Simulating Rigid-Bodies, Strings and Nets for Engineering Applications Using Gaming Industry Physics Simulators

J.H. de Jong*, K. Wormnes**, P. Tiso***

*Delft University of Technology, The Netherlands
e-mail: janhein.dejong@gmail.com

**Automation and Robotics Section, ESA
e-mail: kjetil.wormnes@esa.int

***Delft University of Technology, The Netherlands
e-mail: p.tiso@tudelft.nl

Abstract

The video game industry has been using and developing advanced rigid-body simulation tools, or *gaming engines*, over the past decades. Early research by ESA on simulating throw-nets has indicated that tools specifically developed for computer games can have significant benefits over traditional engineering tools in terms of speed, ease of use and flexibility. However, tools developed for computer games have less stringent constraints on accuracy and realism. The purpose of this research is to investigate the advantages and pitfalls of using gaming engines to perform engineering simulations.

It is found that gaming engines typically use a method called constraint-based simulation for simulating contact, together with a semi-implicit Euler integration scheme and a fixed time-step size. Care should be taken to select the right combination of mathematical methods when creating a simulator based on these models to prevent unphysical behavior. However, through computational and real-world experiments it is shown that constraint-based simulation can provide valuable simulations.

Finally it is shown that in terms of computational efficiency constraint-based simulators are moderately effective in simulating stiff systems, but very effective in simulating contact dynamics.

1 Introduction

Rigid-body simulation, a sub-class of multi-body simulation, studies the movement of systems of interconnected un-deformable bodies under the action of external forces. Established software packages such as SIMPACK and MSC/Adams allow engineers to visualize and study the motion of mechanical systems using rigid-body simulations. Examples in the aerospace industry where rigid-body simulations have been used are numerous and range from visualizing planetary rover behavior to studying separation dynamics for launch vehicles. Rigid-body simulations also have an increasingly important role to play in simulating orbital robotics.

The entertainment industry has also been using rigid-body simulation techniques to create interactive environments for video games. The software packages developed for these applications, usually referred to as *gaming engines* or *physics engines* are able to simulate a large number of bodies on a normal desktop computer in near real-time. The big difference between the software specifically developed for engineering purposes and gaming engines is that the latter only need to provide plausible simulation, and thus have been developed with less stringent constraints on accuracy in mind. Nevertheless, early research by the European Space Agency on throw nets for capturing in-orbit debris (Fig. 1) has shown that these packages could prove to be useful for engineering purposes in some specific cases, due to their ease of use, high computational efficiency and open-source nature. For more information on orbital debris and using throw-nets to capture in orbit debris, the reader is referred to earlier work by K. Wormnes[23].

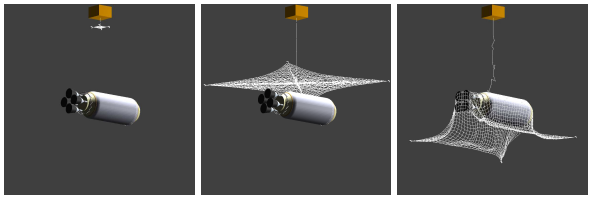


Figure 1. : A simulation of a throw-net capturing a burned-up rocket stage[23].

The main objective of this research is identifying the usability of gaming engines for engineering purposes. This can be split into two sub-objectives. First, we want to identify and present the mathematical models that are common in gaming engines and what their limitations are in terms of accuracy and stability. Second, we want to

identify and show the advantages of using gaming engines over traditional software.

2 Gaming engines

Engineering multi-body simulators typically have a lot of functionality (e.g. frequency-domain analyses, finite element modeling of compliant structures, optimization of cost functions). In the video games the problems that need to be solved are of a different nature. One can think of real-time simulations of rag dolls crashing into each other, a brick wall falling (Fig. 2), or cars crashing into each other. Gaming engines therefore have a strong focus on simulating contact dynamics of large numbers of rigid-bodies in real-time.

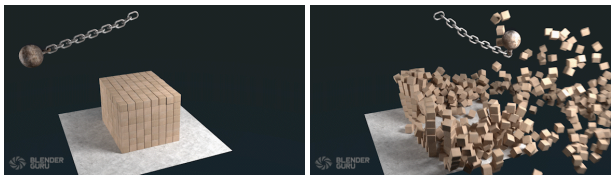


Figure 2. : A typical simulation performed with Blender & Bullet[18].

