Simulation-Based Engineering Lab
University of Wisconsin-Madison

Technical Report TR–2017–08

Overview of the Chrono OpenSim Parser

Conlain Kelly and Radu Serban

Dept. of Mechanical Engineering, University of Wisconsin – Madison

December 6, 2017

1 Introduction

This report presents a brief overview of the Chrono [?] functionality for parsing OpenSim input files. It also includes a basic usage guide with code examples.

OpenSim's multibody dynamics capabilities are based on the open-source Simbody package [1] which uses an *internal coordinate*, recursive formulation, in contrast with the *full coordinate*, Cartesian formulation employed in Chrono. Conceptually, the main difference between a Cartesian and a recursive multibody formulations is in the abstract interpretation of the mechanical joints connecting rigid bodies: in a Cartesian formulation, a joint is modeled as a collection of constraints that restrict relative motion; conversely, in a recursive formulation, a mechanical joint is interpreted as a mobilizer which induces a certain number of relative degrees of freedom.

The basic Simbody components used to construct the multibody system are:

- Body (mass properties and geometry)
- Mobilizer (internal coordinate joint)
- Constraint
- Force

The first two are combined into the *MobilizedBody* construct which represents a rigid body and its inboard mobilizer (which defined the body's degrees of freedom relative to its parent body).

OpenSim models can be constructed programatically, or else specified through an `osim` input file with an XML-based format. The Chrono functionality described here parses such `osim` input files to create an equivalent Chrono mechanical system, or else populate an existing Chrono system with the elements specified in the `osim` input file.

2 Background

We define a position transform from frame A to frame B as:

$$ {}^{A}\mathbf{X}^{B} = \left[{}^{A}\mathbf{R}^{B} \; ; \; {}^{A}\mathbf{t}^{B} \right] \tag{1} $$

where ${}^{A}\mathbf{R}^{B}$ represents a 3×3 rotation matrix and ${}^{A}\mathbf{t}^{B}$ the translation vector between the origins of the two frames. The inverse transform is therefore derived as:

$$ {}^{B}\mathbf{X}^{A} = \left({}^{A}\mathbf{X}^{B} \right)^{-1} = \left[\left({}^{A}\mathbf{R}^{B} \right)^{-1} \; ; \; \left({}^{A}\mathbf{R}^{B} \right)^{-1} \cdot {}^{A}\mathbf{t}^{B} \right] \tag{2} $$

With this, a position vector ${}^{B}\mathbf{v}$ represented in frame B can be expressed in frame A using:

$$ {}^{A}\mathbf{v} = {}^{A}\mathbf{X}^{B} \cdot {}^{B}\mathbf{v} \equiv {}^{A}\mathbf{R}^{B} \cdot {}^{B}\mathbf{v} + {}^{A}\mathbf{t}^{B} \tag{3} $$

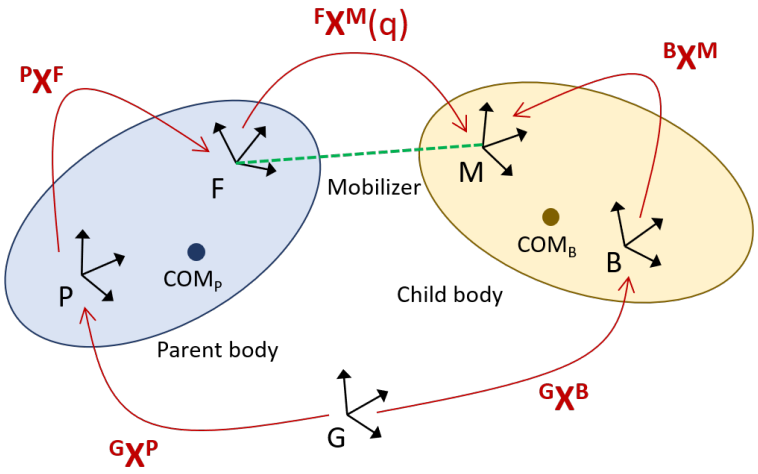

Figure 1: Relationship between parent and child bodies in a recursive multibody formulation. Body reference frames are assumed non-centroidal. All transforms are expressed in and relative to their parent transforms. The mobilizer transform ${}^F\mathbf{X}^M(q)$ is parameterized by its internal coordinates q.

