V 3.2

----- blenderCAVE – INSTALL & HOW TO USE -----

blenderCAVE is a Scene Graph Editor for CAVE and Video Wall architectures, based on the open source software blender (www.blender.org).

Features

- Tracked (adaptive) stereoscopy *(2 Users)*[1].
- Spatial Sound *(Ambisonic, Binaural 2 Users)*[2].

[1]embedded VRPN API
[2]embedded OSC API and Max/MSP Sound Rendering Engine

Associated sources at http://blendercave.limsi.fr/doku.php

Table of Contents

I. Install (on each of the rendering nodes)

To ease further references, lets suppose you work in a folder named ~/BlenderWorkspace.
Any file creation, if not mentioned otherwise, takes place in that folder.

a) Download blenderCave sources

- Create a folder, here “**blenderCave_svn**”

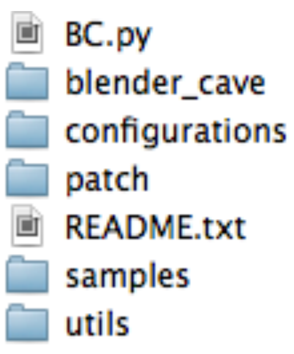
From the LISMI, if you have a developer access:
- Open a terminal in this folder, type:

```
> svn co https://webdev.limsi.fr/svn/blenderCAVE/trunk
> svn up
```

Else:
Download the .zip file from https://github.com/jgascon/BlenderCave and extract its content into your blenderCave_svn/ folder.

According to your needs and blenderCAVE project current state, you will have copied several unnecessary files in your blenderCave_svn folder. As a standard user, you will want to access the last **release**, as a developer the current **trunk**, etc.

Delete the unnecessary versions (trunk or other releases) and make sure that blenderCave_svn/ folder looks like:



```
BC.py
blender_cave
configurations
patch
README.txt
samples
utils
```

Since blenderCAVE is a patched version of blender, you will need to download the blender version corresponding to the blenderCAVE version related patch. You will find the .patch file in blenderCave/patch folder (blender_revision_xxxxx.patch).
Remember it for b).

b) Download, patch and compile Blender sources

<u># Fetch Blender sources</u>
- Create a folder, here “**blender_svn**”
- Open a terminal in this folder, type:

```
> svn co https://svn.blender.org/svnroot/bf-blender/trunk/blender -rXXXXX
```

where XXXX is the desired blender sources revision number.

- Later, to refresh your sources, type

```
> svn up -rXXXXX
```

# Patch blender sources
- Open a terminal in **blender_svn/blender**, type:

```
> patch -p0 < ../../blenderCave_svn/patch/ blender_revision_XXXXX.patch
```

- Later, if you want to reapply the patch:

```
> patch -p0 -R < ../../blenderCave_svn/patch/ blender_revision_XXXXX.patch
```

# Install blender dependencies and compile blender (just patched) sources:
http://wiki.blender.org/index.php/Dev:Doc/Building_Blender

From here, your workspace should look like:

```
~/BlenderWorkspace
        /blender_svn
                /blender
                (/lib)
                /build
        /blenderCave_svn
                …
                /blender_cave
                …
                /patch
```

c) Setup blenderCAVE working environment

Notes:
- The locations of .xml, .py, … scritps suggested hereafter are, if not mentioned otherwise, not compulsory (yet coherent along the tutorial).
- Find further details about the different xml flags of the xml file in the "XML configuration" Annex.

Files of interest:

- blenderCave_svn/configurations/**<confName>.xml**

*Define several parameters that should be constant along the blenderCAVE runs (screens names, positions, eyes separation, etc.) on a given architecture. Basic use involves 2 of them, defining parameters for the **standalone/debug mode** and **the full Virtual Environment mode**.*

<u># Store configuration scripts (.xml)</u>
You want to keep these two files on your computer to be sure that you won't erase them at next update:

- Open a terminal in ~/.config
- Create a directory blenderCave_config
- Copy the content of blenderCave_svn/configurations
in ~/.config/blenderCave_config

<u># Setup configuration scripts (.xml)</u>

.xml configuration script is closely related to your architecture. It contains:

- Rendering nodes names, screen positions, and graphic buffer configurations. OSC and VRPN parameters, etc.

