A Parallel Rigid Body Dynamics Algorithm

Klaus Iglberger and Ulrich Rüde

Friedrich-Alexander University Erlangen-Nuremberg,
91058 Erlangen, Germany
klaus.iglberger@informatik.uni-erlangen.de
http://www10.informatik.uni-erlangen.de/~klaus/

Abstract. For decades, rigid body dynamics has been used in several active research fields to simulate the behavior of completely undeformable, rigid bodies. Due to the focus of the simulations to either high physical accuracy or real time environments, the state-of-the-art algorithms cannot be used in excess of several thousand rigid bodies. Either the complexity of the algorithms would result in infeasible runtimes, or the simulation could no longer satisfy the real time aspects.

In this paper we present a novel approach for large-scale rigid body dynamics simulations. The presented algorithm enables for the first time rigid body simulations of several million rigid bodies. We describe in detail the parallel rigid body algorithm and its necessary extensions for a large-scale MPI parallelization and show some results by means of a particular simulation scenario.

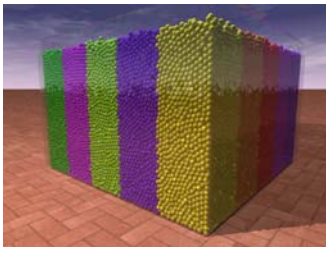
1 Introduction

The field of rigid body dynamics is a subfield of the classical mechanics introduced by Isaac Newton in the 17th century. The only assumption made in this field is that the treated bodies are completely rigid and cannot be deformed by external forces. Though the rigidity assumption is a rigorous idealization of real materials, many phenomena in nature and problems posed in engineering can be well approximated in this way.

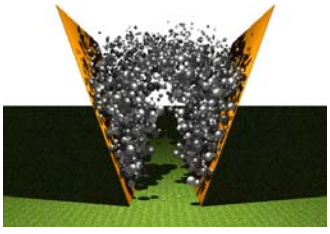
For instance, rigid body simulation has been used for decades in the robotics community that is primarily interested in the accurate prediction of friction forces. In material science, rigid body simulation is used for granular media simulations (see for example Fig. 1(a)) and in the computer graphics, computer games and virtual reality communities rigid body dynamics is used for authentic physical environments. For some years, rigid body dynamics has also been used in combination with lattice Boltzmann flow simulations to simulate the behavior of rigid bodies in a flow [10]. This coupled simulation system is for example used to simulate fluidization processes with a high number of buoyant rigid bodies (see for example Fig. 1(b)).

The most obvious difference in the existing rigid body simulation frameworks is the applied solver for the treatment of collisions between rigid bodies. Currently, the available algorithms for the calculation of collision responses can be

H. Sips, D. Epema, and H.-X. Lin (Eds.): Euro-Par 2009, LNCS 5704, pp. 760–771, 2009.



(a) Medium-sized simulation of a granular media scenario consisting of 175 000 spheres. The colors indicate the domains of the 25 involved MPI processes.



(b) Coupled simulation between a a lattice Boltzmann flow simulation and the *pe* rigid body physics engine.

Fig. 1. Simulation examples for rigid body simulations

very coarsely subdivided into two categories: the accurate formulations based on linear complementarity problems (LCP) that are able to accurately predict frictional contact forces at higher computational costs [5,15,11,4,14], and fast algorithms that scale linearly with the number of contacts but suffer from a reduced accuracy [13,6].

Due to the computational costs of LCP solvers (for instance extensions of the Gauss-Seidel, Gauss-Jacobi or Conjugate Gradient methods), the maximum number of rigid bodies in such a simulation is limited to several thousand rigid bodies. A larger number of rigid bodies would result in completely infeasible runtimes. The faster algorithms, on the other hand, can handle more rigid bodies due to their linear complexity, but are often also limited to several thousand to ten thousand rigid bodies because of their real time requirements.