A Simbody open-loop multibody system is defined recursively, starting from the root body. The generalized state of a *MobilizedBody* B, given the state of its parent P, is completely defined by:

$$
\begin{aligned}
{}^P\mathbf{X}^B &= \left[{}^P\mathbf{R}^B \;;\; {}^P\mathbf{t}^B\right] && (4)\\
{}^P\mathbf{V}^B &= {}^P\mathbf{H}^B \cdot u && (5)\\
\dot{q} &= \mathbf{N}(q)u && (6)
\end{aligned}
$$

where ${}^P\mathbf{H}^B$ is the velocity transform matrix used to obtain the 6×1 spatial velocity ${}^P\mathbf{V}^B$ and $\mathbf{N}$ is the kinematic coupling matrix expressing the relationship between generalized velocities and derivatives of the generalized (internal) coordinates.

A Chrono multibody system is defined by a collection of bodies and constraints modeling mechanical joints between connected bodies. The state vector is the collection of the Cartesian body positions and velocities, all expressed with respect to an absolute (ground) reference frame.

Referring to Fig. 1, the absolute transform of a body, ${}^G\mathbf{X}^B$ (expressed in and relative to the ground frame G), given its Simbody representation as a *MobilizedBody*, is thus determined as:

$$
{}^G\mathbf{X}^B = {}^G\mathbf{X}^P \cdot {}^P\mathbf{X}^F \cdot {}^F\mathbf{X}^M(q) \cdot {}^M\mathbf{X}^B \qquad (7)
$$

where ${}^G\mathbf{X}^P$ is the absolute transform of the parent body, ${}^P\mathbf{X}^F$ represents the position of the outboard joint on the parent body (expressed in the P frame), ${}^F\mathbf{X}^M(q)$ is the mobilizer transform (between the *fixed* and the *moving* frames of the mechanical joint, parameterized

by the internal coordinates of the particular mobilizer), and ${}^{B}\mathbf{X}^{M}$ is the position of the inboard joint on the child body (expressed in the B frame). Note that ${}^{M}\mathbf{X}^{B} = \left({}^{B}\mathbf{X}^{M}\right)^{-1}$.

A Simbody system is defined hierarchically, with each *MobilizedBody* specifying its type and the mobilizer positions on the parent and child body (${}^{P}\mathbf{X}^{F}$ and ${}^{B}\mathbf{X}^{M}$, respectively), as well as body properties (such as mass, inertia tensor, location of the center of mass, etc.). In addition, the `osim` model specification provides initial internal coordinate and velocity values. From the mobilizer type, the parser infers the corresponding Chrono joint, as well as the transform ${}^{F}\mathbf{X}^{M}(q_0)$ for the initial configuration q_0. In other words, a Simbody *MobilizedBody* results in the creation of a Chrono body and a Chrono joint. The initial position of the resulting body is calculated using Eq. 7, from the already calculated position of its parent body.

3 Implementation and Usage

The Chrono parser for `osim` files is encapsulated in a single class, `ChParserOpenSim`, under the `chrono::utils` namespace. Chrono users can instantiate an object of this type in their main program and use it to parse a given `osim` file to create, within a `ChSystem`, the modeling elements (bodies, joints, forces, etc.) specified in the input file.

The parser reads in an xml file using RapidXML [**?**], an open-source C library under the MIT license. RapidXML reads the `osim` file and parses it in-place, making it quick and simple to use. Once read, it creates a tree representation of the XML file. The Chrono parser traverses this tree, reading general simulation data like inertia moments and mass, then reading in bodies. It also has the capability to read additional fields from the `osim` file, if needed. The format of `osim` files is described in the OpenSim User's Guide [**?**].

The parser receives a pointer to the root body of the recursive tree. It assumes the tree to be non-cyclic (i.e., representing a mechanical system with no closed loops), although bodies can have multiple children. It chooses the root body to be the "ground" body, which is fixed to ground. All bodies are defined recursively, with their relative positions and orientations being determined by those of their parents and by the mobilizer-specific transform of their inboard joint.

