Q3OSC OR: HOW I LEARNED TO STOP WORRYING AND LOVE THE ~~BOMB~~ GAME

Robert Hamilton
Center for Computer Research in Music and Acoustics (CCRMA)
Department of Music
Stanford University
rob@ccrma.stanford.edu
http://ccrma.stanford.edu/∼rob

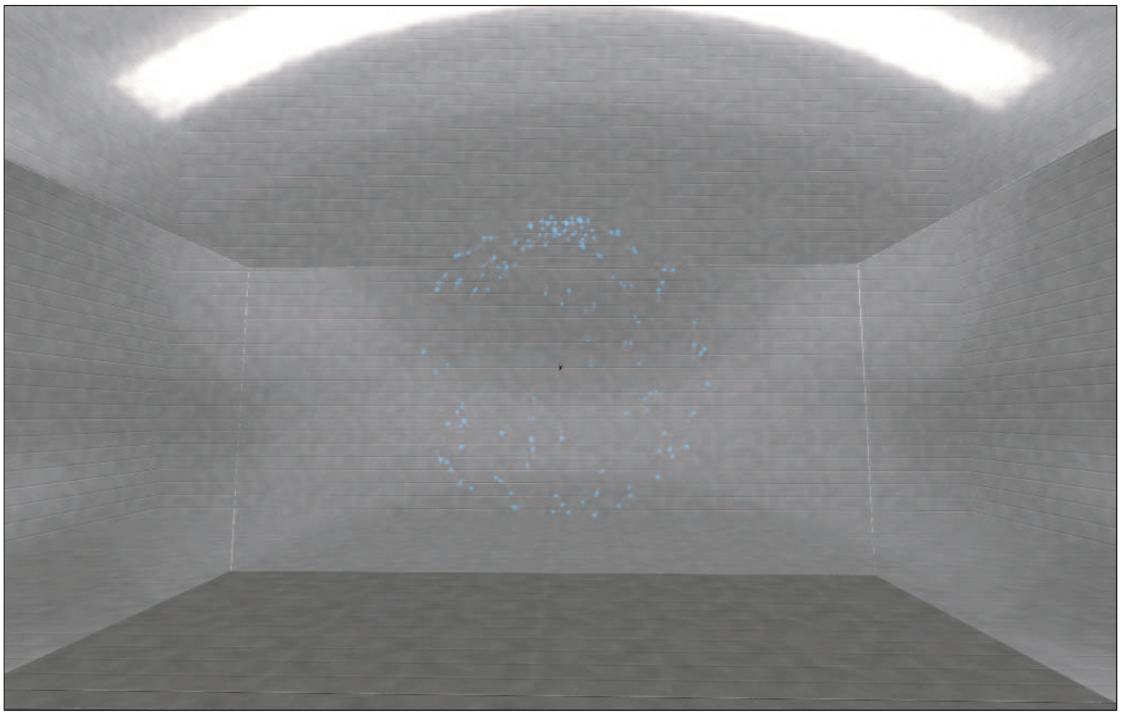
Figure 0. A single q3osc performer surrounded by a sphere of rotating particles assigned homing behaviors, each constantly reporting coordinates to an external sound-server with OSC.

ABSTRACT

q3osc is a heavily modified version of the open-sourced ioquake3 gaming engine featuring an integrated Oscpack implementation of Open Sound Control for bi-directional communication between a game server and one or more external audio servers. By combining ioquake3's internal physics engine and robust multiplayer network code with a full-featured OSC packet manipulation library, the virtual actions and motions of game clients and previously one-dimensional in-game weapon projectiles can be repurposed as independent and behavior-driven OSC emitting sound-objects for real-time networked performance and spatialization within a multi-channel audio environment. This paper details the technical and aesthetic decisions made during the development and initial implementations of q3osc and introduces specific mapping and spatialization paradigms currently in use for sonification.

1. INTRODUCTION

Virtual environments in popular three-dimensional video games can offer game-players an immersive multimedia experience within which performative aspects of game-play can and often do rival the most enactive and expressive gestural attributes of instrumental musical performance. Control systems for moving client avatars through rendered graphic environments (commonly designed for joysticks, gamepads, computer keyboards and mice or any such combination) place a large number of essentially precomposed in-game functions, gestures and movements at a gamer's fingertips, available for improvisatory use. With the advent of easily-accessible fast wired and wireless networks, the focus of computer-based game-play has shifted from predominantly solitary modalities of play towards a peer-oriented and communal gaming experience.

