Optimized Impulse-Based Dynamic Simulation

Daniel Bayer, Raphael Diziol and Jan Bender

Universität Karlsruhe (TH), Karlsruhe, Germany

Abstract

The impulse-based dynamic simulation is a recent method to compute physically based simulations. It supports the simulation of rigid-bodies and particles connected by all kinds of implicit constraints. In recent years the impulse-based dynamic simulation has been more and more used to simulate deformable bodies as well.
These simulations create new requirements for the runtime of the method because very large systems of connected particles have to be simulated to get results of high quality. In this paper several runtime optimizations for the impulse-based dynamic simulation are presented. They allow to compute the same simulations at a fraction of time needed for the original method. Therefore, larger systems or simulations with increased accuracy can be simulated in realtime.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.5]: Computational Geometry and Object Modeling—Physically based modeling, Computer Graphics [I.3.7]: Three-Dimensional Graphics and Realism—Animation

1. Introduction

Today physically based simulations are a major part of many applications. They are widely used in research facilities, for example to test, develop or train robots. The use of virtual simulations also helps to avoid many expensive real life experiments. They even allow to get information about procedures that cannot be observed at all, due to their dimension or speed. In such simulations a high degree of accuracy is needed. Nevertheless, a fast simulation would be appreciated or at least a chance to preview the simulation results to fix gross errors before starting a longtime simulation.

Furthermore, physically based simulations are widely used in the entertainment industry to produce realistic animations for movies or computer games. In such simulations the accuracy is of lower importance. Especially in computer games and other real-time applications like virtual reality, the speed of the simulation comes to the fore. But the simulations have to be accurate enough to be stable and to deliver reliable results.

These requirements show the need for a fast and an accurate simulation method as well. The impulse-based dynamic simulation, which is used in this paper, can meet this expectations. One of its big advantages above other methods is its scalability between accuracy and runtime. Nevertheless, the original method offers many possibilities for optimizations. In this paper we present some of these optimizations to make the impulse-based dynamic simulation faster or more accurate within the same runtime.

2. Related work

In [TPBF87] the first general physical model for dynamic simulations of two- and three-dimensional deformable bodies was presented. It is based on implicit time integration and simulated by the use of a matrix solver. However, because of the limitations of former computer hardware this approach was only suitable for small models and needed much time to compute.

Therefore, explicit methods became more popular in the later days. Those methods where mainly based on meshes of particles connected with springs and dampers. They allow very fast simulations because each connection is handled without consideration of the others. But such simulations have the drawback that the error cannot be limited or predicted. This leads to very elastic simulations that are often recognized as physically implausible. To reduce the extensibility, large spring constants are needed, and therefore

the step sizes have to be chosen very small to solve the upcoming stiff differential equations [HES03].

Through the increased hardware performance and motivated by the problems of the explicit methods, implicit methods regained attention. Such an implicit approach is presented in [BW98], which was analyzed by various groups (e.g. [VT00]). With this work the elasticity could be reduced and the gained stability allowed to take larger time steps. However, additional stabilization techniques are needed to counter numerical drift and therefore to limit the error. And, furthermore, an overdetermined and in general non-linear system of equations has to be solved. This still needs a lot of computational time, especially if the simulations are complex and the resulting matrices are large.

To encounter this performance problems so-called IMEX-methods (mixed implicit and explicit) were proposed. They are based on the idea to split the differential equations in stiff and non-stiff parts and solve them separately. For example [HCJ*05] used implicit constraint enforcement only if the strain exceeds 10%.

Other methods to simulate deformable bodies are geometrically motivated, as for example [MHTG05]. This method is used to simulate soft bodies and is based on shape-matching between the original and the actual model. [SSBT08] proposed another geometric method based on shape-matching to simulate cloth. Such geometric methods are very stable and efficient but in general not physically correct.

Also level of detail methods were used to further optimize the runtime. These methods are mainly based on simplification by the cost of quality. For example [KC02] uses a coarse mesh for the global motion and a fine mesh for details. Such multi-grid methods were further examined, as for example in [OGRG07].

Another way to optimize the runtime is to use special hardware to compute dynamic simulations, as the graphics boards. Since general purpose computation on the GPU (GPCGPU) gained relevance, many of the mentioned methods were transformed to the GPU. Such methods are for example [Zel05] for explicit spring-damper simulations or [BBD09] for the impulse-based dynamic simulation.

Besides the computation on the GPU a further optimization of the impulse-based dynamic simulation is presented in [BB08]. It is based on parallel solving of a linear systems of equations.

3. Impulse-based dynamic simulation

The impulse-based dynamic simulation is a recent method to simulate articulated multi body systems (cf. [BS06]). It handles particles and rigid bodies and enforces constraints by the successive appliance of impulses. It resolves constraints implicitly and therefore can handle large step sizes. These constraints can either be bilateral (holonomic) or unilateral (non-holomic), for example, joints or contacts. This method also handles systems with cycles and is not affected by numerical drift (cf. [BS06]).

3.1. Particle simulation

A particle, or point mass, is a body without extend and therefore has, in contrast to rigid bodies, no rotation. Its physical state can be expressed by its constant scalar mass m, its position $\mathbf{p}$ and its linear velocity $\mathbf{v}$. To dynamically simulate a particle, its position and velocity have to be integrated over a time step of size h. Using external forces $\mathbf{F}$ that are constant during $[t_0, t_0+h]$, the new position and velocity can be directly computed by the following equations:

$$\mathbf{v}(t_0+h) = \mathbf{v}(t_0) + \frac{\mathbf{F}}{m}h \tag{1}$$

$$\mathbf{p}(t_0+h) = \mathbf{p}(t_0) + \mathbf{v}(t_0)h + \frac{1}{2}\frac{\mathbf{F}}{m}h^2 \tag{2}$$

where t_0 is the actual time before the time step.

