μ MAX-UNITY3D INTEROPERABILITY TOOLKIT

Ivica Ico Bukvic
Virginia Tech
Music, DISIS, CCTAD
ico@vt.edu

Ji-Sun Kim
Virginia Tech
CS, CHCI, DISIS, CCTAD
hideaway@vt.edu

ABSTRACT

Object-oriented rapid prototyping tools geared towards multimedia, such as Max/MSP/Jitter and Pd/Gem, serve as a powerful foundation for efficient multimodal cross-pollination and integration. Although both Max and Pd support OpenGL, the lack of scalability, user-friendly 3D editor, and physics engine makes them less desirable solutions for the rapid development of complex environments and physics simulations. Unity3D is a powerful rapid 3D video game prototyping platform with an integrated physics engine. Its audio capabilities, however, are limited mainly to triggering and spatialization of audio buffers. μ is a toolkit offering easy integration of Max/PD with Unity3D, allowing for exchange of control data, as well as importing of dynamic Jitter textures into Unity3D. The former makes it particularly suitable for efficient sonification of physics simulations. μ has been utilized in *Elemental*, an interactive communal soundscape installation (part of the Revo:oveR exhibit) allowing for visitors' motion (Max) to drive a physics engine (Unity3D) and sonify ensuing data across a 12-channel ceiling-mounted speaker array (Max).

1. INTRODUCTION

The increase in processing power and affordability of computing devices has ushered a revolution of rapid software prototyping tools. Through the use of a library of shortcuts tailored towards a specific set of tasks, such development environments offers unprecedented efficiency. In the audio domain, arguably even the earliest tools, such as the Music-N languages [7], have been designed with the rapid prototyping concept in mind. More recently, a number of the existing software, such as CMix [9] and Csound [13] has been retrofitted to offer real-time capabilities [6]. They have been further complemented by visual programming environments, such as Pure Data (Pd) [11] and Max [5].

One of the main challenges associated with the rapid prototyping tools is that their efficiency quickly dissolves once a task at hand delves beyond their primary focus (e.g. an audio project that utilizes visuals). Over the years we've seen two approaches to solving this predicament: software such as Max and Pd embracing new functionality (Jitter and Gem respectively) to expand their applicable domain, and software [3] or libraries [15] offering interoperability between two or more complementing prototyping environments. Both approaches come with a set of advantages and limitations: while inclusion of Jitter in Max has paved way towards efficient integration of audio and visuals in artistic and research contexts [4,14], its implementation of OpenGL layer in particular lacks key components of contemporary 3D engines, such as scalability, support for animated content, user-friendly 3D scene editing, and an efficient, easy to use physics engine. In our recent research, the physics engine had proven a particularly desirable component for a number of audio-oriented scenarios. Equally, tools designed to bridge different rapid prototyping environments commonly require more complex setups (e.g. two applications running concurrently on the same machine) whose synchronization and load balancing at times can be difficult to achieve, thus resulting in a greater number of possible points of failure in a production environment.

Unity3D [12] is a powerful cross-platform gaming engine integrated into a rapid prototyping environment. Its versatility as a visual tool is however hampered by the use of a relatively simple audio engine limited to triggering and spatialization of preloaded audio buffers. While working on a series of prototypes dealing with sonification of moderately complex 3D physics simulations, we identified a need for an efficient way to complement Max with Unity3D and consequently generate a composite rapid prototyping environment capable of efficiently tackling said scenarios. This interest was further warranted by the fact that Unity3D, like Max, is capable of generating Mac and Windows standalone executables, thus minimizing potential production-level overhead.

2. INTRODUCING μ

μ is a Max-Unity3D interoperability toolkit consisting of C/C# code (Unity3D) and Max patches offering near seamless integration between the two environments using TCP/IP protocol. The 1.00 release relies upon the `netsend/netreceive` [10] third-party Max externals for bi-directional messaging and the `jit.net.send` Max object for relaying texture data. μ offers following features:

Max→Unity3D

- Built-in functions using Max message formats, offering Unity3D object search and basic manipulation (move, rotate, and scale; relative (lowercase) and absolute (uppercase)). For instance, `cube1 r 0 10 0` seeks object named cube1 and rotates it 10 degrees around the y-axis.
- Custom object-specific functions (provided as placeholders for custom code), such as changing object color, transparency, animation, physical properties, etc.
- Importing texture data as a 4-plane char matrix (ARGB with 8 bits per color channel) and applying it to a Unity3D object (Fig.1).