2.1 Popular gaming engines

Figure 3 shows the result of a survey performed by electronics company Nvidia on the adaptation of several well known gaming engines by the video game industry. Although possibly not very unbiased it shows what are considered to be the main players in the field. All the mentioned gaming engines are based on the so-called Stewart-Trinkle method[16, 6, 21, 10, 11].

Three out of five of these engines are backed by large budgets from electronics companies. PhysX is developed at Nvidia, Havok at Intel, and Bullet has recently been annexed by Advanced Micro Deviced (AMD).

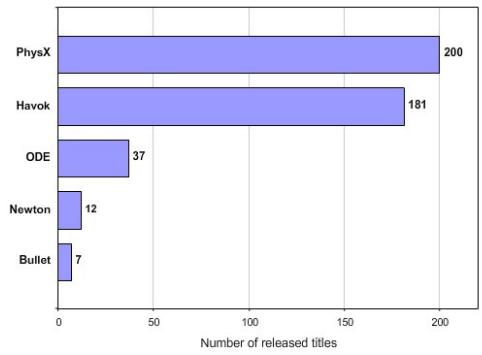


Figure 3. : Comparison of most used gaming engines by number of released commercial video games from 2006 to 2009[17].

Bullet is a strong candidate to use as a basis for exploring the possibilities of gaming engines. Its open-source nature the allows us to more easily define and analyze what steps the software takes during a simulation. Also, Bullet has a strong ongoing development, ensuring that it uses the most modern techniques developed in the rapidly evolving world of rigid-body simulation.

2.2 Current usage of gaming engines for engineering

Gaming engines have been used in engineering before, mainly in robotics. There exist several tools for rapid-prototyping of robotic systems, which typically house one of the gaming engines mentioned above[13, 7]. These tools allows the user to experiment with robot designs and algorithms in a virtual environment, eliminating issues involved with real-world experiments such as short battery life, hardware failures and unexpected and dangerous behavior. In some cases the simulator can run faster than real-life, further speeding up the process and enabling Monte-Carlo analyses to be performed. ESA also funded the development of a constraint-based testbed for simulating planetary rovers exploring the surface of Mars, called 3DRov[12].

3 Mathematical models

On a high level a rigid-body simulator can be divided in two main components[9]. The *collision detection* component uses the position and geometric data of bodies to detect contact and penetration and simplifies this to a number of contact points. The *simulator* component uses this information to compute contact forces and integrates the motion of the bodies.

The simulation component can consist of four modules. The time control module calls other modules and defines the integration time-step size. The motion solver uses all forces acting on the bodies to update the velocities and positions. The constraint solver computes all reaction forces acting on the bodies due to contact behavior or other constraints (e.g. joints, motors). In some cases a fourth module, the collision solver, is required to handle new contacts where the relative velocity is non-zero, or collisions. In other cases handling of collisions is included in the constraint solver.

3.1 Time-stepping

The time control module can have several methods for choosing the size of the time-step. The simplest method is to use a fixed time-step, where the size of the time-step is set at the beginning of a simulation and does not vary.

Adaptive time-stepping algorithms vary the size of the time-step during the simulation, depending on the stiffness of the system, or some tolerance on the relative error

of the state of the system. The advantage of these methods lies in the fact that the error can be controlled, and that the time-step size is only decreased when it needs to. The disadvantage of this method is that the computational time to simulate a certain period varies significantly and unpredictably.

Finally, another way to vary the time-step size is to ensure that all collisions occur exactly at a step, as opposed to in between two steps. This prevents penetrations between bodies, but increases computational expense.

3.2 Penalty based versus constraint based

Contact forces between rigid-bodies can be determined using to two different paradigms. The first method is to use a penalty function based on the level of penetration between the two bodies. Here the contact force $\boldsymbol{f}_c$ acting between two bodies is a function of the level of penetration $\delta\boldsymbol{s}$ and the respective relative velocity $\delta\dot{\boldsymbol{s}}$. Physically, this can be interpreted as adding a non-linear, unidirectional spring between the two bodies.

$$\boldsymbol{f}_c(\delta\boldsymbol{s}, \delta\dot{\boldsymbol{s}}) \in \mathbb{R}^+ \tag{1}$$

Although this method is simple to implement, it results in a model that suddenly increases in stiffness when a contact is present. In order to ensure stability the size of the time-step needs to be reduced significantly, increasing computational expense.

