Chrono::Parallel

Overview of the Chrono::Parallel module

What is Chrono::Parallel?

A software library for OpenMP-based parallel simulation of Chrono models

- Middleware: can be embedded in third parties software
- Open source with BSD license
- Library developed in C++ (requires C++11)
- Cross-platform: compiles on GNU GCC, MSVC, etc.

Relationship to Chrono::Engine

- Chrono-Parallel relies on Chrono for all its modeling capabilities
- Supports a subset of Chrono modeling elements:
  - Rigid bodies with frictional contact (DEM-C or DEM-P)
  - Kinematic joints (revolute, spherical, translational, etc.)
  - Force elements (spring-dampers, actuators, etc.)
  - 1-D shafts and associated elements and constraints (shaft-body connection, gears, motors, etc.)
- No support for FEA
- Implements only the *Implicit Euler Linearized* time-stepper

- Chrono-Parallel uses different data structures and algorithms

Data structures and algorithms

Chrono-Parallel uses structures of arrays (SOA) as opposed to arrays of structures (AOS)

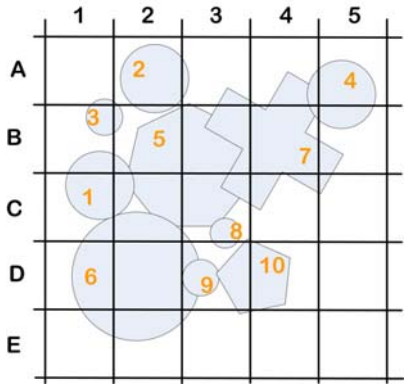
- OpenMP parallel for loops

- Thrust (with OMP backend)
  - parallel algorithms library (sort, scan, reductions, etc.)
  - Interface similar to the C++ STL
  - https://code.google.com/p/thrust/

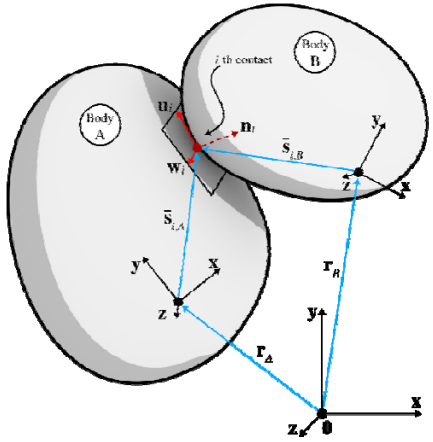
- Blaze
  - High-performance (dense and) sparse matrix operations
  - Smart Expression Template implementation
  - https://code.google.com/p/blaze-lib/

Collision detection

- Broad phase
  - Based on binning with AABBs
  - Adaptive 2-level hierarchical grid



- Narrow phase
  - Different options:
    - MPR (Minkovski Portal Refinement)
    - GJK (Gilbert-Johnson-Keerthi)
    - SAT (Separating Axis Theorem)
  - Hybrid option (SAT with fallback to MPR)



User code

- New type of ChSystem:

```
ChSystemParallelDVI system;
```

or

```
ChSystemParallelDEM system;
```

- Create elements in the mechanical system using Chrono::Engine functions

- Specify solver settings by directly accessing the structures

```
ChSystemParallel::GetSettings()->solver
```

and

```
ChSystemParallel::GetSettings()->collision
```

- Advance simulation by invoking

```
chrono::ChSystem::DoStepDynamics()
```

or

```
chrono::opengl::ChOpenGLWindow::DoStepDynamics()
```

User code – solver settings (DEM-C)

```
system.GetSettings()->perform_thread_tuning = false;

system.GetSettings()->solver.solver_mode = SLIDING;
system.GetSettings()->solver.max_iteration_normal = 0;
system.GetSettings()->solver.max_iteration_sliding = 200;
system.GetSettings()->solver.max_iteration_spinning = 0;
system.GetSettings()->solver.max_iteration_bilateral = 50;
system.GetSettings()->solver.tolerance = 0.1;
system.GetSettings()->solver.alpha = 0;
system.GetSettings()->solver.contact_recovery_speed = 10000;

system.ChangeSolverType(APGD);

system.GetSettings()->collision.narrowphase_algorithm = NARROWPHASE_HYBRID_MPR;
system.GetSettings()->collision.collision_envelope = 0.01;
system.GetSettings()->collision.bins_per_axis = I3(10, 10, 10);
```

User code – solver settings (DEM-P)

```
system.GetSettings()->perform_thread_tuning = false;

system.GetSettings()->solver.max_iteration_bilateral = 50;
system.GetSettings()->solver.tolerance = 0.1;

system.GetSettings()->solver.contact_force_model = ContactForceModel::Hertz
system.GetSettings()->solver.tangential_displ_mode = TangentialDisplacementModel::MultiStep;

system.GetSettings()->collision.narrowphase_algorithm = NARROWPHASE_HYBRID_MPR;
system.GetSettings()->collision.bins_per_axis = I3(10, 10, 10);
```

Chrono utilities for granular dynamics

Chrono utilities

- Samplers for granular dynamics
  - Uniform grid, Poisson-disk sampling, hexagonal close packing
  - Sampling of different domains (box, sphere, cylinder)
  - Allows 2D sampling (rectangle, disk)
- Generators for granular dynamics
  - Create particles, using a given sampler, randomly selecting from a mixture of ingredients with prescribed expectation
  - Mixture ingredient: multivariate normal distributions for particle size and contact material properties

- Convenience functions for creating bodies with simple geometry (contact and graphics)
- Utilities for file I/O, output for POV-Ray post-processing, etc.
- Utilities for validation using golden files, data filtering, etc.

- Classes and free functions implemented in the `chrono::utils` namespace

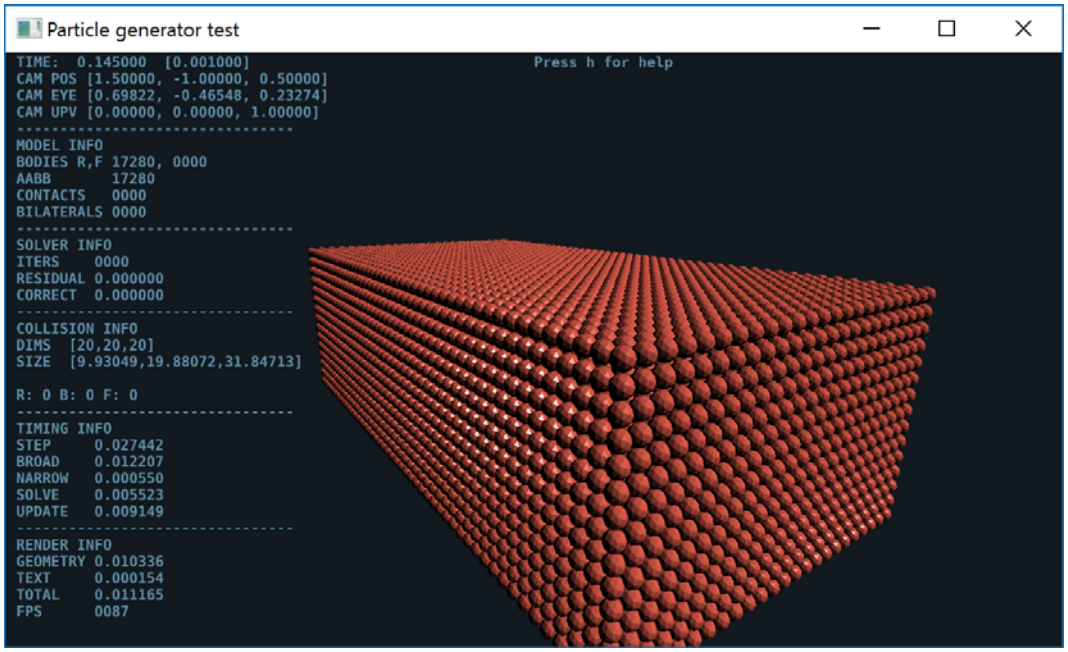
Volume sampling

- `chrono::utils::PDSampler`
  - generates points in 3D domains (box, sphere, or cylinder) using Poisson Disk Sampling. The sampler produces a set of points uniformly distributed in the specified domain such that no two points are closer than a specified distance.
  - based on "Fast Poisson Disk Sampling in Arbitrary Dimensions" by Robert Bridson http://people.cs.ubc.ca/~rbridson/docs/bridson-siggraph07-poissondisk.pdf

- `chrono::utils::GridSampler`
  - generates points in 3D or 2D domain with constant grid spacing
  - grid spacing can be different in the 3 directions

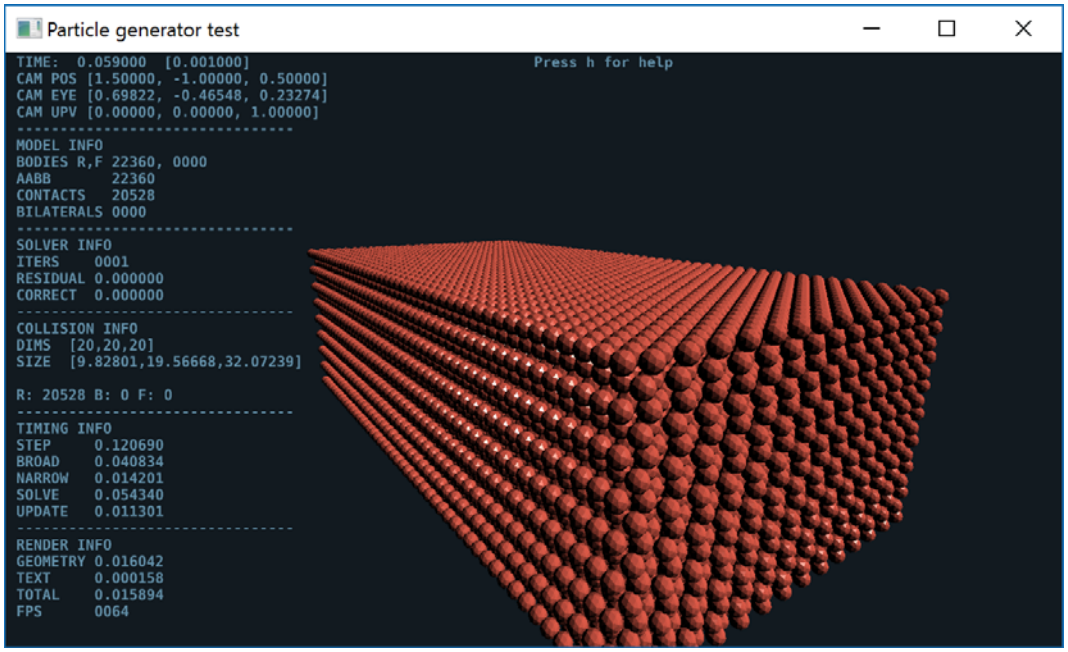
- `chrono::utils::HCPSampler`
  - generates points in a hexagonally close packed structure

Sampling – regular grid

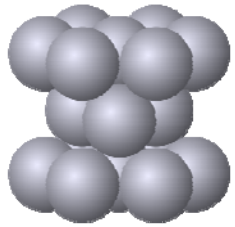



```
utils::Generator gen(system);

std::shared_ptr<utils::MixtureIngredient> m1 = gen.AddMixtureIngredient(utils::SPHERE, 1.0);
m1->setDefaultSize(radius_g);

double r = 1.05 * radius_g;
gen.createObjectsBox(utils::REGULAR_GRID, 2 * r, ChVector<>(0, 0, 0), ChVector<>(1, 0.5, 0.3));
```

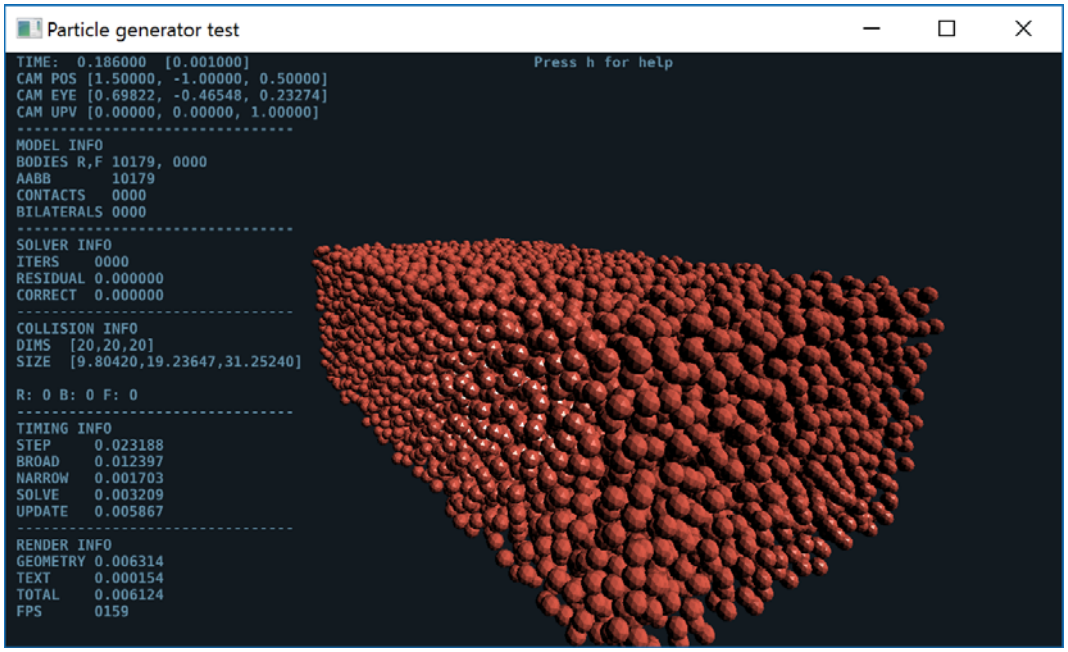
Sampling – hexagonal close packing




```
utils::Generator gen(system);

std::shared_ptr<utils::MixtureIngredient> m1 = gen.AddMixtureIngredient(utils::SPHERE, 1.0);
m1->setDefaultSize(radius_g);

double r = 1.05 * radius_g;
gen.createObjectsBox(utils::HCP_PACK, 2 * r, ChVector<>(0, 0, 0), ChVector<>(1, 0.5, 0.3));
```

