Mech. Sci., 4, 49–64, 2013
www.mech-sci.net/4/49/2013/
doi:10.5194/ms-4-49-2013

Mechanical Sciences
Open Access

Chrono: a parallel multi-physics library for rigid-body, flexible-body, and fluid dynamics

H. Mazhar[1], T. Heyn[1], A. Pazouki[1], D. Melanz[1], A. Seidl[1], A. Bartholomew[1], A. Tasora[2], and D. Negrut[1]

[1]Simulation Based Engineering Lab, Department of Mechanical Engineering, University of Wisconsin, Madison, WI, 53706, USA
[2]Department of Industrial Engineering, University of Parma, V.G.Usberti 181/A, 43100, Parma, Italy

Correspondence to: D. Negrut (negrut@engr.wisc.edu)

Received: 16 November 2012 – Accepted: 26 January 2013 – Published: 12 February 2013

Abstract. The last decade witnessed a manifest shift in the microprocessor industry towards chip designs that promote parallel computing. Until recently the privilege of a select group of large research centers, Teraflop computing is becoming a commodity owing to inexpensive GPU cards and multi to many-core x86 processors. This paradigm shift towards large scale parallel computing has been leveraged in Chrono, a freely available C++ multi-physics simulation package. Chrono is made up of a collection of loosely coupled components that facilitate different aspects of multi-physics modeling, simulation, and visualization. This contribution provides an overview of Chrono::Engine, Chrono::Flex, Chrono::Fluid, and Chrono::Render, which are modules that can capitalize on the processing power of hundreds of parallel processors. Problems that can be tackled in Chrono include but are not limited to granular material dynamics, tangled large flexible structures with self contact, particulate flows, and tracked vehicle mobility. The paper presents an overview of each of these modules and illustrates through several examples the potential of this multi-physics library.

1 Introduction

Over the last decade there has been a manifest trend in the hardware industry to increase flop rates by increasing the number of cores available on a processor. To a very large extent, the tide that propelled sequential computing for several decades is subsiding. The frequency at which cores are operated today has at best plateaued; in many cases, it went down in an attempt to tame power dissipation and overheating. Instruction level parallelism advances that ensured respectable gains through pipelining and out of order execution have largely fulfilled their potential. The bright spot in this evolving hardware landscape has been the growing impetus behind parallel computing hardware. If anything has held steady over the last four decades, it has been the pace at which transistors are packed per unit area in computer chips. This trend allows today chip designs that draw on 22 nm feature length. Intel's road map calls for 14 nm technology in 2014, 10 nm in 2016, 7 nm in 2018, and 5 nm in 2020. In other words, the number of transistors per unit area will continue to double every two years for the current decade. This will translate into immediate access to commodity chips that host multiple compute cores. Given the stagnation in processor operating frequency, an ever growing gap between CPU speed and memory speed, and the waning of instruction level parallelism gains, it becomes apparent that the only way we can continue to enjoy reduced simulation times or ability to rely on refined models is to fall back on parallel computing. There are two major directions in which parallel computing has evolved. The x86 architecture has defined a solution that evolved as a steady and predictable process in which the number of cores on a chip increased over time: AMD produces today 16 core chips, while Intel has 12 core processors. Leveraging these chips requires a low entry point that calls for programming against relatively mature libraries such as OpenMP, MPI, pthreads, cilk, TBB, etc. At memory bandwidths of 75 GB s^{-1} and flop rates of 0.3 TFlop s^{-1}, this has traditionally represented the conservative choice for entering the parallel computing arena. With the release of CUDA 1.0 in 2006, NVIDIA offered a second alternative

to leveraging parallel computing by programming the ubiquitous video cards available on millions of desktops worldwide. This path to parallel computing is less conventional as it requires one to get familiar with the hardware layout and memory hierarchy associated with GPUs. Today, an Nvidia GPU has close to seven billion transistors. Priced at about $6000, an Nvidia Kepler K20x delivers a memory bandwidth of 250 GB s^{-1} and 1.3 TFlop s^{-1} by virtue of using more than 2800 Scalar Processors. It is used side by side with a regular CPU processor, which means that heterogeneous computing, on the CPU and GPU, can lead to substantial speed gains. In this framework, the GPU plays the role of an accelerator by boosting the floating point performance of the CPU. A similar setup is offered now by Intel; i.e., CPU plus accelerator, owing to its recent release of the Knights Corner architecture. A Knights Corner chip has about 60 cores, can deliver up to 320 GB s^{-1} and 1 TFlop s^{-1}, and uses the x86 instruction set architecture, which translates into an easier adoption path provided one is familiar with OpenMP or MPI.

It becomes apparent that in the immediate future, any increase in simulation speed or model complexity in Computational Science will be fueled by parallel computing. This paper outlines an ongoing effort in the area of computational mutlibody dynamics that is motivated by this belief. It starts with a description of a core simulation engine that aims at simulation of many-body dynamics problems with friction and contact. Chrono::Engine handles both rigid and flexible bodies and draws on MPI and/or GPU computing. It then discusses Chrono::Fluid, a GPU parallel simulation tool that aims at fluid-solid interaction problems, which is singled out as an application area that has been largely ignored until recently due to an excessive computational burden incurred by the simulation of systems of practical relevance. Finally, the papers outlines a rendering pipeline that is used for postprocessing of big data. Chrono::Render is capable of using 320 cores and is built around Pixar's RenderMan. All these components combine to produce Chrono, a multi-physics simulation environment that is designed to take advantage of commodity parallel computing made available by many-core and GPU architectures.

2 Chrono::Engine

The Chrono::Engine software is a general-purpose simulator for three dimensional multi-body problems (Tasora and Anitescu, 2011). Specifically, the code is designed to support the simulation of very large systems such as those encountered in granular dynamics, where the number of interacting elements can be in the millions. Target applications include tracked vehicles operating on granular terrain (Heyn, 2009) or the Mars Rover operating on discrete granular soil. In these applications, it is desirable to model the granular terrain as a collection of many thousands or millions of discrete bodies interacting through contact, impact, and friction. Note that such systems also include mechanisms composed of rigid bodies and mechanical joints. These challenges require an efficient and robust simulation tool, which has been developed in the Chrono simulation package. Chrono::Engine was initially developed leveraging the Differential Variational Inequality (DVI) formulation as an efficient method to deal with problems that encompass many frictional contacts – a typical bottleneck for other types of formulations (Anitescu and Tasora, 2010; Tasora and Anitescu, 2010). This approach enforces non-penetration between rigid bodies through constraints, leading to a cone-constrained quadratic optimization problem which must be solved at each time step (Negrut et al., 2012). Chrono::Engine has since been extended to support the Discrete Element Method (DEM) formulation for handling the frictional contacts present in granular dynamics problems (Cundall, 1971; Cundall and Strack, 1979). This formulation computes contact forces by penalizing small interpenetrations of colliding rigid bodies. Various contact force models can be used depending on the application (Mindlin and Deresiewicz, 1953; Kruggel-Emden et al., 2007).

The remainder of this section describes the features of Chrono::Engine, starting with the structure of the code. Next, several sub-sections describe the use of GPU computing in the collision detection task, the use of MPI for distributed solution of large systems, and validation work which has been done to assess the accuracy of the simulation tool.

2.1 Code structure of Chrono::Engine

The core of Chrono::Engine is built around the concept of middleware, namely a layer of classes and functions that can be used by third-party developers to create complex mechanical simulation software with little effort (Tasora et al., 2007). Because of this, graphical user interfaces and end-user tools are not the main focus of the Chrono::Engine core project; it is assumed that programs with graphical interfaces are built on top of such middleware, or should be considered as additional, or optional, modules.

Given the complexity of the project, approaching half a million lines of code, the software is organized in classes and namespaces as recommended by the Object Oriented Programming paradigm, targeting modularity, encapsulation, reusability and polymorphism. The libraries of Chrono::Engine are thread safe, fully re-entrant, and include more than six hundred C++ classes. Objects from these classes can be instantiated and used to define models and simulations that run in third party software, for instance vehicle simulators, CAD tools, virtual reality applications, or robot simulators.

Chrono::Engine is completely platform-independent, hence libraries are available for Windows, Linux and Mac OSx, for both 32 bit and 64 bit versions. Moreover, we followed a modular approach, splitting the libraries in modules that can be dynamically loaded only if necessary, thus

minimizing issues of dependency from other libraries and reducing memory footprint. For instance, we developed libraries for MATLAB interoperability, for real-time visualization through OpenGL, for interfacing with post-processing tools, etc. (see Fig. 1).

Classes and objects have been tested and profiled for fast execution, in order to achieve real-time performance when possible. Modern programming techniques have been adopted, like metaprogramming, class templating, class factories, memory leak trackers and persistent-transient data mapping. C++ operator overloading has been used to provide a compact algebra to manage quaternions, static and moving coordinate systems, and OS-agnostic classes are used for logging, streaming/checkpointing and exception handling.

We embraced an intense object-oriented approach, therefore most C++ objects that define parts of the multibody model are inherited from a base class called `ChPhysicsItem`, which defines the essential interfaces for all items that have some degrees of freedom. For example, specialized classes that inherit the `ChPhysicsItem` are the `ChBody` class, which is used for 3-D rigid bodies as shown in Fig. 2, `ChShaft`, which is used for 1-D concentrated parameter models of power trains, `ChLinkLockRevolute` that is a joint between rigid bodies, and so on. A set of more than thirty mechanical constraints are part of this class hierarchy. Furthermore, the architecture is open to further definition of new specialized classes for user-customized parts and joints. An object of `ChSystem` class stores a list of all moving parts and performs the simulation.

