Chrono::Vehicle – Template-Based Ground Vehicle Modeling and Simulation

Abstract: Chrono::Vehicle is a module of the open-source multi-physics simulation package Chrono, aimed at modeling, simulation, and visualization of wheeled and tracked ground vehicle multi-body systems. Its software architecture and design was dictated by the desire to provide an expeditious and user-friendly mechanism for assembling complex vehicle models, while leveraging the underlying Chrono modeling and simulation capabilities, allowing seamless interfacing to other optional Chrono modules (e.g., its granular dynamics and fluid-solid interaction capabilities), and providing a modular and expressive API to facilitate its use in third-party applications. Vehicle models are specified as a hierarchy of subsystems, each of which is an instantiation of a predefined subsystem template. Written in C++, Chrono::Vehicle is offered as a middleware library.

In this paper, we provide an overview of the Chrono::Vehicle software design philosophy, its main capabilities and features, describe the types of ground vehicle mobility simulations it enables, and outline several directions of future development and planned extensions.

Keywords: Chrono; Vehicle modeling; Template-based design; JSON.

1 Introduction

1.1 Brief overview of Chrono

Chrono [27] is an open-source multi-physics software package, which is distributed under a permissive BSD-3 license [22]. The core functionality of Chrono provides support for the modeling, simulation, and visualization of rigid multibody systems, with additional capabilities offered through optional modules. These modules provide support for additional classes of problems (e.g., finite element analysis and fluid-solid interaction), for modeling and simulation of specialized systems (such as ground vehicles and granular dynamics problems), or for providing specialized parallel computing algorithms (multi-core, GPU, and distributed) for large-scale simulations.

Chrono is almost entirely written in C++ and it is compiled into a library subsequently used by third-party applications. As such, Chrono is middleware software; i.e., software that supports customized solutions that potentially involve other user code and/or third-party software. A user can invoke functions implemented in Chrono via an Application Programming Interface (API) that comes in two styles: C++ and Python. Chrono, which runs on Windows, Linux, and Mac OSX, is organized into modules that are functionally independent. When building the Chrono middleware library only the modules of interest are compiled into libraries that are subsequently linked in a user application. The core module provides basic functionality required to simulate the dynamics of mechanical systems made up of bodies, kinematic joints, force elements, 1-D shaft elements, etc. One of the salient features of Chrono is its ability to account for the geometry (shape) of the elements that make up the mechanical system simulated. This opens the door to analyses in which the

user can simulate the interaction between bodies, when the interplay between shape and frictional contact forces determines the dynamics of the system. In this context, Chrono has been used to simulate granular dynamics of systems containing millions of bodies whose macroscale motion is the outcome of micro-scale interaction between pairs of bodies, modulated by body geometry and frictional contact forces.

Chrono has several modules that provide modeling and simulation support in multi-physics applications. Chrono::FEA (Finite Element Analysis) is designed to address challenges specific to the simulation of dynamic systems that might experience large displacements and/or large rotations and/or large deformations. The flexible bodies can interact with other system elements via forces, friction and contact, and constraints. As of release 3.0, Chrono supports three FEA approaches. The absolute nodal coordinate formulation (ANCF) [25] can be used for large deformations and arbitrary displacements/rotations. The co-rotational formulation [26] is useful in small deformation and large displacements/rotations scenarios. Finally, there is preliminary support for traditional Lagrangian finite elements that can be used for large deformations/displacements/rotations.

Support for fluid-solid interaction (FSI) simulation is provided by Chrono::FSI, which draws on the Smoothed Particle Hydrodynamics (SPH) method to spatially discretize the mass and momentum balance equations in fluid dynamics. Chrono currently supports two approaches to solving the Navier-Stokes equations of motion for incompressible fluids. The weakly compressible SPH is the de-facto standard in the literature and is implemented in Chrono using an equation of state for the pressure [6] to enforce incompressibility via penalty. The pressure acts as a corrective term that seeks to maintain constant fluid density. In a second approach the fluid incompressibility is enforced via kinematic constraint equations that maintain a uniform distribution of SPH particles in the fluid flow. This Constrained Fluid SPH method leads to a set of equations of motion that are solved using the same methodology involved in granular dynamics simulations. For the latter, preliminary support is provided by Chrono::Granular, which allows the user to quickly set up large collections of bodies that each can have a nontrivial geometry.

From an abstract perspective, Chrono rests on five foundation components that provide the following basic functionality: equation formulation, equation solution, collision detection and proximity computation, support for parallel computing, and pre/post-processing, see Fig. 1. The first foundation component, called “Equation Formulation”, supports general-purpose modeling for large systems of rigid and flexible bodies and for basic fluid-solid interaction (FSI) problems. The second component, called “Equation Solution”, provides the algorithmic support needed to numerically solve the resulting equations of motion. Proximity computation support, essential for collision detection and computation of short range interaction forces, is provided by the third foundation component. The fourth component concentrates on parallel computing support at various levels: vectorization, acceleration using dedicated hardware (GPU), multi-core via OpenMP, and large scale as enabled by the MPI standard. Finally, the fifth component provides pre- and post-processing support. For the latter, Chrono has run-time support via Irrlicht and OpenGL, and uses POV-Ray, Mitsuba, and ParaView for off-line, higher-quality, visualization.

One of Chrono’s strengths is its reliance on advanced computing hardware at various stages of the solution process. Indeed, Chrono embraces cache friendly data structures suitable for vectorization, and algorithms that expose parallelism at data and task levels. In a quest to reduce simulation times via parallel computing, we have established three

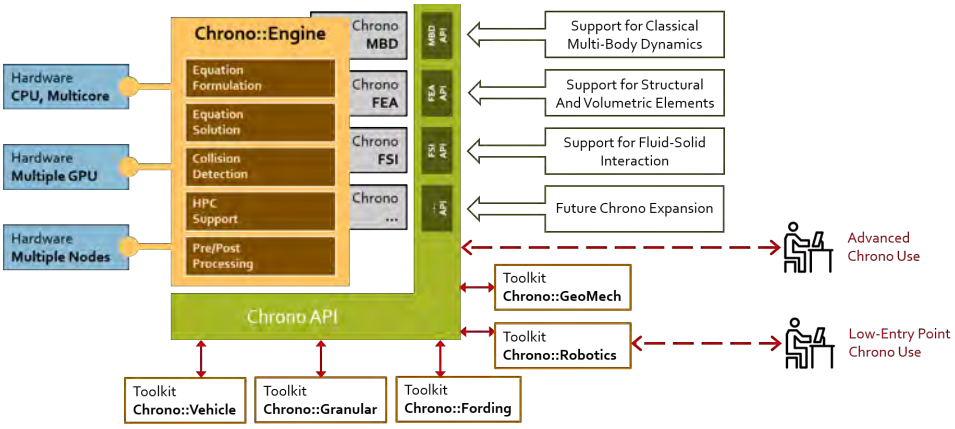


Figure 1: High-level structure of the Chrono multi-physics software package. The public API allows interfacing either directly to the Chrono physics libraries or, for a lower entry point, intermediated by various domain-specific toolkits. These toolkits are currently at different stages of maturity. The Chrono::Vehicle toolkit provides support for the expeditious modeling, simulation, and visualization of ground vehicles.

