Workshop on Virtual Reality Interaction and Physical Simulation VRIPHYS (2008)
F. Faure, M. Teschner (Editors)

Hierarchical Position Based Dynamics

Matthias Müller

NVIDIA

Abstract

The Position Based Dynamics approach (PBD) recently introduced allows robust simulations of dynamic systems in real time. The simplicity of the method is due to the fact, that the solver processes the constraints one by one in a Gauss-Seidel type manner. In contrast to global Newton-Raphson solvers, the local solver can easily handle non-linear constraints as well as constraints based on inequalities. Unfortunately, this advantage comes at the price of much slower convergence.
In this paper we propose a multi-grid based process to speed up the convergence of PBD significantly while keeping the power of the method to process general non-linear constraints. Several examples show that the new approach is significantly faster than the original one. This makes real time simulation possible at a higher level of detail in interactive applications such as computer games.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object ModelingPhysically Based Modeling; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and RealismAnimation and Virtual Reality

1. Introduction

Simulation methods need to meet four major requirements to be applicable in computer games. They must be fast, stable, controllable and easy to code in order for game developers to pick them up and use them in their projects. There is still a significant gap between the academic research community and game developers today. This gap seems to get even wider because simulation techniques in computer graphics get more and more mathematically involved as they get closer to methods used in computational sciences. This increase in complexity is necessary to achieve all the astonishing effects we see in movies today.

In contrast, a majority of the physics in computer games is still rigid body dynamics. Other effects like water, cloth or soft bodies are mainly done with procedural approaches because procedural methods are computationally cheap and cannot get unstable. The Position Based Dynamics approach recently introduced by [MHR06] is an attempt to bridge this gap. It generalizes and extends the method proposed by [Jak01]. A Verlet-based integrator is used which bypasses the force and velocity layers and directly modifies the positions of particles or vertices of a mesh. These modifications are computed using a non-linear Gauss-Seidel type solver. The main advantages of the approach are its simplicity, unconditional stability, its ability to handle non-linear unilateral (inequality) and bilateral (equality) constraints directly and the possibility of manipulating positions which gives the user a high level of control over the simulation process.

Unfortunately, Gauss-Seidel type solvers have one significant drawback. Because they handle constraints individually one by one, information propagates slowly through a mesh. The slow convergence lets cloth or soft bodies look stretchy, especially when high resolution meshes are used. This is definitely an undesirable effect which produces visual artifacts because in the real world the high deformability of cloth is

not an effect of a low stretching stiffness but is due to its low bending resistance.

Convergence can be significantly increased by using global solvers. The most popular method for solving non-linear systems of equations is Newton-Raphson iteration. This approach is both computationally expensive and complex to code because the system of equations has to be linearized and solved multiple times per simulation time step. While not practical for real-time application, it can handle stiff equations effectively in off line applications as [GHF*07] show. In order for the method to be global, each linear system has be be solved using a global linear solver like Preconditioned Conjugate Gradients (PCG). There is no easy way of extending PCG to handle unilateral constraints because they turn the system into a quadratic programming or linear complementarity problem [Ebe04]. Multigrid methods are an important alternative to (PCG). They are designed for linear system as well and cannot handle inequalities either because it is not clear how to propagate these to higher hierarchy levels.

One nice feature of the Multigrid approach is the fact that each level is typically solved by a simple Jacobi or Gauss-Seidel iteration. The core idea of our new approach is to use this fact and replace the linear Gauss-Seidel solver by the non-linear variant used in PBD which results in a non-linear Multigrid solver. There are several questions arising with this approach that we answer in this paper: How to restrict unilateral constraints to coarser meshes, how to translate non-linear constraint functions to coarser levels or how to prolongate the solutions back to finer levels. Thus, our main contribution is a method to turn the non-linear Gauss-Seidel Solver of (PBD) into a non-linear Multigrid based algorithm thereby significantly increasing the speed of convergence of the PBD approach while keeping all its nice features.

2. Related Work

In this paper we focus on the simulation of deformable objects. The state of the art reports [GM97] and [NMK*05] give a comprehensive overview of methods used in this field of research.