Currently, the parser only processes initial positions and ignores initial internal mobilizer velocities, setting them to 0. Processing initial velocity information is significantly more involved, as it would require implementing, within `ChParserOpenSim`, all velocity transforms ${}^{F}\mathbf{H}^{M}$ for known joints. Nonetheless, this option will be considered for future extensions.

3.1 Rigid Bodies

For efficiency considerations, the default body reference frame in Chrono is centroidal. As such, markers, joints, visualization assets, and collision shapes associated with a `ChBody` must be defined with respect to a frame located at the body center of mass. For modeling convenience, Chrono provides a derived class `ChBodyAuxRef` which allows specification of a

body with respect to an arbitrary (possibly non-centroidal) frame. Since *MobilizedBodies* in Simbody are defined with respect to an arbitrary frame, all bodies generated by the `ChParseOpenSim` are of type `ChBodyAuxRef`. This eliminates the need for additional frame transformations during parsing.

3.2 Supported Joint Types

The following Simbody joints have a direct Chrono equivalent and can be therefore completely parsed:

- Revolute (a.k.a. Pin joint)
- Spherical (a.k.a. Ball-and-socket)
- Universal (a.k.a. Cardan joint)
- Weld joint (zero degree of freedom joint)

Simbody joints that are not recognized are replaced with a Chrono spherical joint; in this case, the joint transform ${}^F\mathbf{X}^M$ is properly parsed and calculated and therefore the child body is initialized in its correct configuration. Custom Simbody joints, implemented externally as black boxes, are also replaced with a Chrono spherical joint and their joint transform is set to identity.

3.3 Force and Actuator Elements

Currently, no Simbody force elements are parsed and interpreted (e.g., `PrescribedForce`, `SpringGeneralizedForce`, and `BushingForce`).

OpenSim also includes "ideal" actuators which apply pure forces or torques that are directly proportional to the input control (i.e., excitation) via its optimal force setting (i.e., a gain). Such forces and torques are applied on a body or between two bodies. `ChParserOpenSim` currently parses and interprets the following:

PointActuator : Apply a force on a given body at a specified point and in a specified direction; both th application point and the force direction can be provided in either the absolute or body-local reference frame.

TorqueActuator : Apply equal and opposite torques on two bodies about a specified axis; the torque direction can be provided in either the absolute frame or in a body-local reference frame.

The parser translates these actuators into Chrono *loads* which are generalizations of forces and torques that can be applied on various Chrono modeling elements, including rigid bodies. In particular, a `PointActuator` is translated into a `ChLoadBodyForce`, while

Listing 1: Setting an increasing excitation

```
auto excitation = std::make_shared<ChFunction_Ramp>(0, 1);
parser.SetExcitationFunction("grav", excitation);
```

a `TorqueActuator` is translated into a `ChLoadBodyBodyTorque`[1]. Chrono loads are collected into so-called *load containers*, of type `ChLoadContainer`. All loads parsed from OpenSim actuators are collected into a single load container (a `ChSystem` may contain any number of such containers).

Excitation functions can be specified in several different ways in OpenSim, either directly into the `.osim` file, or else into a separate *controls* file (in XML format) referenced to from within the `.osim` file. In the former case, OpenSim also provides a simple grammar which allows specification of various mathematical functions for these excitations. To simplify the parsing process, while also providing greater flexibility, we opted for allowing the user to specify their own excitation functions natively in Chrono through its `ChFunction` mechanism. The `ChFunction` base class specifies an abstract scalar function of a single variable ($y = f(x)$) with a relatively large set of concrete derived classes (including polynomial, sine, piece-wise linear, etc.); moreover, a user can extend this set by defining their own derived `ChFunction` class (see the Chrono API reference documentation for details).

By default, and following the OpenSim philosophy, all translated actuators are not excited (in other words, their associated excitation is the constant 0 function). Optionally, all Chrono actuator loads can be created with full excitation (i.e., with an associated constant 1 excitation function), by calling `parser.ActivateActuators(true)` prior to the parsing process. After parsing, the excitation of a specific actuator can be set via `ChParserOpenSim::SetExcitationFunction()` as shown in Listing 1, where a particular actuator is referred to using its name in the `.osim` file (which can also be accessed through the *report* object described in §3.7). Actuator types beyond those listed above are currently ignored by the parser, although the user can find the relevant bodies via `ChParserOpenSim::Report::GetBody()` and create a custom force between them.