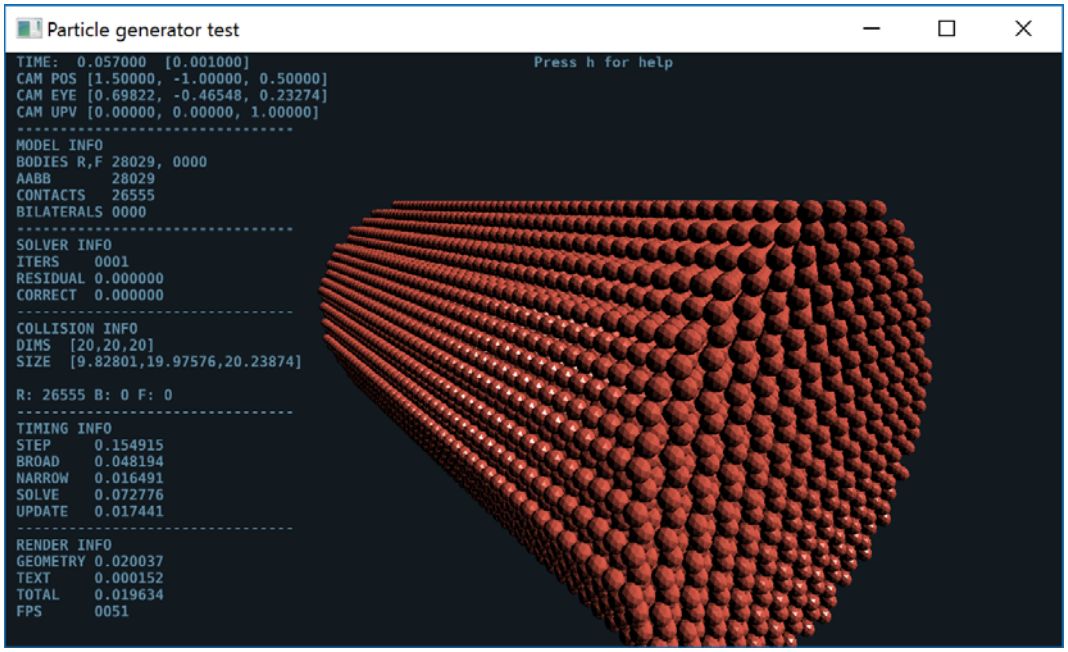


Sampling – Poisson disc




```
utils::Generator gen(system);

std::shared_ptr<utils::MixtureIngredient> m1 = gen.AddMixtureIngredient(utils::SPHERE, 1.0);
m1->setDefaultSize(radius_g);

double r = 1.05 * radius_g;
gen.createObjectsBox(utils::POISSON_DISK, 2 * r, ChVector<>(0, 0, 0), ChVector<>(1, 0.5, 0.3));
```

Sampling of cylindrical volume



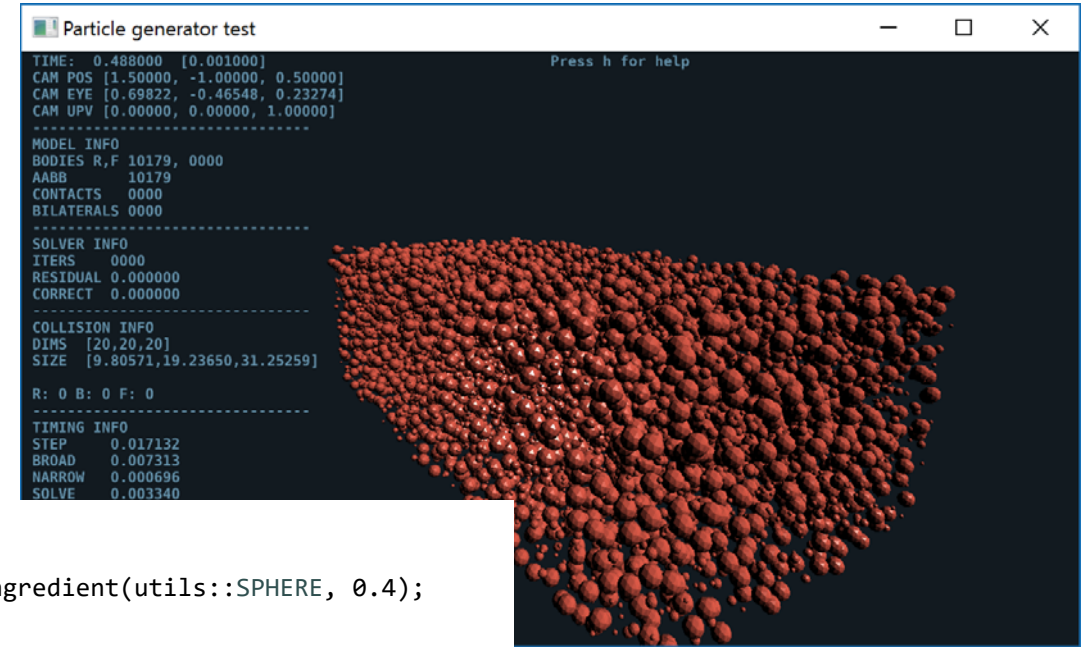

```
utils::Generator gen(system);

std::shared_ptr<utils::MixtureIngredient> m1 = gen.AddMixtureIngredient(utils::SPHERE, 1.0);
m1->setDefaultSize(radius_g);

double r = 1.05 * radius_g;
gen.createObjectsCylinderX(utils::HCP_PACK, 2 * r, ChVector<>(0, 0, 0), 0.5, 1);
```

Mixtures of ingredients

- `chrono::utils::MixtureIngredient`
  - models an ingredient of given shape (`chrono::utils::MixtureType`) in a granular material mixture
  - specifies object size/scaling (constant or drawn from user-defined distribution)
  - specifies material properties (constant or drawn from user-defined distributions)
  - defines the ratio for this particular ingredient in a mixture
  - uses truncated normal distributions

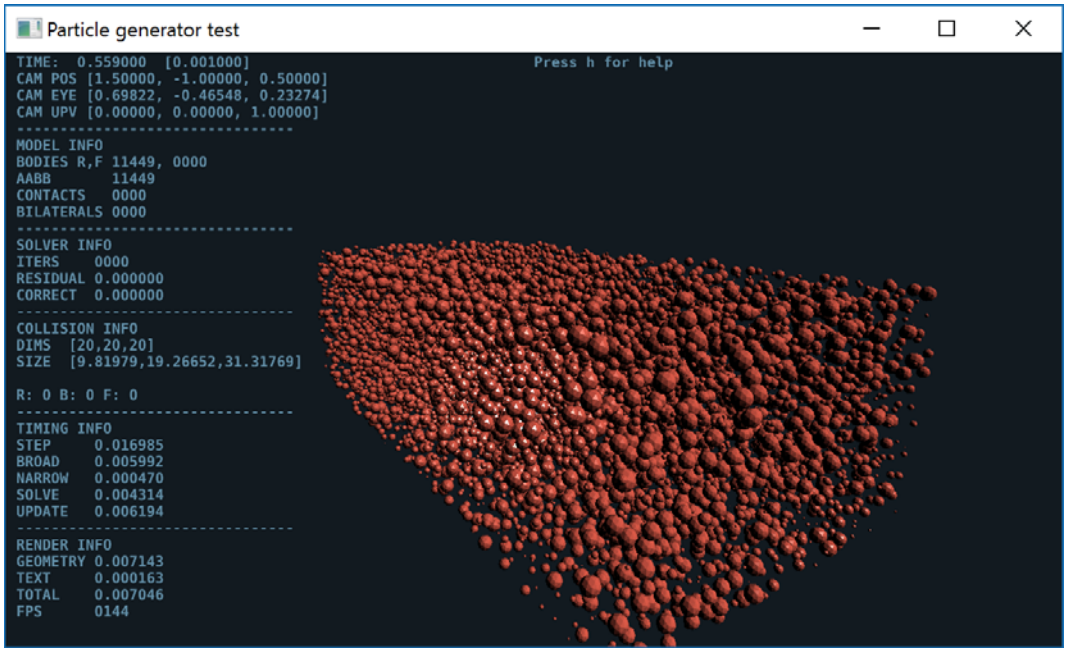
- `chrono::utils::Generator`
  - generates objects in a specified volume (box or cylinder), drawing from a uniform distribution over a mixture of ingredients
  - particle locations are generated with a user-specified sampling method (REGULAR_GRID, HCP_PACK, POISSON_DISK)
  - allows degenerate sampling volumes (i.e., rectangle or circle)

Mixing different sizes (pre-set)



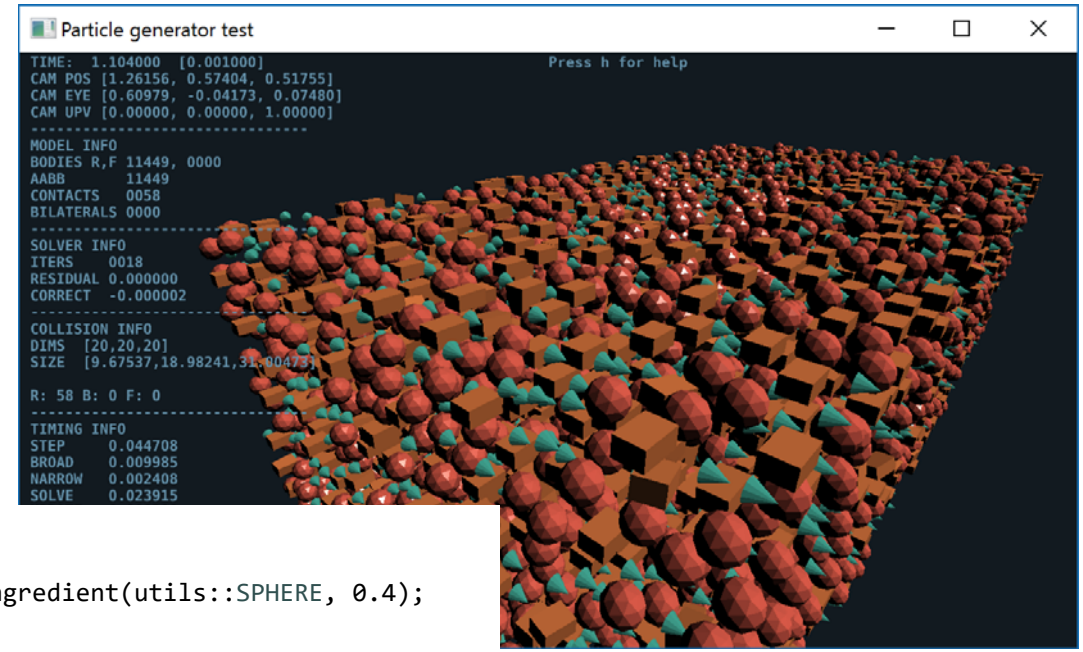

```
utils::Generator gen(system);

std::shared_ptr<utils::MixtureIngredient> m1 = gen.AddMixtureIngredient(utils::SPHERE, 0.4);
m1->setDefaultDensity(rho_g);
m1->setDefaultSize(radius_g);
std::shared_ptr<utils::MixtureIngredient> m2 = gen.AddMixtureIngredient(utils::SPHERE, 0.3);
m2->setDefaultDensity(rho_g);
m2->setDefaultSize(0.5 * radius_g);
std::shared_ptr<utils::MixtureIngredient> m3 = gen.AddMixtureIngredient(utils::SPHERE, 0.3);
m3->setDefaultDensity(rho_g);
m3->setDefaultSize(0.25 * radius_g);

double r = 1.01 * radius_g;
gen.createObjectsBox(utils::POISSON_DISK, 2 * r, ChVector<>(0, 0, 0), ChVector<>(1, 0.5, 0.3));
```

Mixing different sizes (from distribution)



