## GEMNOTES: A REALTIME MUSIC NOTATION SYSTEM FOR PURE DATA

Dr Edward Kelly
11 Spenser Road
London SE24 ONS
morph\_2016@yahoo.co.uk

### **ABSTRACT**

This paper describes a system of realtime music notation using TrueType fonts (TTFs), running in the Graphics Environment for Multimedia (GEM) in the Pure Data computer music environment (PD). The system makes use of dynamic object creation in PD to create subpatches linked to a stave object, so that custom made abstractions for notes, rests, tempo marks, barlines and time signatures are added to the patch on-the-fly to create a visual score.

### INTRODUCTION

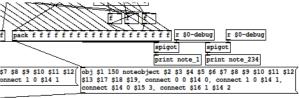
The origins of this system derive from attempts in previous decades by the author to contrive an effective system for the creation of automated systems for musical score presentation. The first of these was a continuous sheet of acetate with a graphic vocal score 7 metres long printed on it, to be scrolled by hand ocross the screen of an overhead projector (Te Deum, 1997). Five further scores were created in a form of proportional polyrhythmic notation devised by the author (see Kelly, 2005) and a prototype system for displaying scrolling scores was created in 2001 at the University of East Anglia using Max. However, the use of bitmaps and the refresh rate of the graphics resulted in a jerky display that was hard to read. These scores originate as fixed scores designed on paper, but the idea of a live notation system based on fonts was considered to offer a degree of flexibility with material that is often found in electronic music that could be applied to instrumental music, such as generative scores, aleatoric structure and feedback between the performer (or ensemble) and computer. The preliminary result of this enquiry is the Gemnotes system, which takes as its input a simple score language to render a score from visual objects in live performance. A further consideration is that this system may be used with the Pd-extended distribution without modification, so that instrumental musicians do not need to understand complex computing issues such as compilation of source code in order to use a score patch.

## **OPEN FONT-BASED NOTATION OBJECTS**

Although the search for a usable font began in 2003, the publication of the Open Font License specification (SIL, 2005) created an open-source framework for TTFs, and

the MusiQwik font (Allgeyer, 2007) was adapted for use with PD by the author soon afterwards. Certain ASCII codes are forbidden in PD ({, }, \) and the font has been adapted so that essential glyphs are moved from these locations, and augmented with tails for hemidemi- and demi-semiquavers. These are rendered using the [text3d] object in GEM.

### DYNAMIC NOTATION OBJECT CREATION

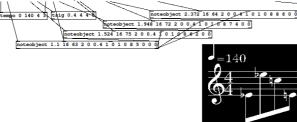


-- to [s pd-\$0-voice]

**Figure 1:** The dynamic object creation messages for the creation of notes, and connection of notes together into beamed groups. Objects 0,1 and 2 are the inlets and outlets of the [\$0-voice] subwindow (see below).

Objects for rendering notes and other notation elements are instances of PD abstractions created within a subpatch of the [makevoice] abstraction. This object sends object-creation commands (figure 1) to a [\$0voice] subwindow within a [\$0-canvas] subwindow, whereby notation abstractions may be linked to a stave abstraction(s) outside [makevoice], so that their positions may be organised relative to the stave. The reason for the nested subwindows is so that connections to the outside world may be destroyed and remade. [makevoice] also connects groups of notes together into beamed groups, and the beam angles and positions are worked out by linking certain outlets and inlets of the [noteobject] abstraction together (figure 2), whereby trigonometry is automatically used to fix these values. The system is clocked so that additional notes added to the group onthe-fly change the beam orientations and positions, although future versions will attempt to eliminate this clocking to reduce CPU load.

Objects are automatically connected to the stave object by [makevoice], and their positions are worked out relative to the stave: the vertical position of a notehead is worked out from a MIDI note number, and the type of accidental by a creation argument for each [noteobject]. Each stave therefore has a system of [noteobject], [tsig] (time signature), [rest] and [tempo] abstractions connected to it, which are the rendered score elements.



