Roger B. Dannenberg
School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213 USA
rbd@cs.cmu.edu

Interactive Visual Music: A Personal Perspective

Interactive performance is one of the most innovative ways computers can be used in music, and it leads to new ways of thinking about music composition and performance. Interactive performance also poses many technical challenges, resulting in new languages and special hardware including sensors, synthesis methods, and software techniques. As if there are not already enough problems to tackle, many composers and artists have explored the combination of computer animation and music within interactive performances. In this article, I describe my own work in this area, dating from around 1987, including discussions of artistic and technical challenges as they have evolved. I also describe the Aura system, which I now use to create interactive music, animation, and video.

My main interest in composing and developing systems is to approach music and video as expressions of the same underlying musical "deep structure." Thus, images are not an interpretation or accompaniment to audio but rather an integral part of the music and the listener/viewer experience. From a systems perspective, I have always felt that timing and responsiveness are critical if images and music are to work together. I have focused on software organizations that afford precise timing and flexible interaction, sometimes at the expense of rich imagery or more convenient tools for image-making.

"Interaction" is commonly accepted (usually without much thought) as a desirable property of computer systems, so the term has become overused and vague. A bit of explanation is in order. Interaction is a two-way flow of information between live performer(s) and computer. In my work, I see improvisation as a way to use the talents of the performer to the fullest. Once a performer is given the freedom to improvise, however, it is difficult for a composer to retain much control. I resolve this problem by composing interaction. In other words, rather than writing notes for the performer, I design musical situations by generating sounds and images that encourage improvising performers to make certain musical choices. In this way, I feel that I can achieve compositional control over both large-scale structure and fine-grain musical texture. At the same time, performers are empowered to augment the composed interactive framework with their own ideas, musical personality, and sounds.

Origins

In the earliest years of interactive music performance, a variety of hardware platforms based on minicomputers and microprocessors were available. Most systems were expensive, and almost nothing was available off-the-shelf. However, around 1984, things changed dramatically: one could purchase an IBM PC, a MIDI synthesizer, and a MIDI interface, allowing interactive computer music using affordable and portable equipment. I developed much of the CMU MIDI Toolkit software (Dannenberg 1986) in late 1984, which formed the basis for interactive music pieces by many composers. By 1985, I was working with Cherry Lane Technologies on a computer accompaniment product for the soon-to-be-announced Amiga computer from Commodore. Cherry Lane carried a line of pitch-to-MIDI converters from IVL Technologies, so I was soon in the possession of very portable machines that could take in data from my trumpet as MIDI, process the MIDI data, and control synthesizers.

Unlike the PC, the Amiga had a built-in color graphics system with hardware acceleration. At first, I used the Amiga graphics to visualize the internal state of an interactive music piece that was originally written for the PC. It was a small step to use the graphics routines to create abstract shapes in response to my trumpet playing. My first experiment was a sort of music notation where performed notes were displayed in real-time as growing squares positioned according to pitch and starting time. Because time wrapped around on the horizontal axis, squares in different colors would pile up on top of other squares, or very long notes would fill large portions of the screen, effectively erasing what was there before. This was all very simple, but I became hooked on the idea of generating music and images together. This would be too much for a performer alone, but with a computer helping out, new things became possible.

Lines, Polygons, Screens, and Processes

One of the challenges of working with interactive systems, as opposed to video or pre-rendered computer animation, is that few computers in the early days could fill a screen in 60 msec, which is about the longest tolerable frame period for animation. Even expensive 3-D graphics systems fell short of delivering good frame rates. The solution was to invent interesting image manipulations that could be accomplished without touching every pixel on every frame.

Animation on Slow Machines

Like many modern computer graphics systems, early computer displays used special, fast, video RAM separate from primary memory. Unfortunately, this RAM was so expensive that it was common to have only 4 to 8 bits per pixel, even for color displays. These limited pixels served as an index into a color lookup table. In this way, the computer could optimize the color palette according to the image (Burger 1993). A common animation trick was to change the color table under program control. For example, if there were 20 objects on the screen, each object could be rendered with a different pixel value, for example the integers 0 through 19. The actual colors of the objects were then determined by the first 20 elements of the color lookup table. To make object 15 fade to black, one could simply write a sequence of fading colors to element 15 of the color lookup table. This was much faster than rewriting every pixel of the object, so it became possible to animate many objects at once.

Another standard trick with color tables was "color cycling," where a set of color values were rotated within the color lookup table. Effects analogous to chaser lights on movie marquees could be created, again with very little computation. Scenes could also be rendered incrementally. Drawing objects one by one from lines and polygons can be an interesting form of animation that spreads the drawing time over many video frames. Polygons can grow simply by overwriting larger and larger polygons, again with low cost as long as the polygons are not too large.