modules – Chrono::Cosimulation, Chrono::Distributed, and Chrono::Parallel – that enable Chrono to map for execution each of the many components of a complex model onto the appropriate parallel computing hardware architecture. Chrono aims at establishing a flexible, object-oriented infrastructure that (1) relies on Chrono::Cosimulation to handle in parallel and independently sub-systems of a complex *system*; (2) uses the MPI standard to further partition a large *sub-system* via Chrono::Distributed into parallel subgroups; and, (3) invokes services provided by Chrono::Parallel to accelerate execution within each *subgroup* using two hardware platforms: NVIDIA GPU and Intel Xeon Phi. This *system* → *sub-system* → *subgroup* cascading is reflected into the Chrono::Cosimulation → Chrono::Distributed → Chrono::Parallel interplay. For instance, a user enlists the support of modeling elements from the Chrono Vehicle, FSI, Terramechanics, and FEA modules to put together, for instance, a simulation of a wheeled vehicle moving on gravel and performing a fording maneuver. This example has seven sub-systems: four tires, the vehicle body, the granular terrain, and the fluid component, whose execution is managed by Chrono::Cosimulation. The large sub-systems; i.e., the terrain and fluid, are further split into subgroups managed by Chrono::Distributed. Each of these subgroups is independently accelerated via fine-grain parallelism, a process overseen by Chrono::Parallel.

1.2 Motivation for Chrono::Vehicle

While arbitrary mechanical systems can be directly modeled in Chrono, using its basic modeling primitives (bodies, joints, force elements, etc.), doing so for models of higher complexity, size, or lacking a simple straight-forward structure is tedious and error-prone. Ground vehicle systems can be complicated, involve many tens of bodies (as is the case for segmented track vehicles), and require intricate connectivity and precise design configurations. However, typical vehicle multi-body systems have relatively standard topologies and well-defined hierarchical structure. Moreover, operational and manufacturing requirements led to a relatively restricted set of designs for the main

vehicle sub-assemblies, such as suspensions, steering mechanism, track assemblies, etc. As recognized by most commercial multi-body dynamics software vendors [16, 23, 4] and by some open-source multi-body simulation software developers, this suggests the design of modeling tools based on predefined parameterized templates for the major ground vehicle subsystems. A template-based modeling capability thus enables the expeditious creation of new vehicles, allows rapid prototyping, and facilitates model re-use.

Chrono::Vehicle is the embodiment of this approach in the Chrono software suite. Its modular design, consistent API, rich set of templates, and organization of the simulation time integration loop additionally enable easy replacement of various systems ("plug-and-play" philosophy) and seamless interfacing to third-party libraries. The architecture of the Chrono::Vehicle module allows both fully coupled and co-simulation approaches for vehicle-terrain simulations. In the latter case, the vehicle, terrain, engine, and driver systems evolve in parallel with periodic data communication, for example in an explicit force–displacement co-simulation framework [21, 24].

In addition to providing Chrono users with the expected benefits of template-based ground vehicle modeling, Chrono::Vehicle was also architected and designed to leverage existing and future extensions modeling capabilities in Chrono and capitalize on its underlying high-performance and parallel computing facilities. Chrono::Vehicle models can be easily incorporated in complex, multiphysics simulations, such as flexible tires on granular terrain [21], fluid-solid interaction (FSI) fording scenarios [13], and autonomous and connected vehicle tests [3].

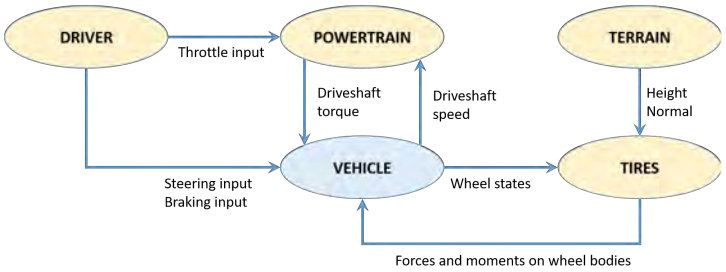
2 Template-based modeling

Chrono::Vehicle provides a collection of templates for various topologies of both wheeled and tracked vehicle subsystems, support for modeling rigid, flexible, and granular terrain, support for closed-loop and interactive driver models, and run-time and off-line visualization of simulation results.

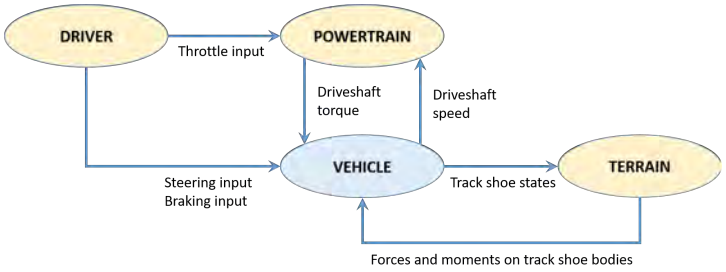
Modeling of vehicle systems is done in a modular fashion, with a vehicle defined as an assembly of instances of various subsystems (suspension, steering, driveline, etc.). Flexibility in modeling is provided by adopting a template-based design. In Chrono::Vehicle, templates are parameterized models that define a particular implementation of a vehicle subsystem. As such, a template defines the basic modeling elements (bodies, joints, force elements), imposes the subsystem topology, prescribes the design parameters, and implements the common functionality for a given type of subsystem (e.g., suspension) particularized to a specific template (e.g., double wishbone). Finally, an instantiation of such a template is obtained by specifying the template parameters (hardpoints, joint directions, inertial properties, contact material properties, etc.) for a concrete vehicle (e.g., the HMMWV front suspension).

The core templates in Chrono::Vehicle are parameterized models of vehicle subcomponents. However, a complete vehicle mobility simulation also requires *auxiliary* systems, external to the vehicle itself, such as a *driver* system to provide input controls (e.g., steering, throttle, braking), a *powertrain* system which encapsulates the engine and transmission and connects to the the vehicle driveline, and a *terrain* system.

A Chrono::Vehicle simulation loop takes the form of a force-displacement co-simulation scheme, with the exchange data illustrated in Fig. 2a for wheeled vehicles and Fig. 2b for tracked vehicles. This software architecture was adopted in order to (*i*)



(a) Wheeled vehicles



(b) Tracked vehicles

Figure 2: Main systems and exchange data flow for the wheeled and tracked vehicles. The software architecture and design of Chrono::Vehicle allows third-party plugins for any or all of the driver, powertrain, tire, and terrain systems.

provide modularity and flexibility; (*ii*) permit use of third-party auxiliary system models and integration of Chrono::Vehicle models in larger simulation frameworks; and (*iii*) enable co-simulation with external tools or with other Chrono modules. This simulation flow is enforced through the Chrono::Vehicle API which imposes that all systems provide a `Synchronize` method, which implements the data exchange, and an `Advance` method, which implements the system dynamics (i.e., advances the system's states to the next data exchange time point). Note however that a vehicle mobility simulation that uses auxiliary systems providing with Chrono::Vehicle (see Section 3.1) may also be set up as a monolithic, all-at-once coupled simulation.