Dynamic relationships between game-users and their peers - both virtual and physical - have shifted in kind, bolstered by massive online communities and expansive game-worlds designed and developed with multi-user collaborative or combative play in mind. Much like traditional collaborative musical performance, many networked interactive game environments allow participants to affect the experiences of other participants in (hopefully) positive or (potentially) negative manners.

q3osc is a musical and visual performance environment which makes use of these paradigms of communal networked conflict and collaboration to re-purpose virtual entities and traditional game control methodologies into agents and actions capable of affecting musical response in physical auditory space. By extending the source-code of a multi-user video game engine to include Open Sound Control [25] input and output libraries, data from within a game-engine's virtual environment can be encapsulated and transmitted to one or more external sound-servers with the intent of sonifying virtual gesture and motion into a musically rich and satisfying aural experience. In turn, data generated by external performative controllers, potentially extracted from the auditory environment or generated using algorithmic processes can be routed back into the virtual environment as control data for the modification of environment parameters and the control of individual game entities.

Custom modifications made to traditional game elements such as the targeting methods of "weapon" projectiles and global world-level variables such as gravity and avatar speed allow for striking and dynamic change in the visual and performative characteristics of both client behaviors and the virtual environment itself. By empowering game-players with the ability through gesture and motion to generate complex evolving visual and musical structures, relatively simple modifications in pre-existing game code can have profound and far-reaching musical and performative effects.

In its initial realization, q3osc has been used to build interactive networked multi-user performance environments for the Stanford Laptop Orchestra (SLOrk) [20], making use of a set of 20 networked laptops, each projecting sound from a six-channel hemispherical speaker. By correlating the placement of physical laptop stations in a performance space with the placement of representative speaker avatars in virtual space, an immersive spatialized performance environment can be created, where performers move freely through virtual space, projecting sound-events across the ensemble's dispersed speaker field.

2. BACKGROUND

q3osc is based on the open-sourced code-base of the wildly successful *Quake 3 Arena* by id Software [9], one of the most commercially successful and widely played FPS or "First-Person Shooter" video games of modern times. In *Quake 3 Arena*, game players control a futuristic gladiator-like warrior fighting with other players connected over the internet in three-dimensionally rendered graphic environments. Initially released as a commercial product in December of 1999, *Quake 3 Arena* used a client-server architecture with a significant amount of server-side prediction to allow players from across the world to join together in fast-moving virtual combat environments which themselves could be created and modified by members of the gaming community. Following continued research by id Software into more realistic and technically-advanced gaming engines, the Quake 3 source-code was released under the GNU GPL in August 2005, providing game-developers and "modders" an invaluable tool for the creation of new gaming platforms and virtual environments of all shapes and sizes. Development of the open-source Quake 3 engine has continued under a community-driven effort know as ioquake3 [10].

3. RELATED WORK

q3osc's use of visual environments both as representative models of physical acoustic spaces and also as virtual interfaces for the control and display of musical data has been directly influenced by works from both the gaming and computer-music communities. Muench and Furukawa's two-dimensional *small_fish* [4] makes use of bouncing projectiles and their subsequent collisions with virtual entities as midi-triggering controllers for the dynamic creation of musical patterns and forms. Oliver and Pickles' ioquake3/Pure-Data (PD) [16] modification *q3apd* [15] made use of PD's string-based FUDI protocol to export game parameters to a Pure-Data sound-server. The real-time visualizations of ChucK [21] processes displayed by Wang and Cook's *Audicle* [24], as well as game-based Audicle interfaces like *ChucK-ChucK Rocket* respectively use three-dimensional graphical environments and features extracted from game-play to both represent and sonify virtual actions. In addition, this author's ioquake3-based work *maps and legends* [5] made use of q3apd and Pure Data to build virtual compositional maps within which game-players' motions triggered sonic events and controlled spatialization within an eight-channel environment.

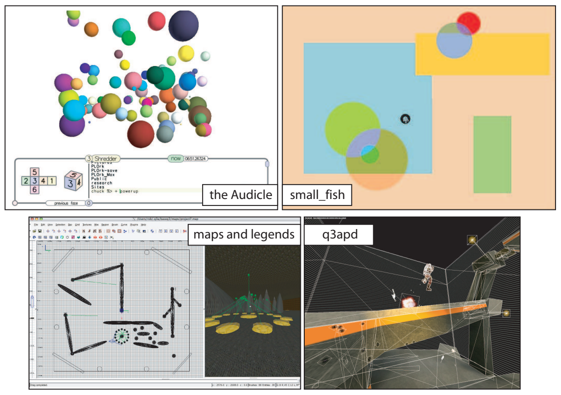

Figure 1. Interactive graphic environments used as generative systems for musical output.

