Gerber, A. and Brown, A. R. (2006). Visualising Music with Impromptu. In S. Wilkie and C. Haines (eds.) *Medi(t)ations: Australasian Computer Music Conference.* Adelaide: ACMA, pp. 38-43.

Anna Gerber, Andrew R. Brown
Music & Sound
QUT
Kelvin Grove, QLD, 4059
Australia
{a.gerber, a.brown}@qut.edu.au

Visualising Music with Impromptu

Abstract

This paper discusses our experiments with a method of creating visual representations of music using a graphical library for Impromptu that emulates and builds on Logo's turtle graphics. We explore the potential and limitations of this library for visualising music, and describe some ways in which this simple system can be utilised to assist the musician by revealing musical structure are demonstrated.

Introduction

> "Rhythm, meter, frequency, tonality and intensity are the periodic parameters of music. There is a similar group of parameters that set forth a picture domain as valid and fertile as the counterpoised domain of sound. This visual domain is defined by parameters which are also periodic. 'Computational periodics' then is a new term which is needed to identify and distinguish this multidimensional art of eye and ear that resides exclusively within computer technology." (Witney 1980:210)

Music occupies time and space, and visualisations can assist the composer by revealing the organisation of music by mapping sonic space to Cartesian space. In this paper we share the results of some explorations into visualising music using simple graphic techniques, with a view to building up a toolkit for assisting with the creation and analysis of music. Inspired by pioneers such as John Witney we pursue visual correlations to musical patterns and form but, unlike Witney, we do so not purely to the ends of visual aesthetics but to provide visualisation tools to assist composers. In this regard our intentions are similar to those of Seymour Papert who used the Logo computer graphics language to assist children to understand mathematical principles, in particular geometry, through animated visualisations.

Background

There is a rich history of music visualisations for the purpose of composing, dating back at least to early notation systems using neumes. In many ways these methods of visually notating music have become more specific, with waveforms on oscilloscopes and computer screens being perhaps the most directly detailed, to the very generalised use of arcs and shapes as representations of musical gestures and sound objects. There is not the space here to cover this territory, the interested reader is referred to Miell et al. (2005), however, we will spend some time demarcating the areas of visualisations that do concern this research and, in particular, some of the history of Logo turtle graphic visualisations.

Music Visualisation

There are a number of uses for visualisations in music, so to avoid confusion it is important to be clear about what we are not concerned with in this paper. There is a considerable amount of research involving the visualization of timbre, from oscilloscopes to spectrograms. While we might be inspired by some of the techniques used in this work, our focus in on compositional structure typically understood as the organisation of musical note or sonic objects over time. Due to the exploratory nature of this project, we have focused solely on note-based music, however we anticipate that these visualisation techniques could be extended to apply to other compositional structures.

There is also a large interest in graphical animations that are driven by music, as found in most mp3 players and the like. In general the correlation between music and sound in these systems is too abstract to be of analytical use to the composer, which is the focus of the visualization strategies of interest to us in this paper. Finally, we are also little interested in the field of synaesthesia, which examines links between the sensation of sound and colour. Whilst, we are concerned with mapping musical attributes to visual cues, which may include colour, we make no claim that these mappings have any instinsic perceptual or cognitive significance.

Logo and Turtle Graphics for Music

We chose to focus on the Logo language's turtle graphics as a means for music visualisation because this language had been designed with ease of use for non-computer-programmers in mind, and because of previous successes in using turtle graphics for exploration of spatio-temporal concepts within the domain of mathematics. A significant source of inspiration for our investigation was the work of Seymour Papert (1980) and others in developing Logo to assist with the understanding of geometry through visualisation (and embodiment). Our goal was to explore whether an implementation of functions provided by turtle graphics within a musical environment could provide useful abstractions to musicians wishing to map musical concepts to spatial and temporal concepts used in visualisations.

