Bern University of Applied Sciences
Engineering and Information Technology

Bachelor Thesis 2009

CAVE Rendering Framework

User Manual

Division	CPVR
Authors	Brigitte Hulliger
	Jonas Walti
	Stefan Broder
Supervisors	Prof. U. Künzler
	Michael Luggen
Version	1.0

Date	Version	Name	Comment
05.06.2009	0.8	all	First draft for expert H. van der Kleij
09.06.2009	0.9	all	Draft for Supervisors
12.06.2009	1.0	all	Last corrections for delivery

Table 0.1: History

Contents

List of Figures

List of Tables

Listings

1 User Guide

1.1 Introduction

This guide should help developers, particularly from the CPVR group, to make use of the CAVE Rendering Framework (CRF). First of all, if not already done, there is a guide on how to setup the CRF environment. Basically, this means to install OpenSceneGraph (OSG) and Equalizer. There are a lot of dependencies and some tricks to make it work. For that reason, read and follow the instructions carefully.

In a second step, we provide some information on how to develop with the CRF. You are introduced on what basic implementations you have to make to build your own application based on the CRF.

1.2 Installation

The CRF was tested on two setups:

- ??
- ??

There are several dependencies you have to be aware of in order to setup a proper environment. Both setups have their own dependencies. They are all listed and described below.

1.2.1 Linux Ubuntu 8.10

We assume that you already have a running Ubuntu on your machine. Make sure you have the correct graphics drivers installed (i.e. nvidia-glx-dev for nvidia graphics cards). It is mandatory that you have the developer drivers installed (not the nvidia-glx only), because the CRF must be able to reference the header files. The CRF requires a running CMake, OSG and Equalizer setup.

CMake

Download	http://www.cmake.org
Minimal Version	CMake 2.6
Documentation	http://cmake.org/cmake/help/documentation.html

Table 1.1: CMake Installation

The CRF as well as OSG requires CMake. To install CMake on Ubuntu, you simply have to run the following command in a terminal:

```
% sudo apt-get install cmake
```

Listing 1.1: Install cmake on Ubuntu

To install CMake on other Linux distributions you can compile it from source. The instructions can be found online.

OSG

Download	http://www.openscenegraph.org/projects/osg/wiki/Downloads
Minimal Version	OSG 2.8
Documentation	http://www.openscenegraph.org/projects/osg/wiki/Support

Table 1.2: OSG Installation

It is currently not possible to install OSG as simple as CMake on Ubuntu since aptitude only includes OSG-2.2 which is too old for the CRF. Therefore, make sure you install OSG from source. OSG requires some libraries which have to be installed on the system before you can start compiling OSG itself. Make sure you have the following packages installed on your system:

fileformat	**unix library**	**ubuntu package**
tiff imges	libtiff	libtiff*-dev
jpeg images	libjpeg	
gif images	libungif	
png images	libpng	libpng*-dev
true type fonts	freetype	libfreetype*-dev
libcurl	libcurl	libcurl4-openssl-dev
mesa	libmesa	mesa-common-dev
freeglut	libglut	freeglut-dev

Table 1.3: Dependencies for OSG

Assuming you have gone through all prerequisists you can install OSG itself now. We suggest that you use the following file structure on *all* your workstations to ensure a proper setup. The CD attached provided with this manual contains an archive file `crf.zip`. This archive contains the source code of OSG as well as any needed data files. Copy the directory to your home directory:

```
% cp crf.zip ~/
% unzip crf.zip
```

Listing 1.2: CRF installation

If you have a proper CMake setup and all required libraries, you should be able to run `configure` and get a similar result as listed below:

```
% cd ~/crf/OSG-2.8/
% ./configure
 [...]
The build system is configured to instal libraries to /usr/local/lib
Your applications may not be able to find your installed libraries unless you:
    set your LD_LIBRARY_PATH (user specific) or
    update your ld.so configuration (system wide)
You have an ld.so.conf.d directory on your system, so if you wish to ensure
    that
applications find the installed osg libraries, system wide, you could install a
OSG specific ld.so configuration with:
    sudo make install_ld_conf

-- Configuring done
-- Generating done

% sudo make install_ld_conf
```

Listing 1.3: OSG configuration

If `configure` doesn't work or gets a different result than the one above, make sure you correct the errors before you advance. You can save yourself a lot of time!

