Okra

Erik Winkels

September 2, 2009

1 Definitions

Whenever I refer to directories, file names or commands issued on the computer they will be written in `mono-space and verbatim`.

2 Introduction

2.1 Goal

Write software that automatically generates Common Lisp (CL) bindings for Ogre.

Ogre is a 3D graphics library written in C++. This means that we'll have to generate code for CFFI (my chosen FFI) in CL and we'll have to generate C wrapper code for Ogre's C++ classes and methods since talking directly to C++ is riddled with problems (which will not be further discussed in this document).

Both the generated C++ code as well as the CL code should be human readable and produce as little warnings as possible when compiling! (When having to choose between the generated code being readable and the code of the bindings generator being readable I have given the former priority. I'm not so sure whether it's a good choice but time will tell...)

The bindings should be a usable subset of Ogre's API, enough to create a game. Due to various reasons (time, man power, differences between C++ and CL) it is probably not possible to ever automatically generate complete bindings to the Ogre API.

At the time of this writing I am not anymore working exclusively on Okra. I do use Okra in my private projects and through that it will slowly grow.

2.2 Running the examples

What you need and should know to get the examples running.

2.2.1 General

The start-up scripts (the `*.sh` and `*.bat` files) expect to be run from the `examples` directory.

2.2.2 Linux

All examples expect Ogre to be installed so download the source from: `http://www.ogre3d.org/download/source` (you'll need "OGRE 1.6.x Source for Linux / OSX").

Configure, compile and install in `/usr/local`. Ogre has some dependencies which need to be installed first. Also, see here for more information on the whole process: `http://www.ogre3d.org/wiki/index.php/Building_From_Source#Linux`

The Linux start-up scripts are written and tested using SBCL (1.0.25.debian at the time of writing) and expect to find `sbcl` in the path.

They expect to find the necessary Ogre libraries in `/usr/local/lib/OGRE` and all others in the `lib` directory of the Okra distribution.

All these options are set in the `*.sh` files and can be changed there.

2.2.3 Windows

All examples depend on the Ogre SDK to be installed so download it from: `http://www.ogre3d.org/download/sdk` (you'll need "OGRE 1.6.x SDK for Code::Blocks + MinGW C++ Toolbox").

Copy the following libraries from `<ogre-sdk>/bin/Release` to the Okra `lib` directory: `cg.dll`, `OgreMain.dll`, `OIS.dll`, `Plugin_CgProgramManager.dll`, `Plugin_OctreeSceneManager.dll`, `RenderSystem_Direct3D9.dll` and `RenderSystem_GL.dll`.

The Windows start-up scripts are written and tested using Clozure CL and expect to find `wx86cl` in the path.

All libraries are expected to be found in the `lib` directory of the Okra distribution.

All these settings are in the `*.bat` files and can be changed there.

2.2.4 Compiling libokra yourself

If the libokra library that comes with the Okra distribution doesn't work you'll need to compile it yourself. You'll need CMake for that: `http://www.cmake.org/cmake/resources/software.html`

If you have installed CMake on Linux you should issue: `cmake -G "Unix Makefiles"`

If you're on Windows you should read the `??` chapter.

2.2.5 Examples

Start with the simplest example first: `simple-okra`.

If that works try the `physics-and-input` example. This example depends on both Buclet and clois-lane which you can find at:

- `http://common-lisp.net/project/buclet/`
- `http://common-lisp.net/project/clois-lane/`

Make sure both are installed correctly and if you're on Windows copy over the DLLs from both the Buclet and clois-lane `lib` directories to the Okra `lib` directory. If you're on Linux copy over the `libclois-lane.so` file from the clois-lane `lib` directory to Okra's `lib` directory.

If you don't want to copy over library files for the above examples you'll need to make sure the OS can find them. (By adapting the example start-up scripts for example.)

3 Executables

There are a couple of things to keep in mind when generating an executable using Okra from your CL implementation.

3.1 Foreign Callbacks

On all the implementations I have tested the callbacks from C to CL had to be reloaded. The (unexported) `initialise-callbacks` functions take care of this.