Since the early work of [TPBF87] in the 80's, many techniques have been proposed to simulate deformable solids and cloth. While early simulations were done as offline computations, the computing power available today allows for real-time simulation of such effects. One of the first systems that made real-time interaction with deformable objects possible was *ArtDefo* described in [JP99]. An effective way to speed up the simulation of deformable objects is to use reduced models such as in [HSO03] and [BJ07]. [MG04] worked with all the degrees of freedom but reduced the non-linear governing equations to a linear system while preventing artifacts from large rotational deformations. Our method also targets real-time simulation although there is nothing that prevents it from being used in hight quality off-line settings with higher resolution models.

As an extension of the PBD approach described in [Jak01] and generalized in [MHR06] our method is based on constraint projection. This idea has been used in cloth simulation by [BFA02] to resolve collisions geometrically and by [VCMT95] in their kinematic collision correction step. Desbrun [DSB99], Provot [Pro95] and [GHF*07] used constraint projection in mass spring systems to prevent springs from overstretching. For the efficient animation of deformable solids [Fau98] represent object by linearized displacement constraints. For the stable simulation of soft bodies [MHTG05] project the positions of particles onto a configuration obtained via shape matching of original to current positions.

The Multigrid method has been used successfully in computational sciences for many decades (see [McC87] for an introduction). Recently, it has been introduced to computer graphics, mainly in connection with the Finite Element Method such as in [DDCB01,CGC*02,WT04,GW06]. To solve the linear Poisson equation arising in incompressible fluid simulation [CFL*07] use an algebraic Multigrid method while [OGRG07] leverage the hierarchical nature of the method to adaptively simulate deformations.

The latter paper uses the Multigrid approach to locally refine objects where stresses are high. In contrast, our approach is able to generate fine detail *everywhere* in the object. Neither locations of refinement nor the number of hierarchy levels used change during the simulation which prevents visual artifacts typically produced by adaptive simulations.

3. Position Based Simulation

Since we propose an acceleration technique for the Position Based Dynamics approach and because we want to keep the paper as self contained as possible we briefly summarize the core ideas of PBD. The objects to be simulated are represented by a set of N particles and a set of M constraints. Each particle i has three attributes, namely

m_i	mass
$\mathbf{x}_i$	position
$\mathbf{v}_i$	velocity

Table 1: *Attributes of a particle*

A constraint j is defined by the five attributes shown in Table 2. With type *bilateral* is satisfied if $C_j(\mathbf{x}_{i_1},\ldots,\mathbf{x}_{i_{n_j}}) = 0$. If its type is *unilateral* then it is satisfied if $C_j(\mathbf{x}_{i_1},\ldots,\mathbf{x}_{i_{n_j}}) \geq 0$. The stiffness parameter k_j defines the strength of the constraint in a range from zero to one.

n_j	cardinality
$C_j : \mathbb{R}^{3n_j} \rightarrow \mathbb{R}$	scalar constraint function
$\{i_1, \ldots i_{n_j}\}$, $i_k \in [1, \ldots N]$	set of indices
$k_j \in [0 \ldots 1]$	stiffness parameter
unilateral or *bilateral*	type

Table 2: *Attributes of a constraint*

Given this data and a time step Δt, the simulation proceeds as follows:

(1) **forall** particles i
(2) initialize $\mathbf{x}_i = \mathbf{x}_i^0, \mathbf{v}_i = \mathbf{v}_i^0, w_i = 1/m_i$
(3) **endfor**
(4) **loop**
(5) **forall** particles i **do** $\mathbf{v}_i \leftarrow \mathbf{v}_i + \Delta t w_i \mathbf{f}_{ext}(\mathbf{x}_i)$
(6) **forall** particles i **do** $\mathbf{p}_i \leftarrow \mathbf{x}_i + \Delta t \mathbf{v}_i$
(7) **forall** particles i **do** generateCollisionConstraints($\mathbf{x}_i \rightarrow \mathbf{p}_i$)
(8) **loop** solverIterations **times**
(9) projectConstraints($C_1, \ldots, C_{M+M_{coll}}, \mathbf{p}_1, \ldots, \mathbf{p}_N$)
(10) **endloop**
(11) **forall** particles i
(12) $\mathbf{v}_i \leftarrow (\mathbf{p}_i - \mathbf{x}_i)/\Delta t$
(13) $\mathbf{x}_i \leftarrow \mathbf{p}_i$
(14) **endfor**
(15) **endloop**

Since the algorithm simulates a system which is second order in time, both the positions and the velocities of the particles need to be specified in (1)-(3) before the simulation loop starts. Lines (5)-(6) perform a simple explicit forward Euler integration step on the velocities and the positions. The new locations $\mathbf{p}_i$ are not assigned to the positions directly but are only used as predictions. Non-permanent external constraints such as collision constraints are generated at the beginning of each time step from scratch in line (7). Here the original and the predicted positions are used in order to perform continuous collision detection. The solver (8)-(10) then iteratively corrects the predicted positions such that they satisfy the M_{coll} external as well as the M internal constraints. Finally the corrected positions $\mathbf{p}_i$ are used to update the positions *and* the velocities. It is essential here to update the velocities along with the positions. If this is not done, the simulation does not produce the correct behavior of a second order system.

