Coney Island: Combining jMax, Spat and VSS for Acoustic Integration of Spatial and Temporal Models in a Virtual Reality Installation

Robin Bargar* (rbargar@ncsa.uiuc.edu), Francois Dechelle[§] (Dechelle@ircam.fr), Insook Choi*, Alex Betts*, Camille Goudeseune*, Norbert Schnell[§], and Olivier Warusfel[§]
*University of Illinois at Urbana-Champaign, 405 N Mathews, Urbana, IL 61801
[§]IRCAM, 1 Place Igor Stravinsky, 75004 Paris, France

Abstract
We present a case study of sound production and performance that optimize the interactivity of model-based VR systems. We analyze problems for audio presentation in VR architectures and we demonstrate solutions obtained by a model-based data-driven component architecture that supports interactive scheduling. Criteria and a protocol for coupling jMax and VSS software are described. We conclude with recommendations for diagnostic tools, sound authoring middleware, and further research on sound feedback methods to support a *topology of interacting observers*.

1. The VR Audio Problem Space

We configure Virtual Reality to provide real-time interaction with geometric and temporal models. The dominant medium for displaying immersive VR is animated computer graphic images. While VR is frequently referred to as a “space” based upon 3D models, in most VR systems the primary physical space is constrained to a flat surface where an image is projected. A virtual camera view is generally a tetrahedron with its apex at the viewing position defining a viewing volume expanding symmetrically into a geometric 3-space. Stereo image computation generates depth cues by offsetting 2D images. For the most part visual “3-D” immersion results from 2D frontal image projection enhanced by interactive camera mobility. Introducing sound into a flat, frontal visual field often induces cinematic solutions: imitating space rather than simulating space. That is, fixed resonance characteristics and fixed timing of wave propagation, rather than spatio-temporal simulation. 3D audio has been touted as an important attribute of VR. However most VR systems presented at academic conferences and trade shows only provide rudimentary sound file playback, often coupled to MIDI-enabled devices. The significant limitation in this approach is the absence of information concerning an interactive simulation. Pre-recorded sounds and MIDI sequences can provide at best a rough approximation of the behavior of a simulated system. They are often observed to be mere imitations which cannot provide an accurate insight into the states of a real-time simulation. While they may confirm simple interaction, pre-determined sounds minimize the acoustic relevance of the degrees of freedom in a simulated system.

If auditory feedback can improve the user’s spatial orientation and sense of real-time interaction, then why is VR audio typically limited to triggering pre-recorded sound file or MIDI file playback? A short answer is “there are no standard alternatives.” Unlike the comprehensive hardware solutions provided by proprietary graphics subsystems, there have been few vendor-supported efforts toward sound synthesis subsystems. Historically it has proven difficult to argue for the commercial profitability of sound synthesis on general computing platforms.

There are several problem areas to establish and maintain a flexible audio development subsystem in VR. These include real-time scheduling and synchronization of sounds, graphics and control signals from interactive sensors. At a higher level of abstraction, grammars are needed for describing sound synthesis in relation to VR models and events. Traditional audio paradigms such as multi-track recording and sample-based or orchestra-score descriptions of sound production, were developed in an era when virtual experiences could only be imitated. From our experience these paradigms are not compatible with the coherent spatial and temporal models central to VR.

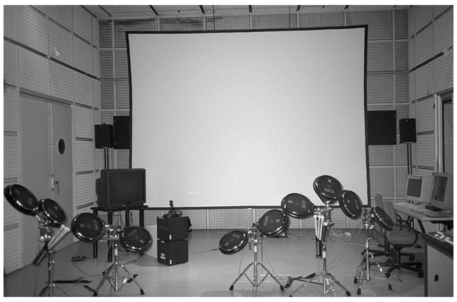
In this paper we discuss sound production and performance that optimize the interactivity of model-based virtual systems. We present a case study of a VR installation called *Coney Island*. The functional software roles of the *Coney Island* architecture include (1) a VR authoring and rendering system, (2) Sound Synthesis engines, (3) Spatial Audio DSP, (4) Sound Authoring, and (5) Scheduling and Synchronization of sounds with graphics and simulations. Software from several research centers was combined to fill these roles: (1) A VR authoring environment called ScoreGraph (Choi 1998) was used to create the *Coney Island* simulations, graphics, message passing and interactive scheduling. ScoreGraph provides the context that determines the requirements for interoperability of sound production modules. (2) Sound synthesis was provided by VSS (Bargar 1994), including synthesis engines from the STK toolkit (Cook 1995), and by jMax. (3) Distance and directional cues for sound sources were generated using Spatialisatuer (Spat) (Jot 1995) running in jMax. (4) VSS performs VR data interpretation to generate synthesis parameter control messages. (5) These messages are exchanged and scheduled in real-time by ScoreGraph and VSS. Our discussion will proceed from the VR environment to the VR software architecture, and then to sound production.

