Eurographics Symposium on Parallel Graphics and Visualization (2004)
Dirk Bartz, Bruno Raffin and Han-Wei Shen (Editors)

Parallel Implicit Integration for Cloth Animations on Distributed Memory Architectures

M. Keckeisen[1] and W. Blochinger[2]

[1] WSI/GRIS, Universität Tübingen, Germany
keckeisen@gris.uni-tuebingen.de
[2] Symbolic Computation Group, Universität Tübingen, Germany
blochinger@informatik.uni-tuebingen.de

Abstract

We present a parallel cloth simulation engine designed for distributed memory parallel architectures, in particular clusters built of commodity components. We focus on efficient parallel processing of irregularly structured and real-world sized problems typically occurring in the simulation of garments. We report on performance measurements showing a high degree of parallel efficiency and scalability 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.1.7 [Numerical Analysis]: Ordinary Differential Equations, G.4.5 [Mathematical Software]: Parallel and Vector Implementations, I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling, I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

1. Introduction

Cloth animation has been an area of active research in computer graphics for several years. Recently, major advances have been achieved in understanding the physical behavior of cloth, the derivation of mathematical models, and the development of fast numerical algorithms and rendering techniques. Although much progress has been made towards visually pleasing animations, as well as towards physically correct simulations, computational demands for high resolution textiles are still very high. Especially if we are interested in high quality animations with comparatively long simulation times, the necessary computational performance is not yet reached. Parallel computing is able to significantly improve the performance of computationally intensive problems. A major challenge of applying parallel techniques to cloth animation originates from its inherent very fine granularity. Each step in the simulation depends on the results of the previous step. This property makes an inter-step parallelization approach not feasible. Consequently, parallelization has to be carried out at the level of an individual simulation step. Compared to similar parallel simulation applications from other domains, typical problem sizes in cloth animation (in terms of the number of unknowns to be computed within each step) are rather small, leading to a poor computation to communication ratio. Moreover, we are dealing with an unstructured problem where naïve approaches for task decomposition and mapping (e.g. not taking into account specific properties of the problem instance) typically lead to unscalable parallel applications.

In this paper, we present our parallel cloth simulator ParTüTex which addresses the aforementioned challenges. The major design goals of our parallel cloth simulator are:

- Efficient parallel processing of irregularly structured problem instances typically resulting from the simulation of garments.
- Achieving good parallel scalability on relevant problem sizes.
- Execution on cost-efficient distributed memory parallel architectures, in particular clusters built of commodity components.

We employ a data-parallel approach of parallel programming using static task decomposition and task mapping techniques to realize the above design goals.

2. Related Work

2.1. Cloth Animation

Since the work of Baraff and Witkin [BW98], implicit time stepping, together with the conjugate gradient (CG) method to solve the systems of linear equations, has become the standard method to solve the ordinary differential equations that arise in cloth animations [VMT00, VMT01, CK02, CK03, KSFS03, EKS03]. While many variants and mixed implicit-explicit methods have been proposed [EEH00, HE01, MDDB01, KCCL01], [CMT02, BMF03, AB03, HES03], current algorithms are still far from computing high resolution textiles (let's say a garment with about 18,000 vertices, as stated in [BWK03]), in real time. Moreover, if we consider geometrically more complex garments, we might even want to use even higher resolutions.

2.2. Parallel Cloth Animation

Due to the aforementioned performance limitations, parallelization of numerical algorithms lying at the core of modern cloth simulators is an attractive way of substantially accelerating computation time, but to our best knowledge only a few contributions have been made in the past.