For wheeled vehicle systems, templates are provided for the following subsystems: *suspension* (double wishbone, reduced double wishbone using distance constraints, multi-link, solid-axle, MacPhearson strut, semi-trailing arm); *steering* (Pitman arm, rack-and-pinion); *driveline* (2WD and 4WD shaft-based using specialized Chrono modeling elements, simplified kinematic driveline); *wheel* (simply a carrier for additional mass and inertia appended to the suspension's spindle body); *brake* (simple model using a constant torque modulated by the driver braking input).

Chrono::Vehicle offers a variety of tire models and associated templates, ranging from rigid tires, to semi-empirical models (such as Pacejka and Fiala), to fully deformable tires modeled with finite elements (using either an Absolute Nodal Coordinate Formulation or a co-rotational formulation).

For tracked vehicles, the following subsystem templates are available: *track shoe* (single- and double-pin) and associated *sprocket* templates (with corresponding gear profiles), *suspension* (torsional spring with either linear or rotational damper, hydraulic), *idler* (with tensioner mechanism), and *rollers*.

As a middleware library, Chrono::Vehicle requires the user to provide C++ classes for a concrete instantiation of a particular template. An optional Chrono library provides complete sets of such concrete C++ classes for a few ground vehicles, both wheeled and tracked, which can serve as examples for other specific vehicle models. While such classes are typically very lightweight, this requires some programming experience and, more importantly, makes it more difficult to encapsulate Chrono::Vehicle in a design exploration and virtual prototyping work-flow. To address this issue, we provide an alternative mechanism for defining concrete instantiation of vehicle system and subsystem templates, which is based on input specification files in the JSON format [2]. Following the hierarchy of subsystem templates for its given type (wheeled or tracked), a vehicle can be completely defined through a corresponding hierarchy of JSON files that specify the concrete template parameters or else defer to further JSON specification files for sub-components. Listing 1 is an example of a top-level JSON specification file for a wheeled vehicle. Together with all other input files it refers to, this JSON file completely describes a concrete wheeled vehicle with two axles, using double wishbone suspensions both in front and rear, a Pitman arm steering mechanism attached to the front axle, and a rear-wheel driveline.

3 Chrono::Vehicle templates and capabilities

In this section, we provide an overview of the system- and subsystem-level templates available in Chrono::Vehicle.

Each vehicle subsystem is defined with respect to its own reference frame; in other words, all hardpoint locations in a vehicle subsystem template must be provided with respect to the subsystem's reference frame. A vehicle system, be it wheeled or tracked, is then constructed as a collection of concrete instantiations of templates for its constituent components by specifying their position and orientation with respect to the vehicle reference frame and providing connectivity information, as required (e.g., attaching a particular steering mechanism to a particular axle/suspension of a wheeled vehicle).

Chrono::Vehicle uses exclusively the ISO vehicle axes convention, namely a right-hand frame with X forward, Z up, and Y pointing to the left of the vehicle (see ISO 8855:2011). Figure 3 illustrates the vehicle reference frame O_1 (by convention aligned with that of the chassis subsystem), as well as subsystem reference frames (O_2' and O_2'' for the front and rear suspensions, and O_3 for the steering mechanism) for a wheeled vehicle with two axles.

3.1 Common systems

We begin by describing the *auxiliary* systems, not part of the vehicle system itself, but required to set up a complete vehicle mobility simulation, namely the *terrain*, *driver*, and *powertrain* systems. For uniformity and modeling flexibility, these systems are also templatized. Moreover, a consistent public API allows a Chrono::Vehicle vehicle model to

Listing 1: Sample JSON specification file for a wheeled vehicle

```
{
  "Name":       "Test vehicle - Double Wishbone",
  "Type":       "Vehicle",
  "Template":   "WheeledVehicle",

  "Chassis": {
    "Input File":  "generic/chassis/Chassis.json"
  },

  "Axles": [
  {
    "Suspension Input File":    "generic/suspension/DoubleWishbone.json",
    "Suspension Location":      [1.25, 0, -0.21],
    "Steering Index":           0,
    "Left Wheel Input File":    "generic/wheel/WheelSimple.json",
    "Right Wheel Input File":   "generic/wheel/WheelSimple.json",
    "Left Brake Input File":    "generic/brake/BrakeSimple.json",
    "Right Brake Input File":   "generic/brake/BrakeSimple.json"
  },
  {
    "Suspension Input File":    "generic/suspension/DoubleWishbone.json",
    "Suspension Location":      [-1.25, 0, -0.21],
    "Left Wheel Input File":    "generic/wheel/WheelSimple.json",
    "Right Wheel Input File":   "generic/wheel/WheelSimple.json",
    "Left Brake Input File":    "generic/brake/BrakeSimple.json",
    "Right Brake Input File":   "generic/brake/BrakeSimple.json"
  }
  ],

  "Steering Subsystems": [
  {
    "Input File":   "generic/steering/PitmanArm.json",
    "Location":     [1.1, 0, -0.4],
    "Orientation":  [0.98699637, 0, 0.16074256, 0]
  }
  ],

  "Driveline": {
    "Input File":           "generic/driveline/Driveline2WD.json",
    "Suspension Indexes":   [1]
  }
}
```

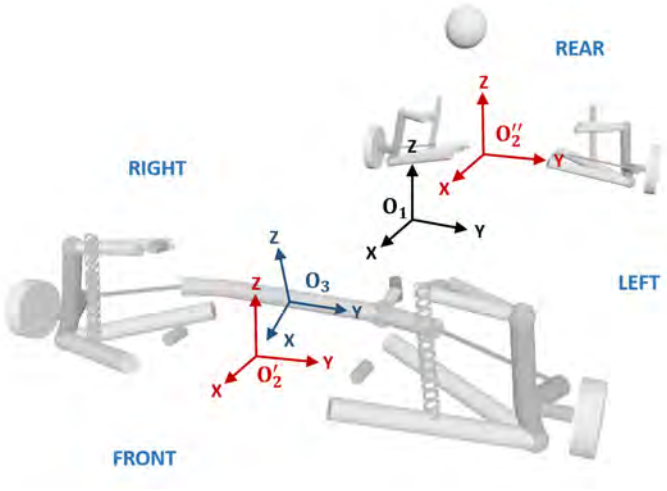


Figure 3: ISO vehicle reference frames.

be integrated in third-party applications which may provide alternative models for any or all of these *auxiliary* systems.

3.1.1 Terrain system templates

Chrono::Vehicle provides several classes of terrain and soil models, of different fidelity and computational complexity, ranging from rigid, to semi-empirical Bekker-Wong type models [29], to complex physics-based models based on either a granular or finite-element based soil representation.

The simplest model, suitable for many on-road vehicle test maneuvers, assumes a perfectly rigid terrain. Depending on the tire model employed (see Section 3.3), the tire-terrain interaction in this case reduces to either a simple height-normal query to the terrain system, or else falls back to the underlying Chrono frictional contact processing. In Chrono::Vehicle, a rigid terrain can be perfectly flat, provided as an arbitrary triangular mesh (specified as a Wavefront OBJ file), or defined through a height-map given as a gray-scale BMP image.

