Workshop on Virtual Reality Interaction and Physical Simulation VRIPHYS (2013)
J. Bender, J. Dequidt, C. Duriez, and G. Zachmann (Editors)

Multilevel Cloth Simulation using GPU Surface Sampling

N. Schmitt[1], M. Knuth[2], J. Bender[1] and A. Kuijper[1]

[1]TU Darmstadt, Germany
[2]Fraunhofer IGD, Germany

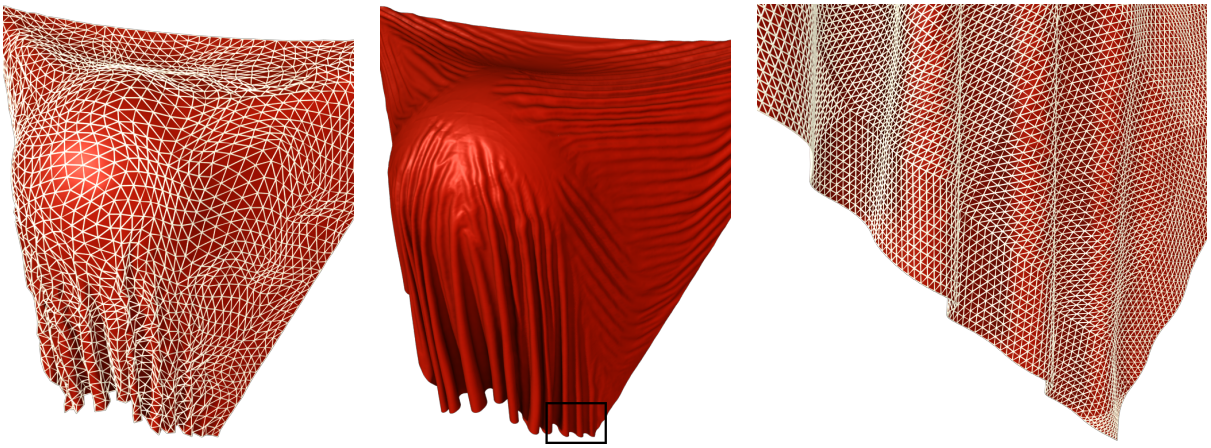


Figure 1: *Cloth draped over a sphere. A coarse triangle mesh with 5100 vertices (left) is used for collision handling on the CPU while a high-resolution regular grid with 230k particles is simulated on the GPU to get fine wrinkles (middle). The right image shows a close-up view of the high-resolution mesh used for rendering.*

Abstract

Today most cloth simulation systems use triangular mesh models. However, regular grids allow many optimizations as connectivity is implicit, warp and weft directions of the cloth are aligned to grid edges and distances between particles are equal. In this paper we introduce a cloth simulation that combines both model types. All operations that are performed on the CPU use a low-resolution triangle mesh while GPU-based methods are performed efficiently on a high-resolution grid representation. Both models are coupled by a sampling operation which renders triangle vertex data into a texture and by a corresponding projection of texel data onto a mesh. The presented scheme is very flexible and allows individual components to be performed on different architectures, data representations and detail levels. The results are combined using shader programs which causes a negligible overhead. We have implemented CPU-based collision handling and a GPU-based hierarchical constraint solver to simulate systems with more than 230k particles in real-time.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

1. Introduction

Several physical models for cloth simulation have been suggested, with discrete systems being among the most common. In the beginning several works focused on grid structures [BHW94, Pro95]. However, the concept was soon generalized for arbitrary triangle meshes [VMT97, BW98].

Grids had a resurgence when first attempts were made to simulate cloth on graphics hardware [KLRS04, Zel05].

Regular grids have various advantages: No explicit connectivity data structure is required; the anisotropic properties of cloth can be incorporated without transformation into a local coordinate system and edge lengths are uniform. Especially in the light of GPU computing where grids are efficiently represented by texture samplers there are numerous optimizations which can be performed. Triangle meshes are often preferred for modelling since they are flexible enough to represent irregularly shaped pieces of cloth. Moreover, most collision handling algorithms are based on triangle meshes (e.g. [BWK03]).

Our idea is to combine both data representations by sampling changes on a coarse triangle mesh into high-resolution grids and projecting updated positions back onto vertices. In order to demonstrate the viability of our approach we implement a simple collision handler on the CPU and a hierarchical solver for Position Based Dynamics [MHHR07] on the GPU. Because upsampling and downsampling of grids can be run efficiently on the GPU, the implementation causes less overhead than approaches based on explicit hierarchies. This allows us to simulate highly detailed grids with more than 230k particles in real-time at large time steps while the time-consuming collision handling on the CPU is performed on a lower level of complexity (see Figure 1).

2. Related Work

Cloth simulation has a long history in computer graphics [TPBF87]. Thalmann et al. [MTV05] as well as Choi et al. [CK05] give surveys of this research area.

In the beginning many works investigated mass-spring systems and used explicit time integration schemes in order to perform interactive simulations [BHW94, Pro95]. Since real textiles do not stretch significantly under their own weight, stiff materials are required for realistic results. However, explicit integration methods are not suited to simulate such materials due to stability problems [HES03]. Hence, in the following years unconditionally stable implicit integration methods were investigated [BW98, CK02]. In general these methods must solve a large system of equations. Therefore, GPU-based solvers became popular in the area of physically-based simulation [BCL09, CA09, WBS*13].