3.2. Constraint enforcement

The particles' motion is constrained by the use of implicit functions

$$\mathbf{c}(\mathbf{p},\mathbf{v},t) = \mathbf{0} \text{ or } \mathbf{c}(\mathbf{p},\mathbf{v},t) \geq \mathbf{0} \ ,$$

where the vectors $\mathbf{p}$ and $\mathbf{v}$ contain all positions and velocities of the particles. Any function of this kind can be used to restrict the motion.

To enforce constraints that are dependent on the particles positions, their state is evolved forward in time. This is done by integrating the particles' positions by the use of equation 2. With this predicted state the constraint error e is computed by solving the constraint function at time t_0+h. If the error is greater than a certain tolerance value ε_d, a correction impulse $\mathbf{i}_p$ is applied. For two bodies i and j, which are linked by a constraint $\mathbf{c}$, this impulse is given by:

$$\mathbf{i}_p = \left(\frac{1}{m_i} + \frac{1}{m_j}\right)^{-1} \frac{\partial \mathbf{c}}{\partial \mathbf{p}} \mathbf{e} \frac{1}{h} \ .$$

The impulse points in the direction $\nabla\mathbf{c}$ as the principle of virtual work states and is applied to both linked bodies in opposite directions:

$$\mathbf{v}_i(t) := \mathbf{v}_i(t) + m_i^{-1} \cdot \mathbf{i}_p \ ,$$
$$\mathbf{v}_j(t) := \mathbf{v}_j(t) - m_j^{-1} \cdot \mathbf{i}_p \ .$$

Hence, $\mathbf{i}_p$ does not change the system's energy. This impulse instantaneously changes the velocity of the linked bodies so that the error is eliminated.

To handle multiple constraints, this method processes them one after another. Unfortunately, the fulfillment of one constraint may violate another one. Therefore, this process is repeated iteratively until all errors are beneath ε. This process converges to the physical correct solution (cf. [SBP05]).

After all position dependent constraints are satisfied, it is ensured that their error is below ε_d in the next time step. Therefore, the new positions and velocities can be computed.

Afterwards the velocity dependent constraints are computed in the same manner. Again, correction impulses are computed iteratively. However, this time not to correct the predicted positions but directly their velocities. Thus, the position prediction is omitted. An impulse which corrects a velocity constraint is obtained by the following equation:

$$\mathbf{i}_v = (\frac{1}{m_i} + \frac{1}{m_j})^{-1} \frac{\partial \mathbf{c}}{\partial \mathbf{v}} \mathbf{e} \ ,$$

where $\mathbf{e}$ is the error of the velocity constraint. This impulse is then, again, applied with opposite signs.

If only velocity independent constraints are simulated, the second loop can be ignored. But since every position constraint also makes demands on the velocities (cf. 3.3), this second loop can increase stability and lower the number of needed iterations in the next time step, even in the absence of velocity dependent constraints.

Figure 1 summarizes the whole process of the impulse-based dynamic simulation as a scheme.

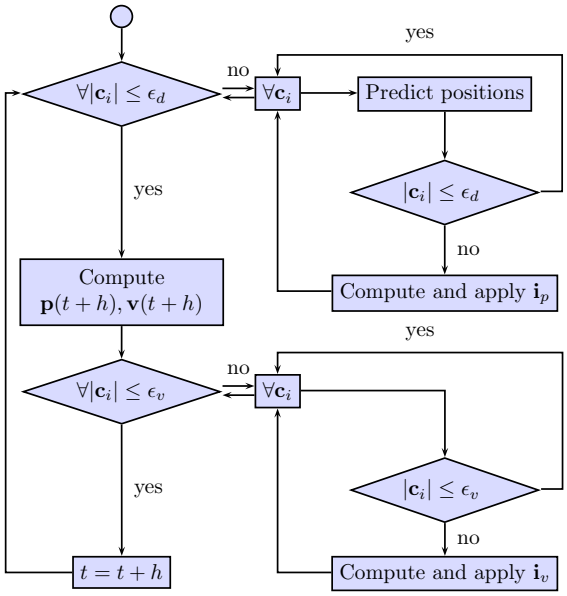

Figure 1: *Scheme of the impulse-based dynamic simulation.*

3.3. Distance constraint

A distance constraint restricts the motion of two particles, so that their initial distance is conserved during the simulation. All other movement is not affected. Thus, exactly one degree of freedom along the connecting line is removed. The constraint is therefore one-dimensional. Such a constraint can be imagined as if the particles would be connected with a bar. As implicit function it can be described as follows:

$$c_{dist}(\mathbf{p}_1(t), \mathbf{p}_2(t)) = |\mathbf{p}_2(t) - \mathbf{p}_1(t)| - dist \ ,$$

where $\mathbf{p}_1(t)$ and $\mathbf{p}_2(t)$ are the positions of the connected particles, $dist$ is their initial distance and t is the time. The connected particles will always have the same distance, if this function returns zero during the whole simulation.