A second terrain template in Chrono::Vehicle provides a lower-fidelity, semi-empirical deformable soil model which is based on the Bekker set of parameters. This model is based on the Soil Contact Model (SCM) developed by Gibbesch and collaborators [5, 11]. In this context, soil is represented by a mesh whose deformation is achieved via vertical deflection of its nodes. Differently from the original SCM model, which uses regular grids, we extended this model to the case of non-structured triangular meshes. Moreover, to address memory and computational efficiency concerns, we implemented an automatic refinement of the mesh in order to create additional fine details where vehicle tires and track shoes interact with the soil. This soil model draws on the general-purpose collision engine in Chrono and its lightweight formulation allows computing vehicle-terrain contact forces in close to real-time.

Finally, leveraging the Chrono::FEA and Chrono::Granular modules, deformable soil can be modeled using either finite elements or granular material including contact,

friction, and cohesion. The former uses a continuum soil model based on multiplicative plasticity theory with Drucker-Prager failure criterion and a specialized 9-node brick element which alleviates locking issues with standard 8-node FEA brick elements without the need for numerical techniques such as enhanced assumed strain [32]. Physics-based deformable soil models based on granular dynamics simulations can be performed using the underlying Chrono support for the Discrete Element Method (DEM). Unlike continuum-based deformable terrain modeling approaches, DEM treats all component particles separately, as distinct entities, by maintaining and advancing in time their states while taking into account pair-wise interaction forces due to frictional contact. Chrono::Vehicle simulations on granular terrain can use either of the two methods supported in Chrono, namely a penalty-based, compliant-body approach, or a complementarity-based, rigid-body approach [27]. Since meaningful mobility simulations of vehicles on granular terrain typically result in DEM problems with millions of degrees of freedom, these are typically run on parallel hardware, using either the Chrono::Parallel module for a coupled vehicle-terrain simulation [20], or else Chrono::Distributed in a co-simulation framework [21, 24].

3.1.2 Driver system templates

Driver inputs (steering, throttle, and braking) are provided from a *driver* subsystem with available options in ChronoVehicle including interactive, data-driven, and closed-loop (e.g., path-following based on PID controllers).

The base C++ class for a driver system in Chrono::Vehicle imposes minimal requirements from a driver system template, in particular the ability to return throttle input (normalized in the $[0, 1]$ range), steering input (normalized in the $[-1, +1]$ range, with a negative value indicating steering to the left), and braking input (normalized in the $[0, 1]$ range). In addition, a driver system can receive information from any other system (e.g., the vehicle state) through its `Synchronize` method and may have internal dynamics (implemented in its `Advance` method). Specific templates for a driver system may extend the set of vehicle inputs generated, for example including the current selected gear for a manual transmission, enabling/disabling the cross-drive capability on a tracked vehicle, etc.

The Chrono::Vehicle library includes several templates for driver systems. For interactive simulations, run in soft real-time, we provide a template for a driver system which produces the vehicle inputs based on user controls, either keyboard and mouse, or a game controller. For design of experiment simulations, we provide a driver system template that is based on inputs provided through a text data file, with the vehicle inputs obtained through linear interpolation. Such data files can also be automatically generated by data collected during interactive runs.

Finally, Chrono::Vehicle includes several closed-looped driver system models, based on underlying support for PID controllers. These include speed controllers (which adjust the throttle and braking input simultaneously to maintain a constant vehicle speed) and path-follower controllers. The latter adjust the steering input so that the vehicle follows a user-defined path specified as a Bezier curve. A second type of closed-loop driver controllers, developed under the Chrono::CAVE module, provide vehicle inputs based on sensor data (LiDAR, GPS, IMU) for simulations involving connected and autonomous vehicles [3] and model-predictive controllers for obstacle-avoidance [8].

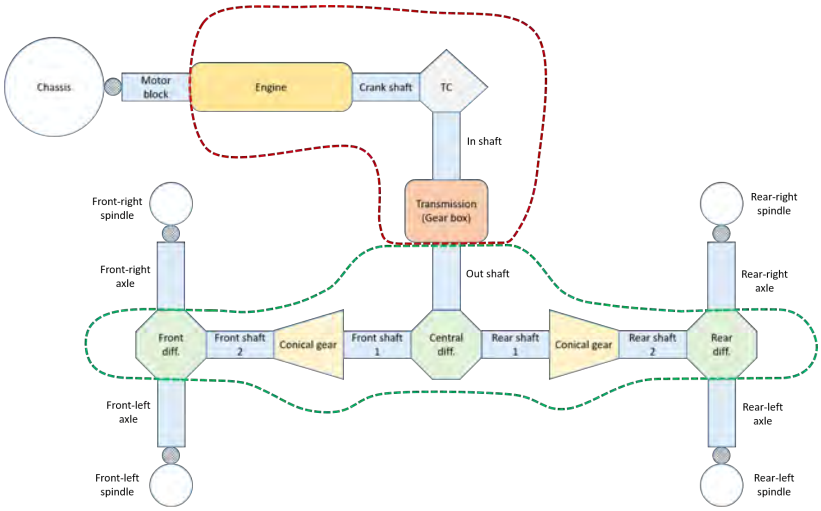


Figure 4: Schematic of a 4WD wheeled-vehicle powertrain. For modeling flexibility in Chrono::Vehicle, the engine, torque converter, and transmission are combined in a so-called *powertrain* system, while the drive shafts, transfer case, differentials, etc. are combined in the vehicle *driveline* subsystem.

3.1.3 Powertrain system templates

Although technically part of the vehicle itself, for additional modeling flexibility and to allow use of more sophisticated third-party engine models, Chrono::Vehicle collects the engine, torque converter, and transmission box into a distinct system, separate from the vehicle driveline, as shown in Fig. 4.

Several powertrain templates, of varying complexity, are provided with the Chrono software distribution. The simplest one uses a trivial engine torque-speed relationship, acting like a DC-motor and has no transmission. A more complex template uses a kinematic model based on arbitrary user-provided torque-speed engine curves and includes a simplified model of a (manual or automatic) transmission box with both reverse and forward gears.

The most sophisticated powertrain template provided in Chrono::Vehicle is based on 1-D shaft elements (i.e. modeling components that carry only rotational inertia) and specialized constraints between such elements and rigid bodies or between shaft elements. This template includes an engine model based on speed-torque curves (power and losses), torque converter based on capacity factor and torque ratio curves, and a transmission model parameterized by an arbitrary number of forward gear ratios and a single reverse gear ratio. The engine block can be mounted longitudinally or transversally and connected to the vehicle chassis to capture torque transfer effects.

