

Parallel Simulation of Cloth on Distributed Memory Architectures

B. Thomaszewski¹ and W. Blochinger²

¹ WSI/GRIS, Universität Tübingen, Germany
b.thomaszewski@gris.uni-tuebingen.de

² Symbolic Computation Group, Universität Tübingen, Germany
blochinger@informatik.uni-tuebingen.de

Abstract

The physically based simulation of clothes in virtual environments is a highly demanding problem. It involves both modeling the internal material properties of the textile and the interaction with the surrounding scene. We present a parallel cloth simulation approach designed for distributed memory parallel architectures, in particular clusters built of commodity components. In this paper, we focus on the parallelization of the collision handling phase. In order to cope with the high irregularity of this problem we employ a task parallel approach with fully dynamic problem decomposition. This leads to a robust algorithm, regardless of the complexity of the scene. We report on initial performance measurements indicating the usefulness of our approach.

Categories and Subject Descriptors (according to ACM CCS):

C.1.4 [Processor Architectures]: Parallel Architectures, G.1.3 [Numerical Analysis]: Numerical Linear Algebra, G.4.5 [Mathematical Software]: Parallel and Vector Implementations, I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

1. Introduction

In the last years, considerable research has been carried out in the field of cloth simulation. Major advances have been achieved on understanding the physical behavior of textiles and on deriving appropriate mathematical models along with sophisticated simulation methods. However, as a result of this development, we face enormous computational demands, especially for high quality animations which are based on high resolution models. We propose to employ parallelism to meet these computational requirements.

In the process of simulation, we can generally distinguish between two different stages within one time step:

- **Physical Modeling**

Internal forces resulting from deformation and external forces due to effects like gravity or wind are determined. Then, updates for nodal velocities and positions are computed according to Newton's law of motion.

- **Collision Handling**

Detection and handling of interactions of the garment with other objects in the scene, as well as self-interference. De-

pending on the actual method used, this results in motion constraints, repulsion forces or position and/or velocity updates for individual nodes.

One difficulty of parallel cloth simulation on distributed memory architectures originates from the very fine granularity of the physical modeling stage. In [KB04] we presented a data-parallel method for the modeling part, especially designed for minimizing inter-processor communication. This was achieved with a data decomposition approach using advanced graph-partitioning methods. In this paper, we deal with the parallelization and integration of the collision handling phase. In all, our work results in a comprehensive parallel textile simulation method which is capable to cope with complex scenes.

In the context of textile simulation, several intrinsic properties of collision handling make its parallelization most challenging. Basically, collision handling is a global problem, because any pair of processors can own interfering elements. Thus, communication cannot be limited to processors owning neighboring elements as within the modeling

phase. During the course of simulation, the geometry of the considered object can change significantly. This means that also communication partners are changing in a highly dynamic manner. Moreover, these interaction patterns cannot be predicted and are extremely unstructured. Together, these properties lead to a high degree of irregularity.

To the best of our knowledge, our work represents the first research effort on parallel collision handling for textile simulation on distributed memory architectures. The main contribution of this work is a task-parallel method for the collision handling process. We employ a fully dynamic problem decomposition to cope with the inherent irregularity.

The rest of our paper is organized as follows: In Section 2 we report on related work. Section 3 gives a brief account of state-of-the-art cloth simulation methods. In Section 4 we discuss our approach to parallel cloth simulation, focussing on parallel collision handling. We report on performance measurements in Section 5.

2. Related Work

2.1. Cloth Simulation

Physically based simulation is a widely adopted paradigm for reproducing the dynamic behavior of deformable surfaces like cloth. The research literature on cloth modeling is abundant and we refer the interested reader to the textbooks [VMT00] and [HB00]. The seminal work of Baraff and Witkin [BW98] laid the ground for fast and stable cloth simulation using implicit time stepping to solve the arising ordinary differential equations. Later extensions and developments addressed further physical as well as numerical aspects [EEH00, VMT01, CK02, EKS03, HE01]. Despite these advances, even on recent workstations the simulation of cloth with high resolution meshes (beyond 10000 vertices) is still very time consuming.