4. The System to be solved

In the rest of the paper we will focus on the solver (8)-(10) because this is the part of the algorithm where almost all of the simulation time is spent. Let us first look at the original Gauss-Seidel Solver proposed in [MHR06].

The goal of the solver step is to correct the predicted positions $\mathbf{p}_i$ of the particles such that they satisfy all constraints. The problem that needs to be solved comprises of a set of M equations for the $3N$ unknown position components, where M is now the total number of constraints. This system does not need to be symmetric. If $M > 3N$ ($M <$3N) the system is over-determined (under-determined). In addition to the asymmetry, the equations are in general non-linear. The function of a simple distance constraint $C(\mathbf{p}_1, \mathbf{p}_2) = (\mathbf{p}_1 - \mathbf{p}_2)^2 - d^2$ yields a non-linear equation. What complicates things even further is the fact that collisions produce inequalities rather than equalities. Solving a non-symmetric, non-linear system with equalities and inequalities is a tough problem.

Let $\mathbf{p}$ be the concatenation $[\mathbf{p}_1^T, \ldots, \mathbf{p}_N^T]^T$ and let all the constraint functions C_j take the concatenated vector $\mathbf{p}$ as input while only using the subset of coordinates they are defined for. We can now write the system to be solved as

$$\begin{aligned} C_1(\mathbf{p}) &\succ 0 \\ &\ldots \\ C_M(\mathbf{p}) &\succ 0, \end{aligned}$$

where the symbol $\succ$ denotes either $=$ or $\geq$. Newton-Raphson iteration is a method to solve non-linear symmetric systems with only equalities. The process starts with a first guess of a solution. Each constraint function is then linearized in the neighborhood of the current solution using

$$C(\mathbf{p} + \Delta\mathbf{p}) = C(\mathbf{p}) + \nabla_{\mathbf{p}} C(\mathbf{p}) \cdot \Delta\mathbf{p} + O(|\Delta\mathbf{p}|^2) = 0. \quad (1)$$

This yields a *linear* system for the global correction vector $\Delta\mathbf{p}$

$$\begin{aligned} \nabla_{\mathbf{p}} C_1(\mathbf{p}) \cdot \Delta\mathbf{p} &= -C_1(\mathbf{p}) \\ &\ldots \\ \nabla_{\mathbf{p}} C_M(\mathbf{p}) \cdot \Delta\mathbf{p} &= -C_M(\mathbf{p}), \end{aligned}$$

where $\nabla_{\mathbf{p}} C_j(\mathbf{p})$ is the $1 \times N$ dimensional vector containing the derivatives of the function C_j w.r.t. all its parameters, i.e. the N components of $\mathbf{p}$. It is also the jth row of the linear system. Both the rows $\nabla_{\mathbf{p}} C_j(\mathbf{p})$ and the right hand side scalars $-C_j(\mathbf{p})$ are constant because they are *evaluated* at the location $\mathbf{p}$ before the system is solved. When $M = 3N$ and only equalities are present, the system can be solved by any linear solver, e.g. PCG. Once it is solved for $\Delta\mathbf{p}$ the current solution is updated as $\mathbf{p} \leftarrow \mathbf{p} + \Delta\mathbf{p}$. A new linear system is generated by evaluating $\nabla_{\mathbf{p}} C_j(\mathbf{p})$ and $-C_j(\mathbf{p})$ at the new location after which the process repeats.

If $M \neq 3N$ the resulting matrix of the linear system is non-symmetric and not invertible. [GHF*07] solve this problem by using the pseudo-inverse of the system matrix which yields the best solution in the least-squares sense. Still, handling inequalities is not possible directly.