One of my first animations had worm-like figures moving around on the screen. Although the "worms" appeared to be in continuous motion, each frame required only a line segment to be added to the head and a line segment to be erased at the tail. Sinuous paths based on Lissajous figures were used in an attempt to bring some organic and fluid motion to the rather flat two-dimensional look of the animation. At other times, randomly generated polygons were created coincident with loud percussion sounds, and color map manipulations were used to fade these shapes to black slowly.

Scheduling Concerns

Because processors in the 1980s were executing at best a couple of million instructions per second, it became important to think about the impact of heavy graphics computations on music timing. In short, one would like for music processing to take place with high priority so that music events such as MIDI messages are delivered within a few milliseconds of the desired time. Otherwise, rhythmic distortions will occur, and sometimes these will be audible. On the other hand, graphics frame rates might be 15 to 60 frames per second (fps), and delaying the output by 15 msec or so is generally not a problem. We do not seem to perceive visual rhythm with the same kind of precision we have in our sense of hearing. The solution is to compute graphics and music in separate processes. This, however, raises another problem: how can we integrate and coordinate images and music if they are running in separate processes?

One solution is to control everything from the high-priority music process. In my work with the

Amiga, which had a real-time operating system, I ran the CMU MIDI Toolkit at a high priority so that incoming MIDI data and timer events would run immediately. Within the CMU MIDI Toolkit thread, there could be many active computations waiting to be awakened by the scheduler. A typical computation would combine music and graphics. For example, a computation might call a function upon every note to generate some MIDI data. The same function might also change a color entry in the color table or draw a line or polygon, creating some visual event or motion. By writing the music and animation generation together—literally in adjacent lines of program code—one is able to use the same timing, control parameters, sensor data, and program state to drive both the music and the image. In addition, the efficient real-time control and scheduling mechanisms available in the CMU MIDI Toolkit became available for all kinds of real-time, interactive graphics control (Dannenberg 1993).

To prevent graphics operations from interfering with low-latency music processing, graphics operations merely placed operations and parameters into a queue. Another lower-priority thread interpreted the data as quickly as possible. Because graphics rendering operated at a low priority, music processing could interrupt (i.e., preempt) the graphical rendering to handle more music computation.

It is worth pointing out some key differences between traditional computer animation and my approach to visual animation for music. The traditional approach, used in virtual reality and game systems, renders a sequence of image frames according to a "world model" that includes the position and geometry of every object to be rendered. Usually, the rendering is asynchronous. As soon as a frame is finished, a new one is started. When frame rates are very low (as they were when this technique evolved), it makes sense to create frames as fast as possible. Therefore, in this approach, the application may be updating the world model more often than the model is actually rendered. Alternatively, the updates to the world model may be synchronized to the frame rate. Each update period, the program computes a new world state given the elapsed time since the previous update. Software for virtual reality, often running on Silicon Graphics workstations, has been used in a number of projects for interactive music performances (Gromala, Novak, and Sharir 1993; Bargar et al. 1994).

In contrast, if enough computing power is available, it makes sense to schedule frames at precise times rather than simply to run as fast as possible. With this approach, there will be more processor time available for music processing, and the computation of frames can be integrated with the scheduling of other events. All events are now driven by time. I believe this approach can be much more logical for the composer/programmer. It is especially appropriate when graphics operations are performed in synchrony with musical events rather than in synchrony with the frame rate.

Now that it is possible to render full frames at animation rates, my programs are organized more like traditional animation programs. The world state is maintained by a set of objects that are called upon to draw themselves on every frame. However, I schedule frame computation at a fixed rate to make timing more deterministic and to leave CPU time for other processes.

Perception and Esthetics

The presence of animation in a music performance does not simply add a new dimension. The psychology of music-listening is the subject of much research (Deutsch 1999; Juslin and Sloboda 2001), but it is not well understood, and things are only more complex when visual processing is involved. Rather than presenting a full theory of interactive visual music or even a review of related work (some of which can be found in the present issue of *Computer Music Journal*), let me make a few observations based on experience as a composer, performer, and listener/viewer.

Complexity and Sensory Overload

First, less can be more. Modern music can be very complex, requiring full attention and repeated listening to even begin to understand it. The idea that an equally rich, complex, visual field can be absorbed at the same time vastly overestimates the powers of human perception. Some composers may pursue this path as a logical extension of the complexity that has arisen in modern music, in spite of the perceptual difficulties. Others may want to consider film music as a model. A successful composer described to me how he sometimes writes very simple lines—lines that might be too simple to stand alone—knowing that the picture and dialog will complete the film and satisfy the audience. This is not to say music and sound in film (see Bordwell and Thompson 1986; Altman 1992; Chion 1994) is simple—only that image and sound perception are intertwined.

