SimulaVR Documentation

Release 0.0.0

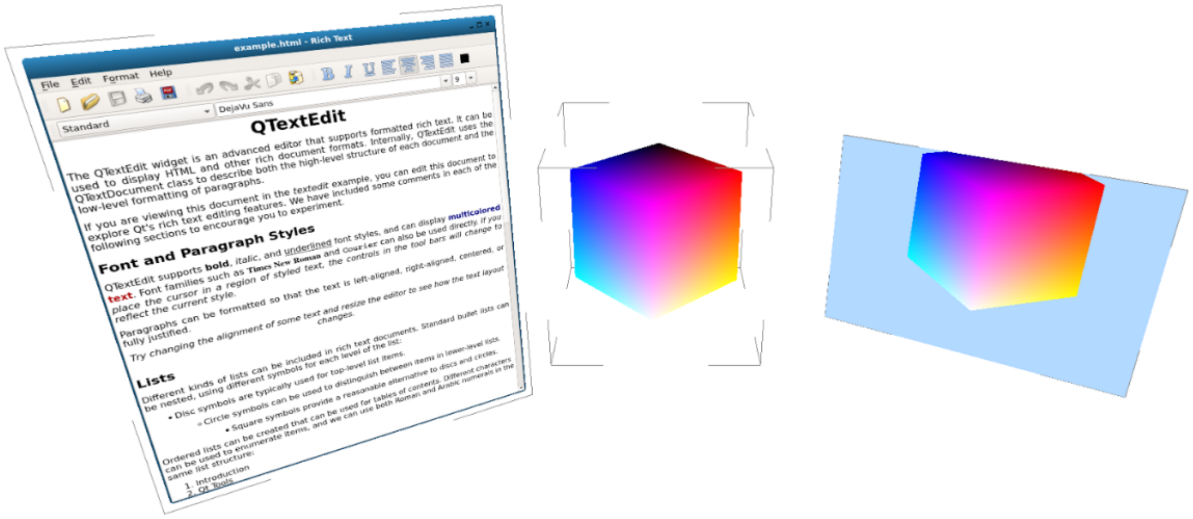
Simula Team

Feb 08, 2018

Users Guide

- *Users Guide*
- *Building Simula*
- *Developer Documentation*

Here's a screenshot showing three different wayland client applications running under the simulavr weston compositor.



QTextEdit

The QTextEdit widget is an advanced editor that supports formatted rich text. It can be used to display HTML and other rich document formats. Internally, QTextEdit uses the QTextDocument class to describe both the high-level structure of each document and the low-level formatting of paragraphs.

If you are viewing this document in the *textedit* example, you can edit this document to explore Qt's rich text editing features. We have included some comments in each of the following sections to encourage you to experiment.

Font and Paragraph Styles

QTextEdit supports **bold**, *italic*, and underlined font styles, and can display **multicolored** **text**. Font families such as Times New Roman and Courier can also be used directly. *If you place the cursor in a region of styled text, the controls in the tool bars will change to reflect the current style.*

Paragraphs can be formatted so that the text is left-aligned, right-aligned, centered, or fully justified.

Try changing the alignment of some text and resize the editor to see how the text layout changes.

Lists

Different kinds of lists can be included in rich text documents. Standard bullet lists can be nested, using different symbols for each level of the list:

- Disc symbols are typically used for top-level list items.
  - Circle symbols can be used to distinguish between items in lower-level lists.
    - Square symbols provide a reasonable alternative to discs and circles.

Ordered lists can be created that can be used for tables of contents. Different characters can be used to enumerate items, and we can use both Roman and Arabic numerals in the same list structure:

1. Introduction
2. Qt Tools

CHAPTER 1

SimlaVR Usage

To use *simulavr* you need to start an *osvr_server* to read the controller data. See the section, *OSVR Server*, below for more details on using and configuring the *osvr_server*.

SimulaVR can be tested with and without additional hardware. Current limitations include the following:

- Headset rendering is non-existent at the moment. Maybe you can help, see *Headset Rendering Status* for more ways to help.
- To get feedback on the headset movement when all hardware is connected and seen by *osvr_server* you must pass the *-w* or *–waitHMD* flag to *simulavr* to properly wait for the connection to *osvr_server*.

