FrameWorks 3D: Composition in the third dimension

Richard Polfreman
University of Southampton
University Road
Southampton, UK
r.polfreman@soton.ac.uk

Abstract

Music composition on computer is a challenging task, involving a range of data types to be managed within a single software tool. A composition typically comprises a complex arrangement of material, with many internal relationships between data in different locations - repetition, inversion, retrograde, reversal and more sophisticated transformations. The creation of such complex artefacts is labour intensive, and current systems typically place a significant cognitive burden on the composer in terms of maintaining a work as a coherent whole. FrameWorks 3D is an attempt to improve support for composition tasks within a Digital Audio Workstation (DAW) style environment via a novel three-dimensional (3D) user-interface. In addition to the standard paradigm of tracks, regions and tape recording analogy, FrameWorks displays hierarchical and transformational information in a single, fully navigable workspace. The implementation combines Java with Max/MSP to create a cross-platform, user-extensible package and will be used to assess the viability of such a tool and to develop the ideas further.

Keywords: Digital Audio Workstation, graphical user-interfaces, 3D graphics, Max/MSP, Java.

1. Introduction

FrameWorks 3D presents a new design for audio and MIDI sequencing user-interfaces. It extends traditional approaches with features to aid the mapping of compositional ideas onto a work, and facilitate rapid experimentation with musical ideas [1]. While such elements could be included in a (combination of) 2D display(s), FrameWorks adopts a 3D space in order to present complex structural information (hierarchical *and* relational) in addition to retaining the visibility of the existing notation; difficult to achieve effectively in a single 2D space. This allows detailed visual exploration of a work in a way which may give the composer new insights.

Once limited to games and scientific/bio-medical visualisation, 3D graphics are becoming pervasive, from Apple's Cover Flow [2] and Microsoft's 3D Flip [3], to Second Life [4]. As 3D representations proliferate, FrameWorks offers one approach to the adoption of this technology for music applications. While 3D has been used in some music systems [5][6], it has yet to be fully exploited in direct manipulation music composition tools.

2. FrameWorks: A Brief History

2.1 Origins

The concept was developed in task analysis research in the late 1990's focusing on music composition, and was first implemented in a 2D prototype in 2001 [7]. The primary concern is to allow rapid experimentation with material and structural ideas within the same interface. This relates to one of Green's Cognitive Dimensions of Notations [8], *viscosity*, described as the resistance to change of a notation. FrameWorks is a highly fluid design, where local changes to a work can propagate throughout allowing experimentation to incur a low time-cost.

2.2 Concept

Clips[1] which are containers for musical data of a particular type (MIDI, audio, OSC, etc) and which can be a) hierarchically arranged on *tracks* and b) connected together by one-to-one mono-directional *relations* expressing a connection between two *clips* (and their descendents). A combination of *clips* and *relations* forms a *framework*. *Clips* may be empty, and therefore the structure of a work can be developed prior to musical material; alternatively the structure can be built up from materials. Thus composers can work in both top-down and bottom-up modes (or some combination thereof) although a *framework* itself is a top-down structure.

The *relations* between *clips* are processes, which take the material in a source *clip*, transform it and place the new material in a target *clip*. These are dynamically maintained at all times, thus any changes to either *clips* or *relations* are immediately reflected throughout the *framework*. Typical *relations* include identity, transposition, time dilatation,

[1] Previously referred to as *components*, the name has been changed to avoid confusion with the programming concept of *component*.

reverse and filtering. *Relations* could be extended to many-to-one, where data from more than one source are combined to form the result (somewhat similar to side-chaining in studio effects).

Hierarchical and computational connections between music elements are not in themselves new, but have mainly been used in programming language based algorithmic composition tools. FrameWorks aim to bring these within the scope of standard DAW software.

2.3 Initial Prototype: FrameWorks 2D

An initial implementation, written in Java 1.1, lacked *clip* hierarchies, supported only MIDI data and was only a sketch of the intended system [9]. Figure 1 shows the *framework* view, where "hanging" from track timelines are several *clips*, connected by lines representing *relations* and whose colour indicates which *relation* is being used.