The algorithm presented in this paper represents the first large-scale rigid body dynamics algorithm. It is based on the fast frictional dynamics algorithm proposed by Kaufman et al. [12] and is parallelized with MPI [9]. Using this algorithm, our rigid body simulation framework named *pe* (the abbreviation for physics engine) enables the simulation of several million interacting rigid bodies on an arbitrary number of cores. As a first example of a large-scale rigid body simulation, the example demonstrated in Fig. 2 shows the simulation of 500 000 spheres and boxes falling into a well built of 3 000 fixed boxes. The simulation domain is partitioned into 91 subdomains, where each subdomain is managed by a single MPI process. All bodies contained in a subdomain are exclusively known to this managing process. Therefore all processes have to communicate with neighboring processes about rigid bodies crossing a process boundary. Due to the hexagonal shape of each subdomain, each process has to communicate with a maximum number of six neighboring processes. Please note that this simulation only rudimentarily demonstrates the full capacity of the parallel algorithm and was chosen such that individual objects are still distinguishable in the visualization.

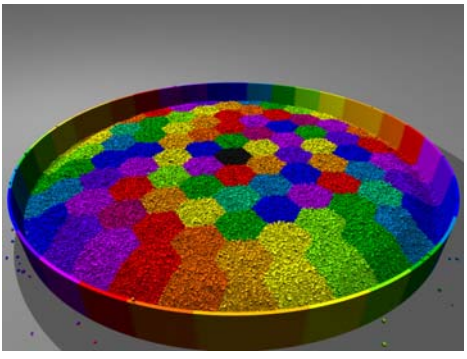


Fig. 2. Simulation of 500 000 spheres and boxes falling into a well built of 3 000 fixed boxes. The colors indicate the domains of the 91 MPI processes. Due to the hexagonal setup of the domains, each MPI process has a maximum number of six neighboring processes.

This paper is structured in the following sections: Sec. 2 gives an overview of related work in the field of rigid body dynamics and large-scale simulations. In Sec. 3, the focus lies on the parallel rigid body algorithm. This section explains the MPI parallelization of the algorithm and will explain the extensions of the original fast frictional dynamics algorithm. Sec. 4 analyzes of the scaling behavior of the parallel algorithm, whereas Sec. 5 concludes the paper.

2 Related Work

Tasora et al. [16] presented a large-scale parallel multi-body dynamics algorithm for graphical processing units. Their algorithm is based on a cone complementarity problem for the simulation of frictional contact dynamics. Their approach is suited for more than one million rigid bodies. There exist several alternative rigid body simulation frameworks that basically offer the same rigid body functionalities as our framework. Two open source examples are the Open Dynamics Engine (ODE) [1] and the OpenTissue library [2]. However, currently none of them offers massively MPI parallel rigid body dynamics as introduced in this paper.

Fleissner et al. [8] focus on parallel simulation of particle-based discretization methods, such as the discrete element method or smoothed particle hydrodynamics, instead of rigid body dynamics. Although their approach offers the option to run large-scale particle simulations, they are only considering point masses instead of fully resolved rigid bodies.

3 The Parallel Rigid Body Algorithm

The parallel rigid body dynamics algorithm presented in this section is based on the fast frictional dynamics (FFD) solver first published by Kaufman et al.

[12] and improved by Wengenroth [17]. Due to its formulation, this algorithm belongs to the group of fast, real time collision response algorithms. For a detailed explanation of the nonparallel version of this algorithm, please refer to either one of the two previously mentioned references. This section will primarily focus on the parallelization of this algorithm.

The FFD solver is particularly suited for a large-scale parallelization due to its strictly local collision treatment: in order to calculate a post-collision velocity for a colliding rigid body, only the states of the contacting rigid bodies are considered. Therefore it is not necessary to set up and solve a LCP in parallel as it would be necessary in case of the LCP-based collision response algorithm. Additionally, the complexity of the FFD algorithm is linearly depending on the number of rigid bodies and the number of contact points between these bodies.

Alg. 1 shows the parallel FFD algorithm. In contrast to the nonparallel algorithm, the parallel version contains a total of four MPI communication steps to handle the distributed computation (see the lines 1, 6, 21 and 34). The rest of the algorithm remains unchanged in comparison to the nonparallel formulation.

