A Visual Programming Environment for Composing Interactive Performance Systems

Bernhard Wagner
MultiMedia Laboratory
University of Zurich
Inst.f.Informatik, Winterthurerstr. 190
CH-8057 Zurich, Switzerland
wagner@ifi.unizh.ch, http://www.ifi.unizh.ch/staff/bwagner.html

Keywords: Performance, Visual Composition, Object-Oriented Framework, Real-time, Media-Patcher

Abstract

Based on a multimedia application framework a visual programming environment was built, that allows non-programmers to compose objects offered in the core framework without having to learn a programming language. Although having similarities with Opcode/IRCAM Max, our visual programming environment has some significant differences: Its connections are bi-directional, other media than MIDI, such as audio, 3D, 3D-animations are supported.

1 Introduction

Based on the portable (UNIX, WinNT) object-oriented multimedia application framework MET++ [1,2] a visual programming environment was built, that allows non-programmers to compose objects offered in the core framework without having to learn a programming language such as Java or C++ [6]. Although having similarities with Opcode/IRCAM Max [3,4,5], our visual programming environment has some significant differences: Its connections are bi-directional, other media than MIDI, such as audio, 2D, 3D, 2D-, 3D-animations, and video are supported. Due to the bi-directionality of the connections and the support for various media, the visual programming environment can act as a Media Patcher. The Media Patcher can use static or time-dependent geometric aspects of 3D-objects (e.g. spatial coordinates, scaling, material, texture information) to control musical parameters. To allow for a more sophisticated mapping than e.g. some coordinate linearly to note pitch, intermediary analyses of media inputs can be programmed visually. Due to its MIDI- and TCP/IP-connectivity it can also be applied as a real-time system. It has been used as a real-time animation generator in a concert at the Conservatory of Zurich.

The musicians improvise on MIDI- as well as traditional instruments. The acoustic output is converted by a pitch-to-midi converter. After being filtered and analyzed, the MIDI data are applied as coefficients to parametric functions controlling several properties of an animated 3D-mesh: x-, y-, z-coordinates and r-, g-, b-components, and transparency, each controlled by an individual function. The controlling functions themselves can be exchanged at run-time. Due to a generic interpolation mechanism the exchanging of functions does not cause an abrupt change in the animation. If the visual programs become more complex the available screen space is quickly used up. To overcome this difficulty, a Tcl-Wrapper was built that allows to formulate more complex functions in terms of scripting.

2 The Object-Oriented Multimedia Application Framework MET++

The Object-Oriented Multimedia Application Framework MET++ has been presented at the ICMC 95 [2]. It is well-suited for rapid application development and prototyping purposes in the area of multimedia and graphical user interfaces with high semantic feedback. Like other frameworks, MET++ has reached a stadium where it is mainly used in a black-box manner. This means that it consists mainly of components that can be instantiated and reused without the need for subclassing. The term black-box is used because the internals of the components don't have to be dealt with, only their interfaces.

MET++ is not just a library or collection of isolated classes but a framework that pre-integrates the components and predefines their style of interaction. For example all time-dependent media can be edited regardless of their specific type in a special grouping editor provided by MET++. This editor allows the redundancy-free, hierarchical grouping of time dependent media. It therefore provides special grouping elements that

define the behaviour of the children it contains, e.g. a Synchro element assures that all its children start whenever it is the Synchro's turn to perform. A Sequence element groups its children so that they are always played in sequence. If a child is deleted from a Sequence all its successors are shifted to the left to close the gap. These grouping elements can themselves be grouped within grouping elements and so define complex temporal dependencies without ever having to calculate any temporal positions manually.

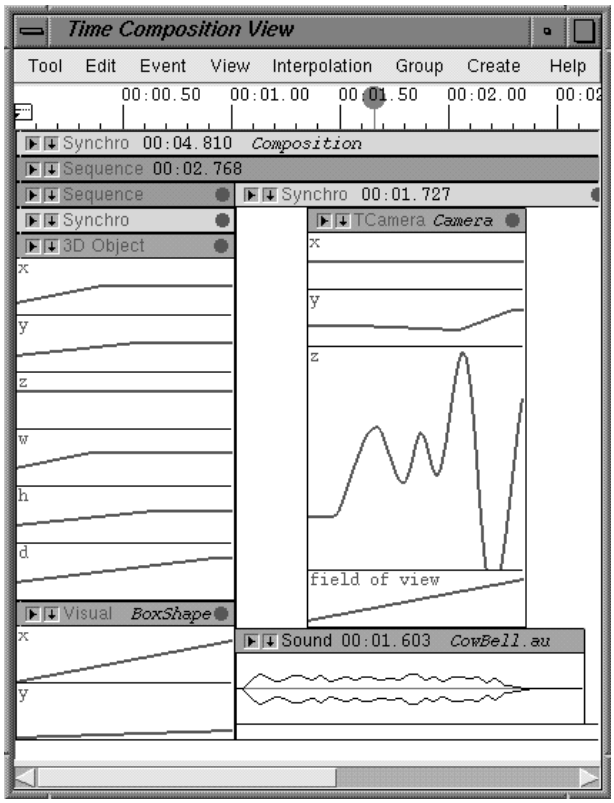

Fig1: Time Composition View

3 Visual Programming

The shift towards black-box reuse of framework components has been taken a step further in [6] by introducing a third control environment orthogonal to direct manipulation and temporal dependency. This environment provides a data-flow mechanism in which the framework's objects appear as IC-like components. The pins of these ICs represent the attributes of the corresponding object. For example a 3D-object has pins for the parameters: spatial x-, y-, z-coordinates, rotation, scaling, etc. but also pins for the definition of the object's color and transparency. Additionally, the visual programming environment provides special filter ICs that perform transformations of the values sent through them.