Note: When everything is working properly you will see a boring white window. You should launch some programs for testing. The developers' goto test program is `weston-terminal`.

1.1 With Nix

You will need two terminals to launch this compositor. In the first terminal, you must launch the *OSVR Server*.

```
nix-shell ./shell.nix
```

In the second terminal, launch *simulavr*:

```
stack --nix --no-exec-pure exec -- simulavr [-h -w]
```

Note: You will need *nvidia-381.26.13* (or higher) video drivers. Unfortunately, nix cannot provide those, so you will have to get them from your system's package manager.

1.2 Simply Stack

Vanilla baseline testing and executing by only using *stack* along with system libraries.

```
stack exec simulavr
```

Note: When run with no arguments *simulavr* will not track the headset. It will update controller positions, but not the headset.

If you have everything working and want to sync with the headset as well as the controllers use the following snippet.

```
stack exec -- simulavr -w
```

1.3 OSVR Server

```
osvr_server ./config/ViveExtendedMode.json # or use ViveDirectMode.json for
direct mode
```

CHAPTER 2

How to Contribute

We're looking for open-source contributors. If you're interested in using Haskell to (cleanly) bring VR and Linux together, but don't have an HTC Vive, PM or email me at georgewsinger@gmail.com.

You can also see the GitHub Issues for a list of ways to immediately contribute. Issued may be tagged as *new contributor* if they are especially appropriate for people getting adjusted to the code. In addition, *Developer TODOs* sketches some additional issues with the code that may be solved via future contributions.

Important: All pull requests to this repo implicitly consent to, and will be subsumed under the terms of the Apache 2.0 license. See *License Details*.

CHAPTER 3

License Details

CHAPTER 4

Git Help

Before you begin you must initialize and update the submodules in the *SimulaHS* repository. This process ensures that you are in sync with previously checked in combinations of commits between the two projects. More information on submodules is [here](https://git-scm.com/book/en/v2/Git-Tools-Submodules).

```
git submodule update --init --recursive
```

CHAPTER 5

Building with Nix

The easiest way to build Simula is to install *nix* and run:

```
stack --nix build --ghc-options="-pgmcg++ -pgmlg++"
source ./swrast.sh # only needs to be run once
```

Nix automatically downloads every non-Haskell dependency for this project and places them in */nix/store* in such a way that they don't conflict with your current distro's libraries. Running *stack* with these flags tells it how to find these libraries. The *swrast.sh* script tells nix how to find your system's OpenGL drivers.

If you don't already have *nix* installed, you can get it from your distro's package manager, or run

```
curl https://nixos.org/nix/install | sh
```

To use *simulavr* see *SimlaVR Usage*.

CHAPTER 6

Ubuntu Build Process

The process to try out *Simula*, is as simple as 1, 2, 3, 4 with only minor software compliation in between. You will need to install software from your linux distributor and have your machine whirl and buzz for minutes on end with no output to test this collection of Haskell, C, and C++ code. If you are up for the challenge then you can find most of a recipie below. Many of the instructions have only been attempted once, if any at all, and can only be cast as a guide to get running. With those precautions aside, here's what someone got working once, sometime, probably.

1. Clone all submodules, see *Git Help*
2. Build and Install requisite packages for your system
3. Build SimulaHS `stack build --extra-lib-dirs="${HOME}"/.local/lib --extra-include-dirs="${HOME}"/.local/include`
4. Execute the program and play around: *SimlaVR Usage*

6.1 Ubuntu 17.10 (Artful) Packages

Install all the dependecies in one shot with the following script:

```
sudo apt install \
    g++ \
    automake \
    autoconf \
    autoconf-archive \
    make \
    cmake \
    libtool \
    pkg-config \
    binutils-dev \
    libegl1-mesa-dev \
    libgles2-mesa-dev \
    libxcb-composite0-dev \
    libxcursor-dev \
    libcairo2-dev \
    libpixman-1-dev \
    libgbm-dev \
    libmtdev-dev \
    libinput-dev \
    libxkbcommon-dev \
    libpam0g-dev \
    libgflags-dev \
    libgoogle-glog-dev \
    libssl-dev \
    libdouble-conversion-dev \
    libevent-dev \
    libboost-context-dev \
    libboost-chrono-dev \
    libboost-filesystem-dev \
    libboost-iostreams-dev \
    libboost-locale-dev \
    libboost-program-options-dev \
    libboost-regex-dev \
    libboost-system-dev \
    libboost-thread-dev \
    libsdl2-dev \
    libopencv-dev \
    libjsoncpp-dev \
    libxml2-dev \
    libusb-1.0-0-dev \
    libspdlog-dev \
    libeigen3-dev
```