To this end we developed a Logo implementation in Impromptu (Sorensen 2005) and some results of our experiments in this environment are described below. Whilst easy to access, we soon found the simple path tracing processes of Logo somewhat restrictive and added other drawing processes to our visualisation toolkit, whilst trying to maintain the features of elements of simplicity and accessibility.

It is useful to consider some of the other attempts to utilise the ideas of Logo for music creation, in particular the work of Jeanne Bamberger (1974), Desain and Honing (1988), Mike Guzdial (1991) and Gregory Gargarian (1996). Many of these attempts focused on providing musical building blocks for the assembling of compositions, however our approach is generally to do the inverse and to draw images with the graphics library that reveal structural attributes of music and thus be an aid to the musician when creating music.

Turtle Graphics for Impromptu

Impromptu is an interactive programming environment created by Andrew Sorensen (2005), utilising the Scheme programming language, and has been designed for live programming performance of music and graphics.

We created a library of turtle graphics primitives by building on top of Impromptu's core graphics (Quartz) functions. The primitives that we implemented included:

- `penup` : start drawing a trail behind the turtle
- `pendown` : stop drawing a trail behind the turtle
- `setbg` : set background colour
- `setpc` : set pen colour
- `forward` : move forward by a number of steps
- `back` : move backwards by a number of steps
- `left` : turn left by a number of degrees
- `right` : turn right by a number of degrees
- `setpos` : move to a specific co-ordinate location
- `seth` : set turtle direction using absolute angle
- `home` : return to origin
- `clear` : clear screen and return to origin

Because of differences between Logo and Scheme, the code for drawing graphics using the turtle graphics library for Impromptu or with Logo turtle graphics is not identical. Mostly the variation is due to syntax and differences between the semantics of iteration and termination in Scheme and Logo. For example, use of infinite recursion and a control key sequence to terminate the recursion is possible in Logo, but this would not be considered to be well-formed in Scheme. However, the basic turtle functions such as moving, changing heading and position, and pen operations have the same parameters and demonstrate the same graphical results.

To illustrate these differences, Figure 1 shows an example of Apple Logo (Ableson, 1982) syntax for defining a procedure called triangle, which takes a single parameter called size, while Figure 2 shows the equivalent code using the turtle graphics library for Impromptu. In Figure 2, `(define triangle (lambda (size) … ))` can be considered to be equivalent to the first and third lines of Figure 1, while `(dotimes (i 3) (begin` can be considered to be equivalent to `REPEAT 3`.

```
TO TRIANGLE :SIZE
REPEAT 3 [FORWARD :SIZE LEFT 120]
END
```

Figure 1: Apple Logo procedure to draw a triangle

```
(define triangle
   (lambda (size)
      (dotimes (i 3) (begin
          (forward size)
          (left 120))))
```

Figure 2: Impromptu code for drawing a triangle

Examples

Using this set of turtle graphics primitives, we experimented with creating simple visualisations that used paths, shape, colour, line thickness, and transparency to focus on particular musical attributes.

Using the library functions, we constructed a path to represent the melodic contour of a monophonic part. We implemented a `draw-note` function to create a path representing the melodic contour. The code for this function is shown in Figure 3.

```
(define cx 0)
(define cy 0)
(define offset 50)
(define draw-note
   (lambda (time p d r)
     (set! cx (+ cx (/ d 50)))
     (set! cy (- p offset))
     (setpos cx cy)))
```

Figure 3: Code to draw melodic contours

Whenever a note was played, a section was added to the path by calling `draw-note`, as shown in Figure 4. The `play-note` function includes the pitch, dynamic (volume) and rhythm (duration) as parameters. The `draw-note` function was called with the same parameters. In this example, the pitch of the note determined vertical distance between path segments, while the duration of the note determined the horizontal distance.

```
(play-note time inst p d r)
(draw-note time p d r)
```

Figure 4: Code for calling the draw-note function

The results of running this visualisation on random generated notes can be seen in Figure 5, while the same visualisation code run with the first part of Bach's Little Fugue in G minor is shown in Figure 6.