Instead of immediately starting with the first position and velocity half-step for every rigid body, the first step in every time step of the parallel algorithm is the synchronization of the external forces (Alg. 1, line 1). In case a rigid body is overlapping a process boundary, part of the external forces can be contributed by the remote processes. This scenario happens for instance in case the rigid body is immersed in a fluid flow (as illustrated in Fig. 1(b)). Part of the hydrodynamic forces are calculated by the local process, another part by the remote processes. Therefore the involved processes have to synchronize their forces and calculate a total force for every rigid body.

After the force synchronization, all local rigid bodies (i.e. all rigid bodies whose reference point is contained in the local process) are updated with the first position half-step and first velocity half-step. Note that only local rigid bodies are treated. Every rigid body is updated by exactly one process, i.e. by the process its center of mass is contained in. Remote rigid bodies (i.e. all rigid bodies whose reference points are not contained in the local process) are updated in the second MPI communication step (Alg. 1, line 6). This communication step involves a synchronization of the position, the orientation and the velocities of remote rigid bodies, where the process that contains the reference point of the body sends the updated values to all neighboring remote processes that only contain part of the body. Due to the position and orientation change of all rigid bodies, it may also be the case that a rigid body now newly overlaps a process boundary to a particular neighboring remote process. In this case the entire rigid body has to be sent to the remote process, additionally including information about its geometry (as for instance the radius of a sphere, the side lengths of a box, the triangle mesh of an arbitrary geometry) and its material (containing information about its density, the coefficient of restitution and frictional parameters). With this information, the remote process is now able to create a copy of the rigid body.

Algorithm 1. The Parallel FFD-Algorithm

```
1  MPI communication step 1: force synchronization
2  for each body B do
3  |  position half-step (φ_B^+)
4  |  φ_B^- = velocity half-step (B)
5  end
6  MPI communication step 2: update of remote and notification of new rigid
   bodies
7  for each body B do
8  |  find all contacts C(B)
9  |  for each contact k ∈ C(B) do
10 |  |  calculate twist n_k induced by contact normal
11 |  |  calculate constraint offset d_k
12 |  |  if constraint is violated ( B, φ_B^-, n_k, d_k ) then
13 |  |  |  add collision constraint n_k, d_k to T(B)
14 |  |  |  for m = 1 ... SAMPLESIZE do
15 |  |  |  |  calculate twist s_m induced by m^th sample ⊥ to contact normal
16 |  |  |  |  add friction constraint s_m, μ_m to S(B)
17 |  |  |  end
18 |  |  end
19 |  end
20 end
21 MPI communication step 3: exchanging constraints on the rigid bodies
22 for each body B do
23 |  if B has violated constraints then
24 |  |  find post-collision velocity φ_B^τ on the border of T(B) closest to φ_B^-
25 |  |  r_B = φ_B^- - φ_B^τ
26 |  |  select friction response δ_B from S(B) minimizing δ_B + φ_B^τ
27 |  |  φ_B^+ = φ_B^τ + ε · r_B + δ_B
28 |  end
29 |  else
30 |  |  φ_B^+ = velocity half-step (B)
31 |  end
32 |  position half-step (B)
33 end
34 MPI communication step 4: update of remote and notification of new rigid
   bodies
```

Note that, due to efficiency reasons, updates should be preferred to sending the complete rigid body. Only in case the remote process does not know the rigid body, sending the full set of information cannot be avoided. Also note, that it may be necessary to send updates for a rigid body that is no longer overlapping a particular boundary in order to give the remote process the information that the rigid body has left the domain of the remote process and can therefore be destroyed on the remote process.