Another fundamentally different way of approaching the problem is constraint-based. In this approach, the reaction forces are defined such that they enforce a constraint on either the position or the velocity of the objects, preventing (further) penetration.

$$\boldsymbol{f}_c \in \mathbb{R}^+ \quad \text{such that} \quad \delta\boldsymbol{s} \geq 0 \vee \delta\dot{\boldsymbol{s}} \geq 0 \tag{2}$$

The disadvantage of this method is that a complex system of equations needs to be solved, and that either the position or velocity of the objects at the next time-step needs to be known explicitly in terms of the reaction forces. The advantage of this method is that the time-step size does not need to be reduced when there is an active contact.

In constraint-based simulation, elastic collisions are resolved through impulse-momentum laws[3], or more specifically through Newton's hypothesis, that relates relative velocity before ($\delta\dot{\boldsymbol{s}}^-$) and after the collision ($\delta\dot{\boldsymbol{s}}^+$) through a coefficient of restitution ϵ, such that

$$\delta\dot{\boldsymbol{s}}^+ = -\epsilon\delta\dot{\boldsymbol{s}}^-. \tag{3}$$

3.3 Stewart-Trinkle method

The Stewart-Trinkle method is the most popular constraint-based method to solve contact problems involving friction[22]. It was coined in the nineties and several adaptations have been developed over the past twenty years. It prevents penetration through inequality constraints such as Eq. 2, and describes friction through a linearized version of Coulomb's friction model.

Since the velocity at the next time-step needs to be explicit in terms of the reaction force, this method is often combined with the semi-implicit Euler integration method:

$$\boldsymbol{u}^{n+1} = \boldsymbol{u}^n + h\boldsymbol{M}^{-1}\left(\boldsymbol{f}_c + \boldsymbol{f}^n\right), \tag{4}$$

$$\boldsymbol{s}^{n+1} = \boldsymbol{s}^n + h\boldsymbol{S}(\boldsymbol{s})\boldsymbol{u}^{n+1}, \tag{5}$$

where $\boldsymbol{u}$ represents the velocity of the bodies, h the time-step size, $\boldsymbol{M}$ the generalized mass matrix and $\boldsymbol{f}^n$ all non-contact forces acting on the bodies. $\boldsymbol{S}$ represents a generalized kinematic map, mapping velocities to the change in positions.

Numerical errors arising from the integration of the constraints in the form of penetrations or joint drift are often stabilized through either *Baumgarte stabilization* or *post-stabilization*[4].

The resulting system is referred to as a complementarity problem[5]. The simplest form of a complementarity problem is a Linear Complementarity Problem (LCP), in which vectors **x** and **y** need to be defined such that

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}, \tag{6}$$

$$\boldsymbol{x} \geq 0 \quad \boldsymbol{y} \geq 0 \quad \boldsymbol{x}^T\boldsymbol{y} = 0. \tag{7}$$

Several types of solvers exist for these complementarity problems.

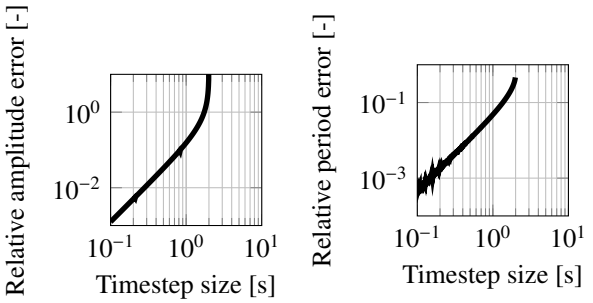
- **Direct methods.** Direct methods can provide very accurate solutions, but at a very high computational expense. Usually convergence time is in the order of $O(K^3)$, where K is the total number of contact points[14, 5, 2].
- **Iterative fixed point schemes.** These methods approximate a solution iteratively, until certain convergence criteria are met. Convergence time increases according to $O(K)$. These methods are fast in computing an initial approximation, but convergence decreases rapidly. Also, these solvers have more stringent convergence criteria. A well known iterative solver is the *Project Gauss-Seidel (PGS) method*[20].
- **Newton methods.** Newton methods also solve the system iteratively, achieving higher accuracy than PGS, but at a higher convergence time $O(K^2)$. A widely used solver is the PATH solver[15].