2. The *Coney Island* Scenario

Coney Island is a VR installation designed to explore and demonstrate advanced auditory display of spatial and temporal models[1]. The installation provides an interactive tour of an archipelago of mechanized islands that comprise a fantastic carnival playground. The islands are driven by simulated mechanics and particle system dynamics, as well as advanced geometric, lighting and camera models for computer graphics. For each island MIDI-enabled drum pads allow up to ten observers to play simultaneously. Simulations provide a time-critical environment where players can impart forces and see and hear the resulting mechanical actions and particle system collisions. The tour continues underwater where the players can impart force to currents that activate sound-producing clusters of floating objects. In each case equations of motion convert the forces into motions of graphical objects and in parallel into sounds.

Coney Island was designed as a case study for close coupling of audio signal processing to spatial and temporal VR paradigms. Multiple independent sound-producing events are determined by sensor data combined with simulated mechanics of rigid polygonal bodies and particle systems. Sensor, graphic and sound events must be scheduled to provide satisfactory temporal feedback. At the same time overlapping audio events must be rendered in a spatial model that allows each player the proper orientation with respect both to a view of the virtual world and a position in the real world adjacent to other players.

Figure 1: *Coney Island* setup at IRCAM



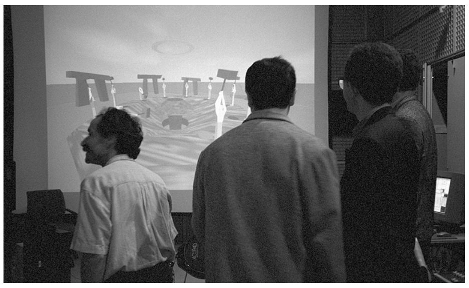
Hardware Configuration

IRCAM Studio 5 was arranged with a large-format video projection on the wall opposite its entrance. Graphics and simulations were rendered in ScoreGraph on an SGI Onyx and the image transmitted to the projector. Figure 1 shows ten MIDI-enabled drum pads positioned in the center of the installation facing the projection screen, with a solo joystick on the left and sound computation hardware on the right. Signals from players' actions were input to ScoreGraph simulations, and the resulting movement events passed to graphics and sounds. Figure 2 shows a group of players at the IRCAM installation; Figure 3 shows the players' view of a *Coney Island* scene.

The sound system consisted of three multi-channel computer sound sources, a mixer and a 4-channel diffusion system with monitors positioned in the corners of the room. Audio software performed in real-time on linux, NT and Irix platforms. Data was transmitted from ScoreGraph to VSS and from VSS to jMax using udp. Audio sources included 2-channel VSS on Linux and NT PCs, 2-channel jMax on Linux PC and 4-channel jMax on an SGI Octane.

Figure 2: IRCAM visitors interact with *Coney Island* simulations using MIDI drum pads.

Figure 3: Players' view of *Coney Island*.



3. *Coney Island*: VR architecture and graphics

Coney Island uses a software framework named ScoreGraph to organize its numerical simulations and interactive graphics, to manage input from a user interface, and to send audio control signals to VSS. ScoreGraph is a system for authoring and managing the presentation of interactive, real-time graphics and sound applications. ScoreGraph provides a scheduler and libraries for data computation and multi-threaded communication. A ScoreGraph application consists of reusable software modules written in C++ and a script that specifies the configuration and behavior of those modules at runtime. Application components are roughly divided into input devices, computational models, and graphics and sound displays. Individual components, called nodes, are organized into a directed graph, the edges of which represent control signal flow. When the application is run it is organized into parallel threads that manage the execution of its nodes. The service rates of the threads are independent of each

[1] *Coney Island* was presented at IRCAM during the June 1999 *Portes Ouvertes*.

other (and, notably, independent of the graphics frame rate), and may in fact change as the real-time system evolves. *Coney Island* integrates user input from ten MIDI drum pads, physically-based mechanical simulations, and three-dimensional geometric models created with Alias|Wavefront's Maya. The application runs on a four processor Silicon Graphics Onyx 2 with an Infinite Reality 2 graphics board.

