Robust Contact Generation for Robot Simulation with Unstructured Meshes

Kris Hauser

Abstract This paper presents a numerically stable method for rigid body simulation of unstructured meshes undergoing forceful contact, such as in robot locomotion and manipulation. The key contribution is a new contact generation method that treats the geometry as having a thin virtual boundary layer around the underlying meshes. Unlike existing methods, it produces contact estimates that are stable with respect to small displacements, which helps avoid jitter or divergence in the simulator caused by oscillatory discontinuities. Its advantages are particularly apparent on non-watertight meshes and can easily simulate interaction with partially-sensed and noisy objects, such as those that emerge from low-cost 3D scanners. The simulator is tested on a variety of robot locomotion and manipulation examples, and results closely match theoretical predictions and experimental data.

1 Introduction

Physics simulators are useful tools for designing robust and safe robot behaviors because robot hardware is expensive and laborious to maintain. Control algorithms can be rapidly prototyped in simulation without the risk of damage to the robot or objects in the robots environment, and simulations provide developers with easy access to objective common benchmarks. For example, the recent DARPA Virtual Robotics Challenge asks teams to control a disaster relief robot capable of human skills in a physics simulator, with the assumption that high-quality virtual behaviors will be duplicable on a physical robot. But so far it has proven challenging to achieve high-fidelity simulations with low setup times. Existing simulators rarely yield "out of the box" performance for complex contact phenomena, and often take prohibitively long to tune. For stability, most existing simulators require that the environment and/or robot be equipped with collision hulls that approximate the true

Kris Hauser
Indiana University, Bloomington, IN, USA, e-mail: hauserk@indiana.edu

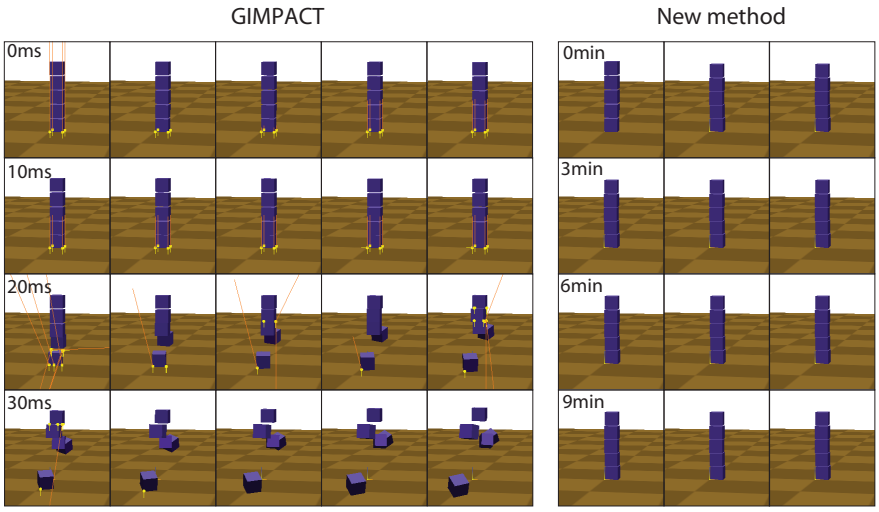

Fig. 1 A stack of triangulated cubes spaced 1 cm apart is dropped. Left: using GIMPACT, the built-in mesh-mesh collision detection in the Open Dynamics Engine and Bullet libraries "blows up" nearly instantly after the second block touches the first (20 ms). Contact points and forces are displayed. Right: the new method exhibits no perceptible jitter over several minutes of simulation time.

geometry with simple spheres, cylinders, boxes, or convex polyhedra. This is due to poor contact handling between unstructured meshes, which induces simulation artifacts like jitter, phantom impulses, or worse, divergence ("blowing up"). (Fig. 1)

This paper presents a simulator that achieves physically plausible simulations with contact between unstructured triangle meshes (so-called "polygon soup"). The heart of the method is a new contact generation scheme that yields stable contact handling (Fig. 1), and is invariant to changes of mesh topology and mesh resolution. It easily accommodates uncleaned CAD models, or noisy, partial meshes captured from 3D sensors, e.g., laser scanners or the Microsoft Kinect. This capability opens up new possibilities for rapid prototyping of robots interacting with complex environments and objects.

Particularly for objects in resting or sliding contact, discontinuities in contact generation reduce simulation accuracy and noticeably detract from realism. The technique presented in this paper is designed expressly to avoid such discontinuities. It introduces a new collision geometry representation that ensures a smooth change of contacts under small displacements for arbitrary meshes. Inspired by prior work on cloth simulation [2], the boundary layer expanded mesh (BLEM) represents object geometry as a triangular mesh fattened by a thin boundary layer. The expanded geometry is not computed explicitly, but rather only the width of the boundary layer is stored. The advantage of this approach is that as long as penetration is not too deep, penetration depth computation on a BLEM is equivalent to a distance computation on the underlying mesh. Distance computation is much more tractable than