2.2. Parallel Cloth Simulation

Gutiérrez *et al.* [GRR*05] report on a cloth simulation method for NUMA parallel architectures which employs an implicit integration method for the modeling phase. Lario *et al.* [LGPT01] describe a rapid parallelization approach of a multilevel cloth simulator on shared-memory architectures using OpenMP. Zara *et al.* [ZfV04] deal with parallel cloth simulation on (distributed-memory) PC clusters employing both, explicit and implicit integration techniques.

The work of Zara *et al.* is the most related to the research presented in this paper both in terms of the employed numerical algorithms and in terms of the target parallel architecture. The other two approaches are based on shared address space parallel computers which are certainly more easy to program but do not scale well and/or have a worse price/performance ratio compared to distributed-memory architectures, like clusters built from commodity components.

A difference between our work and the work of Zara *et al.* is the way problem decomposition and task mapping for the modeling phase is carried out. While we perform a completely static approach based on data partitioning which minimizes inter-processor interaction, the work of Zara *et al.* is based on partitioning dynamically generated task dependency graphs.

In contrast to our work, all other approaches to parallel cloth simulation do not explicitly address collision handling. This limits their usefulness to simple scenes.

2.3. Parallel Contact Detection

Parallel contact detection has previously been studied in the context of various applications from the engineering domain, e.g. simulation of projectile penetration [Kar03] or simulation of foam compression [BASH00]. The basic principle of these approaches is to identify a subset of elements which can potentially get in contact. These elements are called surface elements and typically constitute only a small fraction of the total elements of the simulation. In [BASH00] a separate partitioning is used during the collision handling phase which exclusively considers surface elements. Alternatively multi-constraint, multi-objective graph partitioning algorithms are employed to avoid expensive relocation actions between the two phases [Kar03]. In cloth simulation, every element is a surface element and it is not possible to predict the set of elements which actually interfere. Thus, approaches based on static partitioning are not suitable for collision handling in parallel cloth simulation.

3. Physically Based Cloth Simulation

3.1. Physical Model

For the physical model we rely on an approach based on continuum mechanics. The basic quantities are *strain* which is a dimensionless deformation and *stress* which is a force per area or length. The two are connected via a constitutive relation, i.e. a material law which in our case is linear. The result of this approach is a partial differential equation which has to be discretized in space using numerical methods. To this end, we use a linear finite element approach as described in [EKS03]. This yields a system (or stiffness) matrix relating nodal displacements of the mesh to forces acting on the nodes. Because only local neighbors have influence on the force on one node, this matrix is sparse. The system is extended to account for dynamic motion according to Newton's second law, leaving the following system of coupled ODEs

$$\begin{aligned}\dot{x}(t) &= v(t) \\ \dot{v}(t) &= M^{-1}f(x(t), v(t)).\end{aligned}$$

To obtain the dynamic evolution of the system these equations have to be stepped forward in time. In the case of cloth simulation the system equations are inherently stiff and are

thus susceptible to instabilities when using explicit integration schemes. Since the work of Baraff et al. [BW98] the computer graphics community has settled on using implicit integration schemes for cloth simulation. The heart of this method is the solution of a sparse LES which can be carried out in a convenient way using the conjugate gradient (cg) method [She94].

3.2. Collision Handling

Besides the simulation of the intrinsic properties of cloth the interaction with its environment has to be modeled. This involves the detection of any collisions and an adequate response to prevent the clothes from intersections. The proper treatment of these two components (to which we refer as collision handling in the remainder) is a very complex task [THM*05]. While the physical simulation engine computes new states at distinct intervals only, collisions can occur at any instant in between such intervals. Algorithms that handle these cases in a robust way are often very complex and time consuming such that the collision handling step soon becomes a bottleneck in the simulation pipeline.