3.2 Foreign Libraries

For some CL implementations you need make sure the foreign libraries are reloaded. Of the implementations I have tested this needed to be done for CCL on Windows. This is done using an init script which is supplied when saving the image.

In the libraries (unexported) `load-foreign-libraries` functions are provided for this. (And also `load-libcegui` in the `okra-bindings` package, I have to clean that up.)

4 Okra Bindings Generator

4.1 Generating and testing new bindings

My modus operandi when working on the bindings generator is from within Emacs. I start up Slime with `M-x slime` and change the current directory to that of the bindings generator with `M-x slime-cd`.

Once I made some changes to the source I do a `(asdf:oos 'asdf:load-op :okra-bindings-generator)` in Slime and place the files in the Okra root directory with `bin/rsync-generated-bindings.sh` (this is from within the `bindings-generator` directory).

I use rsync because it will only replace the files that have been changed so that `make` and the CL implementation only have to recompile those changed files which saves a lot of time. Sometimes this leads to problems however and I do `bin/install-generated-bindings.sh` which replaces all the bindings files. (It also deletes the `.fasl` files which shouldn't be really necessary.)

Then I issue a `make` from the Okra root directory to create the new C wrapper libraries and after that I run one of the examples. (`./physics-and-input.sh` from the `examples` directory at the time of writing since I do not have regression tests yet.)

5 Optimisations

5.1 Code

Passing a vector (#(a b ...)) is *much* slower than passing the values as seperate arguments. (At least on SBCL 1.0.25.debian.)

5.2 Implementations

SBCL is quite a bit faster than CCL for me on Windows. At the time of writing (2009–09-02) this was with SBCL 1.0.22 and a recent SVN checkout of CCL.

6 Windows (MinGW)

If your libs directory is empty you'll either to download them from the homepage or you can try to compile them yourself in which case you'll need to download and install MinGW, MSYS and CMake:

CMake

`http://www.cmake.org/cmake/resources/software.html#latest`

MinGW

`http://sourceforge.net/project/showfiles.php?group_id=2435\&package_id=240780`

MSYS

`http://sourceforge.net/project/showfiles.php?group_id=2435\&package_id=24963`

`http://sourceforge.net/project/downloading.php?group_id=2435\&filename=MSYS-1.0.10.exe\&a=14697658`

Install MinGW first and then MSYS. Use MSYS version 1.0.10. There's guides and information on `http://www.mingw.org/` if needed. (`http://www.mingw.org/wiki/HOWTO_Install_the_MinGW_GCC_Compiler_Suite`)

I suggest first trying the libraries that come with Okra and only then the MinGW route unless you're familiar with the latter.

You will need the DLLs from the OgreSDK. You can download the SDK at: `http://www.ogre3d.org/download/sdk` You need the Code::Blocks + MinGW version (which is currently at version 1.6.1 but this doesn't matter). You will need at least the following DLLs (from `bin/Release`):

- `cg.dll`
- `OgreMain.dll`
- `OIS.dll`
- `Plugin_CgProgramManager.dll`
- `Plugin_OctreeSceneManager.dll`
- `RenderSystem_Direct3D9.dll`
- `RenderSystem_GL.dll`

and ofcourse: `libokra.dll` (and `libclois-lane.dll` if you want to run the demo that depends on clois-lane). These last two you will need to compile for yourself.

The CMake scripts do not find Ogre correctly on MinGW yet so you'll have to check `CMakeLists.txt` in the Okra directory and change it to point to the correct locations (the two lines below "if (MINGW)").

If all of this is done you can issue: `cmake -G "MSYS Makefiles"` (not "MinGW Makefiles"!)

And then: `make`.

And if everything goes right (probably not) it should end up in the `lib` directory.

7 FAQ

Q. Why are both LOOP and ITER (package ITERATE) being used?

A. When I started programming Common Lisp I preferred ITER, but lately
I've come to appreciate LOOP, if only because it means less dependencies when writing libraries for public consumption. Ofcourse,
this argument isn't of much use when using both in a library.