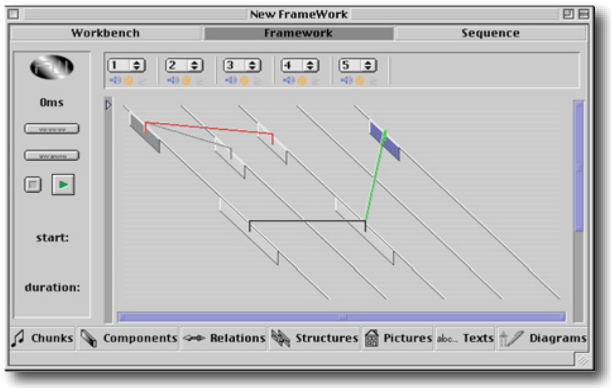


Figure 1. FrameWorks 2D: framework view.

A basic piano-roll display allowed *clip* editing, while *relation* editors specified transformation parameters. For example, *time relations* chain together an arbitrary number of source segments, with start and end points, playback speed and direction settings. In figure 2, the entire source is played once forwards and once again in reverse.

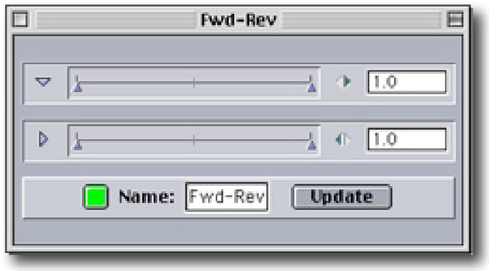


Figure 2. FrameWorks 2D: *time relation* editor

Informal feedback from composers was positive in terms of being able to create (and recreate) works in a fluid manner, particularly lending itself to process based music, but the interface was too limited in basic functionality for serious work and formal evaluations, while support for audio was indicated as essential.

3. FrameWorks 3D

FrameWorks 3D is a new implementation written in Java 5, using the Java 3D API [10] and Max/MSP as an audio engine [11]. Hierarchical arrangements of *clips* are now supported and audio data is used rather than MIDI. Java's MIDI and Audio API, Java Sound [12] has been criticised for a number of limitations in terms of latency and jitter [13], and while a number of solutions have been proposed (e.g [13]), an alternative strategy of using Max/MSP as an audio engine for Java has been adopted here.

3.1 Audio Engine Separation

FrameWorks 3D has been designed so that the audio/MIDI engine, wrapped in an *AMSEngine* class, can be re-implemented for various audio/MIDI API's. Earlier versions used an *AMSEngine* purely for data i/o, i.e. playback and recording of MIDI data, while the data itself was hosted and manipulated in the main FrameWorks code. While this limits the size of the *AMSEngine* and so simplifies switching to different implementations, such a design leads to frequent large data transfers between the FrameWorks model and the *AMSEngine*. While a relatively minor issue when both are written in Java and MIDI data is used, with an external engine and audio data, this may become a significant overhead. The new implementation expands the role of the engine to include managing the audio (and other) data and providing the processing for relations, thus minimising the data crossing the model/engine boundary. In the case of Max/MSP, this also allows us to use Max patches as relation implementations, leveraging a vast resource of audio processing objects, and permitting very rapid development of new relations, potentially by users themselves.

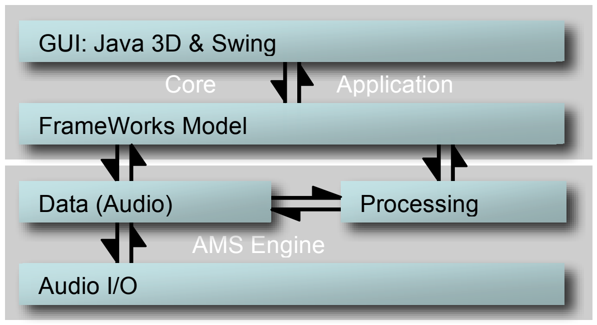


Figure 3. Internal structure of FrameWorks 3D

3.2 Max/MSP Integration

Several programming languages can be used to define new objects that can be used freely in Max patches: Max itself (i.e. abstractions), Javascript (js objects), Java (mxj objects) and C (native). In FrameWorks 3D we subvert this role, with our mxj~ class "taking over" the operation of Max from the user, providing a new application user interface and hiding as much of Max as possible. Max patches are loaded and scripted behind the scenes in order to carry out audio operations. The mxj~ object loads Java code and communicates with hidden Max patches to control audio i/o, the real-time clock, etc. The technical details and issues involved are described elsewhere [14].

