BLENDERCAVE: A MULTIMODAL SCENE GRAPH EDITOR FOR VIRTUAL REALITY

David Poirier-Quinot

EADS Astrium Services, Suresnes, France
ETIS/ENSEA - Univ. Cergy-Pontoise - CNRS
LIMSI-CNRS, Orsay, France
david.poirier-quinot@limsi.fr

Damien Touraine & Brian F.G. Katz

LIMSI-CNRS, Orsay, France

damien.touraine@limsi.fr
brian.katz@limsi.fr

ABSTRACT

This paper presents the BlenderCAVE project, which extends the 3D creation content software Blender and its Game Engine (BGE) to Virtual Reality (VR) applications. Based on a multi-screen non-stereoscopic adaptation of the BGE [Gascon et al., 2010], Blender-CAVE now integrates a complete framework dedicated to Virtual Reality (VR), compatible with the three main Operating Systems for any given VR architecture configuration. It has been developed by audio and VR researchers with support from the Blender Community on LIMSI's state of the art VR platforms. Acting as a Scene Graph, BlenderCAVE handles multi-screen/multi-user tracked stereoscopic rendering through an efficient low-level master/slave synchronization process while controlling spatial audio rendering (ambisonic, multi-user binaural, WFS, etc.) and haptic events through OSC and VRPN protocols. The scene creation process itself is reduced to simple Blender manipulations including basic python programming easily carried out using standard laptops. OSC client and spatial audio rendering methods have thus far been implemented in the Max/MSP Audio Programming Environment.

1. INTRODUCTION, SCENE GRAPH EDITORS FOR RESEARCH IN VR

Research commonly exploits Virtual Reality (VR) for two kind of applications: fast development for *contextualized* proof of concepts and situation simulation, typically requiring full perceptive environments. In the latter, the more *realistic* Virtual Environments offer a better sense of presence that often leads to more relevant results and feedback.

Over the last decade, realism was often associated with graphics. Led by animation and video game industries the scientific VR community acquired numerous ready to use 3D creation content softwares that would latter provide multi-screen tracked stereoscopic rendering solutions. Amidst the set of non-public licensed softwares, Virtools [1] and Quest3D [2] stand out as references, being tailored to VR content creation and real-time exploration. Worth noting, Unity Technologies lately released a plug-in [3] to run the Unity3D software [4] on any VR platform, providing at the same time its immense game designer network and inexhaustible *Asset Store* to VR research community. Along with Eon Reality [5] and Worldviz [6] solutions (Eon Icube and Vizard respectively), commercial solutions are nowadays available to support researchers in their VR based investigations, on the understanding that they are disposed to invest financially in such, typical installations having yearly maintenance fees most often calculated on a per screen basis. This limits development flexibility and mutli-platform/multi-configuration utilization. The relatively closed nature of these solutions also limits their use in research oriented applications.

Hybrid solutions like CaveUT [7], a Pittsburgh University genuine adaptation of the UnrealTournament Game Engine [8] for VR architectures combine affordability and stable developments, with drawbacks such as platform compatibility and limited content creation capabilities.

As for public licensed software, developers proposed solutions to adapt open source software like OpenSceneGraph [9] to VR architectures [10], or created their own from scratch [11, 12, 13, 14], resulting in functional yet seldom supported solutions in addition to low-level coding knowledge requirements for creating rich and interactive VR scene graphs. For a more exhaustive study on 3D creation content softwares for VR research see [15, 16].

Lately, improvements regarding VR technologies however extended realism requirements to other modalities [17, 18], amongst which audio and haptic emerged as obvious immersion enablers. Here-above reported commercial solutions generally support cross-software communication protocols such as VRPN [19] and OSC [20] to delegate these modalities, or handle them through dedicated internal solutions, the plug-in or software counterpart usually adding to the general system expense. Several research teams developed scene graph editors interconnecting graphic and sound/haptic libraries [21, 22, 23] yet these are often tailored to specific research fields or architectures. Thus far, no open source scene graph editor went as far as to be accepted as a durable solution for multi-modal VR scenes development, due usually to the lack of maintenance on the open source hand and the scarce yet demanding research community on the non-public licensed one.

While an optimal VR Suite does not exist yet, one can list many requirements for such, requirements that we obviously tried to achieve in the BlenderCAVE project presented in this paper. Above all, it would have to support the three main Operating Systems exploited in VR architectures (Linux, Windows, and Mac OS). Its main development shall be based on programming languages or software with active communities, potentially open source, where independent communities would ensure ready to use SDK and future developments. Within this scope, what is needed is a main scene graph editor coupled with expendable audio and haptic front ends, where each could be independently extended or removed as need be. It would also have to enable non-programmers to create simple scene graphs, offer straightforward scene portability between architectures, and allow external developers to implement new features as need be.

This paper presents the current state of development of the BlenderCAVE project, first presented in [24] under the project title BlenderCAVE3D-s. Section 2 presents the open source BlenderCAVE VR scene graph editor issued from the Blender software. Section 3 details the Max/MSP based Sound Rendering Engine, handling both Ambisonic and multi-user binaural renderings. Section 4 exposes concrete developments of the BlenderCAVE Scene Graph Editor on two different VR architectures while Section 5 evokes future BlenderCAVE related projects.

2. BLENDERCAVE, A VR SCENE GRAPH EDITOR

Blender is a multi-platform open source 3D creation content software [25] with enough functionalities to create photorealistic pictures, high quality animations and, most of all, video games. Blender based games exploit a real-time rendering engine called the Blender Game Engine (BGE) which handles a multitude of physical interactions through the implemented Bullet Physics Library [26] while general game logic may be defined through blocks and/or embedded python scripts. Gathering users for more than a decade, Blender now boasts a large support community, a dedicated professional network [27], and several scene repositories where one may find plenty of reusable material.

Based on these observations, a Spanish research team sought to use Blender on a VR architecture, which led to the first functional instance of BlenderCAVE in 2010 [28]. Not yet addressing tracked adaptive stereoscopic rendering nor sound spatialization, this version nonetheless permitted one to instantly produce any Blender scene onto a 4-wall CAVE (Cave Augmented Virtual Environment) architecture, considerably easing a usually strenuous VR content creation process. A year latter, our working group composed of VR and acoustic researchers decided to extend their work to a complete scene graph editor that would eventually support and unify future VR projects, focusing on multi-platform and multi-architecture interoperability.

