Building blender on Linux for blenderVR

In the following, we assume that you are using `bash` as shell. You should also have `libboost` (`date-time`, `filesystem`, `locale`, `regex`, `serialization`, `system`, `thread`) developpement packages installed on your Linux. blender requires at least release 1.48 of boost library.
To compile faster, you can replace `make` by `make -j n` with *n* equals to twice the number of cores of your CPUs.

Paths

At first, we must define where to install blender for blenderVR. Due to its dependencies, we will define a whole tree with all necessary libraries.

Set an environment variable with the root folder of blender installation:

```
export BLENDER_ROOT_PATH=/usr/local/blender/2.72
export BLENDER_ROOT_DEPENDENCIES=$BLENDER_ROOT_PATH/dependencies
```

By default, we suggest `/usr/local/blender/2.72`, but you can use whatever path you prefer. You can even choose a path in your home directory although you won't be able to share it with other computer users and you should not do `sudo` commands. Anyway, don't forget that you won't be able to move it after compilation as python built in blender will rely on this given path to find some modules.

Ensure that you have read and write access to this path and create its folder:

```
sudo mkdir -p $BLENDER_ROOT_PATH
sudo chown $(whoami) $BLENDER_ROOT_PATH
cd $BLENDER_ROOT_PATH
mkdir $BLENDER_ROOT_DEPENDENCIES
mkdir compile
```

script for environment variables

First, we will create a little script to set correct environment variable:

```
cat > $BLENDER_ROOT_PATH/compile/.vars << EOF
export BLENDER_ROOT_PATH="$BLENDER_ROOT_PATH"
export BLENDER_ROOT_DEPENDENCIES="$BLENDER_ROOT_PATH/dependencies"
if [[ \$PATH != *$BLENDER_ROOT_PATH/dependencies/bin* ]] ; then
   export PATH="$BLENDER_ROOT_PATH/dependencies/bin:\$PATH"
fi
if [[ \$LD_LIBRARY_PATH != *$BLENDER_ROOT_PATH/dependencies/lib* ]] ; then
   export
LD_LIBRARY_PATH="$BLENDER_ROOT_PATH/dependencies/lib:\$LD_LIBRARY_PATH"
fi
EOF
```

Thus, whenever you want to compile part of blender, you will only have to do

```
. /usr/local/blender/2.72/compile/.vars
```

to get all the relevant environment variables defined (beware of the dot and the space before the path). You can replace `/usr/local/blender/2.72` by your blender root path.

In the following steps, we assume that environment variables defined above are set.

You can make sure by typing:

```
echo $PATH | cut -d ':' -f1
```

If the path is wrong (you should recognize your BLENDER_ROOT_DEPENDENCIES variable before `/bin`), then you can retype the upper command `. /usr/local/blender/2.72/compile/.vars`.

cmake

Current usable cmake to use is version 2.8.12.2. If your system already have this version, you can bypass this step. Beware: 3.x versions are not currently supported by blender.

```
export CMAKE_VERSION=2.8.12.2
echo "export CMAKE_VERSION=$CMAKE_VERSION" >>
$BLENDER_ROOT_PATH/compile/.vars
mkdir -p $BLENDER_ROOT_PATH/compile/cmake-$CMAKE_VERSION/build
cd $BLENDER_ROOT_PATH/compile/cmake-$CMAKE_VERSION/
wget www.cmake.org/files/v$(echo $CMAKE_VERSION | cut -d '.' -f1,2)/cmake-
$CMAKE_VERSION.tar.gz
tar -xf cmake-$CMAKE_VERSION.tar.gz
cd build
cmake -DCMAKE_INSTALL_PREFIX=$BLENDER_ROOT_DEPENDENCIES ../cmake-
$CMAKE_VERSION
make
make install
```

Due to unknown issue, when you try to configure a compilation with cmake after that, you can have wrong cmake behavior (e.g. saying that you have invalid rights). If so, you should start a new shell, load the environment variables (`. /usr/local/blender/2.72/compile/.vars`) and continue the process where it failed.

Python and numpy

python

blender 2.72 requires python 3.3.0. If you have thise version installed in your system (with numpy), you should bypass this step.

```
export PYTHON_VERSION=3.3.0
export PYTHON_MAJOR_VERSION=$(echo $PYTHON_VERSION  | cut -d '.' -f1,2)
echo "export PYTHON_VERSION=$PYTHON_VERSION" >>
$BLENDER_ROOT_PATH/compile/.vars
echo "export PYTHON_MAJOR_VERSION=$PYTHON_MAJOR_VERSION" >>
$BLENDER_ROOT_PATH/compile/.vars
mkdir -p $BLENDER_ROOT_PATH/compile/python-$PYTHON_VERSION/build
cd $BLENDER_ROOT_PATH/compile/python-$PYTHON_VERSION/
wget www.python.org/ftp/python/$PYTHON_VERSION/Python-$PYTHON_VERSION.tgz
tar -xf Python-$PYTHON_VERSION.tgz
cd build
../Python-$PYTHON_VERSION/configure --prefix=$BLENDER_ROOT_DEPENDENCIES -
enable-ipv6 --enable-loadable-sqlite-extensions --enable-shared
make
make install
```