Since the quality of the results becomes more and more important, different works focused on the development of methods based on continuum mechanics [EKS03, VMTF09, BD13]. At the same time position-based approaches were developed [MHHR07, SSBT08, MC10]. These kinds of methods are fast, unconditionally stable and controllable which makes them well-suited for interactive simulation. The main application areas are virtual reality, computer games and special effects since they only provide visually plausible results. A survey on these methods is given by Bender et al. [BMOT13].

A realistic simulation of textiles requires stiff cloth models. Hence, the limitation of the maximal strain is an important topic in the area of cloth simulation. Provot [Pro95] modifies positions to reduce the strain of his model. In contrast, Bridson et al. [BFA02] use an impulse-based strain limiting approach to avoid self-penetrations. While these methods use discrete models, continuum-based approaches are presented by Thomaszewski et al. [TPS09] and Wang et al. [WOR10]. The simulation of totally inextensible cloth is also a research topic and approaches based on constrained Lagrangian mechanics [GHF*07] or impulse-based techniques [BB08] were investigated.

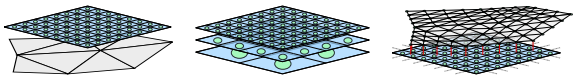
Different works use multigrid methods to increase the convergence rate of their solvers [Mül08, DGW11a, WOR10, BWD13]. Müller et al. [Mül08] use a hierarchical model in their non-linear Gauss-Seidel solver for a faster position-based simulation. Bender et al. [BWD13] introduce a multi-resolution approach to simulate cloth models efficiently with shape matching. Strain limiting is accelerated by Wang et al. [WOR10] using a multigrid approach. Dick et al. [DGW11b] use a regular hexahedral mesh to perform an efficient multigrid finite element simulation on the GPU.

The idea of mapping arbitrary triangle meshes onto grids for easy access via samplers in GPU programs is not new to cloth simulation [Zel05]. Zink et al. [ZH07] develop a mass-spring based framework which leverages the grid layout for faster integration methods. In both works geometry images introduced by Gu et al. [GGH02] are the basis for mapping vertex positions onto a regular grid. The parameterization distorts relative edge length and angles to make use of the entire space of the texture. We choose an isometric parameterization instead, allowing separate simulation of stretch and shear forces and making explicit storage of rest lengths redundant. In contrast to the above-mentioned frameworks we continue to use the triangular base mesh for simulation rather than discarding it.

The framework presented by Li et al. [LWM11] also uses CPU and GPU for simulation tasks. In contrast to our work the same geometry model is used and transferred between both architectures in each frame. The authors argue that the sequential nature of strain limiting algorithms make them better suited for the CPU. However, in our work we show how an efficient parallel constraint solver can be implemented on the GPU.

3. Concept

We propose a multilevel approach where a low-resolution triangle mesh is kept in sync with a high-quality regular grid. Depending on complexity and parallelizability algorithms are either executed on the CPU using the coarse triangle mesh, or on the GPU using the advantages of the grid



(a) (b) (c)

Figure 2: *Overview of the presented process. (a) The initial low resolution triangle mesh is rendered to a texture using the triangle mesh's U/V parameterization. The texels represent particles of a high resolution regular grid. (b) Mipmap levels of the texture are used to perform a hierarchical physics simulation. (c) The result is mapped to a high resolution mesh for rendering as well as back to the low resolution mesh.*

representation (cf. [BDW*10]). To couple both models we define a sampling operation which transforms vertex data such as position changes and velocities into grid particles. Furthermore, we introduce a projection operation which applies changes made on the grid to the vertices of the triangle mesh. The same method is used to project the grid's particle positions onto a high-resolution version of the triangle mesh for rendering purposes. An overview of this process can be seen in Figure 2.

The sampling process is implemented as a render-to-texture operation where vertex positions or velocities are written into the texels. Projection is accomplished in the vertex stage of the rendering pipeline by looking up texel values depending on a 2D parameterization of the surface. By implementing the mapping between the two models as GPU programs the data transfer between both architectures is handled implicitly.

In contrast to previous approaches which use position textures, we use an isometric parameterization of the cloth surface. While this bears various limitations regarding the structure of the mesh, it ensures strain limiting is performed correctly along the warp and weft directions of the fabric. Inconsistencies occurring at the borders of non-square surfaces are handled by a dynamic border growing algorithm at the sampling stage and an extrapolation scheme at the projection stage.

In order to demonstrate the advantages of the grid representation for high-resolution cloth models, a variation of the Position Based Dynamics [MHHR07] approach is implemented on the GPU. Moreover, we perform a comparison between particle-based and edge-based solvers regarding their performance on current graphics hardware.

As the particle count increases, the efficiency of iterative constraint solvers decreases significantly. The structural properties of regular grids can be used to build a hierarchy of systems easily and to define up- and downsampling operations as fragment programs. The resulting multi-resolution solver is several times faster than regular iterative constraint enforcement.

4. Surface Mapping

In order to map vertex data of an irregular triangle mesh onto a regular grid a suitable parameterization which maps coordinates from $\mathbb{R}^3$ to $\mathbb{R}^2$ is required. An isometric parameterization, where the area of each triangle and the angles between edges are preserved, has the advantage of simplifying subsequent operations and ensuring the asymmetric behaviour of cloth is taken into account. While such a parameterization restricts triangle meshes to developable surfaces without loops, we argue that this is not a severe limitation in the field of garment simulation. Individual patches of cloth are usually designed as 2D patterns and stitched together (see also [KKK10b, KKK10a]).