Graphics and Particle Simulations

The visual space presented in *Coney Island* includes five islands floating on top of ocean waves, each of which contains a mechanical game. The games are similar to pinball: users apply forces to move particles toward some goal. Each island consists of a hierarchical geometric model created in Maya, and a physically based particle simulation to drive the animation. The particle systems model the forces applied by the user, particle collision against other particles and against three dimensional geometry, particle mass and radius, gravity, and friction. The differential equations used to compute the physical simulations require a consistent service rate, which was set to 20 Hz. Unfortunately, the graphics frame rate is not predictable, and at a given time falls in the 12-15 Hz range, depending upon which island is being visited and how much particle activity there is. Therefore the particle simulations and OpenGL rendering code are run in distinct parallel processes.

Interactive Presentation and Large-scale Form

The *Coney Island* experience is organized as a tour of the islands, with periodic transitions underwater to tour the debris leftover by the history of gaming on the islands. The computer graphics camera travels to each region where visitors spend some time interacting. Although the overall organization and quality of the presentation is specified by the environment's designers, many of the details of the presentation, in particular the camera angles and the order and timing of events, are controlled by intelligent algorithms. During the tour, the order in which the islands are visited is chosen at random, although each island is visited only a single time. Once at an island, the system becomes sensitive to the level of user activity. If there is no immediate user input, the game will demonstrate itself by briefly running automatically. An algorithm chooses camera positions and camera editing patterns, based upon which parts of an island are active due to user input. The camera algorithm is designed to produce results that make sense cinematically and help explain the operation of the game mechanisms. After a game has been running, a new island will be visited if the amount of input dies down.

A basic feature of the ScoreGraph system is that the directed graph that organizes an application can be reconfigured as it runs. New processes can be started, existing processes may be shut down or reduced in computational load, and connections between nodes can be made or broken. In *Coney Island* this occurs each time an island is visited. The drivers for the MIDI drum pads are reconfigured to control a different mechanism. A new particle system is started and the previously running simulation is shut down. This provides a smooth scene change between processes that are essentially separate applications.

4. *Coney Island* Sound Production

Coney Island includes three classes of interactivity with sounds:

- action space performance and extended causality;
- active navigation and direct manipulation of synthesis parameters;
- passive navigation and positional influences upon auditory space in environmental dynamics.

Action space performance generates sounds from user actions synchronized to motion-based events displayed graphically. Players influence sound production by engaging with motion simulations, an application of the Generative Mechanism principle discussed by Choi (2000a). Mechanical frictions and particle collisions in the islands are applied to control STK physically-based and modal synthesis instruments, creating quasi-realistic friction and collision sounds; at the same time the data is applied to granular synthesis implemented in jMax to produce particles of speech. The palette ranging from realistic to metaphorical sounds is a compositional design applied to virtual locations and simulated mechanics. The *Coney Island* grand tour brings about transitions from realistic to metaphorical sounds, realized at the level of the sound particle. Underwater locations abandon realism in favor of granular speech assemblages determined by wave equations stirred up by percussion pad forces.

Active navigation and direct manipulation of synthesis parameters occurs in select underwater regions where a single player may use a joystick to navigate a small animated submarine. The VR camera follows automatically. The submarine is constrained to traverse floating abstract surfaces, and its position on each surface is applied to the tuning of sound synthesis parameters by mapping position to a high-dimensional parameter control space (Choi 2000b). In these regions the particles of speech may be transformed into intelligible phrases.

Passive navigation with positional influences occurs in regions where the sounds are determined by dynamics that are independent of the players' actions, while the position of the camera determines activation of the sound sources and spatialization of the resulting sounds. These sound sources are distributed in a designated region under the islands, represented visually as a field of floating historical debris. When activated by camera proximity these debris emit complete speech excerpts from historical recordings. Four parallel Spat patches in jMax simultaneously process four source positions to create distance and directional cues. The camera position activates no more than four sources simultaneously so that all sources may be scheduled in one of the four Spats.

jMax Configuration
Despite its architecture that offers interesting features for a use in a distributed application framework, the jMax environment is used most of the time as a ``standalone" application. An application as a synthesis engine integrated in a larger distributed application have been approached in a previous version of jMax, the MAX/FTS environment, with its multi-client capabilities (Dechelle 1995, Dechelle 1996). However, the use of jMax as a synthesis engine driven by data coming from another environment has not been tested prior to the implementation of the *Coney Island* installation.