BlenderCAVE is a patched version of the original Blender software, where internal routines have been added to handle the three basic functions needed in standard CAVE VR applications:

1. Master/Slave synchronization
2. Adaptive stereoscopic rendering
3. External message processing

A basic architecture graph is shown in Figure 1. BlenderCAVE is launched with a shell command (through any of the graphical cluster nodes) while architecture related parameters (e.g. eye separation, screen dimensions and positions, VPRN server address, etc.) are stored in a shared xml file. An instance of the BGE is then executed on all the rendering nodes, external message processing and mapping into scene graph events being conducted through the Master only.

The core BlenderCAVE modifications to the BGE trunk consist in the addition of a `prerender` method prior to the basic `predraw`, the ability to redefine the projection matrix, and the possibility to redefine the aspect-ratio from a python script (allowing easier portability of scenes between architectures). While not developed further in this paper, it is worth noting that BlenderCAVE routines, as well as most of the BlenderCAVE patched modifications have been implemented so as to be as much transparent as possible regarding BGE native processes in the hope that they will be integrated in the official Blender trunk.

2.1. Master/Slave synchronization

Master/Slave synchronization is carried out at each frame, in the `prerender` method. Synchronization, executed via a python script, inspects every object in the scene to see if it has changed. If so, the update information is passed form the Master to the Slave nodes before rendering. Such communications are based on the UDP protocol. The *statelessness* of *broadcasted* messages from Master to Slave usually results in the use of UDP as a standard in VR architectures rather than its TCP counterpart, due to the additional overhead and reception acknowledgments in TCP which hinder real-time rendering. The TCP protocol is however yet used for synchronization acknowledgment messages in the actual BlenderCAVE version. By default, BlenderCAVE synchronizes every object in the scene amongst the rendering nodes. Future versions will support synchronization white and black lists, with forced synchronization without checks or no synchronization or check for specific objects, thereby reducing unnecessary calculation overhead in complex scene rendering.

2.2. Adaptive stereoscopic rendering

Multi-user adaptive stereoscopic rendering mainly consists in changes of coordinate systems and projection operations. The resulting modification in the projection matrix is necessary to take into account tracked users, who are more often than not away from the center of the screen and thereby require non-symmetric projection matrices, updated to take into account the users current head position, orientation, and eye separation. Such variable frustum projections have been implemented so as to be computed locally on each graphical node before rendering in the `prerender` process.

2.3. External message processing

While stereoscopic rendering and synchronization are supposedly basic VR features, message exchange with externals represents a cornerstone in any scene graph editor as it largely impacts end users in their scene developments. BlenderCAVE to external (and conversely) user defined interactions are conventionally implemented and collected in a python script attached to the VR scene named *[sceneName].processor.py* (see Figure 1), to easily differentiate BlenderCAVE from Blender related logic implementations (such as objects displacements, collision triggered events, etc.). Most of the current incoming messages are dedicated to user integration or interaction in the VR scene, from Cartesian position/orientation coordinates to controllers various states and usually resort to VRPN protocol in our architecture. For such, BlenderCAVE implements the *Processor* python Class (different from here above evoked processor.py script) which gathers methods to ease VRPN related message processing, granting simple access to controllers and user parameters on the understanding that they be formerly instantiated in the previously mentioned xml file.

As for outgoing messages, most are currently intended to add sound to the scene, i.e. update scene graph object properties in the Sound Rendering Engine (SRE) using the well known Open Sound Control (OSC) protocol. The SRE here refers to the system gathering audio object definitions (sound source, position, etc.) and sound rendering methods in an Audio Programming Environment (e.g. Max/MSP, PureData, etc.) plus eventual loudspeaker/headset