3.3 FrameWorks 3D GUI

Figure 4 shows the main FrameWorks 3D window. Around the edge of the central 3D *framework* are various editing and navigation tools: a tree view of the framework structure, clip parameters (editable), and zoom controls.

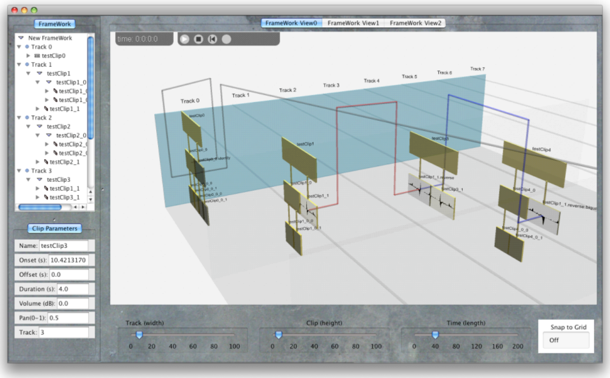
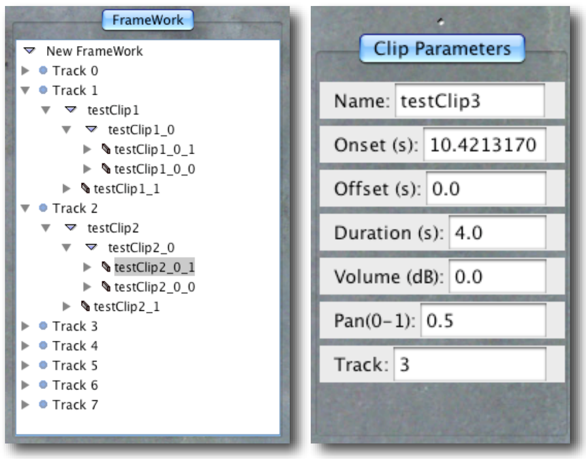


Figure 4. FrameWorks 3D main window.



FrameWork
- New FrameWork
  - Track 0
  - Track 1
    - testClip1
      - testClip1_0
        - testClip1_0_1
        - testClip1_0_0
      - testClip1_1
  - Track 2
    - testClip2
      - testClip2_0
        - testClip2_0_1
        - testClip2_0_0
      - testClip2_1
  - Track 3
  - Track 4
  - Track 5
  - Track 6
  - Track 7

Clip Parameters

Name:	testClip3
Onset (s):	10.4213170
Offset (s):	0.0
Duration (s):	4.0
Volume (dB):	0.0
Pan(0-1):	0.5
Track:	3

Figure 5. FrameWorks 3D: tree view and *clip* parameters.

Navigation is via the computer keyboard as in many 3D environments, which changes the virtual camera position and orientation and thus the user's viewpoint. The tabbed pane for the 3D view provides three independent views of the framework, to help keep track of the various *clips* and *relations* being used. The tree view provides an alternate representation of the framework, and selecting a clip in either, selects that clip in both views and displays its parameter settings where they can be edited (figure 5).

3.4 The Framework

In the 3D space, the x-axis represents time, the y-axis separates one track from another and the z-axis (vertical) is used group *clips* into hierarchical arrangements. Tracks are narrow strips extending along the time axis, from which rectangular *clips* are suspended. *Relations* are shown as pipes that connect a source *clip* to a destination *clip*. The current playback position is shown as a flat sheet in the y-z plane that moves along the x-axis. A small Head-Up Display (HUD) in the 3D space shows the clock and basic transport controls. Figure 6 shows the same *framework* viewed from different positions and orientations.