5. The Non-Linear Gauss-Seidel Solver

The non-linear Gauss-Seidel solver of PBD solves each constraint equation separately. Each constraint yields a single scalar equation $C(\mathbf{p}) \succ 0$ for all the particle positions associated with it. The subsystem is therefore highly under-determined. PBD solves this problem as follows. Again, given $\mathbf{p}$ we want to find a correction $\Delta\mathbf{p}$ such that $C(\mathbf{p}+\Delta\mathbf{p}) = 0$. It is important to notice that PBD also linearizes the constraint function but individually for each constraint. The constraint equation is approximated by

$$C(\mathbf{p}+\Delta\mathbf{p}) \approx C(\mathbf{p}) + \nabla_{\mathbf{p}} C(\mathbf{p}) \cdot \Delta\mathbf{p} = 0. \tag{2}$$

The problem of the system being under-determined is solved by restricting $\Delta\mathbf{p}$ to be in the direction of $\nabla_{\mathbf{p}} C$ which conserves the linear and angular momenta. This means that only one scalar λ - a Lagrange multiplier - has to be found such that the correction

$$\Delta\mathbf{p} = \lambda \nabla_{\mathbf{p}} C(\mathbf{p}). \tag{3}$$

solves Eq. (2). This yields the following formula for the correction vector of a single particle i

$$\Delta\mathbf{p}_i = -s\, w_i \nabla_{\mathbf{p}_i} C(\mathbf{p}), \tag{4}$$

where

$$s = \frac{C(\mathbf{p})}{\sum_j w_j |\nabla_{\mathbf{p}_j} C(\mathbf{p})|^2} \tag{5}$$

and $w_i = 1/m_i$.

As mentioned above, this solver linearizes the constraint functions. However, in contrast to the Newton-Raphson method, the linearization happens individually *per constraint*. Solving the linearized constraint function of a single distance constraint for instance yields the correct result in a single step. Because the positions are immediately updated after a constraint is processed, these updates will influence the linearization of the next constraint because the linearization depends on the actual positions. Asymmetry poses no problem because each constraint produces one scalar equation for one unknown Lagrange multiplier λ. Inequalities are handled trivially by first checking whether $C(\mathbf{p}) \geq 0$. If this is the case, the constraint is simply skipped.

6. The Multi-Grid Solver

In this section we present our non-linear Multi-Grid solver. The crucial part is the construction of a data structure - a hierarchy of constrained particle systems. Once this data structure is constructed, the solution process is straight forward.

6.1. Hierarchical Particle Systems

The constrained particle system composed of the set of N particles and the set of M constraints represents the finest level 0 of the hierarchy. In specific settings, this level 0 particle system has a specific structure. In the case of cloth simulation it represents a triangle mesh while deformable solids are typically represented by tetrahedral meshes.

A particle system at a coarser level l contains a proper subset of the particles at level $l-1$. Thus, a particle present at level l is contained in all finer level particle systems $l-1 \ldots 0$. We define the level of a particle to be the coarsest level which contains it even though it is contained in all finer levels as well (see Fig. 5).

Two types of constraint are distinguished. Cardinality-1 constraints and cardinality-n constraints with $n > 1$. Each particle system in the hierarchy has its own set of cardinality-n constraints while cardinality-1 constraints are associated with their particles and are used in all levels of the hierarchy in which the particle is present.

We define two particles to be connected (or neighbors) at level l if they share at least one cardinality-n constraint on this level. To be able to propagate information from one level to the next, a level l particle must be connected to at least one level $l+1$ particle on level l. This must hold for all levels but the coarsest. We call the set of level $l+1$ particles connected to a level l particle the parents of the level l particle. Thus, all particles except the ones on the coarsest level need to have al least one parent (see Fig. 1).

This definition of the data structure solves one main problem brought up in the introduction, namely how to restrict unilateral constraints to coarser levels. In the simulation scenarios we target, the only unilateral constraints are collision constraints which typically have cardinality 1. Since cardinality-1 constraints are bound to the associated particle, they are simply processed as in PBD on each level the particle exists. The same is true for other cardinality-1 constraints such as attachments.

6.2. Particle Restriction

We first describe how the particle subsets on each level are selected. Given the particles of a fine level (level l say) and the connectivity on the fine level, one has to choose a proper subset of the particles to form the set of particles of coarse level $l+1$. Many choices are possible as long as the restriction is met that each fine particle is connected on the fine level to at least one coarse particle. As [CFL*07] note, this is the so-called *vertex or node cover problem* [Pap97] known to be NP-complete. Fortunately, there is no need to work with the optimal solution, a good suboptimal solution is fine too. The better the solution the faster the method becomes, though.

