**Proceedings of SIMS 2004**
**Copenhagen, Denmark**
**September 23–24, 2004**

# CONTACT GRAPHS IN MULTIBODY DYNAMICS SIMULATION

**Kenny Erleben*** **and Henrik Dohlmann**
**University of Copenhagen**
**Department of Computer Science**
**Denmark**

## Abstract

In rigid body simulation contact graphs are used detecting contact groups. Contact graphs provide an efficient underlying data structure for keeping information about the entire configuration and in this paper we extend their usage to a new collision detection phase termed "Spatial-Temporal Coherence Analysis". This paper will review contact graphs and demonstrate the performance impact in a typical constraint based multibody simulator.

*Keywords:* Multibody Dynamics, Contact Graphs, Contact Groups, Contact Analysis, High Level Control

## Introduction

Historically contact graphs are used for splitting objects into disjoint groups that can be simulated independently. Contact graphs are frequently mentioned between people working with rigid body simulation, but they are often not formally described in the literature. In [1] the idea of using contact groups to break down contact force computations is mentioned. The benefit is obvious and not many people would spend time on explaining it. To our knowledge the first use of the word "graph" appears in [2], where a contact graph is used to properly back-up penetrating objects. In our opinion [3] is the first advanced attempt on using contact groups for distributed simulation. Recently [4] developed a shock propagation algorithm for efficient handling of stacked objects, which uses a contact graph. Today simulators do exploit contact groups for breaking down the computations into smaller independent problems as in Open Dynamics Engine (ODE) (v0.035), however they do not store a graph data structure.

Alternatives to contact graphs are not very surprisingly neither mentioned or talked about. An alternative is to put the contact-matrix into block-form [5]. In comparison with the contact graph approach the "block-form" matrix approach is limited to contact force and collision impulse computations and can not be used for anything else.

*Corresponding author. Phone: +45 3532 1413, Fax: +45 3532 1401, E-mail: kenny@diku.dk

The contact graph algorithm we present in this paper is part of the Spatial-Temporal Coherence (STC) analysis module. The algorithm clearly shows that STC analysis is scattered around the other phases in the collision detection engine. We use contact graphs for caching information, such as contact points. The cached information can be used for to improve run time performance of a rigid body simulator. Several speedup methods are presented, these fall into two categories, the first is real speed-ups due to improvements of simulation algorithms, the second is due to changes of the properties of the mechanical system, which alters the physical system, but still produces plausible results. Our main focus is computer animation and not accurate physical simulation.

## The Contact Graph

A contact graph consists of a set of nodes, a node is an entity in the configuration, such as a rigid body or a fixed body. A node can also be a virtual entity, such as a trigger volume, generating an event notification when other objects penetrates it.

When objects interact with each other, contact information are computed and cached, it is easy to use the edges in the contact graph for storing information of interactions between objects. Edges are also useful for keeping structural and proximity information about objects.

The most important thing is that both nodes and edges should be accessible in constant time and edges are bidirectional and uniquely determined by the two nodes they run between.

These properties can be obtained by letting every entity in the configuration have an unique index, and letting edges refer to these indices, such that the smallest indexed entity is always known as *A* and the other as *B*.

An edge between a physical object and a trigger volume indicates that the physical object has moved inside the trigger volume. This kind of edges can therefore be used to generate trigger volume event notifications. This type of edge is a dynamic edge, meaning that it is inserted and removed dynamically by the collision detection engine during the simulation.

The last type of edges we can encounter are those which tell us something about how the objects in the configuration currently interact with each other. For instance if two rigid bodies come into contact then an edge is created between them. There are some combinations of edges which do not make sense. For instance an edge between two fixed bodies.

More node types and edge types are described in [6].

## The Contact Graph Algorithm

We will now outline how a contact graph can be used in the collision detection pipeline. Notice that although we claim a contact graph to be a higher order contact analysis phase it is not a phase that is isolated to a single place in the pipeline, instead it is spread out around all the other phases, i.e. in between the broad phase module responsible for finding close object pairs, the narrow phase module that determines overlap status for an object pair, and the contact determination module that computes all contact points between an object pair. In the following we will walk through what happens in the collision detection pipeline step by step.