*for a start, use the provided .xml configuration script **main.xml**. see "XML configuration" Annex to define your own script.*

Note:
- BC.py needs to invoke the patched blenderplayer. If not defined in your aliases, you should inform the complete path in the .xml (in the <system> flag).
- You can indicate a path for logs registration in the .xml (in the <system> flag). blenderCAVE logs will be recorded for full Virtual Environment runs only (one for each rendering node) else they will be printed in the terminal window used to launch BC.py

<u># Simple Link to the BC.py script</u>
blenderCAVE is launched through a terminal window, you thus need to be able to access the BC.py script from any folder. Link the script blenderCave_svn/BC.py in one of your $PATH defined folders:

<u>MacOS & UNIX</u>
- Open a terminal in ~/local/bin/, type:
```
> ln –s ~/BlenderWorkspace/blenderCave_svn/BC.py
```

d) Test blenderCAVE Install

<u># Launch</u>
The BC.py script needs argument at first call (afterward, it stores last arguments until told different). >***BC.py –h to get args list***.

- Open a terminal at blenderCave_svn/samples, type:
```
> BC.py –s  -f <sceneName>.blend --config-file <configurationFile.xml> -c
```

```
<configuration>
```

Where xxxxx.blend is one of the ready-to-use blend files included in sample folder (start with BlenderCAVE.API.blend) configurationFile.xml is the .xml configuration file you previously stored in ~/.config/blenderCave_config/ and configuration is one of them configurations (alone, full, etc.) defined in it.

# Quit

There are 3 main ways to properly quit blenderCAVE.
- In the terminal window you used to launch BC.py type

```
> BC.py
```

(BC.py –s will stop the current scene and reload it)

or

- Press ESC while your mouse is over a slave node

or

- Associate a Keyboard Sensor (in Blender Logic Editor) to the python script

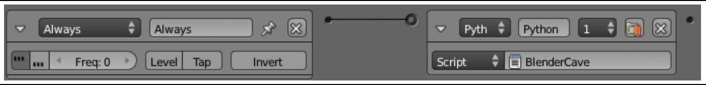
```
import blender_cave
blender_cave.quit(<string>)
```

Note: the <string> is here to be printed in terminal window for debug purpose.

II. Simple blenderCAVE-isation

To adapt a .blend file for a run in blenderCAVE:
- Create a new blender file or fetch one of yours.
- Create a new embedded script (you may name it “blenderCave” for later reference).
- In this script, copy the 2 following lines and link it to an “Always” Sensor and activate level triggering (pulse mode, freq 0):

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



Always | Always
Freq: 0 | Level | Tap | Invert
Pyth | Python | 1
Script | BlenderCave

- Copy the scene (.blend) onto each of the system’s rendering nodes.
- Launch the blenderCAVE using the BC.py launcher described in the previous section.

Note about Blender logic blocks in blenderCAVE:
- If your .blend file already contains logic blocks or any attached (embedded or not) python scripts, they still should be available in its blenderCAVE version. You will however have to place the mouse cursor above the ‘Master’ Node if you want your interactions to be coherently dispatched to your Slaves Nodes, else only the « Mouse cursor associated » Slave node will receive the information.
- Any automatic sensor (Always Sensors, etc…) will be activated through all you nodes (Master and Slaves). You can avoid this by using the ‘blender_cave.isMaster’ property, False if the node is a Slave.

- An alternative consists into re-writing your interactions in the "processor" python script described below.

III. How To Use (with OSC, VRPN & associated APIs)

There are 2 kinds of interactions you may want in your scene:
- blender native interactions
- blenderCAVE interactions

The first work as they should, but for the *Note* in part II.
Useful APIs:
- Python (http://docs.python.org/2/library/)
- blender (http://www.blender.org/documentation/blender_python_api_2_63_17/)

(check for more recent releases)

The last consist into sending/receiving external messages (VRPN or OSC), i.e. actions in the real world that affect the virtual and vice versa.

b) blenderCAVE API

Once blender_cave module imported, the blenderCAVE methods may be used as any other embedded functions.

import bge
scene = bge.logic.getCurrentScene()
	import blender_cave
	blender_cave.getAllUsers()

See embedded python scripts in samples/BlenderCave_API.blend.

a) VRPN API

VRPN is a protocol used in Virtual Reality to exchange data with external devices. See http://www.cs.unc.edu/Research/vrpn/.

blenderCAVE behave like a VRPN client. At the other end, a VRPN server will host different tracker or sensors that would be as many haptic arms, tracked stereoscopic glasses or Wiimote devices. The server will associate a name to these device, along with a variable "info" that holds the useful information about
the considered device.