In [CFL*07] the authors use a heuristic greedy algorithm. They first mark all the particles as fine. Then, while traversing the mesh certain particles are marked as coarse. If the mesh is not connected the process has to be restarted for each

disconnected component. We propose a simpler algorithm that processes a disconnected mesh in one sweep. It can also handle a restricted version of the original constraint, namely the constraint that each fine vertex has at least k parents. For $k > 1$ the number of particles from fine to coarser levels decreases more slowly. This yields smoother error propagation but more projection work on coarser levels. We use $k = 2$ in our examples.

First all particles are marked as coarse. Each particle also stores the number of coarse neighbors. This value is initialized with the total number of neighbors. Then, all the particles are traversed in an arbitrary order. A particle is marked fine if two conditions are met. First, the number of its coarse neighbors must be greater or equal k and second, all the neighboring fine vertices must have strictly more than k coarse neighbors. If the particle is marked as fine, the number of coarse neighbors of all its neighbors is decreased by one.

The convergence of the solver is improved if attached particles are kept in the coarse levels of the hierarchy. However, keeping all of them in all levels makes coarser meshes unnecessarily dense. One way to solve this problem is to mark coarse attached vertices as fine only with a certain probability. An alternative solution is to handle attached vertices last because the probability that a particle is marked as fine decreases towards the end of the process.

To make the solver faster, each particle p_i stores pointers to its parents p_j on the next coarser level plus corresponding weights w_{ij}. The weights are computed as

$$\hat{w}_{ij} = \frac{1}{\mathbf{d}_{ij}/\max_j(\mathbf{d}_{ij}) + \varepsilon}, \tag{6}$$

where $\mathbf{d}_{ij}$ is the distances between child and parent in the original mesh. We use the normalized weights $w_{ij} = \hat{w_{ij}}/\sum_j \hat{w}_{ij}$. The important feature of these weights is the fact that if the particle coincides with one of its parents this parent gets a weight of 1 where all other get zero weights. This is true for $\varepsilon = 0$ which yields $\hat{w}_{ij} = \infty$ for the non-normalized weight. To prevent this numerically problematic situation we add a small ε to the denominator.

6.3. Constraint Restriction

A question still open is how to generate the cardinality-n constraints on coarser levels. Here we handle the case of cardinality-2 distance constraints and discuss higher order constraints later. The algorithm for generating the level l constraints given the level $l-1$ constraints proceeds as follows:

First all level $l-1$ constraints are copied to level l. Now certain constraints contain fine particles not present in level l. This problem is solved by processing all fine particles one by one. Each fine particle p_i is collapsed into one of its coarse neighbors p_j (see Fig. 2). The constraint connecting

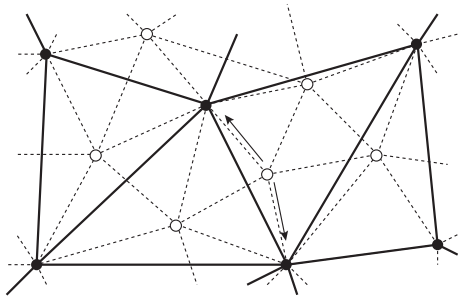

Figure 1: *A fine level l is composed of all the particles shown and the dashed constraints. The next coarser level $l+1$ contains the proper subset of back particles and the solid constraints. Each fine white particle needs to be connected to at least k (=2) black particles – its parents – shown by the arrows.*

p_i and p_j is removed. For all neighbors p_k of particle p_i, if p_k is also a neighbor of p_j the constraint between p_k and p_i is removed. Otherwise, it is replaced by a constraint connecting p_k and p_j. Two question remain open. How to choose p_j among the coarse neighbors of p_i and how to compute the distance for the new constraint from p_k to p_j.

To choose a coarse neighbor p_j we first compute the average of the positions of all coarse neighbors of p_i using the original positions of the particles. For p_j we then choose the neighbor that is closest to the averaged position. This choice produces evenly distributed edge lengths.