The first step in our algorithm is to update the edges in the contact graph. This is done by looking at the results of the broad phase collision detection algorithm. The results of the broad phase collision detection algorithm are really an unsorted list of pairs of nodes, where each pair denotes a detected overlap in the broad phase algorithm. Observe that each pair is equivalent to a contact graph edge. We can therefore insert new edges into the contact graph, which we have not seen before. At the same time we can handle all close proximity information, that is detection of vanished, persistent and new close proximity contacts. This is done by comparing the state of edges with their old state.

Now we can do logical testing and exploit caching, by scanning through all the reported overlaps and remove those overlaps we do not have or want to treat any further.

Overlaps with passive objects are also removed, passive objects do not really exist in the configuration, they are merely objects kept in memory in case they should be turned active later on. This way objects can be preallocated and further more there is no penalty in reallocating objects that dynamically enter and leave the configuration during runtime. We refer to the passive/active scheme as light weighted objects. The opposite is called heavy weighted objects and it means objects are explicitly deallocated and reallocated whenever they are added or removed from the configuration. One drawback of light weighted objects is that there is a penalty in the broad phase collision detection algorithm. Fortunately broad phase collision detection algorithms should have linear running time with very low constants, so the penalty is negligible.

The last screening test is for change in relative placement. Every edge stores a transform, $xform(\cdot)$, indicating the relative placement of the end node objects. If the transform is unchanged there is no need to run narrow phase collision detection nor contact determination, because these algorithms would return the exact same results as in the previous iteration.

We are now ready for doing narrow phase collision detection and contact determination on the remaining overlaps. Output from these sort of algorithms are typical a set of feature pairs forming principal contacts, *PCs* and a penetration state. The contact graph edges provide a good place for storing this kind of information. The output of the narrow phase should of course also be cached in the edge, because most narrow phase collision detection algorithms reuse their results from the previous iteration to obtain constant time algorithms. Notice that the closest principal contact is also determined, closest contacts are often used in impulse based simulation or time of impact estimation computations. We do also test for any contact state changes, that is if touching or pen-

etrating contacts vanishes or are persistent, that is if they were present in the last iteration. If one of the nodes were a trigger volume then we do not mark touching contact, but rather **in**- and **out**- events of the trigger volume, the same applies to the marking that took place earlier on.

Finally we can run the contact determination for all those edges where their end node objects are not separated.

In an impulse based simulator it is often not necessary to do a full contact determination only the closest points are actual needed [7], so an end user might want to turn of contact determination completely.

Also contact determination should be skipped on nodes representing things like trigger volumes, such entities are merely used for event notification, so there is no need for contact determination.

Now we have completed exploiting all of the logical and caching benefits we can gain from a contact graph. We are now ready for using the contact graph for its intended purpose, determining contact groups. The actual contact groups are found by a traditional connected components search algorithm, restricted to the union of the list of edges having survived the logical and coherence testing as described earlier and the structural edges. The algorithm works by first marking all edges that should be traversed as “white”. Afterwards edges are treated one by one until no more white edges exist.

Fixed objects are rather special, they behave like they have infinite mass, so they can support any number of bodies without ever getting affected themselves. They work like an insulator, which is why we ignore edges from these nodes when we search for contact groups.

## The Event Handling

In the method we have described in this paper we have not really explicitly stated when events get propagated back to an end user. Instead we have very clearly shown when and how the events should be detected. We can traverse the edges of the graph, and simply generate the respective event notifications for all those edges that have been marked with an event.

There is one major subtlety to event handling, some simulators are based on backtracking algorithms, meaning they keep on running forward until things go wrong, then they backtrack correct things and tries to go forward again. This could many times. The consequences being that we might detect events which really never occurs.

The problem of backtracking can be handled in two ways, in the first solution events can be queued during the simulation together with a time stamp indicating the simulation time at which they were detected. Upon backtracking one simply dequeues all events with a time stamp greater than the time the simulator backtracks to.

In the second solution events are restricted to only be generated when it is “safe”, that means whenever a backtrack can not occur or on completion of the frame computation. We favor the second solution, the reason for this is that events are most likely to be used in a gaming context, where a backtracking algorithm is unlikely.

