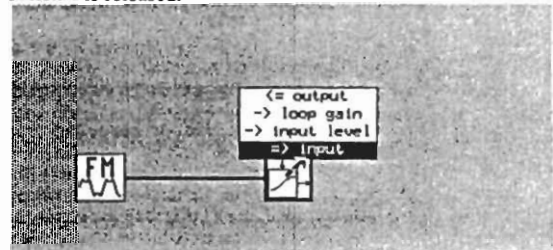


Figure 6 - Connecting a Patch

The system's side of this dialogue is handled by a pair of processes, one in the Atari and one in the controller, which together form the connection manager 'conman'. They maintain a list of connections and allocate resources. Other processes register sockets and their names with conman. These are the names that appear on the pop-up menu when the user goes to make a patch. The Atari portion of conman also contains the graphics code for drawing wires.

When the icons for the two objects were initially selected, the controller portion of conman allocated DSP processors for them and loaded the appropriate code into them. When the connection is made, a message will be sent to the source DSP indicating its proper destination.

The audio application places stringent restrictions on how connections may be made and data changed. Conman is responsible for obeying these rules. Changing the distortion function in waveshaping, for instance, takes about 2500 sample times. We therefore change tables by allocating a *new* waveshaper, filling its tables, and then switching over. As a

second example, unplugging an audio cable should be preceded by fading the original signal down to avoid 'pops'.

Conman is also responsible for enforcing some primitive type conventions. In particular, we distinguish between low-speed (control) data streams and high-speed (audio) data, and do not allow interspecies connections. This is necessary because the audio streams are too fast to be handled by the very general protocols we use for control (or even by the Atari hardware).

Conclusions and Future Work

The virtual patch-cord concept has proven to be a convenient and powerful technique for the integration of many tools, just as the pipe has become an indispensable part of Unix. It is not just an emulation of an outdated hardware concept, but is applicable to any field which makes use of interactive computer graphics. However, there is a great deal of work left to be done.

The software is still in an infantile state. Much work is necessary to make the system more robust, cohesive, and ultimately more powerful. Three areas which are in need of more research are:

- A hierarchical macro facility for grouping collections of objects and their interconnections. This gives the user a powerful way to store 'patches' or (favourite) configurations. It is also a way to control the visibility of objects. If an interconnected collection of objects can be represented by single icon, it affords the user with a way to unclutter the screen. We would also like to use this mechanism to let us edit the algorithms within our existing objects at the operation level: for instance to add feedback to the FM synthesizer.
- Currently the system requires total recompilation if a new object is to be added to the system. This is largely due to the complexity of the interaction between the control processors and the DSPs (e.g. the control processor needs to know some details about the DSP code in order to interface to it properly). A mechanism through which new objects could be added dynamically would be useful particularly for the development of new algorithms. Once the algorithm developer is accustomed to using this system he will undoubtedly wish to test new algorithms immediately with other objects in the system and the time necessary to recode and recompile the system will be intolerable.
- Typing needs work. While audio streams have only one interpretation, control data streams have many. At the moment it would be meaningless to connect a slider to the keyboard input of a synthesizer. One school of thought requires explicit type transformers and helpful messages from objects which are offered the wrong type of jack.

The Katosizer hardware is in the process of redesign to increase the communication bandwidth of the interconnection network and the amount of memory on the processors. When the new hardware is finished, the existing software will be moved to it. The new hardware promises to support a much larger number of DSPs making the possibilities for the number of interconnected objects much more interesting.

Finally, many more objects need to be added to the system. Many of these are unrelated to actual synthesis or filtering, such as more sophisticated recorders and sequencers and editors, sampled sound editing tools, etc.

Acknowledgements

The Katosizer hardware and software is the result of a large amount of work by a small number of people. The authors would like to extend thanks to Steve Germann who has contributed much to improving the hardware design and to William Buxton who made helpful comments about the paper, and who has helped to keep music projects alive at CSRI despite the threats of the bourgeoisie.

References

- [Budg83] Budge, B. (1983). Pinball Construction Set (Computer Program). San Mateo, CA, Electronic Arts
- [Buxt82] Buxton, W., Patel, S., Reeves, W., & Baecker, R. (1982). Objed and the Design of Timbral Resources. *Computer Music Journal*, Vol 6, Summer, pp.32-44
- [Chow73] Chowning, John M. (1973). The Synthesis of Complex Audio Spectra by Means of Frequency Modulation. *Journal of the Audio Engineering Society*, Vol 21(7), 1973
- [Cord81] Cordy, J.R., & Holt, R.C. (1981). Specification of Concurrent Euclid. Report CSRG-133, Computer Systems Research Group, University of Toronto, 1981
- [Gres85] Gresham-Lancaster, Scot (1985). Macintosh as a Live Performance Tool. *Proceedings of the International Computer Music Conference, Vancouver.*
- [Haeb86] Haerberli, Paul (1986). A Data-Flow Manager for an Interactive Programming Environment. *Usenix Summer Conference Proceedings, Atlanta.*
- [Hoar74] Hoare, C.A.R. (1974). Monitors: An Operating System Structuring Concept. *Communications of the ACM*, Vol 17(10), Oct. 1974, pp.549-557
- [Karp83] Karplus, K. & Strong A. (1983). Digital Synthesis of Plucked String and Drum Timbres. *Computer Music Journal*, Vol. 7(2), 1983, pp.43-55
- [Kita85] Kitamura, J., Buxton, W., Snelgrove, M., & Smith, K.C. (1985). Music Synthesis by Simulation using a General-Purpose Signal Processing System. *Proceedings of the International Computer Music Conference, Vancouver.*
- [Kita86] Kitamura, John (1986). A General-Purpose Signal-Processor for Music Synthesis. MASC Thesis, Department of Electrical Engineering, University of Toronto.
- [Leff83] Leffler, S.J., Joy, W.N., & Fabry, R.S. (1983). 4.2BSD Networking Implementation Notes. Computer Systems Research Group, U.C. Berkeley, July 1983
- [Math69] Mathews, Max V. (1969). *The Technology of Computer Music*. The M.I.T. Press, Cambridge, 1969
- [Mime86] Mimetics Corp. (1986). Soundscape (Computer Program).
- [Pike84] Pike, R. (1984). The Blit: A Multiplexed Graphics Terminal. *AT&T Bell Laboratories Technical Journal*, Vol 63(8)
- [Prus85] Prusinkiewicz, Przemyslaw (1985). Graphics Interfaces for MIDI-Equipped Synthesizers. *Proceedings of*

the International Computer Music Conference, Vancouver.

[Road79] Roads, Curtis (1979). A Tutorial on Nonlinear Distortion or Waveshaping Synthesis. *Computer Music Journal Audio*, Vol. 3(2), 1979, pp.29-34

[Rose85] Rose, J.S., Loucks, W.M., & Vranesic, Z.G. (1985). FERMTOR: A Tunable Multiprocessor Architecture. *IEEE Micro*, Aug. 1985

[Shne83] Shneiderman, Ben (1983). Direct Manipulation: A Step Beyond Programming Languages. *IEEE Computer*, Aug. 1983, pp.57-69

[TI83] Texas Instruments (1983). *TMS-32010 User's Guide*. Dallas: Texas Instruments.

[Verc75] Vercoe, Barry (1975). Man-Computer Interaction in Creative Applications. M.I.T. Technical Report, Nov. 1975