If everything is set correctly, you should be able to start python with:

```
$ python3
Python 3.3.0 (default, Sep 12 2014, 12:26:36)
[GCC 4.6.3] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

The date or GCC version should change, but the version must be 3.3.0 !

pip

Pip is a python tool to automagically install python packages in its path (i.e.: not system wide). To install it:

```
cd $BLENDER_ROOT_PATH/compile/
wget bootstrap.pypa.io/get-pip.py
python3 get-pip.py
```

numpy

With PIP, installation of numpy is very simple:

```
pip install numpy
```

VRPN

VRPN install is not compulsory, yet blenderVR will require the VRPN protocol for every VR related device interactions (e.g. tracking system)

```
export VRPN_VERSION=07_31
echo "export VRPN_VERSION=$VRPN_VERSION" >> $BLENDER_ROOT_PATH/compile/.vars
mkdir -p $BLENDER_ROOT_PATH/compile/vrpn-$VRPN_VERSION/build
cd $BLENDER_ROOT_PATH/compile/vrpn-$VRPN_VERSION/
wget www.cs.unc.edu/Research/vrpn/downloads/vrpn_$VRPN_VERSION.zip
unzip vrpn_$VRPN_VERSION.zip
cd build
cmake -DCMAKE_INSTALL_PREFIX=$BLENDER_ROOT_DEPENDENCIES ../vrpn
make
make install
```

blender

Let start the “hard work” ... Here we'll download blender source, patch them and compile blender into its (slightly) modified version that'll serve as blenderVR core application.

download

We straightly get blender from the gitHub repository of blenderVR. As such, it is already patched.

```
export BLENDER_VERSION=2.72
echo "export BLENDER_VERSION=$BLENDER_VERSION" >>
$BLENDER_ROOT_PATH/compile/.vars
export PYTHON_ROOT_DIR=$BLENDER_ROOT_DEPENDENCIES
echo "export PYTHON_ROOT_DIR=$BLENDER_ROOT_DEPENDENCIES" >>
$BLENDER_ROOT_PATH/compile/.vars
mkdir -p $BLENDER_ROOT_PATH/compile/blender-$BLENDER_VERSION/build
cd $BLENDER_ROOT_PATH/compile/blender-$BLENDER_VERSION/
git clone http://github.com/BlenderVR/blender blender-$BLENDER_VERSION
cd blender-$BLENDER_VERSION/release/scripts
git clone git://git.blender.org/blender-addons.git addons
```

configure and compile

```
cd $BLENDER_ROOT_PATH/compile/blender-$BLENDER_VERSION/build
cmake -DCMAKE_INSTALL_PREFIX=$BLENDER_ROOT_DEPENDENCIES -DWITH_CYCLES=OFF -
DWITH_PLAYER=ON ../blender-$BLENDER_VERSION
make
make install
```

Feel free to use ccmake to adapt to your requirements. For instance, you can add FFMPEG or any other option flag to include associated library to blender compilation.

automatically add VRPN ?

If you use VRPN, you'll want to create a symbolic link towards its shared object to avoid an additional VRPN folder inside the blender tree.

```
cd $BLENDER_ROOT_PATH/compile/blender-
$BLENDER_VERSION/build/bin/$BLENDER_VERSION/python/lib/python$PYTHON_MAJOR_V
ERSION/lib-dynload
find $BLENDER_ROOT_PATH/dependencies -name vrpn.so -exec ln -s {} . \;
```

Creation of scripts to run blender and blenderplayer

Due to shared library compilation of python, we will require specific scripts to load blender.

```
mkdir -p $BLENDER_ROOT_PATH/bin
cat > $BLENDER_ROOT_PATH/bin/blender << EOF
#!/bin/bash

export LD_LIBRARY_PATH="$BLENDER_ROOT_DEPENDENCIES/lib:\$LD_LIBRARY_PATH"

exec $BLENDER_ROOT_PATH/compile/blender-
$BLENDER_VERSION/build/bin/\$(basename \$0) "\$@"
EOF
chmod a+x $BLENDER_ROOT_PATH/bin/blender
ln -s $BLENDER_ROOT_PATH/bin/blender $BLENDER_ROOT_PATH/bin/blenderplayer
```

Finally, to run blender or blenderplayer, you only need to run:

```
/usr/local/blender/2.72/bin/blender
```

or

```
/usr/local/blender/2.72/bin/blenderplayer
```

You can copy or link (ln -s) both executable to any folder already in your path for easier access. Their is no more need to load variables with the `/usr/local/blender/2.72/compile/.vars` script. As usual, replace `/usr/local/blender/2.72` by the blender root path you have choosen.

Re-owning for all users

You can re-own all the tree to allow any user of the computer to use this blender compilation:

```
sudo chown -R root:root $BLENDER_ROOT_PATH
```

Moreover, it reduces the chances to accidentally modify files inside this (now sacred) tree 😀.

Adding blender in the main system binary path

Don't do this if you already have a system wide installation of blender (for instance through system package installation process (`yum`, `rpm`, `apt-get`, `dpkg` ...)

You can also add links inside main PATH to use blender:

```
sudo ln -s $BLENDER_ROOT_PATH/bin/* /usr/bin
```