Basically, detecting interference between two arbitrarily shaped objects breaks down to determining the interference between all of the primitives (i.e. faces, edges, and vertices) of one mesh with every primitive of the mesh representing the other objects. With complex objects comprising thousands of faces, this approach soon becomes very expensive.

A common way to accelerate the interference tests is to structure the objects under consideration hierarchically with bounding volumes. Usually, a bounding volume hierarchy (BVH) is constructed for each object in the scene (including deformable as well as rigid objects) in a preprocessing step in the following way (see Fig. 1 and 2): a bounding volume enclosing the entire object is set as the root node of the tree representing the hierarchy. This node is then subdivided recursively until a leaf criterion is reached. Usually, the leaves contain one single primitive.

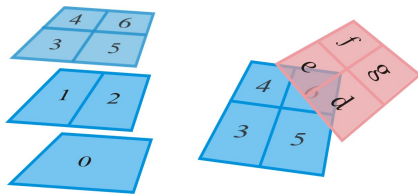


Figure 1: Interfering objects. Left: Different levels of the BVH. Right: Overlapping Faces.

For our implementation we use the approach described in [MKE03] which is based on a BVH with discrete oriented polytopes (k-Dops) as bounding volumes. More specifically, we use binary trees with 18-Dops, i.e. the bounding volumes are enclosed by 18 planes with predefined (discrete)

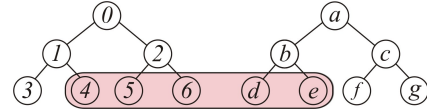


Figure 2: BVH structure for the two objects in Fig. 1. Overlapping leaf nodes are marked.

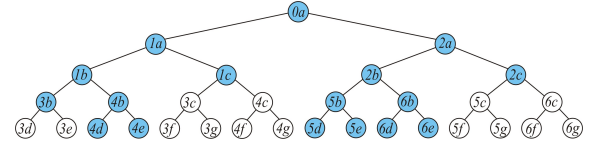


Figure 3: Test tree for the colliding objects shown in Fig. 1.

orientation. With a BVH constructed, the test for intersection between two objects now proceeds as follows: first, the bounding volumes corresponding to the root nodes of the two hierarchies are tested for intersection. Only if these two overlap are the corresponding children bounding volumes recursively tested for intersections (see Fig. 3). Besides the detection of interference with other objects the cloth can also intersect with itself. Basically, the same algorithms can be used to find these self collisions but here, an efficient strategy is even more important. Usually, criteria based on surface curvature are used to rule out non-intersecting parts of the cloth quickly (cf. [VT95]). Finally, the interference test delivers primitives that are close to each other or intersect.

A robust method to prevent the imminent intersection was presented by Bridson et al. [BFA02]. We use a simpler approach based on constraints which allows for an efficient implementation in our context. If the distance of a feature of one object to a primitive of another object is inferior to a certain threshold we constrain the movement of the corresponding vertices such that further approaching is prevented. The resulting constraints can be enforced with a filter procedure inside the cg-method (cf. [BW98] and [AB03]) which is used to solve the LES arising in the context of the implicit Euler integration scheme. Once the collision is resolved (i.e. the features are no longer approaching) the constraints can be released. Self collisions are more complicated to treat and it is difficult to obtain good results with constraints in this case. We therefore chose to employ an impulse-based treatment for the case of self collisions. Instead of constraining the movement of the involved vertices we apply an impulse (i.e. adjust the velocity) to them in order to prevent intersections.

4. Parallel Cloth Simulation

In this section, we describe our parallel cloth simulation approach, focusing on the collision handling phase. In order to provide an appropriate context, we first give a very brief account of our previous work on the parallelization of the



Figure 4: Partitions of a mesh shown in different colors for 12 processors.

physical modeling phase. A more detailed treatment of this topic can be found in [KB04].