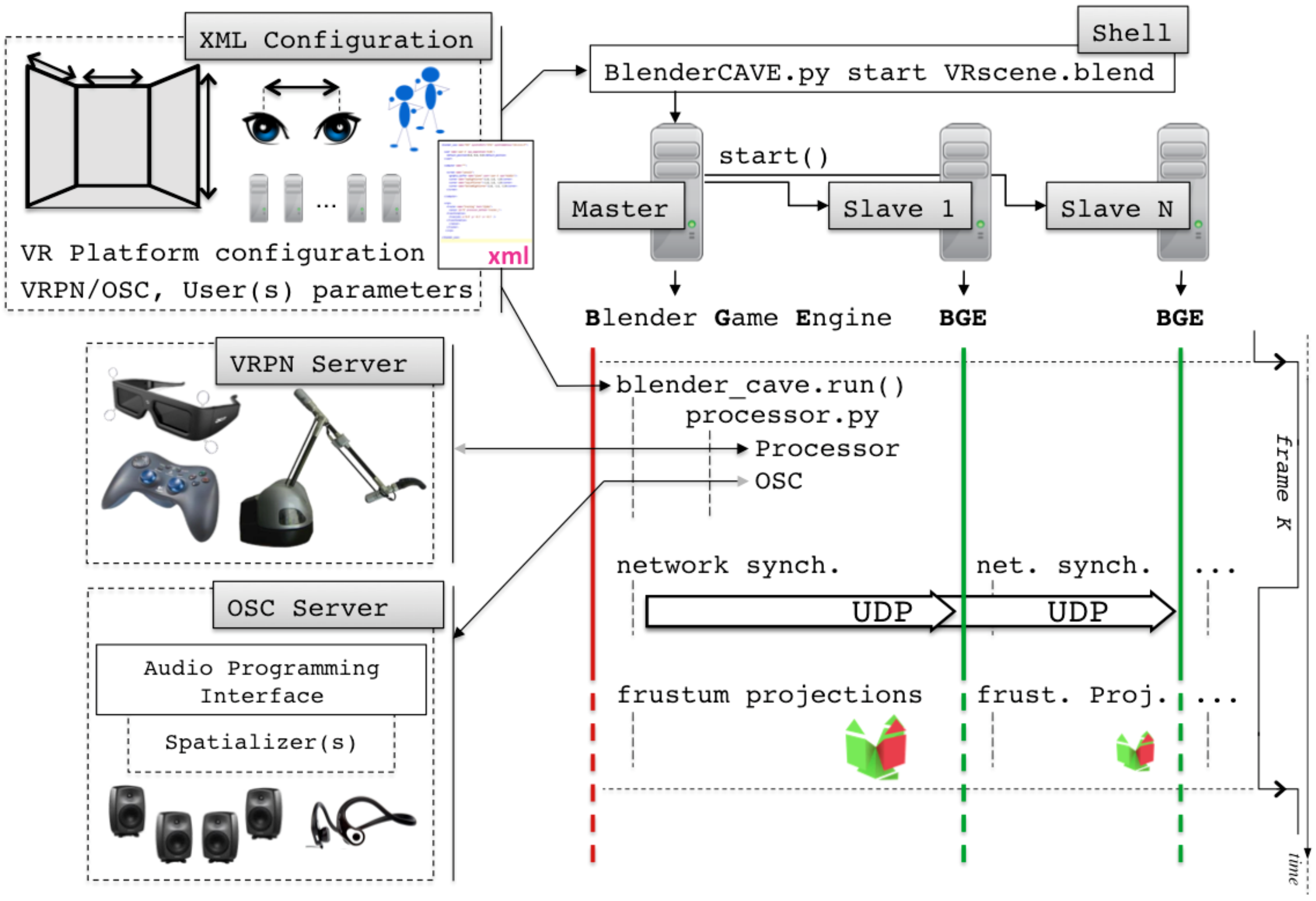


Figure 1: BlenderCAVE architecture graph. The right-hand side exposes the 3 main processes involved in a BlenderCAVE run, whose execution has been integrated in the BGE frame based sequencer. The *network synchronization* process maintains consistency between screens controlled by master/slave rendering nodes while the *frustum projection* handles tracked adaptive stereoscopic rendering related calculations. When required, OSC and VRPN communication related implementations are by convention stored in the *processor.py* python script. Grey arrows indicate features under development (e.g. haptic feedback for sensorimotor exploration is not yet available).

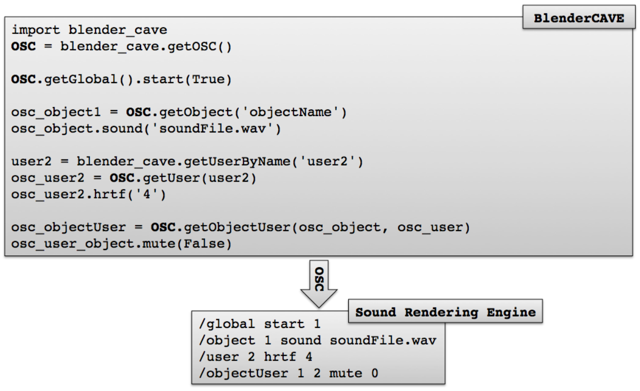
outputs. Here BlenderCAVE proposes an OSC module that encapsulates three main Classes of messages: Global, Object, and User.

Global messages are used in the SRE to set general properties like main volume, room characteristics (reverb volume, warmth, etc.) and architecture specific configurations. A Global message can for example be used to select the spatialization engine and its associated audio output mapping corresponding to a given experiment on a specific rendering architecture. During the course of scene development on a laptop or an architecture that differs from the final high performance rendering system, one could for instance use a Global message to load a basic binaural spatialization engine in the SRE before porting the scene to the real VR architecture where the corresponding message can designate a more sophisticated audio rendering method. With regards to the SRE, an Object is a simple sound emitter that requires messages to define its nature (simple sound file, microphone input, etc.), position, orientation, etc. The User encompasses individuals interacting with the scene graph as well as their associated spatialization engines. The first User could thus be defined at a static position, such as the center of the CAVE associated with an Ambisonic rendering, while second and third could be real users tracked as they explore the scene equipped with headphones and receiving associated individual binaural renderings. User messages are designed to update individual positions for adaptive binaural rendering and set individual parameters like volume or Head Related Transfer Function.

A fourth class of messages has been included that manages the routing of Objects to multiple spatialization engines (i.e. Users), such that certain Objects can be rendered for selected Users if desired, and is thus named the *ObjectUser* Class. These messages define which engine is to handle a given set of objects in the scene graph. Messages from this class are used to tailor Users' sound fields by individualizing perceived sound sources. Figure 2 illustrates a basic Blender to SRE OSC configuration using methods of the BlenderCAVE OSC module.

3. BLENDERCAVE ASSOCIATED SOUND RENDERING ENGINE, EXAMPLE USING MAX/MSP

We chose to develop BlenderCAVE SRE as an external application, i.e. not directly included in the main framework as would



```
import blender_cave
OSC = blender_cave.getOSC()

OSC.getGlobal().start(True)

osc_object1 = OSC.getObject('objectName')
osc_object.sound('soundFile.wav')

user2 = blender_cave.getUserByName('user2')
osc_user2 = OSC.getUser(user2)
osc_user2.hrtf('4')

osc_objectUser = OSC.getObjectUser(osc_object, osc_user)
osc_user_object.mute(False)
```

```
/global start 1
/object 1 sound soundFile.wav
/user 2 hrtf 4
/objectUser 1 2 mute 0
```

Figure 2: Example of a python implementation using the Blender-CAVE OSC module (invoked at line 2, the *blender_cave.getOSC()* command) to send 4 OSC messages and associated Sound Rendering Engine received instructions.

have been an openAL based implementation. Benefits are twofold, offering the flexibility to run both renderers (audio and visual) on different architectures to distribute CPU usage, and allowing for the use of more accomplished audio software providing needed flexibility and functionality for research applications. While expendable (meaning that BlenderCAVE functions without it), the SRE plays a non-negligible role in the scene graph general rendering process, as does the choice of its developing environment in BlenderCAVE's further evolutions. Basic specifications are to natively support OSC protocol [29] and be sufficiently *Object-Oriented* to handle dynamic Object instantiation and attribute sharing amongst spatialization engines.

There is no lack of Audio Programming Environments (APE) that fulfill these requirements; SuperCollider, Open Sound World, PureData, Max/MSP, Csound, ChuncK, CLAM, etc. The latter has already been used as a BGE coupled SRE implementing room acoustics simulation [30]. The auralization process was however clearly dedicated to a specific scene, wrapped around its very logic in embedded scripts. More complete solutions have been designed by researchers at Wollongong and McGill Universities, the former coupling Max/MSP and a Java3D scene graph into a 3D audio-visual interface [31] while the latter designed AudioScape, a modular Pure Data library paired to the OpenSceneGraph editor as a graphical front end [32]. While the Wollonglong project was clearly not designed as a VR scene development toolkit, object motion being handled through xml scripts or command line instructions, the AudioScape solution now proposed as audioTWIST [33] could clearly support future BlenderCAVE developments.

