A Symbiosis of Animation and Music

ROBERT E. PRINGLE BRIAN J. ROSS
Brock University
Department of Computer Science
St. Catharines, Ontario, Canada L2S 3A1
{*rp94bg,bross*}*@sandcastle.cosc.brocku.ca*

December 12, 1995

Abstract

The use of music as a means to automate the sculpting and movement of graphical objects is investigated. An interactive environment for producing musically-controlled computer animations is presented. The graphical objects studied are based on Todd's and Latham's work in evolutionary art. The environment permits the creation of kernel objects using an interactive toolset. In addition to a basic set of morphological definitions, each object incorporates a *script*, which is an instance of programming language code and data definitions. Scripts permit the run-time computation of object characteristics, and when done in a temporal setting, allow complex animation control. The script language has a number of functions that can access MIDI information, as read into the system via a MIDI file. The practical consequence of this is that animations are controllable with music data, in which music determines the movement and morphology of animated objects. The tight integration of music and animation in an interactive production environment such as this one has a number of pragmatic consequences, ranging from the ability to automatically synchronize complex activities to music, to the use of music as a creative source for graphical sculptoring and animation.

Keywords: animation, music, MIDI, evolutionary art.

1 Introduction

Sound effects and music are an intrinsic requirement to most computer animation productions. The synchronization of sound effects and music to animation traditionally has been a latter-stage post-production task. Digital technology is proving to be successful in application to this problem, as can be seen by the growing popularity of multimedia authoring systems and non-linear editors. In some cases, however, it is preferable to automate the synchronization of sound and animation – generating animation features in preparation for audio to be used with the production. For example, the complexity of synchronizing speech to animated character mouth movement is greatly simplified with the use of techniques that automate animated lips to digitized speech signals during the animation production phase. Some commercial animation environments permit external control through system interfaces, which is required for motion capture (eg. Softimage's channels [7]). Although these systems can use music data (with the help of additional software support), interfaces permitting interactive inspection and acquisition of music features by the animator are not included. Most work in the automation of music and animation has been directed to the opposite problem of this paper – creating music from visual input information. Representative examples of this line of research, many of which use fractal equations to generate both graphics and musical information, are in [3, 4, 13, 5, 1, 6, 2, 11].

This paper presents an environment that automates animation and music. An interactive toolset has been implemented that permits the construction of an interesting set of *creature kernels* - basic objects suitable for subsequent mutation and animation. The graphic objects chosen are inspired by Todd's and Latham's evolutionary art [15], and were chosen for their generality and natural, life-like appearance. The system permits the run-time evolution of creatures through the use of *scripts* that are defined within kernel objects. Scripts contain programming language code that control local and global characteristics of objects. With respect to music, the script language recognizes MIDI (Musical Instrument Digital Interface) files [8, 14]. Script functions can access MIDI file information. When executed in a temporal fashion, creature characteristics are essentially controlled by the MIDI music composition, in the manner dictated by the script. The net effect of automated MIDI control is that the creatures' movement and morphology are automatically synchronized with and derived from music. Consequently, the music data can be considered to be a "sculptor" of graphical objects, and in concert with appropriate script control, can control object and animation events in a variety of complex ways. A net effect of the system is that a music composition can be used as a source for mutating and controlling an animated creature in creative and often unexpected ways.

This work was motived by the desire to implement an animation environment that embraces music as a primary means of animation design and control. We designed an interactive production environment that merges music and animation into one integrated toolset. In addition, the system is designed to generate graphical information from the music source data, since musical scores are often predefined and cannot be altered to conform to animation requirements. Thus the approach is conducive towards both synchronizing animation events to music and sound effects, as well as using music as a creative means of inventing new animation content.

Section 2 discusses the interactive modeling environment. The musical and animation control functions of the system are described in section 3. Section 4 presents an example production. Future extensions and related work conclude the paper in section 5.

2 Modeling

Figure 1: Composite object dialogue

The objects modeled are inspired by the work of Todd and Latham [15]. They present a graphics system that supports the user–influenced evolution of graphical "creatures" (eg. figure 7 (ii)). Their system is built upon an object–oriented object construction language that permits hierarchical, recursive object definitions similar in spirit to L-system models [16]. A key characteristic of Todd's and Latham's work is their ascribing a genetic–like metaphor for models. By tweaking object "genes", a rich variety of models can be derived and evolved.

Our system borrows some of the more interesting features of Todd's and Latham's graphical model, incorporating them within a conventional interactive GUI editor. At the lowest level, objects are composed of a number of geometric primitives, such as spheres, tori, and cones. These can then be recursively and hierarchically composed together. The system makes use of transformational inheritance: a graphical transform applied to a parent is inherited by its children. Children objects may have their own specific transforms defined for them as well. The system supplies a number of composite patterns, such as horns and starfish. In concert with standard graphical transforms and user-defined object definitions, a surprisingly rich set of models can be derived. Material and lighting definitions are also assignable to composite and individual objects. Figure 1 is an example dialogue box for constructing composite objects. Figure 2 shows a typical model hierarchy, from which individual objects can be edited and specialized if desired.