The vertex data of the triangle mesh is sampled by using the input parameterization coordinates as position output of the rendering pipeline's vertex stage. This data is passed to the fragment stage which writes it into a 32bit floating point texture. The GPU rasterizes the triangle mesh into a grid by picking data values of the triangle mesh at texel centres and interpolating between vertices when necessary.

The projection operation is implemented in the vertex stage, receiving only the parameterization coordinates as input. These are used to fetch positions from the sampled texture, which can then be processed as usual. For the simulation they are directly written into a buffer and no fragment pass is performed. For rendering they are altered according to world, view and projection transformations with normals computed on-the-fly during the fragment stage for better quality.

Unfortunately, relying on the GPU's bilinear filtering for texture lookups causes errors at texture and mesh boundaries as demonstrated in Figure 3. Texture boundary problems occur due to the fact that a $n \times n$ texture only covers the range $[\frac{1}{2n}, 1 - \frac{1}{2n}]$ and OpenGL will interpolate values outside that range by either clamping or wrapping border texels. Either behaviour leads to erroneous data being read.

A solution is to apply a transformation to the input parameterization, restricting it to the texture range and preventing any texture boundary collisions. However, this leads to problems. Texels which the mesh intersects with, but whose centres are not covered, are discarded by the rasterizer. The same phenomenon can be observed at the boundaries of non-square meshes which inevitably cause holes in the parameterization. In Section 4.1 a technique to deal with the problem by covering additional texels at the sampling stage is described, while Section 4.2 presents a method to ensure only valid texels are read during the projection operation.

4.1. Sampling

If all texels which are covered by the mesh to some degree were written, more information would be available and problems at the mesh boundary would be greatly reduced.

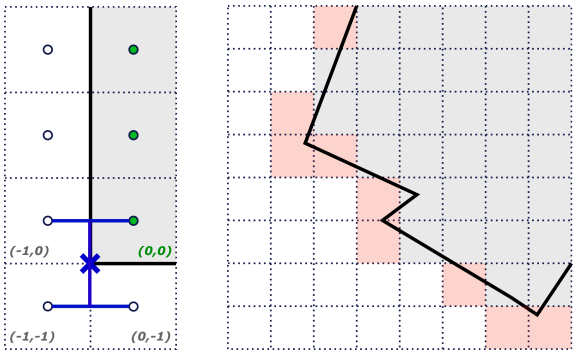


Figure 3: *Left: Interpolation problems at texture boundaries due to clamping. The blue vertex is approximated by its four surrounding texels, but only one of them is within texture boundaries. Right: The mesh intersects with red texels but their centre is not covered and no data is written.*

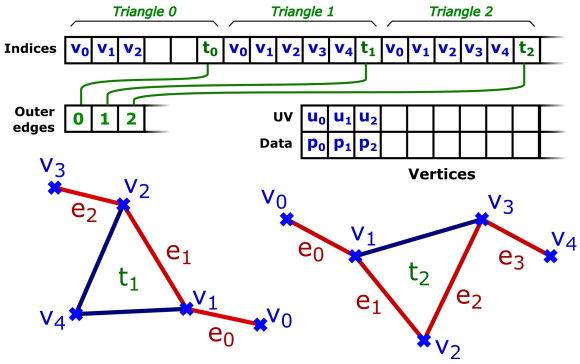


Figure 4: *Different types of triangles are based on their outer edges (marked red). Six vertex indices are encoded into a buffer for each triangle and ordered as depicted to identify adjacent edges in the geometry shader.*

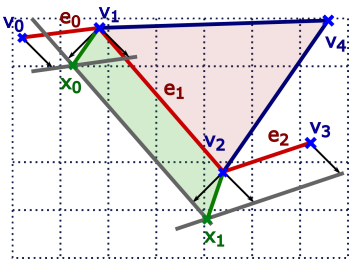


Figure 5: *Border vertices x_0 and x_1 are determined from intersecting shifted edges.*

The literature provides a solution to the coverage problem called Conservative Rasterization [HAMO05]. Triangles are enlarged in the geometry stage of the rendering process to cover the centres of all texels the triangle intersects. Unfortunately this causes additional texels to be covered not only at the boundaries of the mesh, but also for each individual triangle within. Adjacent triangles overlap each other and incorrect data may be written depending on which triangle is rendered first. The approach provides no means of extrapolating vertex data either.

We propose an enhanced version of the Conservative Rasterization approach. By creating a specialized index buffer it is possible to create borders only at the outline of the mesh. The border is created by generating additional triangles in the geometry stage and vertex data is correctly extrapolated to allow subsequent simulation and projection of the data.

Building the index buffer Creating the outline border for a mesh in the geometry stage requires three inputs: The three vertices of the triangle primitive, a flag which determines whether the triangle has one, two or no outer edges, and any adjacent outer edges. By rendering the mesh as triangles with adjacency, six input indices can be passed to the geometry program. Three are used by the triangle itself, two by adjacent outer edges and the last one is used as index for a special buffer storing the triangle type (i.e. outer edge count).

Figure 4 shows two different configurations occurring in a triangle mesh and how they are stored in the index buffer. Depending on which of the triangle edges are outer edges, there are three permutations of t_1 and t_2. Triangle indices read from the mesh are rearranged and written to the buffer such that the order with respect to the location of the outer edges is always as depicted. This enables the geometry program to treat all permutations equally.

