Simulation-Based Engineering Lab
University of Wisconsin-Madison
Technical Report TR–2016–09

Overview of **Chrono** Build Process and Infrastructure

Conlain Kelly, Colin Vanden Heuvel

September 12, 2016

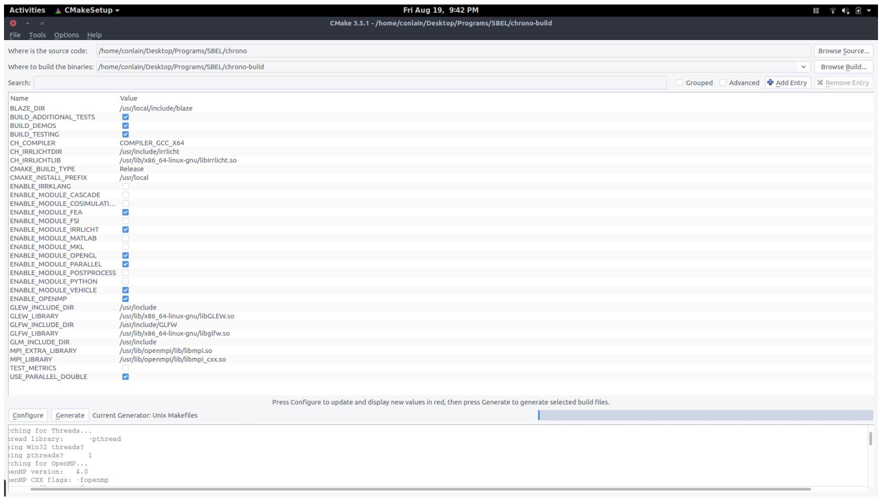
1 Introduction

This report presents a brief overview of the build and install process for the Chrono software infrastructure [1, 8], including its automated testing infrastructure.

2 Installation

The Chrono is built through a 3-step process:

1. Chrono source code is cloned from the Project Chrono Git repository [7].
2. CMake [5] is run with appropriate flags to configure[1] the build and install process.



Where is the source code: /home/conlain/Desktop/Programs/SBEL/chrono

Where to build the binaries: /home/conlain/Desktop/Programs/SBEL/chrono-build

Search:

Name	Value
BLAZE_DIR	/usr/local/include/blaze
BUILD_ADDITIONAL_TESTS	✓
BUILD_DEMOS	✓
BUILD_TESTING	✓
CH_COMPILER	COMPILER_GCC_X64
CH_IRRLICHTDIR	/usr/include/irrlicht
CH_IRRLICHTLIB	/usr/lib/x86_64-linux-gnu/libIrrlicht.so
CMAKE_BUILD_TYPE	Release
CMAKE_INSTALL_PREFIX	/usr/local
ENABLE_IRRKLANG	☐
ENABLE_MODULE_CASCADE	☐
ENABLE_MODULE_COSIMULATI...	☐
ENABLE_MODULE_FEA	✓
ENABLE_MODULE_FSI	☐
ENABLE_MODULE_IRRLICHT	✓
ENABLE_MODULE_MATLAB	☐
ENABLE_MODULE_MKL	☐
ENABLE_MODULE_OPENGL	✓
ENABLE_MODULE_PARALLEL	✓
ENABLE_MODULE_POSTPROCESS	☐
ENABLE_MODULE_PYTHON	☐
ENABLE_MODULE_VEHICLE	✓
ENABLE_OPENMP	✓
GLEW_INCLUDE_DIR	/usr/include
GLEW_LIBRARY	/usr/lib/x86_64-linux-gnu/libGLEW.so
GLFW_INCLUDE_DIR	/usr/include/GLFW
GLFW_LIBRARY	/usr/lib/x86_64-linux-gnu/libglfw.so
GLM_INCLUDE_DIR	/usr/include
MPI_EXTRA_LIBRARY	/usr/lib/openmpi/lib/libmpi.so
MPI_LIBRARY	/usr/lib/openmpi/lib/libmpi_cxx.so
TEST_METRICS	☐
USE_PARALLEL_DOUBLE	✓

Press Configure to update and display new values in red, then press Generate to generate selected build files.