4.1. Parallel Physical Modeling

In order to parallelize the physical modeling phase, we apply a data decomposition scheme. The basic idea is to partition the vertices of the input mesh into regions with roughly the same amount of vertices and assign the regions to the processors. The position of neighboring vertices which belong to different processors have to be communicated in every iteration of the cg procedure at the core of the LES solver. It is therefore crucial that the partitioning also minimizes communication overhead since otherwise this soon becomes a bottleneck. To this end, we use graph partitioning techniques which minimize the number of edge cuts and thus communication among the processors. Figure 4 shows an exemplary partitioning of a shirt.

After the initial decomposition stage, every processor holds its own parts of the global position, vertex and normal vectors. Each processor then sets up its local system matrix and right hand side vector corresponding to its attributed vertices. Subsequently, the LES is solved in parallel in an SPMD style.

4.2. Parallel Collision Handling

The specific challenge of parallelizing the collision handling process originates from its high irregularity as explained in Section 1. Facing this situation, employing static partitioning (like in the modeling phase) is not sufficient. Depending on the actual locations of the collisions, the amount of time spent in collision handling would differ considerably among the processors. The resulting high degree of processor idling during collision handling ultimately limits the parallel efficiency of the whole execution process. Moreover, the location of collisions can change considerably during the course of the simulation and cannot be predicted. Thus, in order to keep all processor busy during the collision handling phase a more dynamic approach to problem decomposition is needed.

Generally, we can distinguish between two different types of collisions: external collisions and self collisions. To detect the first type we have to test our deformable object against every other (rigid or deformable) object in the scene. For the latter case, the deformable object has to be tested against itself. Next, we show how this basic collision handling algorithm can be integrated into the SPMD framework of our cloth simulator. Later we describe how we deal with the irregularity of collision handling using a task parallel approach which is based on fully dynamic problem decomposition.

4.2.1. Parallel Collision Handling Framework

As already pointed out, we use a bounding volume hierarchy to speed up the collision detection stage. For parallel execution, this hierarchy is built as follows: The problem decomposition stage supplies us with a number of disjoint partitions of the vertices of the input mesh. For each processor we now proceed in the following way: a local mesh is constructed corresponding to the attributed vertices. Then, a BVH hierarchy is set up on this mesh using a top-down approach. Once this is done, we combine the root nodes of the different processors to form a global hierarchy of the mesh. Testing a textile for interference with other objects is now carried out in the standard manner. First, the root node of the garment's BVH is tested against the other object's root node. If they overlap, the trees of the processors are recursively tested. This approach works well for standard collisions but for self collisions a different strategy has to be taken. In our specific context we can again distinguish between two different types of self collisions: namely collisions between sub-meshes of different processors and those that are real self collisions on the processor-local mesh. For the latter case we can use existing techniques since this corresponds to the usual self collision problem. For the case of inter-processor self collisions we test the corresponding BVHs against each other similar to the way standard collisions are treated. All described BVH tests form a set of top-level tasks for collision handling. In the following, we discuss a task-parallel approach which dynamically decomposes top-level tasks into smaller sub-tasks.

4.2.2. Dynamic Problem Decomposition Procedure

Our method for dynamic problem decomposition is based on a modification of the BVH testing procedure. For dynamically generating tasks, we introduce a stack data structure which records untried alternatives of the BVH testing tree. In the BVH testing procedure, expansion of a tree node results in two additional tree nodes each representing a test. As in the sequential procedure, the first test is carried out by starting a new recursion level. However, before entering the recursion, the second test is pushed on the stack. Figure 5 shows a snapshot of a BVH testing process along with the corresponding state of the stack.