Figure 2. Networked ensemble projects including performers in local as well as remote locations.

q3osc's goal of creating multi-user networked musical environments draws inspiration from a rich history of the use of WAN and LAN computer networks for collaborative performance. Early networked performances by The Hub [26] stand out as rich examples of the complex musical constructs formed through communal composition, improvisation and performance. Stanford's SoundWIRE group [2] and their networked concert performances with the ensembles Tintinnabulate at RPI (NY) and VistaMuse at UCSD, as well as with performers at Beijing University in 2008's "Pacific Rim of Wire" concert [8], utilizes multiple channels of uncompressed streaming audio over its JackTrip software to superimpose performance ensembles and spaces alike. And both the Princeton Soundlab's Princeton Laptop Orchestra (PLOrk) [19] as well as the SLOrk have displayed the powerful possibilities of collaborative networked compositional form using distributed point-source spatialization.

4. SYSTEM OVERVIEW

q3osc consists of a heavily modified or "modded" version of the open-source ioquake3 project which tracks various in-game actions and attributes and exports their values to one or more OSC client sound-servers for sonification and spatialization. As local or remote clients running the standard and freely-downloadable open-source ioquake3 software connect to a game-server via LAN or WAN network connections and move their avatars through the fully-rendered three-dimensional environment, each client's local software communicates its position and any commands received from the player's control system to the game-server for coordination with other connected clients. Data extracted from the game-server representing client actions and motions, as well as coordinates representing the paths of fired projectiles is output using OSC to one or more sound-servers. While the game-server itself runs highly customized code currently compiled only for Linux, one great advantage of q3osc is that any standard ioquake3 client running any operating system can connect to the server without any additional software other than the currently loaded map data file. q3osc output can be sent to a single multi-channel sound-server (where spatialization across a speaker array is handled in software) or to a series of distributed sound-servers, such as found in a laptop orchestra, each connected to a single point-source speaker. Sound servers used with q3osc can be written in any interactive musical programming language that supports the input and output of OSC messages or bundles including but not limited to ChucK, SuperCollider [14], Max/MSP [13] and PD. Currently under development is an additional layer of communication in which one or multiple sound-servers output OSC messages back into the game environment to control or modify entity, client or environment behaviors.

5. IOQUAKE3 MODIFICATIONS

The task of transforming a game-platform such as ioquake3 from a virtual arena for indiscriminate "deathmatch"-style combat into an abstract musical playground began with elements of FPS gameplay which could be seen as corollaries to musical or spatial gesture: in particular the movements of game-players and the projectiles they fired. In q3osc, the major efforts of game-code modification have focused on support for the output and input of OSC messages and bundles, modified behaviors and controls to weapon projectiles, and the abilities of game-clients to affect system-state and environment parameters.

5.1. Open Sound Control Integration

q3osc makes use of the *Oscpack* [1] C++ implementation of Open Sound Control by compiling the Oscpack source-code directly into the ioquake3 C-based project and calling OSC output methods from a number of game-server classes to track changes in position and state for game-clients and entities as well as changes to the game-environment itself. To facilitate linking between Oscpack's C++ classes and ioquake3's C classes, exposed methods in Oscpack are wrapped in "extern C{}" statements to allow the entire project to be compiled using a modified version of the ioquake3 Makefile calling the gcc compiler. q3osc currently features OSC output of client and entity positions and states, support for multiple OSC output destinations (both multiple IP addresses and Ports), and a limited use of OSC input to control entity and environment characteristics.

5.1.1. Osc Output

In the ioquake3 client-server model, there exist methods called for each instantiated individual game-client and moving entity, triggered by the game server for each rendered frame to calculate and adjust client or entity motion vectors given commands made by the client or calculated by the server. By passing data to a custom OSC bundle and message formatting class, a stream of data representing constantly changing client or entity position or state can be

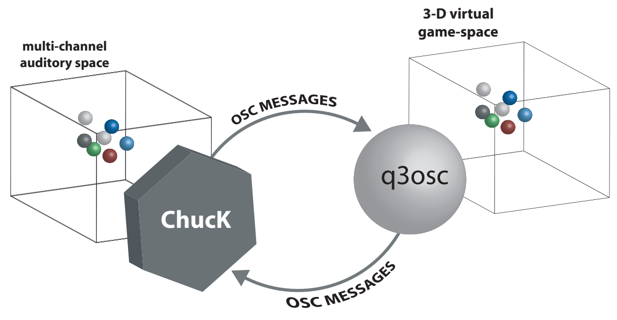

Figure 3. Bi-directional communication between game-server and sound-server allows for close correlation between elements in both the virtual/visual environment and the physical/auditory environments.

formatted either as OSC bundles or messages and routed to one or more OSC clients listening on designated IP and Port addresses. As ioquake3 makes use of persistant data structures to store information about client-state, a number of client-state parameters (such as client-view angle, three-dimensional velocity or selected weapon) are exposed and can be output using OSC as well.