While solutions like SuperCollider, Csound, or the VR oriented Virtual Sound Server propose programming environments and mechanisms closer to BlenderCAVE's, APEs like Pure Data and Max/MSP offer powerful signal processing and sound design features through real-time interactions in user-friendly GUIs. In order to maintain compatibility with parallel research efforts we chose to develop our SRE using Max/MSP. We note however that an alternate SRE could be developed without modification to the BlenderCAVE architecture as long as it respects the established OSC nomenclature.

The two challenging features the SRE had to implement were dynamic Object instantiation and easy audio rendering method substitution. The former would allow audio rendering of an unlimited number of objects without reshaping the Max/MSP SRE patch, limiting CPU usage to the exact number of required items, while a shell like implementation allowing spatialization methods to be easily replaced would serve BlenderCAVE's sustainability. Both features are enabled through the Max/MSP `poly~` object, which allows for dynamic patcher loading (up to 1023 at a time for a single poly instance) in addition to appreciable multithread processing capacities.

As seen in Figure 3, the Max/MSP SRE patch is composed of three main entities: the OSC Dispatcher, the Object Class, and the Sound Encoder/Decoder. The Dispatcher routes OSC messages through the patch, from global configuration messages that load the required Encoder/Decoder pair for a given User to Object messages forwarded to different Object instances. The Sound Encoder contains the spatializer, some associated tools (HRTFs selection, ambisonic order, room characteristics, etc.), while the Decoder ensures coherent audio output routing. Listed ObjectUser messages in Figure 4 have been used to route Object outputs (both audio sig-

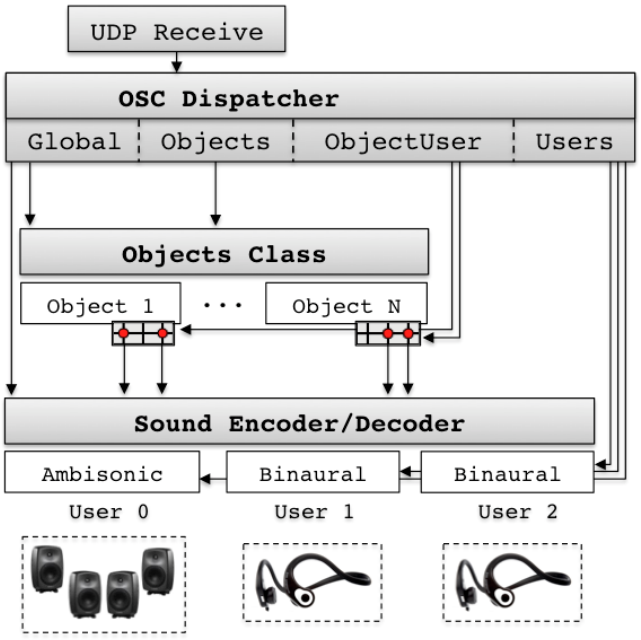
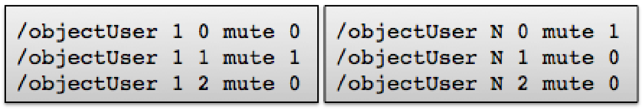


Figure 3: BlenderCAVE associated Sound Rendering Engine architecture implemented in Max/MSP. *objectUser* messages are used to allocate sound sources (Objects) to specific sets of Users. Sounds related to Object N for instance will here be rendered only in Users' headsets through the binaural engines embedded in the Sound Encoder/Decoder module. (See Figure 4 for associated messages)



```
/objectUser 1 0 mute 0
/objectUser 1 1 mute 1
/objectUser 1 2 mute 0
```

```
/objectUser N 0 mute 1
/objectUser N 1 mute 0
/objectUser N 2 mute 0
```

Figure 4: ObjectUser received OSC messages in the SRE that route Object 1 to Users 0 and 2 (i.e. spatialization engine 0 and 2) and Object N to Users 1 and 2.

nal and position) to the desired sound Encoder/Decoder (i.e. User).

Anecdotally, Max/MSP Jitter methods are used to calculate object positions relative to users (homogeneous matrix inversion and multiplication) while the overall spatialization process has been optimized using the `poly~` mute feature which halts signal processing on unused spatializers.

4. INSTALLATION AND PERFORMANCE

BlenderCAVE installation mainly consists in patching original Blender sources (current patch is designed for Blender version 2.64) and configuring environment variables on all the rendering nodes. BlenderCAVE sources and step-by-step installation process are available on the BlenderCAVE webpage [34]. The developed SRE is also available yet not released as a runtime application as it makes use of proprietary spatialization modules, not included in the release. It is however fully compatible with IRCAM's spatialization solutions available for Max/MSP.

Scene graph edition in BlenderCAVE is essentially based on Blender creation and animation tools, while audio rendering related events are written in the Processor script. The overall process being architecture independent, most of our scene development is now carried out on standard laptops, before being ported to the actual VR systems. Once BlenderCAVE installed, basic import of a Blender scene (without tracking nor audio rendering) on a VR system simply requires 2 lines of code in an embedded python script.

BlenderCAVE has been developed on LIMSI's VR platforms, namely the SMART-I2 [35] and EVE [36] systems. Ease of portability of developed scenes between these two very different VR architectures was the driving goal behind the BlenderCAVE project.

The SMART-I2 implements passive adaptive stereoscopic rendering through a pair of front-projected rigid screens which also serve as loudspeakers, each being equipped with twelve electromechanical exciters enabling horizontal Wave Field Synthesis [37] audio rendering. A single computer handles the graphics, with four projectors for the two screens, running under Ubuntu Precise (12.04) on an Intel Core 2 Quad Q9400 at 2.66 GHz with 4 GB of RAM and a pair of NVIDIA GeForce GTX 470 graphic cards. Audio and Wave Field Synthesis rendering are handled on separate machines.