The second communication step is followed by the collision detection that creates contact points for every pair of rigid bodies that is in contact. Directly afterwards,

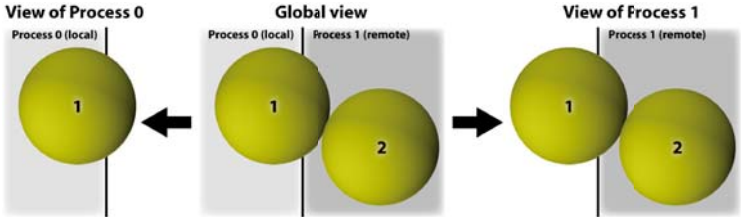


Fig. 3. Some collisions can only be detected on remote processes. In the illustrated example, process 0 only knows the local sphere 1, for which collision constraints have to be calculated. Process 1, on the other hand, knows both sphere 1 and 2 (where sphere 1 is remote and 2 is local). Therefore only process 1 can setup the constraints for the collision between spheres 1 and 2 and has to send these constraints to process 0, where they are used to handle the collision between the two spheres.

the setup of all collision constraints can be performed (see Alg. 1, line 7-20): for every contact point attached to the rigid body, a single collision constraint and several friction constraints are created in case the collision constraint is violated.

In the consecutive third communication step (Alg. 1, line 21), the MPI processes exchange collision and friction constraints among each other such that every process knows all active constraints on all local rigid bodies. This exchange is necessary since some constraints can only be detected on remote processes, as illustrated in Fig. 3. These constraints can now be used to treat all collisions and to update the velocities of the colliding rigid bodies. In case a rigid body is not involved in a collision, the velocity is updated in a second velocity half-step. After the velocity update (either due to collision handling or a velocity half-step), the second position half step is performed. In order to cope with this second change of the position and orientation of the rigid bodies, the fourth communication step (Alg. 1, line 34) performs the same synchronization as the second communication step. After this communication, all rigid bodies on all processes are synchronized and in a consistent state and the time step is finished.

For every communication step during the parallel rigid body simulation, the send buffers for the neighboring processes have to be filled individually with the according informations. This encoding step is succeeded by the message exchange via MPI, which is then followed by the decoding phase of the received messages. The complete MPI communication of the *pe* engine is shown in Alg. 2. The first step during communication is the initialization of the transfer of the according byte-encoded messages to the neighboring MPI processes via the non-blocking message passing function `MPI_Isend()`. While the MPI system handles the send operations, the receive operations are initiated. Since the size of the messages is unknown, the `MPI_Probe()` and `MPI_Get_count()` functions are used to test any incoming message. After that, the size of the receive buffer is adjusted and the blocking message passing function `MPI_Recv()` is used to receive the message. After all messages have been received, it is necessary to wait for all non-blocking send operations to finish. Only then the send buffers can be cleared for the next communication step.

Algorithm 2. MPI communication

```
for each neighboring process do
    Encode the message to the neighboring process
    Start a non-blocking send operation of the according byte encoded message
    via MPI_Isend()
end
for i from 0 to the number of neighboring processes-1 do
    Check for any incoming message via MPI_Probe()
    Acquire the total size of the message via MPI_Get_count()
    Adjust the buffer size of the receiving buffer
    Receive the message via MPI_Recv()
    Decode the received message
end
Wait until all non-blocking send operations are completed
Clear the send buffers
```

Fig. 4 gives an example of a single moving rigid body that crosses the boundary between two processes. The example explains in detail, when the rigid body has to be sent by which process and when the responsibility for a rigid body is transferred to another process.

4 Results

In this section, the scaling behavior of the parallel FFD algorithm is analyzed closely. The machine we are using for our scaling experiments is the Woodcrest cluster at the regional computing center Erlangen (RRZE) [3]:

- 217 compute nodes, each with two Xeon 5160 Woodcrest chips (4 cores) running at 3.0 GHz, 4 MB Shared Level 2 Cache per dual core, and 8 GB of RAM
- Infiniband interconnect fabric with 10 GBit/s bandwidth per link and direction
- Overall peak performance of 10.4 TFlop/s (LINPACK result: 6.62 TFlop/s)

We are using up to 512 cores in order to simulate up to 8 million rigid bodies in a single simulation. All executables are compiled with the Intel 11.0 compiler (using the optimization flag `-O3`) and we are using Intel-MPI in the version 3.1.038.