In a similar way, if we place rich music and performers in the foreground, it makes sense that images can consist of "very simple lines" and still be very satisfying. Audience members have often commented that there is simply too much to see and hear in some of my pieces, and to some extent this is purposeful. This is a much better reaction than "I left early because nothing was happening," but I am still struggling with this problem. Images, sounds, and their impressions cannot be considered separately, because the perception of each one is affected by the other.

Visual and Auditory Time

Secondly, visual events do not work the way sound events do. As an example, consider again the "animated worms" mentioned above. These were set in the context of a steady eighth-note rhythm at about 150 beats per minute (200 msec per eighth note). The worms "moved" by adding to the head and erasing from the tail every eighth note in synchrony with the music. Now, 200 msec means a 5-Hz update rate, which is well below the rate at which frames fuse into continuous motion. This is also well beyond the asynchrony (around 50 msec) below which visual and auditory events seem to be simultaneous. Therefore, it should be quite apparent that the worms are constructed from distinct visual elements, and it should be clear whether these elements appear in synchrony with the music or not.

However, many people who see this performance never notice the coincidence. There may be evolutionary reasons for this. If one hears footsteps at a rate of 300 per minute in the jungle, one might want to pay attention, but it seems unlikely one would ever see flashing lights or visual stimuli at a similar frequency. On the other hand, we are very sensitive to visual patterns, spatial frequency, and motion. Although there are many parallels between music and animation, and the synaesthetic possibilities are very appealing, composers should be careful not to fall into the trap of mapping musical experience directly into the visual world.

Making the Connection Interesting

Third, there is a common temptation to draw connections between music and image at a very superficial level. The best (and worst) examples of this are "music visualizers" as supported by, for example, Windows Media Player, WinAmp, and iTunes. I dislike the animations created by these programs because they offer only what is readily apparent in the music itself. As soon as the obvious connections from sound to image are made, the image ceases to be interesting or challenging. A better approach that is particularly available to composers is to make connections between deep compositional structure and images. Usually in interactive music systems, there are some music generation components with control parameters and state information that affect the music structure but which are not directly perceivable. By tying visuals to this deep, hidden information, the audience may perceive that there is some emotional, expressive, or abstract connection, but the animation and music can otherwise be quite independent and perhaps more interesting.

For example, in my *Nitely News* (1993/1995), four contrapuntal musical voices are connected to four graphical "random walks" that accumulate to form a dense image over time (see Figure 1). Later, a stick-figure dancer appears and reacts to changes in tempo and improvisational style (see Figure 2). Thus, the

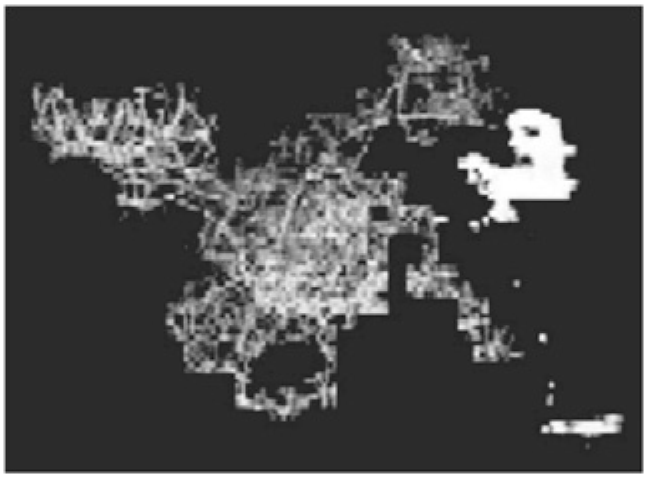
Figure 1. Image from Nitely News *(1993/1995). As computer-generated musical lines evolve, graphical random walks generate a dense image.*

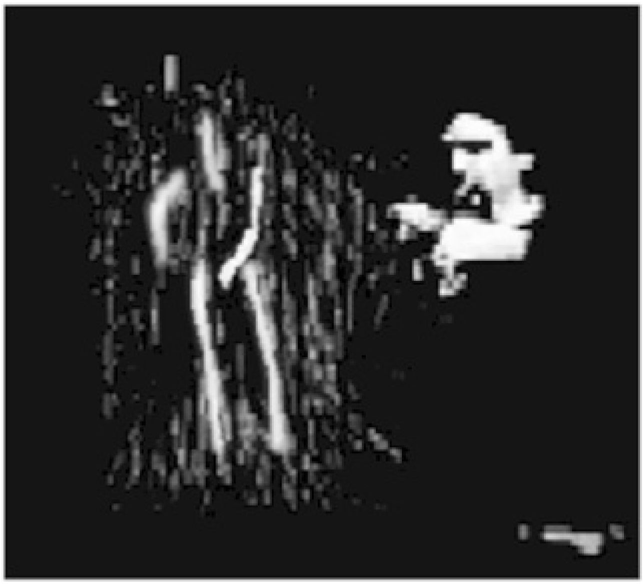
Figure 2. Another image from Nitely News. *An animated stick-figure dancer responds to trumpet improvisations. In this still image, the dancer image has been enhanced manually to make it stand out from the background.*