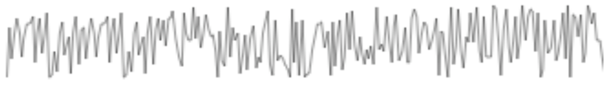

Figure 5: Melodic contour (random notes)

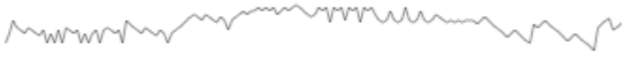

Figure 6: Melodic contour (Bach Fugue part)

By modifying the draw-note function, we experimented with different visualisations of each note based on pitch, dynamics and duration. The visualisation code shown in Figure 7 is based on note pitch and dynamics, and result of running the code on random note input is shown in Figure 8.

```
   (define range-size 60) ; pitch range
   (define (draw-note time p d r)
       (penup)
       (seth (* range-size p))
       (forward 20)
       (pendown)
       (triangle (* 2 d))
       (penup)
       (home)))
```

Figure 7: Code for visualising dynamics

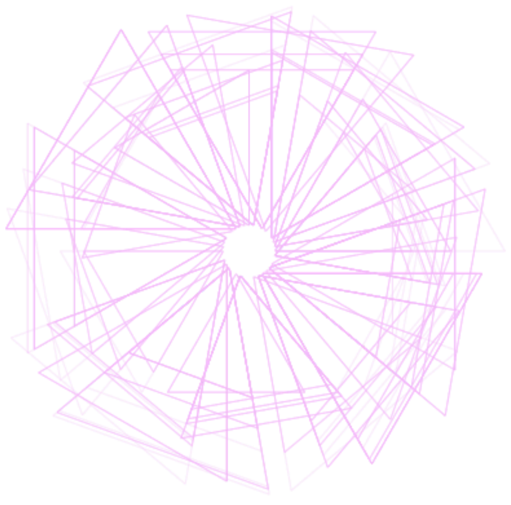

Figure 8: Dynamics visualisation

In this example, the pitch was represented by a position in Cartesian space, determined by segmenting the 360 degrees in a circle around the turtle origin so that each note in the pitch range of the randomly generated note content was represented by a particular angle within this circle. For each note, a shape was drawn in the position corresponding to the pitch, with the size of the shape representing the note's loudness; the smaller the shape, the louder the note.

The visualisations don't have to be based only on absolute values such as a note's pitch, duration or dynamics. In the visualisation code shown in Figure 9, the intervallic relationship between notes is used instead.

```
(define *pcrgb* '(0.5 0.0 0.5))
(define prev lower)
(define (draw-note time p d r)
    (left d)
    (setpc (append *pcrgb*
            (make-list 1 (/ 25.0 d)))
    (forward (* 2 (abs (- p prev))))
    (set! prev p))))
```

Figure 9: Code for drawing intervals & dynamics patterns.

Dynamics are represented by varying the alpha transparency of the colour used to draw the lines, and by the relative angle of each path segment from the previous one. The pitch intervals correspond to the lengths of the line segments. Figures 10 – 12 show the results of running this visualisation over different types of music. Figure 10 is based on randomly generated notes, Figure 11 is based on a looped sequence of generated notes, and Figure 12 is based on Bach's Little Fugue in G minor.

For this example, we stepped outside of the original behaviour of Logo turtle graphics to use features of Impromptu that allow custom colours (based on red, green, blue and alpha transparency values) to be defined in order to use colour to help with identifying patterns in the visualisation. In the first two examples, the colour is changed over time to indicate the temporal progression of the visualisation. In the looped example, a new colour is generated each time through the loop. For the Fugue, each colour represents a different voice of the Fugue.