The EVE system has been designed to provide a multi-user/multi-sensorial environment, deploying active adaptive stereoscopic rendering, a set of fifteen loudspeakers and RF modules for wireless audio input and individual binaural rendering. EVE is composed of four rear-projected screens coupled to seven cinema-sized projectors (only one for the floor screen) each controlled by a different computer, altogether conferring about 60m^2 of high definition projection space. The system comprises eigth i7 computers (one per projector plus a monitoring console) running Ubuntu precise, 12 GB of RAM capacity and a Quadro 6000 graphic card. Audio rendering is handled on a separate machine. Strict acoustical constraints have been applied to enhance immersion sensations, from projector insulation to rendering cluster relocation, ensuring an acoustic noise floor of less than 30 dBA at the center of the rendering area when operating the entire system.

One of the main concerns with any real-time interactive system is the effect of CPU load and resulting performance degradation, such as the impact on frame rate, here relative to the performance of the basic BGE. To examine this question, a relatively complex scene was created and the resulting frame-rates exam-

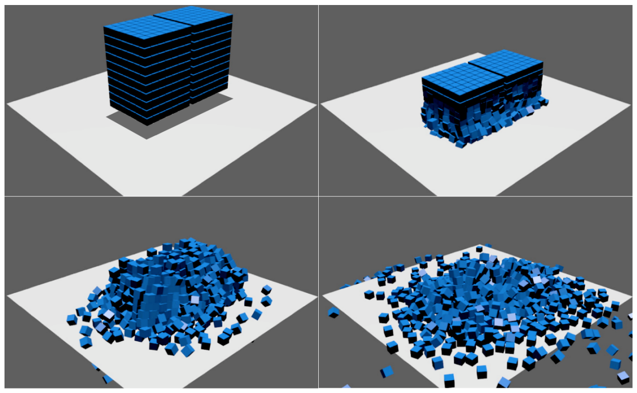


Figure 5: Blender scene used to quantify BlenderCAVE overload regarding basic BGE. It is composed of a stack of 490 cubes which fall on a rigid plane. The objects' motions are driven by the Blender integrated Physics Engine *Bullet*.

ined. The scene, shown in Figure 5, contains a stacked structure composed of 490 independent cubes, which is dropped and whose fall is driven by Blender's Physics Engine. This scene should push the real-time synchronization process until depreciation of the rendered frame rate is observable. Three different scenarios have been compared, simple BGE for reference, a single standalone BlenderCAVE Master, and a multi-screen BlenderCAVE. These test scenarios have been implemented on three different platforms: the EVE system, the SMART-I2 system, and a 2.3 GHz Intel core i7 MacBook with 4 GB of RAM running Mac OS X. The laptop *multi-screen* scenario was configured as two BlenderCAVE windows (Master and one Slave) on a single screen.

The resulting frame-rates during the simulation are reported in Figure 6. The initial drop in frame-rate at around 500 ms corresponds to the first ground impact of the cube stack. While the BlenderCAVE standalone mode with only the BlenderCAVE Master instance does not show any notable affect on rendered frame rate, the multi-screen BlenderCAVE Master/Slave instances have significant reductions, down to 25 fps for EVE's eight display configuration. This degradation in performance is attributed to the frame-by-frame synchronization, and associated testing of movement of every object, highly demanding for such a scene. Ongoing improvements regarding the synchronization process are exposed in next Section.

To present BlenderCAVE's usability we present in the Appendix the creation of an audio based game of hide-and-seek for two players in BlenderCAVE based on material gathered on the Internet.

5. FUTURE WORK

Since submission of this paper, our main concern has been to improve the synchronization process. Previously based on UDP multicast transmissions acknowledged through TCP protocol (Figure 6), BlenderCAVE now exploits only the latter, yet in a multi-unicast fashion. Thus, instead of sending UDP datas packets plus TCP acknowledgment requests, the Master node now merges both processes in distinct TCP communications amongst its Slaves. Enhanced frame-rates related to the dropped stacked structure scene are reported in Figure 7 for the EVE system only.

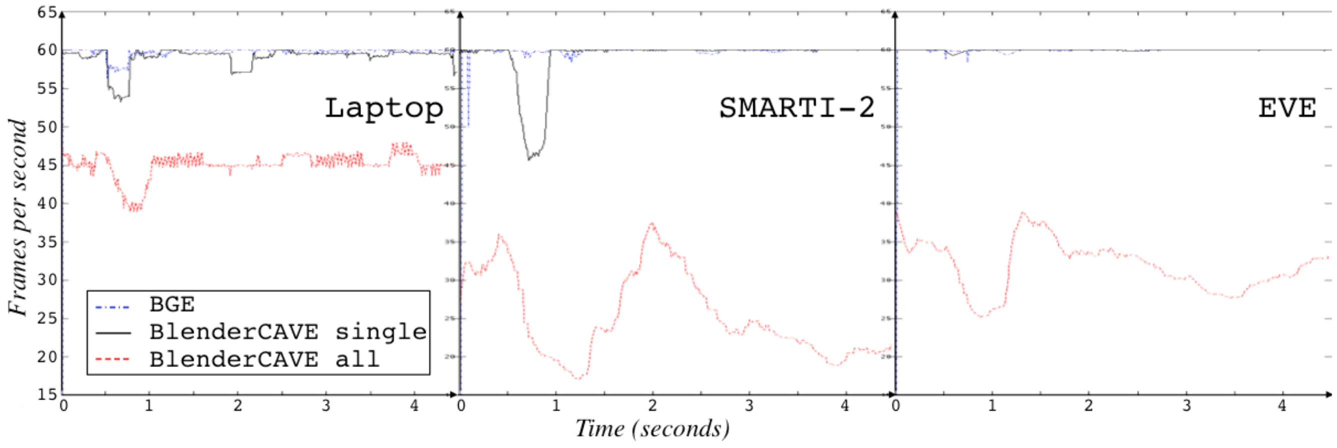


Figure 6: Frame-rates with respect to simulation time on three different architectures running the BGE, a single standalone BlenderCAVE Master (*BlenderCAVE single*), and a multi-screen Master/Slave BlenderCAVE instance (*BlenderCAVE all*). Laptop mutli-screen run involves 2 BlenderCAVE windows (Master and Slave) on a single screen. SMART-I2 multi-screen run involves 4 instances on a single computer. EVE multi-screen run involves a BlenderCAVE instance on each of the 8 rendering nodes.