The wrapping of framework components is realized with an easy to use drag&drop user interface. For example a 3D-object can be dragged on top of the visual program and when released a wrapper is instantiated.

The bi-directionality of the filters implies that they perform a function in one direction and its *inverse* in the opposite direction. For example, there are special "shifters" which add a constant value *b* to the value *x* sent through them. When a value *y* is sent at their output-pin, the constant value *b* is *subtracted* and the result sent out the input pin. Several other mathematical functions (and their inverse) are available.

The concept of bi-direction is maintained also when filter objects are chained together. So when defining a complex filter function by cascading simple filters, the inverse of this function is defined "for free" and applied when a value is sent at the end of the chain.

If a function becomes too complex and cumbersome, it is possible to implement it by a scripting object. So far this object supports the Tcl scripting language [7], but it is planned to incorporate other popular scripting languages like Perl [8], JavaScript [9], Python [10], etc.

The advantage of getting the inverse function for free is lost in case where the scripting object is applied. The (scripted) inverse function has to be provided explicitly.

For media controlled in the visual programming environment the bi-directionality implies that while the properties can be set by sending values at the IC's ports it is also possible to control the object by other means (direct manipulation, temporal control) and the object's status is *reported* at the ICs pins.

This allows for the redundancy-free definition of e.g. animations by declaring relationships or constraints between objects in the visual program and only animating one keyplayer. The keyplayer's IC continuously reports the keyplayer's attributes and the visual program maintains the relationships between the keyplayer and its dependencies (see [6] for an example).

The visual programming environment and the temporal control can also be combined in other ways: A special wrapper for the temporal control element has been developed that allows to start, stop, and position the time pointer at a specific time by setting the corresponding ports in the visual programming environment. Using this technique it is possible to define "local" time

dependent media that are started only when specific conditions are met, e.g. a sound or midi file is only played when certain geometric conditions are met. This allows to automatically create the appropriate sounds for an animation of a collision. The animator of a scene does not have to bother about defining the starting of the sound at the right moment and always adjusting it when the animation changes, but only once define the appropriate condition.

4 "ParaScape": a specialised component

Though working with the visual programming environment consists mostly of composing objects by interconnecting them interactively it might be needed to develop specialized components in C++ mostly for efficiency reasons. The dichotomy between building components in a "system programming language" and gluing them together in a "scripting language" (or visual programming environment in our case) is discussed in [12].

The idea for the performance at the conservatory of Zurich was to let musicians improvise and generate a score out of the music in real-time. This score would then influence the musicians and so generate a cybernetic feedback.

By prototyping we tried several visual artifacts that could be used for the score and quite quickly found out that synthetic landscapes (meshes) have the strongest suggestive effect. Additionally, their multidimensionality accommodate to map the multidimensionality of the room of musical parameters. Three requests that the synthetic landscape should fulfill emerged:

- Shape and colors of the mesh should be controllable for the mesh as a whole to reduce the number of degrees of freedom.
- The changes of the mesh should happen smoothly
- The effect of the played music on the mesh should not be too obvious to the musicians. They should only feel that their playing *has* an effect on the animation but not *what* effect.

The first request could be complied with by introducing parametric functions.

The mesh can be understood as a rectangular set of vertices. Each vertex has the parameters: spatial x-, y-, z-coordinates, r-, g-, b-values defining the color, and transparency α. For a mesh with the dimensions U and V this results in 7 x U x V degrees of freedom. To reduce this great number and to get a better control over the mesh as a whole seven parametric functions were introduced, each controlling one aspect of all vertices of the mesh. These parametric functions indicate the value for the specific attribute depending on the *indices* u and v of a vertex within the rectangular set, e.g.

$$x = f(u,v) = v * \cos(u) \qquad (1)$$

The parametric functions applied to the mesh are specifically designed to be exchangeable at runtime. Since the visual program only supports the basic types like integer, float, and string (see [6]) the mesh contains a collection of about 30 functions. These are chosen by index, e.g. the function in (1) has the index 23. To fulfill the requirement of smooth changes of the mesh an interpolation scheme is used:

$$(1-\lambda) * f(u, v) + \lambda * g(u, v);\ \lambda \text{ in } [0,1] \qquad (2)$$

Where f is the first function and g is the second function. λ is incremented from 0 to 1 linearly. Here is an example set of parametric functions (the color functions are neglected):

$$x = v * \cos(u); \qquad (3)$$
$$y = v * \sin(u);$$
$$z = a * u;$$

and second set:

$$x = (b + a * \cos(u)) * \cos(v); \qquad (4)$$
$$y = (b + a * \cos(u)) * \sin(v);$$
$$z = a * \sin(u);$$

Fig. 2 shows the resulting shapes created by applying the sets (3), a helix and (4), a torus. The shape in between is one of the intermediate shapes.