Romero *et al.* [RRZ00] present a parallel cloth simulation method based on implicit integration designed for NUMA parallel architectures. Lario *et al.* [LGPT01] report on a rapid parallelization approach of a multilevel cloth simulator using OpenMP. The most recent contribution in this field has been made by Zara *et al.* [ZFV02, ZFV04]. This work deals with parallel cloth simulation on 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 clusters built of commodity components. The main difference between the presented work and the work of Zara *et al.* is the way problem decomposition and task mapping is carried out. While we perform a completely static approach based on data partitioning Zara *et al.* carry out dynamic problem decomposition based on partitioning dynamically generated task dependency graphs. This fundamental difference is also reflected by the underlying parallel programming models. Zara *et al.* employ the Athapascan task-parallel language while our work is based on PETSc (Portable, Extensible Toolkit for Scientific Computation) which provides an extensive set of data parallel primitives on top of the message passing programming model.

3. Implicit Integration for Cloth Animations

In this section, we briefly explain implicit integration in the context of cloth animations. Moreover, we describe the cloth simulation engine TüTex, which we use in this work.

3.1. Implicit versus explicit integration

In order to obtain a cloth animation, we have to compute the motion of the vertices of a given polygonal mesh representing the cloth. The motion depends on internal and external forces acting on the cloth, and its mass. The ordinary differential equations we have to solve arise from Newton's equation of motion

$$f(x(t), v(t)) = Ma(t),$$

where x represents the 3D positions of the vertices at a time t, $v = \dot{x}$ the respective velocities, $a = \ddot{x}$ the accelerations, f the forces acting on the cloth and M its mass. We can rewrite this equation as

$$\dot{x}(t) = v(t) \quad (1)$$

$$\dot{v}(t) = M^{-1} f(x(t), v(t)). \quad (2)$$

Note that x, v, a and f are vectors of size $3n$, and M is a quadratic diagonal matrix of dimension $3n$, where n is the number of vertices in the mesh.

Thus, given the forces at a time t, we first have to compute $v(t)$ and then $x(t)$ by solving these two equations. While there exist many explicit and implicit integration schemes to achieve this, the respective Euler methods are among the most widely used for cloth animations because of their simplicity and their relative low computational costs.

The explicit Euler integration method iteratively computes approximate solutions to equations 1 and 2 by

$$x(t+h) = x(t) + hv(t)$$

$$v(t+h) = v(t) + hM^{-1} f(x(t), v(t)),$$

while the implicit Euler integration method achieves this by

$$x(t+h) = x(t) + hv(t+h) \quad (3)$$

$$v(t+h) = v(t) + hM^{-1} f(x(t+h), v(t+h)), \quad (4)$$

where h denotes the time step. The main difference between these two methods is the fact that the implicit method takes forces in the time step $t+h$ into account. This yields more stable solutions at the cost of having to solve for $v(t+h)$ rather than just evaluating a formula as in the explicit Euler method [BW98]. In general, f is a non-linear function which has to be linearized in order to reduce the problem to a system of linear equations. Baraff and Witkin [BW98] give an example of how to linearize a physically based force model and show that the arising sparse linear equation systems can be computed efficiently by using a conjugate gradient method [She94]. Moreover, they describe how constraints can be realized by using a filter function within the conjugate

gradient method, e.g. for pinning particles to specific points in the scene or for collision response.

3.2. The TüTex Cloth Simulation Engine

As seen in the last section, the implicit Euler method leads to a system of linear equations in each time step. Thus, the simulation loop in the animation is structured as shown in algorithm 1.

Algorithm 1 Simulation Loop

loop
  setup LES
  Compute the matrix A and the right hand side vector b.
  solve LES
  Compute the new velocities by solving $Av(t+h) = b$ using the CG method
  compute positions
  Compute the new positions by evaluating $x(t+h) = x(t) + hv(t+h)$
  if reached frame interval **then**
    generate frame
  end if
end loop

Mostly, there will be additional steps for collision detection and response and for rendering the intermediate results or writing them to a file.

The specific computation of A and b depends on the physical model that is used. In most cases, for example in simple mass spring systems, internal forces are modelled locally between vertices in the mesh that are connected by an edge. This means, the internal forces acting on a vertex depend only on the positions and velocities of its neighbors. Thus, the resulting matrix A is sparse and has non-zero entries only in the form of $3x3$ blocks beginning at row $3i$ and column $3j$, if there is an edge between vertex i and vertex j. Moreover, A is obviously symmetric.