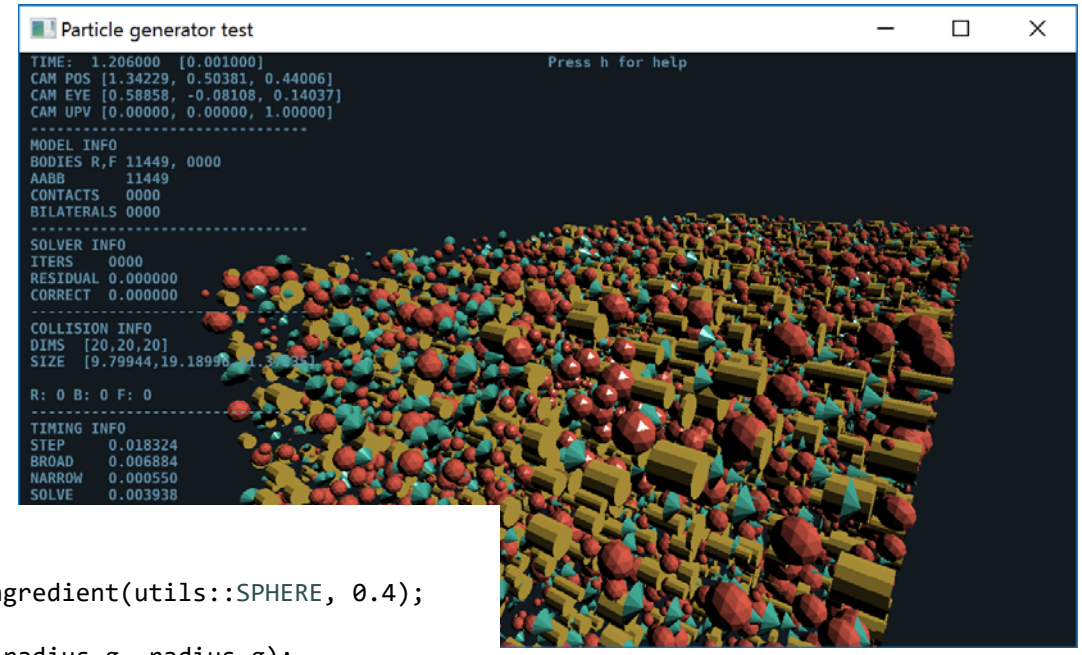

```
utils::Generator gen(system);

std::shared_ptr<utils::MixtureIngredient> m1 = gen.AddMixtureIngredient(utils::SPHERE, 1.0);
m1->setDefaultDensity(rho_g);
m1->setDistributionSize(0.5 * radius_g, 0.5 * radius_g, 0.01 * radius_g, radius_g);

double r = 1.01 * radius_g;
gen.createObjectsBox(utils::POISSON_DISK, 2 * r, ChVector<>(0, 0, 0), ChVector<>(1, 0.5, 0.3));
```

Mixing different shapes




```
utils::Generator gen(system);

std::shared_ptr<utils::MixtureIngredient> m1 = gen.AddMixtureIngredient(utils::SPHERE, 0.4);
m1->setDefaultDensity(rho_g);
m1->setDefaultSize(radius_g);
std::shared_ptr<utils::MixtureIngredient> m2 = gen.AddMixtureIngredient(utils::BOX, 0.3);
m2->setDefaultDensity(rho_g);
m2->setDefaultSize(ChVector<>(radius_g, 0.7 * radius_g, 0.5 * radius_g));
std::shared_ptr<utils::MixtureIngredient> m3 = gen.AddMixtureIngredient(utils::CONE, 0.3);
m3->setDefaultDensity(rho_g);
m3->setDefaultSize(ChVector<>(0.5 * radius_g, radius_g, 0.75 * radius_g));

double r = 1.01 * radius_g;
gen.createObjectsBox(utils::POISSON_DISK, 2 * r, ChVector<>(0, 0, 0), ChVector<>(1, 0.5, 0.3));
```

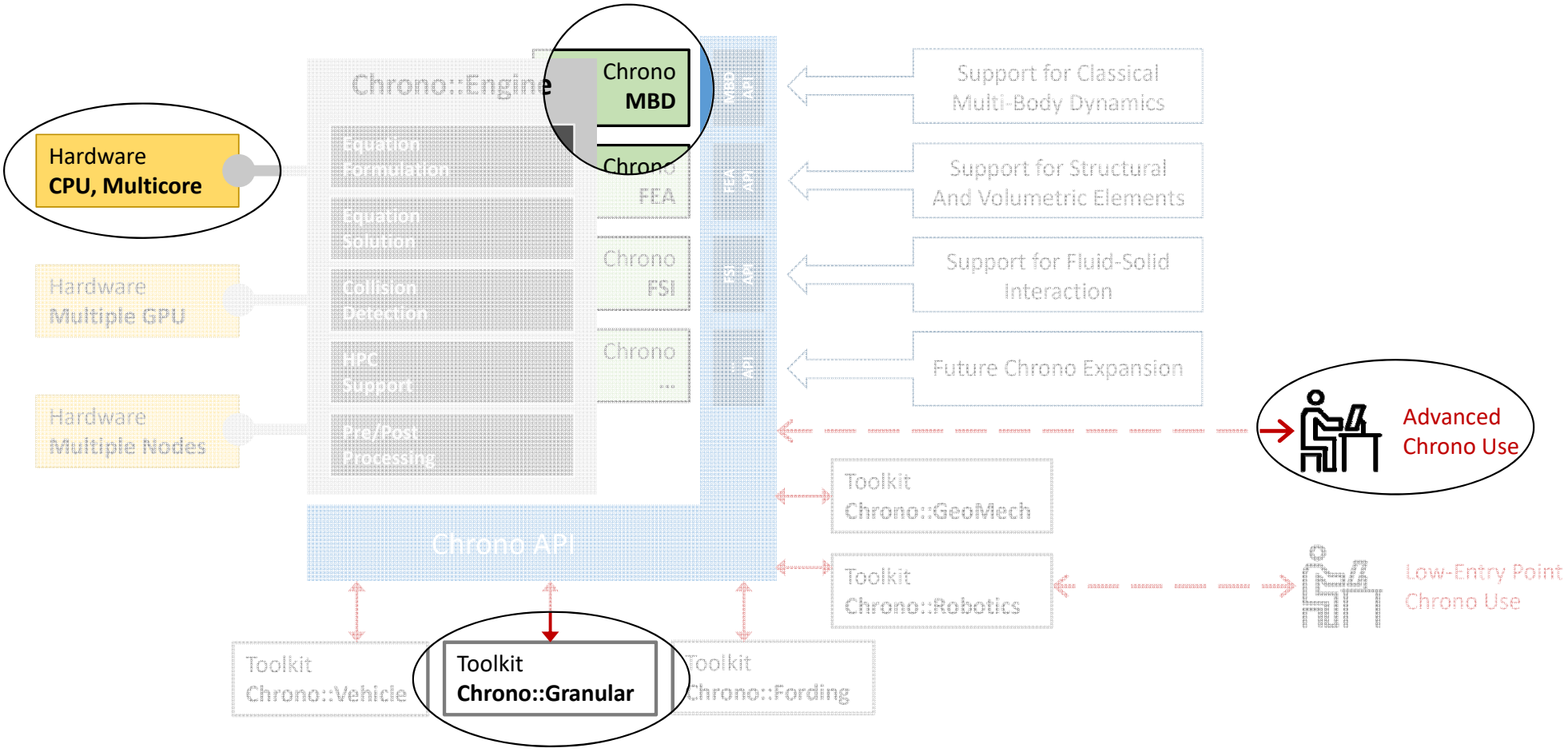
Mixing shapes and sizes




```
utils::Generator gen(system);

std::shared_ptr<utils::MixtureIngredient> m1 = gen.AddMixtureIngredient(utils::SPHERE, 0.4);
m1->setDefaultDensity(rho_g);
m1->setDistributionSize(0.5 * radius_g, 0.5 * radius_g, 0.01 * radius_g, radius_g);
std::shared_ptr<utils::MixtureIngredient> m2 = gen.AddMixtureIngredient(utils::CONE, 0.3);
m2->setDefaultDensity(rho_g);
m2->setDistributionSize(0.5 * radius_g, 0.5 * radius_g, 0.01 * radius_g, radius_g);
std::shared_ptr<utils::MixtureIngredient> m3 = gen.AddMixtureIngredient(utils::CYLINDER, 0.3);
m3->setDefaultDensity(rho_g);
m3->setDistributionSize(0.5 * radius_g, 0.5 * radius_g, 0.01 * radius_g, radius_g);

double r = 1.01 * radius_g;
gen.createObjectsBox(utils::POISSON_DISK, 2 * r, ChVector<>(0, 0, 0), ChVector<>(1, 0.5, 0.3));
```

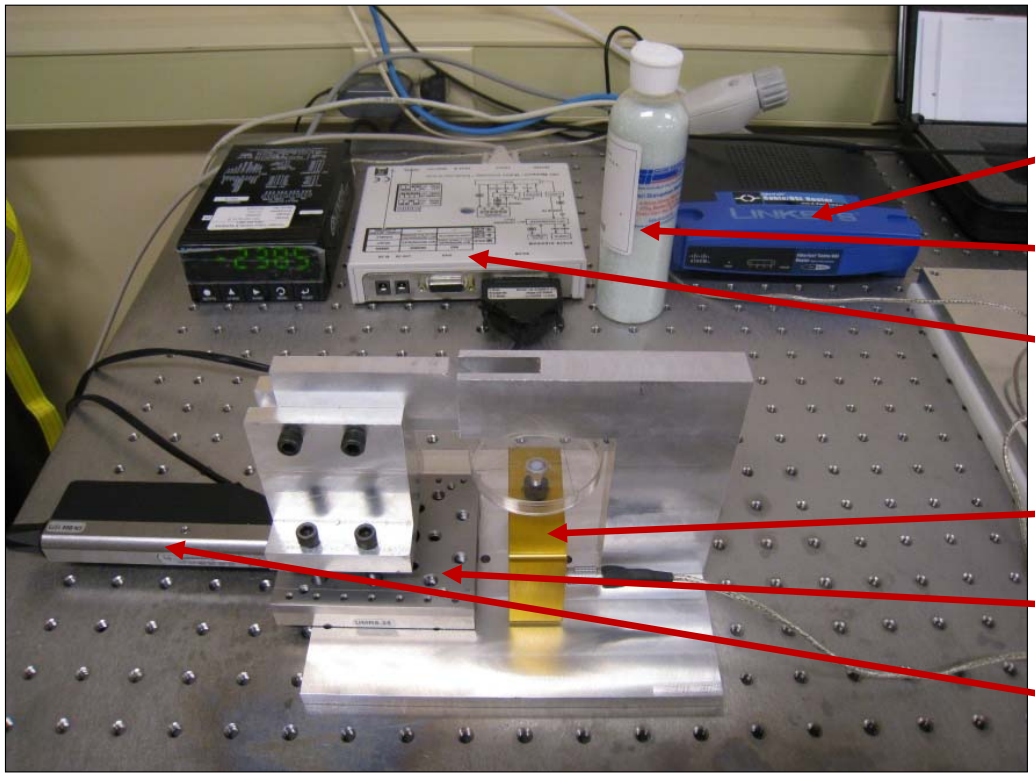
Validation studies for granular dynamics

Chrono structure



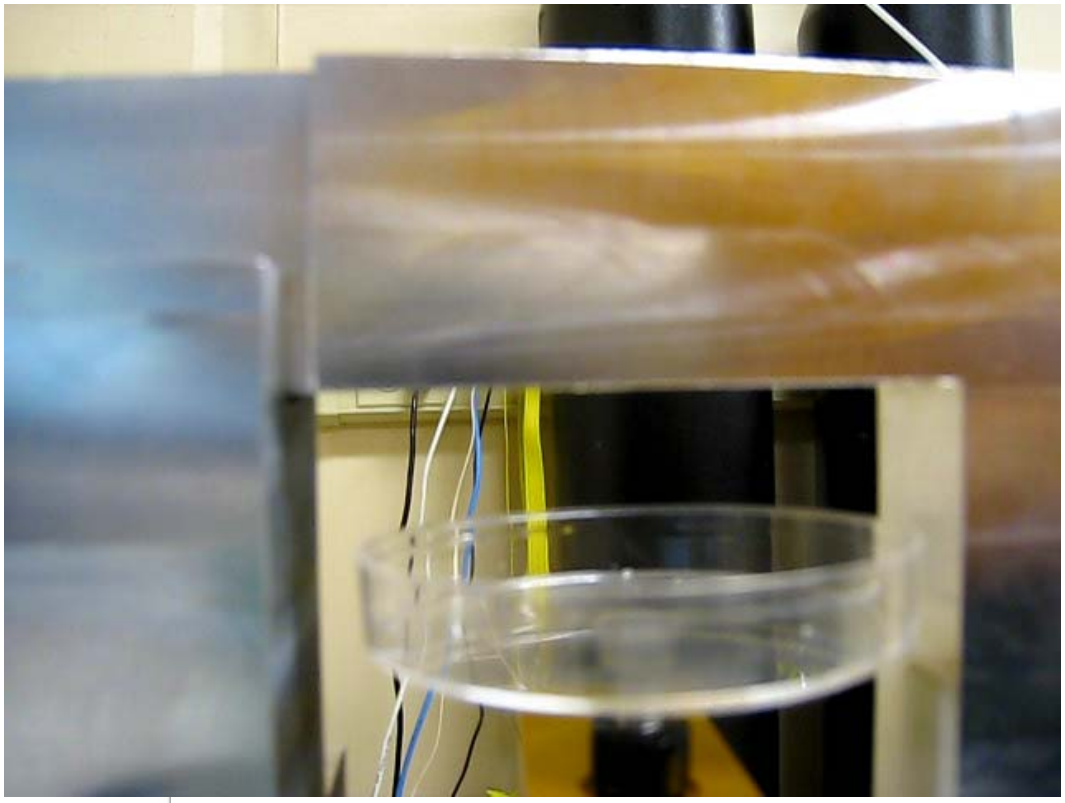
Validation Studies
Mass flow rate experiment

Experimental setup



"On the Modeling, Simulation, and Visualization of Many-Body Dynamics Problems with Friction and Contact," T. Heyn, PhD Thesis, 2013