Creating border triangles For triangles with no outer edges no border is generated, the geometry program simply passes the vertices through. In case there is one outer edge (t_1) two new vertex locations have to be determined and two additional triangles are generated (see Figure 5). If there are two outer edges (t_2) there are three new vertex locations and four triangles to generate. Apart from these differences the process of computing new vertex positions is largely the same. Edges are determined from vertex positions in UV space and the normals of these edges are used to compute offsets. Offsets are diagonals chosen depending on the quadrant the normal resides in and the size of a texel in order to reliably cover the centres of any texel the mesh intersects [HAMO05]. The offsets are then applied to each edge. Positions of the new border vertices are set to the intersection points of the lines through the shifted edges.

Extrapolating vertex data Having determined the positions in UV space for the new vertices, the next step is to infer their data. This is done by first computing the intersection points s_i between the lines from new vertices x_i to the inner vertex of the triangle v_4 and the line defined by the

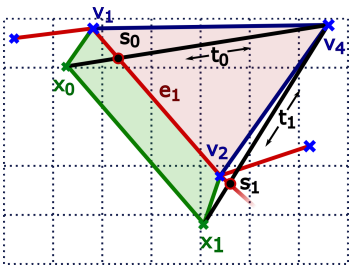


Figure 6: *Further intersections s_0 and s_1 allow extrapolation of data from the triangle's vertices.*

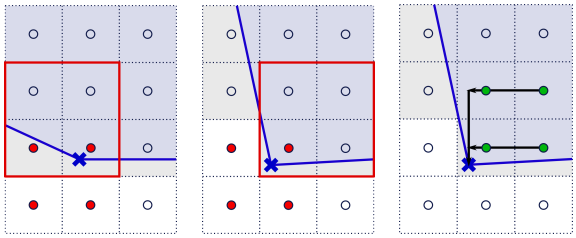


Figure 7: *The four texels closest to a vertex are marked red. An offset of $(0,1)$ is required on the left, leading to the four texels outlined in red. The next scenario requires an offset of $(1,1)$. With the offset determined, extrapolation between the selected texels can commence (right).*

outer edge e_1 (see Figure 6). Data for s_i can be interpolated from v_1 and v_2. Finally, the data for x_i is extrapolated using v_4 and s_i and the relative distance t_i between both points. The computation of vertex data for triangles with two outer edges requires different lines but follows the same concept.

4.2. Projection

The method suggested in the previous section reduces the information loss during the sampling process due to interpolation with empty texels significantly. However, there are still cases where regular bilinear filtering will cause issues when reading position values from a texture in the projection stage. This section describes a method to identify such cases and deal with them by extrapolating from the closest, non-empty texel.

Figure 7 shows two cases where not all of the four texels which are closest to a vertex are defined and suggests nearby texels as basis for extrapolation. Offsets $\mathbf{o} \in \mathbb{Z}^2$ define the shift in x and y direction which, when applied to texel coordinates, results in a coverage mask where all texels contain values. As offsets only depend on the parameterization which remains unchanged during simulation, they can be determined in a pre-computation step by starting with a zero offset, examining consecutively higher offsets, and halting as soon as a valid coverage mask is found.

Given a texture of size n, UV coordinate $\mathbf{u} \in \mathbb{R}^2$ and the offset $\mathbf{o} \in \mathbb{Z}^2$, the final value of a vertex is read from the texture as follows. First, texel indices and the distance to the texel centre are determined:

1: $\mathbf{u} \leftarrow \mathbf{u} \cdot n - \mathbf{0.5}$
2: $\mathbf{r} \leftarrow \mathbf{u} - \text{floor}(\mathbf{u})$
3: $\mathbf{i} \leftarrow \text{floor}(\mathbf{u}) + \mathbf{o}$
4: $\mathbf{l} \leftarrow \max(\mathbf{0}, \text{abs}(\mathbf{o}) - \mathbf{1})$

Then, the four closest, covered texels $t_{i_x,i_y}, t_{i_x+1,i_y}, t_{i_x,i_y+1}$ and t_{i_x+1,i_y+1} are retrieved. Given two texel values t_0 and t_1 the approximated value is computed by either using extrapolation if the offset is not zero, or regular bilinear filtering otherwise. Parameters $o, l \in \mathbb{Z}$ and $r \in \mathbb{R}$ correspond to the x or y component of the vectors defined above, depending on the orientation of the two texels.

1: ***function*** EXTRAPOLATE(t_0, t_1, o, r, l)
2: **if** $o = 0$ **then**
3: **return** $t_0(1-r) + t_1 r$
4: **else if** $o > 0$ **then**
5: **return** $t_0 - (l - (1-r))(t_1 - t_0)$
6: **else if** $o < 0$ **then**
7: **return** $t_1 + (l + r)(t_1 - t_0)$

The final value is determined by first extrapolating the bottom and top two texels and then combining the two results (see Figure 7).

4.3. Error Evaluation

Our intention is to upsample coarse triangle meshes with relatively evenly spread vertices into high-resolution textures for cloth simulation. This ensures the sampling resolution is always sufficiently high and the relative distance between the values stored into texels remains smooth, thereby keeping the error from interpolations low. However, accuracy remains a high priority in garment simulation and a quantification of the error is required.

We measured error terms in an experimental setting chosen to reflect common hazards of surface mapping. A snapshot of our cloth simulation is used including high curvature regions and stretch deformation up to 10%. Irregularly triangulated regions exist at the borders and the curvy shape causes holes in the parameterization. The per-vertex error e_v is defined as the difference between original vertex data and vertex data after projection, normalized by the surface's dimension D.

$$e_v = \frac{\|(v_{new}) - (v_{old})\|}{\|D\|}$$

The total error is calculated as the average error over all vertices in the mesh.