3.4 Visualization

Visualization of the resulting Chrono bodies can be controlled by the user. The following options are available:

none no visualization assets are created (this is the default)

primitives a visualization model is created for each body using primitive Chrono visualization assets, consisting of a sphere centered at the origin of the body reference frame

[1]Note that OpenSim and Chrono have opposite conventions in ordering the bodies on which a TorqueActuator acts (and this is taken into account in the parser's implementation)

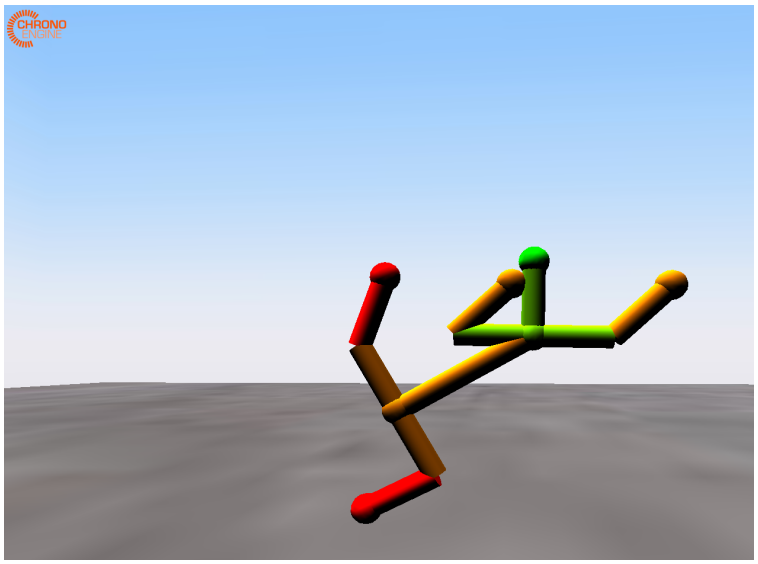

Figure 2: Snapshot from a Chrono simulation of a parsed OpenSim model. The visualization mode was set to `PRIMITIVES` and collision was enabled, also specifying that contact should not occur between adjacent bodies. Each color represents a depth in the tree hierarchy.

and cylinders that connect the body reference frame with the location of all joints adjacent to the body (the inboard joint and all outboard joints). See Fig. 2 for an example.

mesh use visualization meshes, assumed to be available as Wavefront `obj` files and specified under the `<geometry_file>` tag in the input `osim` file. See Fig. 3 for an example.

3.5 Collision and Contact

By default, no collision geometry is created for the resulting Chrono bodies. If enabled, a collision model is created from primitive cylinder shapes, similar to the visualization shapes in the `PRIMITIVES` mode. Since this implies that collision shapes from adjacent bodies overlap at the connecting joint, bodies are assigned to alternating collision families, based on their depth in the tree hierarchy. No collisions are generated between a body and its parent; collisions are allowed with any other body, including siblings.

Contact processing can be done with either of the two methods supported in Chrono, namely `NSC` (non-smooth, complementarity-based approach) or `SMC` (smooth, penalty-based approach). This is controlled by the type of the containing `ChSystem`. Optional methods in the `ChParserOpenSim` class allow specification of contact material properties (see [2]).

3.6 Sample Usage

An example of parsing an `osim` file to populate an existing Chrono system is as shown in Listing 2. The usage pattern is:

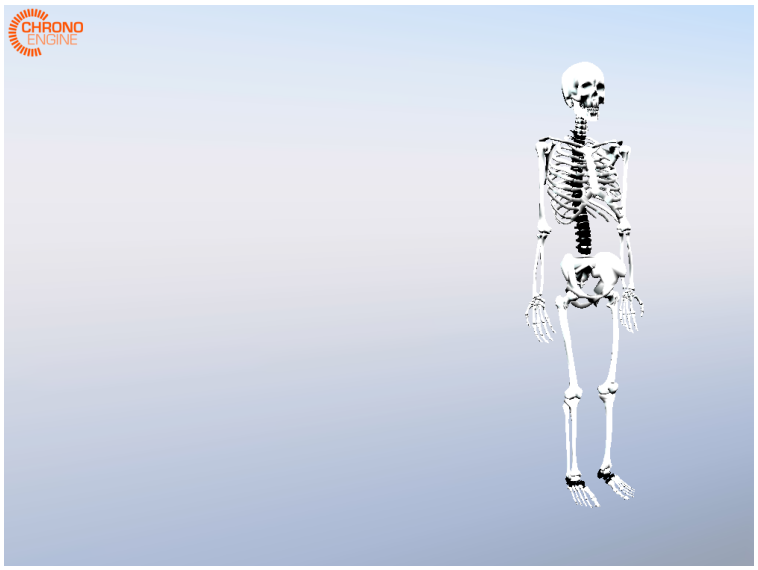

Figure 3: Initial configuration of a Chrono model loaded from an OpenSim input file. The visualization mode here was set to `MESH` and collision was disabled.

Listing 2: ChParserOpenSim usage example

```
ChSystemSMC my_system;
std::string filename = GetChronoDataFile("opensim/skeleton.osim");
ChParserOpenSim parser;
parser.SetVisualizationType(ChParserOpenSim::VisType::PRIMITIVES);
parser.EnableCollision();
parser.SetVerbose(true);
parser.Parse(my_system, filename);
```

Listing 3: ChParserOpenSim::Report class

```
/// Report containing information about objects parsed from file
class ChApi Report {
public:
/// Information about a joint read in from OpenSim.
struct JointInfo {
std::string type;                ///< joint type as shown in osim file
std::shared_ptr<ChLink> joint;   ///< Chrono link (joint)
bool standin;                    ///< joint replaced with spherical?
};

/// Information about a custom load created from OpenSim.
struct ForceInfo {
std::string type;                    ///< load type as shown in osim file
std::shared_ptr<ChLoadBase> load;    ///< Chrono load object
};

std::unordered_map<std::string, std::shared_ptr<ChBodyAuxRef>> bodies;
///< list of body information
std::unordered_map<std::string, JointInfo> joints;
///< list of joint information
std::unordered_map<std::string, ForceInfo> forces;
///< list of force information
```

1. create parser object;
2. set parsing options;
3. invoke one of the `Parse` methods.

The first two lines in Listing 2 create the `ChSystem` to hold the resulting model and specify the name of the file to be parsed. The third line creates the parser object; the fourth sets it to visualize with primitives (explained below). The next 3 set flags within the parser to determine visualization, contact, and debug output. The default behavior is no vis, no collide, and no verbose output. The last line parses the file given by `filename` into `my_system`. There is an alternative `ChParserOpenSim::Parse` function that instead takes a filename and a `ChMaterialSurface::ContactMethod` to create an appropriate system with this contact type, parse the file into that system, and return a pointer to the new system.

3.7 Report

The `ChParserOpenSim::Report` class provides an interface for the user to access bodies, joints, and forces parsed from the `.osim` file. A report object is created during parsing to store, in maps hashed by the element name, the lists of Chrono bodies, joints, and loads. The relevant data structures are shown in Listing 3. The `ChParserOpenSim::Report` provides

methods for printing the report and for accessing bodies, joints, and loads by their name. The report for a parser can be accessed via `ChParserOpenSim::GetReport()`.

3.8 Current Limitations

- Initial velocities are assumed to be zero.
- Unrecognized joints (including custom joints) are put in the correct initial positions but replaced by spherical joints, resulting in different kinematics.
- The mobilizer frame (${}^F\mathbf{X}^M(q)$) for custom joints is set to the identity transform, resulting in a different initial pose.

4 Possible Future Extensions

Currently, `ChParseSimBody` interprets and parses only a subset of the valid attributes in an `osim` file. This allows creating equivalent Chrono models consisting of rigid bodies (with optional visualization assets and collision shapes), a set of recognized joints (i.e., Simbody mobilizers for which there is a direct Chrono joint equivalent) and simple actuators (`PointActuator` and `TorqueActuator`).

Potential future extensions include: parsing and creating markers, other force elements (e.g., generalized spring forces), as well as parsing and interpreting initial internal coordinate velocities.

4.1 Support for Muscle Models

Current capabilities of the OpenSim – Chrono parser allows the creation and simulation of the underlying kinematics model of a human skeleton, with only very limited, and certainly insufficient, support for actuation. Future extensions will need to include models for the muscles and tendons which requires implementation of capabilities currently not available in Chrono; in particular:

- Muscle actuators based on characteristic muscolotendon curves.