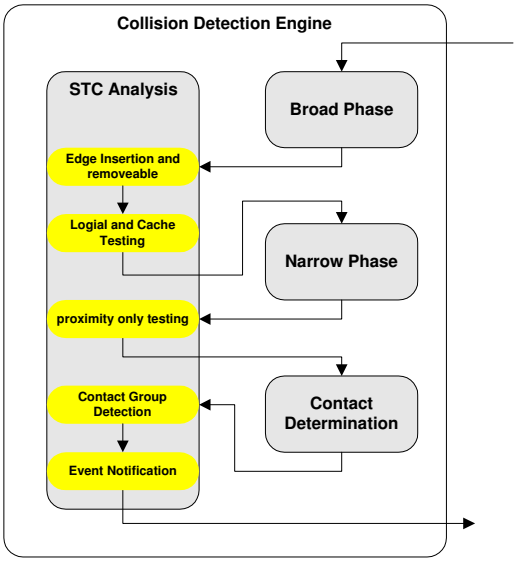



Figure 1: Spatial-Temporal Coherence Analysis Module.

## The STC-Analysis Module

Having outlined how the contact graph should be used in the collision detection pipeline we can schematically sketch the STC analysis module together with the other modules in the collision detection engine. Figure 1 illustrates the interaction between the modules. From Figure 1 we see that the STC analysis occurs in three phases, post-broad-phase, post-narrow-phase and post-contact-

determination. We do not need a pre-broad-phase in the STC analysis at this point, but if any initialization is supposed to take place then a pre-broad-phase analysis would be a good place for doing this.

In our opinion there are basically three different ways to exploit the contact groups in rigid body simulation. We will briefly talk about them in the following.

Time warping: Traditionally one would backtrack the entire configuration when an illegal state is found, such as a penetration of two objects. This is inefficient since there may be a lot of objects whos motion are completely independent of the two penetrating objects. Contact groups could be used to only backtrack those contact groups with penetrating objects see [3] for more details.

Subdivision of Contact Force Computation: Constraint-based methods for computing contact forces are often *NP*-hard, so it is intractable to have large problems, however the contact forces needed in one contact group is totally independent of all the other contact groups. The essence is basically why solve one big problem, when you can solve several smaller problems instead see [1].

Caching Contact Forces: If contact forces from the previous iteration of contact force computations are cached in the contact graph edges then these forces can be used as initial guess for the contact force computation in the current iteration as described in [8].

## Results

We will elaborate on several speedup methods that relies on or relates to contact graphs. The speedup methods are generally applicable to any kind of rigid body simulator. In order to show the effects we have chosen to extend our own multibody simulator, a velocity based complementarity formulation [9] using distance fields for collision detection, with the speed ups. Example code is available from the OpenTissue Project.

In this paper we will focus on performance speedup only. For this reason we have chosen a semi-implicit fixed time stepping scheme with a rather large timestep, 0.01 second.

Using distance fields for collision detection have one major drawback, when objects are deeply penetrating, a large number of contact points will be generated, the consequence will be a performance degradation due to the large number of variables. Performance improvements are therefore particular important even though real-time simulation is out of our grasp.

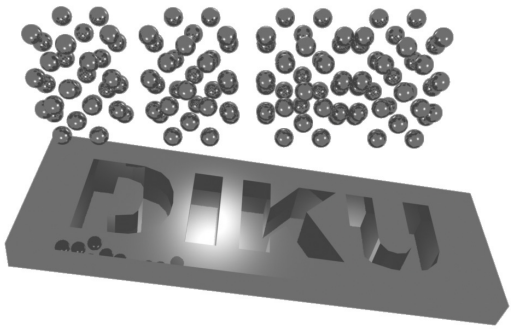


Figure 2: 120 Falling Spheres onto inclined plane with engravings.

We have done several performance measurements and statistics on 120 spheres falling onto an inclined plane with engravings. The configuration is shown in Figure 2. The total duration of the simulation is 10 seconds. In Figure 3 measurements of the brute force method is shown, i.e. without using contact graph. Observe that the number of variables and real-life time per iteration are increasing until the point where the spheres settle down to rest. In comparison Figure 4 shows how the curves from Figure 3 changes when a contact graph is used. Notice that the number of variables per contact group is much smaller than in the brute force method also observe the impact on the real-life duration curves.

The total running time of the brute force method is 28424 sec. Using a contact graph the simulation takes 1011.4 sec., which is a speedup factor of roughly 28. In the following we will explain 7 more speed up methods that further increases performance.

