An Evaluation of Open Source Physics Engines for Use in Virtual Reality Assembly Simulations

Johannes Hummel[1], Robin Wolff[1], Tobias Stein[2], Andreas Gerndt[1], and Torsten Kuhlen[3]

[1] German Aerospace Center (DLR), Germany
{Johannes.Hummel,Robin.Wolff,Andreas.Gerndt}@dlr.de
[2] Otto-von-Guericke University, Germany
contact@tobias-stein.info
[3] Virtual Reality Group, RWTH Aachen University, Germany
kuhlen@vr.rwth-aachen.de

Abstract. We present a comparison of five freely available physics engines with specific focus on robotic assembly simulation in virtual reality (VR) environments. The aim was to evaluate the engines with generic settings and minimum parameter tweaking. Our benchmarks consider the minimum collision detection time for a large number of objects, restitution characteristics, as well as constraint reliability and body interpenetration. A further benchmark tests the simulation of a screw and nut mechanism made of rigid-bodies only, without any analytic approximation. Our results show large deviations across the tested engines and reveal benefits and disadvantages that help in selecting the appropriate physics engine for assembly simulations in VR.

1 Introduction

The simulation of the kinematic and dynamic behavior of multi body systems is an important aspect in many interactive simulation and training applications. In recent years, a large number of frameworks have evolved that intend to bundle the algorithms and data structures required for computing the physics effects in a library so that they can be reused in several projects. Such physics engines stem from academic, commercial and hobbyist efforts. Some try to provide a set of common features to support a wide range of applications, such as games and animation, while others specialize on very specific application areas, such as deformable tissue simulation in surgical training applications. Often, this comes with a trade-off between number of supported features, performance and accuracy. The large number of available physics engines makes it difficult for an application developer to select an engine that suits best for a given project.

The work presented in this paper was mainly driven by an ongoing project called VR-OOS (Virtual Reality for On-Orbit Servicing)[1]. Its goal is to develop an interactive application for the planning, training and analysis of on-orbit servicing tasks within a haptic enabled virtual reality environment used to train astronauts but also robot control. A fundamental feature is the real-time simulation of the realistic dynamic and kinematic behavior of satellite (and robot) components in various on-orbit servicing tasks. These include, for example, the removal of the protection foil (MLI - multi-layer insulation) from the satellite, operating lever switches, removing and inserting module boards using a bayonet handle, and handling connectors and locking mechanisms. As opposed to games, where it is usually sufficient that results look plausible, for this training simulator high accuracy at interactive frame rates is crucial.

The VR-OOS project currently uses Bullet Physics[1] as the physics engine. Bullet was chosen due to its popularity in related projects and its open source license. During preliminary evaluations of the integrated simulation environment, however, there have been observations that the results of the physics engine tend to get unstable in certain scenarios. For example, a switch vibrated during and long after moving the switch, or rigid modules occasionally penetrated their shaft when pushing them in. Such effects are undesired, as they make the simulation environment appear unrealistic and may influence the training effect. Tweaking the engine's parameters did lead to some improvements. However, it influenced stability in other scenarios. Consequently, we had to spend a considerable amount of time tweaking the parameters for every single mechanism for each occurring scenario. As scenarios are modified or combined to simulate a variety of assembly scenarios supporting various on-orbit servicing tasks this reduced the convenient utility of our simulation environment. Hence, we started an investigation to see whether there was an alternative physics engine that would fulfill our needs with generic parameters without scenario based, fine grained tweaking and set up a number of benchmarks specifically designed to test particular aspects of our application scenarios.