For the following sections the normalized connecting vector $\frac{(\mathbf{p}_2(t)-\mathbf{p}_1(t))}{|\mathbf{p}_2(t)-\mathbf{p}_1(t)|}$ is denoted as constraint direction $\mathbf{n}$.

With this notation the derivatives of the constraint function are:

$$\begin{aligned} \frac{\partial c_{dist}}{\partial \mathbf{p}} &= \mathbf{n} \ , \\ \frac{\partial c_{dist}}{\partial \mathbf{v}} &= \mathbf{0} \ , \\ \frac{\partial c_{dist}}{\partial t} &= \mathbf{n}(\mathbf{v}_2(t) - \mathbf{v}_1(t)) \ . \end{aligned}$$

The second derivative is zero because the constraint does not directly depend on velocities. With the first derivative we can compute the constraint's correction impulse as:

$$\mathbf{i}_p = \mathbf{n} \left[(\frac{1}{m_i} + \frac{1}{m_j})^{-1} c_{dist}(\mathbf{p}_1(t+h), \mathbf{p}_2(t+h)) \frac{1}{h} \right] \ .$$

The third derivative defines a constraint for the velocities of the connected particles:

$$\mathbf{n}(\mathbf{v}_2(t) - \mathbf{v}_1(t)) = 0 \ ,$$

which ensures that their relative velocity along the constraints direction is zero. To achieve higher accuracies, this constraint can be computed by the velocity constraint loop described in 3.2, but this is not necessarily required to get stable simulations.

To satisfy the velocity condition a correction impulse is computed by:

$$\mathbf{i}_v = \mathbf{n} \left[(\frac{1}{m_i} + \frac{1}{m_j})^{-1} \mathbf{n}(\mathbf{v}_2(t) - \mathbf{v}_1(t)) \right] \ .$$

3.4. Contact constraint

A contact constraint allows the affected bodies to move only away from each other (cf. [Bar94]). Like the distance constraint only one degree of freedom, along the contact normal, of the connected bodies is affected. Since only particles are discussed in this paper, they are expanded to sample contacts between them. We do this by introducing a radius r for each particle. Then a contact constraint between to particles can be formulated as:

$$c_{contact}(\mathbf{p}_1(t), \mathbf{p}_2(t)) = |\mathbf{p}_2(t) - \mathbf{p}_1(t)| - (r_1 + r_2) \ .$$

It is assured that the two connected particles will not penetrate, if this function is greater or equal to zero during the whole simulation.

The derivatives are the same as for the distance constraint. As well an unilateral version of the distance constraint can be used to create a constraint that behaves like a rope. To do this, c_{dist} is multiplied by -1 to obtain $c_{rope} = -c_{dist} \geq 0$. Such a constraint would ensure that the distance between two particles will not exceed *dist*, figuratively spoken as if they were connected by a rope of length *dist*. The only difference in handling such unilateral constraints using the impulse-based dynamic simulation is, that the correction impulses are only applied if the constraint error is below $-\varepsilon$.

4. Optimized impulse-based dynamic simulation

The following sections describe how the inner loop, meaning one iteration, is optimized. Therefore, we recall that in each iteration, for every constraint, the preview and if needed a correction impulse is computed (cf. 3.2). Depending on the number of connected bodies this can take a lot of computational time. Because this time is needed for each iteration, optimizations of this computation promises a big speedup to the overall process, which is proved in the results section 5.

In the following sections the optimizations are described with the distance constraint as showcase. If not mentioned otherwise all information can be transformed to the contact constraint by setting *dist* to $(r_1 + r_2)$.

4.1. Alternative constraint formulations

As seen in 3.3 the normalized constraint direction $\mathbf{n}$ is needed to apply the correction impulses of a constraint. This involves the computation of the length between the connected particles which is also part of the computation of the constraint error. It is quite obvious that this double computation is unnecessary. Mathematically spoken we can reformulate the constraint functions from 3.3 by reusing the vector $\mathbf{n}$:

$$\begin{aligned} c_{dist} &= \mathbf{n}(\mathbf{p}_2(t) - \mathbf{p}_1(t)) - dist \quad \text{and} \\ c_{contact} &= \mathbf{n}(\mathbf{p}_2(t) - \mathbf{p}_1(t)) - (r_1 + r_2) \ . \end{aligned}$$

Using these reformulated functions result in the same simulation, but due to the avoidance of additional square root computations the runtime is greatly enhanced.

4.2. Anticipation of constraint errors

The default procedure of the impulse-based dynamic simulation is to process one constraint after the other. For each constraint, for both of the connected particles, the preview of their positions is computed. Based on this new positions the constraint function is evaluated to get the constraint error (cf. 3.2). Even if only a few vector operations are needed to obtain the new constraint error this implies some unnecessary computations.