Using contact graphs an improvement comes from ignoring contact groups where all objects are at rest, we call such objects sleepy objects, and we determine them by tracking their kinetic energy, whenever we find an object who's kinetic energy have been zero within threshold over some constant number of iterations, which is user specified, the object is flagged as sleepy. If a contact group only con-

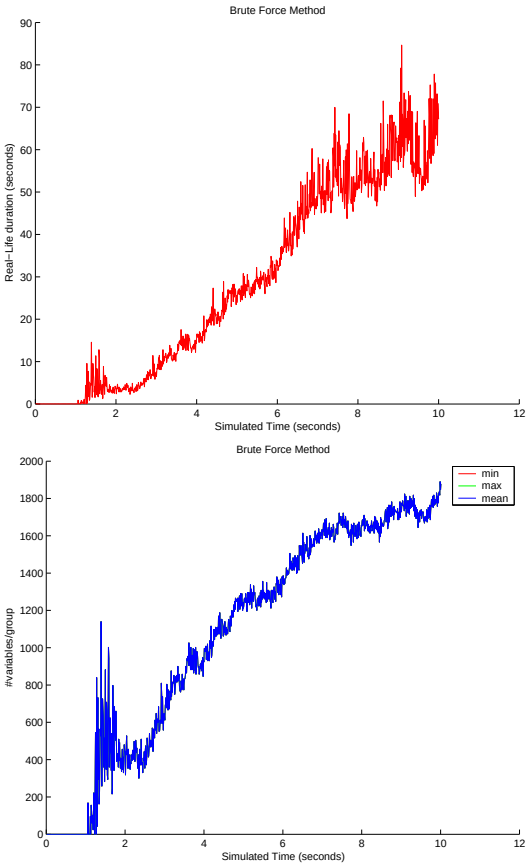



Figure 3: Performance of the brute force method.

tains sleepy objects the group is completely ignored during simulation. Contact graph nodes are used for the kinetic energy tracking. This method is computationally inexpensive, it have been used in all of the measurements in Table 1. The method could have a potentially disastrous effect if the scheme for tagging sleepy objects is not well-picked. Too greedy an approach could leave objects hanging in the air, too lazy an approach would result in no performance improvement.

We exploit contact graph edges for caching contact points and contact forces. Cached contact points are used to skip narrow phase and contact determination whenever two incident objects of a contact edge are at absolute rest, cached contact forces are used to seed the iterative LCP solver, Path from CPNET. We call this speed up “caching”.

A further speed up can be obtained by limiting the number of iteration of the LCP solver, currently from 500 to 15, as a consequence the motion is altered but still looks plausible. The method has noth-