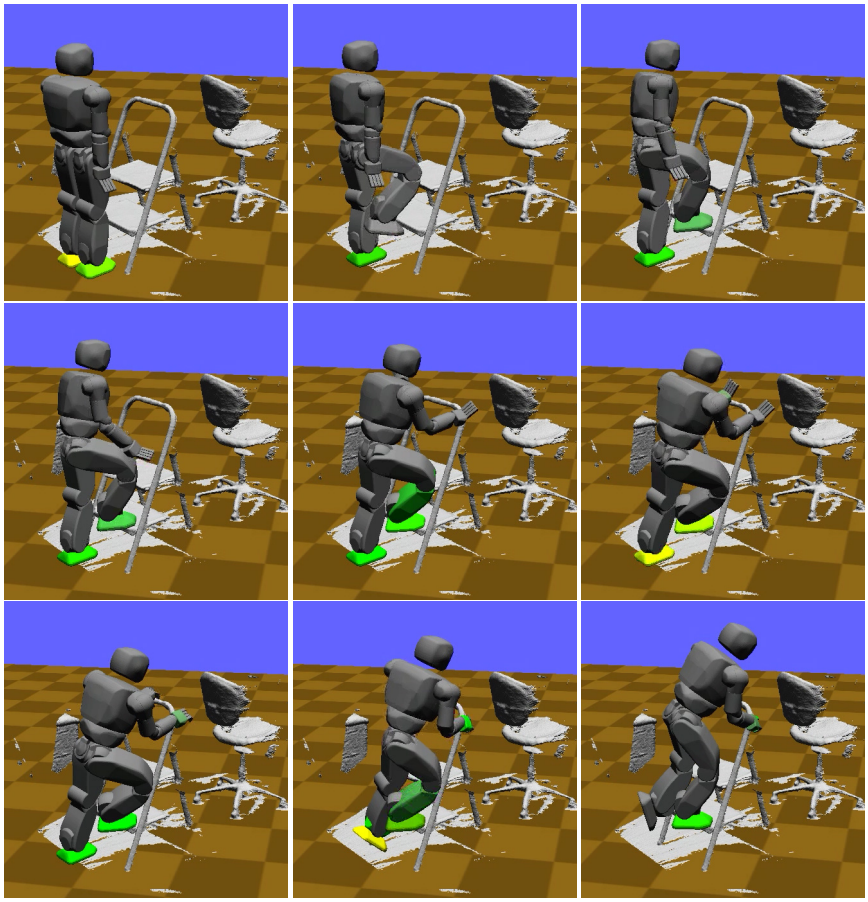

Fig. 2 Simulating the Hubo-II+ robot stepping up a stepladder sensed by a Microsoft Kinect camera. The environment model consists of 1 million triangles has many holes due to occlusions.

mesh-mesh penetration depths, and is exact as long as the boundary layer is not penetrated.

BLEMs are incorporated in an off-the-shelf rigid body simulator along with functionality to simulate robot sensors and motors. Experiments demonstrate that boundary layers only a few millimeters thick significantly improve stability above existing state-of-the-art robot simulators. Results match closely with theoretical predictions and experimental testing on a humanoid robot. The method successfully simulates many-DOF robots picking up sensed household objects and climbing on hands and legs on sensed terrains containing many holes and occlusions (Fig. 2).

The simulator is publicly available at `http://www.iu.edu/~motion/klampt/` as part of the Klamp't (Kris' Locomotion and Manipulation Planning Toolkit) software package.

2 Related Work

Many free and commercial robot simulation packages are available, the most popular ones being Gazebo [7,8], Webots [6], USARSim [4,22], and V-REP [5]. Like this work, all of them are built upon rigid body simulation engines meant for gaming and computer graphics (Open Dynamics Engine [17], Bullet [3], Nvidia's PhysX [16]). Each software package strongly recommends the use of geometric primitives or convex polyhedra for collision hulls. Non-convex, watertight meshes can be automatically decomposed into convex polyhedra [14], but complex geometries with small features and/or noise may require a huge number of convex pieces. The primary contribution of this work is a new contact generation scheme for unstructured meshes that need not be convex nor watertight.

Contact generation must be differentiated from other related problems in the simulation pipeline:

- Collision detection determines whether two geometries touch, but does not make an attempt to determine the objects motion after touching. Continuous collision detection is the problem of finding the first time of contact betwen moving bodies, and is often useful for preventing interpenetration of dynamically colliding bodies [19, 24]. However, it does not help with persistent contacts.
- Contact generation determines a representation for the region of contact that will be resolved by the contact response stage. In most systems this representation takes the form of a finite set of contact points and normal, possibly annotated with penetration depth.
- Contact response determines the future motion of objects in contact in order to prevent future interpenetration by generating physically-plausible contact forces and impulses. Solution techniques include complementarity problems [20], impulse-based methods [15], and penalty methods [21].

The problems of contact generation and contact response are highly interlinked, and the numerical stability of a simulation rests critically on the feedback loop between the two stages.

Interpenetration is impractical to prevent entirely, so simulators must anticipate small penetrations and resolve them after the fact. Hence, it is important for contact generation to compute accurate penetration depth estimates. Although doing so is exact and fast for geometric primitives and convex polyhedra, penetration depth is computationally hard to compute between nonconvex bodies. Minkowski sum methods lead to a complexity of $O(n^6)$ when only considering translational penetration depth, while considering translation and rotation leads to a complexity of $O(n^{12})$ [23]. Penetration depth approximation is an active field of study in computational geometry, but existing algorithms are still not fast enough for handling very large models with hundreds of thousands or millions of triangles [13]. Another approach [10] approximates penetration depth by precomputing a signed distance transform of the object on a volumetric grid, which leads to $O(1)$ time lookup of a point's penetration depth. Object-object contacts are detected by checking mesh vertices of one object with the signed distance table of another object, and vice

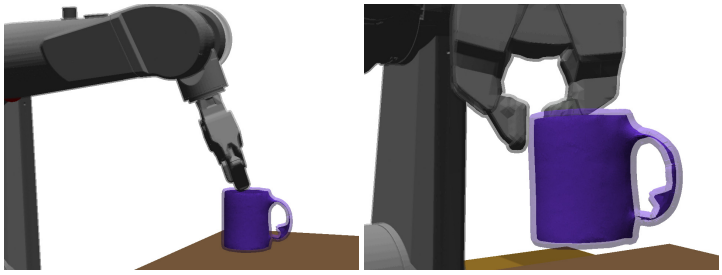

Fig. 3 Virtual boundary layers 2.5 mm thick allows a simulated robot to reliably pick up an object represented by a thin-shell triangulated mesh.

versa. Other work, primarily designed for deformable objects, estimates penetration depths of finite-element meshes by propagating a wavefront from the colliding surface [11]. Unlike these methods, the current work does not require watertight meshes. However, it is less accurate for larger penetrations, so it is best suited for use with complementarity-based or impulse-based solvers that attempt to keep penetration close to zero throughout simulation.