2 Related Work

Seugling et al. [2] evaluated three physics engines (Open Dynamics Engine, AGEIA NovodeX, Newton Game Dynamics) and created nine benchmarks to test the energy preservation, constraint handling and collision detection of the engines. Their energy preservation test consisted of two parts, pushing a sphere slightly into a box and measuring the resulting forces, and shooting a sphere against a box and measuring the resulting velocity and direction. They used a fixed restitution coefficient of 1.0, which resembles a perfect rebound of the objects. Constraint handling was tested by monitoring how a constraint set up as pendulum was reacting on increased weight, and whether the engines could handle more than 20 chained constraints combined to a long pendulum. However, it was not explored whether there was a correlation between these two tests. In order to investigate the scalability of collision contacts, they stacked up to 20 boxes on top of each other and measured the impact on computation performance of each engine. In practice, however, for an assembly simulation the physics engine usually has to handle more than 50 and more objects. Thus a

[1] Bullet Physics Engine: `http://bulletphysics.org`

scalability test should have been realized with at least 100 objects to challenge the collision detection of the physics engines.

A similar work by Boeing et al. [3] presented a qualitative evaluation of freely available physics engines. With seven physics engines (Open Dynamics Engine, AGEIA NovodeX, Newton Game Dynamics, Tokamak, True Axis, Bullet Physics and JigLib) they ran several tests measuring the integrator performance, the material properties, the constraint stability and the performance of collision detection. Integrator performance was measured based on accuracy only. The speed was not measured. For measuring the stability of the constraints they used a chain of spheres connected to each other. They mounted both sides of the chain to fixed points. They also varied the number of the spheres in the chain. The behavior of the engines with different weights of the spheres was not tested. Collision detection performance was measured by stacking about 20 boxes, too. Both papers provided valuable information on the generic performance of the then popular physics engines. Nevertheless, we needed an evaluation of up-to-date engines and with more focus on our specific requirements.

3 Benchmarks

Our overall requirements on the physics engine could be divided into five aspects. Firstly, the physics engine had to be able to detect collisions between objects of relatively large numbers (approx. 250 to 500 objects). The detection of collisions is a fundamental part of physics simulation. This had to be performed at interactive frame rates - with today's common display refresh rates of 60Hz, this means within approx. 16ms, which even goes to 1kHz or 1ms respectively when integrating haptic feedback. Secondly, the computed responses of dynamic objects to collisions had to result in realistic behavior. For example, a rigid object colliding with a hard surface should bounce back with high velocity, while when colliding with a soft surface should bounce with little velocity only. Thirdly, constraints, which limit the degree of freedom of movement of an object with regard to another object, should obey their defined limits and not diverge from them. For example, a switch should only rotate around its single rotation axis and not any other axis. Neither should it move along any translational axis when pushed from the side. Furthermore, rigid bodies should not inter-penetrate each other. When a satellite module is pushed into its module shaft, for example, it should only move along the contact plane within the shaft, but not move through the walls or get pushed through the end. Finally, we needed the physics engine to support the realistic simulation of a screw mechanism. This required the stable computation of both collision and friction for complex geometric objects.

This section outlines the design of a number of benchmarks intended to test the five requirements mentioned above. The first three are broadly based on benchmarks in related work and test the performance and accuracy of more generic features. The benchmarks four to six were designed to test particular features that are specific to our on-orbit servicing simulation and training application.

3.1 Collision Computation Performance

The first benchmark was designed to test the performance and scalability of collision detection with a large number of rigid objects. It was closely based on those tests in related work and consisted of a simple physics scene with a specified amount of equal objects (1 to 2500) placed in a three-dimensional grid along x, y, z axes floating above the ground, see Fig. 1. In contrast to related work, we used spheres instead of boxes, as these are the simplest three-dimensional collision shape, and showed us the minimum time required to detect collisions. Each sphere had a radius of 0.5m and a weight of 1kg. Gravity in this benchmark was defined as $9.81\frac{m}{s^2}$. We measured the time it took to detect the collisions, compute respective collision responses and advance to the next simulation step.