3.2 Wheeled vehicles

A wheeled vehicle in Chrono::Vehicle is a specialization of the abstract generic *vehicle* subsystem and is defined as a collection of subsystems (see Fig. 5). A wheeled vehicle contains a chassis subsystem, a driveline subsystem, and an arbitrary number of axles which, by convention, are numbered starting at the front of the vehicle). Multi-steer vehicles are supported by allowing either an arbitrary number of steering mechanisms (which are

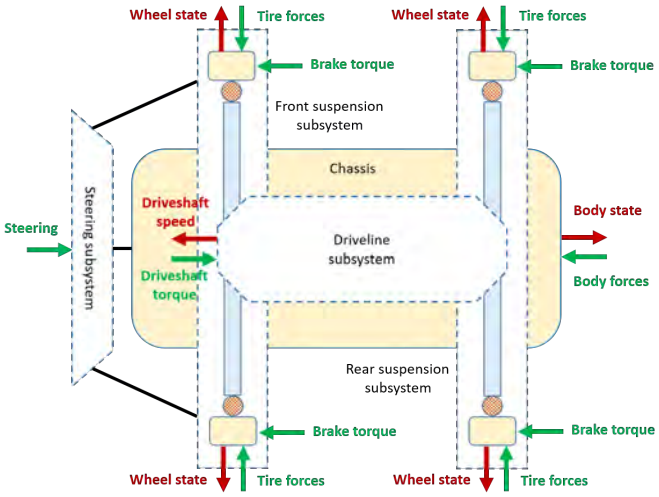


Figure 5: Decomposition of a wheeled vehicle into its subsystems. Also illustrated are the data exchanged with the systems external to the vehicle itself (driver, powertrain, and terrain).

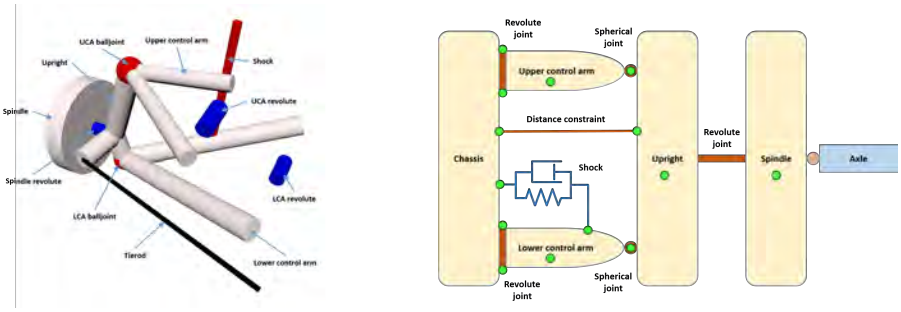
connected to different axles) or by allowing multiple steerable axles to be connected to the same steering mechanism.

This base wheeled vehicle configuration can be easily extended through user code to different topologies. Various demonstration programs distributed with the Chrono package [22], provide examples of such extensions, including tractor-trailer configurations and vehicles with an articulated chassis.

Suspension subsystem. To accommodate both independent and dependent suspension assemblies under a unified API, the Chrono::Vehicle convention is that a suspension subsystem includes both left and right sides. However, for symmetric components, any particular suspension template only requires parameters for the left side (with the right side obtained by automatic mirroring). Furthermore, the Chrono::Vehicle convention is that a suspension subsystem is defined with respect to a local reference frame that is aligned with the vehicle frame (in other words, only an offset is required to place a suspension assembly in a vehicle system).

Every suspension template defines its own set of bodies and joints and the topology of the mechanical system. Template parameters include locations of body center of mass locations, hardpoint locations, and possibly unit vectors for defining joint orientations. In addition, body masses and inertia tensors, as well as rotational inertias of the two axle shafts are template parameters. Figure 6a illustrates the modeling components for the double wishbone suspension template in Chrono::Vehicle which consists of two spindle bodies, two upright bodies, and four (lower and upper) control arms, connected through spherical and revolute joints, with the tie rods modeled using distance constraints. The hardpoint locations for this template are marked in Fig. 6b.

A Chrono::Vehicle user has complete freedom in defining the springs and shocks which, in most cases, need not be collinear. The suspension templates allow use of linear or non-linear spring and damper force elements, the latter defined either through lookup tables or else as arbitrary functions.



(a) Modeling components (b) Template parameters and hardpoints

Figure 6: Double wishbone suspension template. The subsystem topology and modeling elements are shown at left; the schematic at right illustrates the hardpoints defined by this template.

Besides the double wishbone template of Fig. 6, Chrono::Vehicle provides templates for multi-link, MacPhearson strut, solid-axle, semi trailing arm, Hendrickson PRIMAXX, and Polaris independent rear suspensions.

Steering subsystem. At the present time, two templates are provided for steering mechanisms: Pitman arm and rack-pinion. Unlike suspension templates, a steering mechanism can be offset and rotated when placed within a vehicle system. As mentioned before, a Chrono::Vehicle wheeled vehicle can contain an arbitrary number of steering mechanisms or connect more than one axle to a single steering mechanism.

Driveline subsystem. Chrono::Vehicle provides a simplified driveline model, suitable for 4WD vehicles, which uses a constant front/rear torque split and simple models of Torsen limited-slip differentials. In addition, two other templates, both using Chrono 1-D shaft elements and specialized connecting elements are provided, for 2WD and 4WD vehicles. Figure 7 shows a schematic of the shafts-based 4WD driveline template and its parameters, including the inertia of the various shafts and the gear ratios of the transfer case, differentials, and conical gears.

At initialization, a Chrono::Vehicle driveline subsystem is connected to one or two of the vehicle's axles.

Other wheeled vehicle subsystems. In addition to the main components mentioned above, Chrono::Vehicle includes templates for the brake subsystem (simplified model using a braking-input proportional torque), anti-roll bar subsystem (modeled with two bars connected through a rotational spring-damper), and the wheel subsystem (which is simply a carrier of additional mass and inertia to be compounded with that of the associated suspension spindle).

3.3 Tire models

Chrono::Vehicle currently supports three different classes of tire models: rigid, semi-empirical, and finite element. The rigid tires are the simplest of the three tire classes offered. The assumption for these models is that the tire is completely rigid and it interacts with the ground and any other rigid objects through the same underlying friction and contact algorithms as the other rigid bodies in Chrono. The contact geometry for these tires can be as simple as a cylinder or as complex as a 3D triangular mesh. These models are not

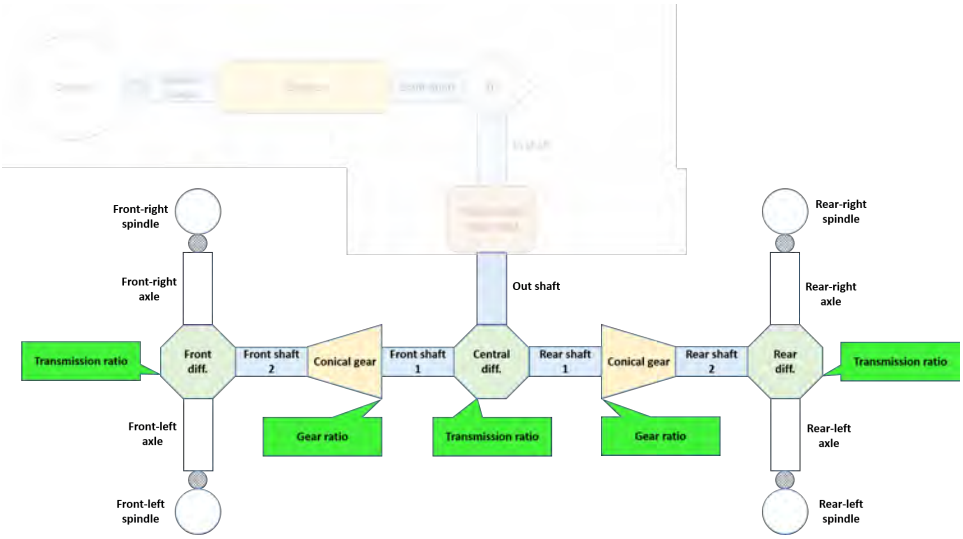


Figure 7: Schematic of the 4WD shafts-based driveline template for wheeled vehicles. The parameters of this template are the inertias of the various shaft and the gear ratios indicated in the figure.

only useful for debugging the overall vehicle model, but they can also be used in cases where runtime is important, the terrain is much softer than the tire, and a highly detailed model of the tire is unnecessary. Such examples can be found in the work by Wong and Reece [30, 31, 29].