There are multiple ways to compute the rest distance of the new constraint from p_k to p_j. One possibility is to use the sum of the rest distances of the constraints $p_k - p_i$ and $p_i - p_j$ (see Fig. 2 (b)). This definition mimics geodetic distances within the manifold of the mesh. If a cut runs through the original mesh the distance constraints will not bridge the gap but will correctly measure the distance around the cut. However, this definition might yield constraints that are too loose because the paths along the mesh edges are only approximations of geodesics.

Another possible way of computing the distance of the coarser constraints is to measure the distance between p_k and p_j in the original pose of the particle system or mesh. For initially flat pieces of cloth for instance, this measure is the correct one while for non flat pieces the resulting constraints might be too tight because they force the cloth back to the original curved state. Also, such constraints will bridge cuts which results in incorrect behavior. Both approaches have applications they are best suited for.

One useful property of this algorithm is the fact that if the distance constraints on the finest level form a triangle mesh, all coarser meshes will be triangle meshes as well, because edge collapses conserve the triangle mesh property. This is particularly useful for level of detail physics because

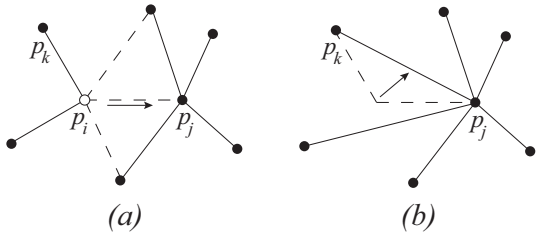

Figure 2: *Fine particle p_i is collapsed into the coarse neighbor p_j. The dashed edges in (a) are removed. Constraint $p_k - p_i$ in (a) becomes the new constraint $p_k - p_j$ in (b). The new constraint length is computed from the constraints $p_k - p_i$ and $p_i - p_j$.*

for objects far from the camera, only coarse meshes can be used for both simulation and visualization.

Higher order constraints such as bending constraints might be generalized in a similar way. However, as mentioned in the introduction, deformable objects typically have low bending resistance and therefore, handling those constraints in coarser levels is not necessary. Also, bending is a local phenomenon that appears when the material is in a non stretched state.

7. Hierarchical Solver

Once the hierarchical data structure is in place, the Multigrid algorithm is fast and simple to implement:

1. Start at the coarsest level: $l \leftarrow l_{max}$
2. Save all positions $\mathbf{p}_i$ of the level l particles in additional state variables $\mathbf{q}_i$.
3. Run one (or more) PBD solver steps on the current level, i.e. project all the cardinality-1 constraints of all the level l particles plus all the cardinality-n constraints of this level using non-linear Gauss-Seidel.
4. If $l = 0$ stop, else go to the next finer level: $l \leftarrow l - 1$
5. Correct the position of each particle i on level l using
$$\mathbf{p}_i \leftarrow \mathbf{p}_i + \sum_{j \in P(i)} w_{ij} \, (\mathbf{p}_j - \mathbf{q}_j), \qquad (7)$$
where $P(i)$ is the set of the indices of the parents of particle i.
6. Go to step 2.

8. Discussion

More sophisticated schemes are possible. Typical Multigrid schemes perform multiple passes up and down the hierarchy. However, we found, that this simple and fast scheme is sufficient to achieve the desired stiffness of the deformable objects.

It is important to note that only the *corrections* computed on a level are propagated down the hierarchy and not the newly computed positions themselves. This guarantees that all the small detail in the high resolution levels is preserved. Coarse geodesic-based distance constraints only grip if the entire material between the adjacent particles is fully stretched as desired. They do not affect the simulation otherwise. For this to be true, we define all higher level distance constraint to be unilateral upper limit constraints.

One of the main reasons developers use physics in games is to make environments destructible. Destructible cloth is tearable cloth. In case of tearing, the hierarchy needs to be updated. The tearing algorithm described in [MHR06] is based on particle splitting. Such a split clones a particle multiple times an assigns subsets of adjacent triangles to each of the clones. In case of a particle split, the parent links as well as the adjacent constraints in the hierarchy get invalid. If they were kept they would potentially bridge cut lines resulting in ghost influences across the cut. We take a conservative approach to solve this problem. First all the parent links and adjacent constraints of the split particle are removed. The algorithm then proceeds recursively up the hierarchy along the parent links and removes constraints and parent links of parents on coarser levels as well (see Fig. 8).