4 Unphysical behavior in Bullet & Blender

Based on the knowledge of the available mathematical models, the models used in Bullet were identified through documentation or comparison of outputs from Bullet & Blender with simple Matlab implementations of these models. Based on these identification we can identify the main sources of unphysical behavior. Please note that these analysis have been performed on the version of Bullet that is implemented in Blender v2.67. It should be noted that according to its development team newer versions might be addressing the presented issues.

4.1 Time-integration

Bullet uses the semi-implicit Euler integration scheme, as described in Eq. 4, combined with a fixed time-step size. This is a symplectic integrator, meaning that, unlike explicit or implicit Euler schemes, it more or less conserves energy over time[8].



(a) Amplitude error.

(b) Period error.

Figure 4. : Amplitude and period error of the time-integration of a harmonic oscillator relative to analytic amplitude and period. Note that $\omega = 1$ $[rad/s]$.

Examining the behavior of a one-dimensional harmonic oscillator with a natural frequency $\omega = \sqrt{k/m}$ shows three important regions with respect to the integration time-step size h. When $\omega \lesssim \frac{1}{h}$ the integration method conserves energy, and shows a $O(h^2)$ convergence in the amplitude error and the period error.

When $\frac{1}{h} < \omega < \frac{2}{h}$ the system is stable, but shows oscillatory behavior in the amplitude and energy of the system. Although the magnitude of this behavior remains the same over time, it increases exponentially when the time-step is approaching $\omega = \frac{2}{h}$.

Finally, when $\omega \geq \frac{2}{h}$ the system becomes fully unstable, and the energy keeps increasing over time.

4.2 Elastic collisions

Bullet uses a combination of post-stabilization and Baumgarte stabilization. Combining the latter with an implementation of Newton's hypothesis for elastic collisions (Eq. 3) results in energy being created during collisions. This behavior is pseudo random, meaning that some collisions create more energy than others, depending on the amount of penetration at the time-step where the collision is detected.

The inequality constraint including the restitution and stabilization terms is

$$\delta\dot{\boldsymbol{s}}^{n+1} \geq -\epsilon\delta\dot{\boldsymbol{s}}^{n} - \frac{k_{\text{erp}}}{h}\delta\boldsymbol{s}, \tag{8}$$

where k_{erp} is a stabilization parameter, indicating the level of stabilization that is applied. In Bullet $k_{\text{erp}} = 0.2$.

Because stabilizing term scales with $O(h^{-1})$, energy creation does not decay when $h \to 0$.

4.3 Friction

To comply with convergence criteria for the PGS algorithm, Bullet employs an even further simplification of Coulomb's friction than the linearized model present in the original Stewart-Trinkle method. In this decoupled problem the friction is split in two separate orthogonal problems in the contact plane[19]. Depending on the direction of the friction with respect to the orthogonal base of the model, the friction can be overestimated by as much as a factor of $\sqrt{2}$.

In Bullet the orthogonal base is aligned with the direction of relative motion at every time-step. This reduces this problem for sliding friction, but not for static friction.

4.4 Complementarity problem solution convergence & uniqueness

Bullet uses the PGS algorithm to solve the complementarity problem. Since PGS has a high initial efficiency, but slows down quickly, it is chosen to set a fixed number of iterations as a convergence criterion, rather than some tolerance on the error[9].

Furthermore, note should be taken that although the Stewart-Trinkle method is shown to always have a solution, uniqueness is not guaranteed[1]. This is inherent in the rigid-body assumption, since it can result in statically undetermined systems. An example of a system with multiple solutions is a table with four legs standing on a plane. A solution to the problem would be to have each leg carry one-fourth of the load. However, another solution would be to have two diagonally opposing legs to each carry half of the load. The solution space is spanned by infinitely many superpositions of these two solutions.

5 Experimental verification

It is the belief of the authors that the best way to show usability of simulation tools is to step out of the digital world and compare simulations to actual physical systems. Therefore a large part of this research has been allocated to experimental verification.