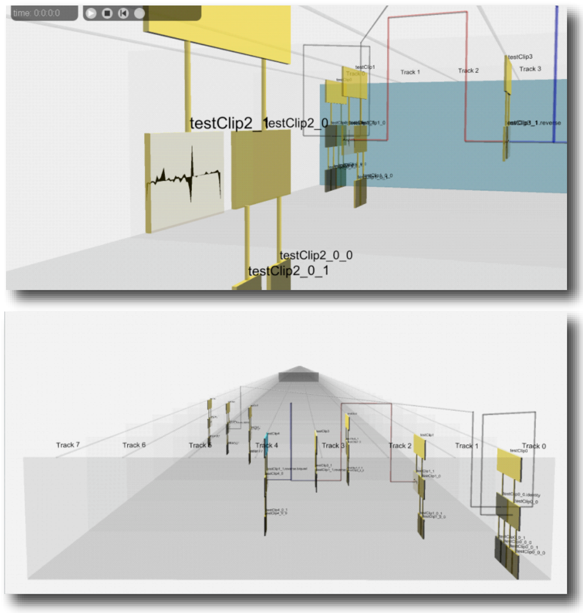


Figure 6. Two views of the same framework structure

3.5 Clips

Clips contain audio data and can be arranged in hierarchical groups (Figure 7 below). Only leaf clips hold audio directly, and these display an overview of the sound waveform when loaded. An audio clip is similar to an audio region in standard DAW software; the user can load a sound file and define a segment of that file to be the current data (by Command-dragging the ends of the clip, or by editing clip parameter values). Clips can be played back individually and the framework played as a whole.

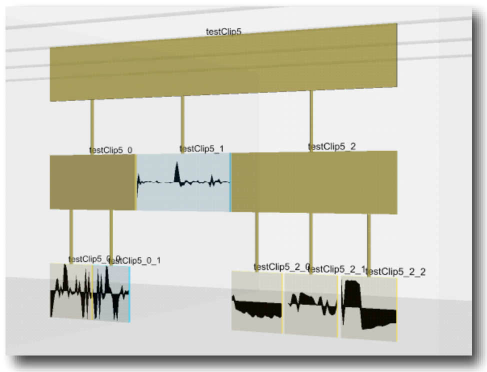
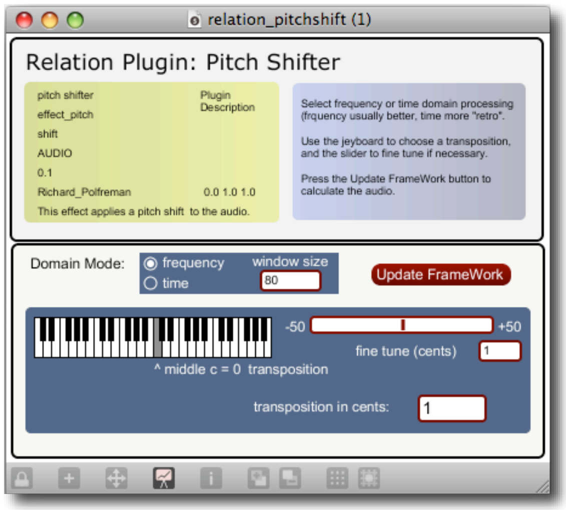


Figure 7. An example of hierarchical arrangement of clips.

3.6 Relations

Relations are implemented as plug-ins hosted by the audio/MIDI engine. These are currently in the form of specifically designed Max patches, which provide both the user-interface and the processing algorithm, much like commercial plug-in architectures such as Steinberg's VST. When FrameWorks 3D is launched, the relation plug-in files are scanned and made available to the user.



relation_pitchshift (1)

Relation Plugin: Pitch Shifter

pitch shifter
effect_pitch
shift
AUDIO
0.1
Richard_Polfreman
This effect applies a pitch shift to the audio.