Boundary layers around primitives have been explored in several contexts. The Blender Game Engine [1] modifies the Bullet simulator to support boundary layers around geometric primitives and convex polyhedra and is used to improve stability and model rounded objects. Most similarly to the current work, cloth simulators often fatten triangles of the mesh and apply penalty forces, which resolve most collisions [2]. The current method applies a similar approach to rigid-body meshes, with the primary effects of producing stabler contact estimates for mesh-mesh collisions and better tolerating numerical errors an off-the-shelf complementarity-based solver.

3 Contact Generation Method

3.1 Boundary-layer expanded meshes

The boundary-layer expanded mesh (BLEM) is used as collision geometry throughout the method. A BLEM consists of a triangular mesh M along with a boundary layer width parameter $r \geq 0$ (Fig. 3), and treats collisions with the Minkowski sum of M and the sphere centered at the origin with radius r, $B(0,r)$:

$$G = M \oplus B(0,r) \tag{1}$$

Collision between objects A and B, represented as (M^A, r^A) and (M^B, r^B), is considered to occur when

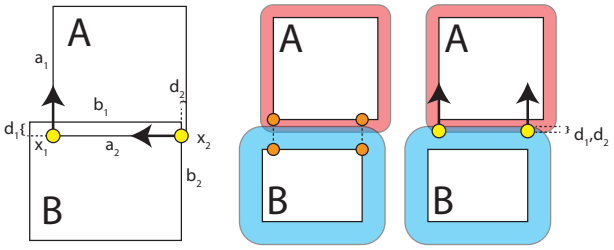

Fig. 4 Left: Even for simple 2D cases , computing contacts with local mesh features produces poor results. The minimum penetration depths from the two pairs of colliding features give rise to one vertical and one horizontal contact. If simulated, this contact formation would lead to a rightward rotation of object A, which would cause increased penetration on the next timestep. Right: the boundary-layer approach computes physically plausible contact points from the closest points on nearby features on the underlying meshes M_A and M_B.

$$(M^A \oplus B(0,r^A)) \cap (M^B \oplus B(0,r^B)) \neq \emptyset \tag{2}$$

or equivalently,

$$(M^A \oplus B(0,r^A + r^B)) \cap M^B \neq \emptyset. \tag{3}$$

3.2 Contact Generation

In addition to detecting collision, contact generation must produce a list of contact points, normals, and penetration depths so that an instantaneous motion of each point on object A in the direction of the normal reduces penetration depth. The method presented here uses a proximity checking approach on the underlying meshes, which is accelerated by a boundary volume hierarchy.

Consider the set of n triangles in a mesh: $M = \bigcup_{i=1}^{n} t_i$. The Minkowski sum of the mesh and a sphere is simply the union of the fattened triangles:

$$M \oplus B(0,r) = \bigcup_{i=1}^{n} t_i \oplus B(0,r). \tag{4}$$

For objects (M^A, r^A) and (M^B, r^B), a pair of triangles t_i^A in M^A and t_j^B in M^B are considered to be in contact iff

$$(t_i^A \oplus B(0,r^A + r^B)) \cap t_j^B \neq \emptyset. \tag{5}$$

This is equivalently expressed using the distance operator $d(t,t')$ that computes the exact minimum distance between triangles.

$$d(t_i^A, t_j^B) \leq r^A + r^B. \tag{6}$$

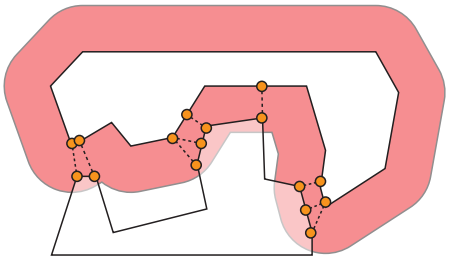

Fig. 5 The contact generator produces a contact point for all pairs of simplices whose distances are less than the sum of the boundary layer widths.

Each evaluation of d is an $O(1)$ operation and uses basic mathematical operators. In other words, the contact generator outputs a single contact point for all pairwise triangles within distance $r^A + r^B$. (Fig. 5).

In the boundary-layer regime the pair of closest points (x^A, x^B) is unique and can also be computed in $O(1)$ time (Fig. 6). The penetration depth p is $r^A + r^B - \|x^A - x^B\|$, with contact normal $n = \frac{x^A - x^B}{\|x^A - x^B\|}$. The outputted contact point is $(x^A + x^B)/2 + n \cdot (r^B - r^A)/2$, which is placed in the middle of the overlap region. If the triangles intersect, the method finds the shortest retraction amongst all candidates as illustrated in Fig. 6. However, it is important to note that once triangles penetrate, all guarantees of globally consistency are lost and the simulator loses robustness.