3.2 Accuracy of Collision Response

In order to test the accuracy of collision responses calculated by the physics engines, we measured the height of a bouncing ball with a defined coefficient of restitution. For the benchmark, we created a sphere with the material properties of a tennis ball (static friction coefficient 0.29, dynamic friction coefficient 0.22, coefficient of restitution 0.636) [4,5]. The sphere had a radius of 6.75cm and a weight of 57g. Gravity was set to $9.81\frac{m}{s^2}$. The sphere was dropped from height H on the floor, see Fig. 2. Over several time steps the heights h' of bounces were measured and compared to the expected heights.

3.3 Fixed Constraint Stability

In order to test the stability of constraints, we created two separate benchmarks. For testing fixed constraints, our benchmark defined a long chain of spheres with radius of 0.5m, see Fig. 3. Each sphere was attached to its neighbor by a fixed constraint between the centers of each sphere at a distance of the double radius so that they touched each other on the surface. The constraints limited the movement around all translation and rotation axes to zero. The outer spheres on the left and right were anchor-points where the chain was connected to horizontally. These were set as static objects, so that their position was not modified by the physics engine during the simulation, but consider collisions and constraints to other objects. When applying gravity of $9.81\frac{m}{s^2}$, the chain was expected to stay in its initial position. In the benchmark, we measured the deviation from the starting position. The test was repeated with different chain lengths up to 20 (without counting the anchor-points) and with increasing weights up to 1000kg per sphere.

3.4 1-DOF Constraint Stability

One issue in our current simulation was that simulated switches often showed unrealistic behavior. The lever trigger vibrated a lot while being moved and after snapping over to the new resting position. In order to test how other physics

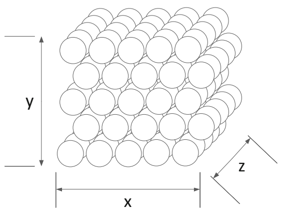

Fig. 1. Setup of the collision computation performance benchmark

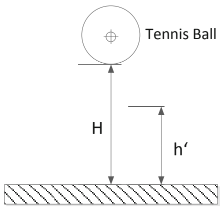

Fig. 2. Setup of the collision response accuracy benchmark

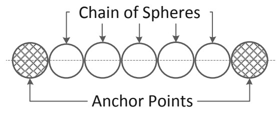

Fig. 3. Setup of the fixed constraint stability benchmark

engines would handle such a scenario, the fourth benchmark recreated the switch mechanism. It consisted of two simple rigid bodies, a base and a trigger, see Fig. 4. Both were connected using a hinge constraint between the lower end of the trigger and centered above the base, high enough to avoid both bodies touching each other. A hinge is a 1-DOF constraint allowing rotation along the z axis (pointing out of the image) in the x,y plane. The constraint has been further limited to only rotate between angles of α=±45°. In its resting position, the trigger would be at α=-45°, pushed down by constant force F_2. A third rigid body, the slider, would move slowly toward the trigger and push it with constant force F_1 to toggle the switch. When the trigger passes α=0°, the force F_2 would be inverted to $F_2^{'}$ pushing the trigger to the opposite resting position at α=+45°. The expected behavior was that the trigger flips to the opposite side and rests at +45°, without bouncing back or penetrating the base or slider bodies. In the benchmark, we measured the translation along each axis and rotation angles around each axis. Gravity was set to zero in this benchmark, as we wanted to simulate a space environment.

3.5 Interpenetration

Another issue in our current simulation was that grasped rigid objects occasionally penetrated other rigid objects. This was especially apparent when interacting with the environment without haptic feedback. Thus, we designed a benchmark resembling the module and shaft scenario. The setup consisted of one dynamic rigid body, the module, surrounded by four static rigid bodies, that held it in place, resembling the shaft. The module had the dimensions of 0.5m x 2m x 1m and a weight of 1kg. The bodies of the shaft were large enough to closely surround the module. The module was slightly taller than the shaft, so that it sticks out at the top, see Fig. 5. A constant force F was applied to another rigid object, the slider, so that it moved towards the module and pushed it. The expected behavior was that the module did not move or rotate. We measured changes in the position and rotation of the module in its center point. Additionally, we repeated the test in several iterations with increased F.