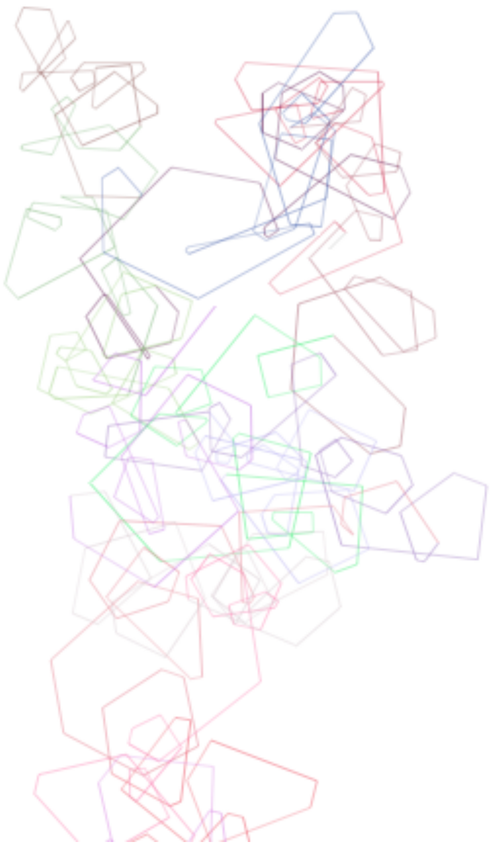

Figure 10: Random notes interval path

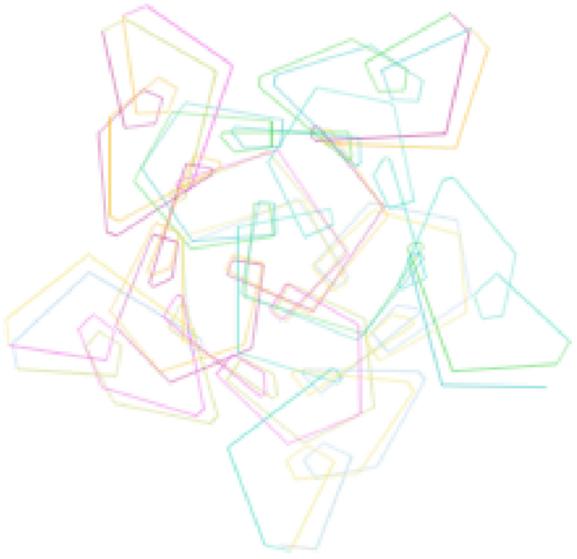

Figure 11: Looped pattern interval path

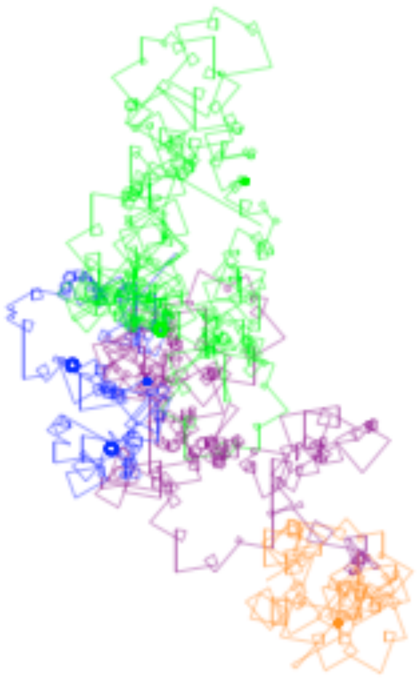

Figure 12: Bach Fugue interval path

In Figures 11 and 12, patterns in the visualisation can be identified. In Figure 11, each iteration through the loop results in the same shape being drawn. A regular, circular visualisation is produced because these shapes are being drawn in sequence, offset by the same angle.

In the Fugue, similar patterns are easy to spot visually, indicating where imitation occurs in different voices of the fugue. Because the interval sequence and dynamics are almost identical, the visual representation is also almost the same for the Fugue subject in the tonic key and for answers in the dominant key. Other features that are easily identified in this visualisation are periods of trill, identified by tight, dark circles.