Each `ChPhysicsItem`-inheriting class can encapsulate a variable number of `ChLcpVariable` objects and/or a variable number of `ChLcpConstraint` objects, that are fed to the solver for Cone Complementarity Problems (CCP) at each time step of the DVI integration; this helps the development of black-box CCP solvers that are independent from the data structures of the physical layer. Also, these data structures represent the sparse data for the model description, which is completely matrix- and vector-free for the sake of a small memory footprint and fast linear algebra. Specifically, tthe system matrices for mass, Jacobians, etc. are never explicitly assembled. The objects of most of the above mentioned classes are managed by smart (shared) pointers with automatic deletion.

This relieves the programmer from the burden of taking care of object's lifetime, given that the relationships between objects can be quite complex as illustrated in Fig. 3. A large portion of the C++ classes are available also as Python modules; this enables the use of most simulation features in a scripted environment. Since novice users are more comfortable with Python than with C++, the Python interface proved to be optimal for teaching purposes. The Python interface was produced using the SWIG utility, a process that automatically generates the code for the Python wrapper.

The software architecture has been designed to accommodate an expandable system for handling assets (meshes, textures, CAD models), with multiple paths from pre-processing to post-processing. To this end, we also provide a C# add-in for a parametric 3-D CAD package (SolidWorks) that can be used to export models into Chrono::Engine without programming efforts (see Fig. 4).

2.2 Collision detection in Chrono::Engine

This section describes the collision detection algorithm designed and implemented for the Chrono::Engine package. Recall that problems of interest are focused on granular dynamics, such as sand flowing inside an hourglass, a rover running over sandy terrain, an excavator/frontloader digging/loading granular material, etc. In this context, the collision detection task is performed on a rather small collection of rigid and/or deformable bodies of complex geometry (hourglass wall, wheel, track shoe, excavator blade, dipper), and a very large number of bodies (millions to billions) that make up the granular material. On this scale, the collision detection task, particularly when dealing with the granular material, fits perfectly the Single Instruction Multiple Data (SIMD) computation paradigm. Specifically, the same sequence of instructions needs to be applied to every individual body and/or contact in the granular material. Therefore, a collision detection algorithm capable of leveraging the SIMD computational power of commodity Graphics Processing Units (GPUs) was developed and implemented to remove collision detection as the bottleneck in large granular dynamics simulations.

The parallel collision detection algorithm is separated into two phases, broadphase, and narrowphase. The broadphase algorithm quickly determines a list of potential contact pairs while the narrowphase algorithm determines actual contact information. A brief outline of the parallel collision detection algorithm is presented below, for more details see (Mazhar et al., 2011; Pazouki et al., 2012, 2010).

2.2.1 Broad-Phase algorithm

The Broad-Phase algorithm is used to compute whether two bodies might be in contact at a given time. The purpose of the broad-phase algorithm is not to find actual contact information, but rather to determine if a contact could potentially occur based on the Axis Aligned Bounding Boxes of the bodies involved.

An Axis Aligned Bounding Box (AABB) is a special case of a bounding box that is always aligned to the global reference frame, simplifying collision detection as the bounding box cannot rotate. Because of this, the volume enclosed by the bounding box will always be equal to or greater than the volume of the shape it encloses. AABB generation is simple and can be easily paralellized on a per object basis. See Fig. 5 for an example of AABB computation for a cylinder in 3-D space.

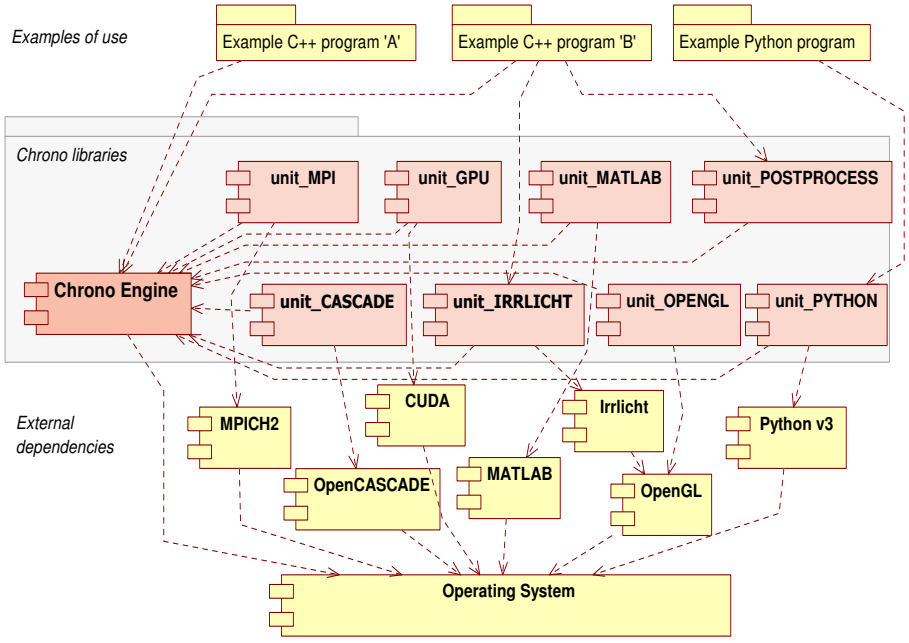


Figure 1. UML graph of dependencies between module libraries.

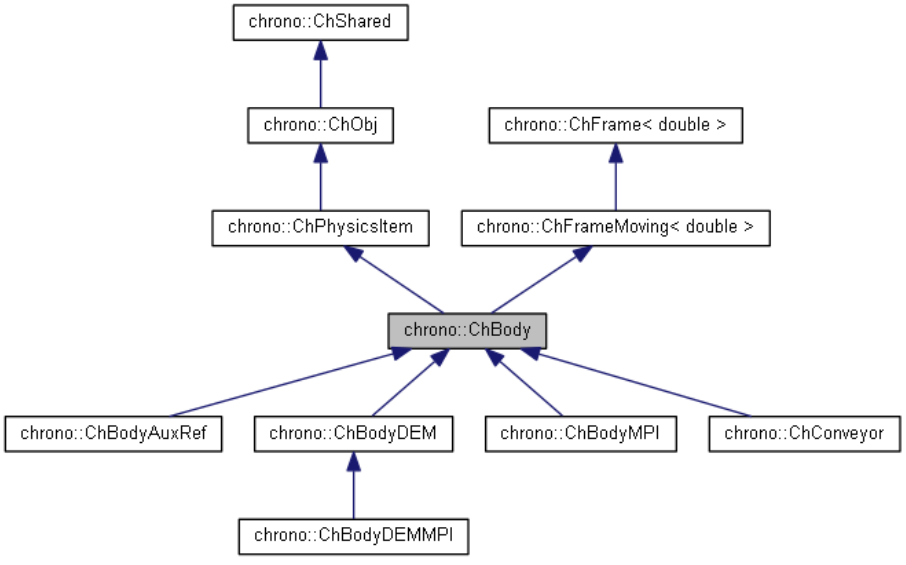


Figure 2. Class ineritance diagram for objects of `ChBody` type.

2.2.2 Spatial Subdivision algorithm

A high-level overview of the GPU-based collision detection is as follows. The collision detection process starts by identifying the intersections between AABBs and bins (see Fig. 6 for a visual representation of a bin). The AABB-bin pairs are subsequently sorted by bin id. Next, each bin's starting index is determined so that the bins' AABBs can be traversed sequentially. All AABBs touching a bin are subsequently checked against each other for collisions.

2.2.3 Narrow-Phase algorithm

Once potential contacts have been determined from the broad-phase collision detection stage, the Narrow-Phase algorithm needs to process each possible contact and determine if it actually occurs. To this end an algorithm capable of determining contacts between convex geometries was implemented on the GPU. This algorithm, called "XenoCollide" (Snethen, 2007), is based upon Minkowski Portal Refinement (MPR) (Snethen, 2008).

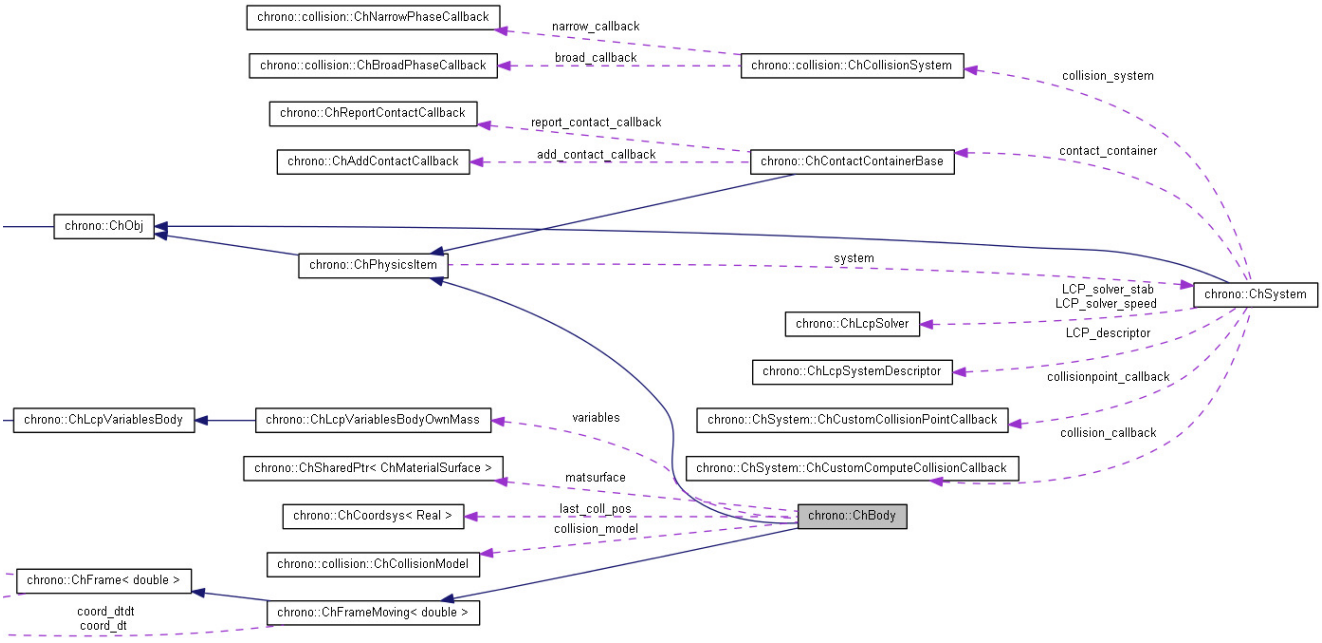


Figure 3. Collaboration graph between classes: example for `ChBody` and `ChSystem`.