To efficiently detect pairs of nearby triangles in sub-quadratic time, the simulator uses using a boundary volume hierarchy (BVH) method heavily based on the Proximity Query Package (PQP) [9]. In precomputation, a BVH consisting of progressively finer bounding volumes is computed for each mesh. Oriented bounding boxes are used in the current implementation. To detect contacts between two BVHs, a depth-first-search is performed on pairs of bounding volumes and branches are pruned if the bounding boxes are more than distance $r^A + r^B$ apart. Each box-box distance computation is an $O(1)$ operation. Finally, at the leaves of the search, primitive triangles are tested for proximity using the distance operator $d(\cdot,\cdot)$. The computational complexity of this procedure is highest when many pairs of triangles are in close proximity, and is negligible when objects are distant.

3.3 Choosing boundary thickness

Contact generation is robust in the boundary layer regime, i.e., when BLEMs overlap but the underlying meshes do not interpenetrate. If meshes penetrate, the simulator falls back to less reliable contact generation methods. Hence, thicker boundary layers lead to more stable simulation. But, they lead to poorer approximations of object geometry. It may be possible to shrink meshes by the boundary thickness before simulation in order to better preserve the volume and dimensions of the object. How-

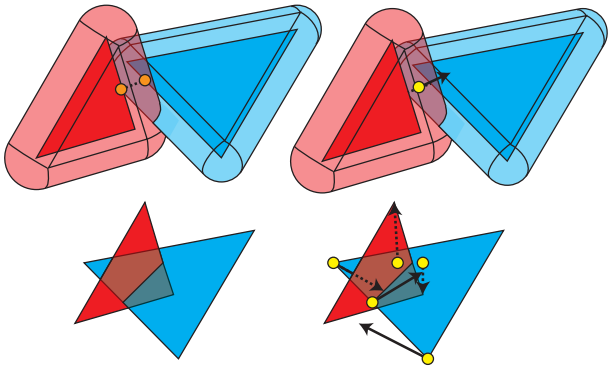

Fig. 6 Top: In the boundary layer regime, the closest points on the underlying triangles correspond unambiguously to the deepest penetrating points of the expanded triangles. It is unnecessary to consider the topology of neighbors in the mesh. Bottom: when triangles intersect, the direction of shortest retraction is not necessarily consistent with that of the neighbors, and the simulator loses robustness.

ever, this approach eliminates small features like corners, blades, and filaments. So, boundary thickness must be chosen carefully to balance the objectives of robustness and geometric fidelity.

To prevent mesh overlap, two colliding objects should not penetrate the sum of their boundary layer widths $r^A + r^B$ in a single time step. So, the boundary layer must be at least $v_{rel} \cdot \Delta t$ where v_{rel} is the relative velocity between the objects and Δt is the simulation time step. Boundary layers must also be thicker if the collision response is "soft", such as with penalty-based methods, or if very high forces are generated on light objects. Because simulation robustness depends on the sum of boundary widths, it is still possible to simulate sharp or very thin objects with zero boundary as long as they only make contact with objects with relatively thick boundaries.

3.4 Contact clustering

In its basic form, the method can produce an excessive number of contact points because a point is generated for all pairs of nearby triangles. For example, edge-edge contacts will have four replicated contact points, one for each combination of adjoining faces. Too many contact points slows down the contact response stage, and also has the potential to reduce numerical stability of complementarity problem solvers. To address this problem, the contact generator uses a clustering method that limits the number of computed contacts to a user-defined maximum k.

Each contact on a single body (x, n, p) is considered a vector in a 7-D space and k clusters are selected from this set using an axis-weighted distance metric. Experiments have tested k-means and various hierarchical clustering methods. Simulation stability does not appear to be sensitive to the choice of clustering method,

although if k-means is initialized with random clusters, a small amount of noise is injected into the simulation. Hierarchical clustering tends to produce marginally stabler results, but can be more expensive when N is large. We use an $O(N^2 \log N)$ implementation. So, when N is large, the simulator switches to k-means for a lower cost of $O(kNs)$, where N is the number of contacts and s is the iteration count (our implementation uses $s = k$).

4 Implementation and Experimental Results

Although BLEMs can hypothetically be used with any existing physics engine that accepts point contacts, the current implementation in Klamp't uses Open Dynamics Engine (ODE) v0.12 for low-level physics simulation and collision response. Experimental results are obtained on a single core of a 2.67 GHz Intel Core i7 laptop. Except for timing results, all times refer to simulation time rather than wall clock time. All simulations use a 1 ms time step, a 2.5 mm boundary on the robot and objects, and a 0 mm boundary on static environment geometry. The maximum number of contacts on the robot is 50 and the maximum number on each rigid object is 20. Supplemental videos for all examples in this paper can be viewed at `http://www.iu.edu/~motion/simulation/`.

4.1 Implementation

A robot is modeled as a linkage of rigid bodies connected by pin joints. The simulator can emulate both torque controlled and PID controlled motors. All examples in this paper use standard PID control with trajectory tracking, so that the torque applied at each motor at time t is:

$$k_P(\theta_D(t) - \theta(t)) + k_D(\dot{\theta}_D(t) - \dot{\theta}(t)) + k_I \int_{s=0}^{t} (\theta_D(s) - \theta(s))ds \tag{7}$$

where θ and θ_D are the actual and desired joint positions and $\dot{\theta}$ and $\dot{\theta}_D$ are the actual and desired joint velocities. Torques are then capped to actuator limits before being applied to the rigid bodies. The gains k_P, k_I, and k_D are specified at each joint.