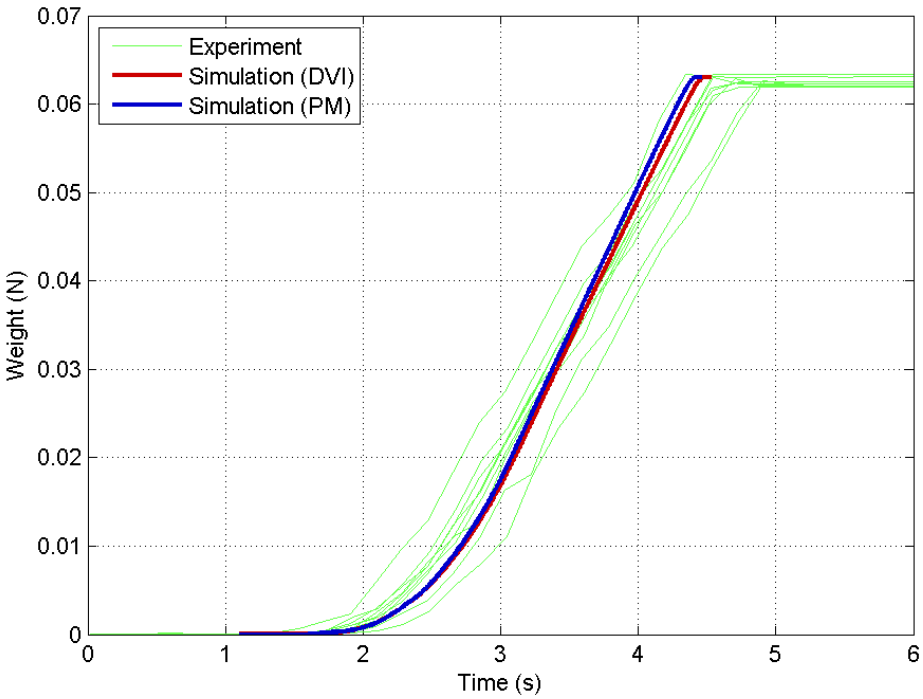
Flow Rate – 500 micron spheres



Simulation results and validation

- Total mass of granular material: 6.38 g
- Width of opening: 9.4 mm
- Opening speed: 1 mm/s
- Maximum opening gap: 2 mm

- Simulation with Chrono::Parallel



"A Comparison of Penalty and Complementarity Approaches to Discrete Element Method Simulations," R. Serban, D. Melanz, P. Jayakumar, D. Negrut, in preparation, 2016

Method comparison

Problem

- 39,000 particles
- 2 mm gap size
- 40 OpenMP threads
- $r = 0.25 \cdot 10^{-3}$ m
- $\mu = 0.3$

Complementarity

- Step-size: 10^{-4} s
- Cost per step: 0.423 s
  - Collision detection: 0.054 s
  - Update: 0.010 s
  - Solver: 0.359 s
- Average number contacts: 140,000

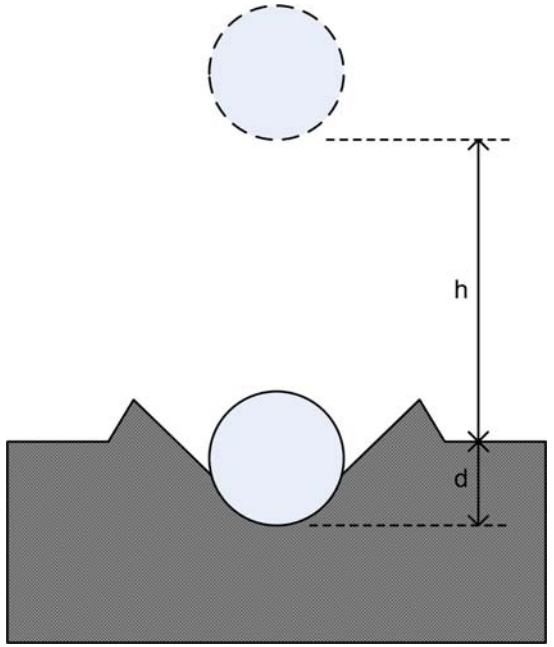
Penalty

- Step-size: 10^{-5} s
- Cost per step: 0.054 s
  - Collision detection: 0.026 s
  - Update: 0.010 s
  - Solver: 0.016 s
- Average number contacts: 118,400

Validation Studies
Low-speed impact cratering

Experimental setup

- Experiment:
  - Measured penetration of spherical projectiles into loose non-cohesive granular media
  - Parameters: ρ_b, D_b, h, ρ_g, μ
- Simulations:
  - ρ_g =1.51 g/cm^3, D_b=2.54 cm, μ=0.3, and 1 mm grains
  - h={5,10,20} cm, ρ_b={0.28,0.7,2.2} g/cm^3



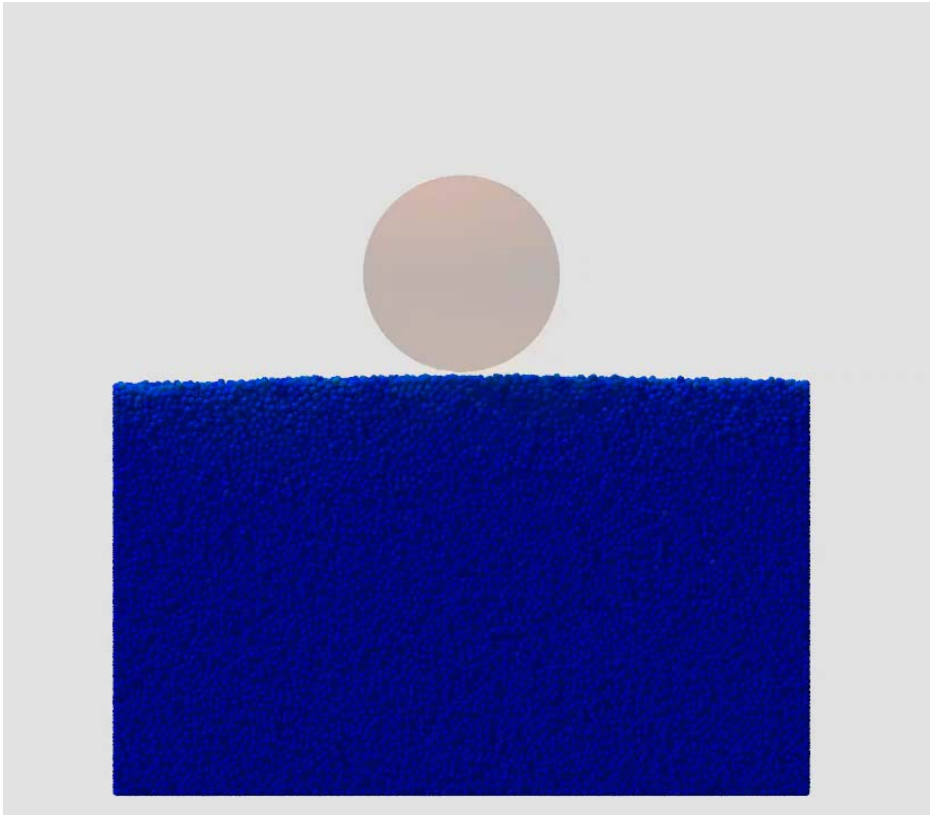
"Low-speed impact craters in loose granular media," J. S. Uehara, M. A. Ambroso, R. P. Ojha, D. J. Durian, Phys. Rev. Lett., 90:194301, 2003.
"Scaling of impact craters in unconsolidated granular materials," D. Dowling, T. Dowling, American Journal of Physics, 81, 2013

Impact simulation

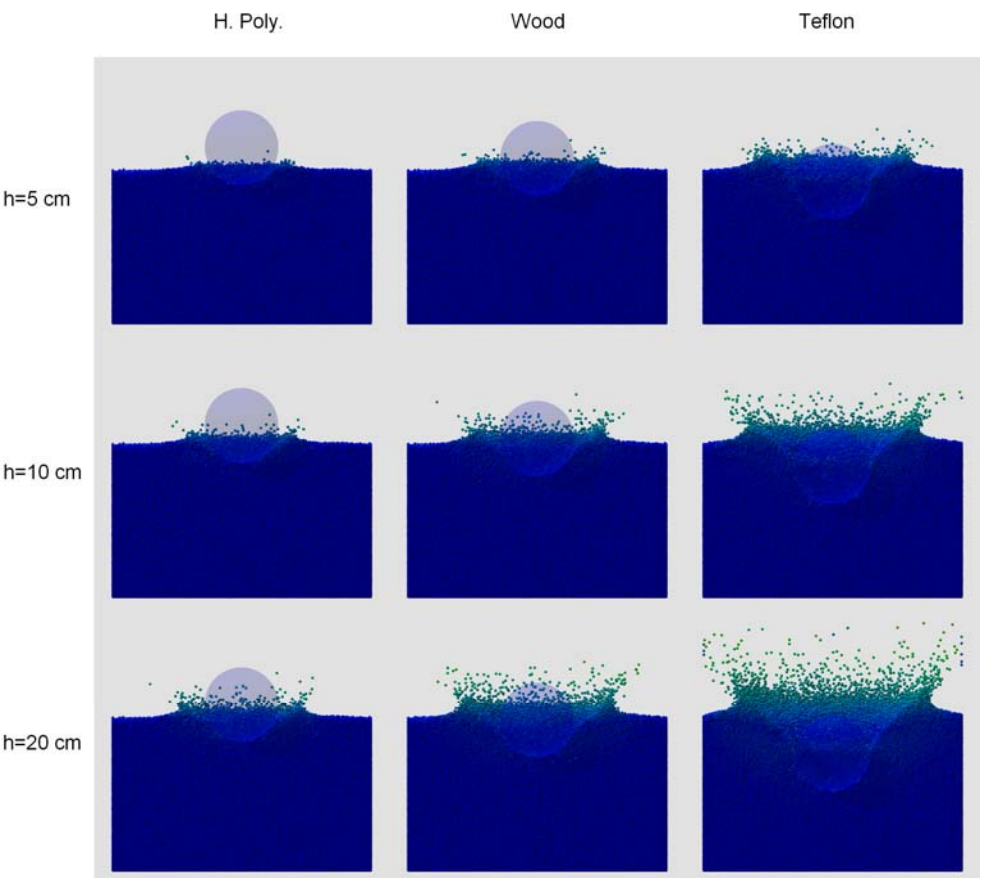
Simulation parameters (PM)

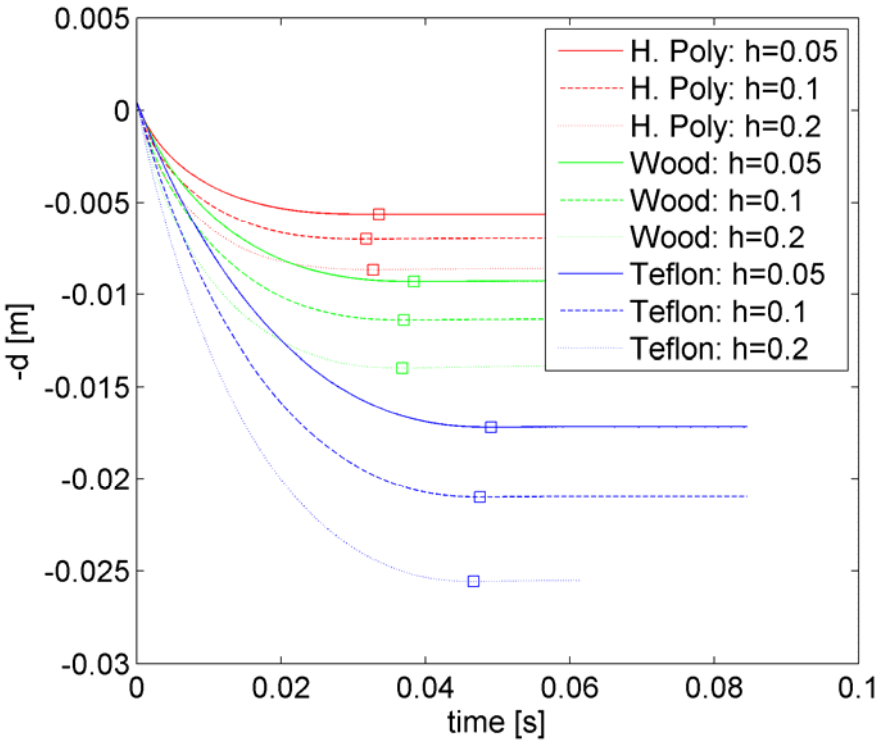
- 379,000 spheres
- r = 0.5 mm
- ρ = 1500 kg/m^3
- E = 10^8 Pa
- μ = 0.3