In this setting the average error was generally below 0.06 %. Error terms at mesh borders rise up to 0.5 % where cloth deformation is high but we have discovered no notable

artefacts in simulation. The error can be reduced even further by applying a pre-processing step to the mesh, snapping inner vertices to the nearest texel centre and thus ensuring an accurate sampling. Borders must remain unchanged in order to preserve the mesh shape.

5. Hierarchical Solver

The cloth simulation is performed on the GPU using the regular grid representation. We use a high resolution for the finest level to enable detailed folds and wrinkles. A position-based approach [MHHR07] is employed to simulate deformation of the surface. We define distance constraints for neighbouring particles in the grid and solve them iteratively in order to guarantee a given maximum strain. The iteration process is accelerated by a multi-resolution solver.

A distance constraint between two particles $\mathbf{p}_1$ and $\mathbf{p}_2$ with rest length l_0 is defined by $C(\mathbf{p}) = \|\mathbf{p}_2 - \mathbf{p}_1\| - l_0 = 0$ and enforced by moving both particles along their shared edge:

$$\Delta\mathbf{p}_i = -k w_i \frac{C(\mathbf{p})}{\sum_j w_j |\nabla_{\mathbf{p}_j} C(\mathbf{p})|^2} \nabla_{\mathbf{p}_i} C(\mathbf{p}), \qquad (1)$$

where $w_i = 1/m_i$ is the inverse particle mass and $k \in [0,1]$ adjusts the strength of the correction. Since constraints with common particles influence each other, a global enforcement is required.

5.1. Global Solvers

The most popular approach is to use a *Gauss-Seidel solver* to solve the system of non-linear equations which is implied by the constraints. This is an iterative method which computes a local position change $\Delta\mathbf{p}$ for each constraint and applies it immediately. Subsequent constraints take the changed positions into account. The sequential nature of the algorithm makes it ill-suited for an implementation on the GPU. Each particle position is part of multiple constraints which consequently cannot be processed in parallel.

The *Jacobi method* treats each constraint independently. Local position changes are based solely on the values at the start of the current iteration. For each particle the position changes of all corresponding constraints are averaged to get an approximation for the next step. This allows constraints to be processed in parallel but at a slower convergence rate as the Gauss-Seidel solver.

5.2. Implementation

We implemented two approaches for a comparison: an *edge-based* and a *particle-based* approach.

The *edge-based* method, which was introduced in the original paper [MHHR07], iterates over all edges (i.e. constraints). For each edge it determines the position changes for both corresponding particles. In GPU terms this means one thread per edge which needs to write two values, requiring multiple framebuffers or arbitrary writes into global memory. Furthermore, one particle is affected by multiple constraints and thus multiple threads, leading to synchronization issues.

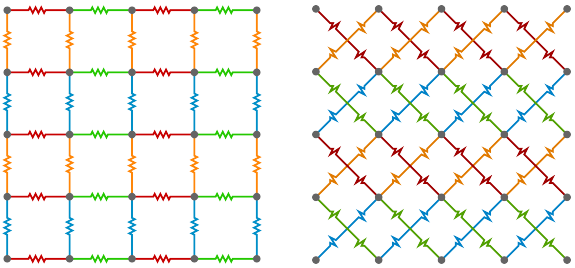


Figure 8: *Four independent sets each for stretching and shearing constraints.*

The *particle-based* approach spawns one thread per particle instead. Each thread computes the position changes for all corresponding constraints and writes exactly one value to the texture. All threads may run in parallel and only one framebuffer is required, however each constraint must be handled twice since it depends on two particles.

Each particle is part of at most eight constraints (or outgoing edges), thus the implementation of Gauss-Seidel requires eight separate render calls to ensure each constraint is solved based on the results of previous constraints. Independent sets can be formed as shown in Figure 8 [Zel05]. Our edge-based implementation passes particle indices of constraints to a vertex shader and updates particle positions via the image operations of OpengGL 4. The particle-based approach was not implemented in combination with the Gauss-Seidel solver. Due to the sequential nature of Gauss-Seidel, the advantage of parallel execution cannot be incorporated, but constraints would still have to be computed twice.

The Jacobi method on the other hand can be executed in a particle-based fashion by binding the output texture to the framebuffer and computing updated positions in the fragment shader. Only one memory location is written per instance, thus only a single render call is required and all constraints are processed in parallel. Implementing the Jacobi method in an edge-based manner would require multiple output textures to allow parallel writes without clashes, as well as at least one additional render call to perform a reduction which averages the collected per-constraint values for each particle.

5.3. Multi-Resolution Solver

Independently of whether the interleaved Gauss-Seidel or the parallel Jacobi approach is chosen, the number of iterations necessary to enforce low strain throughout the cloth

increases dramatically with a higher grid resolution. We use a multigrid scheme which allows faster propagation of changes through the cloth model to reduce the number of required iterations and therefore to improve the performance.

A hierarchy of grids is built by downsampling the highest resolution positions. The final level runs on the full-resolution grid while each preceding level halves the resolution. Downsampling is implemented by sampling only every second (every fourth, every eighth, etc.) texel of the full-resolution texture. Upsampling from one level to the next higher level is implemented by looking up coordinates in-between texel centres, leaving the bilinear interpolation of the two neighbouring particles to the graphics card.