The chosen implementation has been to add to jMax network communication objects, using UDP as transport protocols. The *udp* object receives on a UDP socket a stream of datas of simple types (numbers and strings), encoded with the same protocol as used in the communication between jMax's JAVA user interface and real-time server (Dechelle 1999a, Dechelle 1999b). The *udp* object outputs messages that can be processed by a usual control patch. The *udp* objects were used to receive from VSS data coming from the virtual reality processing. The formatting and scaling was then realized by patches using the standard jMax objects set, and the results were used to drive both granular synthesis and Spat. The choice of this architecture offers several advantages: portability, inter-operability, flexibility and good latency.

VSS Configuration
VSS communicates with several other real-time audio protocols, such as Midi, OpenSound Control (Wright 1997), real-time audio streaming, RAT Mbone teleconferencing (Varakliotis 1998), and Jmax. This flexibility has allowed VSS to be used for diverse integrated applications: Midi controllers driving software sound synthesis; data from scientific applications driving Midi and OpenSound Control sound generators; streaming sound and control data from a musical instrument to a Max patch running on a computer across campus, and then streaming the sound back to the instrumentalist; and in the present case, sending control signals from a virtual environment (VE) to a real-time 8-channel sound spatializer (while another VSS, also controlled by the VE, computes the sound being spatialized). All this can be done with VSS running under operating systems including several versions of Irix, Linux, and Windows.

Such interoperability is made possible by VSS's flexible internal message-passing architecture, and by its low-overhead C++ class hierarchy. C has become a *de facto* machine-independent assembly language; most packages have a low-level interface written in C or C++, and it has proven fairly straightforward to embed such interfaces in wrappers in the form of shared libraries (DSO's or DLL's), which is how all but the very core of VSS is constructed.

The connection between VSS and Jmax is built on an internet-domain socket. As far as Jmax is concerned, VSS looks like just another Jmax patch running on a remote machine. First, this "patch" in VSS is initialized to establish a communications socket with a hostname and port number, where it expects Jmax to be listening. Then it can receive commands from other parts of VSS (other "actors") which cause it to send data messages through the socket. Upon termination, it closes the socket cleanly. Part of this installation used both Jmax and VSS on the same machine. In this case VSS acted solely as controller, not synthesizer, so it did not need to access the sound hardware; Jmax therefore had its usual exclusive access to the sound hardware.

The format of the data messages sent from VSS to Jmax is an internal Jmax format. The C source code which encodes and decodes these messages into C structures was provided by IRCAM. The VSS shared library correspondingly encodes and decodes these C structures into the message format internally understood by VSS. For this project a simple interface sufficed. From the point of view of other actors in VSS, the Jmax actor was something to which they could send a command in the form of a small number of strings, integers, and floating-point numbers. This involved a certain amount of copy/pasting in the C++ code, but kept the code simple and reliable; this was a primary concern given the short amount of testing and stabilizing time we had. A more general interface would allow arbitrarily long argument lists, but would also demand a formal description of such lists instead of encoding instances on a case by case basis. We considered streaming the individual audio channels from VSS to Jmax over Ethernet, but the possibility of dropouts was a concern (eight uncompressed 44.1 KHz audio signals, a sustained rate of over 5 megabits per second, is impractical over 10base-T ethernet). Since the computers running VSS and Jmax were physically close to each other, an ADAT optical audio cable sufficed for our application.

5. Discussion

From *Coney Island* we are able to assess four areas for further development:

1. diagnostic tools for aiding the cross-platform coupling of software synthesis modules,
2. middleware for coordinating virtual environments and audio architectures,
3. multiple-user solutions for interactive environments, and
4. sound feedback for enhancing group interaction

(1) The installation demonstrates that it is feasible with current operating systems to realize a distributed real-time application combining virtual reality and complex sound processing. But once the communication architecture is established, a deep lack of analysis and debugging tools becomes obvious: the global latency, from user action to sound, cannot be measured, and no guarantee can be given on its upper bound. The use of MIDI to transmit users' actions was the source of greatest latency: an intelligent MIDI filter was required in ScoreGraph to discard redundant values when user activity increased. Without the filter when latency increased users tended

to repeat their actions as they searched for the system response, exacerbating the delay. Potential diagnostic methods are discussed by Brandt (1998).