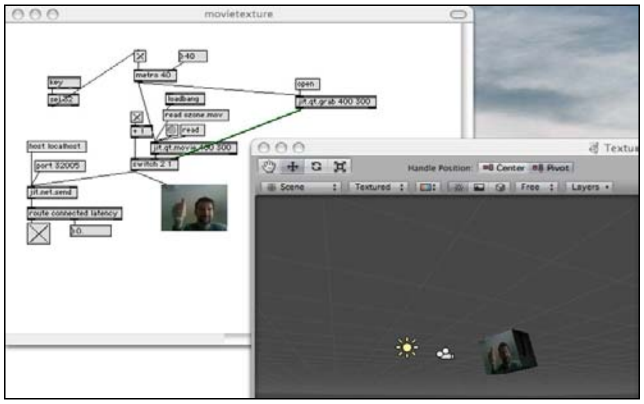

Figure 1. Max texture imported into Unity3D via μ.

Unity3D→Max

- Custom object-specific data (provided as placeholders for custom code), such as state, position, speed, etc. This is particularly useful in 3D physics simulations where object properties can be fed back into the audio engine.

By coupling Max with Unity3D's powerful API μ aims to broaden rapid prototyping domain of both tools to include a wide array scenarios, including physical simulations (objects' movement and interaction with other assets and surfaces can be accompanied with sound cues spatialized across an array of speakers), complementing aural with complex 3D content (audio-visual installations), games requiring advanced DSP features of Max and/or where visuals are driven by audio (e.g. Audiosurf [1]), and projects dealing with the cross-pollination of rich dynamic aural and visual content.

3. PERFORMANCE

Given that one of μ's primary goals was to complement Max with Unity3D's ability to efficiently manipulate large number of assets and more importantly do so while ensuring that all messages will arrive at their destination, we settled for the TCP protocol. The UDP approach proved too unwieldy for this purpose, requiring that each network packet contain the state of all assets in the scene to minimize sync issues associated with potential network packet loss. Otherwise, critical one-time triggers contained within a single UDP packet (e.g. a climactic moment in an interactive installation driven by a single networked command) could be entirely lost. From a texture streaming perspective, considering that the `jit.net.send` object supports only TCP protocol, such choice also circumvented a need to design custom Max external. Thus, even with a greater CPU and network overhead the TCP approach proved a simpler solution, ensuring that each packet would contain only the information of assets whose states needed change (Max→Unity3D) or had just changed (Unity3D→Max). The fact that the μ utilizes network protocol has also allowed for the system to run on two or more computers without requiring their co-location.

In order to assess the CPU overhead induced by μ we compared average CPU utilization of both Max and Unity3D at runtime with μ communication enabled and suspended. No manual load balancing was employed and since both clients were configured to run on a single CPU core, no OS-specific multi-core optimizations had any observable impact on their CPU load distribution. All tests were run on a G5 with dual Intel Xeon 2.67GHz.

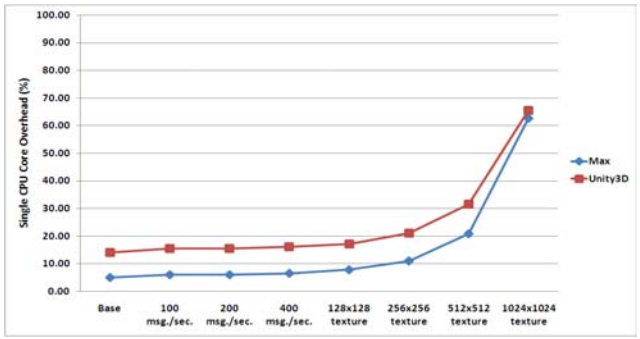

Figure 2. μ's CPU overhead for Max and Unity3D.

The bi-directional communication of control data is formatted as `netsend/netreceive` space-delimited character array that is parsed into object names (String), one-letter commands, integers, floats, and/or other relevant data formats as defined in the custom commands. At 400 messages per second the measured CPU overhead of both applications combined rarely exceeded 10% (Fig.2).

From a performance perspective, perhaps one of the most interesting aspects of μ is its ability to relay texture data in Jitter matrix format to Unity3D. An example included with the toolkit offers mapping processed video and/or webcam texture data onto a Unity3D asset, in this case a cube (Fig.1). Although uncompressed texture data is bandwidth intensive, tests with either both clients on the same machine or on two separate machines on a local area network using Gigabit Ethernet have shown stable operation for textures up to 1024x1024 at 25 frames per second albeit with a considerable CPU load (50+% per application). 256x256 textures had a considerably lower CPU footprint at 6-7% per application (Fig.2) with no noticeable deterioration in image quality, making it an adequate solution for common scenarios.