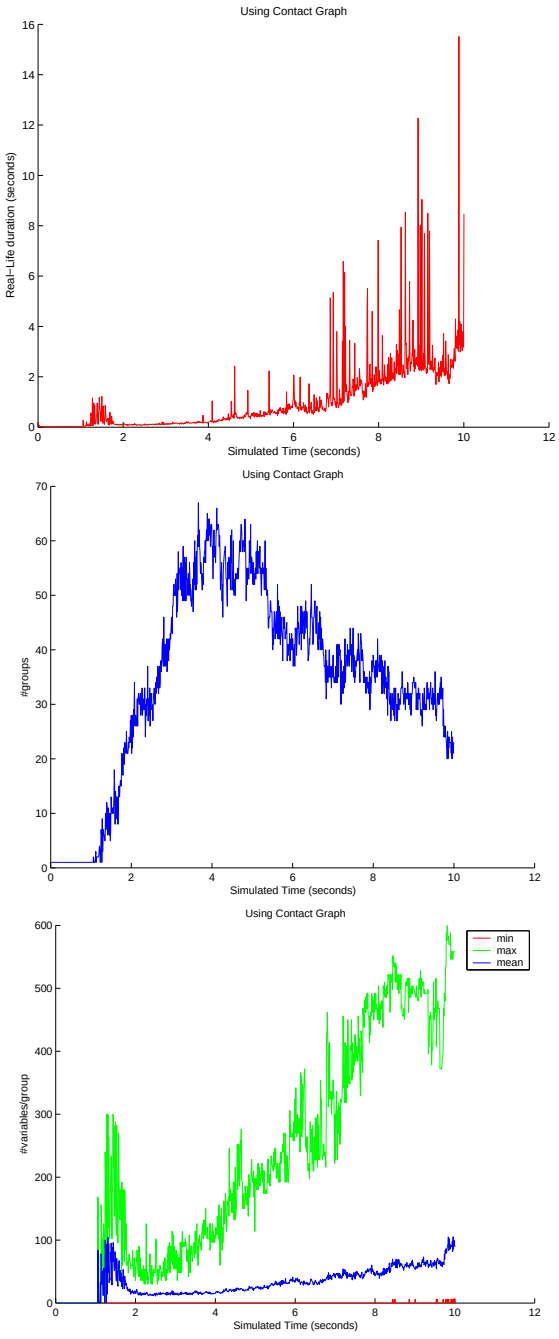



Figure 4: The performance impact of using a contact graph.

ing to do with contact graphs, but it is interesting in combination with the other speed ups we apply. We refer to this speed up as “tweaking”.

Another speed-up we use is to reduce the number of contacts between two objects in contact, the reduction is applied to objects that are deeply penetrating, all contacts are pruned except the single contact of deepest penetration. The contact graph edges are a convenient storage for this. We have named this “reduction”. Reduction have an effect on the motion of the objects, we believe that it is actually more correct, because intuitively the deepest point of penetration better resembles the original idea behind using the minimum translational distance as a separation measure. Besides theoretically reduction should decrease the number of variables used in the complementarity formulation.

Inspired by the speed up of detecting independent contact groups, further subdivision into independent groups seems attractive. An idea is to prune away sleepy objects from those contact groups containing both non-sleepy and sleepy objects, thereby hopefully breaking these into subgroups. We refer to this as “subgrouping”. Other subgrouping/sleepy object schemes can be found in [5, 10]. To help objects settle down and become sleepy faster it intuitively seems to be a good idea to let the coefficient of restitution fall to zero the more sleepy an object gets, meaning that sleepy objects are sticky objects. Currently we set the coefficient of restitution to zero whenever at least one of the incident objects are sleepy. We call this “zeroing”. In the same spirit a linear viscous damping term is added to the motion of all objects in the simulation, the intention is to slow down objects making them less willing to become non-sleepy. We call this heuristic “damping”. The contact graph is used for the subgrouping and zeroing by classifying edges dependent on the sleepy-state of the incident objects.

The last method we have applied consist of setting the inverse mass and inertia tensor to zero for all sleepy objects. The main intuition behind this is to “force” sleepy objects to stay sleepy. We have named this “fixation”. It has a dramatic impact on the simulation as seen form the ♠-simulation in Figure 5. Fixation makes only sense when subgrouping is used otherwise the LCP solver will have to solve contacts between two fixated objects.

Table 1 contains performance measurements of most promising combinations of the previously described heuristics and speedup methods.

| | Cache | Tweak | Reduce | Zero | Damp | Subgroup | Fixate | Time |
|---|---|---|---|---|---|---|---|---|
| ♢ | + | - | + | - | - | - | - | 624.274 |
| ♡ | - | + | - | - | + | - | - | 528.633 |
| ♣ | + | - | - | - | + | + | - | 460.561 |
| ♠ | + | + | + | - | + | + | + | 134.721 |

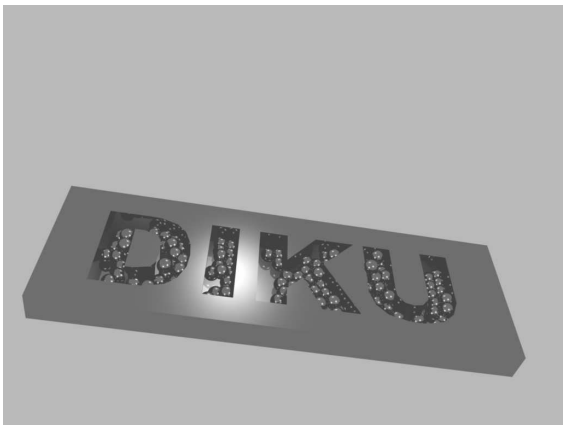
Table 1: Comparison of various combinations of speed-up methods. “+” means enabled, “-” means disabled. More combinations can be found in[6].