  Such force elements can be implemented either as a separate, domain-specific, hierarchy of C++ classes, or else leveraging the current `ChLoader` mechanism, which allows the definition of general concentrated or distributed loads on *loadable* elements (e.g., bodies and/or meshes) in a Chrono system.

  Implementation of the former approach would parallel, as an example, the current implementation of generalized spring-damper force elements in Chrono, such as the `ChLinkSpringCB` class which allows specification of an arbitrary force through a user-supplied functor object.

As examples of the latter approach, we mention the implementation of contact forces on FEA meshes (through `ChLoadContactSurfaceMesh`), or the implementation of internal tire pressure in Chrono::Vehicle models through a load based on a surface loader of type `ChLoaderPressure`.

In either case, the implementation could provide pre-defined Hill-type muscle force models based on characteristic curves (for active force length, passive force length, force velocity, and tendon force length), while also accommodating user-supplied muscle models through callback mechanisms.

- Geometric calculation of tendon length and tendon path.

  Additional Chrono support is required to characterize and operate with geometrical information related to attachment points of muscle-tendon complex ends to bones, muscle/tendon paths and via points, as well as support for calculation of tendon length. These could be implemented as extensions to the Chrono geometry utilties (in the `chrono::geometry` namespace) and optionally be included in the collision and contact processing.

We recommend that such an extension of Chrono modeling and simulation capabilities to biomechanical systems be encapsulated in an optional module (e.g., Chrono:Biomechanics). Similar to the Chrono::Vehicle module – dedicated to flexible and expeditious modeling of ground vehicle systems – a future Chrono::Biomechanics module could provide templates for biomechanical systems, to represent parameterized models of multibody systems with fixed, known, topology.

A ChParserOpenSim documentation

We list here some of the more important functions in the `ChParserOpenSim` class. For more details, see the Project Chrono API documentation [2].

Parse the specified OpenSim input file and create the model in the given system.

```
void chrono::utils::ChParserOpenSim::Parse(
    ChSystem& system,
    const std::string& filename
)
```

Arguments:
system containing Chrono system
filename `osim` input file name

Parse the specified OpenSim input file and create the model in a new system. Note that the created system is not deleted in the parser's destructor; rather, ownership is transferred to the caller.

```
ChSystem* chrono::utils::ChParserOpenSim::Parse(
    const std::string& filename,
    ChMaterialSurface::ContactMethod method = ChMaterialSurface::NSC
)
```

Arguments:
filename `osim` input file name
method contact method (NSC: non-smooth, complementarity-based; SMC: smooth, penalty-based)

Enable collision between bodies in this model (default: false). Set collision families to disable collision between a body and its parent.

```
void chrono::utils::ChParserOpenSim::EnableCollision(
    int family_1 = 1,
    int family_2 = 2
)
```

Arguments:
family_1 first collision family
family_2 second collision family

Set body visualization type.

```
void SetVisualizationType(VisType val)
```

Arguments:
val visualization mode (default: NONE)
The visualization mode can be one of:
PRIMITIVES use visualization primitives (cylinders)
MESH use meshes specified via a Wavefront input file
NONE no visualization

Activate actuators. By default, any actuator read in from the osim file is inactive (zero excitation). If enabled, all actuators will receive a constant excitation function with value 1. The excitation function for an actuator can be specified, after parsing, using SetExcitationFunction().

```
void ActivateActuators(bool val)
```

Arguments:
val enable/disable automatic full excitation

Set excitation function for the actuator with the specified name. This method should be invoked only after parsing an osim file.

```
void SetExcitationFunction(
    const std::string& name,
    std::shared_ptr<ChFunction> modulation
)
```

Arguments:
name name of an actuator (load)
modulation function object to provide excitation signal

References

[1] Michael A Sherman, Ajay Seth, and Scott L Delp. Simbody: multibody dynamics for biomedical research. *Procedia Iutam*, 2:241–261, 2011.

[2] Project Chrono. ProjectChrono API Web Page. `http://api.chrono.projectchrono.org/index.html`. Accessed: 2016-03-07.