Figure 2: Model hierarchy

3 Musical control of graphical objects

Besides containing graphical transform and material information, each object also may have a *script* defined for it. A script is a programming language instance executed for that object for each frame of the animation. An object may have a unique script, no script, or share a script with other objects. In the latter case, execution of scripts may use the same inheritance scheme employed with transforms.

The language defined is termed ACL (Animation Control Language), and is an interpreted language with a syntax and semantics similar to C [9]. An example script in a script dialogue window is in figure 3. ACL supports basic data types and control mechanisms

Figure 3: Object script

(tests, loops, iteration). Because ACL is fairly conventional, we forgo further discussion of its design. Of importance to the animation system, however, are a number of reserved ACL keywords that refer to graphical transform parameters of objects, and MIDI events as resident in a globally accessible MIDI data file. We will discuss these aspects of the language in more detail below.

MIDI (Musical Instrument Device Interface) is a protocol for exhanging information between digital musical instruments and computers [14]. MIDI is a serial protocol in which musical events are converted into standardized packets of information. Events can include notes turned on and off on a keyboard, volume and pitch alterations, timing, instrument channels, and specialized events particular to specific manufacturer's hardware. For example, when a key is pressed on the keyboard, a *note on* message is transmitted. This event is also followed by information describing which key on the keyboard is pressed (represented as a number between 0 and 127) and the velocity at which the key was pressed (also between 0 and 127). Other events follow this same pattern - an EVENT descriptor followed by data values for that event. MIDI software on instruments or computers will

Figure 4: Midi viewer/filter

collect the serial stream of MIDI information, and save it in a file for later playback and editing.

An important feature of MIDI information that is useful in an animation environment is the means for denoting the temporal delays between musical events. The *delta* field of a MIDI data stream is a packet item that describes the temporal delay in clock ticks between the previously seen and current events in the stream. When MIDI data is being transmitted in real-time, the delta parameter is often superfluous, as the real-time transmission of events implicitly denotes their timing in the performance. However, if the MIDI data is being stored in a file, delta values are the means used for reconstructing the timing of events. Given a number of chronological events, their absolute times are computed by incrementing each event's delta value to a global time value.

The system handles the more essential events defined by the MIDI standard. This includes note on, note off, polyphonic aftertouch, channel, aftertouch, and pitch wheel events. Figure 4 shows a midi viewer that correlates music events with animation frames. Because the current incarnation of the system requires the animator to implement a programmed ACL script that controls an animated object according to some musical criteria, it is important that access to the music data is made as manageable and intelligible as possible. Therefore, a number of midi filters are available for removing superflous events from view. For example, the more sophisticated keyboards monitor key pressure over time after a key has been pressed, and there can be an enormous number of such signals generated during realtime MIDI recording.

The MIDI display of events in the interactive environment is for information feedback purposes only. When generating the final animation, the MIDI data is processed from start to finish, and is correlated in time with the animation . Each script defined will access the current state of MIDI information according to the programming instruction in the script. Accessing MIDI information can be done visaviz globally defined variables. For example, the current event, channel, and data values can be immediately accessed by referring to the appropriately defined system variable. For example, in

$$ROTATE\_X = ROTATE\_X + CURRENT\_NOTE\_ON;$$

the X-rotation transformation is incremented at a rate determined by the current note being played. Since higher octave notes have higher note numbers, the rotation rate accelerates as higher pitched notes are played.

Another useful ACL ability is a function that searches ahead in time for particular events. This is extremely useful for determining tweening rates, as it permits us to move forward in time from the current frame to see when a desired musical event happens in the future, and thereby determine an appropriate interpolation factor.

In combination with the conventional expression and programming language constructs defined in ACL, the musical control possibilities are unlimited. Using flags and conditional processing, an object's script can specify different activities for the object during the course of the animation. In addition, extracted music values can be arithmetically altered and computationally processed. All such computations can then be used to control any graphical transformation of the object.

4 Examples

Figure 5: Screen shots from *Gyroscope*

Figure 5 shows three screenshots from an animation entitled *Gyroscope*. Three tori are used. One torus spins at a constant rate. A second tori spins 180 degrees every time a note is played on MIDI channel 1. It's script is in figure 6. Should the frame be one in which a "note on" event is occurring on MIDI channel 1, then the velocity is set to a maximum

```
float delta_rz, velocity; /* variable declarations */

if (CUR_FRAME == 0) delta_rz = 0.0; /* Initialize for first frame */
if (NEW_FRAME) { /* only one note event per frame */
        if (EVENT) { /* Did an event occur? */
                /* Check to see if a NOTE_ON occurred on channel 1 */
                if ((midi.event == NOTE_ON) && (midi.channel == 1)) {
                       velocity = 200.0; /* Spin torus at max. velocity */
        } }
        GLOBAL_RZ = GLOBAL_RZ + velocity;
        GLOBAL_RY = GLOBAL_RY + velocity;
        velocity = velocity - delta_rz;
        /* make sure velocity never drops below 0... */
        if (velocity < 0.0) velocity = 0.0;
}
```

Figure 6: Script to note-trigger torus rotation

value. Each subsequent frame will have this velocity decremented, until it reaches zero. A third tori also spins 180 degrees, but when notes are played on channel 2. It's initial velocity and deceleration, however, are dependent upon the interval duration between the trigger and the following one in the MIDI stream. Hence it's rotation adapts to the relative rhythm of notes on channel 2.