To facilitate the reading of q3osc OSC output by any standard OSC client, the q3osc output can be toggled between OSC bundle and message formats using in-game console flags. OSC output tracking an individual plasma-bolt's current owner or the client which fired the projectile (/ownernum). unique projectile id (/projectilenum), X, Y, and Z coordinate position (/origin), whether this message signifies a bounce-event (/bounce), and whether this message signifies the destruction of this particular game-entity (/explode) is represented below, first as an OSC bundle, and second as a single-line OSC message.

```
[ /classname "plasma"
  /ownernum 0
  /projectilenum 84
  /origin 102.030594 2550.875000 -2333.863281
  /bounce 1
  /explode 0 ]

/projectile "plasma" 0 84 102.030594 2550.875000
 -2333.863281 1 0
```

5.1.2. *Multiple OSC Output Streams*

The ability to stream q3osc's output OSC data to multiple sound-servers affords users the choice to expand beyond single sound-server architectures, allowing both the use of distributed client sound-server environments as used with the Stanford Laptop Orchestra as well as the ability to load-balance the potentially large number and rapid-rate of OSC streams across multiple parallel or redundant sound servers. q3osc allows developers the choice of routing individual client data streams to one or more specified servers as well as the choice of routing all osc data to all specified IP addresses; at this time q3osc has been run successfully with as many as twenty distributed sound servers across local networks. In a mapping schema where each projectile entity fired in virtual space can result in the dynamic instantiation of a potentially complex and processor-intensive sound-object spatialized across an array of real-world speaker locations, load-balancing greatly decreases the possibility of audio drop-outs, distortions or clicks generated by an overloaded audio-server. In this manner, multiple sound-servers running different audio softwares can be utilized together with little-to-no added complexity. Additionally, the ability to route all OSC output to any number of IP addresses allows geographically distributed clients across over WAN networks to each simultaneously replicate a shared audio environment in multiple locations.

5.1.3. *Osc Input*

One extremely powerful feature of OSC integration into the game-engine currently under development is the ability to use external OSC sending-servers to control in-game entity motions and environment states by sending control messages back into the game-engine. By allowing an external sound-server to communicate changes in the real-world auditory environment to the game engine, a more complex synergy between environments can be achieved. For example, by creating an inverse correlation between amplitude generated by a real-world sound-server and the speed at which projectiles can move in the virtual-world, a strong sense of relation is formed between virtual and

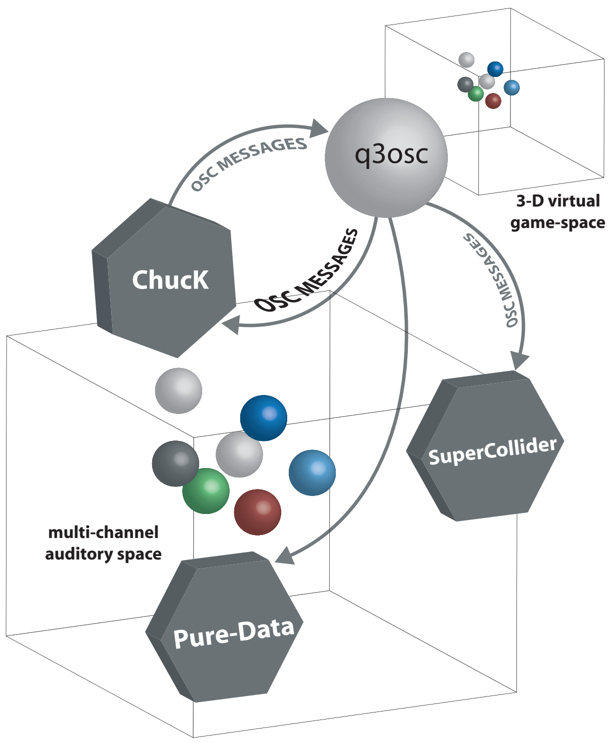

Figure 4. Multiple OSC output streams allow for the simultaneous use of multiple sound-servers utilizing various software languages such as ChucK, PD, SuperCollider or Max/MSP. Q3osc can receive OSC messages on multiple ports allowing for multi-sourced OSC input as well.

physical space. In a more complex example, algorithmic patterns of motion controlling X, Y, and Z coordinates for in-game projectiles can be used to implement independent swarming or flocking behaviors or other strongly patterned group behaviors and causal relationships [3].

5.2. Projectile Behaviors and Controls

The use of in-game weapon projectiles as controllable or behaviored entities, each generating their own individual OSC output stream of data representing coordinate position, velocity and other attributes has been achieved by modifying existing code within the game-engine as well as through the creation of several new methods for effecting physics-based changes on entity behaviors. In this manner, not only does a weapon projectile like the "plasma-bolt" - a visually striking blue ball of energy - output its current X, Y, and Z coordinates as OSC messages, but such projectiles also exhibit behaviors such as the ability to bounce off walls (instead of being destroyed by contact with environment surfaces) or the ability to be attracted to individual game-clients as "homing" projectiles.

