A Symbiosis of Animation and Music

Robert Pringle
Side Effects Software
Toronto, ON, Canada M8V 2E7
springle@global-sidefx.com

Bria J. Ross
Dept of Computer Science, Brock University
St. Catherines, ON, Canada L2S 3A1
bross@cs.brocku.on.ca

Abstract

An interactive environment for producing musically controlled computer animations is presented. Objects are initially modeled using an interactive toolset. Each object incorporates a script, which is an instance of programming language code and data definitions that permit complex object and animation control. The script language has a number of functions that access MIDI information, which therefore allows music to determine the movement and morphology of animated objects. The tight integration of music and animation in an interactive production environment results in the ability to automatically synchronize complex animation activity to music, and lets musical compositions act as a creative source for graphical sculpting and animation.

1 Introduction

This paper describes an environment that automates animation and music. An interactive toolset has been implemented that permits the modeling of organic-like objects suitable for subsequent mutation and animation. The graphic objects used are inspired by Todd's and Latihan's evolutionary art (Todd & Latihan, 1992), 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 (IMA, 1993). Script functions can access MIDI file information. When exercised in a temporal fashion, creature characteristics are controllable by the MIDI sonic 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.

This work was motivated by the desire to implement an animation environment that embraces music as a primary stream of animation design and control. We designed an interactive production environment that integrates music and animation into a single environment. In addition, the system is designed to generate graphical information from the music source data, since musical scores are often pre-defined 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 source for deriving animation content.

Section 2 reviews related work. Section 3 discusses the interactive modeling environment. The musical and animation control functions of the system are described in section 4. Section 5 reviews some example productions. Future extensions conclude the paper in section 6.

2 Background

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 later-stage post-production task. Digital technology is being successfully applied 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 an-
imation behavior appropriate for the audio to be used in the production. For example, the complexity of synchronising speech to animated character mouth movement is greatly simplified with lip synchronisation software. Some commercial animation environments permit external control through system interfaces, which is required for motion capture (eg. Softimage's channels (Softimage, 1995), Prisma's surface operations (Prisma, 1995)). Although these systems can use music data (with the help of additional software support), it does not permit interactive inspection and acquisition of music features by the animator as noted above.

The use of music to control graphical information is not new, although little work has been done in it. "Light organs" were popular in the sixties, in which light flashes in synchronisation to musical audio signals. The video More Bells and Whistles is a recent example of an animation derived largely from real source data (Lytle, 1990). Much more work in the automation of music and animation has been directed to the opposite problem of this paper — creating music from visual input information (Evans, 1987; Bledack, 1992; Nakanuma et al., 1993).

3 Modeling

The objects modeled are inspired by the work of Todd and Latham (Todd & Latham, 1992). They present a graphics system that supports the user's influence of evolution of graphical "creatures" (eg. Figure 3). Their system is built upon an object-oriented graphics language that permits hierarchical, recursive object definitions similar in spirit to L-systems (Peitgen et al., 1992). A key characteristic of their system is the scripting of a genetic-like metaphor to the 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 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.

4 Musical control of graphical objects

Besides controlling 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.

Figure 1: Object script

The scripting language defined is called ACL (Animation Control Language), and is an interpreted language with a syntax and semantics similar to C (Kernighan & Ritchie, 1978). An example script in a script dialogue window is in Figure 1. ACL supports basic data types and control mechanisms (e.g., loops, iteration). Because ACL is fairly conventional, we forego further discussion.

ICMC Proceedings 1996

Peingle & Ross
of its design. Of importance to the animation system, however, are a number of ACL functions that refer to graphical transform parameters of objects, and MIDI events as resident in a globally accessible MIDI data file (IMA, 1983).

MIDI (Musical Instrument Device Interface) is a protocol for exchanging information between digital musical instruments and computers. 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 delay 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 delay parameter is often superfluous, as the real-time transmission of events implicitly defines their timing in the performance. However, if the MIDI data is being stored in a file, delay values are necessary to reconstructing the timing of events. Given a number of chronological events, their absolute times are computed by adding each event’s delay 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, and pitch wheel events. Because the current version of the system requires the animator to implement an 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 superfluous events from view. For example, most keyboards generate aftertouch signals, and there can be an enormous number of such signals generated during realtime MIDI recording. Filtering these events from view greatly helps the animator.

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 will access the current state of MIDI information according to its ACL code. MIDI information can obtained through globally-defined variables. For example, in

rotate_x = rotate_x + current_note_delay;

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.

ACL is also able to search ahead in the musical score for particular MIDI events that will occur in the future. This is extremely useful for determining twinning 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 curve.

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 mathematically altered and computationally processed. All such computations can then be used to control any animatable characteristic of the object.

5 Example

Figure 2 shows three screenshots from an animation entitled Gynoscope. 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. 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 value. Each subsequent frame will have this velocity decremented, until it reaches zero. A third torus 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 3 gives an idea of the organic style of creatures possible in the system. This snapshot is based on an animation in which the creature size, shape, and color is controlled by various aspects of music – such as pitch, velocity, and tempo.

Pringle & Ross
6 Conclusion

Our coalescence of music and graphics is more substantial than merely synchronising sound with animation, but rather, uses music as a mechanism for defining the fundamental form of graphical objects and their animated behavior. Todd and Latham mention the chaotic nature of this style of graphical modeling, as the small change in one parameter can profoundly alter a creature's overall form (Todd & Latham, 1992). Likewise, minute changes in the music data can result in vastly different animations. The system is object-oriented in flavour, as the object definitions and scripts are analogous to object inns and methods 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. As GL is a Gouraud renderer, utilizing a rendering engine such as a ray-tracer would be a useful enhancement. The object models 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 would be to permit the mutation of object primitives 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 AGL with QUIT control.

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 prediction of event times, which are required for swamping 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. One possible solution is to perform real-time predictive beat tracking (P.E. Allen & Dannenberg, 1990; Large, 1995).

Acknowledgement: This work is supported through NSERC Operating Grant 013467.

References