To install OSG, simply run `make` as user:

```
% make
```

Listing 1.4: OSG installation

The installation of OSG may take up to two hours, depending on your hardware. So don't hesitate to get yourself a coffee now...
Assuming a proper `make`, run `make install` as root:

```
% sudo make install
```

Listing 1.5: Finish OSG installation

This copies the OSG libraries to the right directories on your system.

Setting System Variables

To finish your OSG setup you have to set some system variables. To do so, edit your `/etc/environments` as root. Add the following line to this file:

```
OSG_FILE_PATH=/home/<user>/crf/OSG-Data
```

Listing 1.6: /etc/environment

Equalizer

Download	http://www.equalizergraphics.com/downloads/major.html
Minimal Version	Equalizer 0.6
Documentation	http://www.equalizergraphics.com/documents/EqualizerGuide.pdf

Table 1.4: Equalizer Installation

Equalizer has to be installed from source too. Before you start compiling it, make sure you have installed the following packages on your system:

unix library	ubuntu package
bison	bison
flex	flex
libuuid	e2fsprogs

Table 1.5: Dependencies for Equalizer

After installing the prerequisits, simply run `make` and `make install` in the source folder of Equalizer:

```
% cd ~/crf/equalizer-0.6-rc1/
% make
% sudo make install
```

Listing 1.7: Equalizer installation

You can simply test your Equalizer installation by running the sample application `eqPly`, which is the sample application provided by Equalizer:

```
% cd /usr/local/bin
% eqPly
```

Listing 1.8: Test Equalizer by example application eqPly

A screenshot of the expected output of the `eqPly` and the `eVolve` application is attached in Appendix **??**.

SSH Setup

In order to use Equalizer on a distributed setup with multiple rendering clients, each client as well as the server necessarily have to run a SSH server. The server must be able to connect to each client without entering a passwort. Analogously, each client must be able to connect to the server. The easiest way to achieve this is using public key authentication. This can be set up as follows.

Step 1: If not already done, we have to create a key pair (public and private key) on each component:

```
% ssh-keygen
 Generating public/private rsa key pair.
 Enter file in which to save the key (/home/stefan/.ssh/id_rsa):
% <enter>
 Enter passphrase (empty for no passphrase):
% <enter>
 Enter same passphrase again:
% <enter>
```

Listing 1.9: SSH Key generation

We want neither to store the keys somewhere other than in the default directory nor want we to enter a passphrase. Therefore, just press <enter> without typing anything. In this case, we just care about the connection, not so much about the security provided by SSH.

Step 2: Now, the public key has to be added to the authorized_keys file on each node we want to connect to. Remember that this proceedings have to be done for both directions, from the node running the eqServer to each client and vice versa.

```
% scp ~/.ssh/id_rsa.pub <ip-of-host>:/home/<username>
# enter password
% ssh <ip-of-host> "cat␣id_rsa.pub␣>>␣~/.ssh/authorized_keys␣&&␣rm␣id_rsa.pub"
# enter password again
```

Listing 1.10: RSA-Key distribution

If everything worked fine, then you should be able to connect without entering a password from now on. If not, go back to the last step and ensure you followed consequently. To test the connection enter

```
% ssh <ip-of-host>
```

CRF

Change to the CRF directory where the CMakeLists.txt file resides. There, type the following commands.

```
% cmake .
% make
% sudo make install
```

Listing 1.11: CRF installation

If you experience any errors after the `cmake` command, you most probably forgot to follow one of the previous instructions. Otherwise, you installed the CRF on you system. If you use a distributed system with multiple clients, you can distribute the binaries and header files using the shell script distributed with the sources.

Be careful, if you want to use debug libraries. To do so, tell CMake by setting the SET_DEBUG_LIBS variable: `SET( USE_DEBUG_LIBS ON )`. Verify that CMake did choose the correct libraries (libnames of OSG end with d for debug). Be consistent, take eiter the debug libraries from both, OSG and Equalizer, or from none. This can be done using `ccmake`.

1.2.2 Windows XP

In order to install the CRF, you a build environment (e.g. Ms Visual C++). Furthermore, it is expected that you have installed the OpenGL library and header files. In the following, we guide you through the installation of CMake, OSG, Equalizer and, finally, the CRF.

CMake

Download	http://www.cmake.org
Minimal Version	CMake 2.6
Documentation	http://cmake.org/cmake/help/documentation.html

Table 1.6: CMake Installation

The CRF as well as OSG require CMake to be built. To install CMake on Windows XP, just visit the CMake and follow the install instructions.