The second class of tires models offered are the semi-empirical ones commonly used for vehicle handling. Chrono::Vehicle currently supports a LuGre friction based tire model, a Fiala tire model, and a Pacejka based model. The LuGre tire model implemented in Chrono::Vehicle is a multi-point contact tire model based on a series of stacked rigid discs. Each disc generates its normal load with respect to ground via a penalty method based on its penetration within the ground surface [14]. The longitudinal and lateral dynamics for each disc are based on a LuGre bristle based friction model [1]. Since tires typically have different longitudinal and lateral slip characteristics, the there are independent and uncoupled friction state equations for each direction. The force contributions from each disc are then summed and translated to the center of the wheel. Since multiple discs are used, the model is able to generate an overturning moment that would not exist if only a single disc was used. Like the other semi-empirical models, the friction and other model parameters for the LuGre tire model can be generated by fitting measured data.

The Fiala tire model implemented in Chrono::Vehicle is largely based on the transient Fiala tire model presented in the MSC ADAMS/tire help documentation [28, 17] which uses tire slip state equations to improve the model's behavior at slow to zero forward velocities. Like the LuGre tire model, the Fiala tire model is also based on a brush model assumption and only requires a small number of coefficients. Unlike the LuGre tire model, the Fiala tire model assumes that the tire is at zero camber with respect to the road and does not have any provisions for generating overturning moments. It does however couple the lateral and longitudinal slip states of the tire in its force and moment calculations, providing a more realistic description of combined slip than the uncoupled approach in the LuGre model.

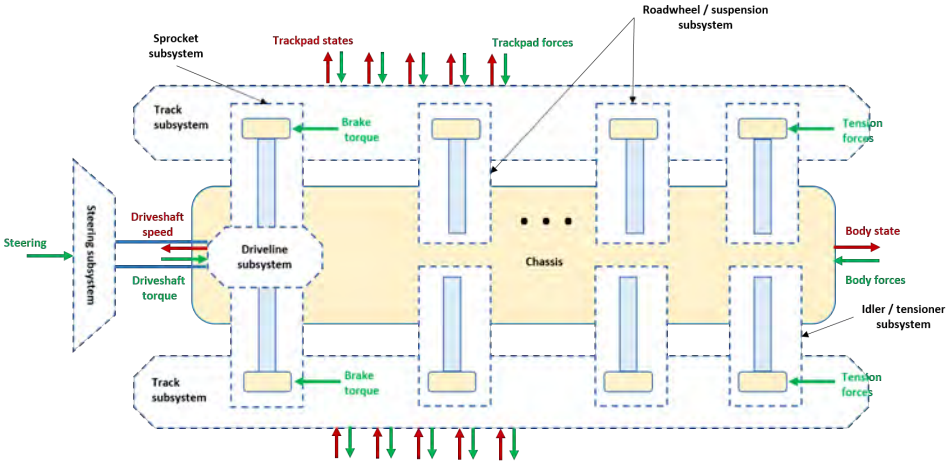


Figure 8: Decomposition of a tracked vehicle into its subsystems. Also illustrated are the data exchanged with the systems external to the vehicle itself (driver, powertrain, and terrain).

The third and most complex semi-empirical tire model offered is based off of the equations in Pacejka's "Tire and Vehicle Dynamics" book as well as the equations in the MSC ADAMS/tire help for the PAC2002 tire model [19, 17]. This model is an extension of Pacejka's earlier Magic Formula tire model with additional equations and coefficients. Since a large number of vehicle dynamics maneuvers do not occur under steady-state slip conditions, the contact patch slip state equations were included to provide more accurate results under transient conditions. The Chrono::Vehicle implementation of Pacejka's tire model has been validated against the tire test rig in MSC ADAMS/car [12].

Finally, the third class of tire models offered are full finite element representations of the tire. While these models have the potential to be the most accurate due to their detailed physical model of the tire, they are also the most computationally expensive of the tire model currently available in Chrono::Vehicle [21]. Unlike the rigid or semi-empirical tire models, the finite element based tire models are able to account for the flexibility in both the tire and in the ground at the same time, which is an important characteristic for many types of off-road mobility and vehicle dynamics studies. Since these finite element tire models leverage the nonlinear finite element capabilities in Chrono, tires have been modeled using co-rotational continuum elements, co-rotational shell elements, and ANCF shell elements.

3.4 Tracked vehicles

Similar to the wheeled vehicle, a track vehicle in Chrono::Vehicle is a specialization of the generic *vehicle* system and is defined as a hierarchy of subsystems, as illustrated in Fig. 8. Currently, a single topology of tracked vehicles is supported which includes, at the top-level a chassis subsystem, the vehicle driveline, a steering mechanism, and two track assembly subsystems. The latter are containers of further subsystems and each includes a sprocket mechanism, an idler tensioner subsystem, an arbitrary number of suspension components, and an arbitrary number of track-shoe assemblies.

To eliminate the burden of consistent initialization of the track shoe bodies, the track assembly subsystem provides algorithmic support for automatic assembly of the

track around the sprocket, idler, road-wheels, and any existing rollers. Specialized track assemblies and corresponding assembly routines are provided for different combinations of sprocket profiles and associated track shoe models.

A salient feature of tracked vehicles, which differentiates them from wheeled vehicles, is the important role played by contacts between internal components. While contact between track shoe bodies and the track wheels can be directly supported by the underlying frictional contact algorithms in Chrono, the sprocket-track shoe contact, involving complex and non-convex profiles, requires special treatment. In Chrono::Vehicle, sprocket profiles are defined as 2D curves (parameterized differently for sprockets engaging single- and double-pin track shoes) and the sprocket-track shoe contact is processed with a custom collision detection algorithm.

Track-shoe subsystem. Chrono::Vehicle offers templates for both single-pin and double-pin track shoes, each of which can have central or lateral guiding pins. The single-pin track shoe consists of a single body with non-trivial contact geometry which is connected to its neighbors through revolute joints. The double-pin track shoe template contains, in addition to the main track pad body, two additional connector bodies which are connected through revolute joints to the adjacent pads and which carry the contact geometry for collision with the track's sprocket gears.