Although all of the visualisations shown so far were based around single notes, visualisation functions are easily created at different levels of granularity or to visualise different aspects of a piece by modifying the parameters to the visualisation function. Figure 13 shows code for visualising chords.

```
(define draw-chord
   (lambda (time p p2 p3 p4 p5  d r)
      (penup)
      (set! cx (+ cx 10))
      (setpos cx cy)
      (pendown)
      (draw-shape p p2 p3 p4 p5)))
(define draw-shape
   (lambda (p p2 p3 p4 p5)
    (setpc (make-list 4 (lambda (i)
       (random))))
    (seth 0)
    (forward 15)
    (left (* 10 (- p p2)))
    (forward 15)
    (left (* 10 (- p2 p3)))
    (forward 15)
    (left (* 10 (- p3 p4)))
    (forward 15)
    (left (* 10 (- p4 p5)))
    (setpos cx cy)))
```

Figure 13: Triad visualisation code

The `draw-chord` visualisation function is called when the notes of the chord are played. The `draw-shape` function uses the pitch values of the notes of the chord to draw polygons with the number of sides equal to the number of notes in the chord. The angles are based on the distances between the pitches in the chord. Each chord is drawn in a random colour. The results of applying the chord visualisation are shown in Figure 14, for randomly generated triads.

Figure 14: Triad visualisation

Applying the draw-chord function to a keyboard ‘comping performance of John Coltrane’s Giant Steps produces the output shown in figure 15. The irregularities of the dense jazz voicings produced a correspondingly diverse visual appearance.

Figure 15: Giant Steps chord sequence

Discussion

It is possible to create many different visualisations using the library, however over the course of our experimentation, most of the visualisations that we created fell into the following categories:

- Graph-like paths such as the melodic contour example from Figures 3 - 6, where the x and y position of each path segment is based on musical attributes.

- Relative paths, where the length and angle of the next path segment is determined by musical attributes. The interval examples shown in Figures 9 - 12 are examples of this kind of visualisation. In the presence of repeated patterns or loops, the result of this type of visualisation can be visually similar to the patterns drawn by a spirograph toy.

- Stamp-based visualisations, where a template function (the "stamp") is parameterised with the attribute being visualised to determine the position, size, colour and even the shape of the stamp. The dynamics example shown in Figures 7 and 8 and the chord example in Figures 13 and 14 are examples of this style of visualisation.

Because turtle graphics is a predominately path-based graphical paradigm, the library that we implemented is best suited to music visualisations based on continuous paths. Multiple paths can be constructed to represent polyphony or different views or aspects of a single part. Both the graph-like and relative path styles of visualisations are based on paths, with the difference being that the graph-like visualisations are based on absolute positioning to construct the path segments, while the spirograph-like visualisations use relative angles and distances.

Our explorations revealed that our library has potential in creating the kinds of visualisation described above. However, it was also clear that there are many other mappings from musical compositional structures to cartesian space that are not easily implemented using the path-tracing approach afforded by Logo's turtle graphics. For other types of visualisation, such as those that make use of many discrete visual objects, using our turtle graphics library was a hindrance as it was more intuitive to implement such visualisations in the underlying graphics functions provided by Impromptu.

Remaining strictly within the bounds of existing Logo turtle graphics functionality limits the possible visualisations that can be created with the library. Hence, we plan to build future libraries directly on top of Impromptu's core graphics functions rather than on top of turtle graphics.

Drawing isolated shapes and separate paths at absolute positions such as those used in the stamp-based style visualisations can be cumbersome using the turtle graphics library. Turtle graphics requires a `penup`, `setpos`, and `pendown` prior to drawing a disconnected shape or path. For drawing basic shapes, the code may be clearer, and would be more efficient if the functions were based directly on Impromptu's native functions, rather than implemented using turtle graphics.

Even for path-based visualisation, Impromptu's built-in core graphics functions can be used to augment the visualisation with functionality outside of that provided by traditional turtle graphics such as changing the thickness of the path, using alpha transparency and custom colours beyond the traditional pallet of seven. In the examples presented in this paper, we have already moved beyond the capabilities of Logo turtle graphics in terms of defining custom colours.