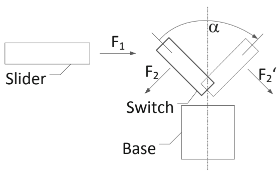

Fig. 4. Setup of the 1-DOF constraint benchmark

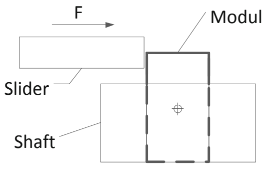

Fig. 5. Setup of the rigid body interpenetration benchmark

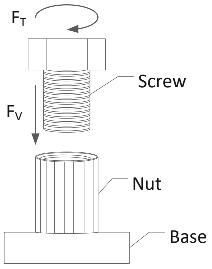

Fig. 6. Setup of the advanced collision and friction benchmark

3.6 Advanced Collision and Friction

A common scenario in many on-orbit servicing tasks is to screw and unscrew certain components. The simulation of a screw mechanism could be accomplished with a 2-DOF constraint allowing both translation and rotation along one single axis only and using a function that makes the translation dependent on the rotation. This, however, must be scripted in the application layer, as common physics engines do not support a screw mechanism. Additionally, the interactions between screw, thread and other involved rigid bodies have to be scripted too, as the 2-DOF constraint must be attached dynamically to a screw and thread as soon as they collide, but only if their axes are aligned and either screw or thread rotates in the correct direction.

Although it was done for the switch scenario described above, we wanted to avoid scripted behavior and rather solve this problem with rigid bodies and standard constraints only. Simulating the loosening and tightening of a screw using rigid bodies, however, puts a high challenge on a physics engine, as it involves a large number of colliding faces between screw and screw thread. A screw has a complex geometry. The thread is a concave object, which is computationally expensive during collision detection. Therefore, we approximated the geometry using a compound collision shape made up of several convex collision shapes arranged in a spiral. Fig. 6 shows the general setup of the benchmark.

The collision shapes of the screw spiral (bolt) were placed around a cylinder, while the collision shapes of the thread spiral (nut) were placed inside a number of tall collision shapes of 1m, arranged in a circle with radius of 0.5m to form a hollow cylinder as fixed base. We used segments of 30 box collision shapes with 5cm height per turn of the spiral. The static and dynamic friction coefficient was set to 0.001 for each physics engine in the test. Gravity was set to zero.

A constant torque F_T of 1Nm and a vertical force F_V of 1N, were applied every simulation step. The expected behavior was a rotation of the the screw with increasing speed and a continuous subtle movement into the nut without any interruptions and penetrations. The benchmark tested the stability of the computed collision and friction for complex compound objects and could be transferred to other scenarios, such as a bayonet mechanism.

4 Test Environment

Similar to Seugling et al. [2] we did a pre-selection of freely available physics engines, in which we collected information, such as supported platforms, used licenses, types of supported rigid body collision shapes and constraints, and then chose those physics engines that seemed most suitable for our needs. We finally selected Bullet Physics, Newton Game Dynamics, Havok Physics, Open Dynamics Engine (ODE) and NVIDIA PhysX (formerly Novodex), as they share a common set of required features and are widely used.

Bullet is the physics engine in our current simulation environment and was used as reference. It is distributed under the zlib license, runs on all major platforms and supports various optimized collision shapes. Many games and some movies are using Bullet making it one of the most popular open source physics libraries. We used version 2.80 of Bullet, which was released on March 5th, 2012. Newton[2] recently switched to the open source zlib license and provides the common set of features for rigid body simulation. However, instead of using linear complementarity problem or iterative methods, Newton uses a deterministic approach in its solver, promising very accurate results. We used version 2.33, which was released on April 1st, 2011. Havok[3] is a commercial physics engine with its own end-user license agreement (EULA) and promises robust dynamics and constraint solving. It is used in more than 150 game releases. We used version 7.1 of Havok, which was released on December 22th, 2009. ODE[4] is available under the open source LGPL/BSD license. Since its introduction in 2001, ODE has gained great popularity for hobbyist robotics simulation. We used version 0.12 released on February 11, 2012. PhysX[5] is now developed by NVIDIA and distributed as closed source under the terms of its own EULA. Used in approx. 200 game releases, similar as Havok, it is widely adopted by the games industry. We used the stable release 9.12.0213, which was published on March 13, 2012.