image is related to the musical organization and mood rather than superficial parameters such as instantaneous amplitude or pitch.

Revealing the Connection

There is also the danger of making connections so incomprehensible that no one senses any connection at all. One is reminded of John Cage's comment that the relationship between his music and Merce Cunningham's choreography was that the music was playing while Merce was dancing. But many would agree that their collaborations resulted in spontaneous connections at some level that could be felt and enjoyed.

Many composers take care to "teach" their listeners what their music is about by stating themes clearly, by repeating important material, and by developing ideas slowly, at least early on. This approach can be taken with visual material as well. For example, if a particular sound or gesture affects some visual parameter, the audience is much more likely to "get it" if the sound and the visual effect are presented in relative isolation. In contrast, if this takes place amid a chaotic blend of other sounds and images, it is unlikely that anyone will detect the connection. As in music composition and perhaps any art form, finding an interesting middle ground between the obvious and the obtuse is always a challenge.

Pixel Graphics, 3-D Graphics, and Video

As processors and graphics accelerators became faster, new interactive animation techniques became possible. For example, around 1995, Intel processors ran at about 100 MHz, and it was possible to create some fairly interesting animation in software. In particular, one could copy pixels from images to screen buffers at reasonable frame rates, allowing for interesting "sprite" animation. (A sprite is a small moveable image that is displayed over a fixed background image.) In *The Words Are Simple* (1995), I digitized colorful fall leaves using a flatbed scanner and developed software that could erase a set of leaves, replace the background image, and redraw the leaves in a new position, all within a single frame (see Figure 3). The machines I had were still too slow to redraw the entire background, but by restricting the total leaf area, I could run at 15 to 20 fps. Double buffering was used so that the redrawing would not be visible. Having more memory than speed, I precomputed images of the leaves rotated to many different angles so that I could control position and rotation. The final composition used a combination of drawing onto a background image, controlling sprites, and manipulating the color map.

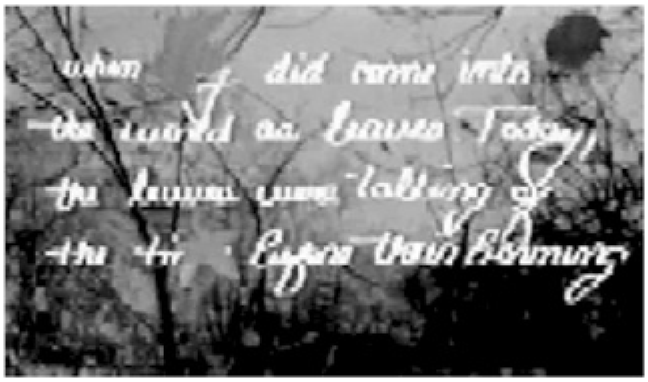

Figure 3. In The Words are Simple *(1995), a background fall image is overlaid with handwriting from a graphics tablet and falling, spinning leaves.*

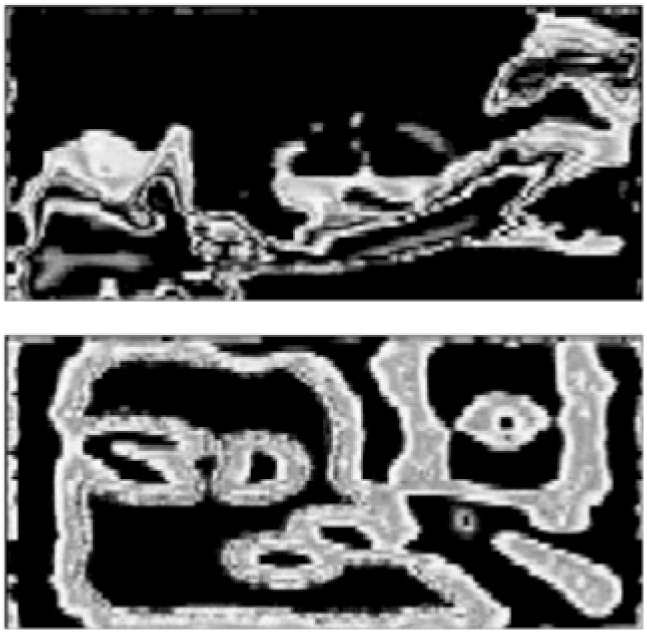

Figure 4. In Transit *(1997) uses iterated function systems and simulated heat diffusion to generate chaotic, flowing images with high-level controls derived from music improvisation.*