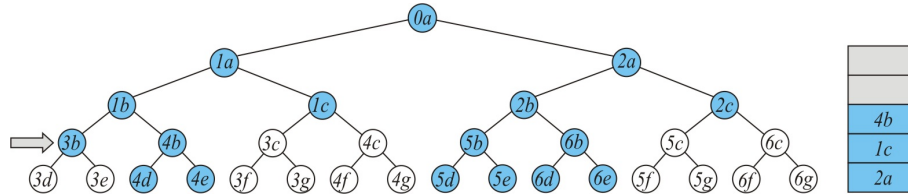


Figure 5: Dynamic problem decomposition. The arrow indicates the current state of the BVH testing procedure. The stack on the right stores the current untried alternatives.

Tests which are recorded on the stack can be executed in one of the following ways:

- A test can be removed from the top of the stack and executed sequentially when the recursion gets back to the current level. This corresponds to the procedure of the original algorithm.
- A test can be removed from the bottom of the stack and assigned to a newly generated task. This task executes a BVH testing procedure for which the assigned test represents the root of the BVH testing tree.

The rationale behind taking nodes from the bottom of the stack for creating new tasks is that such nodes have a higher potential of representing a large testing tree since they are closer to the root. This heuristic helps to prevent that tasks with too fine a granularity are generated. For highly irregular problems, initially several tests can be removed at once from the stack and assigned to one task. The described method for dynamic problem decomposition is triggered by the load balancing procedure as discussed in the next section.

4.2.3. Dynamic Load Balancing

For controlling the overall amount of generated parallelism, we use a self-adapting approach where the dynamic problem decomposition and the load balancing process are tightly coupled. Specifically, our method is based on the distributed task pool model, i.e. every processor maintains a local task pool. Upon creation, a task is first placed in the local task pool. Subsequently, it can be instantiated and executed locally, when the processor gets idle. It also might first be transferred to a remote task pool for load balancing purposes.

Tasks are dynamically generated using the decomposition procedure discussed previously. A decomposition operation is initiated when the size of the local pool falls below a given threshold. In order to prevent that too much parallelism is generated, a consecutive task decomposition can only take place when a given time interval (e.g. 10ms) has elapsed. For transferring threads between task pools we employ a receiver initiated scheme. When a processor runs idle and the local task pool is empty, it tries to steal tasks from remote pools. The victim node is chosen randomly. If the request is not satisfied within a given period of time another victim node is chosen and a corresponding request is issued.

In the context of our application this approach establishes self-adapting parallelism. For regular scenes where collisions are evenly distributed on the processors, no further task decomposition and load balancing is carried out. In contrast, if processors run idle due to an uneven distribution of the collisions, additional parallelism is dynamically generated and balanced over the processors.

4.2.4. Implementation

The implementation of the previously described methods is based on the parallel system platform DOTS [BKLwW99]. DOTS provides extensive support for the multithreading parallel programming model (not to be confused with the shared-memory model) which is particularly suited for task-parallel applications that employ fully dynamic problem decomposition.

DOTS Programming Model The key concept of the DOTS programming model are *thread group* objects which serve as links between different primitives of the API. Upon creation, a thread is either *explicitly* or *implicitly* placed to a thread group, calling `dots_fork` or `dots_hyperfork`, respectively. In the former case, the thread group object has to be supplied as argument to `dots_fork`. In the latter case, the thread is placed implicitly in the same thread group as its parent thread. In both cases, a procedure to be executed by the child thread and an argument-object has to be supplied. For all subsequently applied primitives it is not relevant whether a thread has been placed explicitly or implicitly into a thread group. Threads return result objects employing `dots_return`. The `dots_join` primitive is used to retrieve results of threads from a given thread group applying *join-any* semantics: The first result which becomes available from any thread in the group is delivered. If no results are available, the calling thread is blocked until a thread of the group delivers a result. If a thread has finished execution and all result objects of the thread have been joined, it is removed from the thread group. By checking the return value of `dots_join`, termination of a computation can be determined.