Figure 5 shows motion results of the two combinations: ♢ and ♠. Here ♢ is identical to the brute-force method. These four combinations: ♢, ♡, ♣, and ♠ were picked because they resembles the best performance. It should be noted the motion diverges more and more from the brute force method the more speed ups that are used. Especially ♠ is different, during the last seconds objects actually fly up in the air. Animations of the ♢, ♡, ♣, and ♠ simulations are available from corresponding authors homage. In Figure 6 a comparison is done between the performance statistics of ♣, and ♠. The ♢ and ♡ behave similar to ♣ as can be seen in [6]. The plots of ♣ are similar to Figure 4. The ♠, has very different plots for the real-life duration and variables per group plots, these appear to be nearly asymptotically constant.

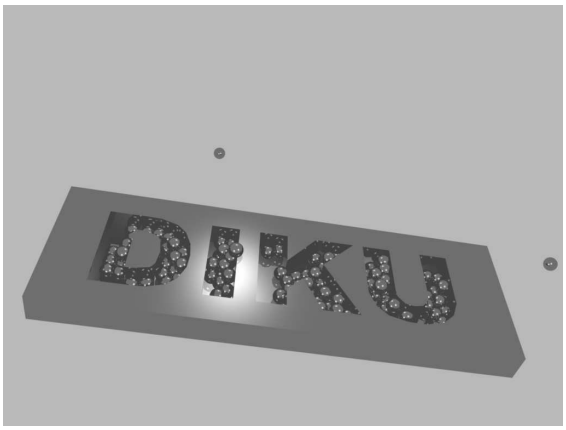
## Discussion

It is obvious from Table 1 that the prober combination of the speed ups is capable of producing a speed up factor of $\frac{28424}{135} \approx 210$. It is difficult to describe the impact on the resulting motion, however it is clear that using Contact Graphs, Caching Contact Forces and Sleepy Groups do not alter the mechanical system, but the other speed ups presented change the physical properties and thus the motion of the objects as can be seen in Figure 5. Especially the reduction and the subgrouping also with zeroing and fixation have great impact on the motion.

The tagging of sleepy objects can have a drastic impact on the simulation. As an side effect objects are sometimes left hanging. For instance in Figure 7 spheres land on top of each other, while the top-most spheres rumbles off, the bottom-most sphere is kept in place and prohibited from gaining kinetic energy, at the end of the simulation a single hanging sleepy sphere can be seen near the K-letter. We have to be




(a) ♢



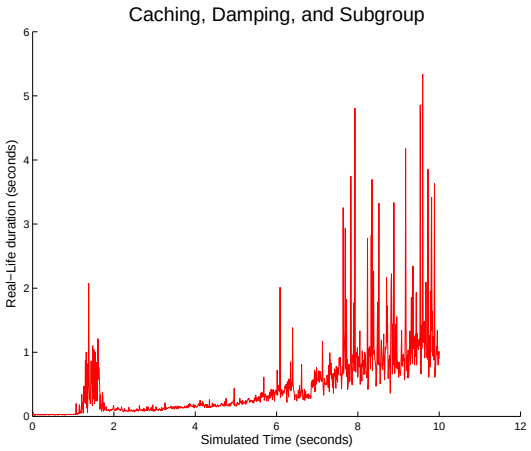

(b) ♠

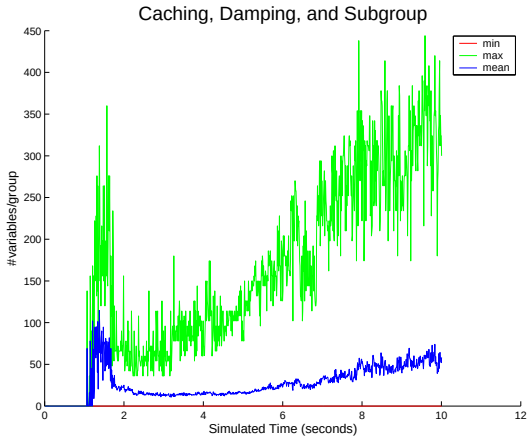
Figure 5: Motion Results at time 9.20 seconds for ♢ and ♠.

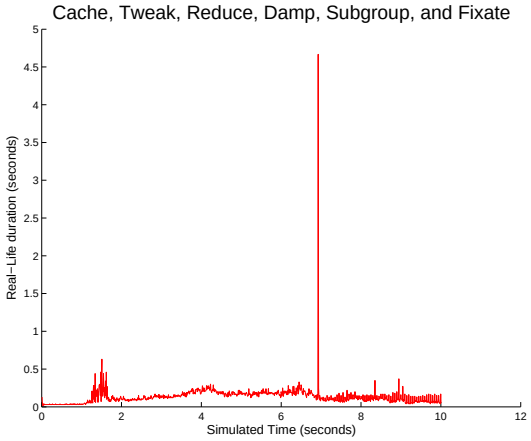
careful not making general conclusions based on the measurements in this paper, since only one configuration have been examined.