4. ELEMENTAL

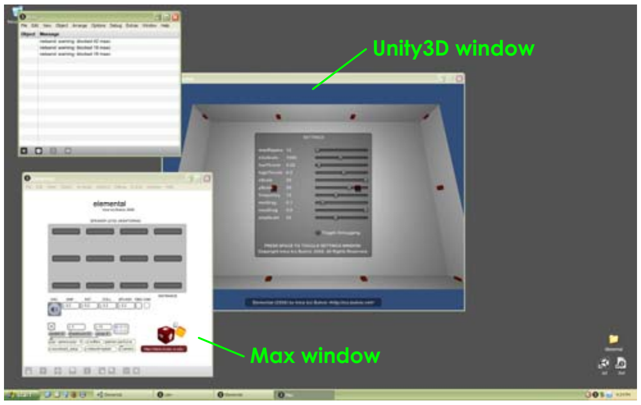

Figure 3. Screenshot of the *Elemental* tech setup.

μ's first real-world test was its integration in the *Elemental* (Fig.3) interactive communal soundscape installation (part of the Revo:oveR exhibit) that opened in November 2008 at the new Taubman Museum of Art in Roanoke, Virginia (USA). The piece uses series of infrared (IR) LED floodlights in conjunction with a homebrew ceiling-mounted IR webcam to concurrently monitor motion of every visitor populating the exhibit space. The resulting data is fed into a physics engine and used to spawn 3D spheres within a virtual rendition of the exhibit space, in locations corresponding to visitor's real-world position. Based on the visitors' motion spheres are assigned speed, trajectory, and inertia parameters and are set in motion resembling simplified particle simulation of water ripples. These meta-particles behave in many ways as their real-world counterparts, reflecting from the walls and other obstacles, as well as colliding with each other until they exhaust all of their kinetic energy. The sphere position and collision events are then forwarded to an audio engine that sonifies them across a 12-channel 4x3 ceiling-mounted speaker array. The sphere position is marked by a sound of a water ripple, its reflection against the walls by a water splash, while its interaction with other spheres is accompanied by an "aural fireworks" consisting of an algorithmically-generated consonant swarm of pitches with pizzicato-like envelopes. The resulting interactive soundscape paints a picture of communal dynamics in the exhibit space marking points where trajectories of individuals may already have or are about to converge.

Given the task at hand, μ proved an ideal solution for cross-pollinating motion tracking via webcam (Max), using visitor trajectories to run a physics simulation through an efficient 3D engine (Unity3D), and finally sonifying the resulting data across the 12-channel 4x3 ceiling-mounted speaker array (Max). While Unity3D's 3D capabilities were secondary to the task, the fact we could easily visualize and therefore troubleshoot the system's behaviour had proven instrumental in fostering time-efficient development. The installation was optimized to run on a single Intel Core 2 Duo 2.67GHz machine non-stop for the duration of the exhibit (approximately 5 months). Apart from a few occurrences of seemingly random errors we traced down to the `netsend/netreceive` external objects, the installation has been running for several months without interruption.

5. LIMITATIONS

The CPU overhead for large textures is currently one of μ's greatest bottlenecks and in this respect μ is unable to scale well, particularly in situations where multiple dynamic video streams are to be used within the same 3D scene. Alternatives include shared memory model that would circumvent the TCP/IP stack overhead however requiring that both clients run on the same system, and/or the development of a framework for relaying a compressed video stream, thus retaining the ability to interface across the network.

As any bridging toolkit, μ is susceptible to fluctuations and peculiarities of environments it interfaces with. In our testing we've identified a number of inconsistencies, many of which were traced down to the video card driver issues.

Considering that Windows video drivers provide much better support of Direct3D than OpenGL, Unity3D on Windows platform by default relies upon the Direct3D. Yet, the Unity3D's C plug-in architecture as of version 2.1 does not provide means of exposing texture data pointers to the DirectX library, making it difficult to update game engine texture with networked data through optimized plug-ins. Possible workarounds for Windows users include running Unity3D in OpenGL mode and thus dealing with possible visual artefacts due to inadequate driver support, or manipulating texture updates using Unity3D's API at the expense of noticeably higher CPU overhead.