**Figure 2:** Abstractions are automatically connected together (top-left) to create a beamed group (bottom-right)

# A Hard-Coded Musical Structure Object

[makevoice] contains the **gemnotes\_counter** object - an external written in C that handles the creation and linking together of notation elements. With a system as complex as this it became clear that a single object was needed to handle the beamgroups, bars and the creation index of each object. The creation of beamed groups of notes on the fly, linking all objects to a stave abstraction, and ensuring that elements are horizontally spaced correctly are handled by this object. It is essentially a set of nested counters, and is presently capable of handling polyrhythmic note durations and user-defined groupings of notes within a bar, the automatic placement of barlines, tempo marks and time signatures.

Since this object handles the placing and index of each notation object, the introduction of new notation objects requires a certain amount of rewriting of the code, although graphical objects can be added to the GEM window outside of the [makevoice] structure. Imminent additions include dynamic marks, and articulation.

### SCORE LANGUAGE AND TRANSLATION

### **Minimal Text Score Notation**

The Gemnotes system translates a text-based score language (figure 3) into rendered notation elements on a stave. One of the design goals of the Gemnotes system was to keep the text-based score language as simple as possible. There are a number of commlands that set the bar (and time signature if different from the last) note and tempo elements. Dynamics are the next feature to be implemented graphically, but the score language already contains MIDI velocities that may be interpreted as dynamics and articulation elements. The second reason for creating a minimal score language is that it make the process of real-time linking with pitch detectors etc, or translation of MIDI files much easier. To this end, a further set of objects has been created – polyquant that quantizes inter-onset times and durations with optional polyrhythmic complexity, and gemnotes barcount which generates gemnotes scores from MIDI input. Note commands are formatted as a list: ["note" position duration pitch flat/sharp(0/1) velocity(. Work is underway to link together these two processes, so that music notation can be rendered directly from live input.

```
tempo 175 16 2 9 11 7;
bar 5 4 4 16;
note 0 58 2 0 40;
note + 70 2 0 46;
note + 72 2 0 51;
note + 65 2 0 58;
note + 74 2 0 63:
note + 64 2 0 70;
note . 76 2 0 70;
rest 8 0.5;
bar 4 4 4 16;
note 0 80 2 0 77;
note + 68 2 0 81;
note + 65 2 0 85;
note . 76 2 0 85;
rest 4 0.5;
note + 64 4 0 100;
end:
```

**Figure 3:** A section of a Gemnotes score. The bar commands also set the group lengths and note values for beam groups. A "+" means "next note" whereas a "." means it is a chord with the previous note.



Figure 4: A complete line of notation.

### CONCLUSION: NOTES ABOUT A SCORE

This system provides a basis for live score rendering using PD and Gem. A benefit of the system in question is that the material presented to the performer as a score can be re-interpreted in the electronic processing of the performer's audio, and may be reorganised indeterminately during performance. Scores may be created with a number of outcomes, or material may be arranged in a non-sequential manner using multiple staves for process music, or for stimulating improvisation, and the same pitches and rhythms used to set effect or playback parameters for a concurrent audio patch. Future aims are to integrate more notation elements, and to make the system less CPU-intensive. Also, live re-interpretation of material should be possible by directly rendering the output from a [fiddle~]-based pitch tracker for monophonic instruments.

### REFERENCES

Kelly, E. Time in Music, Strategies for Engagement,
 PhD Thesis, 2005, University of East Anglia,
 Norwich, UK.
 SIL International. Open Font License (OFL),

http://www.sil.org/ 2005

Allgeyer, R. *MusiQwik*, <a href="http://cg.scs.carleton.ca/~luc/allgeyer/allgeyer.html">http://cg.scs.carleton.ca/~luc/allgeyer/allgeyer.html</a>, 2007