CHAPTER 7

Obtaining Source Dependencies

1. wayland-protocols - [https://github.com/wayland-project/wayland-protocols]

Currently HEAD is fine, but any tag >= 1.7 will work fine.

2. libweston - [https://github.com/wayland-project/weston]

You will need to checkout tag *2.0.0*.

3. libfunctionality - [https://github.com/OSVR/libfunctionality]

Currently HEAD is fine, previous versions not tested (2017-07-20).

4. folly - [https://github.com/facebook/folly]
5. OSVR-Core - [https://github.com/OSVR/OSVR-Core]

Currently HEAD is fine, v0.6 does not build on debian stretch.

7.1 Per Repository Notes

7.1.1 Dependencies for *wayland-protocols*

There are no special dependencies, but a pro-tip, run *configure* with a custom **PREFIX** and copy the *wayland-protocols.pc* file to **PREFIX**/*lib/pkgconfig* and set **PKG_CONFIG_PATH** to the same.

Example:

```
./configure --prefix="$HOME"/.local
make && make install
cp wayland-protocols.pc "$HOME"/.local/lib/pkgconfig
export PKG_CONFIG_PATH="$HOME"/.local/lib/pkgconfig
```

7.1.2 Dependencies for *libweston*

Make sure you have installed *wayland-protocols* before proceeding to building *libweston*.

1. EGL - libegl1-mesa-dev
2. glesv2 - libgles2-mesa-dev
3. xcb-composite - libxcb-composite0-dev
4. xcursor - libxcursor-dev
5. cairo-xcb - libcairo2-dev
6. automatically install by libcairo2-dev - libpixman-1-dev
7. gbm - libgbm-dev
8. mtdev - libmtdev-dev
9. libinput - libinput-dev
10. xkbcommon - libxkbcommon-dev
11. pam - libpam0g-dev

After installing the above packages you can configure and build *libweston*. Here is a recipie for success.:

```
git checkout -b v2.0.0 2.0.0
./autogen.sh
./configure --prefix="$HOME"/.local --disable-setuid-install
make && make install
```

You will see a notice about needing to set **LD_LIBRARY_PATH** and also for setting **LD_RUN_PATH** to use these newly installed libraries. You may want to set these in your *.bashrc* file or other shell startup file. For your interactive shell you can just use the following lines:

LIBDIR="$HOME"/.local/lib export LD_LIBRARY_PATH="$LD_LIBRARY_PATH":"$LIBDIR":"$LIBDIR"/libweston-2:"$LIBDIR"/weston export LD_RUN_PATH="$LD_RUN_PATH:"$LIBDIR":"$LIBDIR"/libweston-2:"$LIBDIR"/weston

7.1.3 Dependencies for *libfunctionality*

You will need *cmake* to build any of the projects from *OSVR*. When building *cmake* projects you should perform out-of-tree builds by creating a build directory and running *cmake* from that directory. For example you can repeat this pattern for any cmake project.:

```
mkdir $PROJECT-build
git clone $PROJECT_URI
cd $PROJECT-build
cmake ../$PROJECT
```

To set a custom PREFIX for cmake projects you need to use the following incantation. `cmake -D CMAKE_INSTALL_PREFIX="$HOME"/.local ../$PROJECT`

7.1.4 Dependencies for *folly*

1. boost-context - libboost-context-dev
2. boost-chrono - libboost-chrono-dev

3. boost-filesystem - libboost-filesystem-dev
4. boost-regex - libboost-regex-dev
5. boost-program-options - libboost-program-options-dev
6. boost-system - libboost-system-dev
7. boost-thread - libboost-thread-dev
8. gflags - libgflags-dev
9. google-glog - libgoogle-glog-dev
10. libssl - libssl-dev
11. double-conversion - libdouble-conversion-dev
12. libevent - libevent-dev

To build folly you need to run `autoreconf -ivf` from the folly subdirectory of the cloned repository.:

```
cd folly
autoreconf -ivf
./configure --prefix="$HOME"/.local
make && make install
```

7.1.5 Dependencies for *OSVR-Core*

To proceed ensure you have installed *folly*, *libfunctionality*, *libweston*, and *wayland-protocols* as described above.