In order to reduce the programming overhead and the possibility of errors while implementing the scenarios separately for each engine, as well as to easily iterate several test runs with varying parameters and collect data, we desired a method to decouple the high-level benchmark implementation from the low-level execution in the underlying engine without compromising any features. A few frameworks exist, such as Physics Abstraction Layer (PAL)[6], *Open Physics Abstraction Layer (OPAL)*[7] and *The Gangsta Wrapper (TGW)*[8], that provide an abstraction layer and bindings for multiple physics engines, enabling developers to easily exchange physics engines at run-time without reinitializing the entire simulation. However, they have partly deprecated bindings or limited support for our chosen physics engines. Hence, we implemented our own physics

[2] Newton Game Dynamics: `http://www.newtondynamics.com`
[3] Havok Physics: `http://www.havok.com`
[4] ODE: `http://www.ode.org`
[5] PhysX: `http://developer.nvidia.com/physx`
[6] PAL: `http://www.adrianboeing.com/pal`
[7] OPAL: `http://thatopal.sourceforge.net`
[8] TGW: `http://gangsta.sourceforge.net`

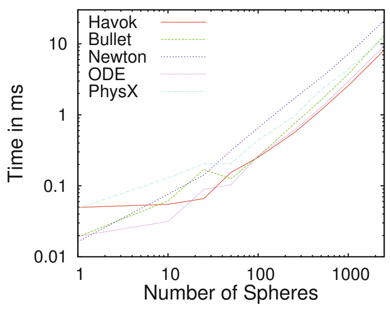

Fig. 7. Mean computation times in ms for collision detection with 1 to 2500 spheres

Table 1. Mean computation times in ms for collision detection with 1 to 2500 spheres

Spheres	Bullet	Havok	Newton	ODE	PhysX
1	0.02	0.05	0.02	0.02	0.05
10	0.06	0.05	0.08	0.03	0.13
25	0.17	0.07	0.14	0.09	0.20
50	0.13	0.15	0.31	0.10	0.20
100	0.26	0.25	0.66	0.26	0.44
250	0.74	0.56	1.69	0.61	0.95
500	1.62	1.17	3.34	1.27	2.26
1000	3.75	2.58	7.13	2.96	4.38
2500	13.17	8.13	21.07	9.39	12.21

abstraction layer framework. It consists of a number of C++ classes that encapsulate repeatedly used interface and utility methods for creating, running, controlling and monitoring a benchmark.

The benchmarks were performed on a workstation with an Intel i7-870 CPU with 2.93GHz, 8GB RAM, Nvidia GeForce GTX 460, running Windows 7 Enterprise 64 Bit (Service Pack 1).

5 Results

5.1 Collision Computation Performance

Fig. 7 shows a graph of the average time for computing collisions with 1 to 2500 spheres, and Table 1 shows the mean values. The computation times had slight deviations with few collision objects and continued to increase almost logarithmically with 50 objects and more. Most plots bend at between 20 and 50 objects, suggesting that these areas were influenced from the engines' internal optimizations. All engines were able to compute collisions of 100 spheres in under 1 ms and all, except Newton, could handle up to 250 spheres in under 1 ms. In overall, all engines performed equally well, with Newton lagging slightly behind.

5.2 Accuracy of Collision Response

In the accuracy of collision response benchmark, we measured the height of a simulated bouncing tennis ball. Table 2 shows the heights produced by the different physics engines compared to the expected height. Seven rebounds were recorded. Apart from Bullet, all physics engines computed bounces very close to the expected value with less than 1cm difference. Havok and PhysX simulated only the first three bounces and then set the rigid body to sleep mode to save computation time, which is a normal optimization method, but makes the simulation appear unrealistic. Bullet appeared to damp the bounces more than other engines, although the restitution coefficients have been set correctly for both the ball and the floor.