Stick-slip friction is also modeled at the joints, with a simple friction model that includes dry and viscous coefficients μ_D and μ_V, respectively. In sticking mode, the friction torque is equal and opposite to the torque applied to the motor, with a breaking force of μ_D. In slipping mode, the friction torque is $-sign(\dot{\theta})(\mu_D + \mu_V|\dot{\theta}|)$.

For each motor in a robot, k_P, k_I, k_D, μ_D and μ_V must be set to reasonable values. Due to explicit time stepping in ODE, large values, especially of k_D, may cause motor instability. These parameters must be either hand-tuned or calibrated from experimental data before simulating.

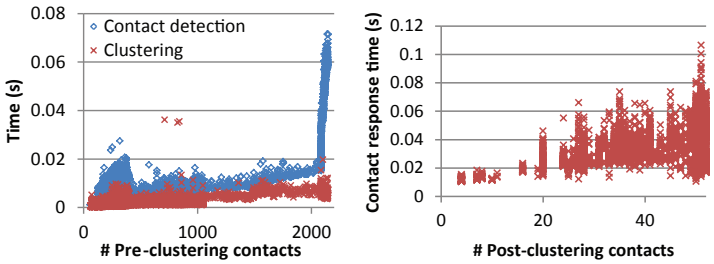

Fig. 7 Timing results for contact detection and contact response as a function of the number of contacts.

The simulator also provides functionality to emulate force/torque sensors, actuator current sensors, contact sensors, accelerometers, and tilt sensors. Sensing feedback can be incorporated into the control strategy using a straightforward plugin mechanism.

4.2 Timing

Fig. 7 shows timing statistics collected on the example of Fig. 2. On a few frames, contact generation is the limiting factor, with occasional spikes of up to 83 ms where a large number of triangles of the robot come near the environment. But on average, contact response is the dominant cost, taking 25 ms compared to 10 ms for contact generation. Clustering comprises approximately 2 ms of the average time. These plots also demonstrate that contact response becomes a dominant cost at a relatively small number of contacts (approximately 50) while the original contact detection stage routinely produces thousands of contacts for a similar computational cost. This highlights the fact that contact clustering is a critical step for achieving interactive simulation times.

4.3 Humanoid Robot Force Sensor Simulation

The Hubo-II+ is a 42 kg, 130 cm tall humanoid robot built by HuboLab at the Korean Advanced Institute of Science and Technology (KAIST). It consists of 57 rigid links including individual finger links, and 38 actuators (one actuator per finger). Numerical stability is a potential issue for simulating this robot, particularly when significant supporting forces are placed on the hands, because the mass ratio of the robot's torso to a finger link is over 150:1. The simulator has been used to simulate the robot's flat ground walking controller, and to verify the torque requirements and

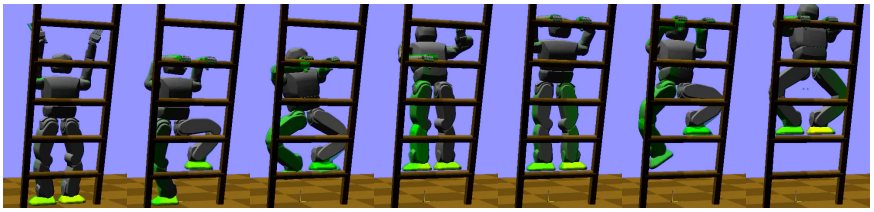

Fig. 8 A simulation of the Hubo-II+ dynamically climbing up a ladder to verify the feasibility of a quasi-statically stable motion calculated by a motion planner.

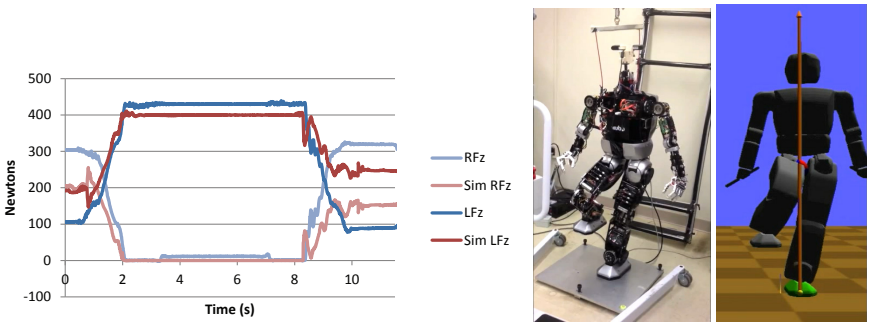

Fig. 9 Experimental comparison with force sensors at the feet of the Hubo-II+ humanoid robot as it shifts from two feet to balancing on its left foot, and then back to two feet.

dynamic feasibility of a trajectory produced by a ladder climbing motion planner (Fig. 8).

Moreover, the simulator works sufficiently stably to generate physically plausible force / torque sensor readings that could be used for force control tasks. Fig. 9 compares the results with experimental data from the force / torque sensors in the physical Hubo's ankles. Surprisingly, simulation came closer to ideal theoretical predictions! It appears that the discrepancies between the simulation and experimental data are caused by calibration issues in the physical robot: the physical robot appears to put 75% of its weight on its right foot while standing on two feet, and moreover the sum of the support forces is not constant at rest.