Therefore, we propose a new efficient method to compute the correction impulses by anticipating the constraint errors of the other connected constraints. To do this, the constraint error of all constraints is computed prior to the first iteration. This can be done efficiently during the computation of the constraint direction for each constraint. At this point the preview position of each particle in the absence of any constraint is computed. In doing so we avoid to compute the same particle more than once by saving the time step of the last update. Since the constraint direction has to be normalized, the initial constraint error e can be directly computed without any further operations:

$$\begin{aligned} \mathbf{n} &= (\mathbf{a} - \mathbf{b}) \\ l &= |\mathbf{n}| \\ e &= l - dist \\ \mathbf{n} &= \frac{1}{l}\mathbf{n} \ , \end{aligned}$$

with the previewed positions $\mathbf{a} = \mathbf{p}_a(t+h)$ and $\mathbf{b} = \mathbf{p}_b(t+h)$ of the two connected particles a, b and their initial distance *dist*. During the further processing we update this error instead of computing it again in every step. Hence, we need only to test if $e > \varepsilon$ at the beginning of each joint correction. If this is the case, a correction has to be done. But we do not compute the correction impulse $\mathbf{i}$ explicitly, we rather save the velocity change as scalar magnitude v' based on the error. If the maximum impulse per correction is not limited, this value can be computed by:

$$v' = v' + (\frac{1}{m_a} + \frac{1}{m_b})^{-1} \cdot \frac{e}{h}$$

with the masses m_a and m_b of the connected particles. To avoid unnecessary calculus, we do not divide by the time step size h and save the position change directly: The error itself simply can be set to zero, because this change of the positions completely eliminates it.

$$\begin{aligned} p' &= p' + (\frac{1}{m_a} + \frac{1}{m_b})^{-1} \cdot e \\ e &= 0 \end{aligned}$$

This is not the case, if the impulse is limited by $i_{max} = p'_{max}/h$ and the error is greater than ε/h. In this case the following two equations yield the new error and position change magnitude:

$$\begin{aligned} p' &= p' + (\frac{1}{m_a} + \frac{1}{m_b})^{-1} \cdot p'_{max} \\ e &= e - (\frac{1}{m_a} + \frac{1}{m_b})^{-1} \cdot p'_{max} \ . \end{aligned}$$

Furthermore, we anticipate the error of the additional constraints connected to one of the both connected bodies during the correction of this constraint. This can be done very efficiently in the following way.

Without loss of generality let a be one of the connected bodies of constraint c_i and C_a the set of all constraints connected to a with common body a. Then for every $c_j \in C_a$

with $c_j \neq c_i$ the error e_j is computed by:

$$e_j = -dist_j + \mathbf{n}_j(\mathbf{a}_j - \mathbf{b}_j) \ .$$

Since only one constraint is allowed to connect the same bodies and $\mathbf{a}_j = \mathbf{a}_i$, $\mathbf{b}_j$ remains unchanged. But $\mathbf{a}_j$ is modified through the correction of c_i by:

$$\mathbf{a}_j = \mathbf{a}_i + \mathbf{n}_i e_i \ ,$$

with 4.2 follows:

$$\begin{aligned} e'_j &= -dist_j + \mathbf{n}_j((\mathbf{a}_i + \mathbf{n}_i e_i) - \mathbf{b}_j) \\ &= -dist_j + \mathbf{n}_j(\mathbf{a}_j - \mathbf{b}_j) + \mathbf{n}_j\mathbf{n}_i \cdot e_i \\ &= e_j + \mathbf{n}_j\mathbf{n}_i \cdot e_i \ . \end{aligned}$$

Since $\mathbf{n}_j\mathbf{n}_i$ is constant for at least one iteration this dot product can be precomputed during the initial computation of the constraint direction. Therefore, the new error of the additional constraints can be anticipated by a single multiplication.

At the end of all iterations the position change magnitude can be used to obtain the new particles' positions and velocities. For a particle i with the set of constraints C_i and external forces $\mathbf{F}$, this is done by the following equations:

$$\begin{aligned} \mathbf{p}' = \forall c_j \ \in \ C_i : &\begin{cases} \mathbf{n}_j \cdot p'_j & ;i \text{ is 1st body of } c_j \\ -\mathbf{n}_j \cdot p'_j & ;i \text{ is 2nd body of } c_j \end{cases} \\ \mathbf{p}_i(t+h) &= \mathbf{p}_i(t) + \mathbf{p}' \\ \mathbf{v}_i(t+h) &= \mathbf{v}_i(t) + 1/h \cdot \mathbf{p}' + h \cdot \mathbf{F} \ . \end{aligned}$$

Velocity constraints can be handled in the same manner, with the only exceptions that the constraint error e is not divided by the step size h and the positions of the particles are not altered.

The use of this anticipated error and optimized impulse computation decreases the needed computational time as section 5 shows. Especially, if the constraint directions do not change between the iterations the runtime is greatly lowered. As section 5 shows simulations can be computed more than two times faster than without this optimization.

If the constraint direction is changed between every iteration (cf. 4.4), the constraint errors have only to be anticipated for constraints that have not yet been computed in this iteration, because the error in the next iteration will be reseted according to the new constraint direction. Even if the constraint direction is changed in every iteration, the proposed method decreases the runtime up to 43% in the measured scenarios. Furthermore, it enables an efficient way to implement the optimization described in the next section allowing further acceleration of the computation. This all comes with no loss, which means the result of the simulation is not changed.

4.3. Skipping satisfied constraints

The original simulation procedure visits every constraint in every iteration to test if the constraint error is below a certain tolerance ε. But at an average only about 50% of the constraints have to be corrected. This implies a lot of wasted computational time, especially, if the optimization described in chapter 4.2 is not used. Therefore, we present two ways how this unnecessary computations can be avoided.

The basic idea is to keep track of which constraints are satisfied and which have to be corrected. For that, every time the velocity of one particle is changed, the attached constraints are checked whether they have to be corrected or not. By the use of the anticipation optimization described in section 4.2 this can be done very easily by a reminder in each constraint. A simple boolean variable is used to save whether the constraint needs to be corrected or not. This variable is initially set during the initial constraint error computation described in 4.2. Afterwards the reminder is updated during the anticipation computation of the attached constrain errors. This comes with nearly no additional costs, because the new errors are computed anyway. In an iteration, we then simply skip every constraint which is marked as satisfied and correct the other ones without further comparisons.