5.3 Fixed Constraint Stability

This benchmark tested the stability of fixed constraints in chains with varying numbers of spheres with 1kg and 1000kg. Fig. 8 shows the maximum euclidean distance between actual and start position of the 1kg spheres under the effects of gravity. One can see that the constraint stability of Havok and ODE decreased exponentially with increasing numbers of spheres. In contrast, Bullet's constraints appeared somewhat soft and let spheres move down by more than 30cm. Newton and PhysX performed best in this benchmark. There were very little to no deviations in sphere positions. Neither the number of spheres nor the increased mass had significant impact on this result. However, with chains of 1000kg spheres (not shown in the graph), the constraint stability in Newton decreased proportionally towards that of ODE and Havok. The other physics engines did not appear to be affected by very large masses.

5.4 1-DOF Constraint Stability

Fig. 9 shows the difference in the angle around the z-axis over time, at which the trigger object rotated to the specified limit of the constraint from the moment it has reached its end position at 45°. It was expected that all physics engines leave the trigger at its final position, and thus show a line around 0 degrees. However, only PhysX and Newton approached zero after shortly exceeding the constraint limit with approx. 2°. Newton performed best in this benchmark. The constraint limit was exceeded by 1.87° for 0.68 ms, and converged to a distance of 0.07°. PhysX caused spikes of 1.6° every 0.4 ms. A closer look into the data showed, that these occurred each next calculation step when the angle was exactly 45°. ODE and Bullet both started to vibrate after the trigger bounced back from its end position, suggesting a high default restitution value. Bullet's vibration appeared stable with a frequency of about 18.5 Hz and an amplitude of 0.86°, mostly below the constraint limit. The amplitude of ODE's vibration was about five times the size of that of Bullet and did not appear to have distinct frequencies in it. The plot for Havok shows that the force F_2' was stronger than the constraint limit and pushed the trigger beyond with up to 16°.

5.5 Interpenetration

Table 3 shows the maximum deviation in translation along the x and y axes and rotation around the z axis of the module object, resting in its module shaft, when being pushed at the top end by the slider object along the positive x axis with varying force from 25N to 500N. Looking at the horizontal translation along the x axis only, Bullet appeared to perform best. The module moved very little, less than a millimeter at forces up to 100N. The same applied to PhysX, with only slightly larger values. However, the module appeared to be moved along the y axis with Bullet. Also, the rotation around the z axis in Bullet was with up to 10° significantly larger than that of PhysX. Havok, ODE and Newton showed a similar linear increase of deviation from the resting position with increasing

Table 2. Comparison of computed heights of bounces of a tennis ball dropped from 2.54m as reported by the physics engines (All values in meter.)

# Bounce	Expected	Bullet	PhysX	Havok	ODE	Newton
1	1.027	0.567	1.067	1.007	1.018	1.084
2	0.416	0.145	0.493	0.423	0.384	0.441
3	0.168	0.035	0.237	0.210	0.157	0.195
4	0.068	0.010	0.133	0.132	0.061	0.080
5	0.028	0.003	0	0	0.026	0.029
6	0.011	0.002	0	0	0.011	0.008
7	0.005	0	0	0	0.005	0.001

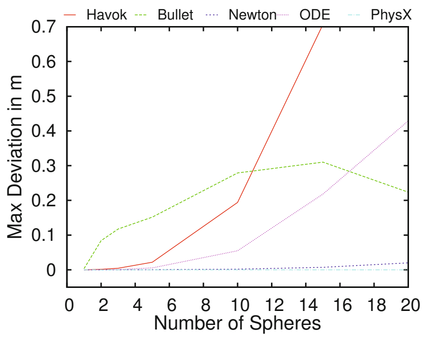

Fig. 8. Maximum deviation in horizontal position in chains of 1kg spheres in the fixed constraint stability benchmark