In 1996, Scott Draves, a graduate student at the time, was writing some very interesting programs that iterated image-processing algorithms over an array of pixels. Many of the algorithms included some non-linearities, resulting in chaotic behavior and self-organizing patterns. The organic, abstract look of the images was very appealing at any given moment, but lacked any long-term evolution or development that could sustain one's interest. One way around this was to control the program manually using keyboard commands, an approach seen earlier in the "ImproVision" program of Bernard Mont-Reynaud in 1987 (Mont-Reynaud 2005). In our collaboration, Scott and I created an interactive piece (see Figure 4) where musical parameters at different levels of structure would cause systematic changes in the images (Rowe 2001).

Around this time, consumer-oriented 3-D graphics cards became available for PCs, although it took some time to achieve high performance, solid drivers, and operating system support. Now, one can develop very sophisticated, interactive 3-D animations using OpenGL, a very flexible software interface that runs on all popular operating systems and is supported by many graphics cards (Woo et al. 1999). Modern graphics processors are much faster than CPUs, allowing dramatic improvements in frame rates, image resolution, and image sophistication. For example, Figure 5 shows some live animation written in OpenGL and projected behind soloists in my work *Uncertainty Principle* (2000).

One of the problems, though, is that OpenGL and graphics processors are optimized for the kind of graphics we see in video games: a virtual world where most objects are frozen in place, constructed with flat polygons, and lacking in surface detail. As with any technology (MIDI synthesizers come to mind), one must be careful and creative to use technology artistically, especially when the technology was designed with other goals in mind.

A promising new direction is video. While just a few years ago, it was impossible to update a computer display at video rates, we can now capture live video, manipulate it, and copy it to a screen in real time. Many artists have used videodisc (and now DVD) players essentially as "video samplers" capable of playing video clips under computer control. This allows arbitrarily complex video to be displayed but sacrifices the ability to manipulate images in real time. Just as sampling-based synthesis brought rich sounds of the real acoustic world into the realm of digital manipulation, video brings real-world images into the domain of interactive computer graphics. The unit-generator, stream-processing approach of computer music languages can be applied to video processing, and this is already possible in programs such as EyesWeb (Camurri et al. 2000), Jitter by Cycling '74, GEM (Danks 1997), DIPS

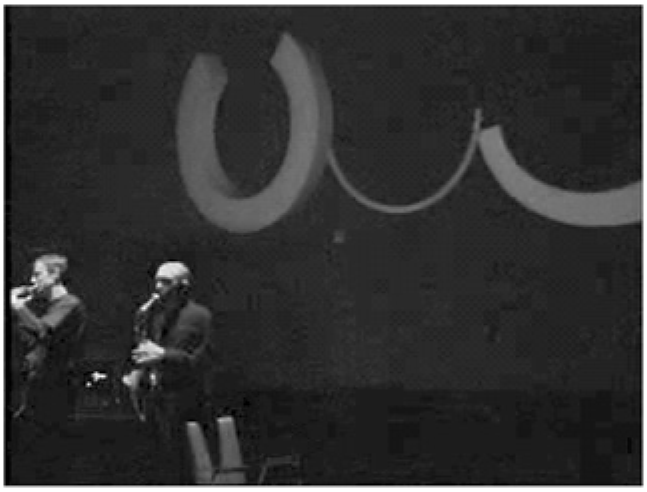
Figure 5. Live animation using OpenGL in Uncertainty Principle *(2000). Roger Dannenberg (trumpet) and Eric Kloss (alto saxophone) are performing.*

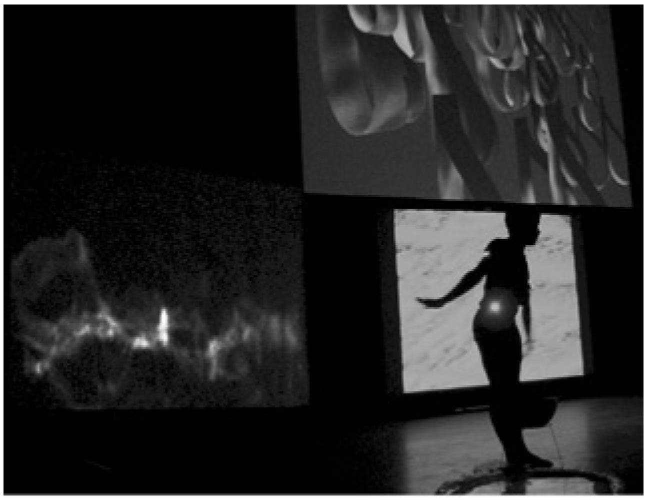
Figure 6. In The Watercourse Way *(2003), reflections (left) from a pool of water (lower right) are captured by video camera and control evolving audio spectra.*

