Integration of Docker Containers and Project Chrono

Thomas Liang
Department of Computer Sciences, University of Wisconsin-Madison

May 24, 2023
Contents

1 Introduction 2
2 Approach 2
3 Visualization 2
4 Performance Tests 4
5 Using the Docker Image 5
  5.1 Sensor Module ................................................. 5
  5.2 Modifications .................................................. 5
6 Conclusion and Future Work 6
1 Introduction

Project Chrono [1, 2] is a physics-based modeling and simulation infrastructure based on a platform-independent, open-source design. Over time, various features have been added to the multi-physics library, increasing Chrono’s dependencies on external packages such as Eigen [3], NVIDIA CUDA Toolkit [4], Open MPI [5], OpenGL [6], etc. This results in a convoluted build process as packages need to be properly installed, potentially resulting in complications for users. In addition, Chrono developers and users have published papers and journals that depend on modifications or advancements in Chrono [7, 8, 9, 10, 11, 12, 13]. In order for researchers to reproduce the results, even years after the paper has been published, Docker [14] is utilized for delivering Chrono. However, we have been missing the ability for users to visualize the simulations in run-time with modules in Chrono such as Irrlicht [15].

This report seeks to find the solution to all three of the aforementioned issues, by creating a streamlined install process of Chrono, access to components in papers, and visualization of all simulations in Chrono.

2 Approach

The proposed Docker solution maintains the open-source design philosophy of Project Chrono, and it offers us control over the environment that users use so that we can fix any issues quickly without any disparity at the user’s end and our end. Because Docker uses kernel namespaces, it is faster and more efficient than traditional virtual machines. Docker also hosts a repository of container images called Docker Hub, allowing us to deploy images that users can then pull and run with few commands.

Numerous features in Chrono rely on NVIDIA GPUs meaning if Chrono was to run properly in the Docker container, it would need to access host GPUs. NVIDIA provides Docker images with CUDA on Docker Hub, meaning that it is possible to use NVIDIA’s CUDA images as a parent image, enabling Chrono to use the host GPUs from inside a container.

The Docker image uses Ubuntu 22.04 LTS as it is officially supported by Chrono and the NVIDIA CUDA Docker image. XFCE is the relatively lightweight window manager of choice. Virtual Network Computing (VNC) enables access to the Docker container’s Graphical User Interface (GUI) and noVNC [16] extends VNC access to modern web browsers.

3 Visualization

For visualization, TigerVNC is used to host the Docker image as a VNC server and noVNC is a VNC client used to connect to the VNC server using web browsers. Both projects are open-source and installed into the Docker image with scripts to start VNC and XFCE up when the Docker container is run. Figure 1 demonstrates running a Chrono demo with Irrlicht visualization using the Docker container with noVNC.
Figure 1: Docker container running noVNC, XFCE, Ubuntu 22.04, and Two Cars Demo from Chrono on an Intel i7-5960X and NVIDIA GTX Titan X.

Additionally, users can alter the visualization or drive the vehicle with key-maps, as shown in Fig. 2.

Figure 2: CTRL + I key-map function.
There is a noticeable difference in fluidity of simulations between running the Two Cars demo (Figure 1) in Docker and directly on the host. The real-time factor (RTF) from the visual demonstrations in Docker was between 1 and 2 while the RTF on the host was slightly less than 1.

4 Performance Tests

To further compare the timing difference between running Chrono natively and inside a Docker container, Fig. 3 plots the CPU time of each unit test. Each timing test is carried out twice to average the timing. Note that unit tests do not have run-time visualization enabled.

![Chrono Benchmarking Tests: Host and Docker](image.png)

Figure 3: Chrono utest comparison between running on the Host and running inside the Docker container.

The time it takes to run the same unit test (utest) between Docker and the host is insignificant in most unit tests besides Test 49: utest_FEA_ANCFshell3833_Formulation. It should be pointed out that there were instances where the utests inside the Docker container ran faster than the host. The host took 1110.48 seconds while the Docker container took 1255.42 seconds to run all the unit tests, a 13% increase in time.
5 Using the Docker Image

Pull the Chrono Docker image from Docker Hub and run as a container (~14GB uncompressed size):

```
docker pull uwsbel/projectchrono_novnc
```
```
docker run -d -p 5901:5901 -p 6901:6901 --gpus all uwsbel/projectchrono_novnc
```

Note that this official Chrono Docker image does not include the following Chrono modules due to the size the dependencies will add to the Docker image:

- Cascade
- Pardiso MKL
- Matlab
- Synchrono
- VSG
- Benchmarking and Testing

After the Docker container has started running, navigate to the following address on a web browser: localhost:6091. The password is sbel.

5.1 Sensor Module

Due to licensing issues, OptiX cannot be shipped inside the Docker image. To address this, one has to manually download OptiX 7.5.0 from NVIDIA’s website [4], and then copy it into the running Docker container:

```
docker cp optix-7.5.0 {container-id}:/Packages/
```

Note that the directory name of ”optix-7.5.0” needs to be verbatim.

The command above is also useful for moving files in and out of the container.

5.2 Modifications

This project’s source code can be modified to specific needs such as adding scripts or modifying Chrono itself:

- Necessary Docker files are found in the chrono/contrib/docker folder of the Chrono GitHub repo
- Modifications can be done in “buildChrono.sh” to alter what is built for Chrono
  - “git checkout” to modify the commit that Chrono is built on
- “desktop/” includes all the files that will be added to the Desktop on the Docker container

After modifications are done, run inside the directory with the Dockerfile:

```
docker build . --tag {tag name of image}
```

Note this process will take at least ~30 minutes depending on the specifications of your computer.
6 Conclusion and Future Work

A solution was created to reduce the complexity of the install process of Chrono, stable access to demonstrations in SBEL papers, and view simulation visualizations produced by Chrono at run time. This Docker image will be modified to allow SBEL members to add their work to publish alongside their papers; readers can easily run simulations from the papers on their own computers. There are some performance drawbacks with running Chrono inside a Docker container, however this is compensated by the simplicity of using these Docker images.

As previously mentioned, there are Chrono modules missing from the current Docker image, and the plan is to add them as different versions (tags) of this image on Docker Hub in the future. The problem is the resulting increase in size of Docker images may not be optimal for some users meaning that the current plan is to maintain this Chrono configuration as the default latest image on Docker Hub.

References