While in the standard ioquake3 code-base certain server parameters such as the amount of global gravity or a client's top running speed could be modified by individual game-clients during gameplay using in an in-game "console" window (similar to a Linux/Unix "Terminal" window"), modifications to the lower-level behavioral patterns of projectiles were not accessible during game-play. Weapons fired by individual clients would be assigned a direction vector (with an origin at the client's current position) and a pre-determined velocity for the selected weapon type. The game-server would be sent a message that the weapon had been fired by a particular client and would trace a vector moving at the defined velocity until either a collision occurred - where an entity such as a wall or another game-client was detected to be "hit" - or a pre-determined amount of time expired, causing the entity to destroy itself. These static behaviors and settings were hard-coded into the game engine and could not be modified on-the-fly.

In Figure 5, screen captures of q3osc display a number of modified entity behaviors. In examples I and II, large rotating spherical masses of projectile entities are created by enabling behavior flags which cause all subsequently generated projectiles to a) persist indefinitely, b) bounce from any contact with walls, floors or other parts of the environment itself, and c) constantly update their directional vectors to track any client entity within a given radius. By changing environment variables which control both this radius of attraction and the speed at which the projectiles move, projectiles are easily coaxed into orbits around their targets of attraction. Additionally, by toggling projectile homing behaviors on or off, these dynamic spheres can be made to expand, contract and collapse (as seen in example III). A linear path of homing particles can be seen in example IV, where particles fired by one game client track the position of another game client.

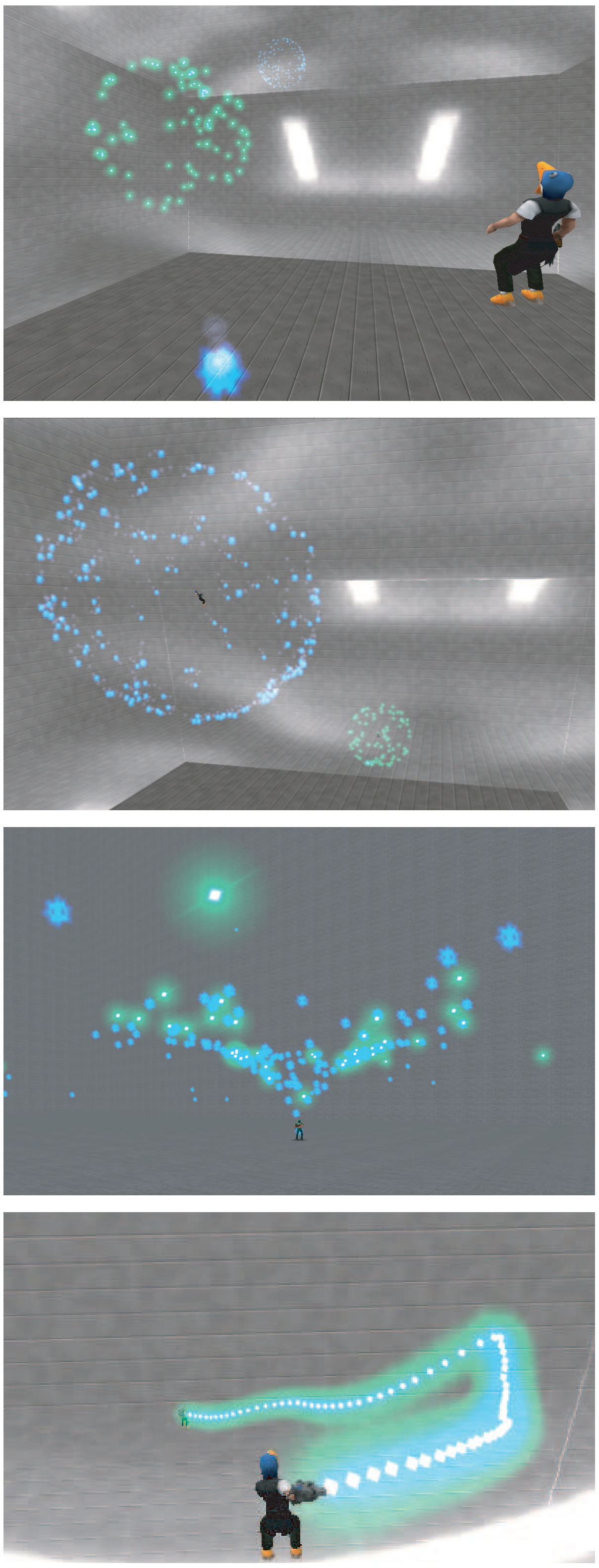
Figure 5. In q3osc, complex patterns of entities can be formed and controlled by performers within the game environment: I) game-client observes orbiting/homing projectiles; II) clients surrounded by orbiting/homing projectiles; III) projectiles collapsing on a stationary client; IV) client-to-client homing projectiles.