2.3 Using MPI for distributed Chrono

Chrono has been further extended to allow the use of CPU parallelism for certain problems. To efficiently simulate large systems, a domain decomposition approach has been developed to allow the use of many-core compute clusters. In this approach, we divide the simulation domain into a number of sub-domains in a lattice structure. Each sub-domain manages the simulation of all bodies contained therein. Note that bodies may span the boundary between adjacent sub-domains. In this case, the body is considered shared and its dynamics may be influenced by the participating sub-domains. The implementation leverages the MPI standard (Gropp et al., 1999) to implement the necessary communication and synchronization between sub-domains.

This approach enables the simulation of large systems in two ways. First, it relies on the power of parallel computing since one computer core can be assigned to each MPI process (and therefore to each sub-domain). These processes can execute in parallel, constrained only by the required communication and synchronization. Second, it allows access to the larger memory pool available on distributed memory architectures. Whereas a single node or GPU card may have about 6 GB of memory, a distributed memory cluster may have on the order of 1 TB of memory, enabling the simulation of vastly larger problems.

Note that the domain decomposition approach currently uses the discrete element method to resolve friction and contact forces between elements in the system. The approach also supports constraints between bodies in the simulation by considering an assembly of constrained rigid bodies as a unit which must always be kept together. Therefore, if any body in a chain of constrained bodies is contained in a given sub-domain, all bodies in the chain are considered by that sub-domain and used to correctly solve the constraint equations.

2.3.1 Sub-division and set-up

A pre-processing step is used to discretize the simulation domain into a specified number of sub-domains, set up the communication conduits between processes, and initialize the sub-domains as appropriate. The sub-division is based on a cubic lattice with support for arbitrary sized divisions. The sub-domain boundaries are aligned with the global cartesian coordinate system, and their locations are user-specified. Separate MPI processes are mapped to each sub-domain. Note that at this time, the sub-division is static and does not change during the simulation. Therefore, the user should be careful to set up the discretization to maintain the best possible load balancing.

In terms of communication, each sub-domain in the grid can communicate with all other sub-domains. These communication pathways are set up and initialized during the preprocessing step and persist throughout the simulation.

Note that this implementation relies heavily on inheritance and the class-based structure of Chrono. For example, `ChSystem` is extended to `ChSystemMPI` by including the code to perform communication and synchronize the subdomains.

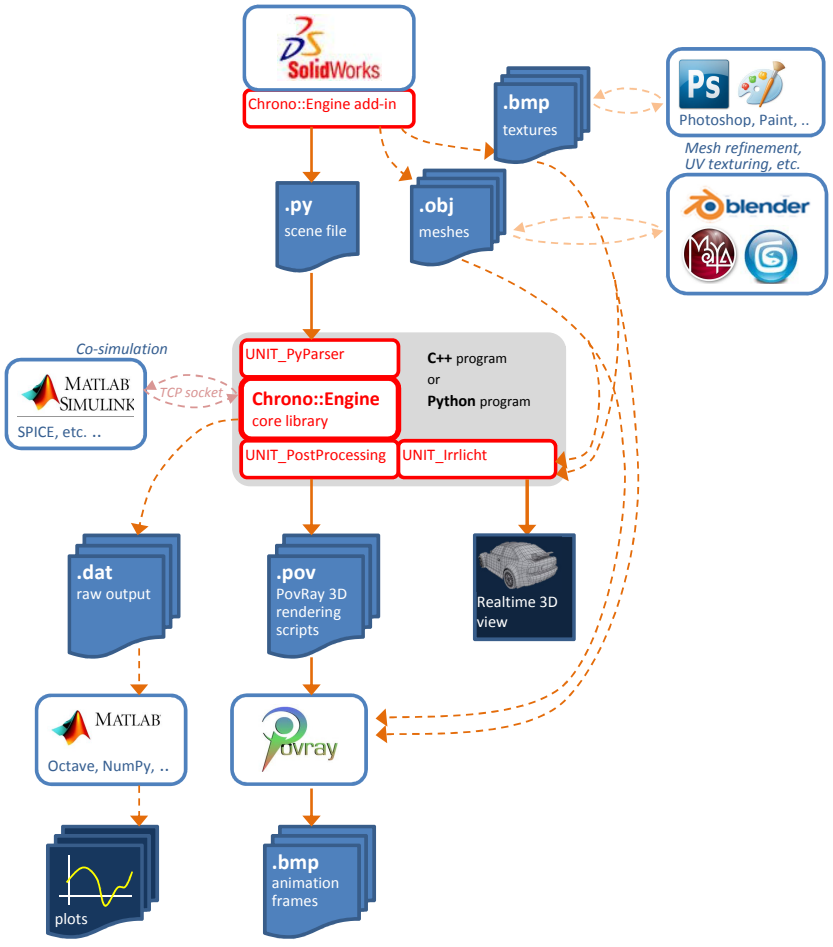


Figure 4. Network of asset workflows.

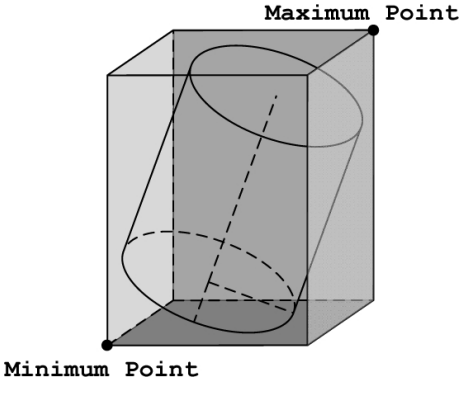


Figure 5. Example of AABB generation for 3-D cylinder.

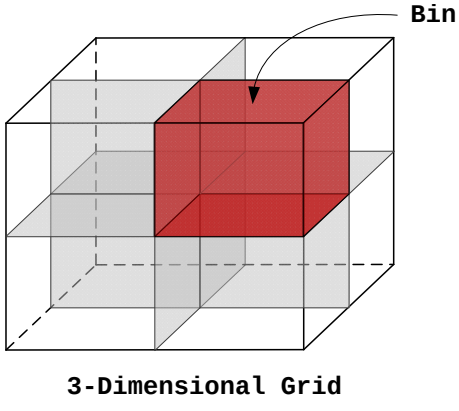
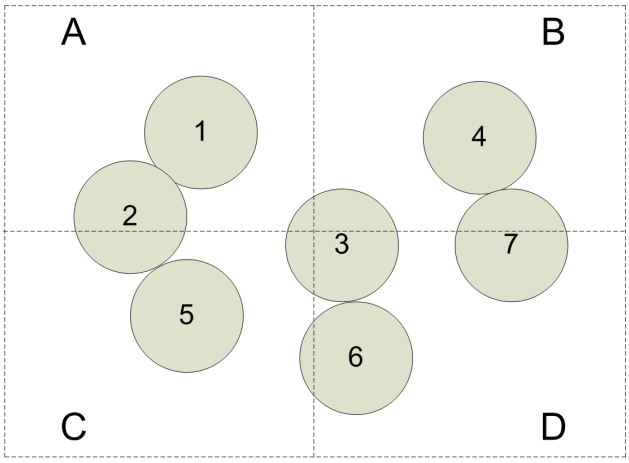


Figure 6. Example of 3-D space divided into bins.

2.3.2 Simulation and communication

Each sub-domain is now represented by a `ChSystemMPI` object and an associated MPI process. For example, assume a simulation is discretized into a set S of m sub-domains. In this case, let $S = \{A, B, C, D\}$ and $m = 4$, and map an MPI rank to each sub-domain so that A is mapped to MPI rank 0, $B \rightarrow 1$, $C \rightarrow 2$, and $D \rightarrow 3$. Each sub-domain maintains at all times $m + 1$ lists of objects. The first list contains all objects which are even partially contained in the associated sub-domain. These are the objects which must be considered



A	B	C	D
A_O: 1,2,3	B_O: 3,4,7	C_O: 2,3,5,6	D_O: 3,6,7
A_A:	B_A: 3	C_A: 2,3	D_A: 3
A_B: 3	B_B:	C_B: 3	D_B: 3,7
A_C: 2,3	B_C: 3	C_C:	D_C: 3,6
A_D: 3	B_D: 3,7	C_D: 3,6	D_D:

Figure 7. (Top) Sample 2-D simulation domain with four sub-domains and seven objects. (Bottom) Corresponding object lists for each sub-domain.

when computing contact forces, for example. The next m lists contain bodies which are shared with other sub-domains.