In blenderCAVE, the receiving of VRPN messages and definition of associated methods is done in the <blender_scene_name>.processor.py script attached to a scene (see examples in the */samples* folder). **blenderCAVE works fine without any processor script**.

To be able to interact in your blenderCAVE scene with a VRPN compatible interface you will have to:

i) Define the interface in **your** *vprn.cfg* script
ii) Define the related processor method in the blenderCAVE .xml configuration script
iii) Define the processor method in the <blender_scene_name>.processor.py script attached to your blenderCAVE scene

Example with a Nintendo Wii Controller:

i) in your vrpn.cfg file, add:

```
vrpn_WiiMote          WiiMote0       1 0 0 1
```

ii) in the blenderCAVE .xml configuration script (e.g. single.xml), add:

```
<processor>
 <vrpn>
  <analog name="WiiMote0" host="<vrpnServer@>" processor_method="wiiAnalog"/>
  <button name="WiiMote0" host="<vrpnServer@>" processor_method="wiiButton"/>
 </vrpn>
</processor>
```

(analog will receive accelerometer data from the WiiMote, button only the pressed button states)

iii) in the <blend_file_name>.processor.py script (e.g. BlenderCave_API.processor.py), add:

```
def wiiAnalog(self, info):
    print (''Analog from Wii through VPRN '', info)

def wiiButton(self, info):
    print ('Button from Wii through VPRN '', info)
```

Here, both functions will be executed whenever the VRPN server receives data from the WiiMote (the wiiButton when your touch a button, the wiiAnalog when you move the WiiMote).

c) OSC API

OSC is a protocol used to send / receive data through applications. See http://opensoundcontrol.org/.
blenderCAVE includes a MaxMSP (http://cycling74.com/) Sound Rendering Engine available at http://blendercave.limsi.fr/doku.php. It is however possible (and advised) to make it work with any other OSC client and fathom it for other purposes.

While the OSC API allows to easily send OSC (UDP) flags, the MaxMSP associated Sound Rendering Engine has been design to receive an process these flags.
Once you've opened the blenderCAVE_Sound_Rendering_Engine_vX.maxpat on the OSC server as defined in the .xml configuration file

```
<processor>
 <osc host='serverName' port='3819'/>
</processor>
```

and modified it to fit to your needs (spatializer, speakers mapping, microphone inputs, etc.), the rest of the sound adding process takes place in blenderCAVE.

See samples/BlenderCave_OSC.blend and samples/BlenderCave_OSC_API.blend

LIMSI members, see http://wikivenise.limsi.fr/index.php/Open_Sound_Control

IV. Annex

a) OSC API

Output of BlenderCave_OSC_API.blend :

```
Global OSC methods:
Global OSC methods for sync. with OSC Sound Rendering Engine

global_OSC = blender_cave.getOSC().getGlobal()
 - configuration():
        Set the OSC client configuration (e.g. “ambisonic EVE”)
        use: global_OSC.configuration(<string>)
        OSC flag: /global configuration 'ambisonic EVE'
 - mute():
        Mute/Unmute global audio
        use: global_OSC.mute(<bool>)
        OSC flag: /global mute <1/0>
 - reset():
        Reset OSC client (used automatically when calling blender_cave.quit())
        use: global_OSC.reset()
        OSC flag: /global reset
 - start():
        Start/Stop gloabal Audio
        use: global_OSC.start(<bool>)
        OSC flag: /global start <1/0>
 - volume():
        Set global Volume
        use: global_OSC.volume(<string>)
        OSC flag: /global volume +<int> / -<int> (relative)
                  or /global volume %<int>  (absolute)

Object OSC methods:
Object OSC methods for sync. with OSC Sound Rendering Engine

object_OSC = blender_cave.getOSC().getObject(<bge.types.KX_Gameobject>)
 - loop():
        Set loop on Object sound
        use: object_OSC.loop(<bool>)
        OSC flag: /object <ID> loop <1/0>
 - mute():
        Mute/Unmute Object audio
        use: object_OSC.mute(<bool>)
        OSC flag: /object <ID> mute <1/0>
```