Parallel Collision Handling Using DOTS For starting the task-parallel execution process, we create on every processor threads which execute the top-level tasks for collision handling as described in Section 4.2.1. Threads that execute

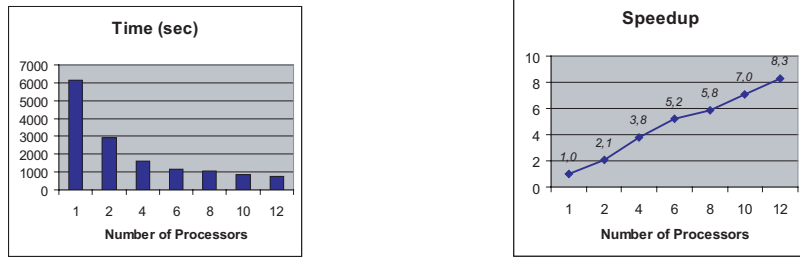


Figure 6: Results of performance measurements for scene 1.

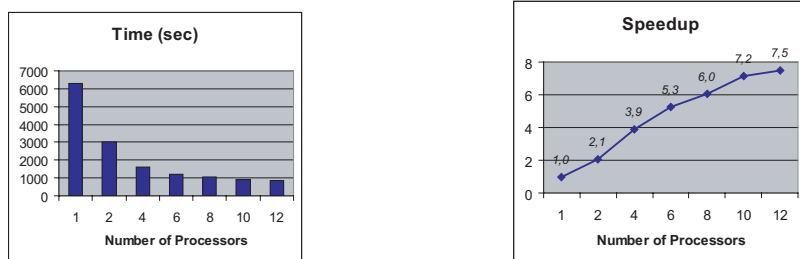


Figure 7: Results of performance measurements for scene 2.

top-level tasks are created using the `dots_fork` primitive. All threads are placed into the same thread group. Upon completion, each thread delivers a set of constraints which represent an appropriated collision response for the corresponding top-level collision handling task.

Tasks resulting from dynamic problem decomposition (cf. Section 4.2.2) are modeled by DOTS threads that are created with the `dots_hyperfork` primitive. This approach makes it possible to easily synchronize with the completion of the collision handling phase regardless how many decomposition operations took place. We simply apply `dots_join` operations on the thread group until termination of the execution is indicated. (Note that the actual number of generated tasks largely depends on the considered scene and cannot be statically determined.) Every time a thread is joined, the resulting set of constraints is stored and applied in the next modeling phase.

Using the extension framework of DOTS we integrated the load balancing scheme described in Section 4.2.3.

5. Performance Measurements

5.1. Test Scenarios

We evaluate the performance of our approach using two test scenarios. To demonstrate the robustness of our method we focused on problems with a high degree of irregularity.

The first scene consists of a square cloth comprising 10000 vertices which drapes over a sphere lying on the floor (see Figure 8). The cloth is offset from the center of the sphere such that initially, collisions only occur in a locally

restricted region. Only when the cloth reaches the floor, collision occur (almost) everywhere in the mesh and the setting becomes more regular. In this scenario rigid collisions are predominant while self collisions appear only marginally.

For the second test scene we let the same piece of cloth drape over a slightly smaller sphere (see Figure 9). (We could as well use any other polygonal mesh as a collision object but the sphere is sufficient for our interests.) In this case however, the movement of the cloth is not vertically constrained by a floor such that inter-processor self collisions occur at the tips of the cloth. Additionally, normal self collision appear more accentuated.

5.2. Results

For carrying out performance measurements we used a Linux based cluster. All compute nodes are equipped with Intel Xeon processors running at 2.667 GHz and with 2 GB of main memory. The nodes are connected by a Myrinet-2000 high-speed network.

All subsequently presented performance results are based on the arithmetic mean of the wall-clock times of three individual parallel runs for each investigated setting. The time values given for one processor are based on a sequential version of our application that employs sequential data structures and sequential arithmetic operations.