If the optimization proposed in 4.2 is not used, the costs of this procedure exceeds its benefit. But with the use of the following optimized method skipping constraint corrections still can be advantageous. The trick is to save the reminder not in every constraint, but in every body. Therefore, we use one unsigned integer per particle. Every constraint is represented by one bit of this variable. At first, every constraint is marked as unsatisfied by setting each reminder to -1. Then, for every constraint the reminder is requested by simple boolean operations. If the bit corresponding to this constraint is set in one of the both connected bodies the constraint correction is executed. If the error of this constraint is indeed above ε, the error is corrected and every other constraint is marked unsatisfied. Otherwise we delete only the corresponding bit, marking this constraint as satisfied. The use of this method avoids iterating over all other connected constraints by the smart use of bit operations. This significantly reduces the overhead generated and makes this method also feasible for smaller models. However, a slight limitation of this method is that one particle can only have up 64 constraints.

Nevertheless, by the use of the optimization described in section 4.2 this overhead is not created at all, since the constraints are processed anyway to anticipate the errors.

The result section 5 shows the speedup of this two methods. It demonstrates that the skipping of satisfied constraints enhances the runtime of the anticipated simulation. But since the computation is already optimized the speedup is not so big compared to the just anticipated simulation. On the contrary, the second proposed method brings a larger advancement compared to the original simulation, because many ex-

pensive computations can be ignored. But if the constraint direction is changed in every iteration, the overhead for marking the constraints as unsatisfied may exceed the profit resulting in a larger overall runtime.

As the method described in 4.2, this optimization does not change the simulations result.

4.4. Avoiding changes of the constraint direction

The main bottleneck in the constraint computation are the changes of the constraint directions. They create effort because the constraint directions have to be normalized after each change. Furthermore, they also lower the benefit of the optimizations described in chapters 4.2 and 4.3. The directions have to be recomputed after each iteration to get absolutely correct results, because the positions of the particles are permanently changing.

If the constraint direction is not updated, it can only be guaranteed that the constraint error projected onto the initial direction $\mathbf{n}_{init}$ is below ε. For the distance constraint this means

$$\mathbf{n}_{init}(\mathbf{p}_j(t+h) - \mathbf{p}_i(t+h)) \leq \varepsilon_d$$

but not necessarily

$$|\mathbf{p}_j(t+h) - \mathbf{p}_i(t+h)| \leq \varepsilon_d \quad .$$

In theory the error normal to $\mathbf{n}$ can not be limited under this circumstances. This error depends on the acting forces, the step-size and the complexity of the scene. Nevertheless, in praxis with limited forces and adequate step-sizes and scene complexities, fast and at least visual plausible simulations can be achieved anyway.

To get stable simulations under any circumstances we propose a method which still avoids the majority of constraint direction changes. This is done by first driving the constraints errors below ε along the initial direction. After this is done we update the constraint directions. Then we start a new run and reduce the errors along the new directions. This process is repeated till directly after a direction update within the first iteration all errors are again below ε. With this procedure we can ensure that for all constraints the error $|\mathbf{p}_j(t+h) - \mathbf{p}_i(t+h)|$ is below ε, because in the last iteration $\mathbf{n}$ points along the direct connecting line.

This procedure allows to simulate models with very high complexity under the influence of large forces, even by using high step-sizes. In general, more iterations are needed for the same result, but since single iterations are computed much faster the overall runtime is less, as the results section 5 shows.

5. Results

In this section the results of the different optimization techniques proposed in this paper are presented. To measure these optimizations, a rectangular piece of cloth connected by distance constraints was simulated.

The cloth was deflected, so that it was parallel to the floor and then released. In the first and second simulation the particles in the left row were fixed, so that the cloth swings around an axis. In the first case the cloth moved freely and in the second it was colliding with a fixed sphere. The third simulation consisted of a piece of cloth with fixed particles on the top and on the bottom side with a very heavy sphere dropped upon it.

In all simulations the cloth had a length and width of $1m$, the time step size was $0.01s$ and the gravity set to $10 m/s^2$. They were computed with a maximum extensibility of 0.001%, 1% and 10% and a dimension of 20×20 connected particles. For the contact constraints $\varepsilon_d = 0.1$ and for the velocity correction $\varepsilon_v = 0.01$ was chosen. In all measurements $2s \mathrel{\widehat{=}} 200$ time steps were computed.

Figure 5 illustrates the three different simulations.

Figure 2 shows the benefit of the optimizations described in sections 4.2 and 4.3. For that, the runtime proportional to the unoptimized simulation in percent is displayed. For the unoptimized simulation as well as for the optimized ones the alternative constraint formulation from section 4.1 was used. As figure 2 shows, the average runtime proportional to the original method with skipping of the satisfied constraints and without anticipation is 103%. Thus, it is about 3% slower than the original method. This is the case because the time to reset the remainder in every iteration exceeded the gain of skipping unnecessary computations of the constraint function. Nevertheless, with activated velocity correction the average runtime decreased about 5%. Above all others, the third simulation with a strain of 10% benefits from this optimization. It could be computed about 14% faster.