1. sdl2 - libsdl2-dev
2. opencv - libopencv-dev
3. jsoncpp - libjsoncpp-dev
4. boost-thread - libboost-thread-dev
5. boost-locale - libboost-locale-dev
6. boost-filesystem - libboost-filesystem-dev
7. boost-program-options - libboost-program-options-deu
8. libusb - libusb-1.0-0-dev
9. libspdlog - libspdlog-dev

When fetching from github you must fetch the submodules and initialize them before attempting a build.

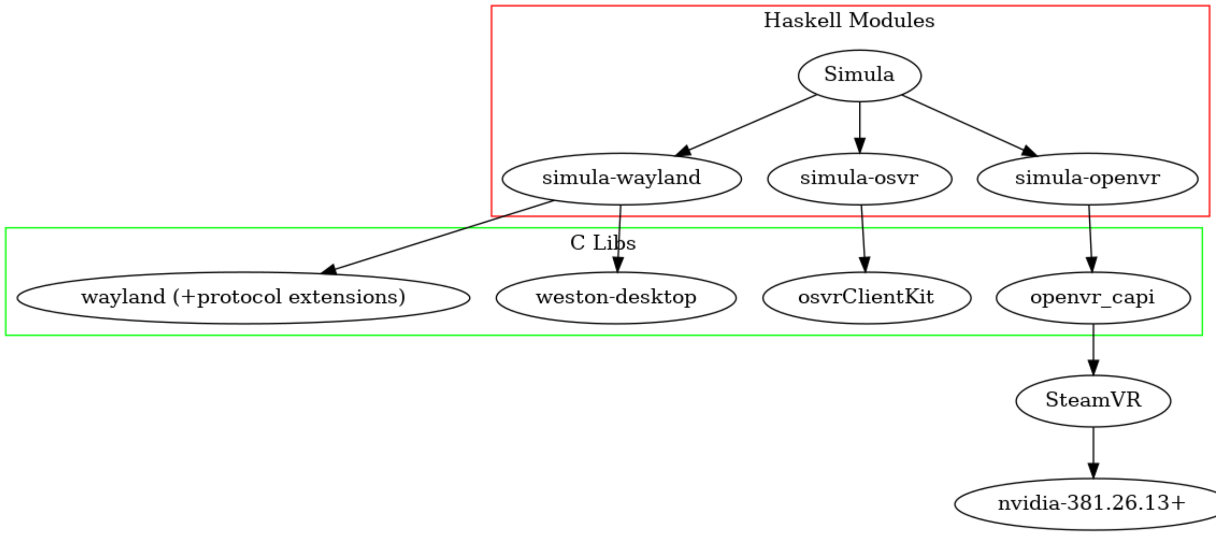
```
git submodule update --init --recursive
```

OSVR-Core is a *cmake* project so refer to the instructions above in the *libfunctionality* section to perform an out-of-tree build.

CHAPTER 8

Source Module Layout

Here is the project's dependency graph:



- The top level of this project hosts the primary Haskell modules.
- The embedded submodules *simula-wayland*, *simula-osvr*, and *simula-openvr* contain FFI bindings (via c2hs) that connect to their respective C libraries shown above.
- In order to run the vive-compositor, you will need *nvidia-381.26.13* or greater installed on your system, which (unfortunately) *nix* cannot provide. You do not need nvidia drivers, however, to run the *simulavr*.
- A list of technologies in use: Haskell, C/C++ (c2hs, inline-c), OpenGL, wayland & weston, OSVR, OpenVR, nix (for build dependencies).