We can say as much as contact graphs are a valuable extension to a rigid body simulator, even when not trading accuracy for performance speed up factor of order 20-30 is not unlikely, disregarding accuracy the speed up factor can be increased further by an order of magnitude.

Better performance is not always achieved by using more speed ups, in some cases one speed up cancels the effect of another. For instance using caching seems to make tweaking needles. Also speed ups can alter the motion. Thirdly our experiments indicate that high-performance can be achieved by combining subgrouping and fixation, however from the motion results it also clear this is a non-trivial task to embark upon.



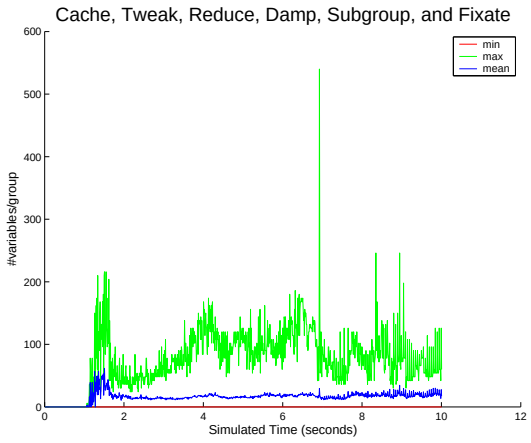




Figure 6: Performance measurements of ♣, and ♠ from Table 1.

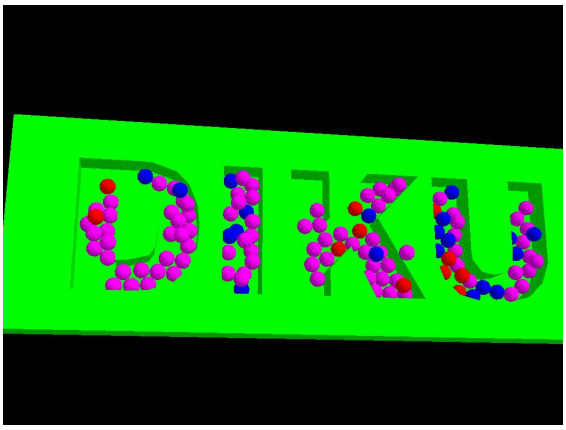



Figure 7: Figure showing hanging sphere near K-letter. Frame grab after 9.87 sec. using zeroing, damping, subgrouping and fixation.

Our numerical experiments clearly indicates that sleepy objects are a promising strategy, it seems promising to look into better methods for more quickly making objects sleepy and stay sleepy. For instance to pre-process the complementarity formulation with a sequential collision method truncating impulses, this was used for a sequential impulse based method in [4].

## References

[1] Anitescu M, Potra F.A, Stewart D. *Time-stepping for three-dimensional rigid body dynamics*, Comp. Methods Appl. Mech. Engineering, 1998.

[2] Hahn J. K. *Realistic animation of rigid bodies*, In Computer Graphics volume 22, 1988.

[3] Mirtich B. *Timewarp rigid body simulation*, In Proceedings of the 27th annual conference on Computer graphics and interactive techniques, 2000.

[4] Guendelman E, Bridson R, Fedkiw R. *Nonconvex rigid bodies with stacking*, ACM Transaction on Graphics, Proceedings of ACM SIGGRAPH, 2003.

[5] Barzel R. *Physically-based Modelling for Computer Graphics, a structured approach*, Academic Press, 1992.

[6] Erleben K. *Contact graphs in multibody dynamics simulation*, report no. 04/06, Copenhagen, DIKU, 2004.

[7] Mirtich B. *Impulse-based Dynamic Simulation of Rigid Body Systems*, PhD thesis, Berkeley, University of California, 1996.

[8] Baraff D. *Dynamic simulation of non-penetrating Rigid Bodies*, PhD thesis, Cornell University, 1992.

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

[10] Schmidl H. *Optimization-based animation*, PhD thesis, University of Miami, 2002.