(2) The independent Sound Authoring layer of VSS was a useful mediator for interpreting VR data and converting it into synthesis instructions for both jMax and VSS. Although VSS supports a library of synthesis engines, the Authoring architecture functions as an independent middle layer which can be applied to control alternative sound production engines once message-passing has been implemented. During *Coney Island* development the question was raised why VSS Authoring and scheduling capabilities could not be implemented as a jMax subpatch. The VSS architecture is specifically designed to separate the process of data transformation and synchronization from particular sound synthesis patches or languages. In *Coney Island* this division of labor made it possible to combine the optimal capabilities of VSS and jMax: real-time STK instruments in the former; granular synthesis and Spat in the latter. This agnostic position with respect to sound production is relevant for the further development of independent real-time synthesis engines on multiple platforms.

(3 - 4) *Coney Island* extends previous VR composition projects at NCSA by investigating strategies for multi-user participation. From a musical perspective we examine the modernist proposition to "engage the audience", anticipating wired/wireless networked and distributed participants. Technology changes the audience engagement problem into a perceptual feedback problem: if an entire audience participates in a musical performance then how do they know what they are doing? And how do they recognize the results of their actions? VR applications typically avoid shared representations by providing a single-user first-person "shooter" perspective. One person, one point of view, one control providing isolated feedback requiring minimal disambiguation. The relevance of audio decreases with the decrease of ambiguity.

Multi-player solutions are needed. We propose development of composition solutions to accommodate a *topology of interacting observers*. By sound enhancing group interaction we envision alternatives to the isolationist perspective. *Coney Island* adopts a 1:1 feedback system: each player controls a unique object (avatar); avatars share a common display space; movement constraints and camera automation ensure that players can see and hear their avatars at all times. The scalability of this approach for large audiences is questionable. Statistical methods are a candidate for further investigation Devices such as MIDI drum pads bypass the need for technical expertise at the interface. But the feedback problem remains: to encourage group interaction while displaying the relevance of interactive input from each participant.

6. References

Bargar, R., Choi, I., Das, S. and Goudeseune, C. "Model-based interactive sound for an immersive virtual environment." *Proc. Int. Computer Music Conference*, Aarhus, Denmark: International Computer Music Assn., pp. 471-474, 1994.

Brandt, E. and Dannenberg, R. "Low-latency Music Software Using Off-the-Shelf Operating Systems." *Proceedings of the ICMC98*. San Francisco, 1998. ICMA. pp. 137-41.

Choi, I. "ScoreGraph: dynamically activated connectivity among parallel processes for interactive computer music performance." *Proceedings of the ICMC98,* San Francisco, 1998. ICMA. pp. 527-535.

Choi, I. "Gestural Primitives and the context for computational processing in an interactive performance system." In *Trends in Gestural Control of Music.* Battier and Wanderly, eds., IRCAM, Paris. 2000.

Choi, I. "A Manifold Interface for Kinesthetic Notation in High-Dimensional Systems." In *Trends in Gestural Control of Music.* Battier and Wanderly, eds., IRCAM, IRCAM, Paris. 2000.

Cook, P. "A Hierarchical System for Controlling Synthesis by Physical Modeling." In *Proceedings of the 1995 International Computer Music Conference*. San Francisco, 1995. ICMA.

Dechelle, F. and De Cecco, M. "The *Ircam* real-time platform and applications." In *Proceedings of the 1995 International Computer Music Conference.* San Francisco, 1995. ICMA

Dechelle, F., De Cecco, M., Maggi, E.,. and Schnell, N. "New {DSP} applications on *FTS*." In *Proceedings of the 1996 International Computer Music Conference*, San Francisco, 1996. ICMA.

Dechelle, F., Borghesi, R., De Cecco, M., Maggi, E., Rovan, B. and Schnell, N. " *jMax*: an environment for real-time musical applications." *Computer Music Journal*, 23(3):50--58, 1999.

Dechelle, F., De Cecco, M., Maggi, E.,. and Schnell, N. " *jMax*: recent developments." In *Proceedings of the 1999 International Computer Music Conference,* San Francisco, 1999. ICMA

Jot, J-M. and Warusfel, O. "A real-time spatial sound processor for music and virtual reality applications." *Proceedings of the ICMC,* San Francisco, 1995. ICMA. pp. 294-95.

Varakliotis, S., Hodson, O., and Hardman, V. "A software platform for multi-way audio distribution over the Internet." In *_Audio and music technology: the challenge of creative DSP_IEE Colloquium,* London, November 1998.

Wright, M. and Freed, A. "OpenSound Control: A New Protocol for Communicating with Sound Synthesizers." *Proceedings of the 1997 ICMC,* Thessaloniki, 1997. ICMA. pp. 101-104.