4.4 Comparison to GIMPACT

Because the simulator is built on top of Open Dynamics Engine, the BLEM contact generation method can be directly compared against the built-in contact detectors OPCODE and GIMPACT. This represents a fair comparison to the state-of-the-art in robot simulation: Gazebo and V-REP allow a choice between ODE and Bullet engines, both of which use GIMPACT; USARSim uses OPCODE; and Webots uses ODE. Our experiments immediately determined OPCODE to be unsuitable for re-

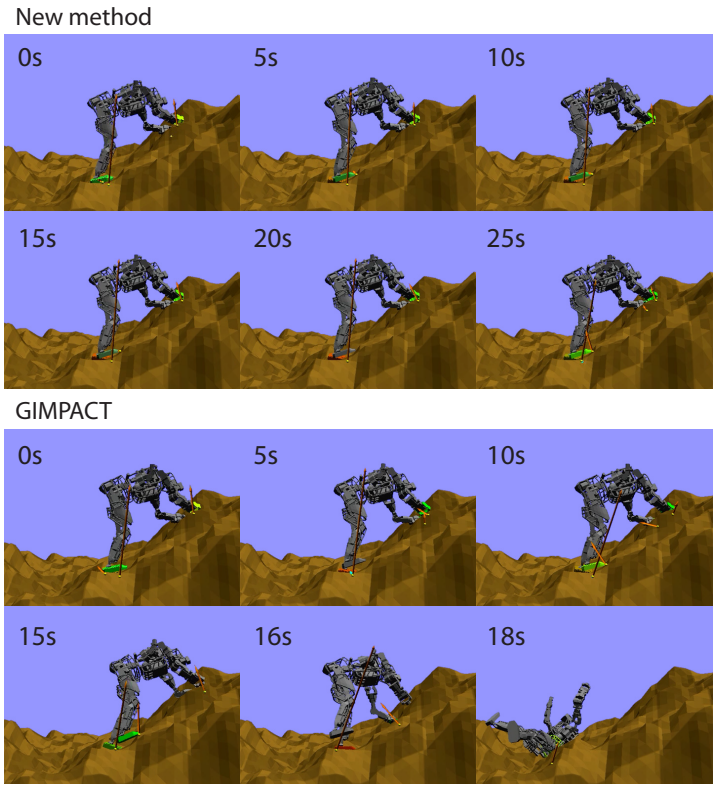

Fig. 10 A model of the Boston Dynamics Atlas humanoid balancing on hands and feet on rough, fractal-generated terrain. Contact forces are depicted as arrows. Using GIMPACT, the robot jitters until it begins an unrecoverable fall at 15 s.

alistic simulation because it would quickly "blow up" even on simple examples. GIMPACT is significantly better, and obtains realistic results for many scenarios: coarse meshes that are not-too-concave, and no edgewise contact between triangles.

Fig. 1 illustrates that GIMPACT "blows up" almost instantaneously on a simple cube stacking example where each of the cubes is given by a triangulated mesh. At the instant that cube-to-cube contact is made, GIMPACT produces a problematic set of contacts that causes catastrophic failure soon after. By contrast, the new method stays stable for tens of minutes, and appears to be stable in perpetuity. It has also been tested on a 10 cube stack for 10 minutes with less than a tenth of a millimeter of deviation from the theoretical prediction.[1]

The example of Fig. 10 shows that even when GIMPACT appears to work for a short while, subtle jitter can accumulate to yield implausible results. The Boston Dynamics Atlas robot model from DRCSim v2.2.0 is placed in a stable stance on its

[1] ODE provides an option to disable simulation of rigid bodies that are not moving, which improves the stability of stacks. Such functionality was not used in these tests.

hands and feet on a rough terrain. The robot attempts to maintain its posture. Using the new method, the robot flexes somewhat due to PID control but ultimately stays stably in place. After three minutes of simulation the robot maintained the same pose. Using GIMPACT, the robot stabilizes, but contact handling artifacts periodically apply impulses that cause the robot to jitter. After approximately 15 s these impulses grow sufficiently strong to push the robot off-balance.

The example of Fig. 11 demonstrates that the BLEM approach can handle contact between meshes of fine and coarse resolutions, while GIMPACT has problems with fine-resolution meshes. Here each fingertip of the Willow Garage PR2 gripper is modeled with approximately 1,000 triangles 1 mm^2 in size, while the cubes are coarse, 12-triangle meshes. With BLEMs, the robot lifts the cube, shakes it left and right, and places it back on the table as would be expected. With GIMPACT, the triangles of the finger mesh simply pass through the cube and the grasp fails.

4.5 Simulating Interaction with Sensed Objects

The next set of examples evaluate the ability to simulate robots interacting with objects or environments that are partially-modeled by 3D sensors.

Figure 12 shows an industrial robot interacting with several mugs sensed by stereo vision moving around the mug. The resulting point cloud was segmented from the table and meshed into a thin shell. The model has no volume and no bottom. The robot reaches into a tray of several objects and stirs them around. No interpenetration between the mugs, robot, or tray occurred during this simulation. The robot can also lift mugs from the handle or from the sides using friction (Fig. 3).

The final examples demonstrate locomotion on static environment meshes captured using the Microsoft Kinect sensor. Figs. 2 and 13 demonstrate the Hubo-II+ humanoid climbing with its hands and legs on meshes obtained through the KinectFusion algorithm [12]. These meshes are generated under indoor lighting conditions with a hand-held Kinect, with no attempt to build a complete model. As a result, the back and lower faces of the ladder and the chair are missing. Although the real chair is a swivel chair, the simulation model is fixed. Because these meshes are extremely large — 1 million and 150,000 triangles, respectively — simulation is relatively slow. For 1 second of simulation, both examples take approximately 1 minutes of wall clock time, with computation dominated by the contact response step.