With anticipation all simulations ran faster. The average runtime was 72% without, and 66% with velocity correction of the original method's runtime. Thus, at an average, the simulations ran about 31% faster. The best value with a speedup of 45% was reached by the third simulation with an extensibility of 10% and velocity correction. The worst improvement was measured in the second simulation with velocity correction and 0.001% strain, which needed 76% of the original runtime.

Further advancements were achieved with anticipation and skipping. The average runtime was 66% without, and 59% with velocity correction of the original runtime. The runtime improved at an average of about 38%. Again the third simulation with 10% strain and velocity correction could be improved most with 57%. The second simulation with velocity correction and 0.001% strain produced the worst improvement with 28%.

Figure 3 displays the same measurements with the exception that the constraint directions were only updated after

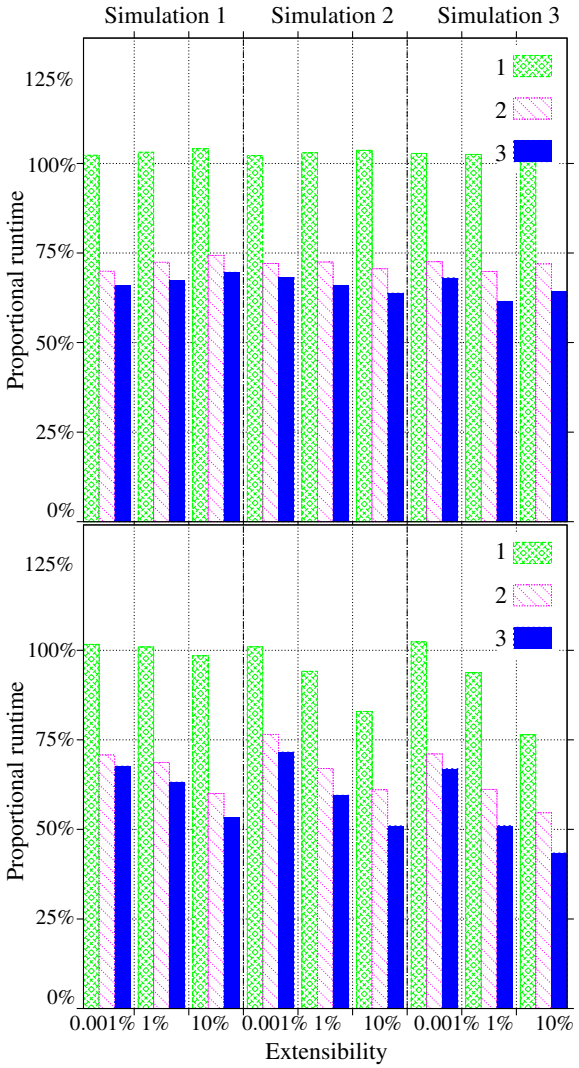

Figure 2: *Runtime acceleration of the optimizations in percent, compared to the original simulation, with (bottom) and without (top) velocity correction. The constraint directions were changed in every iteration. The first dataset is optimized with the constraint skipping described in section 4.3, the second with the constraint anticipation described in section 4.2 and the third with constraint anticipation and constraint skipping.*

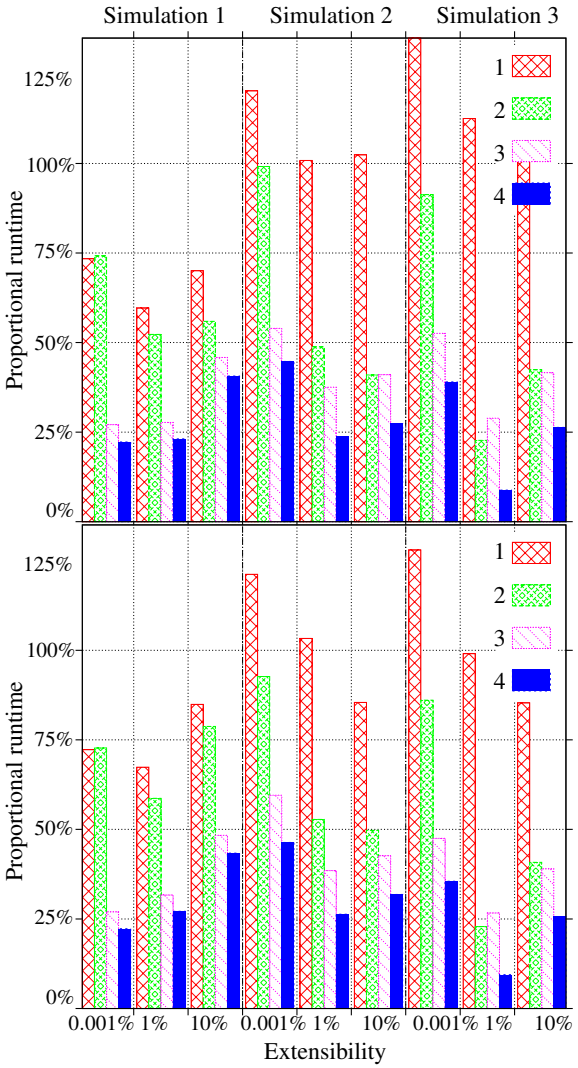

Figure 3: *Runtime acceleration of the optimizations in percent, compared to the original simulation, with (bottom) and without (top) velocity correction. The constraint directions were changed according to section 4.4. All datasets use the approach described in section 4.4. The first dataset is the original computation, the second is optimized with the constraint skipping described in section 4.3, the third with the constraint anticipation described in section 4.2 and the fourth with constraint anticipation and constraint skipping.*

all constraints were completely satisfied along their last direction (as described in section 4.4). Again the values are proportional to the original version with changing of the directions in each iteration. Additionally the unoptimized version is shown with the directions changed according to section 4.4. The figure shows that the avoidance of changes of the constraint direction is not always faster. This is the case because the number of overall iterations rise. But with unchanged directions the constraint error does not have to be recomputed, so that the optimizations have a greater effect. In any case and for all optimizations the additional benefit exceeded the loss generated by the larger number of iterations. Hence, the overall runtime of every simulation could be improved.