In this work we use the TüTex cloth simulation engine which uses a physical force model based on a finite element discretization of the linear Cauchy strain tensor. By constructing a local reference frame for each element in each time step this linear strain formulation can be applied to cloth animations, which leads to efficient and physically accurate results [EKS03]. Moreover, an implicit Euler method is used to solve the ordinary differential equations, as described in the last section. Thus we have to solve a system of linear equations in each time step. The respective matrix also has non-zero entries exactly for all edges in the mesh. For a detailed description of how to set up the system of linear equations we refer to [EKS03].

4. The ParTüeTex Cloth Simulator

Our parallel cloth simulator ParTüTex is built on top of the TüTex cloth simulation engine employing PETSc [BGMS97, BBG*02] for enabling parallel execution. PETSc is a suite of parallel data structures and routines for the scalable parallel solution of scientific applications. It is based on the MPI (Message Passing Interface) standard and supports an SPMD (Single Program Multiple Data) style of parallel programming which is located at a higher level of abstraction than the pure SPMD message passing programming model. PETSc has been used for parallelizing applications from a wide range of domains [BBG*01]. Besides ready-to-use standard components (e.g. parallel linear equation solvers), it also provides a rich set of lower-level primitives for dealing with advanced issues like the parallelization of irregularly structured problems.

4.1. Task Decomposition and Task Mapping

Two important design issues in parallel programming are task decomposition and task mapping. Both can be accomplished statically or dynamically. While dynamic approaches are more flexible and are also inescapable in some application domains, static schemes generally impose less parallel run-time overhead. Consequently, static approaches should be preferred whenever possible in order to achieve good scalability. This is especially important when dealing with relatively small problem instances which typically exhibit a poor computation to communication ratio.

We employ a static task decomposition and mapping scheme which is based on data decomposition. The goal of data decomposition is to partition the data structures on which computations are performed (which are essentially vectors and matrices in our case) in order to induce a decomposition into parallel tasks.

Generally, a partitioning method should optimize the following (commonly conflicting) objectives to attain high parallel efficiency:

- balancing of computational load
- minimizing communication overhead

Balancing computational load requires a task mapping that assigns to all processors the same amount of work. In our case in each step identical operations are carried out on individual data elements and all matrices are unstructured. Thus an even partitioning of the data structures also leads to a balanced computational load. Moreover, this property makes it possible to limit the number of generated parts (and hence the number of induced parallel tasks) to the number of available processors. This results in a static one-to-one partition to task to processor mapping. For matrices we choose a 1-dimensional row-oriented parallel layout, vectors are aligned accordingly.

For minimizing inter-task communication it is not sufficient to generate any balanced partition, but we must also

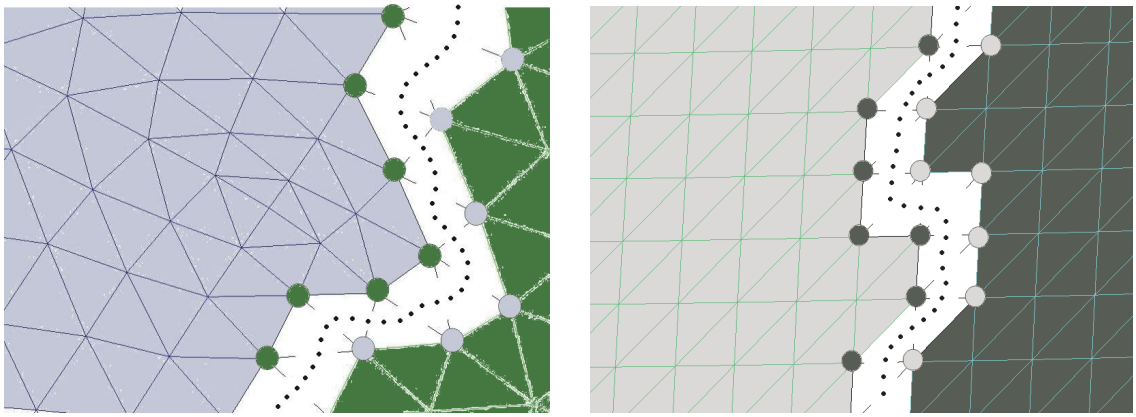

Figure 1: *Examples of ghosted meshes. The left image shows a part of a shirt's triangulation, the right image is taken from a square piece of cloth (cf. Section 5)*