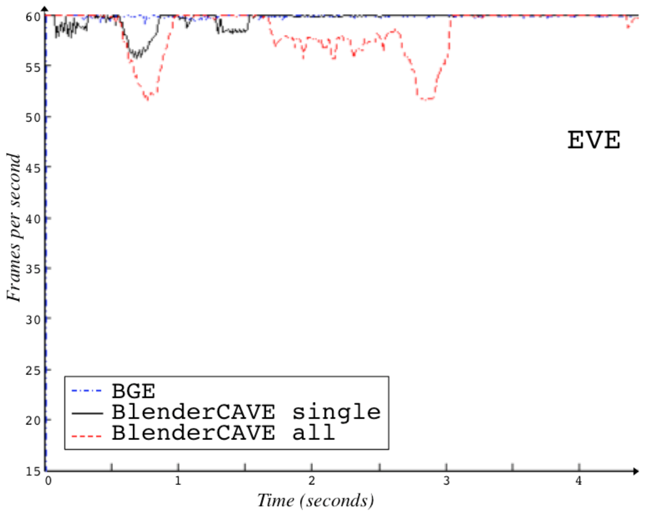


Figure 7: Enhanced frame-rates with respect to simulation time on the EVE system running the BGE, a single standalone Blender-CAVE Master (*BlenderCAVE single*), and a multi-screen Master/Slave BlenderCAVE instance (*BlenderCAVE all*) on 8 rendering nodes.

The next major step in BlenderCAVE development is to have the main elements natively integrated in the Blender trunk, limiting maintenance issues regarding future Blender releases. Discussions are underway with Blender community members and we are working with external developers to confirm BlenderCAVE's multi-architecture aspect. Any Beta testers willing to join the process are obviously welcome.

Room acoustic simulation in BlenderCAVE is being discussed, porting a former project [38] employing a real-time image-source based rendering engine from its previous architecture to BlenderCAVE. Basically, the Blender mesh of a room feeds the geometric acoustics engine while Max/MSP and BlenderCAVE provide the audio and visual renderings respectively.

One could also take advantage of the afore-mentioned audioTWIST OpenSource library to add source directivity and room acoustics to a PureData based SRE, thus providing a fully open source BlenderCAVE suite.

The integration of haptic rendering is a future project of great interest for multimodal interaction. The implementation of haptic control and messaging is therefore among the works currently under development.

We note that the current BGE version does not natively implement particles emission, which may represent an hindrance to some VR scene development. Based on the X-Emitter add-on [39] however, smoke and particles can be generated in the BGE. X-Emitter has been successfully tested in BlenderCAVE, the next step would be to natively integrate this feature in the next release.

6. CONCLUSION

Because of the lack of affordable scene creation software dedicated to Virtual Reality in augmented environments, we decided to implement a new scene graph editor based on the popular Blender Game Engine. Adding multi-user tracked stereoscopic rendering and master-slave synchronization features to adapt the BGE to VR and CAVE architectures, we integrated OSC and VRPN methods to facilitate BGE communications with external applications. With VRPN chiefly associated with I/O interfaces and tracking processes, a Max/MSP based Sound Rendering Engine has been especially implemented to interpret BlenderCAVE OSC messages. This reduces VR scene sound spatialization within the scene graph to basic python instructions, handling multi-user spatialization methods (e.g. binaural, Ambisonic) and has been built as to easily substitute the embedded spatialization engine. Compatible with Windows, Linux, and MacOS for any CAVE like architecture, BlenderCAVE is released as open source. Built on top of self-evolving software accepted as references in their respective domains, BlenderCAVE is aimed at those in need of a straightforward scene development solution for multimodal VR creation.

7. ACKNOWLEDGMENTS

Part of the work reported here was supported by grants from the French government, through the ANR Equipex DIGISCOPE project and the ANRT in the context of a CIFRE industry-linked research funding program with Astrium Services.

8. REFERENCES

[1] "3DVIA Virtools," www.3ds.com/products/3dvia/3dvia-virtools, last check 2013-05-24.

[2] "Quest3D," www.quest3d.com, last check 2013-05-24.

[3] "I'm in VR. Middle VR: Unity VR Plug-In," www.imin.fr/middlevr, last check 2013-05-24.

[4] "Unity3D game engine," www.unity3d.com, last check 2013-05-24.

[5] "Eon Icube," www.eonreality.com, last check 2013-05-24.

[6] Worldviz, "Vizard VR software toolkit," www.worldviz.com/products/vizard, last check 2013-05-24.

[7] J. Jacobson and M. Lewis, "Game engine virtual reality with CaveUT," *Computer*, vol. 38, no. 4, pp. 79–82, April 2005.

[8] "UnrealTournament," www.unrealtournament.com, last check 2013-05-24.

[9] "OpenSceneGraph," www.openscenegraph.org/projects/osg, last check 2013-05-24.

[10] J. P. Schulze, A. Prudhomme, P. Weber, and T. A. DeFanti, "CalVR: an advanced open source virtual reality software framework," in *IS&T/SPIE Electronic Imaging*. International Society for Optics and Photonics, 2013, pp. 864 902–864 902.

[11] A. Bierbaum, C. Just, P. Hartling, K. Meinert, A. Baker, and C. Cruz-Neira, "VR Juggler: A virtual platform for virtual reality application development," in *Proceedings of Virtual Reality, 2001*. IEEE, 2001, pp. 89–96.

[12] D. Germans, H. J. Spoelder, L. Renambot, and H. E. Bal, "VIRPI: a high-level toolkit for interactive scientific visualization in virtual reality," in *Proc. Immersive Projection Technology/Eurographics Virtual Environments Workshop (IPT/EGVE), May*, 2001, pp. 16–18.

[13] D. Touraine, P. Bourdot, Y. Bellik, and L. Bolot, "A framework to manage multimodal fusion of events for advanced interactions within virtual environments," in *ACM International Conference Proceeding Series*, vol. 23, 2002, pp. 159–168.

[14] M. Courgeon, J.-C. Martin, and C. Jacquemin, "MARC: a multimodal affective and reactive character," in *Proceedings of the 1st Workshop on AFFective Interaction in Natural Environments*, 2008.

[15] T. Wright and G. Madey, "A survey of collaborative virtual environment technologies," *University of Notre Dame-USA, Tech. Rep*, 2008.