With skipping of the satisfied constraints and without an-

ticipation the average runtime is about 59% without, and 62% with velocity correction. This is an improvement about 40%. The best value achieved the third simulation with 1% strain and without velocity correction with a runtime improvement of 78%. However, the runtime of the third simulation with 0.001% strain could be hardly improved about 1%.

With anticipation the average runtime was 40% without, and 40% with velocity correction compared to the original runtime. Interestingly, the best value with an improvement of 73%, achieved in the same simulation, is less than the best value achieved by the method without anticipation. This shows that the avoidance of unnecessary constraint corrections gains importance as the number of iterations rise. Nevertheless, the worst value with an improvement of 40% is again much faster than the corresponding value without anticipation.

The combination of all three techniques show the best results. At an average, the runtime was 28% without, and 30% with velocity correction compared to the original runtime. This is an average improvement of 71%, more than three times faster as the unoptimized version. The best value achieved simulation three with a maximum extensibility of 1%. This simulation took only 9% of the original runtime and therefore could be simulated more than 11 times faster.

Table 5 summarizes the best, worst and average runtime improvements of the different optimizations. Figure 4

	Times faster		
Optimization	Best	Worst	Average
4.3	1.31	0.96	1.01
4.2	1.83	1.31	1.46
4.2 & 4.3	2.30	1.40	1.60
4.3 & 4.4	4.40	1.01	1.66
4.2 & 4.4	3.75	1.68	2.51
4.2 & 4.3 & 4.4	11.35	2.16	3.44

Table 1: *Summary of the runtime improvements measured. The shown values are proportional to the runtime of the unoptimized simulation.*

shows the absolute runtime of the first scenario with different details. For that, grids with $5 \times 5 = 25$, $10 \times 10 = 100$, $20 \times 20 = 400$, $30 \times 30 = 900$ and $40 \times 40 = 1600$ connected particles were simulated. The figure shows that the proposed optimizations are also applicable for systems with less complexity. It also shows, that a piece of cloth with high complexity of 40×40 could be simulated in real time, however only with a maximum extensibility of 10%. But even with a extensibility of 1% a cloth containing 900 particles could be simulated in realtime. It is also shown, that the accuracy could be increased by the use of the same computational time. For example, the optimized runtime needed to compute a cloth with 1% nearly equaled the unoptimized computation time needed for a cloth with maximum extensibiliy of 10%.

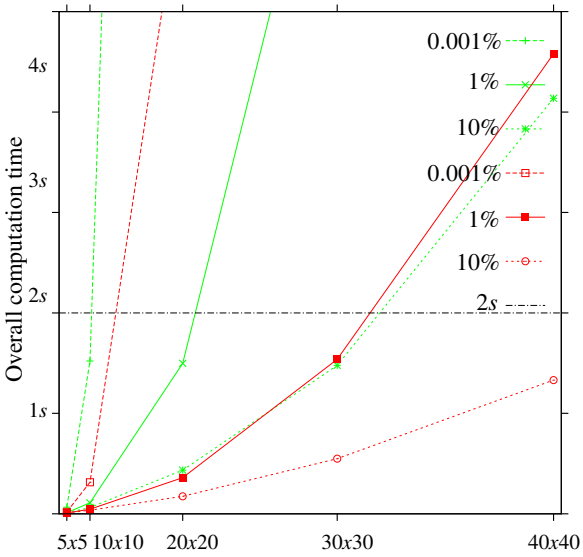

Figure 4: *Absolute runtime of the first scenario with different details and extensibility using the optimized and original simulation. The first three data sets correspond to the original method and the last three to the fully optimized method.*

5.1. Conclusion

The proposed optimizations greatly enhance the runtime of the impulse-based dynamic simulation as section 5 shows. Since they do not affect the overall simulation, this advantage comes without loss in quality. Thus, larger systems can be simulated. On the other hand, the optimizations allow to compute more iterations within the same time and, therefore, given simulations can be computed with higher accuracies.

Especially, if the constraint directions are not updated between the iterations the proposed optimizations in sections 4.2 and 4.3 show their best effect. With the method described in section 4.4, stable simulations with predictable error can be computed with large step sizes and under the influence of large forces without changing the constraints direction in every time step. Therefore, a larger number of iterations is needed, but with the optimizations in sections 4.2 and 4.3 the overall runtime is decreased by up to 92%. Even if the directions are changed in every iteration the simulation still benefits from this optimizations.

The proposed optimizations also have the potential to further optimize the impulse-based dynamic simulation - especially in connection with the linear time dynamics proposed in [BB08].