Figure 6 and Figure 7 show the results of the performance measurements for the previously described scenes. Despite the high degree of irregularity, for both scenes the computation time can be substantially decreased using parallelism.

For computations on 2 processors super-linear speedups can be observed. In data parallel applications the main source of super-linear speedups are cache effects. With an increasing number of processors the data working-set of each individual processor becomes smaller, resulting in an improved cache performance.

6. Conclusion

In this paper, we presented a parallel approach for the collision handling phase of cloth simulation. Our approach employs task-parallelism with fully dynamic problem decomposition to cope with the high irregularity of the collision handling problem. We integrated our method into an SPMD based parallel cloth simulator. The conducted performance measurements delivered substantial speedups for scenes with a high degree of irregularity. A future direction could be to extend our work to an hierarchical approach, where shared-memory and distributed-memory parallelism is combined.

Acknowledgements

We would like to thank Johannes Mezger for letting us use his sequential k-dop collision detection implementation and for the valuable discussions.

References

- [AB03] ASCHER U., BOXERMAN E.: On the modified conjugate gradient method in cloth simulation. *The Visual Computer* (2003).
- [BASH00] BROWN K., ATTAWAY S., S.J.PLIMPTON, HENDRICKSON B.: Parallel strategies for crash and impact simulations. *Computer Methods in Applied Mechanics and Engineering* 184 (2000), 375–390.
- [BFA02] BRIDSON R., FEDKIW R. P., ANDERSON J.: Robust Treatment of Collisions, Contact, and Friction for Cloth Animation. In *Computer Graphics (Proc. SIGGRAPH)* (2002), pp. 594–603.
- [BKLwW99] BLOCHINGER W., KÜCHLIN W., LUDWIG C., WEBER A.: An object-oriented platform for distributed high-performance Symbolic Computation. *Mathematics and Computers in Simulation* 49 (1999), 161–178.
- [BW98] BARAFF D., WITKIN A.: Large Steps in Cloth Simulation. In *Computer Graphics (Proc. SIGGRAPH)* (1998), pp. 43–54.
- [CK02] CHOI K.-J., KO H.-S.: Stable but Responsive Cloth. In *Computer Graphics (Proc. SIGGRAPH)* (2002), pp. 604–611.
- [EEH00] EBERHARDT B., ETZMUSS O., HAUTH M.: Implicit-Explicit Schemes for Fast Animation with Particle Systems. In *Eurographics Computer Animation and Simulation Workshop* (2000).
- [EKS03] ETZMUSS O., KECKEISEN M., STRASSER W.: A Fast Finite Element Solution for Cloth Modelling. *Proc. Pacific Graphics* (2003).
- [GRR*05] GUTIERRÉZ E., ROMERO S., ROMERO L. F., PLATA O., ZAPATA E. L.: Parallel techniques in irregular codes: cloth simulation as case of study. *Journal of Parallel and Distributed Computing* 65, 4 (April 2005), 424–436.
- [HB00] HOUSE D. H., BREEN D. E. (Eds.): *Cloth Modeling and Animation*. A K Peters, 2000.
- [HE01] HAUTH M., ETZMUSS O.: A High Performance Solver for the Animation of Deformable Objects using Advanced Numerical Methods. In *Computer Graphics Forum* (2001), pp. 319–328.
- [Kar03] KARYPIS G.: Multi-constraint mesh partitioning for contact/impact computations. In *SC '03: Proceedings of the 2003 ACM/IEEE conference on Supercomputing* (Washington, DC, USA, 2003), IEEE Computer Society, p. 56.
- [KB04] KECKEISEN M., BLOCHINGER W.: Parallel implicit integration for cloth animations on distributed memory architectures. In *Proc. of Eurographics Symposium on Parallel Graphics and Visualization 2004* (Grenoble, France, June 2004).
- [LGPT01] LARIO R., GARCIA C., PRIETO M., TIRADO F.: Rapid Parallelization of a Multilevel Cloth Simulator Using OpenMP. In *Third European Workshop on OpenMP* (2001).
- [MKE03] MEZGER J., KIMMERLE S., ETZMUSS O.: Hierarchical Techniques in Collision Detection for Cloth Animation. *Journal of WSCG* 11, 2 (2003), 322–329.
- [She94] SHEWCHUCK J. R.: An Introduction to the Conjugate Gradient Method Without the Agonizing Pain, 1994. <http://www.cs.cmu.edu/quake-papers/painless-conjugate-gradient.ps>.
- [THM*05] TESCHNER M., HEIDELBERGER B., MANOCHA D., GOVINDARAJU N., ZACHMANN G., KIMMERLE S., MEZGER J., FUHRMANN A.: Collision Handling in Dynamic Simulation Environments. In *Eurographics Tutorials* (2005), pp. 79–185.
- [VMT00] VOLINO P., MAGNENAT-THALMANN N.: *Virtual Clothing*. Springer, 2000.
- [VMT01] VOLINO P., MAGNENAT-THALMANN N.: Comparing Efficiency of Integration Methods for Cloth Animation. In *Computer Graphics International Proceedings* (2001).
- [VT95] VOLINO P., THALMANN N.: Collision and Self-Collision Detection: Efficient and Robust Solutions for Highly Deformable Surfaces. In *Comp. Animation and Simulation* (1995).
- [ZfV04] ZARA F., FAURE F., VINCENT J.-M.: Parallel simulation of large dynamic system on a pcs cluster: Application to cloth simulation. *International Journal of Computers and Applications* (march 2004). special issue on cluster/grid computing.