6. SPACE AND SPATIALIZATION

In standard modes of single-user or networked multi-user gameplay, the focus of both the visual and auditory representations presented to each user has traditionally been wholly user-centric. The user in his or her real-world seat is presented with an illusory visual and sonic representation of the virtual environment complete with signal processings designed to strengthen the related illusions of space, distance and motion. As game-users most commonly listen to game audio output through headphones, stereo speakers, or an industry-standardized multi-channel configuration such as 5.1 or 8.1, all audio processing done in game-engines tends to attempt to create realistic illusions of motion for one user sitting in the sound-system's centralized "sweet-spot". Such individualistic presentation by its very nature restricts the communal sensory experience fostered in the virtual environment from existing anywhere except within the game-world itself. By inverting these traditional models of sound-presentation and by focusing on a space-centric model of sound projection for game-environments, a communal listening experience can be fostered inclusive of all listeners within a shared physical space, including game-users and audience members alike.

6.1. Single-Server Configuration

In initial realizations of ioquake3-based sound works such as 2006's *maps and legends*, objects within a virtual game-environment were spatialized across an 8-channel horizontal sound-field. Eight speakers surrounding audience and performers alike were mapped to virtual speaker locations within the game-environment and entity positions in virtual space were spatialized across the sound field by correlating simple distance measures from entity to virtual speaker with speaker amplitude levels. In this manner, data sent from q3osc can be sonified using a sound-server connected to a multi-channel sound system, spatializing user and projectile motions across two or three-dimensional soundfields. Demonstration installations of the project running in this kind of configuration have been presented using CCRMA's sixteen-channel spherical sound-space located in the CCRMA Listening Room [11], as well as more standard 8-channel configurations in the CCRMA Stage and studios. The use of spatialization techniques such as Ambisonics [12] or VBAP [17], creating more realistic auditory models to correspond with variously sized and shaped virtual environments, can be implemented in a number of OSC compatible audio processing languages.

6.2. Multiple-Server Configurations

The flexibility of q3osc's outbound OSC routing scheme allows for a number of sound-servers to be run simultaneously. For computationally-intensive works, multiple sound servers can be addressed, effectively load-balancing the sonification of virtual environments populated with large numbers of sound-generating entities. For systems with exceptionally large numbers of individual audio output channels (such as the 48-channel multi-tiered Soundlab [18] in Belfast's Queen's University Sonic Arts Research Centre (SARC)), the addressing of sets of speakers can be distributed across a number of sound-servers. In this manner, q3osc enables the scaling of technical architectures to accommodate the use of larger speaker arrays and more complicated environmental correlations.

In conjunction with the SLOrk, q3osc has been utilized in a multiple speaker configuration which has seen as many as twenty laptops connected to hemispherical speaker-arrays acting as spatializing sound-servers for the actions in virtual space of as many as twenty performers. OSC messages generated by q3osc were sent over a wireless 802.11n local network to each laptop running a game client and an instance of a ChucK audio server. In this configuration, the physical distribution of the speaker arrays - hemispherical radiating point sources - was correlated to the position of specific virtual locations in the game-environment.

7. SONIFICATION AND MAPPING OF BEHAVIOR AND GESTURE

While the aim of q3osc is not to define a particular mapping scheme for sonification of game-entity generated data but rather to facilitate the dialog between virtual action and analog sounding-gesture, basic example sonifications and data mappings such as the bouncing of projectile entities and their continuous motion in space should prove useful models for more complex future sonifications.

7.1. Projectile Bounces

A direct and effective mapping technique currently used with q3osc is the sonification of projectile bounces, or collisions between projectile entities and environment surfaces. At the point of contact between a given projectile and environment surface, the projectile bounces off the surface, setting and transmitting a "bounce" OSC flag. To best accommodate a massive number of bouncing projectiles, an extremely simple sonification running on a ChucK sound-server of an impulse routed through a resonant filter has proven to be extremely efficient and robust. By mapping coordinate data for a single plane - say the X axis - received in the same OSC message as the bounce flag to a desired musical scale, each bounce event can be made to trigger simple notes. As a possible example of additional non-spatial bounce mappings, speed of the projectile can be mapped to velocity or amplitude of each bounce-triggered note-event, Y-axis coordinate data can be mapped to note-duration and Z-axis coordinate data can be mapped to length of decay on the reverb. At any such point of impact, data points such as angle-of-incidence, distance-traveled, and duration of existence all are easily extracted.