CHAPTER 9

Development Status

Goal	Status	Short-Run Horizon	Long-Run Horizon
Basic, Launchable Compositor	DONE	X	
Wayland App Compatibility	DONE	X	
X Applications Compatibility		X	
HTC Vive Compatibility		X	
Usable VR Desktop		X	
Test Suite		X	
Clear Text Resolution in VR			X
Special-Purpose 3D Linux Apps			X
A "VR Linux Distro"			X
Standalone HMD Compatibility			X

9.1 Developer TODOs

This is a compilation of old TODO notes. Important items from this list should moved to Issues.

9.1.1 General

- Awful C++-esque typeclass structure. In most cases, *Some a* can be replaced with something better. However due to time/etc limitations, the current code architecture closely mirrors the simula_cpp one.
- hs-boot files due to the above. Annoying extra bookkeeping and should be phased out whenever possible.
- General stinginess with comments. In case something is unclear, chances are it's either unclear in the motorcar source and/or unclear to me (since it was ported from motorcar/simula_cpp). Ask anyways though.
- MVars are used whenever mutable state is required. This is not always needed, and I've attempted to reduce the amount of mutable references, but it's likely I missed some.

9.1.2 Simula.BaseCompositor

- .Event is basically unused. Weston handles all of that stuff.
- Geometry is mostly clean, apart from Rectangle being a misnomer (Rect is the actual Rectangle)
- OpenGL contains viewport code (relatively straightforward) and shader code (slightly suboptimal, as it compiled shaders too often; they should be cached)
- checkForErrors terminates the program on an OpenGL error with a backtrace; this can be deactivated by commentiong out the error line
- The scene graph is a big unholy mess of C++-esque code. It needs a massive refactoring.
- Gratitious use of RecursiveDo (aka mfix) to deal with (*x needs reference of y, and vice versa, to be constructed*) and superclass stuff. Could be solved by refactoring the above C++ mess.
- Generally, resources need to be cleaned up better. Consider ResourceT.
- The Weston surfaces currently don't have a type. This should be fixed.
- Weston.hs in general is a bit messy, and could use some cleaning up.
- The OSVR branch Weston.hs has modifications in the render loop and a reference to OSVR.hs' SimulaOSVR-CLient. It should be merged back into the main branch.

CHAPTER 10

Rendering Roadmap

With the HTC-Vive using osvr, **simulavr** can see and get reports on the location information for the two controllers and the headset. The main renderer is a wayland server and provides a full compositor. OSVR is also providing all the distortion information needed to render static images to the headset display.

10.1 Headset Rendering Status

Table 10.1: Missing Puzzle Pieces in Rendering

Feature to Implement	Implementation Status	Priority
Respond to Controllers	Partially implemented	Blocker
Respond to Buttons	Possible to implement	Low
Render Controllers	Needs Haskell	Normal
Viewport Responds to HMD	Needs Haskell	Blocker
Render Left Eye to Texture	Needs Haskell	Low
Render Right Eye to Texture	Needs Haskell	Low
Render to HMD Screen	library missing	High

10.1.1 HMD Rendering Notes

There are several libraries available for use in different aspects of VR. The two main libraries in use by Simula are OSVR and OpenVR. Both of these libraries are meant to be high level and provide a unified interface for dealing with inputs and headset rendering. *OpenVR <openvr-link>* depends on the SteamVR runtime library to work, but seems to be the most robust library for interfacing with the headset.

The OSVR library does a wonderful job of working with the HTC lighthouse devices and gives a callback registration interface for handling input devices. This works really well. It has the potential to scale to multiple headsets connected over a network. With all that said the headset portion of the library is still binary only and Linux support is non-existent at the moment. We have tried using OSVR-RenderManager and can't seem to get direct or extended mode to render to the headset.

The current plan is to move ahead trying to work with OpenVR and explore additional library options. OpenHMD is a potential library to integrate and test. There is the possibility of going more low-level and using another OpenGL context to just render textures. Another would be to simply get a dumb XOrg window and render the eye distortion textures there.

10.1.2 Evaluating Libraries

When evaluating a library for inclusion, there are a couple of criteria to keep in mind.

1. Does it work?
2. Can you write a small example Haskell program using it?

CHAPTER 11

Origins

Simula is a reimplementation fork of motorcar. To read about motorcar, see Toward General Purpose 3D User Interfaces: Extending Windowing Systems to Three Dimensions.