- position():
    set OSC Object position
    use: object_OSC.position(<Mat4x4>)
    OSC flag: /object <ID> position <float> (x16)
- sound():
    Set Object sound
    use: object_OSC.sound(<str>)
    OSC flag: /object <ID> sound <str>
- start():
    Set Object volume
    use: object_OSC.start(<bool>)
    OSC flag: flag: /object <ID> start <1/0>
- volume():
    Set Object Volume
    use: object_OSC.volume(<string>)
    OSC flag: flag: /object <ID> volume +<int> / -<int> (relative)
    or /object volume <ID> %<int> (absolute)

User OSC methods:
User OSC methods for sync. with OSC Sound Rendering Engine

user_OSC = blender_cave.getUserByName(<string>) where <string> is the user name defined in you .xml file.

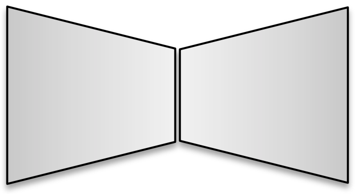
- brightness():
    NOT DEFINED
- envelop():
    NOT DEFINED
- heavyness():
    NOT DEFINED
- hrtf():
    NOT DEFINED
- livelyness():
    NOT DEFINED
- mute():
    Mute/Unmute User audio
    use: user_OSC.mute(<bool>)
    OSC flag: /user <ID> mute <1/0>
- name():
    NOT DEFINED
- position():
    set OSC User position
    use: user_OSC.position(<Mat4x4>)
    OSC flag: /user <ID> position <float> (x16)
- presence():
    NOT DEFINED

```
 - start():
         Set User volume
         use: user_OSC.start(<bool>)
         OSC flag: flag: /user <ID> start <1/0>
 - volume():
         Set User Volume
         use: user_OSC.volume(<string>)
         OSC flag: flag: /user <ID> volume +<int> / -<int> (relative)
                     or /user volume <ID> %<int>  (absolute)
 - warmth():
         NOT DEFINED

ObjectUser OSC methods:
objectUser OSC methods for sync. with OSC Sound Rendering Engine

object_user_OSC = blender_cave.
 - mute():
         Mute/Unmute User audio
         use: user_OSC.mute(<bool>)
         OSC flag: /user <ID> mute <1/0>
 - volume():
         Set User Volume
         use: user_OSC.volume(<string>)
         OSC flag: flag: /user <ID> volume +<int> / -<int> (relative)
                     or /user volume <ID> %<int>  (absolute)
```

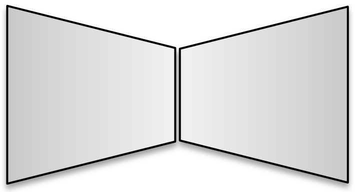
b) XML configuration

Here above, different architectures are exposed. Associated .xml configuration files can be found in /configuration.

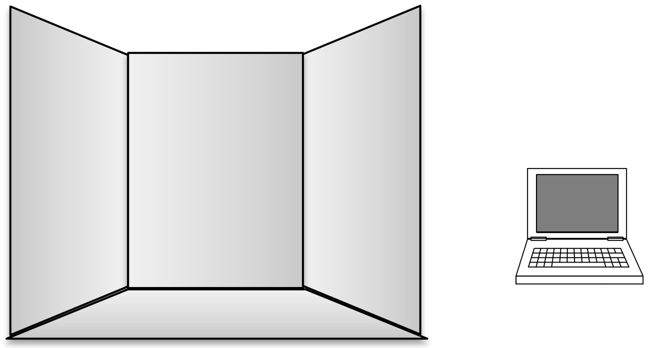
Configuration 1:
2 Computers (*Bob* and *Alice*) / 2 Screens (Flat) / no VRPN server / OSC Sound Rendering Engine on *Chopin*.



<u>Configuration 2:</u>
1 Computers (*Bob*) / 2 Screens (Stereoscopic) / VRPN server on *Bob-VRPN* / OSC Sound Rendering Engine on *Chopin*.



<u>Configuration 3:</u>
7 Computers (*Bob*-N) / 5 Screens / VRPN server on *Bob-VRPN* / OSC server on *Chopin*.



Notes:
*- **screen name** is "console" or "main", the former being used for control screen not part of the rendering cluster.*
*- **player options** defines blenderplayer options (type >blenderplayer ? in terminal if need be).*
*- **graphic buffer** link graphic buffer to a given user/eye.*
*- **corner** (x,y,z) coordinates of screen corners.*