The scenario we have chosen for our scaling experiments is illustrated in Fig. 5: we are simulating the movement of spherical particles in an evacuated box-shaped volume without external forces. For this we create a specified number of uniformly distributed spheres, each with a random initial velocity. This scenario is well suited for our scaling experiments, since it guarantees an equally distributed load even without load balancing strategies and similar communication costs on all involved processes.

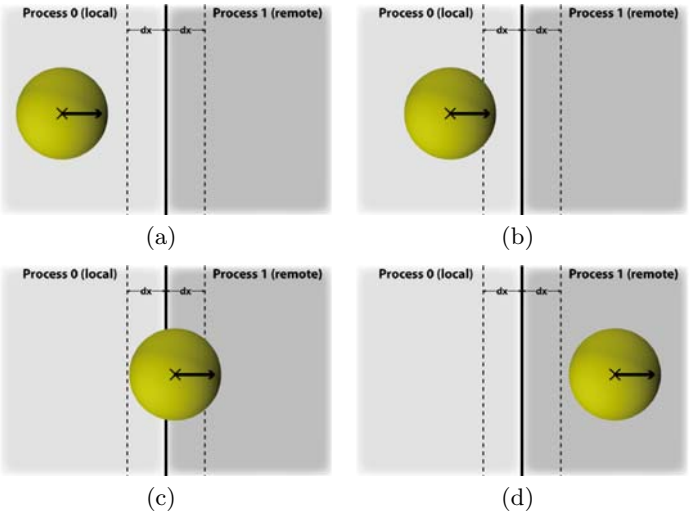


Fig. 4. Example of the communication of a single rigid body. The reference point of the rigid body is indicated by the cross in its center, its linear velocity by the arrow. In Fig. 4(a), the rigid body is solely contained in process 0. Process 1 does not know the rigid body. In Fig. 4(b), the rigid body has moved into the overlap region between the two processes and therefore has to be sent to process 1. Only in the first communication, the entire rigid body, including information about its geometry and its material have to be sent. In subsequent communications, it is sufficient to send updates for the body. In Fig. 4(c), the rigid body's reference point is no longer contained in process 0, but has moved to process 1. Therefore process 1 now considers the rigid body as local, process 0 only as remote. In Fig. 4(d), the rigid body has left the overlap region between the two processes and therefore doesn't have to be communicated anymore. Therefore process 0 no longer contains any part of the body, so the body is locally destroyed. Note that the choice of the size of dx is arbitrary, but larger than the minimum distance between two colliding bodies.

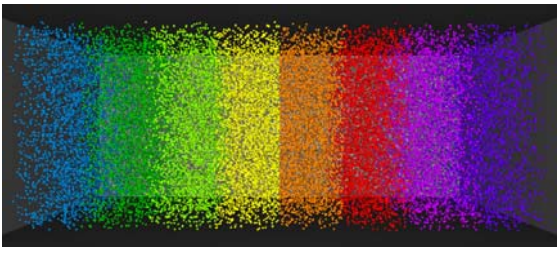


Fig. 5. Simulation scenario for the scaling experiments: the specified number of spherical rigid bodies are moving randomly in a closed, evacuated, box-shaped domain without external forces.

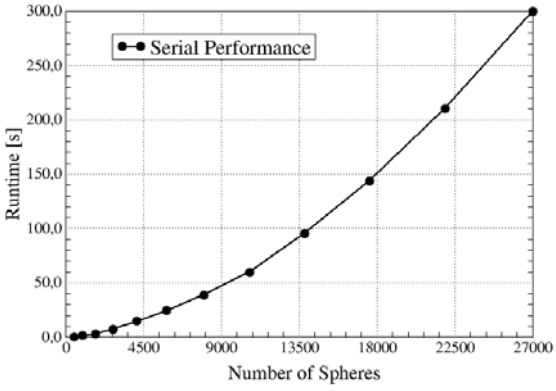


Fig. 6. Serial performance of 1 000 time steps in the scenario for the strong and weak scaling experiments

In order to be able to analyze the parallel performance of the algorithm, it is first necessary to investigate the serial performance of the rigid body framework. Fig. 6 shows a serial scaling experiment with up to 27 000 spheres in the scenario described above. Apparently the rigid body simulation exhibits a scaling behavior nonlinear with the number of bodies. One of the main reasons for this behavior is the lack of perfectly linear algorithms for some phases of the algorithm, as for instance the contact detection phase. In our simulation, we are using the sweep-and-prune algorithm as described in [7] for the coarse collision detection. Although this is one of the fastest available algorithms, it doesn't offer a perfectly linear scaling behavior.