The figures in this paper provide static pictures of the final result of running the visualisation functions over a period of time. We've shown the entire visualisation, however patterns would be clearer to the observer while watching the visualisation unfold over time when older graphics are faded out or erased over time. This fading can be achieved by periodically washing the canvas using a semi-transparent cleared image in the same colour as the background. Impromptu includes `create-image`, `clear-image` and `draw-image` functions that can be used to add to wash functionality to our library. It is also possible to set the animation rate of drawing using Impromptu's facility to schedule functions using the `callback` command. This allows for temporal structures to unfold over time at a speed specified by the code author, or to be synchronised with music playback.

Visualisations often creep off the screen as the current position of the path or turtle progresses over time. This could be mitigated by writing smarter visualisations, however a better approach would be to provide functions in the library that encode common strategies for dealing with this problem, for example, when the turtle hits the bounds, to bounce it back into the canvas like a ball bouncing off a wall or to shift it back to the centre of the drawing canvas.

Currently, we are extending the library by adding additional abstractions over the primitive visualisation functions provided by Impromptu and the turtle graphics library, including functions for simple shapes such as triangle, circle, rectangle etc and wrappers for other core graphics functions using relative distances, angles and points, in the same way that turtle graphics uses relative distances and angles for path construction.

Future directions for the library include adding more music-specific functions allowing composers to parameterise the basic shape functions using musical attributes such as pitch, duration, dynamics, timbre etc and to parameterise relative visualisation functions based on musical relationships.

Conclusions

We have shown that even with a simple set of graphical elements (line, angle, triangle and colour) some useful visualisations of musical structure can be achieved. We have also limited ourselves to two-dimensional representations at this stage. With the additional graphic elements and the utilisation of three-dimensional space we believe that some quite complex yet informative visualisations will be possible. The interactive philosophy of Logo programming and the real-time abilities of the Impromptu environment allow this approach to visualisation to become a fluid addition to the computer musician's creative toolkit.

While our work reported here has been deliberately limited to turtle graphic functions in order to test the possibilities of a limited visual repertoire, our future research will extend the library by removing limitations such as the requirement for a continuous drawing path and maintenance of a single "turtle" position. We are also interested to explore issues of interaction design and usability to see how musicians may incorporate visualisations in their music making activities.

References

Ableson, H. (1982) *Apple Logo.* NH: BYTE/McGraw-Hill.

Bamberger, J. (1974) *Progress report: Logo music project*. Technical report, A.I. Laboratory, Massachusetts Institute of Technology.

Desain, P. and Honing, H. 1988. LOCO: A composition microworld in Logo. *Computer Music Journal* 12(3): 30-42.

Gargarian, G. 1996. "The Art of Design". In K. B. Kafai and M. Resnick (Eds.), *Constructivism in practice: Designing, thinking, and learning in a digital world.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Guzdial, M. 1991. *Teaching Programming with Music: An Approach to Teaching Young Students About Logo*. MA: Logo Foundation. Available from from http://el.media.mit.edu/Logo-foundation/pubs/papers/teaching_progr.html. Last accessed 12 April 2006.

Miell, D., Raymond, M. and Hargraves, D. (2005). *Musical Communication*. Oxford: Oxford University Press.

Papert, S. (1980). *Mindstorms: Children, Computers, and Powerful Ideas*. New York: Basic Books.

Sorensen, A. (2005) *Impromptu: An interactive programming environment for composition and performance,* paper presented to the Australasian Computer Music Conference 2005, Brisbane: ACMA, pp. 149-153.

Sorensen, A. (2006) Impromptu. Available from http://impromptu.moso.com.au/. Last accessed 12 April 2006.

Whitney, J. (1980). *Digital Harmony: On the complementarity of music and visual art.* Peterborough, NH: Byte Books.