Current Generator: Unix Makefiles

```
ching for Threads...
read library:      -pthread
ing Win32 threads?
ing pthreads?      1
ching for OpenMP...
enMP version:   4.0
enMP CXX flags: -fopenmp
```

Figure 1: The CMake GUI

3. The configured source code is built using either the GCC, Clang, or Windows Visual Studio compiler, which produces platform-specific libraries and executable binaries.

2.1 Installers

As an alternative to manually building Chrono from source, there are currently installers available for installation of Project Chrono, one for Windows and one for RedHat, Fedora, or CentOS Linux users.

[1]CMake allows users to configure their build and select optional modules to install via flags that are set either from a graphical interface(Figure 1) or a command line. The ability to pass CMake flags via command line is essential to the automated buildtesting infrastructure (Section 4).

2.1.1 Windows Installer

The Windows installer for Project Chrono is a self-extracting executable that triggers a command-line interface, allowing users to select their build configuration and location. It also allows users to use prepackaged versions of dependencies or use their own installations of said dependencies. The installer the copies over precompiled libraries and executables as well as header files for each module selected.

2.1.2 Linux RPM

Project Chrono now provides a YUM repository of precompiled binaries for RPM-based linux distributions. Official support for the packages is limited to RHEL/CentOS 7 64-bit, but any EL7-derived distribution (Scientific Linux, ROSA Enterprise Linux Server, Oracle Linux, ClearOS) should support the repository out-of-box (OOB). Rather than going through the process of cloning and building from the Project Chrono git repository, users of Project Chrono can install the repository on their system using the following rpm command.

```
rpm -i http://projectchrono.org/repos/projectchrono-repo-0.1.0-1.noarch.rpm
```

Further components can be installed using their native package manager. The repository provides a core Project Chrono package and number of different installation targets that are designed to provide the same functionality found in a chrono module. These subpackages include commonly used features such as Chrono::Parallel or Chrono::Vehicle and are denoted using the Addon Package naming convention defined by the Fedora Project (e.g. projectchrono-parallel). Even less-savvy RPM users should find the installation process to be familiar.

```
yum install projectchrono projectchrono-parallel projectchrono-vehicle []
```

Users who wish to build their own software against Project Chrono can install development headers for the components of their choice using another familiar convention: by appending the devel suffix to the package name of their choice.

```
yum install projectchrono-parallel-devel []
```

One distinct advantage of this method of installation is the ability for a new or veteran user to obtain a consistent and tested copy of Project Chrono that works OOB. Our goal for this software was to provide a method for a potential Chrono user to go from the initial download to running a completed demo in fifteen minutes or less. This particular set of packages provides such a user with an ongoing platform to work from the early stages of a Project Chrono application to the publication of production-quality software.

This is not to say that RPM installation should become the de facto method for obtaining Project Chrono on Linux. In fact, there are some limitations that can only be overcome with a source build. Certain modules and features such as Intel MKL support are only available when configured from CMake. In addition, the provided package is configured to utilize a broad range of hardware, and does not utilize the AVX optimizations that are available on systems with more recent CPUs. Perhaps its greatest limitation doesnt have to do with Chrono itself but with typical YUM repositories; proprietary software such as NVIDIAs

CUDA Toolkit is not always readily available in RPM package form for distrubutions that are not RHEL-derived.

To remedy such issues, a number of additional dependencies have been packaged and included in the Project Chrono repository. By checking at install-time for the presence of the CUDA Runtime Library libcudart, for example, it is possible to circumvent the need for the non-standard CUDA RPM. In addition, the Project Chrono yum repository contains fixes for outdated CMake versions common in RHEL-derived distributions, and less dependencies such as libGLFW or the header-only Blaze math library.

These packages may prove themselves to be a double-edged sword, however, as they may require additional attention on behalf of the Project Chrono maintainers.

3 Build Automation

There are currently 2 automated build-testing systems in use for Project Chrono, Travis CI [9] and BuildBot [3]. Travis is a limited system that only uses Linux machines and can only use open-source packages. Buildbot is a more robust system that supports various platforms and configurations but requires more configuration to provide full functionality.