One iteration of the hierarchical solver can be summarized as follows:

1. downsample positions for all levels
2. perform strain limiting algorithm on the first level
3. compute the position differences by subtracting the initial values and upsample these differences to the next level
4. apply position changes on next level and perform strain limiting
5. if the final level is not reached, continue with step 3

6. Results

The cloth simulator used in the following tests combines the techniques presented in this paper. It first projects the current particle state stored in the position textures onto the low-resolution triangle mesh. A simple CPU-based algorithm is used to detect collisions with rigid bodies. Collision handling is performed by projecting colliding vertices onto the surface of the corresponding body. The resulting position changes are written into the mesh's vertex buffer and sampled into an offset texture. This is incorporated into a Verlet integration step on the GPU which also applies gravity as an external force. The resulting positions from the integration are then adjusted by running the hierarchical solver. All timings were measured on an AMD Radeon HD 7850 graphics card.

The first experiment is conducted with a square piece of cloth which is fixed at two corners and draped over a sphere. The highest resolution grid has 50k particles, resulting in roughly 150k constraints to be solved in the last level of the multigrid scheme. Collisions are handled using a coarse triangle mesh with 1300 vertices. Timings given in Table 1 show the GPU-based simulation only requires 3 ms (19%) of simulation time using a time step size of $1/60$ s, leaving the rest for collision detection and rendering. The overhead incurred by using two different models on different hardware is minimal. Sampling and projection steps need together less than 0.1 ms per step.

The times were achieved by running the particle-based Jacobi and the edge-based Gauss-Seidel relaxation steps at different stages of the hierarchical solver. A direct comparison

Timestep 16 ms							
Surface mapping		Inte-	Hierarchical solver				
Sample	Project	gration	0	1	2	3	4
0.06	0.02	0.08	0.27	0.31	0.49	0.37	1.62

Table 1: *Time measurements of GPU tasks in ms.*

of both approaches is given in Figure 9. While one iteration of the edge-based algorithm is much slower due to expensive image operations and synchronization between independent sets, it does require far less iterations to converge. This results in both approaches being equally viable to enforce a certain strain limit.

For particle systems of size 10k and higher neither approach produces acceptable results. Therefore, we introduced a multi-resolution solver which allows us to target higher resolutions in real-time without sacrificing the fixed strain limit of 10 %. A different iterative solver can be used at each level of the hierarchy. Figure 10 suggests that using the edge-based Gauss-Seidel solver at lower and the particle-based Jacobi method at the highest resolutions yields the best overall performance. The simulation of a 230k particle system with 6 stages takes 16.6 ms when Gauss-Seidel is used for all resolutions. If the last two stages are solved with the Jacobi method, 21 instead of 4 iterations are required to keep strain at 10 %, but the total process only takes 6 ms. The Jacobi method's disadvantage of slow propagation through large grids led to poor overall performance despite extremely fast single iteration times (see Figure 9). Solving the system on lower resolutions compensates for this weakness. This explains the Jacobi method's exceptional performance in a hierarchical environment. Experiments with more complex cycles up and down the hierarchy did not lead to improvements over simply using more iterations on each level or reducing the timestep.

While the introduction of additional levels leads to significant reduction of strain in cloth (see Figure 11), the visual quality starts to suffer if the resolution drops too low. Folds diagonal to the grid become blocky unless the number of iterations on the highest level is sufficiently large to smooth the artefacts out. We found that a size of approximately 20×20 for the lowest resolution grid typically provides the greatest boost to performance without negatively impacting visual quality. It is also important to solve compression and bending constraints only at the highest resolution, as reducing edge compression on low levels of the hierarchy impairs bending of the cloth.

7. Conclusion and Future Work

The resolution of cloth models is limited in real-time simulation due to the complexity of solving systems of equations and collision handling algorithms. We presented a method

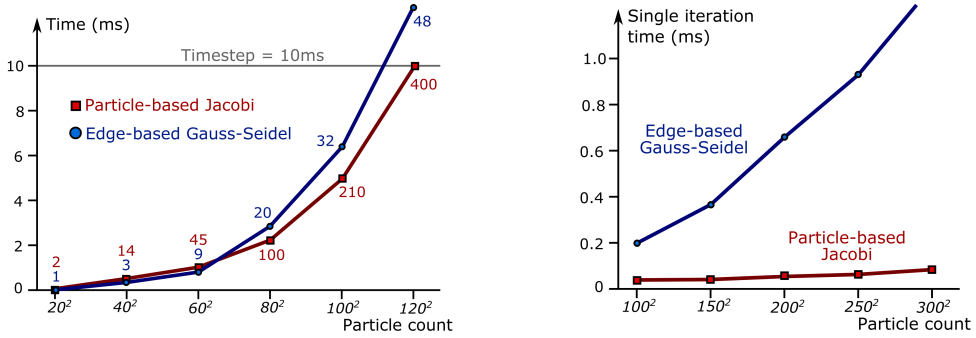


Figure 9: *Comparison of particle-based Jacobi and edge-based Gauss-Seidel constraint relaxation without multi-resolution. Both methods show similar execution times when enforcing a strain limit of 10 % but the Jacobi method requires far more iterations (left). Due to its parallel nature, the single iteration time is however less affected by increasing grid resolution (right).*

Figure 11: *Hierarchical solver running on a* 100×100 *grid at 100 frames per second. Left: 1 level, 1 ms, 60 % strain. Middle: 3 levels, 1.2 ms, 16 % strain. Right: 4 levels, 1.3 ms, 8 % strain.*

to combine the flexibility and low complexity of a coarse triangle mesh with the efficiency and detail of high-resolution regular grids.