Generally, to build the desired project file for Visual C++ using CMake you most probably want to use the CMake GUI. Watch out for the CMakeLists.txt. Then, let CMake create the project files at the selected location.

OSG

Download	http://www.openscenegraph.org/projects/osg/wiki/Downloads
Minimal Version	OSG 2.8
Documentation	http://www.openscenegraph.org/projects/osg/wiki/Support

Table 1.7: OSG Installation

Visit the OSG website and look for the downloads section. Download the above mentioned version of OSG (this guide was tested with 2.8.0 and 2.8.1). You can either simply download and install the binaries or download the source code and compile it by yourself. Just be warned, compile

OSG can take up to two hours. Verify that you have fulfill all the requirements mentioned on the website. Now, unpack the zip archive.

If you have chosen to compile OSG by yourself, then go for it and compile it. In any case, you should install the OSG on your system. Make shure to set the path of the OSG data files in the PATH environment variable.

Equalizer

Download	http://www.equalizergraphics.com/downloads/major.html
Minimal Version	Equalizer 0.6
Documentation	http://www.equalizergraphics.com/documents/EqualizerGuide.pdf

Table 1.8: Equalizer Installation

If you use Equalizer with Windows you have to change a line in the Equalizer 0.6 sources, before compiling. In the Equalizer file `src/lib/client/event.h` you need to add `EQ_EXPORT` in front of the default constructor of the `Event` struct to make this constructor visible for the Equalizer.dll users (like the CRF). This change will be integrated in further Equalizer releases.

Now the source is ready to compile.

SSH Setup

In order to use Equalizer on a distributed setup with multiple rendering clients, you must run a SSH server on every client. This scenario was not tested with Windows clients. My suggestion is to emulate a linux using Cygwin[1] or equivalent. If you want to try, install Cygwin and continue with the "SSH Setup" chapter of Ubuntu. Again, this was not tested, so you are on your own here.

CRF

Finally, you should be able to install the CRF. Unzip the crf archive, use CMake to generate the project files, build and install it. Now you can take full advantage of the CRF.

Be careful, if you want to use debug libraries. To do so, tell CMake by setting the SET_DEBUG_LIBS variable: `SET( USE_DEBUG_LIBS ON )`. Verify that CMake did choose the correct libraries (libnames of OSG end with d for debug). Be consistent, take eiter the debug libraries from both, OSG and Equalizer, or from none. This can be done using the CMake GUI.

[1] http://www.cygwin.com/

2 Programming Guide

2.1 Overview

This section shows how the CRF can be used. This is achieved by showing some examples in a tutorial style. There are also some special features described here. To get a better overview of the used techniques, please consider the Technical Report. For an overview of all classes with its functions, take a look at the provided Application Programming Interface (API). The base classes of the `eqOsg` and the `crf` namespaces provide some basic functionality. When overriding the functions of theses classes, be careful not to lose a desired functionality. It is often useful and sometimes a must to call the original function of the base class in your reimplementation. To avoid already known errors, please consult the *Technical Report* (especially the *Limitations* section).

2.2 Prerequisites

All the following examples require a correct and complete installation of Equalizer, OSG and the CRF with well set environment variables and all necessary files available.

2.3 World Coordinate System

The coordinate system of Equalizer, used in all the Equalizer demo-applications and for the head transform matrix of Equalizer, differs from the one of OSG. OSG uses the z-axis as up-vector. In contrast, Equalizer uses the y-axis as up-vector. Therefore in Equalizer, the z-axis points away from the user. This is well known as left-handed system. To convert the OSG world coordinate system to the one of Equalizer, just use `eqOsg::Pipe::correctCoordSys(yourRootNode)` to rotate the scene graph. This function rotates the passed node 90° around the x-axis and 180° around the y-axis and returns the converted node.

The conversion to the behaviour of Equalizer makes it easier for further framework extensions like tracker support or similar.

2.4 Global Keystrokes

If none of the `eqOsg` event related functions are overridden (like in the following HelloWorld examples) there are several key bound actions available:

General Keys

- F2: toggle through the OSG overlay statistics (consider the OSG documentation **?** for further information)
- F3: enable/disable the Equalizer camera handling
- F4: enable/disable the CRF framerate counter
- F5: enable/disable the Equalizer drawing statistics (consider the Equalizer Programming Guide **?** for further information)
- F6: enable/disable the bottom-left info text

Camera Handling

There is a basic FPS[1]-like camera built in.