- Simulation with Chrono::Parallel



Deepest penetration

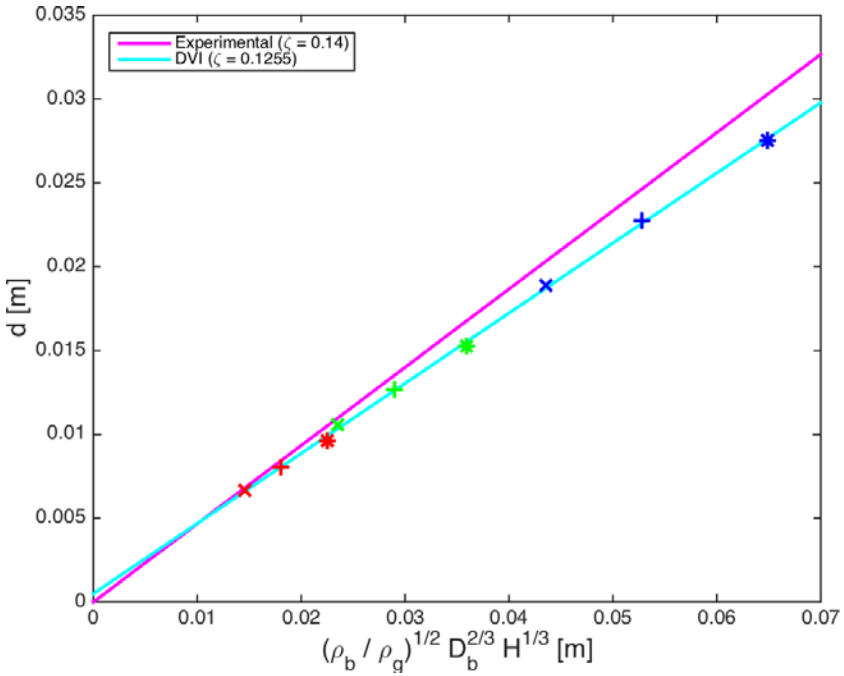




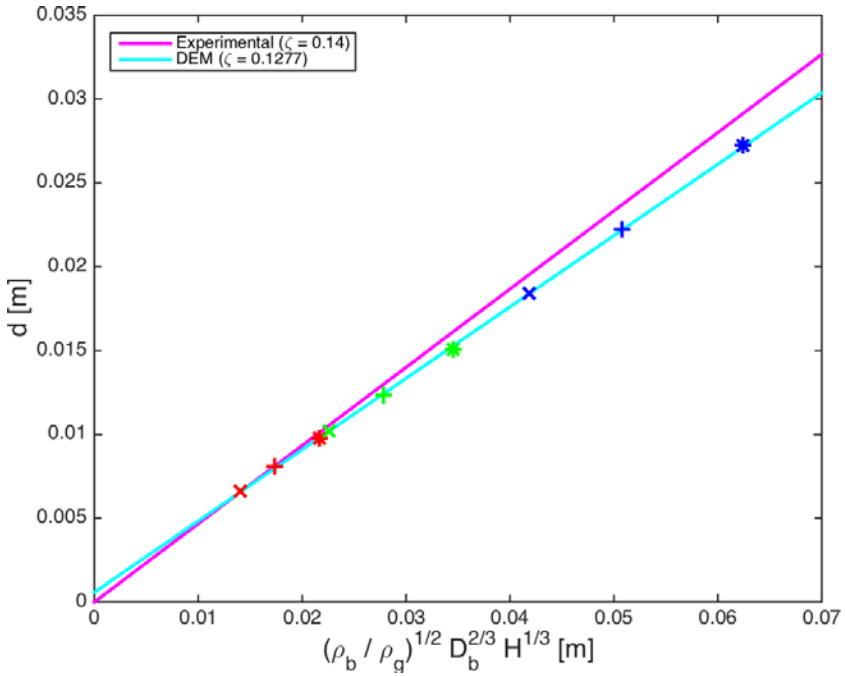
Simulation results and validation

$$d = \frac{\zeta}{\mu}\left(\frac{\rho_b}{\rho_g}\right)^{1/2} D_b^{2/3} H^{1/3}$$

Complementarity $\zeta = 0.1255$



Penalty $\zeta = 0.1277$



"A Comparison of Penalty and Complementarity Approaches to Discrete Element Method Simulations," R. Serban, D. Melanz, P. Jayakumar, D. Negrut, in preparation, 2016

Method comparison

Problem

- $\rho = 700$ kg/m^3
- $h = 0.1$ m
- 64 OpenMP threads

Complementarity

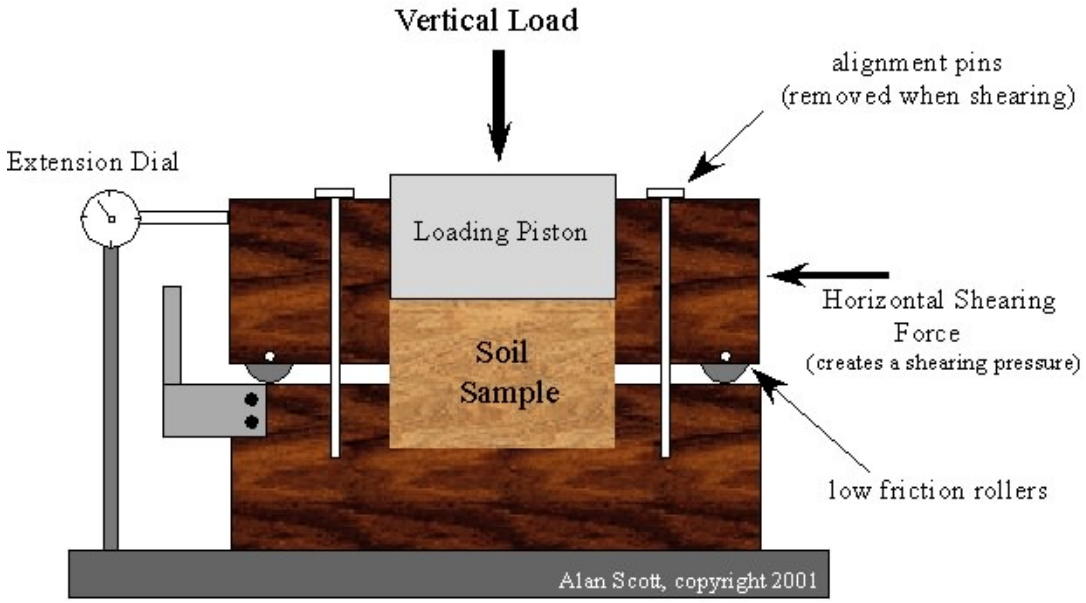
- Step-size: 10^{-4} s
- Cost per step: 3.471 s
  - Collision detection: 0.087 s
  - Update: 0.012 s
  - Solver: 3.372 s
- Average number contacts: 1.37 millions

Penalty

- Step-size: 10^{-5} s
- Cost per step: 0.144 s
  - Collision detection: 0.061 s
  - Update: 0.010 s
  - Solver: 0.072 s
- Average number contacts: 1.04 millions

Validation Studies
Low-speed direct-shear test

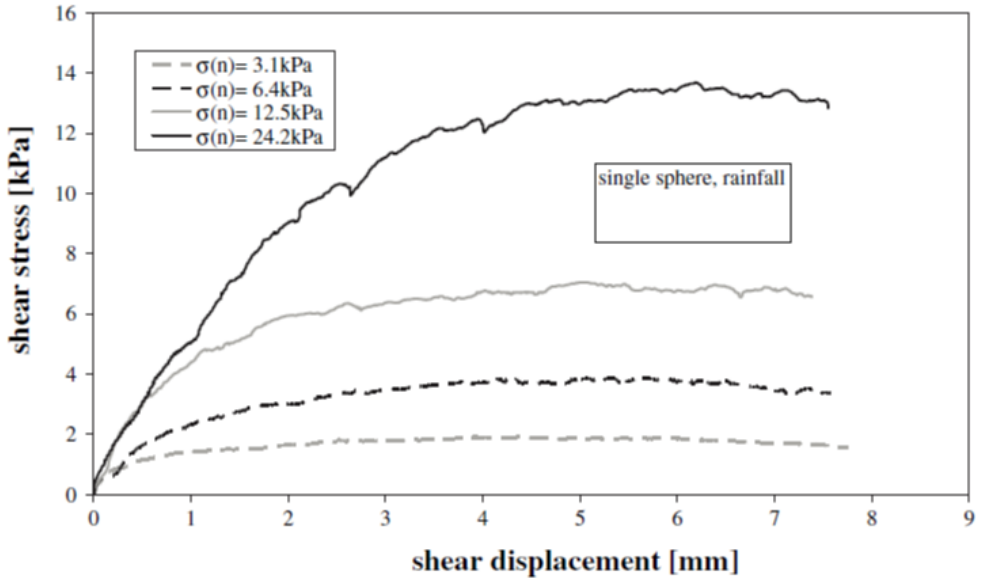
Direct Shear test



$$\tan\phi = \mu_{macro} = \frac{Shear\ Stress\ on\ Shear\ Plane}{Normal\ Stress\ on\ Shear\ Plane} = \frac{Horizontal\ Shearing\ Force}{Vertical\ Load}$$

Direct Shear validation: Uniform glass beads

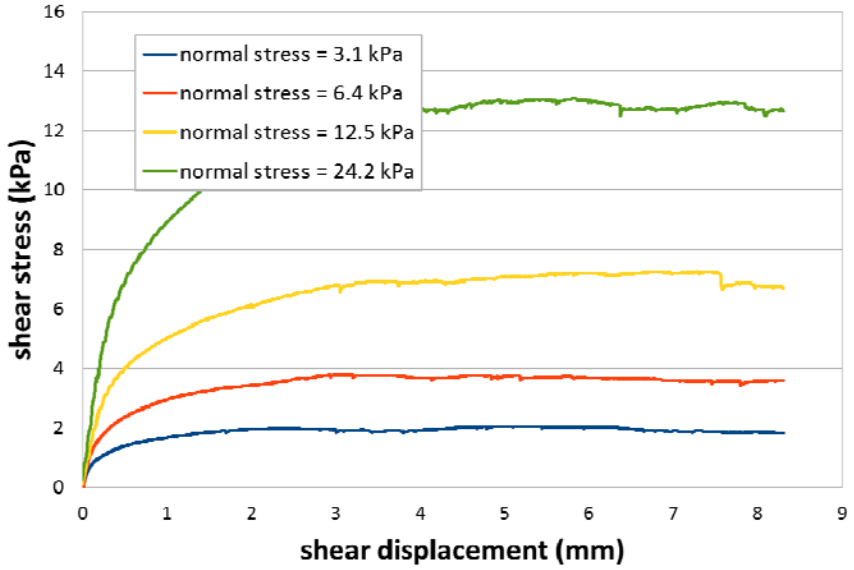
Physical Experiment: Härtl and Ooi, 2008



Void ratio e ≈ 0.7

Young's modulus E = $4(10^{10})$ Pa

Chrono: DEM-P (with tangential history)



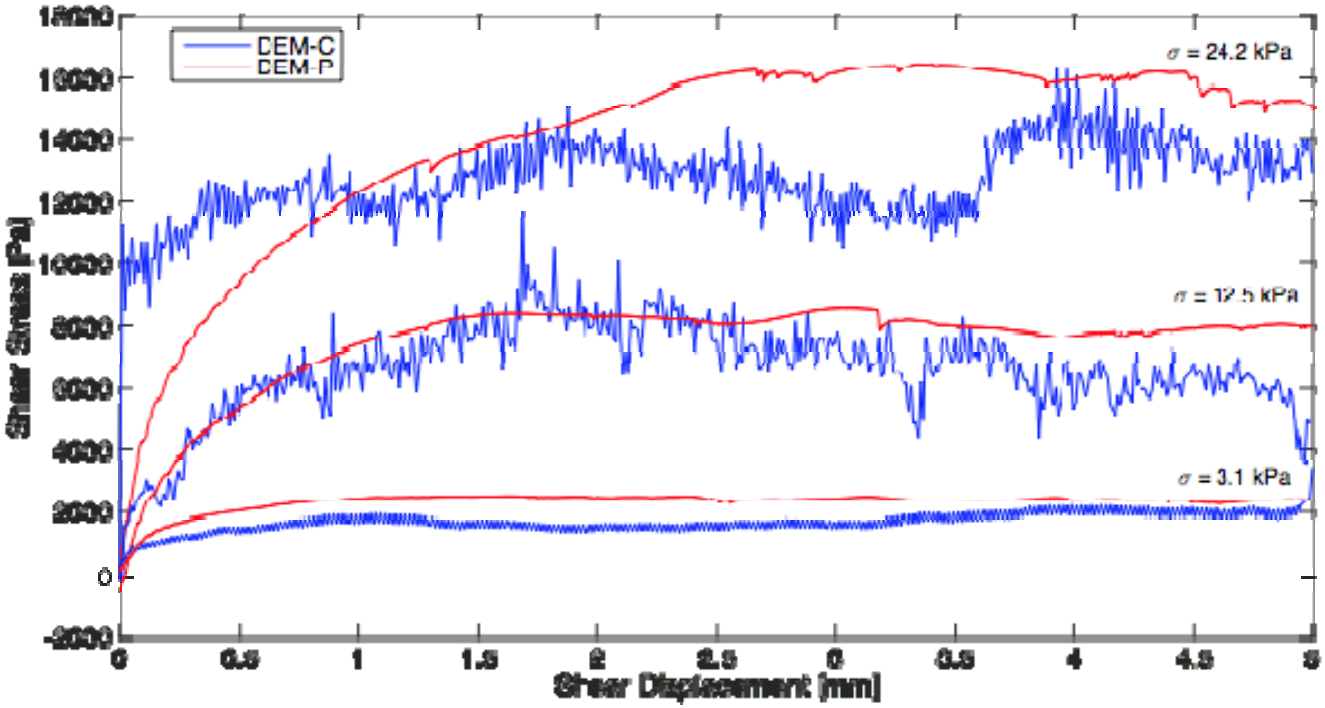
Void ratio e = 0.66, 0.65, 0.62, 0.60

Young's modulus E = $4(10^{7})$ Pa

"Experiments and Simulations of Direct Shear Tests: Porosity, Contact Friction, and Bulk Friction," J. Hartl and J.Y. Ooi, Granular Matter, 10(4), 2008.
"On the Importance of Displacement History in Soft-Body Contact Models," J. Fleischmann, R. Serban, P. Jayakumar, and D. Negrut, ASME JCND, 11(4), 2015

Simulation results

- 1000 uniform spheres randomly packed
- Particle Diameter: 6 mm
- Shear Speed: 1 mm/s
- Inter-Particle Coulomb Friction Coefficient: $\mu = 0.18$ (Glass-on-Glass)
- Void Ratio: $e \approx 0.7$ (rainfall packing)
- Simulation with Chrono::Parallel