In our example, sub-domain B maintains the lists B_O, B_A, B_B, B_C, and B_D. B_O is the list of all objects that intersect (touch) sub-domain B, while B_A is the list of objects which are in sub-domain A and B. Note that sub-domain A has a list A_B which should contain the same objects as B_A. Further, list B_B is not used but is created for the sake of generality. All lists are maintained in order sorted by object ID number (see Fig. 7).

The sub-domains are now ready for time-stepping. Each sub-domain X performs collision detection among all objects in list X_O and computes the associated collision forces based on the DEM force model. Then, sub-domain X computes the net force on each object in list X_O, taking into account the contact forces, gravitational forces, and applied forces.

Next, mid-step communication occurs. Sub-domain X should send to each sub-domain Y the net force on each body in list X_Y. Similarly, X should receive from each Y the net force on each body in list X_Y. Finally, X should compute the total force on each body in list X_O. Note that X may receive force contributions for a given body from any or all of the other sub-domains in the system.

At this point, each sub-domain X has the true net force on each body in its list X_O. Each sub-domain can advance the state of its bodies in time by one time step by computing the new accelerations, velocities, and positions of all objects in the sub-domain given their mass/inertia properties and the set of applied forces. We perform an end-of-step communication to synchronize object states among sub-domains. All sub-domains which share a given body should compute its new state identically, but due to the potential for round-off error we synchronize the state from the master sub-domain (where the center-of-mass is located) to all others. The final stage is to process the $m + 1$ lists in each sub-domain, as objects may enter or leave a given sub-domain or be shared between a different set of sub-domains, necessitating updates of the contents of the lists.

2.3.3 Example simulation

In this example we simulate a Mars Rover type wheeled vehicle operating on granular terrain. The vehicle is composed of a chassis and six wheels connected via revolute joints. The wheels are driven with a constant angular velocity of π rad s^{-1}. The granular terrain is composed of 2 016 000 spherical particles. The simulation is divided into 64 sub-domains and uses a time step of 10^{-5} s. This small time step is necessary due to the use of the DEM approach to compute contact forces – a stiff force model is used to achieve small normal interpenetration, requiring a small step size to maintain stability. A snapshot from the simulation can be seen in Fig. 8. In the figure, note that the wheels of the rover are checkered blue and white. This signifies that the master copy of the rover assembly is in the blue sub-domain and the rover spans into adjacent sub-domains. In Fig. 8, the rover has settled into the granular terrain and is starting to move forward. The rear wheels displace more granular material than the front wheels because the center of mass of the rover is closer to the rear of the vehicle.

2.4 Validation and demonstration of technology

This section describes a validation effort in which experimental results were compared to simulation results obtained from Chrono::Engine. To this end, a test rig was designed and fabricated to measure the rate at which granular material flowed out of a slit due to gravity. Chrono::Engine was used to set up a corresponding simulation to match the experimental results. For more detail, see Melanz et al. (2010).

2.4.1 Experimental model

The experimental set-up consisted of a fixed base, a movable wall (angled at 45°), a translational stage, a linear actuator, and a scale (see schematic in Fig. 9). The linear actuator was capable of quickly opening a precise gap, out of which the granular material would flow due to gravity. The

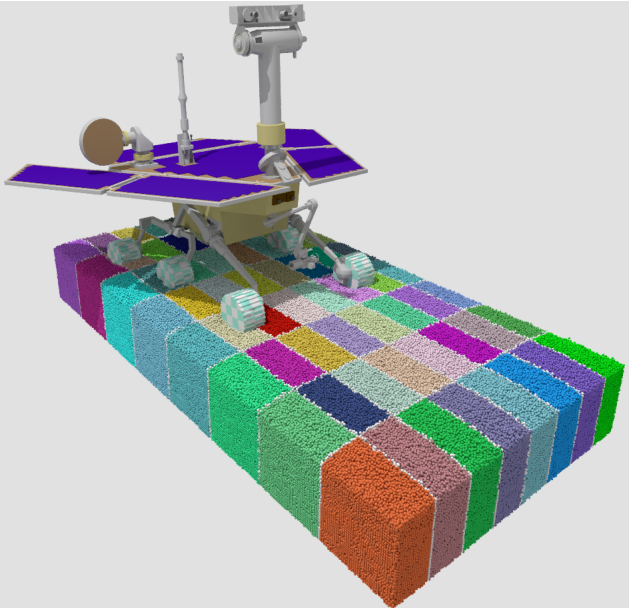


Figure 8. Snapshot of Mars Rover simulation with 2 016 000 terrain particles using 64 sub-domains. Bodies are colored by sub-domain, with shared bodies (those which span sub-domain boundaries) colored white.

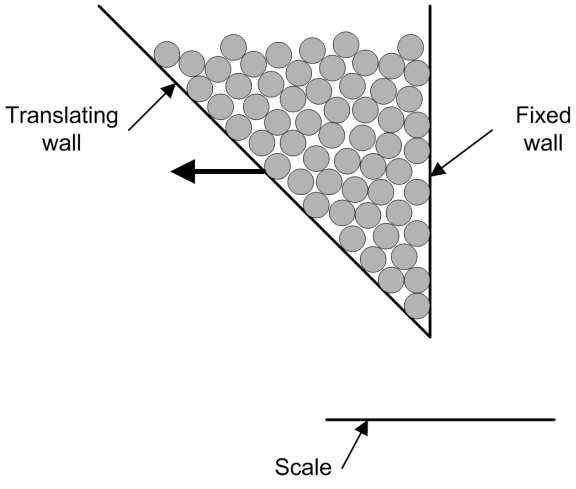


Figure 9. Schematic of validation experiment. A linear actuator and translational stage moved the left angled side a fixed amount, opening a precise gap from which the particles flowed. The mass flow rate was measured by the scale. Schematic not to scale.

scale recorded the mass of collected granular material as a function of time. The granular material consisted of approximately 40 000 uniform glass disruptor beads with diameter of 500 microns. Experiments were performed for gap sizes of 1.5 mm, 2 mm, 2.5 mm, and 3 mm. At least 5 experiments were performed for each gap size.

2.4.2 Simulation model

Chrono::Engine was used to build a model representing the experimental set up described above. In the model, the trough was represented by four rectangular boxes of finite dimensions. The motion of the box representing the angled side was captured from the data sheet of the translational stage. The granular material was modeled as perfect, identical spheres with the same mass and coefficient of friction.

The load cell measured the outflow through the gap. In the simulation, the scale was modeled by counting the number of spheres below a certain height. The number of spheres multiplied by the mass and gravity yielded the weight which was compared with experimental results. A plane was used to contain the spheres after they had been counted.

In order to save computational time, the simulation was split into two parts: one representing the process of filling the trough and the other the opening and measuring process. In this way, the trough was filled with randomly positioned spheres which were allowed to settle. Once the kinetic energy of the system was below 0.001 Joules and had reached a relatively constant value, the x-, y- , and z-position of each sphere was saved to a file.

The same initial conditions from the settling simulation were used to perform all of the necessary simulations. At the beginning of each simulation the position data set of the spheres was loaded into the model and the spheres were created at the same positions they appeared in the filling process. The motion was applied to the translating side to achieve the desired gap size, and the material began to flow.

The simulations setup consisted of 39 000 rigid body spheres with a radius of 2.5×10^{-4} m and a mass of 1.631×10^{-7} kg. The following parameters were set for this simulation. A time step of 10^{-4} s with 500 CCP iterations, and a tolerance of 10^{-7} for the maximum velocity correction. Simulations were generally run for 8 s. SI units were used for all parameters.

2.4.3 Procedure used to select the friction coefficient

The friction coefficient of a certain material is not a constant value. It can depend on various environmental influences such as humidity, surface quality, temperature etc. The friction coefficient of glass was an unknown in the validation process and needed to be determined before further observations could be done. To achieve this, one experiment at a gap size of 1.5 mm was performed and multiple simulations with the same setup and different friction coefficients were performed. The simulation results were compared to the experimental test results to determine which friction coefficient resulted in the best match, see Fig. 10. It was determined that $\mu = 0.15$ most closely matched the experimental results. This value was used for all subsequent simulations.

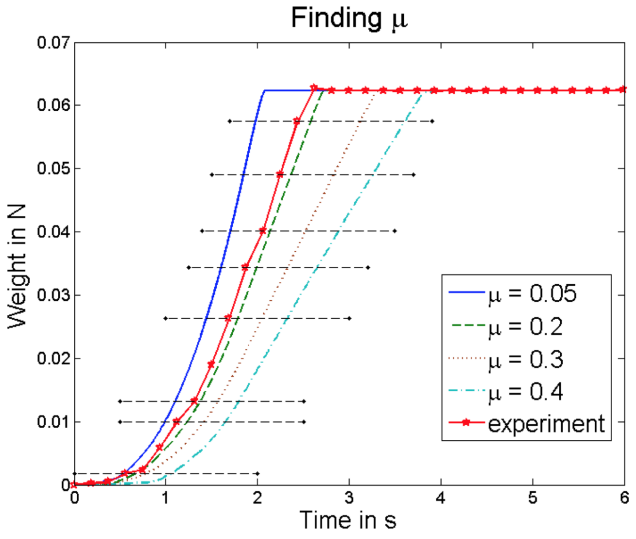


Figure 10. Selection of μ.