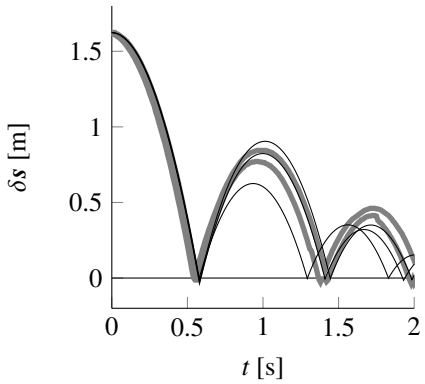


Figure 5. : Maximum and minimum boundaries of experimental data compared to simulated position of the tennis ball using Bullet & Blender for various time-step sizes. The thick grey lines depict the boundaries on the experiment, the thin lines the simulations.

5.1 Elastic collisions

To verify Bullet & Blenders usability for simulating elastic collisions, a simple bouncing tennis ball was used.

The tennis ball was dropped ten times from the same height, and recorded using a high-speed camera. The resulting video was analyzed using Matlab to extract positions of the ball. The boundaries of all resulting trajectories are plotted in Fig. 5 together with simulation results from Bullet & Blender. In the three simulations all physical properties are kept the same, with except for the time-step size.

Where we should see convergence, we actually see divergence due to the pseudo random energy creation in collisions. The spread of the simulations is much higher than that of the experiment.

5.2 String dynamics

A rigid-body simulator can simulate highly flexible bodies such as strings and nets through a lumped-mass approach, where the flexible body is modeled as a set of smaller rigid-bodies connected through spring-dampers. To verify the validity of this method in combination with the semi-implicit Euler integration scheme, the behavior of an elastic string was examined and compared to simulations.

As depicted in Fig. 6, the string was suspended vertically, excited manually at several points along the string and recorded with a high-speed camera. The resulting video was processed using Matlab to extract the transient and frequency behavior of the string. The material properties of the string (i.e. weight, dimensions, natural length, stretched length) were measured, and the Young's modulus was determined experimentally by suspending a known probe mass on a known piece of string and measuring the elongation.

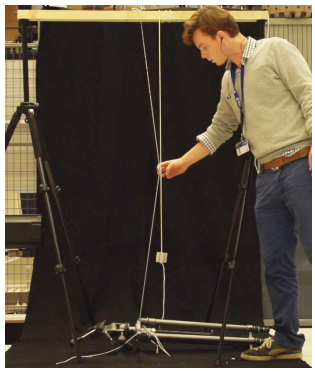


Figure 6. : Experiment setup for verification of string dynamics.

The lumped-mass model of the string consisted of circa thirty point-masses, connected with linear spring-dampers. The latter are limited to providing only tensile forces. Furthermore aerodynamic drag was added to the system in the horizontal direction.

The resulting comparison between the experiment and simulation is shown in Fig. 7. It can be seen that a lumped-mass model is quite successful in capturing all the observed oscillatory modes of the string.

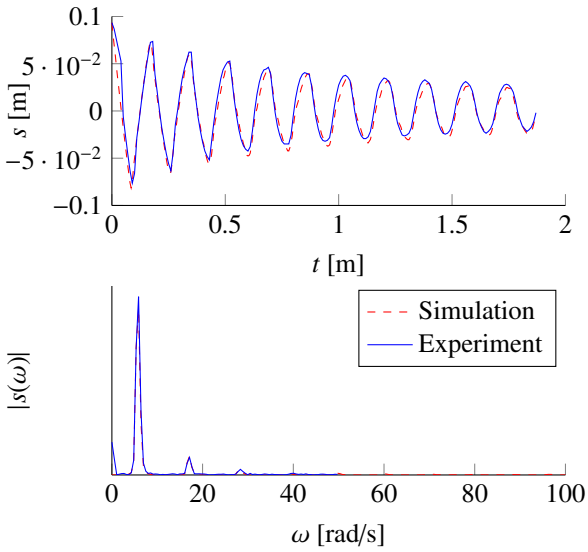


Figure 7. : Transient and frequency response of the middle point of the string.