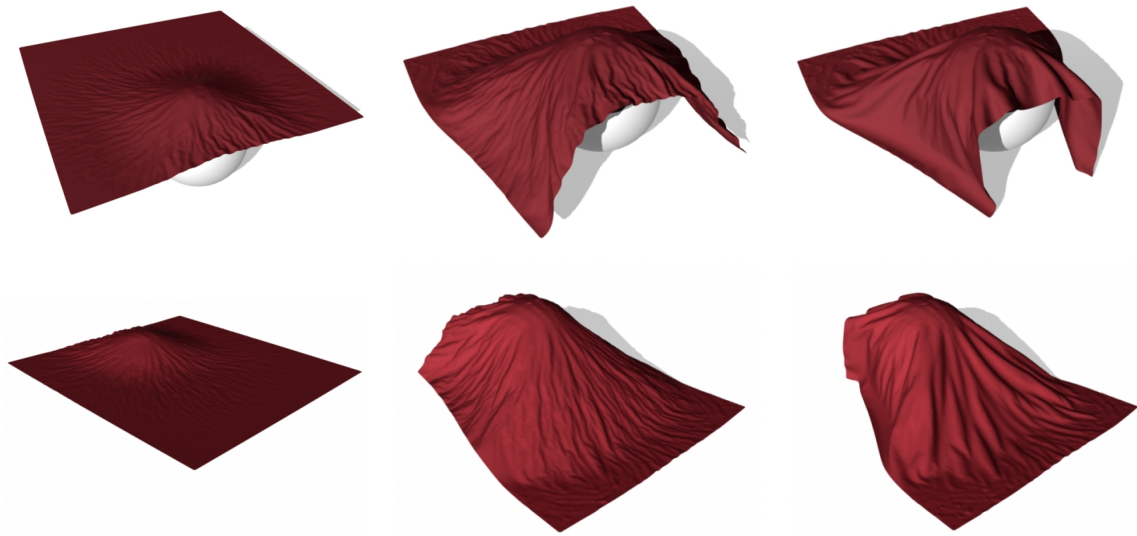


Figure 8: *Test Scene 1 (shown from two different view angles)*



Figure 9: *Test Scene 2 (shown from two different view angles)*