Plugin Description

0.0 1.0 1.0

Select frequency or time domain processing (frquency usually better, time more "retro".

Use the jeyboard to choose a transposition, and the slider to fine tune if necessary.

Press the Update FrameWork button to calculate the audio.

Domain Mode: [✓ frequency] time

window size: 80

fine tune (cents): 1

transposition in cents: 1

Figure 8. User interface for the Pitch Shifter relation.

Figure 8 shows a Pitch Shifter relation editor. This applies a constant transposition to the source material using either a time or a frequency domain algorithm. Once the required settings are set, the update framework button applies the new settings to the audio, which will in turn update all dependent audio throughout the framework.

A number of relations have been developed so far, including identity, reverse, filter (biquad), brassage and pitch shifter.

4. Further Work

Current development is focussed on refining the interaction between Java code and Max/MSP, designing additional *relations*, and including further user-interface features in order to aid user testing. The 3D interface is deliberately sparse at this stage in order to focus on user-assessment of the basic concept and gain user input on how additional interface elements might be developed.

As the tool develops we expect to reinstate MIDI data, add automation of *clip* parameters, add track parameters and effects, to bring the whole system closer to a DAW style environment.

In addition we are looking for further opportunities to exploit 3D user-interface elements within the environment, such as in relation editors.

5. Conclusions

FrameWorks 3D represents a novel approach to sequencing tasks by extension of existing DAW metaphors into a 3D space which features both hierarchical arrangements of content and dynamically maintained relationships between elements within the structure.

An initial 2D prototype showed some promise, and this has now been significantly enhanced with a true 3D implementation. While it is still early in the overall development of the system, we are aiming to disseminate the ideas embodied in the software and gain feedback from composers. A useable demonstrator system will be freely available to users in late 2009.

6. Acknowledgments

FrameWorks 3D development was in part funded by the i10 (www.i10.org.uk) by way of an Enterprise Fellowship.

References

[1] R. Polfreman, "A task analysis of music composition and its application to the development of Modalyser," *Organised Sound*, vol. 4, pp 31-43, 1999.

[2] http://www.apple.com/pro/tips/coverflow.html, last accessed 20/01/2009.

[3] http://www.microsoft.com/windows/products/windowsvista/features/details/flip3D.mspx, last accessed 18/01/2009.

[4] http://secondlife.com/whatis, last accessed 18/01/2009.

[5] T. Kunze and H. Taube, "SEE—A Structured Event Editor: Visualizing Compositional Data in Common Music", in *Proceedings of the 1996 ICMC*, 1996, pp. 63-66.

[6] N. Castagne, and C. Cadoz, "L'environnement GENESIS : créer avec les modèles physiques masse-interaction", in *Journées d'Informatique Musicale, 9e édition*, 2002, pp 71-82.

[7] R. Polfreman, "Supporting Creative Composition: the FrameWorks Approach," in *Les Actes des 8e Journées d Informatique Musicale*, 2001, pp. 99-111.

[8] T. R. G. Green, "Cognitive dimensions of notations", in *People and Computers V*, A Sutcliffe and L Macaulay, Eds. Cambridge : CUP, 1989 pp. 443-460.

[9] R. Polfreman, "FrameWorks - A Structural Composition Tool," in *Music without walls? Music without instruments? Proceedings of the International Conference*, 2001, CD-ROM.

[10] "Java3D API Tutorial", http://java.sun.com/developer/onlineTraining/java3d/index.html

[11] M. Puckette, "Max at 17", *Computer Music Journal*. Vol 26, no. 4, pp 31-43, 2002.

[12] "JavaSound API Programmer's Guide", http://java.sun.com/javase/6/docs/technotes/guides/sound/index.html

[13] N. Juillerat, S. M. Arisona, S. Schubiger-Banz. "Real-Time, Low Latency Audio Processing in Java," in *Proceedings of the International Computer Music Conference, ICMC 2007*.

[14] R. Polfreman, "Role-Reversal: Max/MSP as an Audio Engine for Java," in preparation, submitted to ICMC 2009.