All track shoe templates are fully parameterized in terms of dimensions, masses and inertias of the constituent bodies, as well as their contact geometry.

Sprocket subsystem. The sprocket subsystem connects the tracked vehicle driveline to the track assembly and is responsible for collision detection and contact processing with the track shoe bodies. A sprocket subsystem template implements the custom collision detection algorithm for a consistent pair of sprocket gear profile and associated track shoe. Chrono::Vehicle provides two templates for the sprocket subsystem, corresponding to the two types of supported track shoes, namely single-pin and double-pin. The sprocket gear profile is defined as a 2D path composed of line segments and circular arcs which are parameterized for each type of profile. Collision detection is performed in 2D, working in the plane of the sprocket gear, but contact forces are calculated in 3D before being applied to the sprocket and interacting track shoe bodies.

In addition to the gear profile, a sprocket template is parameterized by the mass and inertia of the sprocket body, the rotational inertia of the sprocket axle, and the separation distance between the two gears.

Suspension templates. Different suspension configurations are available, including torsion spring with linear or rotational dampers and a hydropneumatic suspension template. A track assembly can contain an arbitrary number of suspension subsystems which, for the templates using a torsion spring, may or may not include a damper. A Chrono::Vehicle suspension subsystem also contains a road-wheel, themselves templatized based on the type of track shoe used (central or lateral guiding pins).

Similar to the case of wheeled vehicle, a tracked vehicle suspension template allows complete freedom in specifying spring and damper forces which can be linear or non-linear, defined through table lookup or implemented in user-provided C++ functions.

Idler subsystem. A Chrono::Vehicle idler mechanism consists of the idler wheel and a connecting body. The idler wheel is connected through a revolute joint to the connecting body which in turn is connected to the chassis through a translational joint. A linear actuator acts as a tensioner which is modeled as a general spring-damper with optional preload. An idler subsystem is defined with respect to a frame centered at the origin of the idler wheel,

optionally pitched relative to the chassis reference frame. The translational joint is aligned with the X axis of this reference frame, while the axis of rotation of the revolute joint is aligned with its Y axis.

Like the road-wheels, different templates are provided for the case of tracks with central or lateral guiding pins.

3.5 Visualization and post-processing

Chrono::Vehicle provides visualization support both for run-time interactive simulations, as well as for high-quality post-processing rendering for generating animations. Currently, run-time simulation support expands on the underlying Chrono::Irrlicht module for sequential simulations [9] or the more computationally efficient but more limited Chrono::OpenGL module for parallel simulations involving large-scale granular terrain representations [10]. Ongoing development effort involves a transition to an Ogre-based run-time visualization system [15], as a replacement for Chrono::Irrlicht. Support for ray-traced renderings of individual simulation frames is offered through utility functions that can be called from within the simulation loop to export data files with current visualization assets information and a POV-Ray script [18] that can batch-process these files to generate frame images.

Extraction of simulation results at vehicle-, subsystem-, or model component-level is currently limited to using the Chrono::Vehicle C++ API which provides an exhaustive set of methods for this purpose, as well as utility for data filtering, I/O, etc. A specialized module is currently under development to provide a formal mechanism for extracting a list of all available output channels and metrics of interest from a vehicle simulation, automatic data collection during the simulation, and final reporting and plotting of user-specified quantities of interest.

4 Ground vehicle mobility simulations with Chrono::Vehicle

Chrono::Vehicle has been used extensively in many ground vehicle mobility studies, both at the University of Wisconsin - Madison, by our external collaborators, and by external Chrono users. Some typical simulations, involving both wheeled and tracked vehicles, are illustrated in Fig. 9.

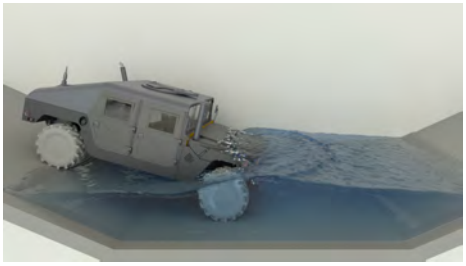
The simulation snapshot in Fig. 9a is an example of a standard mobility test on rigid flat terrain. The double lane change in this demonstration used a 4WD wheeled vehicle with Fiala tire models and a path-follower driver system with a constant-speed controller. The relevant mobility metrics extracted from such a simulation include the maximum speed at which the maneuver can be safely performed, accelerations at the driver location, and the vehicle inputs generated by the closed-loop driver system.

Figure 9b is a snapshot from a step-climbing validation test for determining the maximum obstacle height that can be managed by a tracked vehicle from rest. The vehicle used in this test contains over 150 bodies and is modeled with single-pin track shoes and linear-damper suspensions.

In conjunction with the Chrono::FSI module, Chrono::Vehicle has been used in fluid-solid interaction problems to simulate fording maneuvers and sloshing of liquids in vehicle-mounted tanks [13]. These simulations capture the two-way coupling between the vehicle dynamics and the fluid phase, the latter being governed by the mass and momentum (Navier-Stokes) equations discretized in space via SPH. The simulation illustrated in Fig. 9c involved

(a) Double-lane change maneuver.

(b) Obstacle climbing test.



(c) Fording maneuver.

(d) Simulating a convoy of autonomous vehicles.

Figure 9: Snapshots from various Chrono::Vehicle-enabled simulations.

a 4WD wheeled vehicle with a constant-speed controller negotiating a body of water and fully accounts for the interaction between the fluid phase and the vehicle chassis and tires (for collision detection purposes, the chassis and tire meshes were decomposed in sets of convex hulls). This simulation used approximately 1.5 million SPH particles for the fluid phase and was carried out with Chrono::Parallel using 40 OpenMP.

Figure 9d is a snapshot from a simulation with Chrono::CAVE implementing functionality from Chrono::Vehicle, multiple agents, sensors, and a virtual world [3]. This setup involves four vehicles simulated on separate network clients, all connected to a single server. The lead vehicle is interactively driven and the three trailing vehicles are autonomous. The latter are equipped with virtual LiDAR, GPS, and IMU sensors which are used in a simple driver controller to follow the vehicle ahead. The outputs of these sensors are overlaid (LiDAR on the left and GPS/IMU combined on the right) for the last vehicle in the convoy.

As examples of other types of simulations enabled by Chrono::Vehicle we mention studies of wheeled vehicles with flexible tires on granular terrain [21, 24] to investigate the effect of tire and terrain deformation on mobility metrics; and design of model predictive controllers for obstacle avoidance on deformable terrain [8] to investigate controller performance and robustness. Chrono::Vehicle is an integral part of the Mercury framework for mobility simulation of wheeled ground vehicles [7].

5 Conclusions and future developments

We provided a brief overview of the Chrono::Vehicle library which provides support for ground vehicle modeling, simulation, and visualization within the multi-physics package Chrono. This module is designed in a modular manner, using a template-based approach to allow rapid prototyping of existing and new vehicle configurations

and to facilitate its integration in third-party simulation frameworks. The simulation loop in Chrono::Vehicle is structured such that it permits both monolithic coupled vehicle simulations and co-simulation for large-scale vehicle–terrain–environment multi-physics and multi-scale simulation.