Fig 2: interpolating between helix and torus

In this example all three parametric functions for x, y, and z have been exchanged at once. But in a real-time performance they would be exchanged individually and independently of each other.

Fig. 3 shows a sample setup of a visual program containing a mesh and receiving a subset of its parameters from two different MIDI-channels. The output of the first MIDI-channel is transformed by a script and by two simple filters.

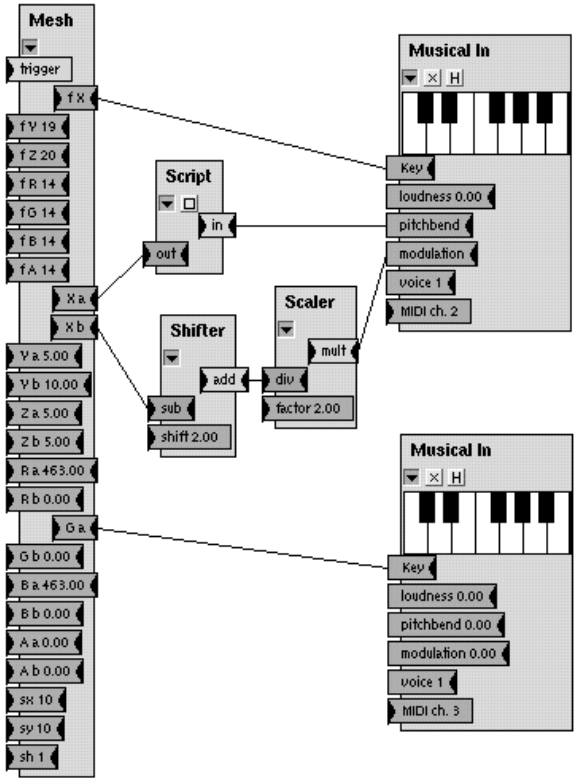

Fig. 3: a sample setup containing a mesh-object

The mesh equipped with the parametric functions and the ability to interpolate between them is called "ParaScape" (*para*metric land*scape*).

Outlook and Conclusion

A useful enhancement (possibly implemented by the time this paper is published) is the control not only over the color of a 3D-object but also over the texture applied. This could include the positioning of the texture relative to the 3D-object and the dynamic exchange of texture bitmaps, possibly including the interpolating between adjacent bitmaps.

The interpolation scheme (2) is implemented by incrementing λ in steps of 0.1. It might be interesting to allow the parameterization of this process and thus providing different interpolation modes.

We plan to experiment with the I-cube system [11] that provides a collection of sensors (pressure, acceleration, temperature, etc.). It would be easily integrated in our system due to its MIDI connectivity and could provide other means to control the ParaScape than MIDI instruments.

We believe that the visual programming environment can give artists a more intuitive interface for specification of real-time performances. It is also a useful prototyping environment before coding a component directly in C++.

References

[1] Philipp Ackermann: *Developing Object-Oriented Multimedia Software*; dpunkt-Verlag Heidelberg 1996.

[2] Philipp Ackermann: *Design and Implementation of an Object-Oriented Media Composition Framework*; Proceedings of the ICMC 1994.

[3] David Zicarelli, Miller Puckette: *Getting Started with MAX*, Opcode Systems Inc., Palo Alto, California, 1995.

[4] David Zicarelli, Miller Puckette: *MAX Reference*, Opcode Systems Inc., Palo Alto, California, 1995.

[5] *MAX Product Announcement* CMJ Vol. 15(1): pp. 81-82;1991.

[6] Bernhard Wagner et al.: *Black-Box Reuse Within Frameworks Based on Visual Programming*; 1st Component User's Conference, Munich 1996. Also available online (see my homepage)

[7] John K. Ousterhout: *Tcl and the Tk Toolkit*, Addison-Wesley, 1994.

[8] Larry Wall et al.: *Programming Perl, 2nd edition*, O' Reilly & Associates, 1996.

[9] John R. Vacca: JavaScript Development, AP Professional, 1997.

[10] Mark Lutz: *Programming Python*, O' Reilly & Associates, Inc. 1996.

[11] Alex Mulder: The I-Cube System: Moving Towards Sensor Technology for Artists, ISEA 1995 proceedings, Montreal. Also available online: http://fas.sfu.ca/people/ResearchStaff/amulder/personal/infusion/ISEA95.html

[12] John K. Ousterhout: *Scripting: Higher Level Programming for the 21st Century*, White Paper from 03/28/97 available at: http://www.sunlabs.com/people/john.ousterhout/scripting.html