(Matsuda and Rai 2000), and Isadora by TroicaTronix. Other approaches can be seen in Videodelic by U & I Software, Processing (see www.proce55ing.org), and Electronika by Aestesis. As more processing power becomes available (which is inevitable), many new video processing and image making techniques will be explored.

Video processing opens up the possibility of images controlling sounds. For many years, researchers have created interactive systems that use video cameras for sensing (Krueger 1983; Rokeby 1997; see also STEIM's BigEye software, available online at www.steim.org/steim/bigeye.html). In some ways, this is simpler than image generation, because low frame rates and very-low resolution can often be used for sensing, and software does not necessarily need to process every pixel. On the other hand, there is a tendency to use video merely as a triggering device, and there are relatively few examples where video-based controllers achieve continuous and subtle sound control. In *The Watercourse Way* (2003), I used video to capture light reflected from water, transforming this into time-varying spectra for synthesizing sound (Dannenberg et al. 2003; Dannenberg and Neuendorffer 2003). The patterns of light were clearly audible in the sounds. This was augmented by the theatrical aspect of a pool of water on stage as well as the human aspect of a dancer making ripples in the water (see Figure 6). In addition, the video images were texture-mapped onto pulsating graphical objects that were projected behind the dancer and ensemble.

In the future, we will see more technology and greatly enhanced processing power. While "faster" does not mean "better," new technology will certainly enable new artistic directions. Advances in consumer technology such as video games and animated films will influence how audiences perceive and interpret computer music with video and animation. Low resolution and uneven frame rates may have been considered "high tech" years ago, but the same technology might appear to today's audiences as intentionally "retro."

Software Architecture

Creating works for integrated media poses some difficult problems for software design and implementation. As mentioned earlier, there is a mismatch between the timing requirements for audio and video, corresponding to fundamental differences in our aural and visual perception. Video and graphics processing can slow down audio performance if the two are tightly integrated at the software level. At the other extreme, highly decoupled systems can suffer from a lack of coordination and communication between image and sound.

Timing Requirements

Let us begin by examining some requirements. Audio computation should never be delayed by more than a few milliseconds to avoid perceptible latency and jitter, whereas video and graphics can be delayed by about one frame time (15 to 40 msec) or more without noticeable problems.

The audio timing requirement stems from two sources. First, there is the desire to achieve low-latency for audio effects applied to live sound. To keep latency down, audio buffers must be short. If audio samples are not computed soon enough to refill the buffer, audible pops and clicks will result. Therefore, the deadline to complete each audio computation is bounded by the low overall audio latency. The second reason for low latency is that we can perceive jitter of only a few milliseconds in a sequence of equally spaced sounds, such as a run of fast 32nd notes.

Video is more forgiving, in part because when video deadlines are missed and a new frame is not ready, the previous frame is displayed again, at least in most modern double-buffered graphics systems. Our eyes are not very sensitive to this; in fact, when converting from film (24 fps) to NTSC video (29.97 fps), roughly every fourth frame is doubled, causing frequent timing errors of ±20 msec, yet we do not notice.

In actual practice, timing requirements depend upon the nature of the composition and performance, and many wonderful works have been created without meeting such stringent timing requirements. On the other hand, taking an engineering perspective, we do not want to preempt artistic decisions by building in limitations at the system level.

The Aura System

Aura is a software framework for creating interactive, multimedia performances (Dannenberg and Rubine 1995; Dannenberg and Brandt 1996; Dannenberg 2004a, 2004b). Aura has a number of interesting design elements that support the integration of real-time graphics, video, and music processing. The timing issues mentioned above drive much of the Aura architecture. The mixture of high- and low-latency computation calls for multiple processes and a priority-based scheduling scheme so that audio computations can preempt graphics, video, and other high-latency tasks. This naturally leads to a separation or decoupling of graphics and music processing, so one of the goals of Aura is to restore close coupling and communication without sacrificing real-time performance. Aura also looks forward to future parallel-processing systems based on multicore, multiprocessor, and distributed systems.