[16] C. Just, K. Meinert, A. Bierbaum, and P. Hartling, "Open source virtual reality," in *Virtual Reality, 2002. Proceedings. IEEE*, 2002, pp. 303–303.

[17] Q. Zhao, "A survey on virtual reality," *Science in China Series F: Information Sciences*, vol. 52, pp. 348–400, 2009. [Online]. Available: http://dx.doi.org/10.1007/s11432-009-0066-0

[18] T. Fernando, N. Murray, G. Gautier, S. Mihindu, K. Loupos, P. Gravez, H. Hoffmann, J. Blondelle, S. Di Marca, M. Fontana *et al.*, "Contrat INTUITION IST-NMP-1-507248-2, WP1.B State-of-the-art in VR," Information Society Technologies, Tech. Rep., 2007.

[19] R. M. Taylor, II, T. C. Hudson, A. Seeger, H. Weber, J. Juliano, and A. T. Helser, "VRPN: a device-independent, network-transparent VR peripheral system," in *Proceedings of the ACM symposium on Virtual reality software and technology*, ser. VRST '01. New York, NY, USA: ACM, 2001, pp. 55–61. [Online]. Available: http://doi.acm.org/10.1145/505008.505019

[20] M. Wright, "Open Sound Control: an enabling technology for musical networking," *Organised Sound*, vol. 10, pp. 193–200, 11 2005.

[21] G. Reitmayr and D. Schmalstieg, "An open software architecture for virtual reality interaction," in *Proceedings of the ACM symposium on Virtual reality software and technology*. ACM, 2001, pp. 47–54.

[22] L. Valbom and A. Marcos, "Wave: Sound and music in an immersive environment," *Computers & Graphics*, vol. 29, no. 6, pp. 871–881, 2005.

[23] G. Wakefield and W. Smith, "Cosm: A toolkit for composing immersive audio-visual worlds of agency and autonomy," in *Proceedings of the International Computer Music Conference (ICMC)*, 2011.

[24] D. Poirier-Quinot, D. Touraine, and B. F. Katz, "Blender-Cave 3D-s project, OpenSource architecture adaptation to virtual reality research expectations," in *Blender Conference*, Amsterdam, Oct 2012.

[25] "Blender," www.blender.org, last check 2013-05-24.

[26] "Bullet," bulletphysics.org, last check 2013-05-24.

[27] "BlenderNetwork," www.blendernetwork.org, last check 2013-05-24.

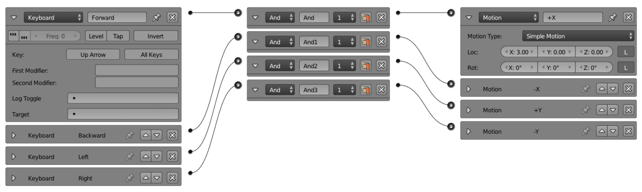
[28] J. Gascón, J. M. Bayona, J. M. Espadero, and M. A. Otaduy, "BlenderCAVE: Easy VR authoring for multi-screen displays," *SIACG 2011: V Ibero-American Symposium in Computer Graphics*, 2011.

[29] M. Wright, A. Freed, and A. Momeni, "Open Sound Control: State of the Art 2003," in *International Conference on New Interfaces for Musical Expression*, Montreal, 2003, pp. 153–159.

[30] N. Olaiz, P. Arumí, T. Mateos, and D. Garcia, "3D-audio with CLAM and Blender's Game Engine," in *Linux Audio Conference*, 2009.

[31] M. F. O'dwyer, G. Potard, and I. Burnett, "A 16-speaker 3D audio-visual display interface and control system," in *International Conference on Auditory Display*, 2004.

[32] M. Wozniewski, Z. Settel, and J. R. Cooperstock, "Audioscape: A Pure Data library for management of virtual environments and spatial audio," in *Pure Data Convention, Montreal*, 2007.

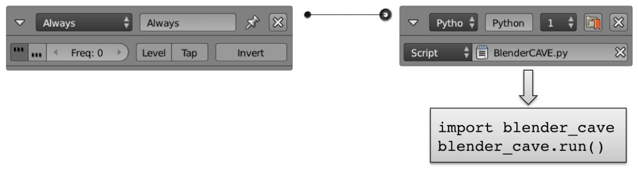


Keyboard | Forward | Freq: 0 | Level | Tap | Invert
Key: Up Arrow | All Keys
First Modifier:
Second Modifier:
Log Toggle
Target
Keyboard Backward
Keyboard Left
Keyboard Right

And | And | 1
And | And1 | 1
And | And2 | 1
And | And3 | 1

Motion | +X
Motion Type: Simple Motion
Loc: X: 3.00 | Y: 0.00 | Z: 0.00 | L
Rot: X: 0° | Y: 0° | Z: 0° | L
Motion -X
Motion +Y
Motion -Y

(a) Simple keyboard control logic brick configuration for the BGE. Here for example the *up arrow* has been mapped as to move the user in the positive X axis direction.



Always | Always | Freq: 0 | Level | Tap | Invert

Pytho | Python | 1
Script | BlenderCAVE.py

```
import blender_cave
blender_cave.run()
```

(b) Python script and logic bricks needed to import a scene from BGE to BlenderCAVE. The *blender_cave.run()* command execution is enough to import a scene from Blender to BlenderCAVE.

Figure 8: Blender logic examples.

[33] "audioTWIST," tot.sat.qc.ca./logiciels_audiotwist.html, last check 2013-05-24.

[34] "BlenderCAVE - LIMSI," blendercave.limsi.fr, last check 2013-05-24.

[35] M. Rébillat, E. Corteel, and B. Katz, "SMART-I2 "Spatial Multi-user Audio-visual Real-Time Interactive Interface"," in *125th Convention of the Audio Engineering Society*, 2008, pp. 7609_1–16.

[36] "Evolutive Virtual Environment," www.limsi.fr/venise/EVEsystem, last check 2013-05-24.

[37] A. J. Berkhout, D. de Vries, and P. Vogel, "Acoustic control by wave field synthesis," *The Journal of the Acoustical Society of America*, vol. 93, p. 2764, 1993.