Table 1 contains the results for a strong scaling experiment with 65 536 spheres initially arranged in a $64 \times 64 \times 16$ grid. In this experiment, the spheres use approximately 1% of the total volume of the domain. The first observation is that doubling the number of CPU cores improves the performance of the simulation beyond a factor of two. This can easily be explained by the serial scaling results: dividing the number of rigid bodies per MPI process by two yields more than a factor of two in performance. The second observation is that the reduction of communication overhead by a smaller cut area improves the performance considerably.

Weak scaling in a rigid body simulation proves to be more complicated than strong scaling, because increasing the number of rigid bodies inevitably changes the simulated scenario due to additional interactions. It also raises the question of where to create the new bodies. Therefore we chose to use weak scaling to analyze the performance influence of the number of communication neighbors and to demonstrate simulations with several million rigid bodies.

In our weak scaling experiments, each process owns a cubic subdomain of initially $25 \times 25 \times 25$ spheres. These blocks of spheres are now put together depending on the arrangement of the processes. Note that, due to this setup, the weak scaling experiments cannot be directly compared to the strong scaling

Table 1. Strong scaling experiment of 1 000 time steps with 65 536 spheres initially arranged in a 64x64x16 grid. The total volume of the spheres occupies approximately 1% of the domain volume.

# Cores	Partitioning	Runtime [s]
4	4 × 1	428.1
	2 × 2	351.0
8	8 × 1	201.1
	4 × 2	157.5
16	16 × 1	84.69
	8 × 2	62.38
	4 × 4	60.59
32	32 × 1	33.85
	16 × 2	24.95
	8 × 4	20.99

Table 2. Weak scaling experiment of 1 000 time steps with up to 8 000 000 spheres. Each process initially owns a block of 25 × 25 × 25 spheres. The total volume of the spheres occupies approximately 1% of the domain volume.

# Cores	# Spheres	Partitioning	Runtime [s]
64	1 000 000	64 × 1 × 1	273.2
		8 × 8 × 1	428.9
		4 × 4 × 4	609.7
128	2 000 000	128 × 1 × 1	278.5
		16 × 8 × 1	437.4
		8 × 4 × 4	641.1
256	4 000 000	256 × 1 × 1	279.9
		16 × 16 × 1	447.8
		8 × 8 × 4	645.5
512	8 000 000	512 × 1 × 1	278.5
		32 × 16 × 1	458.9
		16 × 8 × 4	670.6

experiments: whereas in both strong scaling experiments the shape of the domain was fixed and the shape of the subdomains was adapted according to the arrangement of the processes, in the weak scaling experiments the shape of a subdomain is fixed and the shape of the domain results from the arrangement of the processes.

The weak scaling experiments is shown in Table 2. The first observation is that the scaling behavior clearly depends on the number of communication neighbors: when choosing a particular number of processes, a one-dimensional partitioning is clearly the fastest, whereas a two-dimensional partitioning increases the maximum number of communication neighbors from two to eight. In case a three-dimensional partitioning is chosen, the maximum number of communication neighbors even increases to 26. Note that this results primarily from the choice of cubic blocks of spheres per process, which directly leads to an increased

communication overhead in case of more neighbors, since for every neighboring process, messages have to be individually encoded and decoded. The second observation is that different partitionings with a similar communication overhead result in approximately the same runtime. Considering the fact that every single measurement represents a different scenario, this is a result that demonstrates the generally good scaling behavior of the MPI parallelization of the *pe* rigid body engine.