The key to communication in Aura is an efficient object and message-passing system, which aims to give the illusion of ordinary object-oriented programming even when objects are spread across multiple threads and even multiple address spaces (Dannenberg and Lageweg 2001). To accomplish this, objects are referenced by 64-bit, globally unique identifiers rather than memory addresses. Whereas message passing in object-oriented languages like C++ and Java amounts to a synchronous procedure call, Aura messages are true messages: a sequence of bytes containing a size, destination, timestamp, method name, and parameters. Also, Aura messages are normally asynchronous, so there is no return value, and the sender does not wait for the completion of the operation. This is important because it allows low-latency operations to continue without waiting.

One could imagine a system in which every object is executed by its own thread, allowing the scheduler to decide which objects receive priority. Every object would then have a private stack and message queue, requiring lots of memory and adding a lot of context-switching overhead. To avoid this, Aura uses relatively few threads and partitions objects by assigning each to a single thread. A thread with its collection of objects is called a *zone* in Aura, and zones run at different priority levels. Typically, there are three zones: one for hard-deadline audio processing, one for soft-deadline music control, and one for high-latency graphics, animation, video, and user-interface processing. Each zone can exist in a separate address space, but communication is faster using shared memory, so Aura implements zones as multiple threads sharing the address space of a single process. Aura also allows multiple processes to communicate via networks.

It is important that only one thread can ever access a given object. If it were possible for two threads to access the same object concurrently, then they might read data in an inconsistent state, a classic problem of concurrent systems. The standard solutions to concurrency are error-prone and therefore require great programming care. Aura avoids most concurrency problems by making each object directly accessible to only one zone and by allowing only one thread to execute within each zone.

Because of the distributed nature of Aura, there are three implementations of the message-send operation. When the message is destined for an object in another zone, the message is copied to a queue, where it is soon read and delivered to the destination object by the other zone. Message delivery between zones typically takes about 1 μsec of CPU time. When a message is destined for an object in the same zone, the message is sent immediately and synchronously to avoid copying the message. This will block the sender, but because the receiver is in the same zone, it has the same priority and real-time requirements, so it is reasonable to run it immediately. Intra-zone messages take about 0.5 μsec of CPU time. The third case occurs when the destination is in another address space. In that case, Aura delivers the message to another Aura process, typically using a local-area network. Here, the time is dominated by the overhead of the network protocol.

The programmer writes the same code regardless of the delivery mechanism, so programming concurrent distributed systems in Aura is very similar to writing ordinary single-process programs. It is easy, therefore, to send information between audio processes and graphics processes. Also, there is no hard boundary between audio and graphics processing. For example, sound-file input typically runs in the same zone as graphics owing to the potentially long latency opening or accessing a sound file. Sound-file data is transferred to the audio processing zone by means of a simple method call.

Animation in Aura

Aura offers some special objects for 3-D rendering. An OpenGL window object opens a graphics window or screen and serves as the root of a display tree. Interior nodes in the tree perform rotation, scaling, and translation in three dimensions, while leaf nodes of the tree draw objects. Because these are all Aura objects, it is easy to control their parameters by sending messages. For example, one can use MIDI faders to rotate and scale the graphical scene, or audio processing objects can send parameter updates to graphical objects (and vice versa). Aura objects can send timed messages to themselves, so it is easy schedule periodic parameter updates that in turn generate animation.

Video in Aura

Aura also has some facilities for video processing. A video-input object reads a video input frame into an array of pixels. This data can be processed to extract features for interactive control, or the image data can be mapped onto polygons using calls to OpenGL. One problem that has not been resolved is the latency of video when processing or sensing is involved. With NTSC cameras, a frame is only available every 33 msec. Assuming some processing time, and assuming that data is written to a double-buffered output, there can be almost 100 msec of latency from input to output. When video is used just for sensing, the latency is less than for processing and display, but if output parameters are passed through a smoothing filter, the result often seems to be delayed. Faster frame rates and careful synchronization may be necessary to reduce latency.

Conclusions

Interactive performance combining images and music is a natural application for computers. Because computer animation, video processing, and audio processing have all become possible in the digital domain, the computer allows for a high degree of integration and control. In the early days, computer power was lacking, resulting in low frame rates or various shortcuts to achieve some form of animation. Computers have gained enough speed to process video at the pixel level, and graphics processors now produce high-resolution, three-dimensional images with amazing flexibility.

The Aura system is designed to facilitate the integration of these new and evolving technologies by offering a flexible, distributed, real-time object system. Aura provides the "glue" to combine video, graphics, audio, and control. Aura emphasizes performance and ease-of-integration with hardware and software systems, but its open-endedness can be a problem for less-than-expert programmers. Fortunately, there are alternatives that fall at different points in the spectrum from very general to very easy to use.