Besides the simplicity and effectiveness of the method, it also has its drawbacks. With very low iteration counts of the PBD solver, visual artifacts may occur if the hierarchical meshes are of poor quality. This effect is removed by increasing the iteration count of the subsequent PBD step to three or four. Also, the hierarchical solver cannot be parallelized as easily as the Gauss-Seidel type solver of PBD. Currently, we run the hierarchical solver sequentially before we start the parallel PBD solver. We already mentioned that we do not consider the bending constraints in higher levels which is a restriction of the current implementation. However, while increased stretching stiffness increases the quality of the simulation, increased bending stiffness typically removes small detail and wrinkles. Such an effect can be achieved simply by making the simulation mesh coarser.

9. Results

We have integrated the method into a real time physics engine. The results presented here and in the video were recorded using a Intel Core2 CPU at 2.66 GHz, 2 GB of main memory and an NVIDIA GeForce 8800 GTX graphics card.

Position Based Dynamics has become quite popular in the field of game physics. It has recently been used in the open source physics engine *Bullet* for adding the new soft body feature. Our method is especially designed for developers who use PDB and want to make it substantially faster. Therefore, we compare the performance of the new approach against the original method and not against more complex solvers.

The diagram in Fig. 3 depicts the number of solver itera-

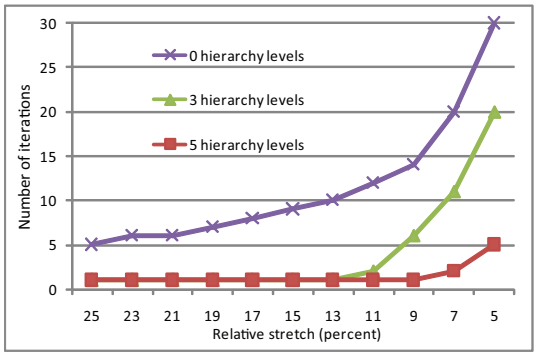

Figure 3: *Number of regular PBD iterations needed to restrict the stretch of a piece of cloth with 126 rows of triangles under gravity to a specific value without a hierarchy and with 3 and 5 hierarchy levels.*

tions needed to restrict stretching under gravity to a certain value. The measurements were done for the piece of cloth shown in Fig. 6. It contains 126 rows of triangles which makes the propagation of pressure waves through the cloth quite slow if the hierarchy is not used. The numbers clearly show the benefit of using the hierarchical approach.

Fig. 7 shows the effect of the Multigrid solver and the influence of the iteration count. At each time step, we first run the Multigrid solver as described in Section 7. After that, the original Gauss-Seidel solver is run because the Multigrid solver does not consider the original constraints on level 0. We run the Gauss-Seidel solver two times while processing the constraints in the opposite order in the second iteration. This makes the process symmetric. The piece of cloth shown is composed of 11,500 triangles and simulated at 60 frames per second. Next, the Multigrid part is switched off while keeping the number of iterations at two. There is almost no performance gain because the total number of constraints in the entire hierarchy is typically smaller than the number of constraints on level 0. With only two iterations the material gets very stretchy. Increasing the number of iterations makes the cloth stiffer but at the same time slows the simulation down. Only with about 20 iterations the stiffness gets comparable with the one using Multigrid. The frame rate in that case drops to about 12 frames per second however. In this example we used a regular base mesh (see Fig. 6) which is suitable for objects like flags or canvases. The hierarchy in this case can be generated directly by using coarser versions of the base structure in coarser levels.

In order to test the algorithm with irregular meshes we simulated a woman with a long dress (see Fig. 5). The final results are shown in Fig. 4. The entire simulation including character animation and character - cloth interaction runs at about 30 frames per second. Without the Multigrid solver, wind and friction on the floor stretch the dress significantly. As the second and the fourth image show, the Multigrid approach makes the simulation of inextensible cloth possible at interactive rates.

10. Conclusions and Future Work

We have presented a non-linear multigrid algorithm to speedup position based dynamics significantly. With this method, the nice features of PBD such as simplicity and stability are kept while the problem of slow error propagation is removed.

One line of future work is the improvement of the tearing algorithm. As mentioned earlier, our procedure is conservative. It typically removes more constraints then necessary which makes the cloth around the tear line weaker than it should. We also work on other methods to generate the coarser meshed in the hierarchy.

So far we have tested the method on cloth only. As a next step we will port it to our soft body simulation engine which we expect to be a straight forward task because the method is not restricted to triangles.