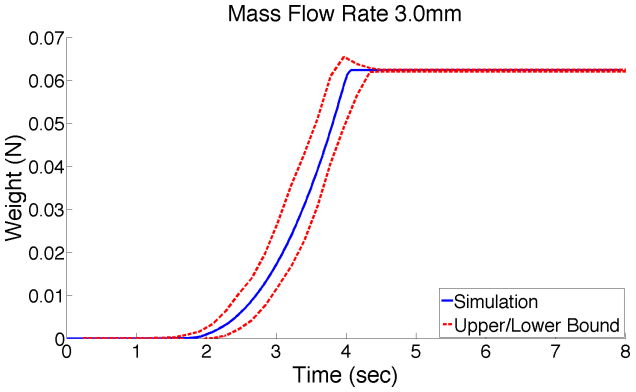


Figure 11. Weight vs. time for a gap size of 3 mm.

2.4.4 Results

The weight of the collected granular material is plotted versus time for various gap sizes in Fig. 11 through Fig. 14 using the friction coefficient determined in Fig. 10. For each experiment, the result from the simulation in Chrono::Engine, shown by the solid line, is overlaid on top of the standard deviation of the experimental runs, shown by the dashed line. Note that the simulated result lies within a single standard deviation of the experimental data.

3 Chrono::Flex

The Chrono::Flex software is a general-purpose simulator for three dimensional flexible multi-body problems and provides a suite of flexible body support. The features included in this module are multiple element types, the ability to connect these elements with a variety of bilateral constraints, multiple solvers, and contact with friction. Additionally, Chrono::Flex leverages the GPU to accelerate solution of large problems.

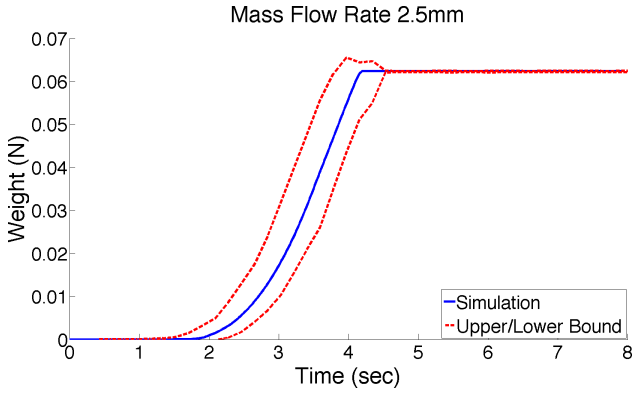


Figure 12. Weight vs. time for a gap size of 2.5 mm.

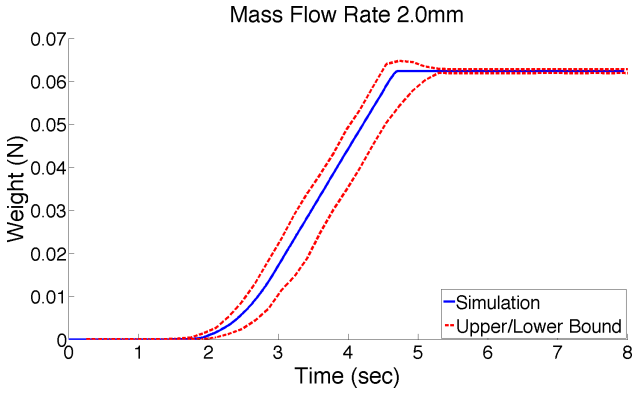


Figure 13. Weight vs. time for a gap size of 2 mm.

3.1 Element types

Chrono::Flex includes two element types implemented using the Absolute Nodal Coordinate Formulation (ANCF) (Berzeri et al., 2001; von Dombrowski, 2002). The gradient-deficient beam element and the gradient-deficient plate element are described below.

3.1.1 Gradient-deficient beam elements

This implementation uses gradient deficient ANCF beam elements to model slender beams, examples of which are shown in Fig. 15. These are two node elements with one position vector and only one gradient vector used as nodal coordinates. Each node thus has 6 coordinates: three components of the global position vector of the node and three components of the position vector gradient at the node. This formulation displays no shear locking problems for thin and stiff beams and is computationally more efficient compared to the original ANCF due to the reduced number of nodal coordinates (Gerstmayr and Shabana, 2006). The gradient deficient

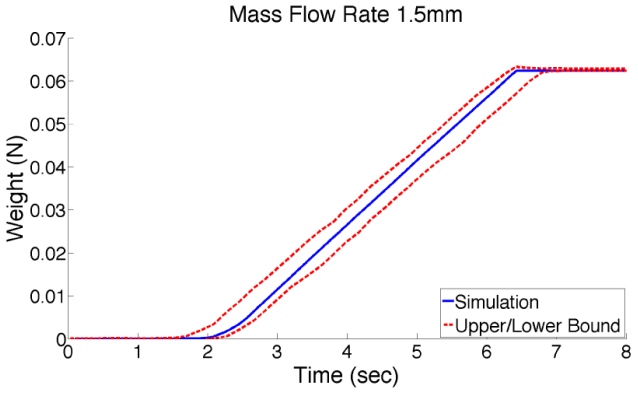


Figure 14. Weight vs. time for a gap size of 1.5 mm. This was the test case that was used for calibration.

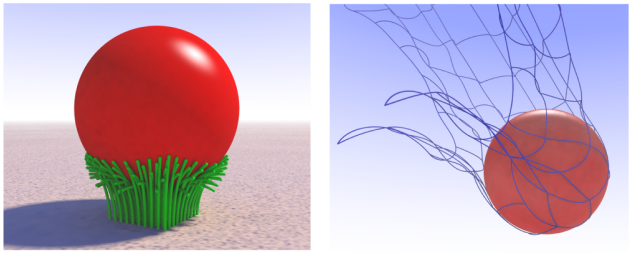


Figure 15. Two models with friction and contact using Chrono::Flex beam elements: a ball sitting on grass-like beams and a ball hitting a net.

ANCF beam element does not describe a rotation of the beam about its own axis so the torsional effects cannot be modeled.

3.1.2 Gradient-deficient plate elements

Much like beams, numerical difficulties are encountered in the fully parameterized plate element when the system has very thin and stiff components (Dufva and Shabana, 2005). The high frequencies that are induced along the thin direction of the element require an extremely small time step, resulting in longer simulation times. In the case where the aspect ratio (length divided by thickness) of the element is high, plane stress assumptions can be made that allow a reduced-order element to be accurate. Specifically, Kirchhoff's plate theory, which does not account for shear deformation, is used and results in an element with 36 degrees of freedom, or nodal coordinates, are shown in Fig. 16.

3.2 Kinematic constraints

Several types of mechanical joints are modeled in Chrono::Flex. A spherical joint (Shabana, 2005) between two nodes of any two bodies will require the position vector of each node to be identical. A revolute joint will have two additional constraints to the spherical joint constraints. In this case, the gradient vectors of the two nodes will remain

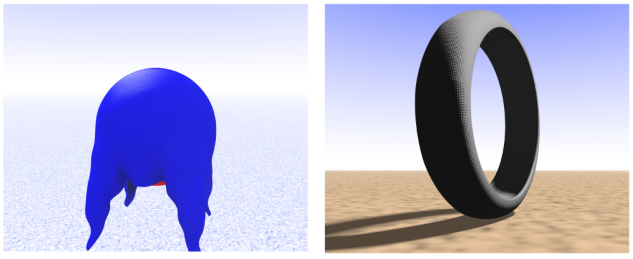


Figure 16. Two models with friction and contact using Chrono::Flex plate elements: a cloth hanging on a sphere and a closed contour shaped like a tire.

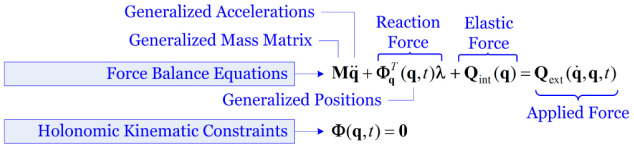


Figure 17. The equations of motion for Chrono::Flex.

in a plane perpendicular to the axis of revolute joint. There are also additional constraints due to the element connectivity in each beam. The element connectivity can be modeled as a fixed joint between the nodes. Here the common node between two elements is treated as two different nodes attached to each other through the fixed joint. This fixed joint requires all the nodal coordinates of the two nodes be identical. The generalized coordinates of the system change in time under the effect of applied forces such that these constraint equations are satisfied at all times. The time evolution of the system is governed by the Lagrange multiplier form of the constrained equations of motion.

3.3 Solvers

The equations shown in Fig. 17 form a system of index-3 Differential Algebraic Equations (DAEs). Although several low order numerical integration schemes have been effectively used to solve index-3 DAEs, Chrono::Flex utilizes the Newmark integration scheme (Hussein et al., 2008). Originally used in the structural dynamics community for the numerical integration of a linear set of second order ODEs, it was adapted for the discretization of DAEs. This implicit solver was proved to have convergence of order 1 or 2, depending on the choice of parameters γ and β.