Certainly, the biggest challenges ahead are artistic rather than technological. While better projection technology, greater resolution, faster computers, and lower costs may be desirable, we already have plenty of capabilities to explore right now. One of the attractions of this pursuit is that there are relatively few precedents and no established theory. Interactive visual music welcomes the creative spirit.

References

Altman, R. 1992. *Sound Theory/Sound Practice.* New York: Routledge.

Bargar, R., et al. 1994. "Model-Based Interactive Sound for an Immersive Virtual Environment." *Proceedings of the 1994 International Computer Music Conference.* San Francisco: International Computer Music Association, pp. 471–474.

Bordwell, D., and K. Thompson. 1986. "Sound in the Cinema." In *Film Art,* 2nd ed. New York: Newbery Award Records, pp. 233–260.

Burger, J. 1993. *The Desktop Multimedia Bible.* Reading, Massachusetts: Addison-Wesley.

Camurri, A., et al. 2000. "EyesWeb: Toward Gesture and Affect Recognition in Interactive Dance and Music Systems." *Computer Music Journal* 24(1):57–69.

Chion, M. 1994. *Audio-Vision: Sound on Screen,* trans. C. Gorbman. New York: Columbia University Press.

Danks, M. 1997. "Real-Time Image and Video Processing in GEM." *Proceedings of the 1997 International Computer Music Conference.* San Francisco: International Computer Music Association, pp. 220–223.

Dannenberg, R. B. 1986. "The CMU MIDI Toolkit." *Proceedings of the 1986 International Computer Music Conference.* San Francisco: International Computer Music Association, pp. 53–56.

Dannenberg, R. B. 1993. "Software Design for Interactive Multimedia Performance." *Interface* 22(3):213–228.

Dannenberg, R. B. 2004a. "Aura II: Making Real-Time Systems Safe for Music." *Proceedings of the 2004 Conference on New Interfaces for Musical Expression.* Hamamatsu, Japan: Shizuoka University of Art and Culture, pp. 132–137.

Dannenberg, R. B. 2004b. "Combining Visual and Textual Representations for Flexible Interactive Audio Signal Processing." *Proceedings of the 2004 International Computer Music Conference.* San Francisco: International Computer Music Association, pp. 240–247.

Dannenberg, R. B., et al. 2003. "Sound Synthesis from Video, Wearable Lights, and 'The Watercourse Way'." *Proceedings of the Ninth Biennial Symposium on Arts and Technology.* New London, Connecticut: Connecticut College, pp. 38–44.

Dannenberg, R. B., and E. Brandt. 1996. "A Flexible Real-Time Software Synthesis System." *Proceedings of the 1996 International Computer Music Conference.* San Francisco: International Computer Music Association, pp. 270–273.

Dannenberg, R. B., and P. v. d. Lageweg. 2001. "A System Supporting Flexible Distributed Real-Time Music Processing." *Proceedings of the 2001 International Computer Music Conference.* San Francisco: International Computer Music Association, pp. 267–270.

Dannenberg, R. B., and T. Neuendorffer. 2003. "Sound Synthesis from Real-Time Video Images." *Proceedings of the 2003 International Computer Music Conference.* San Francisco: International Computer Music Association, pp. 385–388.

Dannenberg, R. B., and D. Rubine. 1995. "Toward Modular, Portable, Real-Time Software." *Proceedings of the 1995 International Computer Music Conference.* San Francisco: International Computer Music Association, pp. 65–72.

Deutsch, D., ed. 1999. *The Psychology of Music,* 2nd ed. San Diego: Academic Press.

Gromala, D., M. Novak, and Y. Sharir. 1993. "Dancing with the Virtual Dervish." Paper presented at the Fourth

Biennial Arts and Technology Symposium. New London, Connecticut: Connecticut College. March 1993.
Juslin, P., and J. Sloboda, eds. 2001. *Music and Emotion: Theory and Research.* Oxford: Oxford University Press.
Krueger, M. W. 1983. *Artificial Reality.* Reading, Massachusetts: Addison-Wesley.
Matsuda, S., and T. Rai. 2000. "DIPS: The Real-Time Digital Image Processing Objects for Max Environment." *Proceedings of the 2000 International Computer Music Conference.* San Francisco: International Computer Music Association, pp. 284–287.
Mont-Reynaud, B. 2005. Personal communication, 6 February.
Rokeby, D. 1997. "Construction of Experience." In J. C. Dodsworth, ed. *Digital Illusion: Entertaining the Future with High Technology.* New York: ACM Press, pp. 27–48.
Rowe, R. 2001. "In Transit." In R. Rowe. *Machine Musicianship.* Cambridge, Massachusetts: MIT Press, pp. 334–343.
Woo, M., et al. 1999. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.2,* 3rd ed. Reading, Massachusetts: Addison-Wesley.