- w, Up-Arrow: move along the z-axis (forward)
- s, Down-Arrow: move along the z-axis (backward)
- a, Left-Arrow: move along the x-axis (to the left)
- d, Right-Arrow: move along the x-axis (to the right)
- Page-Up: move along the y-axis (upwards)
- Page-Down: move along the y-axis (downwards)
- Left-Mouse-Button & Mouse Movement: rotate the local coordinate system of the camera, thus its view matrix (all the above described axes do change accordingly, similar to navigate an aircraft)

2.5 HelloWorld

The most simple CRF application can be realised by these lines of code:

```
{
#include <crf/crfStarter.h>

//1. Create the main method, which calls the crfStart.run()
//method to start Equalizer and stuff
int main(int argc, char** argv)

  //create the CRF starter
  crf::crfStarter starter;

  //run the application
  starter.run(argc,argv);
}
```

To run this example with a custom Equalizer topology, start your Equalizer server with the desired configuration file. If multiple nodes are used, make sure that on every machine the HelloWorld application and all the OSG files are placed in the same directory.

[1]FPS - First Person Shooter: A video game genre with a common and well known camera controlling behaviour. The head is rotated by the mouse and the movement is done with keys.

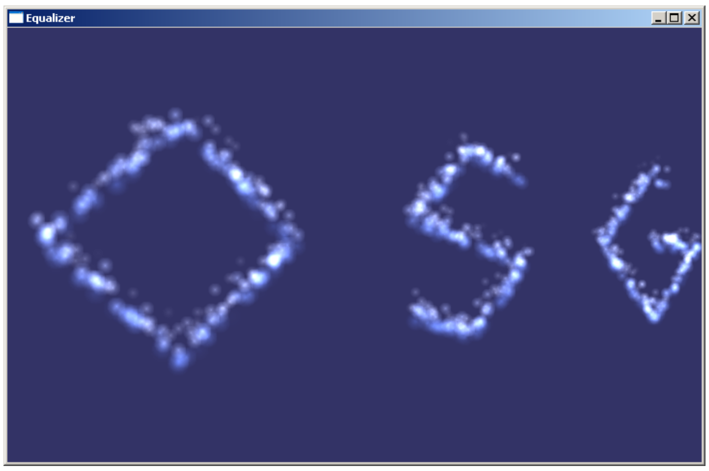

Figure 2.1: HelloWorld output with the standard Equalizer configuration

Background Information

Why does this work? The crfStarter class is a façade which initialises the whole Equalizer application. Thereafter, the default functions of the framework are called and if the environment variable `OSG_FILE_PATH` is set and the OSG file `osgcool.osg` is available there, a simple animated text "OSG" should appear on the screen(s). If this works, your parallel rendering system with Equalizer and OSG is set up correctly and you are able to continue.

2.6 Loading OSG Models

With a working HelloWorld example, you are now able to load any `.osg` file into this CRF application. To do this, run your compiled and fully linked HelloWorld example with the `–model=<filename>.osg` argument. Make sure that this model is available on every node in the same directory.

IMPORTANT: Several segmentation faults occurred when trying to use `osgDB::readNodeFile()` (like in the HelloWorld example) with multipipe configurations and just one running instance of the application. This is because `osgDB::readNodeFile()` messes up memory when executed multiple times. Therefore, we introduced a mechanism which does load the model just once and shares it with the different OSG viewers on the same machine (use the `-mode=pipesafe` argument when starting the shipped crfHelloWorld application). This did not accuron on multinode configurations with just one running instance per pipe.

Windows Example

To start the application with a desired model:

```
c:\HelloWorld\>HelloWorld.exe --model=cow.osg
```

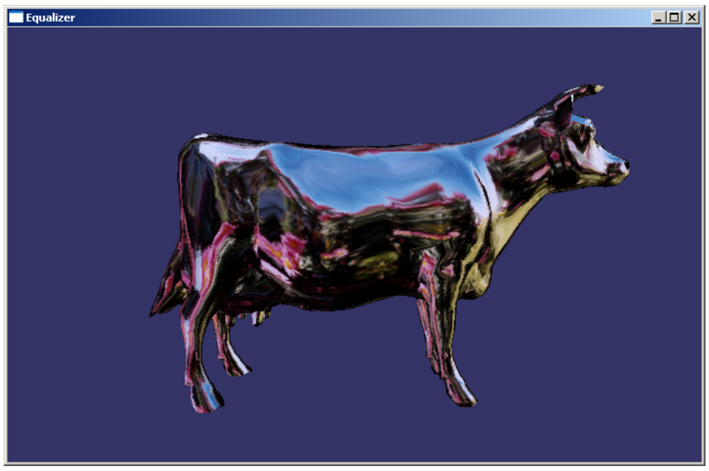

Figure 2.2: HelloWorld output with a model as command line argument

2.7 Custom Scene Graph

To use the whole power of the CRF and to create animated and dynamic scene graphs, the following steps have to be realised:

1. create your own pipe class, inherited from the `crfPipe` class
2. override desired functions like `createSceneGraph`, `updateSceneGraph` or others
3. create your own factory class which returns your custom objects, based on `crfNodeFactory`
4. pass your factory instance to the `crfStarter` using `crfStarter::setNodeFactory(yourFactory)` function
5. run the application with the `crfStarter.run` function

With this approach, the CRF user is able to override a lot of functions and therefore customise his CRF application. The following examples should provide an overview of some of the possibilities. To get a complete survey, consider the API and the Technical Documentation of the CRF and Equalizer.

2.8 Scene Graph Creation

A common way for creating custom scene graphs is:

1. Create a class with all your behaviours of your scene graph. This class provides a `getScene()` function which returns the root node of your scene graph.
2. in your custom `createSceneGraph()` function, add this root node to the OSG `_viewer->setSceneData(mySceneGraph->getScene())`

2.9 Animated Scenes

Example 1

To create an animation for your scene graph, a common CRF approach would be:

1. override the `crfPipe::updateSceneGraph` function to call the update function of your scene graph

```
void crfDemoApp::Pipe::updateSceneGraph(){
 //call the univers' update function
 myUniverse->update(getConfig()->getTime());
}
```

Listing 2.1: calling the updated function of your scene graph

Note: `GetConfig()->getTime()` calls the synchronous Equalizer time.

Example 2

A more straight-forward approach:

1. create a new member for the animation in your pipe class (a rotation node e.g.)
2. attach this node to your scene graph in your `createSceneGraph` implementation
3. attach the to-rotate-node to this rotation node
4. update this node in `updateSceneGraph()`

The `updateSceneGraph()` function gets called every time a new frame is drawn. As one can see, the `config->getTime()` function of Equalizer is used to rotate framerate independently. One could also use the `elapsedTime()` function provided by the OSG viewer but in that case, the time has to be reset when the first frame is drawn, because it is not granted that every clock of the OSG viewer starts at the same time in multinode and/or multipipe configurations. Remember: Every pipe has its completely independent OSG viewer!

```
void crfDemoApp::Pipe::updateSceneGraph(){
 osg::Matrix rotation = _rotateMatrix->getMatrix();
 rotation.makeRotate(getConfig()->getTime()*0.0024, osg::Vec3( 0., 0., 1. ) );
 _rotateMatrix->setMatrix(rotation);
}
```

Listing 2.2: the update function

2.10 Example: Porting An OSG Application

2.10.1 Overview

In order to give another example of the power of the CRF, we took the original OSG demo `osgkeyboard.cpp`, shipped with the OSG source code, and ported it to a fully functional CRF application. This example shows the ability of the CRF to handle common OSG based eventhandlers. Nothing has been changed in the eventhanlder class of this example.

2.10.2 Step 1: Small Example Changes

To obtain a better structured application, we exported all the class definitions of the `osgkeyboard.cpp` into a separate header file named `keyboard.h`. Now we are able to use all the classes of the example by including just this header file.

2.10.3 Step 2: Building The CRF Application

As mentioned before, we have to build up a more complex CRF application. To keep this demo simple, we do all this in the `main.cpp` file.

1. necessary inlcudes & namespaces:

```
#include <crf/crfPipe.h>
#include <crf/crfNodeFactory.h>
#include <crf/crfStarter.h>

#include "keyboard.h"

using namespace osg;
using namespace std;
```

Listing 2.3: setting up the include directives

2. define a namespace and declare our custom pipe and node factory classes:

```
///to avoid conflicts with naming, create a own namespace for the concrete
     demo
namespace crfDemoApp {

///we create our own pipe by inheriting from the crfPipe
class Pipe : public crf::crfPipe {
  public:
       Pipe(eq::Node* parent) : crf::crfPipe(parent){
         _sceneGraphCreated=false;
       }
    protected:
       ///Our own method to create the scene graph
       void createSceneGraph();
};

///Create a custom-nodefactory to create our own custom pipe-object
class NodeFactory : public crf::crfNodeFactory {
  protected:
    virtual eq::Pipe* createPipe(eq::Node *parent){
    return new crfDemoApp::Pipe(parent);
  }
};
```