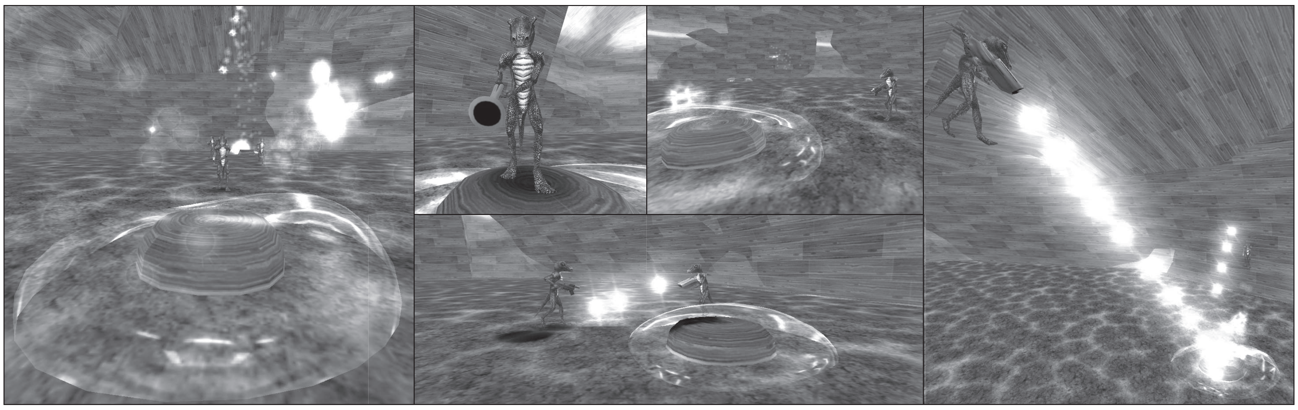

Figure 6. Screen captures from a performance of *nous sommes tous Fernando...* show performers firing sound projectiles and speaker locations as hemispherical constructs.

7.2. Continuous Projectile Output

As coordinate data for all projectiles in the game-engine are transmitted continuously over OSC, sonification of projectile motion across coordinate space can be introduced in addition to each projectile's discrete bounce events. Towards this end, individual functions can be instantiated (or "sporked" in the ChucK language) for each projectile, spatializing the projectile's motion across speakers within the same array or with careful osc analysis, across distributed speaker arrays. Development of various sonification for continuous projectile output is currently ongoing both using ChucK as well as Supercollider-based sound-servers.

8. *NOUS SOMMES TOUS FERNANDO...* (2008)

Premiered in May, 2008 by the Stanford Laptop Orchestra, the ensemble piece *nous sommes tous Fernando...* [7] is a flexible performance environment comprised of a series of richly textured performance maps for laptop ensembles of various sizes ranging from 5 virtual speaker locations to twenty. In its most basic form, the map for *nous sommes tous Fernando...* utilizes a series of five hemispherical-shaped objects placed on the floor in a cross-pattern in virtual space. Laptop-stations and connected hemispheres are placed in an analagous pattern in the performance space with a subwoofer connected to the center speaker and performers sitting at each point of the cross. Audience members sitting around and within the speaker configuration watch the output from one laptop connected to a video projector, acting as a virtual camera.

In SLOrk performances, OSC messages carrying position data for projectiles fired by each performer are sent from the Linux game server to each of the Mac OS X client machines. The bounces of in-game projectiles are then sonified across the multi-hemisphere soundfield in ChucK, correlating bounce positions in the virtual environment with scaled amplitudes across the physical speaker layout using a simple distance function. Performers sit at each speaker station with additional performers as desired connecting via client machines not necessarily connected to sound-servers. Frequency and resonance coefficients for a dynamically allocated array of tuned filters are mapped to additional distance and coordinate data, creating a dynamically shifting sound world controlled by the projectile firings of in-game performers..

Performers in *nous sommes tous Fernando...* are visualized as giant green lizards, moving through an environment where all surfaces save the floor are made up of blocks of random angles and sizes. In this manner, any projectile bouncing off the walls or ceiling will be reflected in a completely unpredictable manner, minimizing the chances of simple periodic or infinitely repetitive note sequences. Each performer is allowed to affect changes on a given set of environmental parameters such as each client's speed of motion, the gravity of the environment and the speed of fired projectiles. Additionally all performers can control whether all projectiles in the map persist or whether they are destroyed. And while *nous sommes tous Fernando...* is at heart an improvisatory work, the virtual camera operator more often than not plays the role of conductor, typing messages to performers in the game-engine while attempting to shape the ensemble performance into a coherent form.