Method comparison

Problem

- 1,000 particles
- 1 mm/s shear speed
- 10 OpenMP threads
- $r = 3 \cdot 10^{-3}$ m
- $\mu = 0.18$

Complementarity

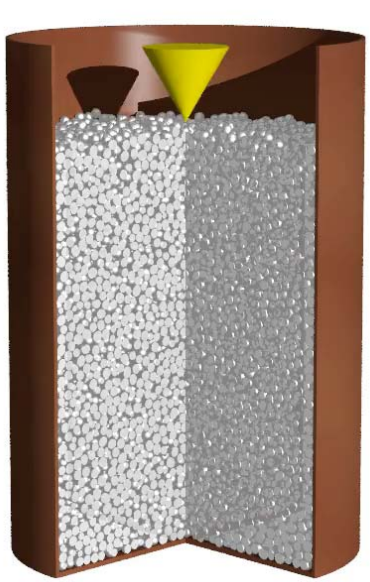
- Step-size: 10^{-3} s
- Cost per step: 0.1615 s
  - Collision detection: 0.0036 s
  - Update: 0.0007 s
  - Solver: 0.1571 s
- Average number contacts: 4800

Penalty

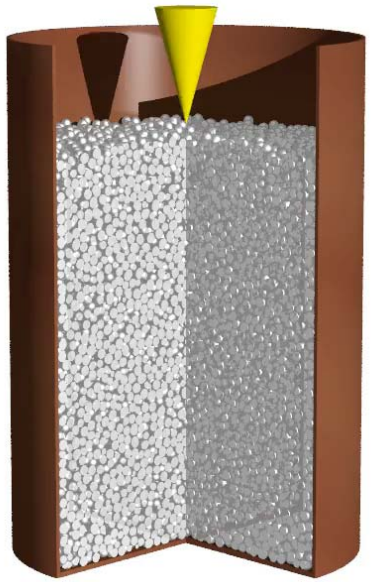
- Step-size: 10^{-5} s
- Cost per step: 0.0025 s
  - Collision detection: 0.0008 s
  - Update: 0.0007 s
  - Solver: 0.001 s
- Average number contacts: 4200

Validation Studies
Cone penetrometer test

Cone penetrometer simulations



DEM-C – 60° cone



DEM-P – 30° cone

Simulation results and validation

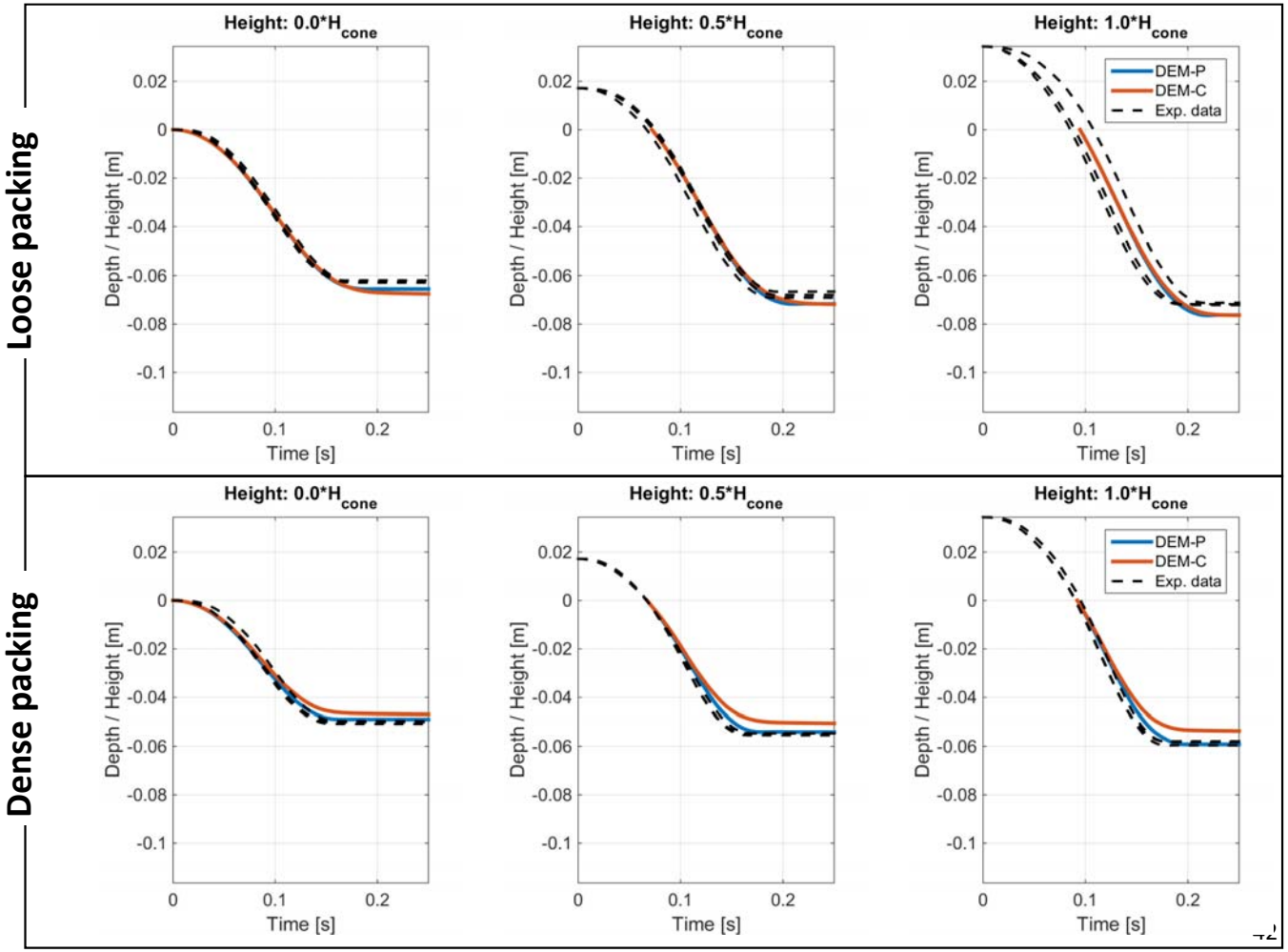
Container
- Diameter: 10.16 cm
- Height: 11.64 cm

Granular material
- Imperfect glass spheres
  $\mu = 2.84\,mm\,, \sigma^2 = 0.0834$
- Density: 2500 kg/m^3
- Friction: 0.658

Loose packing
- Density: 1504.3 kg/m^3
- Void ratio: 0.66
- Number of particles: 48864

Dense packing
- Density: 1630.3 kg/m^3
- Void ratio: 0.53
- Number of particles: 53296

Penetrometer
- Apex angle: 30°
- Base diameter: 9.21 mm
- Height: 34.36 mm



Method comparison – settling phase

Problem

- Loose packing
- 48,865 particles
- 8 OpenMP threads

Complementarity

- Step-size: 10^{-4} s, Max. iterations/step: 50
- Cost per step: 1.803 s
  - Collision detection: 0.044 s + 0.010 s
  - Update: 0.007 s
  - Solver: 1.742 s
- Final number contacts: 164,369

Penalty

- Step-size: 10^{-5} s
- Cost per step: 0.117 s
  - Collision detection: 0.050 s + 0.010 s
  - Update: 0.012 s
  - Solver: 0.045 s
- Final number contacts: 120,300

Method comparison – dropping phase

Problem

- Loose packing
- 48,865 particles
- 8 OpenMP threads

Complementarity

- Step-size: 10^{-4} s, Max. iterations/step: 50
- Cost per step: 1.994 s
  - Collision detection: 0.045 s + 0.010 s
  - Update: 0.007 s
  - Solver: 1.930 s
- Final number contacts: 165,263

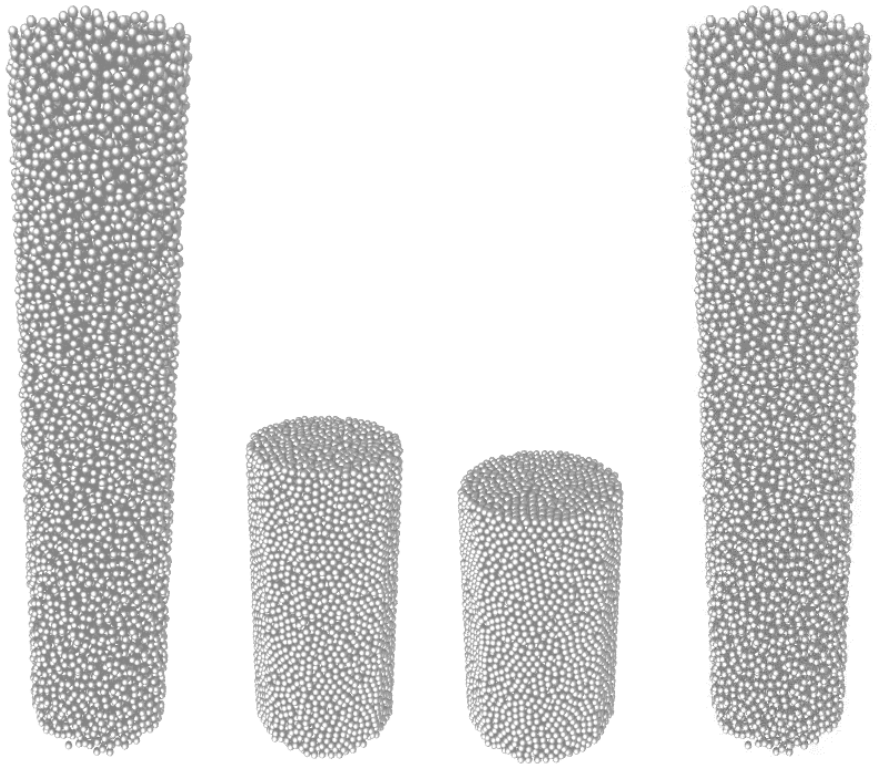
Penalty

- Step-size: 10^{-5} s
- Cost per step: 0.122 s
  - Collision detection: 0.051 s + 0.010 s
  - Update: 0.011 s
  - Solver: 0.050 s
- Final number contacts: 117,068

Validation Studies
Triaxial compression test

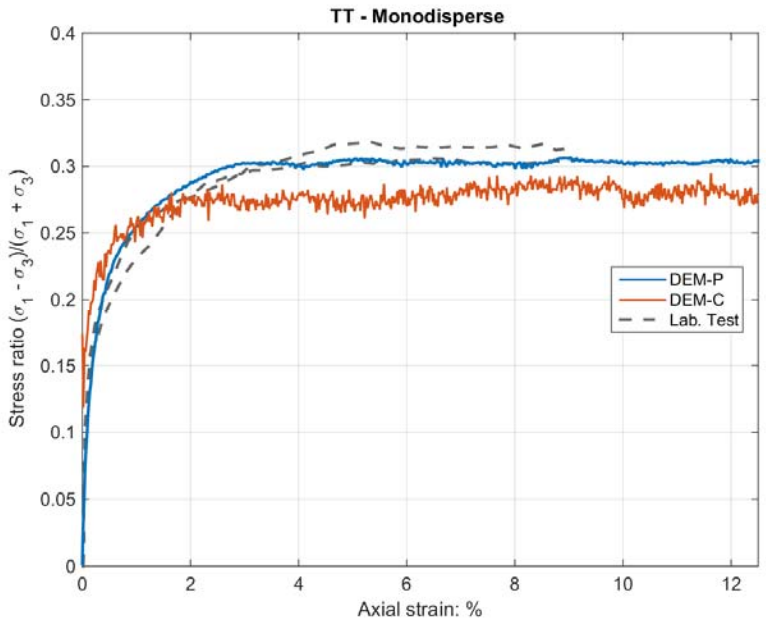
Standard triaxial test phases

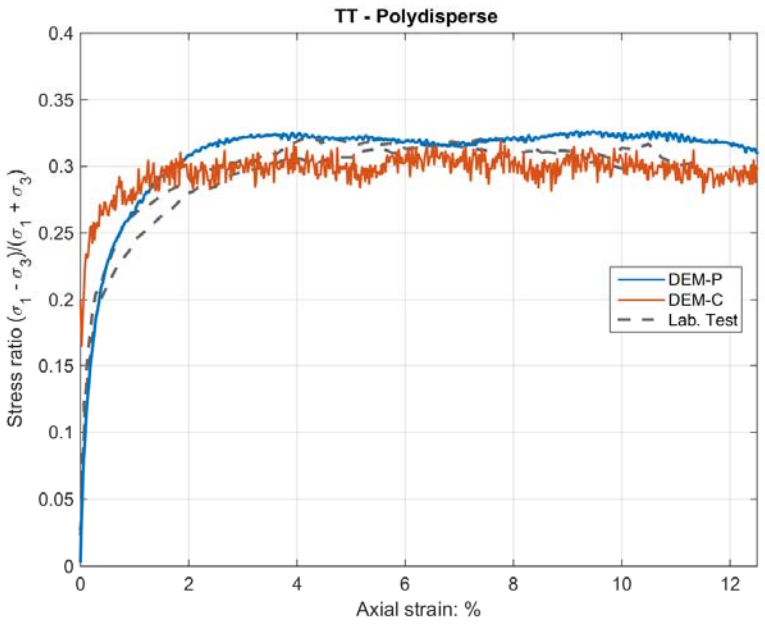
- Cylindrical Container: Φ_{cont} = 100 mm, H = 200 mm
- Granular material consists of **perfect spheres**
- Monodisperse case: φ_{beads} = 5 mm
- Polydisperse case: $\varphi_{beads} \in \{4, 5, 6\}$ mm; beads evenly distributed