When simulating an un-damped model of the string and trying to find the largest time-step for which the total energy in the system does not increase beyond some small tolerance, we find that it relates to the natural frequency of

one element ω_e, such that

$$h_{\max} \approx \frac{1}{\omega_e}. \tag{9}$$

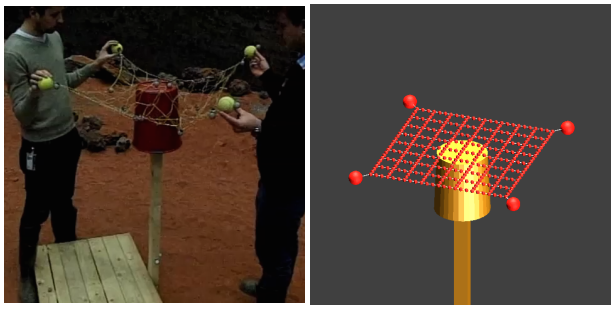
For damped systems the maximum time-step is slightly lower, depending on the level of damping present in the system.

The error of the simulation relates to the number of masses used to represent the string N according to $O(N^{-2})$. However, the computational time required to finish a stable simulation increases with $O(N^2)$.

5.3 Net dynamics

Simulation of contact dynamics and elastic systems are brought together in a net wrapping itself around an object. A simple net was constructed out of rubber bands and tennis balls, and a target out of a dustbin. This net was outfitted with markers for a 3D motion capture system to extract position information. Physical properties such as mass, lengths and the Young's modulus were measured in the same way as for the string experiment.

The model used for the simulations is a lumped-mass model based on earlier models created at ESA[23]. No aerodynamic drag was included in this model, merely contact dynamics, gravity and springs-dampers are acting on the objects.



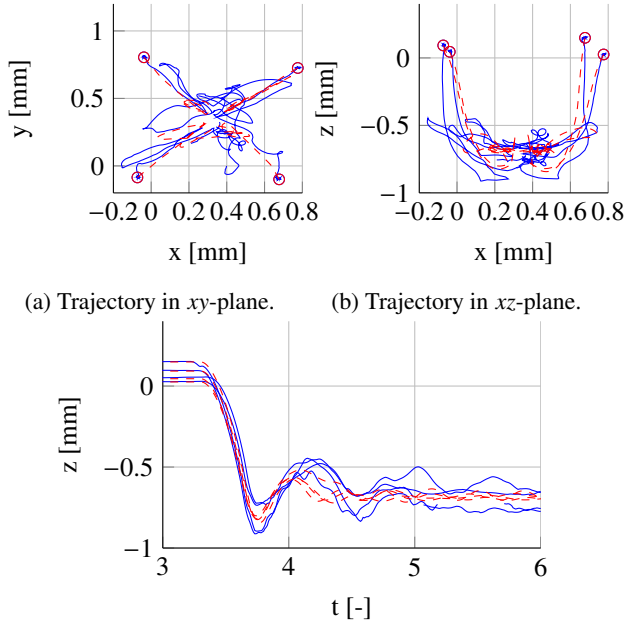
(a) Experiment setup. (b) Model in Blender.

Figure 8. : Net setup used for experiments and model in Blender.

Comparison between simulation and experimental data (Fig. 9) show promising results. The initial wrapping of the net is modeled quite accurately, especially considering the crude setup and simple model. After the corner masses have reached the center of the target, the system becomes chaotic and the simulation looses accuracy.

However, overlaying the trajectories of several simulations and experiments indicated that the movement of the corner masses describes the same area, indicating that Monte-Carlo analyses could be valid.

The simulations of the net and target were run on a simple desktop computer and took roughly four times the



(a) Trajectory in xy-plane. (b) Trajectory in xz-plane.

(c) z-position over time.

Figure 9. : Position of corner masses of net for experiment and simulation. Blue represents the experiment, red the simulation. xy represents the horizontal plane, z is the vertical direction.

simulated time to finish, depending on the stiffness and level of damping applied to the model.

6 Computational efficiency

To confirm the suspected speed advantage a constraint-based simulator has over a penalty based implementation, two types of models were simulated with Bullet & Blender and MSC/Adams and timed. The first model is the lumped-mass string model, the second a stack of several blocks collapsing on itself.

6.1 Stiff systems

For varying stiffness and damping one second of string motion was simulated and timed. The required computational times are plotted in Fig. 10.