9. CONCLUSIONS

Virtual game-environments repurposed as extensible multi-user networked performance environments offer performers and audiences alike rich platforms for musical expression. The Quake III engine's history as a successful commercial gaming platform affords users a massive pre-existant code base, a strongly user-supported and open-sourced community-driven development environment and toolset, and low financial and temporal barriers to entry. By integrating OSC support into the game-engine itself, q3osc virtual environments can interface with and drive any number of commercial and custom computer-based music and media systems. In this manner, the possibility exists for the use of q3osc as the core of any virtual networked multimedia performance system, ranging from the performance of electroacoustic music to distributed network opera or theatre. Towards this end, the source code for q3osc as well as a detailed history of its development and a media archive of its uses is maintained and made available for download on the CCRMA Wiki site [6].

10. REFERENCES

[1] Bencina, R., Oscpack, 2006. URL http://www.audiomulch.com/ rossb/code/oscpack/.

[2] Chafe, C., S. Wilson, R. Leistikow, D. Chisholm, G. Scavone. "A Simplified Approach to High Quality Music and Sound Over IP," In *Proceedings of the COSTG6 Conference on Digital Audio Effects (DAFx-00)*, Verona, 2000.

[3] Davis, T. and O. Karamanlis. "Gestural Control of Sonic Swarms: Composing with Grouped Sound Objects." In the Proceedings of the 4th Sound and Music Computing Conference.

[4] Furukawa, K., M. Fujihata, and W. Muench, http://hosting.zkm.de/wmuench/small_fish.

[5] Hamilton, R., "maps and legends: FPS-Based Interfaces for Composition and Immersive Performance" In *Proceedings of the International Computer Music Conference.*, Copenhagen, Denmark, 2007.

[6] Hamilton, R., q3osc Wiki Page, 2008. URL https://cm-wiki.stanford.edu/wiki/Q3osc.

[7] Hamilton, R., *nous sommes tous Fernando...*, q3osc Wiki Works Page, 2008. URL https://cm-wiki.stanford.edu/wiki/Q3osc:_works.

[8] Hamilton, R., The Stanford Pan-Asian Music Festival: China on Stage, 2008. URL http://panasianmusicfestival.stanford.edu.

[9] id Software, 2008. URL http://www.idsoftware.com.

[10] ioquake3 Project Page, 2008. URL http://www.ioquake3.org.

[11] Lopez-Lezcano, F. and C. Wilkerson. "CCRMA Studio Report" In *Proceedings of the International Computer Music Conference.*, Copenhagen, Denmark, 2007.

[12] Mahlam, D. and A, Myatt. "3-D Sound Spatialization using Ambisonic Techniques" Computer Music Journal, 19;4, pp 58-70, Winter 1995.

[13] Cycling '74, "Max/MSP", 2008. URL http://www.cycling74.com.

[14] McCarthy, J. SuperCollider, 2008. URL http://supercollider.sourceforge.net.

[15] Oliver, J., *q3apd*, 2008. URL http://www.selectparks.net/archive/q3apd.htm.

[16] Puckette, M., 1996. "Pure Data." In *Proceedings of the International Computer Music Conference.* San Francisco, 1996, pp. 269-272.

[17] Pulkki.V., "Virtual sound source positioning using vector based amplitude panning." Journal of the Audio Engineering Society, 45(6), June 1997, 456-466.

[18] SARC Soundlab, 2008. URL http://www.sarc.qub.ac.uk.

[19] Trueman, D., P. R. Cook, S. Smallwood, and G. Wang. "PLOrk: Princeton Laptop Orchestra, Year 1" In *Proceedings of the 2006 International Computer Music Conference (ICMC)*, New Orleans, U.S., November 2006.

[20] Wang, G. The Stanford Laptop Orchestra, 2008. URL http://slork.stanford.edu.

[21] Wang, G. *The ChucK Audio Programming Language: A Strongly-timed and On-the-fly Environ/mentality.* PhD Thesis, Princeton University, 2008.

[22] Wang, G. A. Misra, and P.R. Cook. "Building Collaborative Graphical interFaces in the Audicle" In *Proceedings of the International Conference on New Interfaces for Musical Expression.*, Paris, France, 2006.

[23] Wang, G. and P. R. Cook. "ChucK: A Concurrant and On-the-fly Audio Programming Language" In *Proceedings of the International Computer Music Conference.*, Singapore, 2003.

[24] Wang, G. and P. R. Cook. "The Audicle: A Context-sensitive, On-the-fly Audio Programming Environ/mentality" In *Proceedings of the International Computer Music Conference.*, Miami, USA, 2004.

[25] Wright, M. and A. Freed. "Open Sound Control: A New Protocol for Communicating with Sound Synthesizers" In *Proceedings of the International Computer Music Conference.*, Thessaloniki, Greece, 1997.

[26] Lancaster, S., "The Aesthetics and History of the Hub: The Effects of Changing Technology on Network Computer Music", Leonardo Music Journal, Vol. 8, pp. 39-44, 1998.