The simulations with 512 cores deserve some special attention. Although these simulations do by far not show the limits of the *pe* framework, they demonstrate that it is possible to simulate several million interacting rigid bodies in reasonable runtimes.

5 Conclusions

In this paper we have presented a novel approach to large-scale rigid body simulations. With the algorithm presented, we are able to simulate several million rigid bodies on an arbitrary number of processes. This increase in the number of rigid bodies in a single simulation by at least two magnitudes in comparison to other algorithms enables for the first time the simulation of large-scale rigid body scenarios such as granular media, fluidization processes or sedimentation processes. The obvious advantage in comparison to other large-scale, particle-based simulation methods, for example molecular dynamics, the discrete element method, or smooth particle hydrodynamics, is that the geometry of the rigid bodies is fully resolved. Therefore it is possible to simulate these scenarios with arbitrarily shaped rigid bodies.

We have demonstrated the good scaling behavior of our algorithm. However, in this paper we have only used spheres, and we have not used any load balancing extension. Instead, we have focused on the analysis of the performance with fixed process boundaries. Further performance experiments should consider load balancing and investigate the influence of different geometries on the performance. Future work will include experiments on a larger number of cores and rigid body simulations of physically relevant scenarios.

References

1. Homepage of the Open Dynamics Engine (ODE), `http://www.ode.org/`
2. Homepage of the OpenTissue simulation framework, `http://www.opentissue.org`
3. Homepage of the Regional Computing Center Erlangen (RRZE), `http://www.rrze.uni-erlangen.de`
4. Anitescu, M.: Optimization-based simulation of nonsmooth rigid multibody dynamics. Math. Program. 105(1), 113–143 (2006)
5. Cottle, R.W., Pang, J.S., Stone, R.E.: The Linear Complementarity Problem. Academic Press, Inc., London (1992)
6. Eberly, D.: Game Physics. Series in Interactive 3D Technology. Morgan Kaufmann, San Francisco (2003)

7. Erleben, K., Sporring, J., Henriksen, K.: Physics-Based Animation. Delmar (2005)
8. Fleissner, F., Eberhard, P.: Parallel load-balanced simulation for short-range interaction particle methods with hierarchical particle grouping based on orthogonal recursive bisection. International Journal for Numerical Methods in Engineering 74, 531–553 (2007)
9. Gropp, W., Skjellum, A., Lusk, E.: Using MPI: Portable Parallel Programming with the Message Passing Interface, 2nd edn. MIT Press, Cambridge (1999)
10. Iglberger, K., Thürey, N., Rüde, U.: Simulation of Moving Particles in 3D with the Lattice Boltzmann Method. Computers & Mathematics with Applications 55(7), 1461–1468 (2008)
11. Jean, M.: The non-smooth contact dynamics method. Computer Methods in Applied Mechanics and Engineering 177(3–4), 235–257 (1999)
12. Kaufman, D.M., Edmunds, T., Pai, D.K.: Fast frictional dynamics for rigid bodies. ACM Transactions on Graphics (SIGGRAPH 2005) 24, 946–956 (2005)
13. Millington, I.: Game Physics Engine Development. Series in Interactive 3D Technology. Morgan Kaufmann, San Francisco (2007)
14. Preclik, T.: Iterative rigid multibody dynamics. Diploma thesis, Friedrich-Alexander University of Erlangen-Nuremberg, Computer Science 10 – Systemsimulation (2008)
15. Renouf, M., Alart, P.: Conjugate gradient type algorithms for frictional multicontact problems: applications to granular materials. Computer Methods in Applied Mechanics Engineering 194, 2019–2041 (2005)
16. Tasora, A., Negrut, D., Anitescu, M.: Large-scale parallel multi-body dynamics with frictional contact on the graphical processing unit. Proceedings of the Institution of Mechanical Engineers, Part K: Journal of Multi-body Dynamics 222(4), 315–326 (2008)
17. Wengenroth, H.: Rigid body collisions. Master's thesis, University of Erlangen-Nuremberg, Computer Science 10 – Systemsimulation 2007, Computer Science Department 10 (System Simulation), University of Erlangen-Nuermberg (2007)