It was found that un-damped systems in Bullet & Blender and MSC/Adams run with comparable computational speeds. However, for damped systems Bullet & Blender need more time to finish a stable simulation, whereas in MSC/Adams computational time is reduced. The reason for this could be that since MSC/Adams uses an adaptive time-stepping method, it reduces the time-step size once the higher modes in the tether have damped out, and hence increases its computational efficiency.

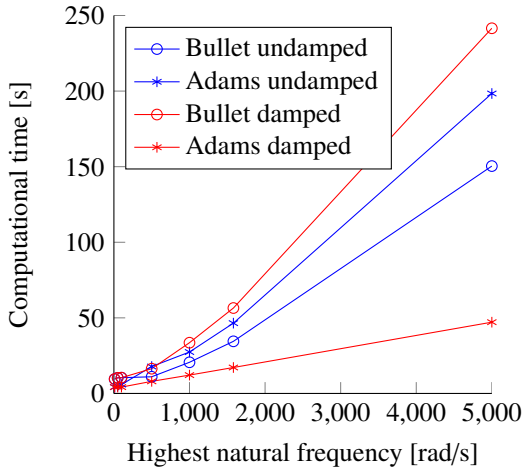


Figure 10. : Computational time required to perform stable simulation of string model for various stiffness values with Bullet & Blender and MSC/Adams.

6.2 Contact dynamics

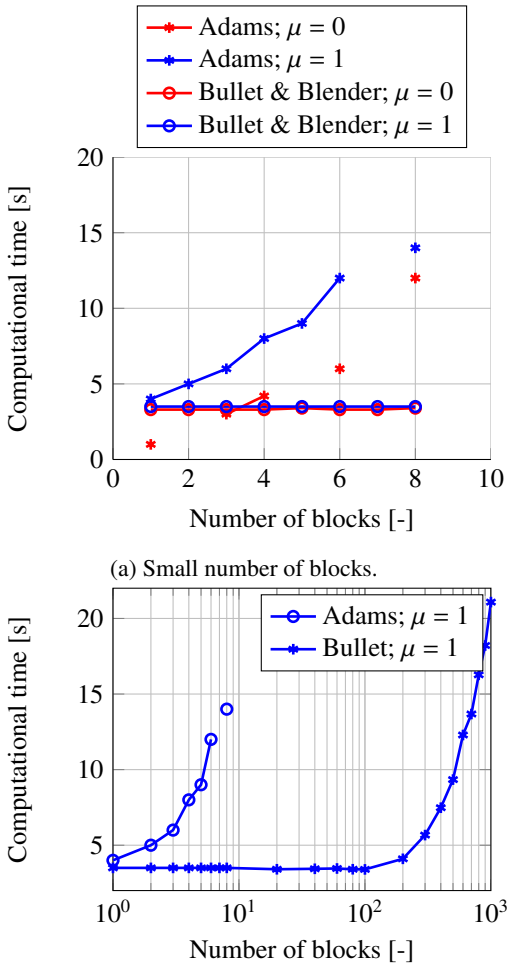
More interesting however is a comparison of a model that contains complex contact dynamics. For this purpose a simple model was created where a number of blocks was positioned above each other and dropped. Although numerous parameters affect the computational time required to simulate this model (e.g. type of mesh used for collision detection, friction model, visualization step-size), it was chosen to set these to a fixed value and increase the number of blocks that were stacked, and simulate 3.0 seconds.

The results, shown in Fig. 11a, indicate that run-time in Adams increases with the number of blocks, but in Bullet & Blender stays constant. This is due to the fact that Blender is set to run in real-time, meaning that if the simulation is faster than real-time, the output is merely slowed down. We can conclude that with this number of objects, the physics computation is not the bottleneck in the simulation.

In Fig. 11b we see that not before a hundred bodies simulated does the physics computation become the bottleneck in Bullet & Blender. We can conclude that Bullet is several orders of magnitude faster than Adams when it comes to computing contact dynamics.

7 Conclusions

The main distinction between a gaming industry rigid-body simulator and commercial engineering software is that it uses a constraint-based contact model instead of a penalty based model. Time integration is performed through a simple but effective semi-implicit Euler scheme combined with a fixed time-step size.



(a) Small number of blocks.

(b) Large number of blocks.

Figure 11. : Computational time required to perform stable simulation of stacking blocks, for various number of blocks. No entry means that Adams was unable to finish the simulation.