At each time step, the numerical solution commences by solving the nonlinear set of equations shown in Fig. 18. The numerical solution of the nonlinear algebraic system falls back on a Newton-type iterative algorithm that requires the computation of its sensitivity matrix. Advancing the numerical solution in time draws on three loops: the outer-most loop marches forward in time, while at each time step the second loop solves the algebraic discretization problem in Fig. 18. Each iteration in this second loop launches a third

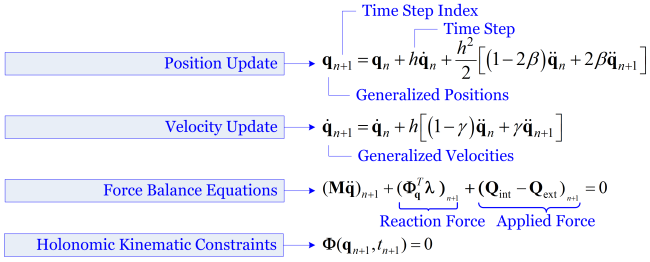


Figure 18. The discretized equations of motion for Chrono::Flex (fully implicit).

loop whose role is that of producing a vector of corrections for the acceleration and Lagrange multipliers. The corrections are computed using the BiCGStab iterative solver (Yang and Brent, 2002), which also provides for a matrix-free solution. A serial solver was implemented using the Armadillo Matrix Algebra Library (Sanderson, 2010) and a GPU parallel solver was implemented using CUSP (Bell and Garland, 2012), a linear algebra library built on top of CUDA. Chrono::Flex was validated in Khude et al. (2011) as well as in Melanz (2012) against the commercial code ADAMS (MSC.Software, 2012), and the nonlinear finite element analysis code ABAQUS (ABAQUS, 2004).

4 Chrono::Fluid

The Chrono::Fluid component aims at leveraging GPU computing to efficiently simulate fluid dynamics and fluid-solid interaction problems. Fluid-Solid Interaction (FSI) covers a wide range of applications, from blood and polymer flow to tanker trucks and ships. Simulation of the FSI problem requires two components: Fluid and Solid simulations. Simulation of the Solid phase, either rigid or flexible, in an HPC fashion, is described in previous sections. To leverage the existing solid phase simulation, the fluid flow simulation should satisfy some conditions, introduced by the aforementioned target problems. First, the fluid flow may experience large domain deformation due to the motion of the solid phase. Second, the two phases should be coupled via an accurate algorithm. Third, target problems may experience free surface as well as internal flow. Finally, the whole simulation should be capable of an HPC implementation to maintain the scalability of the code.

Fluid flow can be simulated in either an Eulerian or a Lagrangian framework. Provided that the interfacial forces are captured thoroughly, the Lagrangian framework is capable of tracking the domain deformation introduced by the motion of the solid phase at almost no extra cost. Smoothed Particle Hydrodynamics (SPH) (Lucy, 1977; Gingold and Monaghan, 1977; Monaghan, 2005), its modifications (Monaghan, 1989; Dilts, 1999), and variations (Koshizuka et al., 1998) have been widely used for the simulation of the fluid domain in a Lagrangian framework. The main evolution equations of the fluid flow using SPH are expressed as

$$\frac{\mathrm{d}\rho_a}{\mathrm{d}t} = \rho_a \sum_b \frac{m_b}{\rho_b}(v_a - v_b).\nabla_a W_{ab} \tag{1}$$

$$\frac{\mathrm{d}v_a}{\mathrm{d}t} = -\sum_b m_b\left(\left(\frac{p_b}{{\rho_a}^2} + \frac{p_a}{{\rho_b}^2}\right)\nabla_a W_{ab} - \frac{(\mu_a+\mu_b)r_{ab}.\nabla_a W_{ab}}{\bar{\rho}_{ab}{}^2(r_{ab}^2+\varepsilon\bar{h}_{ab}^2)}v_{ab}\right) + f_a \tag{2}$$

which are solved in conjunction with

$$\mathrm{d}x/\mathrm{d}t = v \tag{3}$$

to update the fluid properties. In Eqs. (1) to (3), ρ, v, and p are local fluid density, velocity, and pressure, respectively, m is the representative fluid mass assigned to the SPH marker, W is a kernel function which smooths out the local fluid properties within a resolution length $l = \kappa h$, and r_{ab} is the distance between two fluid markers denoted by a and b. Fluid flow evolution equations, defined by Eqs. (1) to (3), are solved explicitly, where pressure is related to density via an appropriate state equation to maintain the compressibility below 1 %. To increase the accuracy and stability of the simulation, an XSPH modification (Monaghan, 1989) and Shephard filtering (Dalrymple and Rogers, 2006) were applied.

4.1 FSI with Smoothed Particle Hydrodynamics: a quick overview

A proper choice of fluid-solid coupling should satisfy the no-slip and impenetrability conditions on the surface of the solid obstacles. By attaching Boundary Condition Enforcing (BCE) markers on the surface of the solid objects, the local relative velocity, i.e., at the markers location, of the two phases will be zero (see Fig. 19). The position and velocity of the BCE markers are updated according the motion of the solid phase, which results in the propagation of the solid motion to the fluid domain. On the other hand, the interaction forces on the BCE markers are used to calculate the total force and torque exerted by the fluid on the solid object.

4.1.1 FSI with Smoothed Particle Hydrodynamics: proximity computation

The overall simulation of the FSI framework is performed in parallel, where each thread handles the force calculation of a fluid or BCE marker first, and a rigid body later. Next, the parallel threads perform the kinematics update of the fluid markers, rigid bodies, and BCE markers, respectively. An essential part of the force calculation stage is the proximity computation, which will be explained briefly herein.

Proximity computation used in our work leverages the algorithm provided in CUDA SDK (NVIDIA Corporation, 2012), where the computation domain is divided into bins

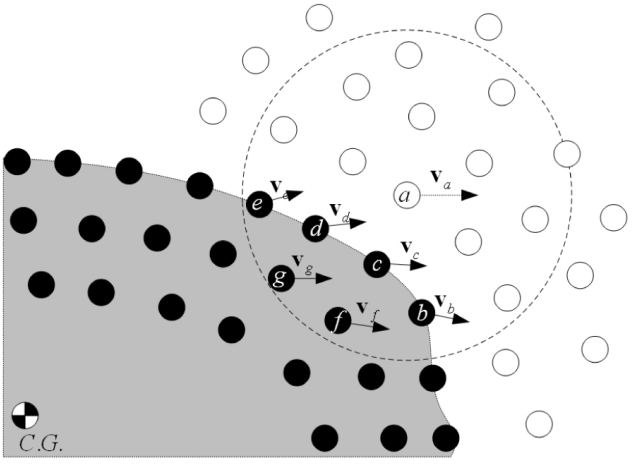


Figure 19. Coupling of the fluid and solid phases. BCE and fluid markers are represented by black and white circles, respectively.

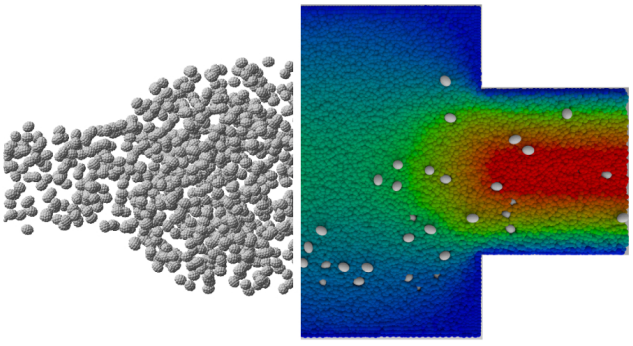


Figure 20. Simulation of rigid bodies inside a fluid flow: rigid ellipsoids with their BCE markers are shown in the left image while the fluid's velocity contours and rigid ellipsoids at the mid-section of the channel are shown in the right image.

whose sizes are the same as the resolution length of the SPH kernel function. A hash value is assigned to each marker based on its location with respect to the bins. Markers are sorted based on their hash value. The sorted properties are stored in independent arrays to improve the memory access and cache coherency. To compute the forces on a marker, the lists of the possible interacting markers inside its bin and all 26 neighbor bins are called. The hash values of the bins are used to access the relevant segments of the sorted data.

4.2 Validation and demonstration of technology

The aforementioned FSI simulation engine was used to validate the lateral migration of cylindrical particles in plane Poiseuille flow, spherical particles in pipe flow, and particle distribution in Poiseuille flow of suspension (Pazouki and Negrut, 2012,?). Due to the scalability of Chrono::Fluid in both fluid and solid phases, increasing the number of rigid bodies, which translates into decreasing the number of fluid

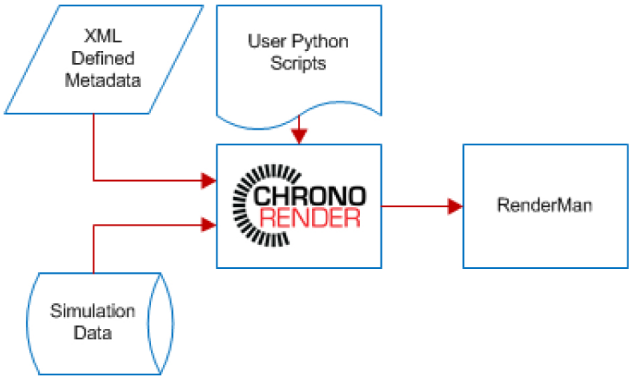


Figure 21. Chrono::Render architecture.