5 Conclusion

This paper described a contact generation method that achieves state-of-the-art stability for “out of the box” robot simulation with unstructured triangle meshes. Experimental results demonstrate physically plausible simulations on a variety of ma-

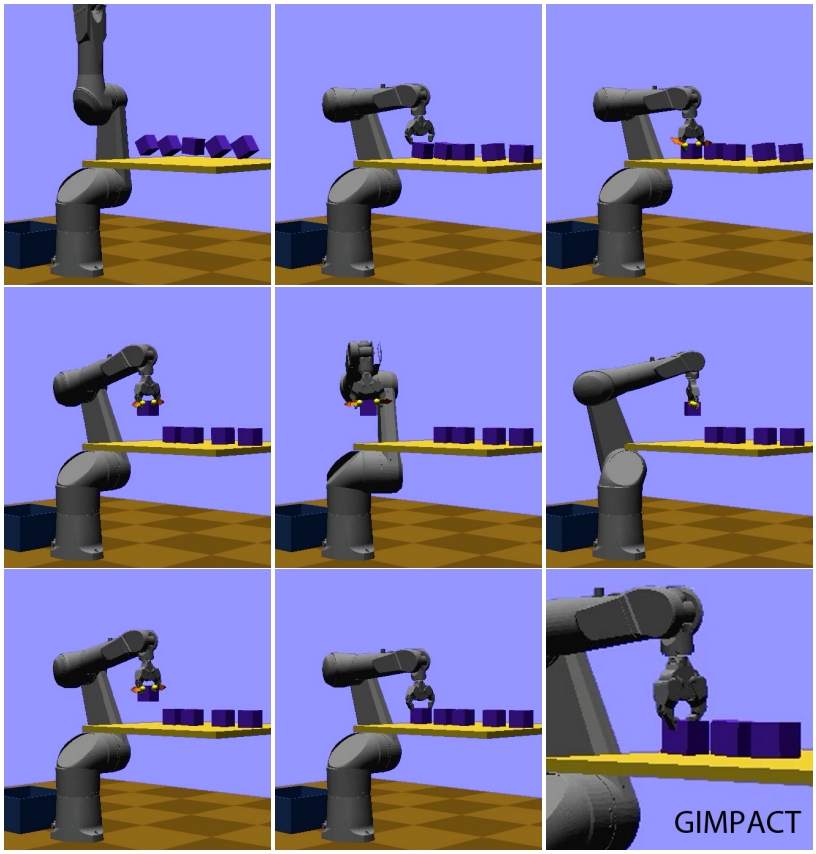

Fig. 11 A Staubli TX90L robot affixed with the gripper from the Willow Garage PR2 picks up a block, shakes it, and places it down using the new simulator. Contact forces are visualized as arrows. The last frame shows that using GIMPACT, the gripper fingers immediately slip through the block and fails to lift the block entirely.

nipulation and locomotion examples, including interactions with noisy, partial 3D scans of real-world objects.

Interesting future directions might include studying the accuracy/speed tradeoffs of clustering, and developing better clustering methods. For example, contact points scattered on a plane can safely be reduced to their convex hull without any loss of accuracy in contact response. Another possibility is to incorporate methods for accelerating BLEM contact detection with large meshes or point cloud data sets [18], possibly using GPUs. Finally, it may be possible to develop methods that adapt boundary thickness and time stepping dynamically to support collisions between very thin objects, or to employ fallback methods that retract meshes out of collision once boundary layers are penetrated.

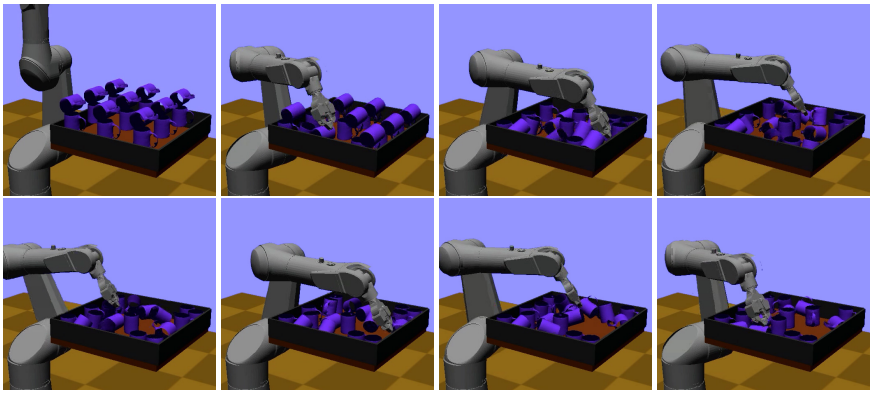

Fig. 12 A Staubli TX90L robot reaches into a tray of 18 mugs and stirs it around. Each mug was sensed using stereo vision. The mug model consists of 22,300 triangles, located only on the outward-facing sides of the cylinder and the outward-facing side of the handle (i.e., a thin shell). Throughout simulation no interpenetrations occur.