Simulation of irregularly shaped triangle meshes is shown in Figure 12. By choosing an isometric parameterization the constraint solver can reflect the asymmetric behaviour of cloth. However, several restrictions are imposed on the nature of the input mesh. Only surfaces which can be flattened with minimal distortion are applicable. Stitching of individual pieces of cloth can be handled by the constraint solver but requires an additional data structure. Neighbouring texels are implicitly treated as connected; if they are to be severed by a cut a separate connectivity texture must be utilized. Apart from these limitations there is also a slight error every time the mesh is mapped from its triangle representation to the regular grid and back. However, our results show that the error does not negatively impact simulation behaviour.

The implemented hierarchical solver takes full advantage of the grid layout and allows massively parallel simulation of cloth with more than 230k particles in real-time. At the same time complex tasks such as collision handling need not be affected by the large amount of particles as they are resolved on a mesh with only 10k triangles. Our approach allows complete freedom of choice over which part of the simulation is executed on which data structure, architecture and level of detail. The GPU-based sampling and projection algorithms cause virtually no performance overhead.

Future work will concentrate on implementing further collision and self-collision handlers and exploring the potential of combining GPU-based detection methods (e.g. image-based or voxel-based) with CPU-based resolution. There are also opportunities to improve performance of the Gauss-Seidel solver when the compute shader introduced in OpenGL 4.3 reaches maturity.

Acknowledgment

The work of Jan Bender was supported by the Excellence Initiative of the German Federal and State Governments and the Graduate School CE at TU Darmstadt. The research leading to these results has received funding from the European Commission's Seventh Framework Programme (FP7/2007-2013) under grant agreement n^{o} 285026.

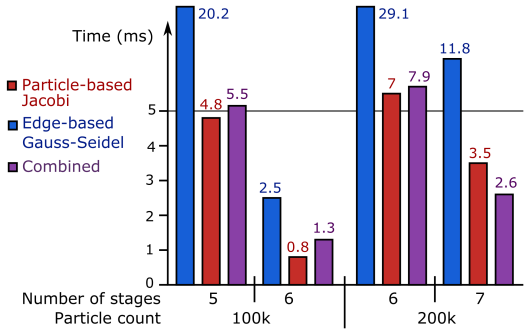


Figure 10: *Timings at varying number of solver stages and two different resolutions. Cloth strain is limited to 10 %. Gauss-Seidel produces better results at low levels while the Jacobi method excels at higher resolutions, leading to the combined approach.*

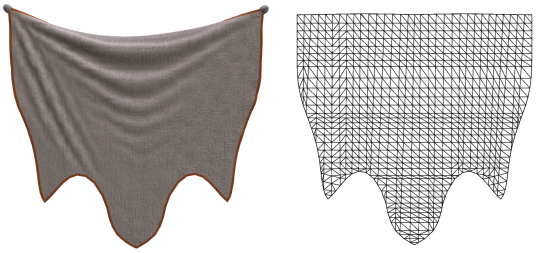


Figure 12: *Irregularly shaped triangle mesh simulated by our grid-based solver (left) using an isometric parameterization (right).*

References

[BB08] BENDER J., BAYER D.: Parallel simulation of inextensible cloth. In *Proc. Virtual Reality Interactions and Physical Simulations* (2008), Eurographics Association, pp. 47–56. 2

[BCL09] BUATOIS L., CAUMON G., LEVY B.: Concurrent number cruncher: a GPU implementation of a general sparse linear solver. *Int. J. Parallel Emerg. Distrib. Syst. 24* (2009), 205–223. 2

[BD13] BENDER J., DEUL C.: Adaptive cloth simulation using corotational finite elements. *Computers & Graphics 37*, 7 (2013), 820 – 829. 2

[BDW*10] BINOTTO A. P. D., DANIEL C., WEBER D., KUIJPER A., STORK A., PEREIRA C. E., FELLNER D. W.: Iterative sle solvers over a cpu-gpu platform. In *12th IEEE International Conference on High Performance Computing and Communications, HPCC 2010, 1-3 September 2010, Melbourne, Australia* (2010), pp. 305–313. 3

[BFA02] BRIDSON R., FEDKIW R., ANDERSON J.: Robust treatment of collisions, contact and friction for cloth animation. *ACM Trans. Graph. 21* (2002), 594–603. 2

[BHW94] BREEN D. E., HOUSE D. H., WOZNY M. J.: Predicting the drape of woven cloth using interacting particles. In *Proc. SIGGRAPH* (1994), ACM, pp. 365–372. 1, 2

[BMOT13] BENDER J., MÜLLER M., OTADUY M. A., TESCHNER M.: Position-based methods for the simulation of solid objects in computer graphics. In *EUROGRAPHICS 2013 State of the Art Reports* (2013), Eurographics Association. 2

[BW98] BARAFF D., WITKIN A.: Large steps in cloth simulation. In *Proc. SIGGRAPH* (1998), ACM, pp. 43–54. 1, 2

[BWD13] BENDER J., WEBER D., DIZIOL R.: Fast and stable cloth simulation based on multi-resolution shape matching. *Computers & Graphics 37*, 8 (2013), 945 – 954. 2

[BWK03] BARAFF D., WITKIN A., KASS M.: Untangling cloth. *ACM Trans. Graph. 22*, 3 (July 2003), 862–870. 2

[CA09] COURTECUISSE H., ALLARD J.: Parallel dense gauss-seidel algorithm on many-core processors. In *Proc. High Performance Computing and Communications* (2009), pp. 139–147. 2

[CK02] CHOI K.-J., KO H.-S.: Stable but responsive cloth. In *Proc. SIGGRAPH* (2002), ACM, pp. 604–611. 2