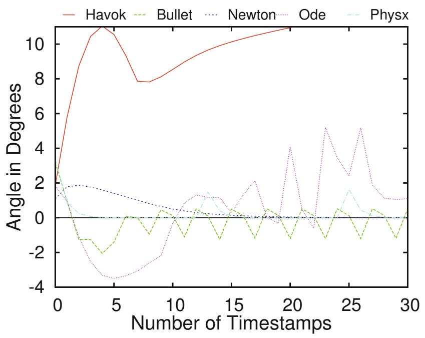

Fig. 9. Difference of actual angle around the z-axis to the specified limit in the 1-DOF constraint stability benchmark

forces. Both, the x and y translations were approx. 5x, 15x and 20x, respectively, more than that of PhysX. With ODE, however, the module started to get pushed out of the module shaft, with movements of more than 20cm along y. The module rotated slightly with 1° to 2° at forces up to 100N. At 250N and 500N, Bullet showed rotations about double of that with the other engines. However, PhysX appeared very stable and always showed rotations less than 1°.

5.6 Advanced Collision and Friction

In the advanced collision and friction benchmark, we measured the position and orientation of the head of the screw when screwed into the nut. Fig. 10 shows the translation over time along the y axis, which corresponded to the height of the screw. One can see, that Havok did not manage to solve the task. After about 12% of the total length it jumped out of the nut and fell down. Although ODE performed well half way, it also did not manage to complete the task. The screw got stuck in the long compound object resembling the nut and did not move anymore. This behavior was reproduced each time we ran the test with ODE. Bullet and Newton passed the test very well. While Newton showed a more continuous movement, with one interruption at about one third of the height, Bullet showed more occasional interruptions. This was presumably

Table 3. Maximum deviations with different forces applied to the slider for translation in x & y direction (in m) and the rotation on the z axis (in deg)

Force	Bullet			PhysX			Havok			ODE			Newton		
in N	X	Y	α	X	Y	α	X	Y	α	X	Y	α	X	Y	α
25	0.02	0.07	0.42	0.02	0.00	0.029	0.13	0.20	0.29	0.32	0.04	0.18	0.46	0.13	0.35
50	0.07	0.27	0.93	0.04	0.01	0.061	0.25	0.34	0.59	0.67	0.33	0.92	0.89	0.05	0.71
100	0.07	0.68	1.95	0.09	0.02	0.126	0.46	0.45	0.99	1.24	0.31	0.90	1.82	0.38	1.49
250	0.11	2.21	5.13	0.28	0.07	0.379	1.15	1.13	2.48	2.91	2.14	2.57	4.79	1.77	3.72
500	0.45	4.49	10.45	0.63	0.16	0.845	2.29	2.27	4.98	5.78	23.80	5.34	9.85	4.20	7.69

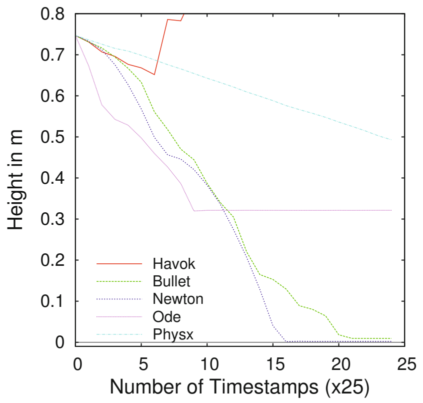

Fig. 10. Height of the head of the screw (in meter)

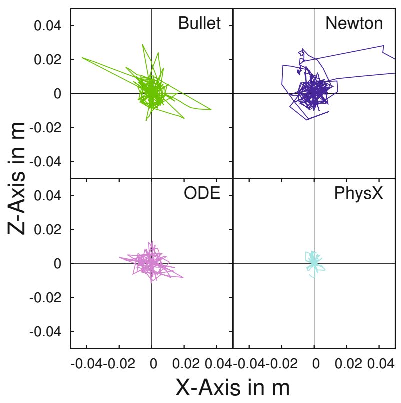

Fig. 11. Movement of the head of the screw on the x-z plane (in meter)

due to increasing collisions while moving into the nut. Compared to the other engines, PhysX showed a very slow but linear motion. It suggests that the default damping setting was very large. Due to the slow movement, PhysX could not manage to complete the task within our given 1000 timestamps.