3.1 Travis CI

Travis CI is a service free to Open-Source projects that uses a Linux cluster to automatically build code after each push to its Git repository. Travis builds Project Chrono under 4 different configurations to test several modules. It is most useful for testing builds with multiple configurations, but is less useful for testing

3.2 BuildBot

Buildbot is a Python [6] script that is triggered by a push to the central Project Chrono repository that runs a typical build sequence on 3 different build machines: one with Arch Linux, one with Windows 10, and one with OSX El Capitan. After the build finishes, the Buildbot script triggers sets of metrics, validation, and unit tests. Each test's results are reported on a dedicated site [2] and failed builds can be configured to trigger a warning email to be sent to the developer who committed the faulty code.

3.3 Testing Infrastructure

There are current 3 kinds of tests in use to ensure reliability and performance from Chrono: Metrics tests, Validation tests, and Unit tests. Metrics tests return results that are tracked over time, whereas Validation and Unit tests are simply pass/fail.

3.3.1 Metrics tests

Metrics tests track the performance of Project Chrono over time, and are currently run only on pushes to the main development branch of the Project Chrono repository. A Metrics test can track almost any aspect of the Chrono runtime, including forces inside of a system or time elapsed in various stages of a program's execution. These can be very useful to determine, for example, how changes to the code slow or speed simulations.

The Metrics test results are exposed via the status page on the Project Chrono website (`http://projectchrono.org/status`) and are also available through an HTTP GET request to an endpoint hosted on the Project Chrono page. Anyone can read results but only authenticated users can push test results.

3.3.2 Validation tests

Validation tests compare Project Chrono test results to experimental results to ensure Project Chrono's physical validity. They are hosted in their own Git repository and are triggered by every run of a Buildbot script, which is in turn triggered by a push to the main Project Chrono repository.

3.3.3 Unit tests

Unit tests are configured to ensure that Project Chrono is internally consistent in how it handles physics. Most unit tests simply check that forces and collisions are handled properly and are realistic given the constraints.

4 Website

The Project Chrono website [1] is built using Jekyll, a static site compiler that compiles Markdown and JavaScript into a fully functional website with dynamic functionality. The website is hosted in separate Git repository and rebuilt on every push to either that repository or the main Project Chrono repository so that it is always up-to-date. The website content is served using NGINX, a lightweight but high-performance web server.

5 Documentation

The documentation for Project Chrono is made using Doxygen [4], which can compile comments in the Project Chrono source code into an HTML website or a LaTeX-built PDF. The documentation website is built using identical JavaScript and CSS to the Jekyll website to provide the same outward appearance and is also hosted using NGINX.

References

[1] Project Chrono. Chrono: An Open Source Framework for the Physics-Based Simulation of Dynamic Systems. `http://projectchrono.org`. Accessed: 2016-03-07.

[2] Project Chrono. ProjectChrono Build Results. `http://buildbot.sbel.org`. Accessed: 2016-03-07.

[3] Buildbot. Buildbot – An open-source framework for automating software build, test, and release. `http://buildbot.net/`. Accessed: 2015-05-31.

[4] Doxygen. Doxygen – A Documentation Generator From Annotated C++ Code. `http://www.doxygen.org`. Accessed: 2015-05-31.

[5] Kitware. CMake – A cross-platform, open-source build system. `http://www.cmake.org`. Accessed: 2015-05-31.

[6] Python Software Foundation. Python. `http://www.python.org/`, 2014.

[7] Project Chrono. Chrono: An Open Source Framework for the Physics-Based Simulation of Dynamic Systems. `https://github.com/projectchrono/chrono`. Accessed: 2015-08-15.

[8] A. Tasora, R. Serban, H. Mazhar, A. Pazouki, D. Melanz, J. Fleischmann, M. Taylor, H. Sugiyama, and D. Negrut. Chrono: An open source multi-physics dynamics engine. In T. Kozubek, editor, *High Performance Computing in Science and Engineering – Lecture Notes in Computer Science*, pages 19–49. Springer, 2016.

[9] TravisCI. Test and Deploy with Confidence. `https://travis-ci.org/`. Accessed: 2015-10-29.