Listing 2.4: declaring the custom classes

3. Reimplement the `createSceneGraph()` function. The custom OSG eventhandler, implemented exactly as in the original OSG example, is linked to the viewer of the pipe. Thereafter, the rootnode of the keyboard is converted to the Equalizer coordinate system by passing the node to the `correctCoordsys()` function, which returns the node, rotated 90° around the x-axis and 180° around the y-axis to fit the commonly used coordinate system of Equalizer.

```
void crfDemoApp::Pipe::createSceneGraph(){
  osg::ref_ptr<keyboard::KeyboardModel> keyboardModel = new keyboard::
       KeyboardModel();

  _viewer->addEventHandler(new keyboard::KeyboardEventHandler(
       keyboardModel.get()));
  _viewer->setSceneData( correctCoordsys(keyboardModel->getScene()));
  _sceneGraphCreated=true;
}
```

Listing 2.5: the createSceneGraph function

2.10.4 Step 3: The Main Function

1. Create a `crfStarter` object, pass the custom factory and run the application.

```
int main(int argc, char** argv)
{
  //create the demo factory
  crfDemoApp::NodeFactory* fac = new crfDemoApp::NodeFactory();

  //create the CRF starter
  crf::crfStarter starter;

  //pass the factory to the starter
  starter.setNodeFactory(fac);

  //run the application
  starter.run(argc,argv);
}
```

Listing 2.6: the main function of the ported OSG example

2.10.5 The Result

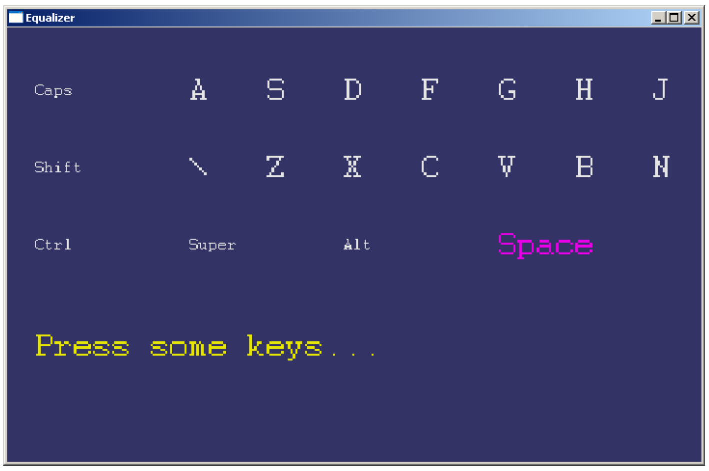

Figure 2.3: screenshot of the ported OSG keyboard example

By the way: This example can easily be used to check which key events are passed to the OSG viewer, because the eventhandler highlights the pressed keys.

A Equalizer Example Output

A.1 Equalizer Test Applications

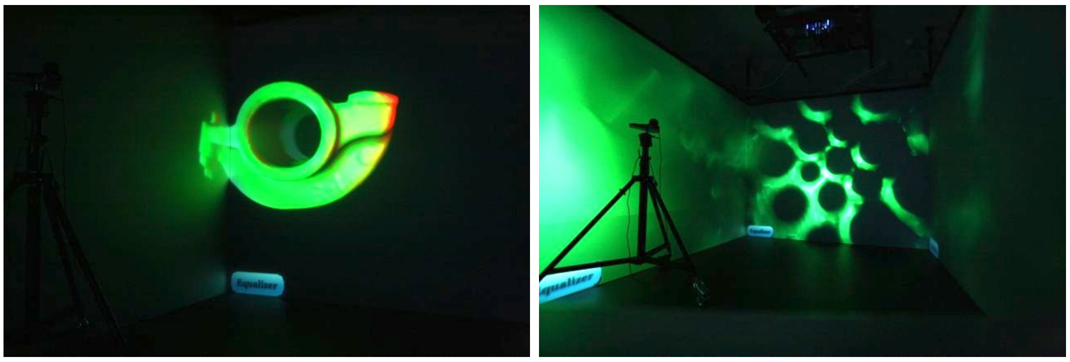

Figure A.1: Equalizer Test Applications