References

[Bar94] BARAFF D.: Fast contact force computation for nonpenetrating rigid bodies. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1994), ACM, pp. 23–34. 3

[BB08] BENDER J., BAYER D.: Parallel simulation of inextensible cloth. In *Virtual Reality Interactions and Physical Simulations (VRIPhys)* (Grenoble (France), Nov. 2008). 2, 8

[BBD09] BAYER D., BENDER J., DIZIOL R.: Impulse-based dynamic simulation on the gpu. In *Computer Graphics, Visualization, Computer Vision and Image Processing (CGVCVIP2009) - IADIS Multi Conference on Computer Science and Information Systems* (Algarve, Portugal, Portugal, 2009). 2

[BS06] BENDER J., SCHMITT A.: Fast dynamic simulation of multi-body systems using impulses. In *Virtual Reality Interactions and Physical Simulations (VRIPhys)* (Madrid (Spain), Nov. 2006), pp. 81–90. 2

[BW98] BARAFF D., WITKIN A.: Large steps in cloth simulation. *Computer Graphics 32*, Annual Conference Series (1998), 43–54. 2

[HCJ*05] HONG M., CHOI M.-H., JUNG S., WELCH S., TRAPP J.: Effective constrained dynamic simulation using implicit constraint enforcement. In *International Conference on Robotics and Automation* (Apr 2005). 2

[HES03] HAUTH M., ETZMUSS O., STRASSER W.: Analysis of numerical methods for the simulation of deformable models. *The Visual Computer 19*, 7-8 (2003), 581–600. 2

[KC02] KANG Y.-M., CHO H.-G.: Bilayered approximate integration for rapid and plausible animation of virtual cloth with realistic wrinkles. In *CA '02: Proceedings of the Computer Animation* (Washington, DC, USA, 2002), IEEE Computer Society, p. 203. 2

[MHTG05] MÜLLER M., HEIDELBERGER B., TESCHNER M., GROSS M.: Meshless deformations based on shape matching. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers* (New York, NY, USA, 2005), ACM, pp. 471–478. 2

[OGRG07] OTADUY M. A., GERMANN D., REDON S., GROSS M.: Adaptive deformations with fast tight bounds. In *SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation* (Aire-la-Ville, Switzerland, Switzerland, 2007), Eurographics Association, pp. 181–190. 2

[SBP05] SCHMITT A., BENDER J., PRAUTZSCH H.: *On the Convergence and Correctness of Impulse-Based Dynamic Simulation*. Internal Report 17, Institut für Betriebs- und Dialogsysteme, 2005. 2

[SSBT08] STUMPP T., SPILLMANN J., BECKER M., TESCHNER M.: A geometric deformation model for stable cloth simulation. In *Virtual Reality Interactions and Physical Simulations (VRIPhys)* (Grenoble (France), Nov. 2008). 2

[TPBF87] TERZOPOULOS D., PLATT J., BARR A., FLEISCHER K.: Elastically deformable models. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1987), ACM, pp. 205–214. 1

[VT00] VOLINO P., THALMANN N. M.: Implementing fast cloth simulation with collision response. In *CGI '00: Proceedings of the International Conference on Computer Graphics* (Washington, DC, USA, 2000), IEEE Computer Society, p. 257. 2

[Zel05] ZELLER C.: Cloth simulation on the gpu. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Sketches* (New York, NY, USA, 2005), ACM, p. 39. 2

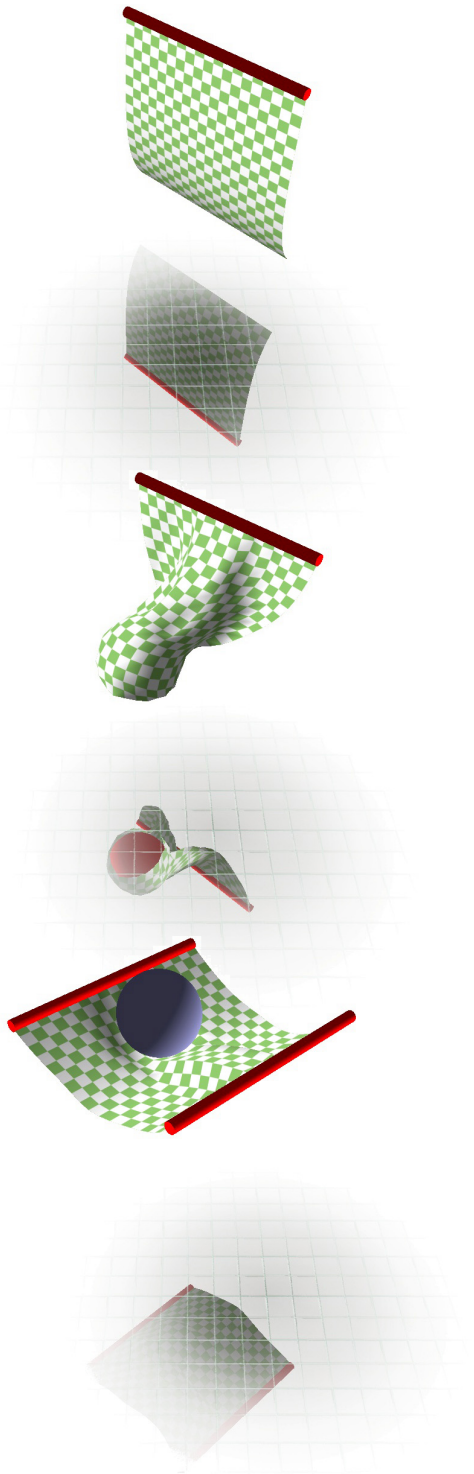

Figure 5: *The optimizations were measured with three different simulations, each with a maximum strain of* 0.001%, 1% *and* 10%.