Setup Settling Test

Simulation results and validation





"An analysis of the triaxial apparatus using a mixed boundary three-dimensional discrete element model," L. Cui, C. O'Sullivan, S. O'Neill, Geotechnique, 57(10), 2007
"*A fundamental examination of the behaviour of granular media and its application to discrete element modelling," O'Neill,* Master's thesis, University College Dublin, Ireland, 2005

Method comparison

Sim. Info		Monodisperse	Polydisperse
DEM-P	dt [s]	5.E-06	9.E-06
DEM-C	dt [s]	1.E-04	
	num. of iter	700	
	crs [m/s]	0.02	

Performance Comparison		Simulation Time [s]	Execution Time
Settling Part	DEM-P	0.5	1h 25 min
	DEM-C		9h 21min
Triaxial test	DEM-P	1.5	4h 55 min
	DEM-C		44h 14min

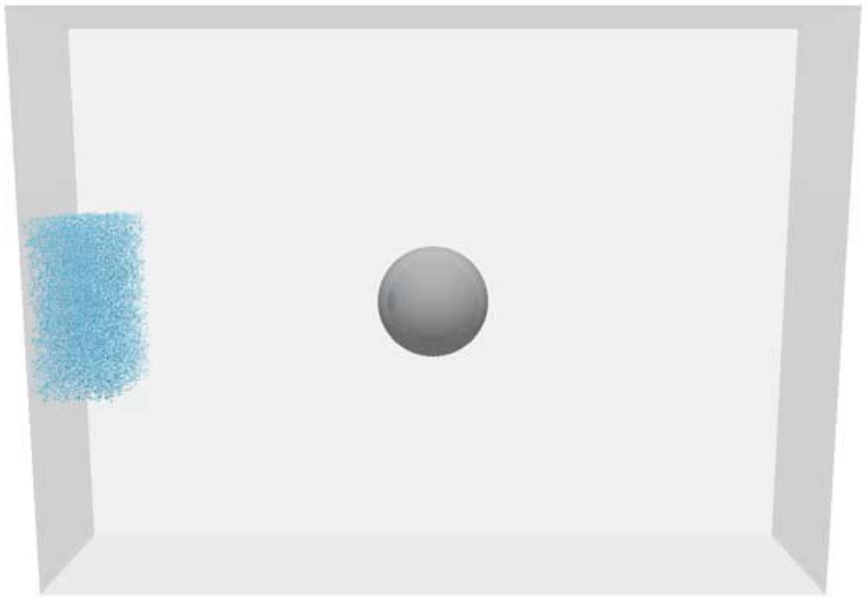
		Number of Spheres		Relative Error [%]		Void Ratio		Relative Error [%]	
Lab. Test		15382	15420			0.615	0.612		
Monodisperse	DEM-P	15918		3.485	3.230	0.641		4.266	4.777
	DEM-C					0.611		0.640	0.153
Polydisperse	DEM-P	15740		2.327	2.075	0.660		7.323	7.849
	DEM-C					0.626		1.847	2.346

Performance evolution

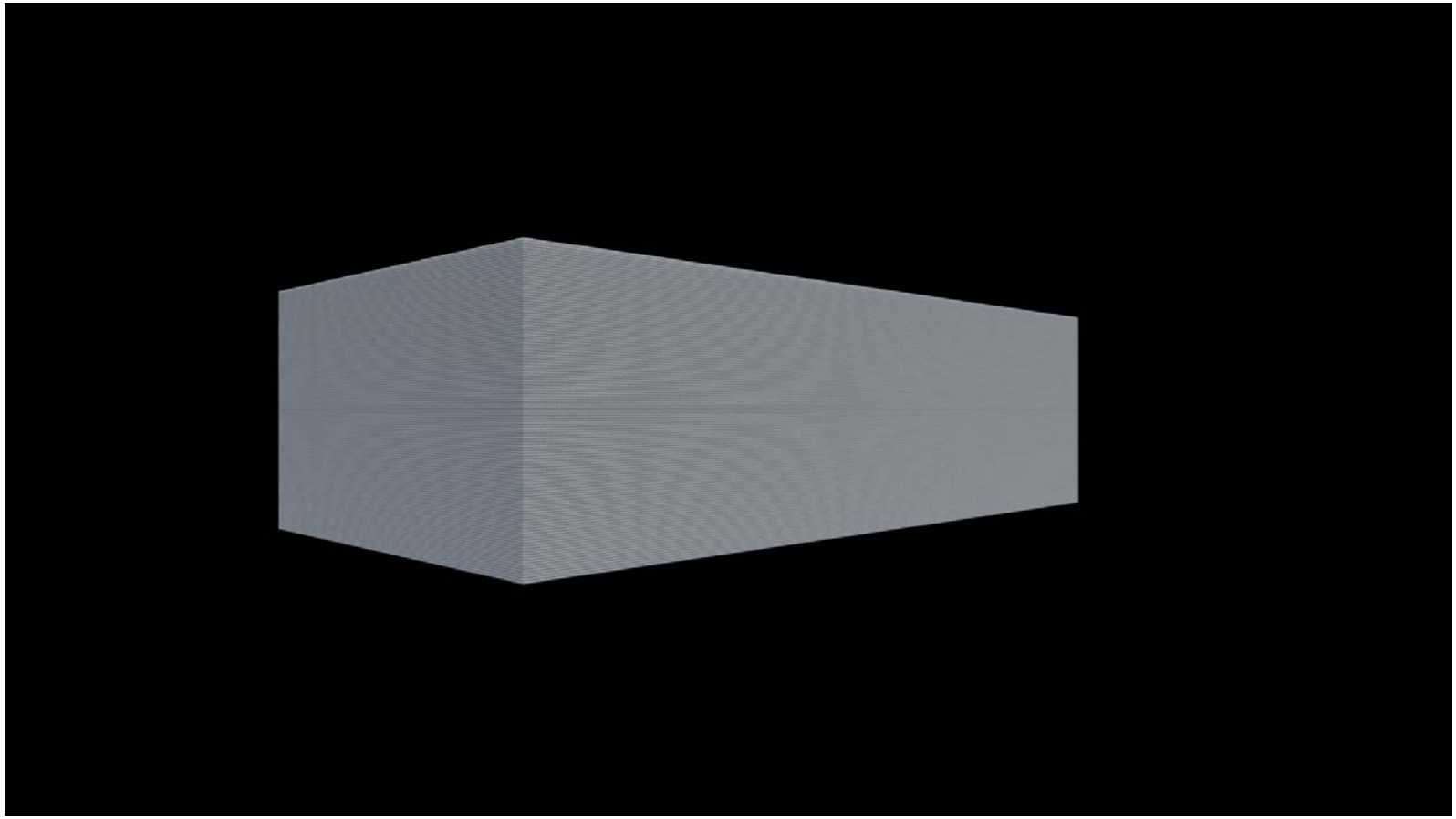
How things have evolved in the last 5 years

- Look at benchmark simulations we have been running for a while
  - Granular dynamics
  - Vehicle mobility on deformable terrain

Granular dynamics (2009)



Granular dynamics (2015)



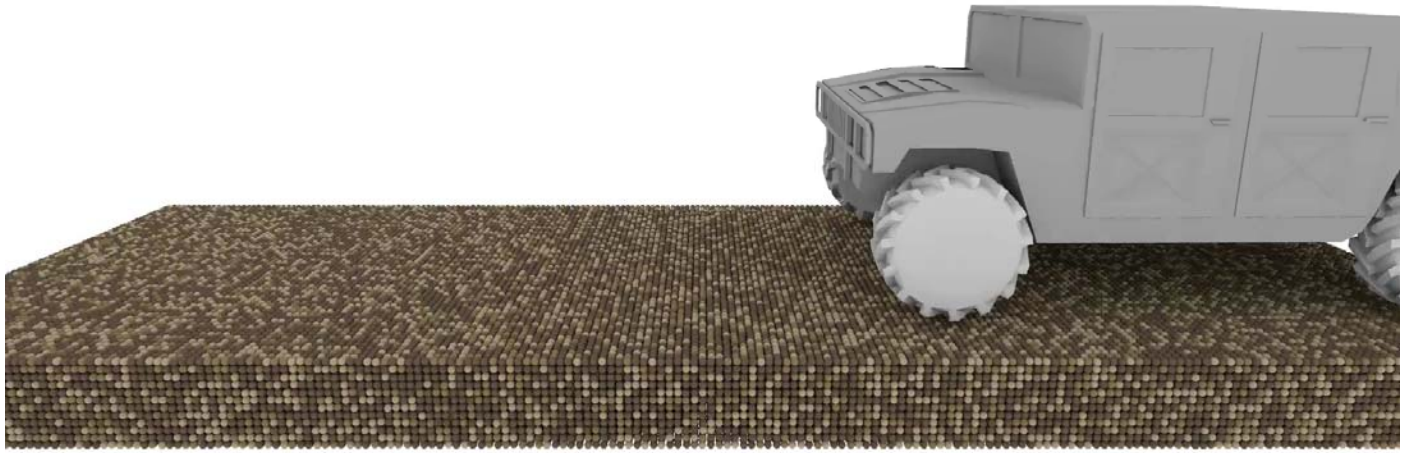
Tank wave simulation

- 2009 Implementation
  - 1 million rigid frictionless spheres
  - Integration time step: 0.01s
  - 24 sec per step
  - Hardware used: GPU
    - NVIDIA Tesla C1060
  - Velocity based complementarity
  - 20 seconds long simulation
  - Simulation Time: ~2 days

- 2015 Implementation
  - 10,648,000 rigid spheres
  - Integration time step: 0.00025s
  - 1 second per step
  - Hardware Use: GPU
    - NVIDIA Tesla K40x
  - Position based complementarity
  - 10 seconds long simulation
  - Simulation Time: 11 hours

2015 Simulation: given the number of bodies, about 20 times faster

Vehicle on granular terrain (2012)



Vehicle on granular terrain (2015)

Vehicle on granular terrain

- 2012
  - 300k rigid spheres
  - Length of simulation: 15 seconds
  - Hardware used: CPU (Intel)
    - Multicore, based on OpenMP
  - Integration time step: 0.001s
  - Velocity Based Complementarity
  - 17 seconds per time step
  - Simulation time: ~2.5 days

- 2015
  - ~1.5 million rigid spheres
  - Length of simulation: 15 seconds
  - Hardware: GPU (NVIDIA)
    - Tesla K40X
  - Integration time step: 0.0005s
  - Position Based Dynamics
  - 0.3 seconds per time step
  - Simulation time: ~2.5 hours

2015 Simulation: although 5X more bodies, runs about 25 times faster

How things have Changed in 5 years

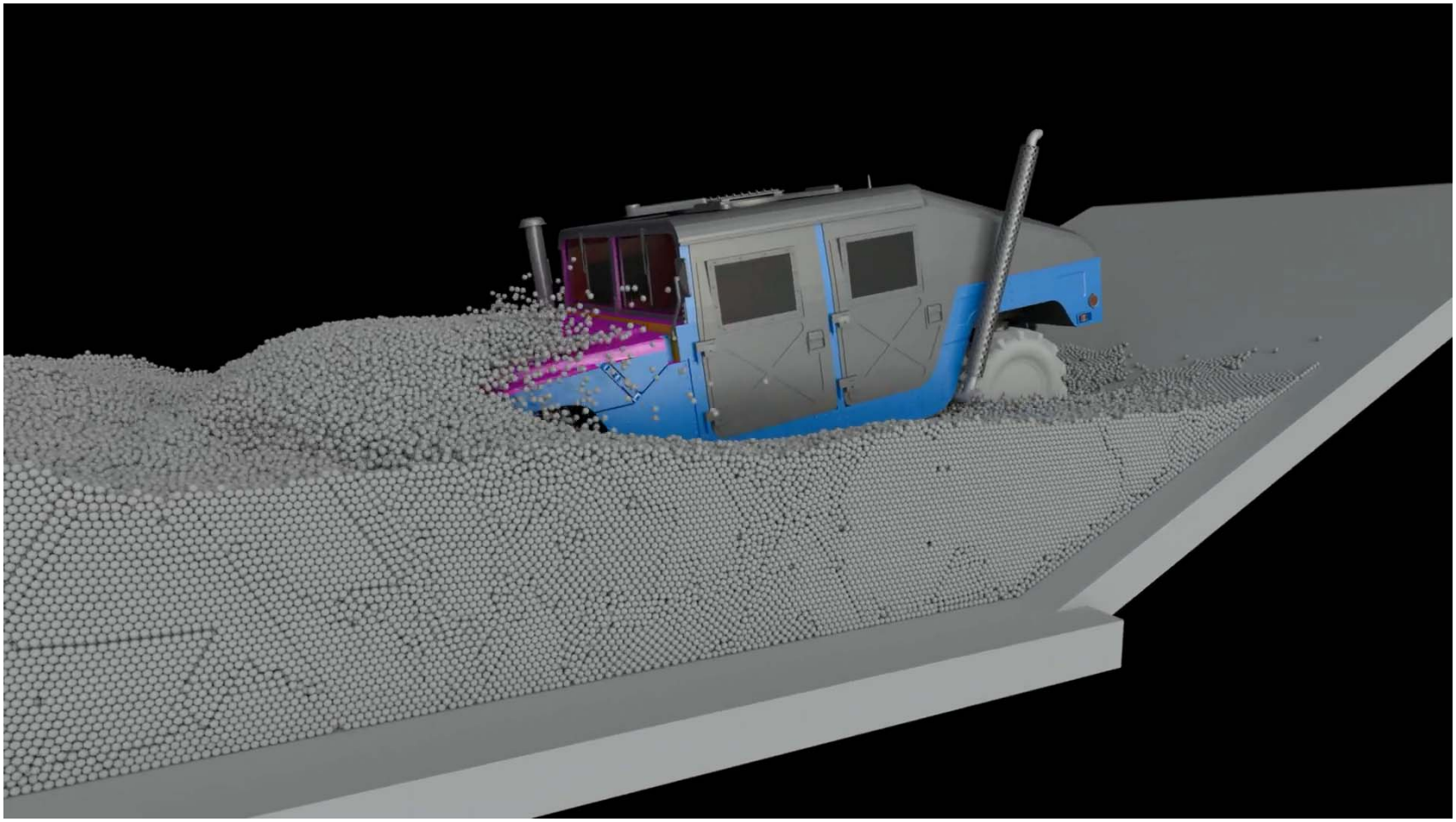
- Look at the simulation runtimes
  - Granular dynamics: about 20 times faster
    - From two days, to about two hours
  - Vehicle mobility on deformable terrain: about 100 times faster
    - Comparing a bit apples and oranges (CPU vs. GPU)
    - Say that about 25X faster when you compensate for the CPU vs. GPU difference