[CK05] CHOI K., KO H.: Research problems in clothing simulation. *Computer-Aided Design 37*, 6 (2005), 585–592. 2

[DGW11a] DICK C., GEORGII J., WESTERMANN R.: A hexahedral multigrid approach for simulating cuts in deformable objects. *IEEE TVCG 17*, 11 (2011), 1663–1675. 2

[DGW11b] DICK C., GEORGII J., WESTERMANN R.: A real-time multigrid finite hexahedra method for elasticity simulation using CUDA. *Simulation Modelling Practice and Theory 19*, 2 (2011), 801–816. 2

[EKS03] ETZMUSS O., KECKEISEN M., STRASSER W.: A fast finite element solution for cloth modelling. In *Proc. Computer Graphics and Applications* (2003), IEEE Computer Society, pp. 244–. 2

[GGH02] GU X., GORTLER S. J., HOPPE H.: Geometry images. In *Proc. SIGGRAPH* (2002), ACM, pp. 355–361. 2

[GHF*07] GOLDENTHAL R., HARMON D., FATTAL R., BERCOVIER M., GRINSPUN E.: Efficient simulation of inextensible cloth. *ACM Transactions on Graphics 26*, 3 (2007). 2

[HAMO05] HASSELGREN J., AKENINE-MÖLLER T., OHLSSON L.: Conservative rasterization. *GPU Gems 2* (2005), 677–690. 4

[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

[KKK10a] KNUTH M., KOHLHAMMER J., KUIJPER A.: Embedding hierachical deformation within a realtime scene graph - a simple approach for embedding gpu-based realtime deformations using trilinear transformations embedded in a scene graph. In *GRAPP 2010 - Proceedings of the International Conference on Computer Graphics Theory and Applications, Angers, France, May 17-21, 2010* (2010), pp. 246–253. 3

[KKK10b] KNUTH M., KOHLHAMMER J., KUIJPER A.: A geometry-shader-based adaptive mesh refinement scheme using semiuniform quad/ triangle patches and warping. In *Proceedings of the Seventh Workshop on Virtual Reality Interactions and Physical Simulations, VRIPHYS 2010, Copenhagen, Denmark, 2010* (2010), pp. 21–29. 3

[KLRS04] KOLB A., LATTA L., REZK-SALAMA C.: Hardware-based simulation and collision detection for large particle systems. In *Proc. ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware* (2004), ACM, pp. 123–131. 2

[LWM11] LI H., WAN Y., MA G.: A cpu-gpu hybrid computing framework for real-time clothing animation. In *Proc. Cloud Computing and Intelligence Systems* (2011), IEEE, pp. 391–396. 2

[MC10] MÜLLER M., CHENTANEZ N.: Wrinkle meshes. In *Proc. Symposium on Computer Animation* (2010), Eurographics Association, pp. 85–92. 2

[MHHR07] MÜLLER M., HEIDELBERGER B., HENNIX M., RATCLIFF J.: Position based dynamics. *Visual Communication and Image Representation 18*, 2 (2007), 109–118. 2, 3, 6

[MTV05] MAGNENAT-THALMANN N., VOLINO P.: From early draping to haute couture models: 20 years of research. *The Visual Computer 21* (2005), 506–519. 2

[Mül08] MÜLLER M.: Hierarchical position based dynamics. In *Proc. Virtual Reality Interactions and Physical Simulations* (2008), Eurographics Association, pp. 1–10. 2

[Pro95] PROVOT X.: Deformation constraints in a mass-spring model to describe rigid cloth behavior. In *In Graphics Interface* (1995), pp. 147–154. 1, 2

[SSBT08] STUMPP T., SPILLMANN J., BECKER M., TESCHNER M.: A Geometric Deformation Model for Stable Cloth Simulation. In *Proc. Virtual Reality Interactions and Physical Simulations* (2008), Eurographics Association, pp. 39–46. 2

[TPBF87] TERZOPOULOS D., PLATT J., BARR A., FLEISCHER K.: Elastically deformable models. In *Proc. SIGGRAPH* (New York, NY, USA, 1987), ACM, pp. 205–214. 2

[TPS09] THOMASZEWSKI B., PABST S., STRASSER W.: Continuum-based strain limiting. *Computer Graphics Forum 28*, 2 (2009), 569–576. 2

[VMT97] VOLINO P., MAGNENAT-THALMANN N.: Developing simulation techniques for an interactive clothing system. In *Proc. Virtual Systems and MultiMedia* (1997), IEEE Computer Society, pp. 109–. 1

[VMTF09] VOLINO P., MAGNENAT-THALMANN N., FAURE F.: A simple approach to nonlinear tensile stiffness for accurate cloth simulation. *ACM Trans. Graph. 28*, 4 (2009), 105:1–105:16. 2

[WBS*13] WEBER D., BENDER J., SCHNOES M., STORK A., FELLNER D.: Efficient GPU data structures and methods to solve sparse linear systems in dynamics applications. *Computer Graphics Forum 32*, 1 (2013), 16–26. 2

[WOR10] WANG H., O'BRIEN J., RAMAMOORTHI R.: Multi-resolution isotropic strain limiting. *ACM Trans. Graph. 29*, 6 (2010), 156. 2

[Zel05] ZELLER C.: Cloth simulation on the gpu. In *ACM SIGGRAPH 2005 Sketches* (2005), ACM, p. 39. 2, 6

[ZH07] ZINK N., HARDY A.: Cloth simulation and collision detection using geometry images. In *Proc. Computer graphics, virtual reality, visualisation and interaction in Africa* (2007), ACM, pp. 187–195. 2