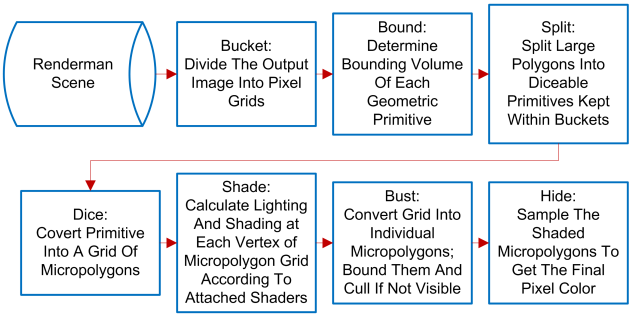


Figure 22. An overview of the REYES Pipeline.

particles, does not affect the simulation time. Therefore, the simulation of a highly dense suspension is possible. Figure 20 shows the result of the simulation of the flow of suspension including 1500 particles through a channel. A similar scenario with 13 000 particles in suspension was simulated in Chrono::Fluid.

5 Chrono::Render

Chrono::Render is a software package that enables simple, streamlined, and fast visualization of arbitrary data using Pixar's RenderMan (Pixar, 1988, 1989, 2000, 2005). Specifically, Chrono::Render contains a hybrid of processing binaries and Python scripting modules that seek to abstract away the complexities of rendering with RenderMan. Additionally, Chrono::Render is targeted for providing rendering as an automated post-processing step in a remote simulation pipeline, hence it is controlled via a succinct XML specification for "gluing" together rendering with arbitrary processes. As seen in Fig. 21, Chrono::Render combines simulation data, XML describing how to use the data, and optional user-defined Python scripts into a complex, visually-rich scene to be rendered by RenderMan.

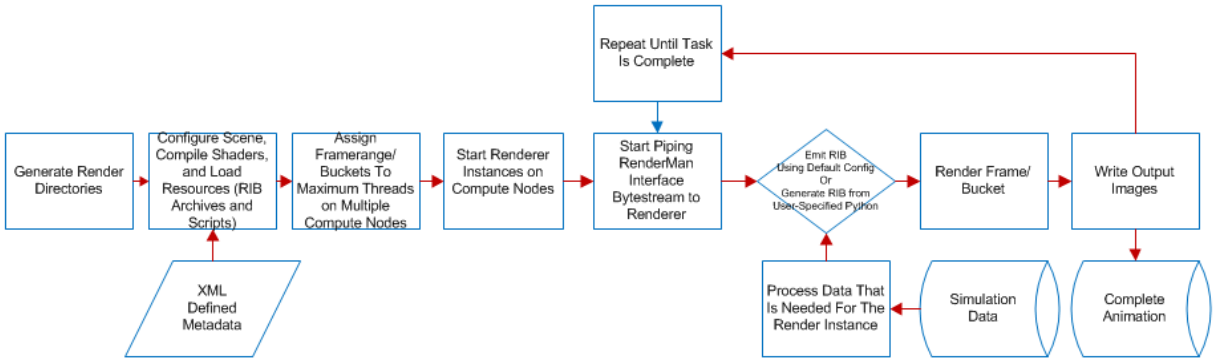


Figure 23. Chrono::Render execution workflow.

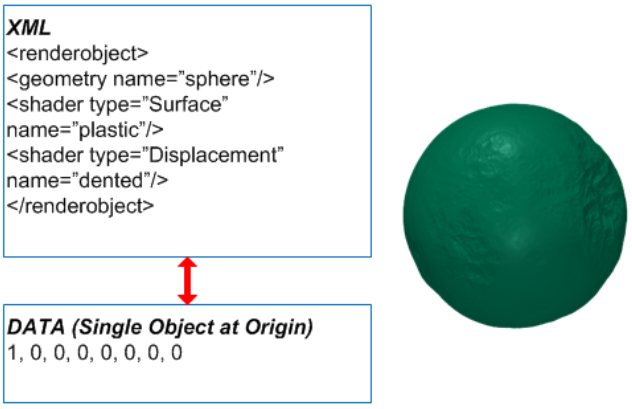


Figure 24. Simple XML for a sphere with a Surface and Displacement shader.

5.1 On the choice of RenderMan

Using RenderMan for rendering is motivated by the scope of arbitrary data sets and the potentially immense scene complexity that results from big data; REYES, the underlying architecture for RenderMan is ideally suited for this task. REYES works by dividing each surface in the scene into a grid of micropolygons and shades at the grid vertices (Cook et al., 1987) (see Fig. 22).

This results in tractable rendering for complex scenes because: (a) only a small portion of the scene needs to be in memory at any given time; (b) grid-based computation leads to optimal memory access patterns; (c) non-visible objects need not be loaded into memory; (d) fully-rendered objects can be removed from memory; and (e) objects are tessellated according to size on the screen; less complex geometry is dynamically loaded whenever possible.

REYES is perfectly suited for parallel processing since it scales linearly with the number of cores. Considering that REYES needs only a handful of relevant scene elements at a time, this data can be parsed into low-memory buckets and distributed amongst cores for parallel rendering; thus REYES' low memory-footprint and efficient concurrent resource usage for the complex scenes makes it a great renderer for a distributed-computing platform.

5.2 Accessibility of high-quality graphics

Although REYES can manage the issue of scene complexity, leveraging this power is difficult without computer graphics expertise. The guiding principle of Chrono::Render is to make high-quality rendering available to researchers, most of whom do not have the background or bandwidth to spend time learning how to use complex graphics applications or make sense of REYES' intricacies. Consequently, Chrono::Render encapsulates into the XML specification the complicated steps needed to make interesting visual effects, such as multipass rendering. The user must only instance the correct XML components to achieve high-quality renders. The program flow of Chrono::Render is shown in Fig. 23.

The XML specification allows for the concise expression of salient features and scene objects. For example, the snippet in Fig. 24 illustrates the XML file that translates a single line from a comma-separated value (CSV) data file into a RenderMan sphere using two shaders.

Although simple, the render is visually rich. This description is often enough to visualize most generic data, but it cannot handle all arbitrary visualizations, so in order to maintain generality we make use of Python scripts and wrappers to enable simplified procedural RenderMan Interface Bytestream generation. Any XML element can be scripted such that at runtime, the script output will be piped into the same rendering context. This makes it possible to perform processing for specialized data as well as modularize the rendering of specific effects. Obviously this adds more complexity for defining the scene, but Chrono::Render provides Python modules with methods and classes intended to ease this programming as much as possible. Additionally, most of the Chrono::Render Python modules wrap C++ functions and classes with the purpose of exploiting speed while still making use of the syntactical/type-free simplicity of Python. Figure 25 gives an example of combining XML with Python scripts to achieve a more complicated render.

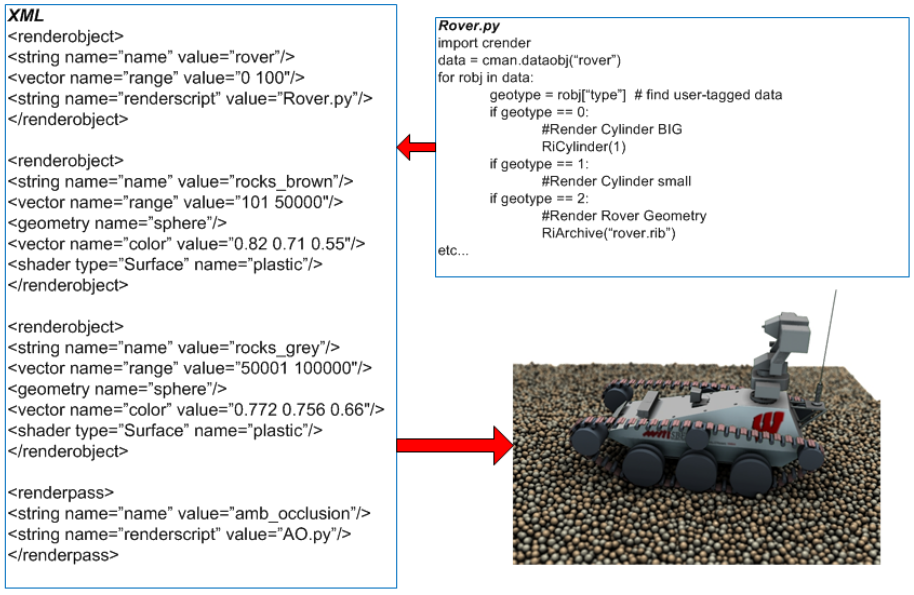


Figure 25. General purpose rendering with Chrono::Render. The Rover body contains multiple shape descriptions of which are generated from a Python script. Data is tagged with a name which can be later be accessed using some of Chrono::Render's Python functionality.

5.3 Other capabilities

Beyond interpreting parameters and data into RenderMan calls, Chrono::Render provides tools for bootstrapping rendering projects. Chrono::Render can: (a) construct directory structures for localizing and managing scene resources; (b) automate distribution of rendering across a multi-node network; (c) convert common graphics file formats into RenderMan file formats such as Wavefront Objs and Mtls to RenderMan RIBs and Shaders; (d) generate XML for automatically adding parameters to the scene description for describing advanced visual effects such as subsurface scattering, ambient occlusion, reflections, etc.; (e) mesh point-clouds, particularly useful for particle-based fluid simulations; and (f) dump the generated RenderMan calls to disk for reuse.

Chrono:Render is currently available for free download as a pre-built binary for Linux. Members of the Wisconsin Applied Computing Center can use this capability remotely as a service by leveraging 320 AMD cores on which Chrono::Render is currently deployed.