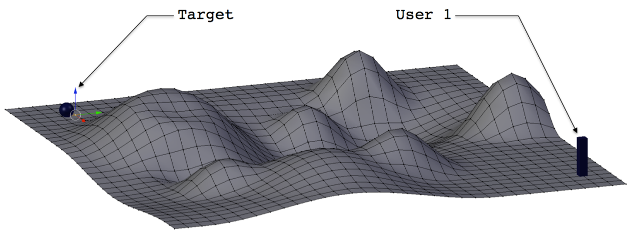
[38] M. Noisternig, B. F. Katz, S. Siltanen, and L. Savioja, "Framework for real-time auralization in architectural acoustics," *Acta Acustica United with Acustica*, vol. 94, no. 6, pp. 1000–1015, 2008.

[39] "X-emitter," solarlune-gameup.blogspot.fr/search/label/X-Emitter, last check 2013-05-24.

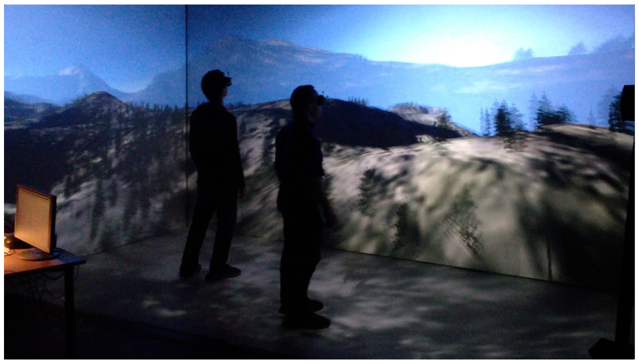
Hide-and-seek game creation

We describe here the step-by-step creation of a VR scene in BlenderCAVE. Using classic first person controls (keyboard for motion, mouse for orientation), the first user is to find an object hidden in the scene. It will be auralized by a second user equipped with a microphone through binaural rendering in both users headsets, to which we superposed an ambisonic rendered ambiance to improve immersion and hinder the localization task. We chose keyboard/mouse control, not really CAVE like, to simplify the implementation description.

The first step is to edit scene objects (meshes) and configure the keyboard/mouse controls to be used in the BGE. While not straightforward for beginners, landscape and python control scripts can be quickly downloaded from one of the numerous Blender material repositories on the Internet. Keyboard motion may be easily implemented with the BGE logic bricks, as seen in Figure 8(a). A



(a) Hide-and-Seek Game reduced landscape mesh.

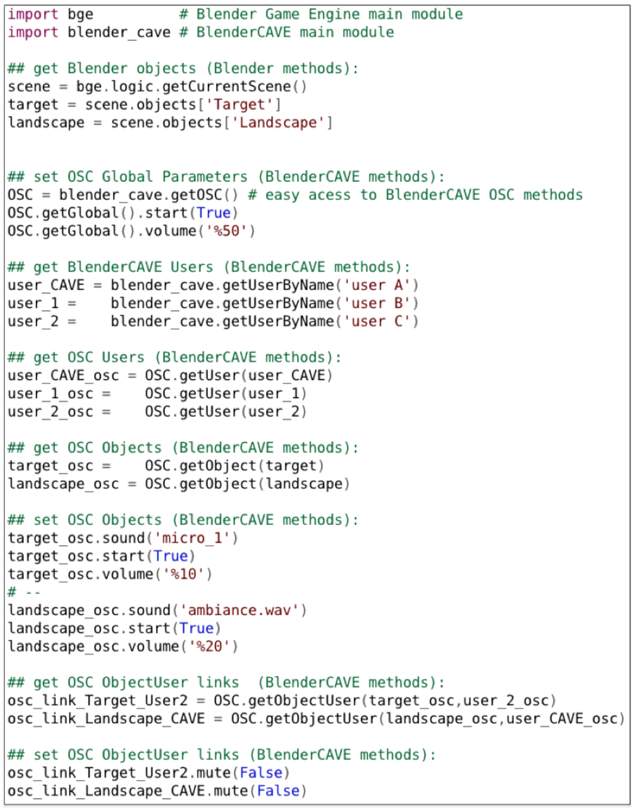


(b) BlenderCAVE Hide-and-Seek Game displayed in the EVE environment.

Figure 9: Hide-and-Seek Game.

Target object is added and placed away from *User 1* starting point, illustrated in the landscape of Figure 9(a).

Once the scene has been checked in the BGE, the python script and logic bricks of Figure 8(b) are added to display the scene in our CAVE. Events' audio rendering is implemented in a simplified processor script (we skipped the set user position method description) shown in Figure 10. Once the remote SRE is launched, the game is ready to start (Figure 9(b)).



```python
import bge          # Blender Game Engine main module
import blender_cave # BlenderCAVE main module

## get Blender objects (Blender methods):
scene = bge.logic.getCurrentScene()
target = scene.objects['Target']
landscape = scene.objects['Landscape']


## set OSC Global Parameters (BlenderCAVE methods):
OSC = blender_cave.getOSC() # easy acess to BlenderCAVE OSC methods
OSC.getGlobal().start(True)
OSC.getGlobal().volume('%50')

## get BlenderCAVE Users (BlenderCAVE methods):
user_CAVE = blender_cave.getUserByName('user A')
user_1 =    blender_cave.getUserByName('user B')
user_2 =    blender_cave.getUserByName('user C')

## get OSC Users (BlenderCAVE methods):
user_CAVE_osc = OSC.getUser(user_CAVE)
user_1_osc =    OSC.getUser(user_1)
user_2_osc =    OSC.getUser(user_2)

## get OSC Objects (BlenderCAVE methods):
target_osc =    OSC.getObject(target)
landscape_osc = OSC.getObject(landscape)

## set OSC Objects (BlenderCAVE methods):
target_osc.sound('micro_1')
target_osc.start(True)
target_osc.volume('%10')
# --
landscape_osc.sound('ambiance.wav')
landscape_osc.start(True)
landscape_osc.volume('%20')

## get OSC ObjectUser links  (BlenderCAVE methods):
osc_link_Target_User2 = OSC.getObjectUser(target_osc,user_2_osc)
osc_link_Landscape_CAVE = OSC.getObjectUser(landscape_osc,user_CAVE_osc)

## set OSC ObjectUser links (BlenderCAVE methods):
osc_link_Target_User2.mute(False)
osc_link_Landscape_CAVE.mute(False)
```

Figure 10: Audio rendering implementation based on the BlenderCAVE getOSC() module. This python script is issued from the *processor.py* script.