Depending on the right combination of mathematical algorithms, these methods can lead to unphysical behavior. Nonetheless, if setup properly, constraint-based simulation can provide accurate solutions to the simple physical models it represents.

Also it should be noted that gaming engines are constantly being developed in an open-source way, and are therefore developing and improving rapidly. According to the development team of Bullet, many of the problems with these simulations are being addressed in more recent versions of Bullet.

We have seen that Bullet & Blender are not as good in handling stiff, damped systems as Adams, but are much more effective in simulating contact behavior involving several bodies. A possible way to combine the best of both worlds could be to use constraint-based simulation

with adaptive time-stepping.

8 Acknowledgment

The author would like to thank the author of Bullet, Erwin Coumans, and Kenny Erleben for their support and the time they have taken to answer some very vital questions.

References

[1] M. Anitescu and F.A. Potra. Formulating dynamic multi-rigid-body contact prob- lems with friction as solvable linear complementary problems. *Nonlinear Dynamics*, 14:231–247, 1997.

[2] David Baraff. Fast contact force computation for nonpenetrating rigid bodies. *Computer Graphics*, 28:23–24, 1994.

[3] Jan Bender, Kenny Erleben, Jeff Trinkle, and Erwin Coumans. Interactive simulation of rigid body dynamics in computer graphics. In *State of the art reports*, 2012.

[4] Michael B. Cline and Dinesh K. Pai. Post-stabilization for rigid body simulation with contact and constraints. In *IEEE Intl. Conf. on Robotics & Automation*, 2003.

[5] Richard Cottle, Jong-Shi Pang, and Richard E. Stone. *The linear complementarity problem*. SIAM, 1992.

[6] Erwin Coumans. *Bullet 2.80 Physics SDK Manual*, 2012.

[7] Cyberbotics. Webots 7. http://www.cyberbotics.com/.

[8] Denis Donnelly and Edwin Rogers. Symplectic integrators: An introduction. *American Journal of Physics*, 73:938–945, 2005.

[9] Kenny Erleben. *Stable, Robust, and Versatile Multibody Dynamics Animation*. PhD thesis, University of Copenhagen, Denmark, 2005.

[10] Intel. Havok physics. http://www.havok.com/products/physics.

[11] Julio Jerez and Alain Suero. Newton game dynamics. http://www.newtondynamics.com/.

[12] K. Kapellos. Planetary exploration missions simulation using 3drov. In *Euromech colloquium on "Nonsmooth con- tact and impact laws in mechanics", Grenoble, France*, 2011.

[13] N. Koeng and J. Polo. Gazebo, 3d multiple robot simulator with dynamics. http://gazebosim.org/.

[14] C.E. Lemke. Bimatrix equilibrium points and mathematical programming. *Management Science*, 11:681–689, 1965.

[15] O.L. Mangasarian. Linear complementarity problems solvable by a single linear program. *Mathematical Programming*, 10:263–270, 1976.

[16] NVidia. Physx. www.physxinfo.com.

[17] NVidia. Popular physics engines comparison. http://physxinfo.com/, December 2009.

[18] Andrew Price. Introduction to rigid-body simulations. http://www.blenderguru.com/videos/quick-tutorial-make-a-wrecking-ball-with-rigid-body-physics/comment-page-1/, 2014.

[19] M. Silcowitz, S. Niebe, and K. Erleben. Nonsmooth newton method for fischer function reformulation of contact force problems for interactive rigid body simulation. *The Visual Computer*, 26:893–901, 2010.

[20] Morten Silcowitz, Sarah Niebe, and Kenny Erleben. Interactive rigid body dynamics using a projected gauss-seidel subspace minimization method. *Computer Vision, Imaging and Computer Graphics. Theory and applications*, pages 218–229, 2011.

[21] Russel Smith. Open dynamics engine. http://www.ode.org/.

[22] D.E. Stewart and J.C. Trinkle. An implicit time-stepping scheme for rigid body dynamics with inelastic collisions and coulomb friction. *International Journal of Numerical Methods in Engineering*, 1996.

[23] K. Wormnes, J.H. de Jong, and G. Visentin. Thrownets and tethers for robust space debris capture. In *64th International Astronautical Congress, Beijing, China*, 2013.