Fig. 11 shows the translation in meter of the head of the screw on the x-z plane (looking from top at the screw). The expected behavior was that it would not show any movement to the sides. We did not plot Havok, as it could not complete the task. In the graph of PhysX, one can see the most subtle movement of less than 10cm (remember the screw was 1x0.5m). The graphs of ODE, Bullet and Newton show average translations of similar distances. However, Bullet and Newton showed occasional longer deviations, presumably when collisions occurred.

6 Conclusion and Future Work

As Boeing [3] mentioned, there is no general physics engine that performs best for any given task. Each engine we tested showed strengths and weaknesses. PhysX showed the most stable constraints and little inter-penetrations, presumably due to its high damping default settings. However, the high damping hindered the calculation of realistic bounces and advanced friction, as in the screw benchmark, leading to unrealistic behavior. Havok was highly optimized for speed. But this came at the expense of less accuracy in most of our benchmarks. Bullet performed poorly in the restitution test and tended to vibrate in the 1-DOF constraint stability test, which confirmed our observations within our target simulation environment. However, Bullet showed reliable constraints and performed very well in the advanced collision and friction benchmark simulating screw. Newton's approach of the deterministic solver appeared to produce indeed most accurate results. Apart from the collision computation performance benchmark, Newton performed always very well. ODE was fast and accurate, as long as its constraint and collision handling was not stressed too much. In this case, it reacted unpredictable.

Our benchmarks were designed with assembly simulations in mind. They test the performance of collision computations, preservation of energy, constraint reliability, inter-penetration, and evaluate the computation of both collision and friction for complex compound objects simulating a screw mechanism. The first three tests were mainly standard tests, seen in related work too. We added three novel benchmarks that evaluated the constraint reliability for a switch mechanism and the penetration between rigid bodies when applying a large force. In a more complex benchmark, we evaluated how the physics engines could handle a screw mechanism. Based on our requirements and the results of the benchmarks, we can now say that Newton and PhysX would be valuable candidates to compete with Bullet for integration in our current simulation environment.

As our test environment decouples the benchmark implementation from the actual underlying physics engine, it can be easily extended to evaluate and compare further physics engines as well. As described in section 4, we only selected the five most common engines. Other engines, such as OpenTissue Library, SOFA, Tokamak, Dynamechs, True Axis, or CM-labs Vortex, could be evaluated too. Furthermore, we consider extending our suite of benchmarks to also test the performance of mesh-mesh collisions with concave geometries, as well as handling soft body dynamics.

References

1. Wolff, R., Preusche, C., Gerndt, A.: A modular architecture for an interactive real-time simulation and trainning enviroment for satellite on-orbit servicing. In: Proceedings of IEEE Distributed Simulation and Real-Time Applications, pp. 72–80 (2011)
2. Seugling, A., Rölin, M.: Evaluation of physics engines and implementation of a physics module in a 3d-authoring tool. Master's thesis, Umea University Department of Computing Science Sweden (2006)
3. Boeing, A., Bräunl, T.: Evaluation of real-time physics simulation systems. In: Proceedings of the 5th International Conference on Computer Graphics and Interactive Techniques in Australia and Southeast Asia (GRAPHITE), pp. 281–288 (2007)
4. Federation, I.T.: Itf approved tennis balls, classified surfaces & recognised courts 2011 - a guide to products and test methods (2011)
5. Farkas, N., Ramsier, R.D.: Measurement of coefficient of restitution made easy. Physics Education 41, 73–75 (2006)