Current and planned development involves inclusion of additional subsystem templates, such as support for continuous rubber band tracked vehicles, as well as new capabilities, primarily related to formalization of data collection, output of mobility metrics, and post-processing.

Complex vehicle–terrain simulations, particularly those involving high-fidelity FEA-based flexible tires, granular terrain, and multi-phase FSI problems, continue to pose serious challenges, including the still considerable computational requirements. Plans for future research and development in this area focus on leveraging hybrid parallel computing as enabled by the interplay of Chrono::Distributed and Chrono::Parallel in a co-simulation framework.

References

[1] N. B. Do, A. A. Ferri, and O. A. Bauchau. Efficient simulation of a dynamic system with lugre friction. *Journal of Computational and Nonlinear Dynamics*, 2(4):281–289, 2007.

[2] ECMA. The JSON data interchange format. Technical Report ECMA-404, ECMA International, 2013.

[3] A. Elmquist, D. Hatch, C. Ricchio, R. Serban, and D. Negrut. Virtual autonomous vehicle testing via a Connected Autonomous Vehicle Emulator (CAVE). In J. Michopoulos, D. Rosen, C. Paredis, and J. Vance, editors, *Advances in Computers and Information in Engineering Research*. ASME Press, New York, 2017.

[4] FunctionBay. Recurdyn. `http://eng.functionbay.co.kr`. Accessed: 2015-02-07.

[5] A. Gibbesch and B. Schafer. Multibody system modelling and simulation of planetary rover mobility on soft terrain. In *8th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS 2005), Munich, Germany, September*, pages 5–8, 2005.

[6] R. A. Gingold and J. J. Monaghan. Kernel estimates as a basis for general particle methods in hydrodynamics. *Journal of Computational Physics*, 46(3):429–453, 1982.

[7] C. Goodin, J. Mange, S. Pace, T. Skorupa, D. Kedziorek, J. Priddy, and L. Lynch. Simulating the mobility of wheeled ground vehicles with mercury. *SAE Int. J. Commer. Veh.*, 10(2), 2017.

[8] N. Haraus, R. Serban, and J. Fleischmann. Performance analysis of constant speed local obstacle avoidance controller using an MPC algorithm on granular terrain. In *Ground Vehicle Systems Engineering and Technology Symposium*, 2017.

[9] Irrlicht. Open Source 3D Irrlicht Engine. `http://irrlicht.sourceforge.net/`, 2014.

[10] Khronos Group. Open Graphics Library - OpenGL. `http://www.opengl.org/`, 2014.

[11] R. Krenn and A. Gibbesch. Soft soil contact modeling technique for multi-body system simulation. In *Trends in computational contact mechanics*, pages 135–155. Springer, 2011.

[12] J. Madsen. Validation of a Single Contact Point Tire Model Based on the Transient Pacejka Model in the Open-Source Dynamics Software Chrono. Technical Report TR-2014-16: `http://sbel.wisc.edu/documents/TR-2014-16.pdf`, Simulation-Based Engineering Laboratory, University of Wisconsin-Madison, 2014.

[13] H. Mazhar, A. Pazouki, M. Rakhsha, P. Jayakumar, and D. Negrut. A differential variational approach for handling fluid-solid interaction problems via Smoothed Particle Hydrodynamics. *Journal of Computational Physics (under review)*, 0:0, 2017.

[14] A. Mikkola. Lugre Tire Model for HMMWV. Technical Report TR-2014-15: `http://sbel.wisc.edu/documents/TR-2014-15.pdf`, Simulation-Based Engineering Laboratory, University of Wisconsin-Madison, 2014.

[15] I. Milne and G. Rowe. OGRE-3D program visualization for C++. In *Proceedings of the 3rd Annual LTSN-ICS Conference*, 2002.

[16] MSC Software. ADAMS. `http://www.mscsoftware.com/product/adams`. Accessed: 2015-02-07.

[17] MSC Software. ADAMS/Tire help - ADAMS 2015. `https://simcompanion.mscsoftware.com/infocenter/index?page=content&id=DOC10813&cat=2015_ADAMS_DOCS&actp=LIST/`, July 2015.

[18] Persistence of Vision Pty. Ltd. Persistence of Vision (TM) Raytracer. `http://www.povray.org`, 2004.

[19] H. B. Pacejka. *Tire and vehicle dynamics*. Elsevier, 2005.

[20] A. Pazouki, M. Kwarta, K. Williams, W. Likos, R. Serban, J. Jayakumar, and D. Negrut. Compliant versus rigid contact – a comparison in the context of granular dynamics. *Phys. Rev. E (under review)*, 0:0, 2017.

[21] A. M. Recuero, R. Serban, B. Peterson, H. Sugiyama, P. Jayakumar, and D. Negrut. A high-fidelity approach for vehicle mobility simulation: Nonlinear finite element tires operating on granular material. *Journal of Terramechanics*, 72:39 – 54, 2017.

[22] Project Chrono Development Team. Chrono: An Open Source Framework for the Physics-Based Simulation of Dynamic Systems. `https://github.com/projectchrono/chrono`. Accessed: 2017-05-07.

[23] Siemens PLM Software. Siemens Virtual.Lab. `https://www.plm.automation.siemens.com/en_us/products/lms/virtual-lab/`. Accessed: 2016-09-13.

[24] R. Serban, N. Olsen, D. Negrut, A. M. Recuero, and P. Jayakumar. A co-simulation framework for high-performance, high-fidelity simulation of ground vehicle-terrain interaction. In *AVT-265: Integrated Virtual NATO Vehicle Development*, Vilnius, Lithuania, May 2017.

[25] A. A. Shabana and R. Y. Yakoub. Three dimensional absolute nodal coordinate formulation for beam elements: Theory. *ASME Journal of Mechanical Design*, 123:606–613, 2001.

[26] J. C. Simo and L. Vu-Quoc. Three-dimensional finite-strain rod model. part ii: Computational aspects. *Computer Methods in Applied Mechanics and Engineering*, 58:79–116, 1986.

[27] 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.

[28] M. Taylor. Implementation and validation of the Fiala tire model in Chrono. Technical Report TR-2015-13: `http://sbel.wisc.edu/documents/TR-2016-06.pdf`, University of Wisconsin–Madison, 2015.

[29] J. Y. Wong. *Theory of Ground Vehicles*. John Wiley & Sons, New York, 2001.

[30] J. Y. Wong and A. R. Reece. Prediction of rigid wheel performance based on the analysis of soil-wheel stresses part i. performance of driven rigid wheels. *Journal of Terramechanics*, 4(1):81–98, 1967.

[31] J. Y. Wong and A. R. Reece. Prediction of rigid wheel performance based on the analysis of soil-wheel stresses: Part ii. performance of towed rigid wheels. *Journal of Terramechanics*, 4(2):7–25, 1967.

[32] H. Yamashita, P. Jayakumar, and H. Sugiyama. Modeling of deformable tire and soil interaction using multiplicative finite plasticity for multibody off-road mobility simulation. In *Proceedings of the ASME 2016 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference IDETC/CIE 2016*, 2016.