On the Mac side, we've identified Unity3D inconsistencies associated with different video card manufacturers. NVIDIA hardware using C-based OpenGL plug-in requires 1:1 ratio textures that must be of size other than 64x64 (or multiples thereof). The latter would cause Unity3D to consistently crash. Ironically, ATI setup would accept textures of varying ratios as long as both dimensions were power of 2 (e.g. 256x256 or 64x32). The obvious workaround for both is to utilize texture sizes that conform to the said parameters.

While both Max and Unity3D allow for easy building of standalone executables, the use of μ does not alleviate the production environment from a concurrent use of two applications. This however can be advantageous at times, particularly when the resulting simulation is too demanding to be run on a single computer.

6. CONCLUSION AND FUTURE WORK

Elemental has in many ways served as the test of μ's ability to provide a stable and for the most part scalable way of integrating Max and Unity3D into a composite rapid prototyping tool. Lessons learned through this production cycle have given us a clear roadmap of what μ needs to address in its next iteration.

Having encountered instabilities with the `netsend/netreceive` third-party Max externals we are looking into ways to either patch said objects or alternately integrate support for the `jit.net.send`-formatted messaging. The latter would also alleviate the need to rely upon third-party externals, thus lowering the number of potential points of failure. In order to provide better scalability of importing dynamic textures into Unity3D we are currently looking into shared memory models that would circumvent the TCP/IP protocol and consequently lower CPU overhead, as well as compressed video streaming frameworks for a more efficient communication over network. Finally, while the existing messaging model is fully compatible with the Pd's `netsend/netreceive` objects, the ability to import Pd textures is currently lacking. We will therefore look into support of Pd's Gem library textures as well as interfacing with open-source alternatives to Unity3D, such as Blender3D [2] game engine. From an artistic perspective we look forward to exploring μ's potential in performance and online collaboration contexts.

7. OBTAINING μ

μ is a GPL-licensed [8] open source toolkit freely downloadable from http://disis.music.vt.edu.

8. REFERENCES

[1] Audiosurf. "Audiosurf: Ride Your Music", http://www.audio-surf.com/. Last accessed Feb. 2009.

[2] Blender. "blender.org", http://www.blender.org. Last accessed Feb. 2009.

[3] Bukvic, I. "RTMix – towards a standardized interactive electroacoustic art performance interface", *Organised Sound*, 7(3), 2002, pp. 275-286.

[4] Bukvic, I., Gracanin, D. and Quek, F. "Investigating Artistic Potential of the Dream Interface: The Aural Painting", in *Proceedings of the International Computer Music Conference*, Belfast, Northern Ireland, 2008.

[5] Cycling'74. "Max/msp: A graphical programming environment for music, audio, and multimedia", http://www.cycling74.com/products/maxmsp. Last accessed Feb. 2009.

[6] Garton B. G. and Topper, D. "RTcmix - Using CMIX in Real Time", in *Proceedings of the International Computer Music Conference*, Thessaloniki, Greece, 1997.

[7] Gerzso, A. "Paradigms and Computer Music", *Leonardo Music Journal*, 2(1), 1992, pp. 73-79.

[8] GNU General Public License. "The GNU General Public License – GNU Project – Free Software Foundation (FSF)", http://www.gnu.org/licenses/gpl.html. Last accessed Feb. 2009.

[9] Lansky, P. "The Architecture and Musical Logic of Cmix", in *Proceedings of the International Computer Music Conference*, Glasgow, Scotland, 1990, pp. 91-93.

[10] Matthes, O. "netsend and netreceive", http://www.akustische-kunst.org/maxmsp/. Last accessed Feb. 2009.

[11] PD. "Pure Data", http://puredata.info/. Last accessed Feb. 2009.

[12] Unity. "Unity3D", http://unity3d.com/. Last accessed Feb. 2009.

[13] Vercoe, B. and Ellis, D. "Real-Time CSOUND: Software synthesis with Sensing and Control", in *Proceedings of the International Computer Music Conference,* Glasgow, Scotland, 1990, pp. 209-211.

[14] Wakefield, G., Overholt, D., Putnam, L., Smith, W., Novak, M. & Kuchera-Morin, J. "The Allobrain: An Interactive, Stereographic, 3D Audio Immersive Environment", in *Presentation at the CHI Workshop on Sonic Interface Design*, Florence, Italy, April 2008.

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