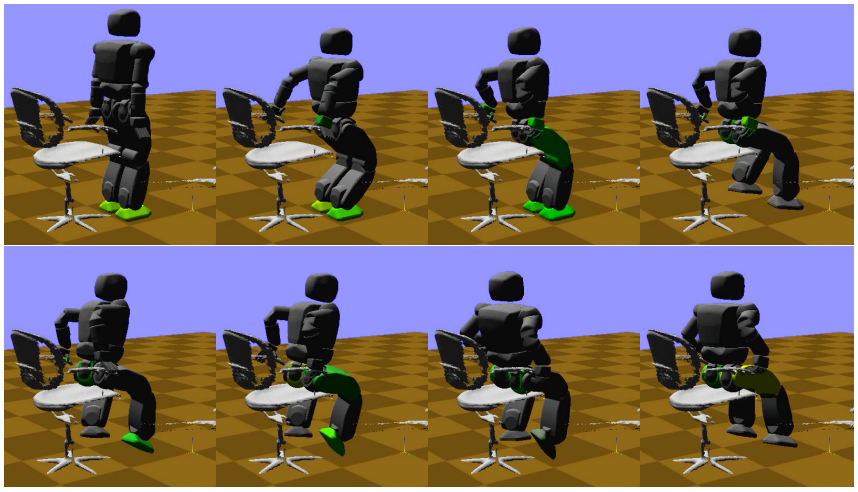

Fig. 13 The Hubo-II+ sits on a chair sensed by a Kinect. The environment model consists of 150,000 triangles and has missing data on the underside of the seats and armests and on the back of the seat. The robot is somewhat too short to sit directly onto the chair, and instead it grasps and regrasps the armrests multiple times to readjust its posture and shift onto the chair.

Acknowledgment. The author thanks Anna Eilering for capturing the Kinect meshes and H. Andy Park for testing the Hubo robot's force sensors. This work is partially supported by Defense Advanced Research Projects Agency (DARPA) award # N65236-12-1-1005 for the DARPA Robotics Challenge. Any opinion, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of DARPA.

References

1. The blender project. http://www.blender.org/ (accessed Oct 31, 2013)
2. Bridson, R., Fedkiw, R., Anderson, J.: Robust treatment of collisions, contact and friction for cloth animation. In: ACM Transactions on Graphics (Proc. ACM SIGGRAPH) (2002)
3. Bullet physics engine. http://bulletphysics.org/ (accessed Oct 31, 2013)
4. Carpin, S., Lewis, M., Wang, J., Balakirsky, S., Scrapper, C.: Usarsim: a robot simulator for research and education. In: IEEE Int. Conf. Rob. Aut., pp. 1400–1405 (2007). DOI 10.1109/ROBOT.2007.363180
5. Coppelia Robotics: Virtual robot experimentation platform (v-rep). http://www.coppeliarobotics.com/ (accessed Oct 31, 2013)
6. Cyberbotics: Webots: the mobile robot simulation software. http://www.cyberbotics.com/ (accessed Oct 31, 2013)
7. Gazebo. http://gazebosim.org/ (accessed Oct 31, 2013)
8. Gerkey, B.P., Vaughan, R.T., Howard, A.: The player/stage project: Tools for multi-robot and distributed sensor systems. In: Proc. 11th International Conference on Advanced Robotics, pp. 317–323 (2003)
9. Gottschalk, S., Lin, M., Manocha, D.: OBB-tree: A hierarchical structure for rapid interference detection. In: ACM SIGGRAPH, pp. 171–180 (1996)
10. Guendelman, E., Bridson, R., Fedkiw, R.: Nonconvex rigid bodies with stacking. In: ACM Transactions on Graphics (Proc ACM SIGGRAPH) (2013)
11. Heidelberger, B., Teschner, M., Keiser, R., Müller, M., Gross, M.: Consistent penetration depth estimation for deformable collision response. In: Proc. Vision, Modeling, Visualization, pp. 339–346 (2004)
12. Izadi, S., Kim, D., Hilliges, O., Molyneaux, D., Newcombe, R., Kohli, P., Shotton, J., Hodges, S., Freeman, D., Davison, A., Fitzgibbon, A.: Kinectfusion: Real-time 3d reconstruction and interaction using a moving depth camera. In: ACM Symposium on User Interface Software and Technology (2011)
13. Kim, Y., Otaduy, M., Lin, M., Manocha, D.: Fast penetration depth computation for physically-based animation. In: Symposium on Computer Animation (2002)
14. Lien, J.M., Amato, N.M.: Approximate convex decomposition of polyhedra. In: Proc. of the ACM Symposium on Solid and Physical Modeling (SPM). Beijing, China (2007)
15. Mirtich, B.: Impulse-based dynamic simulation of rigid body systems. Ph.D. thesis, University of California, Berkeley (1996)
16. Nvidia Corporation: Physx. http://www.nvidia.com/object/nvidia_physx.html (accessed Oct 31, 2013)
17. Open dynamics engine. http://www.ode.org/ (accessed Oct 31, 2013)
18. Pan, J., Sucan, I., Chitta, S., Manocha, D.: Real-time collision detection and distance computation on point cloud sensor data. In: IEEE Int. Conf. Rob. Aut. (2013)
19. Redon, S., Kheddar, A., Coquillart, S.: Fast continuous collision detection between rigid bodies. Computer Graphics Forum **21**(3), 279–287 (2002)
20. Stewart, D., Trinkle, J.: An implicit time-stepping scheme for rigid body dynamics with inelastic collisions and coulomb friction. International Journal of Numerical Methods in Engineering **39**, 2673–2691 (1996)
21. Tang, M., Manocha, D., Otaduy, M.A., Tong, R.F.: Continuous penalty forces. In: ACM Transactions on Graphics (Proc ACM SIGGRAPH) (2012)
22. Usarsim. http://usarsim.sourceforge.net/ (accessed Oct 31, 2013)
23. Zhang, L., Kim, Y., Varadhan, G., Manocha, D.: Generalized penetration depth computation. Computer Aided Design **39**(8), 625–638 (2007)
24. Zhang, X., Lee, M., Kim, Y.J.: Interactive continuous collision detection for non-convex polyhedra. Vis. Comput. **22**(9), 749–760 (2006). DOI 10.1007/s00371-006-0060-0. URL http://dx.doi.org/10.1007/s00371-006-0060-0