Figure 7 (i) is a shot from an animation used to show the utility of the system for autosynchronization of sound and animation for more traditional production content. A stylized birthday cake is modeled, and candles appear on the cake synchronized to notes from the tune "Happy Birthday". Flames then appear in a similarly synchronized fashion. The flames flicker according to a trigonometric sine function in ACL, showing the practicality of combining MIDI and user–derived algorithmic control.

Finally, figure 7 (ii) gives an idea of the organic style of creatures possible in the system. This snapshot is based on an animation in progress, in which the creature size, shape, and colour is controlled by various aspects of music - such as pitch, velocity, and tempo.

5 Conclusion

This paper has shown the creative possibilities of using music to control computer animation. Our coalescence of music and graphics is more substantial than merely synchronising

Figure 7: (i) Birthday cake; (ii) Creature

sound with animation, but rather, uses music as a mechanism for defining the fundamental form of graphical objects and their animated behaviour. Todd and Latham mention the chaotic nature of this style of graphical modeling, as the smallest change in one parameter can profoundly alter a creature's overall form [15]. Likewise, minute changes in the music data can result in wildly different animations. The system is object–oriented in flavour, as the object definitions and scripts are analogous to object instances and demons in object–oriented programming languages.

A number of extensions of this work are possible. The system is implemented in C on a Silicon Graphics Elan workstation, using GL and the Forms user interface library. Animated frames are output as Targa files. As GL is a Gourand renderer, utilizing a rendering engine such as a raytracer would be a useful enhancement. The kernel objects themselves can be extended with a more complete subset of the constructions used in Todd's and Latham's systems. The script language is easily extensible, and a library of musically-interesting script functions is possible. A powerful extension is to permit the mutation of kernel definitions themselves by the script language, thereby permitting musically-controlled morphing. In addition, it would be interesting to interface the MIDI controls of the system into a commercial animation environment, perhaps replacing ACL with a GUI control system.

A major enhancement worth considering is the real–time control of animation using music data. The most difficult problem arising from real-time control is the problem of predicting event times, which are required for tweening purposes. For example, a musical source with varying tempo poses great difficulty to an animation that is trying to synchronize some event to a regular tempo. A possible solution is to perform real-time predictive beat tracking [12, 10].

Acknowledgement: This work is supported through NSERC Operating Grant 0138467.

References

[1] P. Beyls. Musical Morphologies from Self-organizing Systems. *Interface*, 19:205–218, 1990.

[2] R. Bidlack. Chaotic Systems as Simple (but Complex) Compositional Algorithms. *Computer Music Journal*, 16(3):33–47, 1992.

[3] C. Dodge and C. R. Bahn. Musical Fractals. *Byte*, pages 185–196, June 1986.

[4] B. Evans. Integration of Music and Graphics through Algorithmic Congruence. In *Proceedings 1987 International Computer Music Conference*, pages 17–24, 1987.

[5] B. Evans. Enhancing Scientific Animations with Sonic Maps. In *Proceedings 1989 International Computer Music Conference*, pages 105–108, Columbus, Ohio, 1989.

[6] M. Gogins. Iterated Functions Systems Music. *Computer Music Journal*, 15(1):40–48, 1991.

[7] Softimage Inc. *Softimage 3D User's Guide.* Microsoft Corp, Montreal, 1995.

[8] International MIDI Association, North Holliwood, CA. *MIDI 1.0 Specification*, 1983.

[9] B.W. Kernighan and D.M. Ritchie. *The C Programming Language.* Prentice-Hall, 1978.

[10] E.W. Large. Beat Tracking with a Nonlinear Oscillator. In G. Widmer, editor, *IJCAI Workshop in Artificial Intelligence and Music*, pages 24–31, Montreal, Canada, 1995.

[11] J. Nakamura, T. Kaku, T. Noma, and S. Yoshida. Automatic Background Music Generation based on Actors' Emotion and Motions. In *Proceedings Pacific Graphics '93*, volume 1, pages 147–161, Seoul, Korea, 1993.

[12] P.E.Allen and R.B. Dannenberg. Tracking Musical Beats in Real Time. In *Proceedings 1990 International Computer Music Conference*, pages 140–143, Glasgow, Scotland, 1990.

[13] J. Pressing. Nonlinear Maps as Generators of Musical Design. *Computer Music Journal*, 12(3):35–46, 1988.

[14] J. Rothstein. *MIDI: A Comprehensive Introduction.* A-R Editions, Inc, Madison, WI, 1992.

[15] S. Todd and W. Latham. *Evolutionary Art and Computers*. Academic Press, 1992.

[16] C. van der Mark. Lparser: a tutorial. *3D Artist*, (17):34–37, 1994.