- Where is this 20X speedup coming from?
  1. Improved hardware
     Moore's law – more transistors, more compute power packed inside chip
  2. Faster numerical methods and algorithms
     2013 PhD thesis led to better numerical method
     4X-5X speedup

Sample simulations

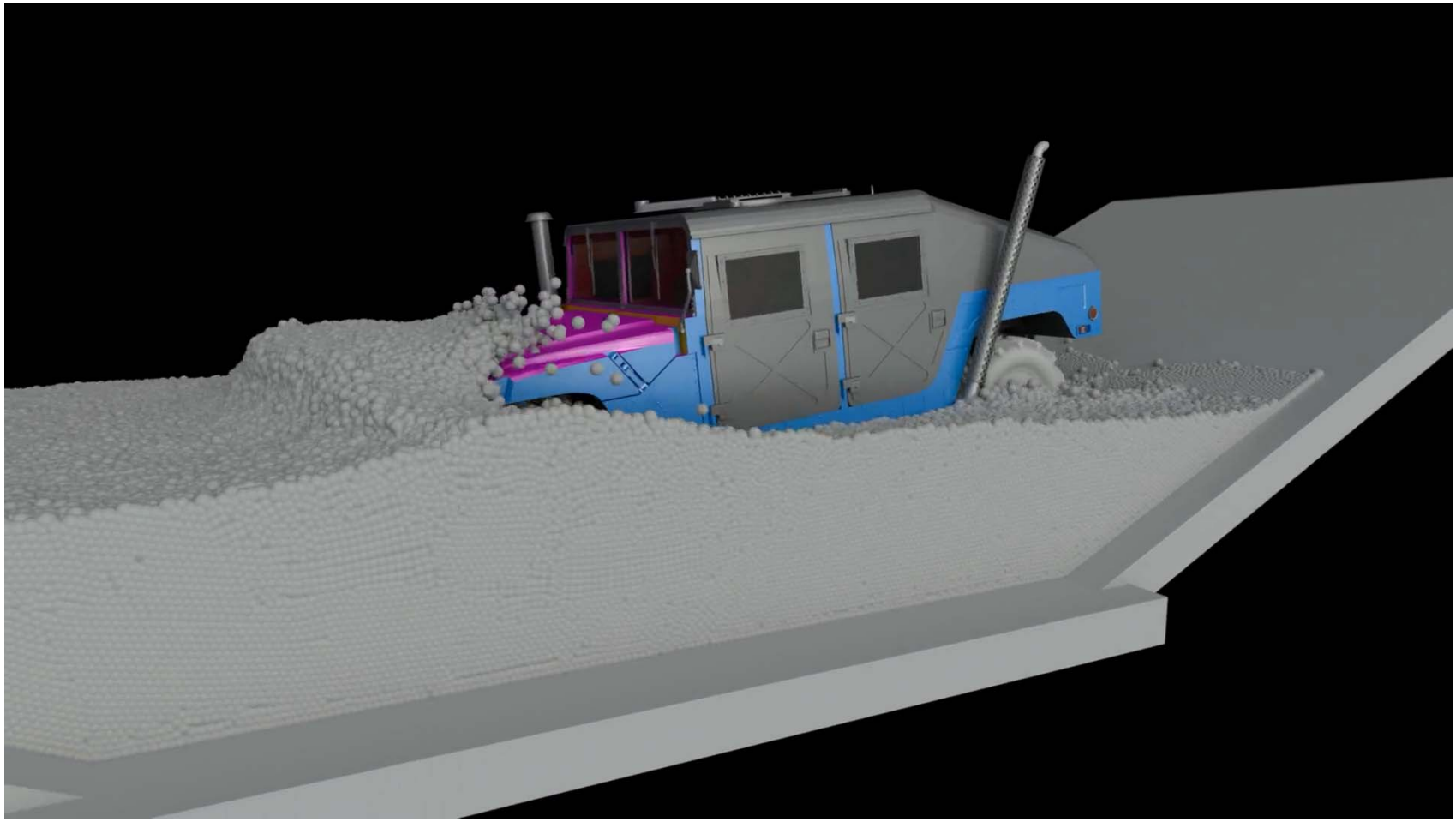
Fording simulation – frictionless granular fluid

- Fording Setup: (dimensions reduced from the original physical model)
  - Depth 8ft, Width 14ft
  - Length of bottom 15ft
- Chrono-Vehicle HMMWV Model
  - 9 Body Model, 4WD, simple drivetrain, no driver
  - Chassis/Wheels converted to convex hulls from OBJ
  - Throttle set to 1 and kept constant
- 1,051,840 rigid spheres in an HCP grid
- Integration time-step: 0.001 s, ~50 seconds per step

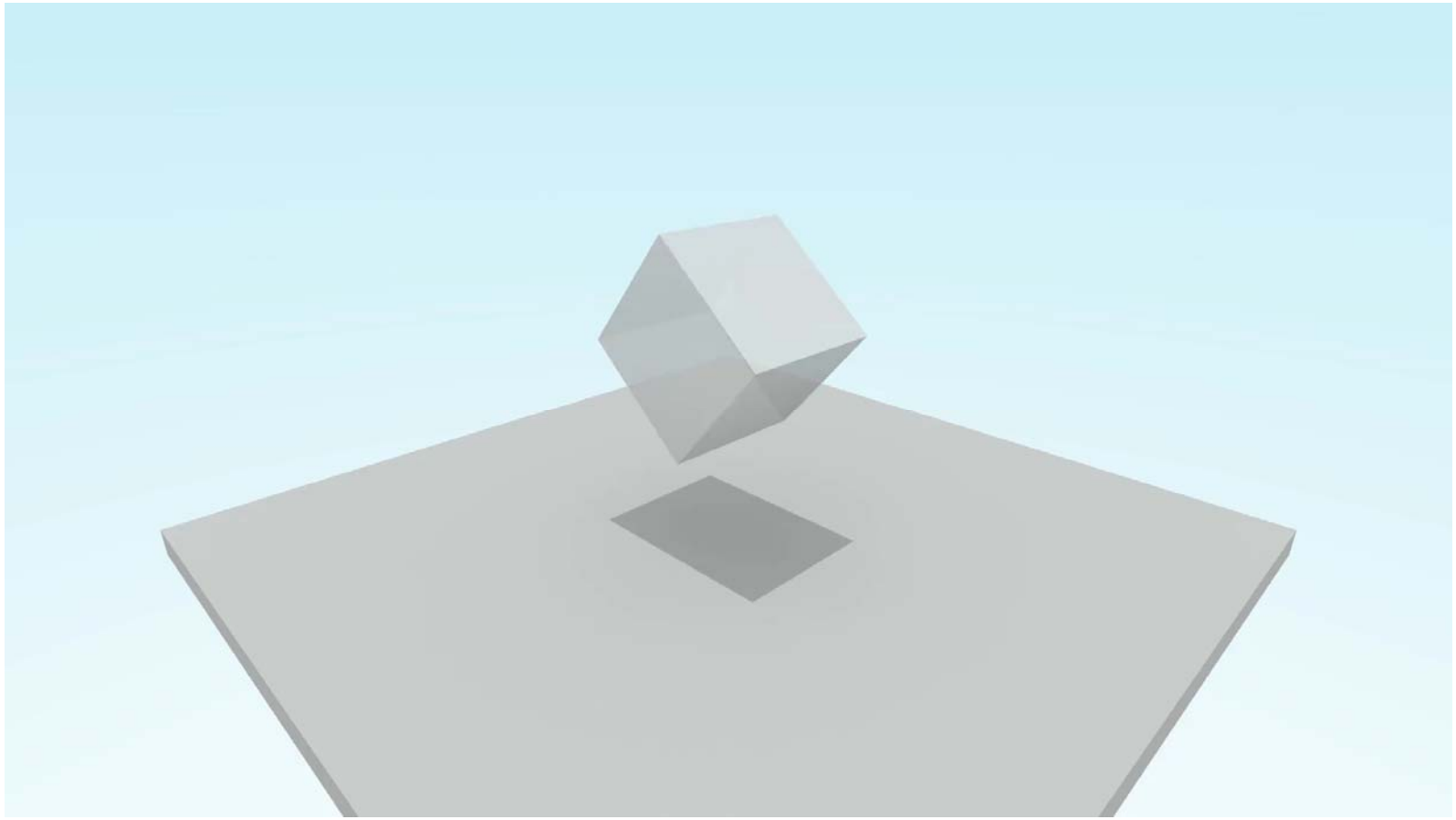


Fording simulation – constraint fluid

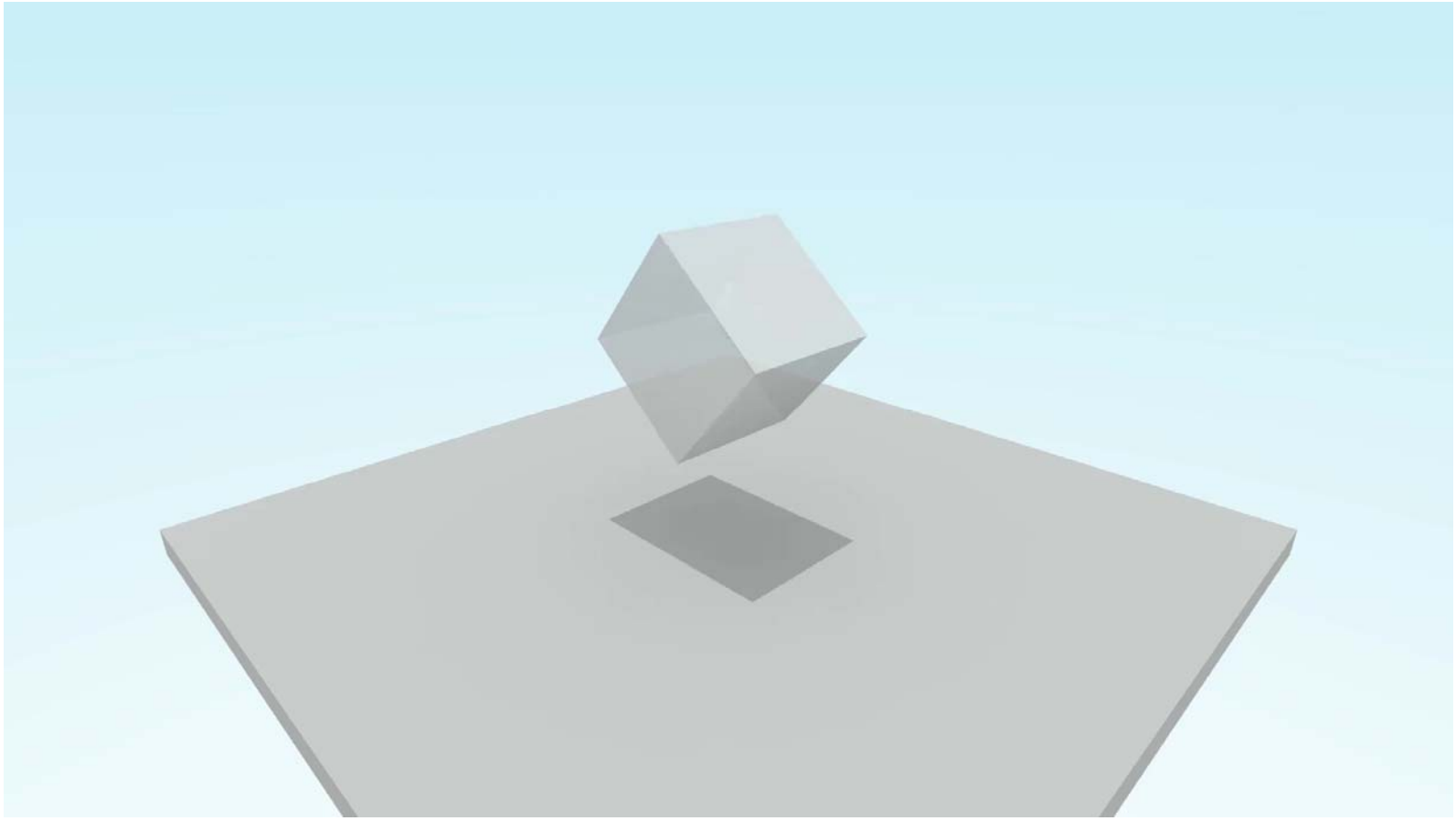
- 1,050,225 fluid markers in an HCP grid
- Integration time-step: 0.001 s, ~20 seconds per step



Cohesion (DEM-C)



Cohesion (DEM-C)



Mixing of granular material