6 Conclusions and future work

The Chrono simulation package is composed of a collection of components designed to perform multi-physics simulations leveraging emerging high-performance computing hardware. Chrono::Engine provides support for rigid body dynamics, focusing on large granular dynamics problems, Chrono::Flex enables simulation of flexible beam and plate elements interacting through contact and bilateral constraints, while Chrono:Fluid allows the simulation of fluid flows and fluid-solid interaction problems. Finally, Chrono::Render provides high-quality visualization of arbitrary simulation data from the other Chrono components. These components have been designed to leverage high-performance computing hardware whenever possible. Chrono::Engine supports CPU parallelism through a domain-decomposition approach, while Chrono::Engine, Chrono::Flex, and Chrono::Fluid all support GPU parallelism to further improve simulation performance.

While these components provide useful simulation capabilities on their own, ongoing work seeks to further integrate the various Chrono components.

Chrono availability

Major releases of the Chrono::Engine software are available from the Chrono::Engine website at http://chronoengine.info. Chrono in its entirety can be downloaded from http://sbel.wisc.edu/chrono. The latter site also displays the nightly build status for various platforms and unit testing results.

Acknowledgements. Financial support for the Wisconsin authors was provided in part by the National Science Foundation Award 0840442 and Army Research Office W911NF-12-1-0395. Financial support for A.Tasora was provided in part by the Italian Ministry of Education under the PRIN grant 2007Z7K4ZB. We thank NVIDIA and AMD for sponsoring our research programs in the area of high-performance computing.

Edited by: A. Müller
Reviewed by: two anonymous referees

References

ABAQUS: User Manual – Version 6.5, Hibbitt, Karlsson and Sorensen, Inc., Pawtucket, RI, 2004.

Anitescu, M. and Tasora, A.: An iterative approach for cone complementarity problems for nonsmooth dynamics, Comput. Optim. Appl., 47, 207–235, doi:10.1007/s10589-008-9223-4, 2010.

Bell, N. and Garland, M.: CUSP: Generic Parallel Algorithms for Sparse Matrix and Graph Computations, http://cusp-library.googlecode.com, version 0.3.0, 2012.

Berzeri, M., Campanelli, M., and Shabana, A. A.: Definition of the Elastic Forces in the Finite-Element Absolute Nodal Coordinate Formulation and the Floating Frame of Reference Formulation, Multibody Syst. Dyn., 5, 21–54, 2001.

Cook, R. L., Carpenter, L., and Catmull, E.: The Reyes Image Rendering Architecture, SIGGRAPH 1987 Proceedings, 95–102, 1987.

Cundall, P.: A computer model for simulating progressive large-scale movements in block rock mechanics, in: Proceedings of the International Symposium on Rock Mechanics, Nancy, France, 1971.

Cundall, P. and Strack, O.: A discrete element model for granular assemblies, Geotechnique, 29, 47–65, 1979.

Dalrymple, R. and Rogers, B.: Numerical modeling of water waves with the SPH method, Coast. Eng., 53, 141–147, 2006.

Dilts, G.: Moving-least-squares-particle hydrodynamics I. Consistency and stability, Int. J. Numer. Meth. Eng., 44, 1115–1155, 1999.

Dufva, K. and Shabana, A.: Analysis of thin plate structures using the absolute nodal coordinate formulation, P. I. Mech. Eng. K-J. Mul., 219, 345–355, 2005.

Gerstmayr, J. and Shabana, A.: Analysis of thin beams and cables using the absolute nodal co-ordinate formulation, Nonlinear Dynam., 45, 109–130, 2006.

Gingold, R. and Monaghan, J.: Smoothed particle hydrodynamics-theory and application to non-spherical stars, Mon. Not. R. Astron. Soc., 181, 375–389, 1977.

Gropp, W., Lusk, E., and Skjellum, A.: Using MPI: Portable Parallel Programming with the Message-Passing Interface, 2nd Edn., MIT Press, 1999.

Heyn, T.: Simulation of Tracked Vehicles on Granular Terrain Leveraging GPUComputing, M.S. thesis, Department of Mechanical Engineering, University of Wisconsin-Madison, http://sbel.wisc.edu/documents/TobyHeynThesis_final.pdf, 2009.

Hussein, B., Negrut, D., and Shabana, A.: Implicit and explicit integration in the solution of the absolute nodal coordinate differential/algebraic equations, Nonlinear Dynam., 54, 283–296, 2008.

Khude, N., Melanz, D., Stanciulescu, I., and Negrut, D.: A Parallel GPU Implementation of the Absolute Nodal Coordinate Formulation With a Frictional/Contact Model for the Simulation of Large Flexible Body Systems, ASME Conference on Multibody Systemss and Nonlinear Dynamics, 2011.

Koshizuka, S., Nobe, A., and Oka, Y.: Numerical analysis of breaking waves using the moving particle semi-implicit method, Int. J. Numer. Meth. Fl., 26, 751–769, 1998.

Kruggel-Emden, H., Simsek, E., Rickelt, S., Wirtz, S., and Scherer, V.: Review and extension of normal force models for the discrete element method, Powder Technol., 171, 157–173, 2007.

Lucy, L.: A numerical approach to the testing of the fission hypothesis, Astron. J., 82, 1013–1024, 1977.

Mazhar, H., Heyn, T., and Negrut, D.: A scalable parallel method for large collision detection problems, Multibody Syst. Dyn., 26, 37–55, doi:10.1007/s11044-011-9246-y, 2011.

Melanz, D.: On the Validation and Applications of a Parallel Flexible Multi-body Dynamics Implementation, M.S. thesis, University of Wisconsin-Madison, 2012.

Melanz, D., Tupy, M., Smith, B., Turner, K., and Negrut, D.: On the Validation of a Differential Variational Inequality Approach for the Dynamics of Granular Material-DETC2010-28804, in: Proceedings to the 30th Computers and Information in Engineering Conference, edited by: Fukuda, S. and Michopoulos, J. G., ASME International Design Engineering Technical Conferences (IDETC) and Computers and Information in Engineering Conference (CIE), 2010.

Mindlin, R. and Deresiewicz, H.: Elastic spheres in contact under varying oblique forces, J. Appl. Mech., 20, 327–344, 1953.

Monaghan, J.: On the problem of penetration in particle methods, J. Comput. Phys., 82, 1–15, 1989.

Monaghan, J.: Smoothed particle hydrodynamics, Rep. Prog. Phys., 68, 1703–1759, 2005.

MSC.Software: ADAMS: Automatic Dynamic Analysis of Mechanical Systems, Ann Arbor, Michigan, 2012.

Negrut, D., Tasora, A., Mazhar, H., Heyn, T., and Hahn, P.: Leveraging parallel computing in multibody dynamics, Multibody Syst. Dyn., 27, 95–117, doi:10.1007/s11044-011-9262-y, 2012.

NVIDIA Corporation: NVIDIA CUDA Developer Zone, available at: https://developer.nvidia.com/cuda-downloads, 2012.

Pazouki, A. and Negrut, D.: Direct simulation of lateral migration of bouyant particles in channel flow using GPU computing, in: Computers and Information in Engineering, CIE32, ASME, Chicago, IL, USA, 2012a.

Pazouki, A. and Negrut, D.: A numerical study of the effect of rigid body rotation, size, skewness, mutual distance, and collision on the radial distribution of suspensions in pipe flow, in review, 2013.

Pazouki, A., Mazhar, H., and Negrut, D.: Parallel Ellipsoid Collision Detection with Application in Contact Dynamics-DETC2010-29073, in: Proceedings to the 30th Computers and Information in Engineering Conference, edited by: Fukuda, S. and Michopoulos, J. G., ASME International Design Engineering Technical Conferences (IDETC) and Computers and Information in Engineering Conference (CIE), 2010.

Pazouki, A., Mazhar, H., and Negrut, D.: Parallel collision detection of ellipsoids with applications in large scale multibody dynamics, Math. Comput. Simulat., 82, 879–894, doi:10.1016/j.matcom.2011.11.005, 2012.

Pixar: The RenderMan Interface, Technical specification, Pixar, 1988, 1989, 2000, 2005.

Sanderson, C.: Armadillo: An open source C++ linear algebra library for fast prototyping and computationally intensive experiments, Tech. rep., Technical report, NICTA, 2010.

Shabana, A. A.: Dynamics of Multibody Systems, Cambridge University Press, 3rd Edn., 2005.

Snethen, G.: XenoCollide Website, http://www.xenocollide.com, 2007.

Snethen, G.: XenoCollide: Complex Collision Made Simple, in: Game Programming Gems 7, edited by: Jacobs, S., Charles River

Media, 165–178, 2008.

Tasora, A. and Anitescu, M.: A convex complementarity approach for simulating large granular flows, J. Comput. Nonlin. Dyn., 5, 1–10, doi:10.1115/1.4001371, 2010.

Tasora, A. and Anitescu, M.: A matrix-free cone complementarity approach for solving large-scale, nonsmooth, rigid body dynamics, Comput. Method. Appl. M., 200, 439–453, doi:10.1016/j.cma.2010.06.030, 2011.

Tasora, A., Righettini, P., and Silvestri, M.: Architecture of the Chrono::Engine physics simulation middleware, in: Proceedings of ECCOMAS 2007 Multibody Conference, 2007.

von Dombrowski, S.: Analysis of Large Flexible Body Deformation in Multibody Systems Using Absolute Coordinates, Multibody Syst. Dyn., 8, 409–432, doi:10.1023/A:1021158911536, 2002.

Yang, L. and Brent, R.: The improved BiCGStab method for large and sparse unsymmetric linear systems on parallel distributed memory architectures, in: Algorithms and Architectures for Parallel Processing, 2002. Proceedings. Fifth International Conference on, IEEE, 324–328, 2002.