decide for each individual data element to which partition it should belong to according to data-dependencies occurring within its computation. Conceptually, this process delivers a new ordering of the data elements, called the parallel numbering. In the parallel numbering each processor owns a consecutive range of vertex numbers.

Below we identify all data dependencies within the simulation for a 1-dimensional row-oriented parallel data layout:

- Setup LES
  - computation of the matrix and the right hand side requires communication of all non-local mesh neighbors.
- Solve LES (Conjugate Gradient Method)
  - dense vector inner product requires one all-reduce collective communication operation. Note, that in general the communication overhead of parallel dense vector inner product does not depend on the ordering of the data elements.
  - sparse matrix vector multiplication requires communication of non-local vector elements that correspond to non-zero entries of each matrix row.

Since the mesh structure is fixed and translates directly into the sparsity pattern of all involved matrices (non-zero entries indicate mesh neighbors), all task interaction patterns remain the same throughout the whole computation. Moreover, the Setup LES phase and the Solve LES phase can be optimized jointly.

In our case, we are dealing with unstructured sparse matrices, making specific partitioning schemes impossible. Therefore, we employ generic graph partitioning techniques. The graph partitioning problem deals with determining a partitioning of a graph with equally sized parts and a minimum number of edge cuts for a given partitioning size. Since in our application edge cuts indicate inter-task communication, graph partitioning reduces the overall communication overhead and at the same time preserves a balanced workload.

4.2. Parallel Algorithm

Algorithm 2 shows the pseudo code of our SPMD based parallel algorithm. Subsequently, we discuss the main steps of the algorithm and give some explanations how we have implemented the steps employing PETSc constructs.

Algorithm 2 SPMD based parallel algorithm

```
partition mesh
redistribute positions vector
loop
  update ghost values
  setup LES
  solve LES
  compute positions
  if reached frame interval then
    gather vector x
    if NODE-ID = 0 then
      generate frame
    end if
  end if
end loop
```

For mesh partitioning we use a parallel multilevel k-way graph partitioning method which is provided by the ParMetis [KK96] graph partitioning library. PETSc features an interface for accessing ParMetis functionality in a straight forward manner. The result of this step is a so called *index set* that represents a mapping between the application specific vertex numbering and the new parallel numbering induced

by the partitioning process. Basically, index sets are used for defining communication patterns of collective operations.

Before entering the parallel simulation loop, the initial positions vector has to be redistributed according to the determined parallel numbering. PETSc supports this process by means of its generic vector-scatter collective communication primitives.

Vertices that are adjacent of a cut edge (i.e. vertices located on the border of a partition) are accessed by both corresponding tasks during matrix setup and sparse matrix-vector multiplication steps. Such vertices are called ghost-points, because they physically belong to one, but logically belong to two processors. Figure 1 shows two examples of ghosted meshes. As PETSc is based on the message passing model, ghost-points have to be explicitly communicated by a collective communication operation at the beginning of each iteration of the simulation loop. Since this communication operation is highly performance critical, we use overlapping techniques. The part of the computation of the right hand side vector not depending on ghost vertices is carried out while the messages are in transit. In PETSc, overlapping of computation and communication is accomplished by placing code between calls of *VectorScatterBegin()* and *VectorScatterEnd()* primitives.

In order to realize a constraint enabled conjugate gradient method (cf. Section 3.1) we extended the parallel conjugate gradient component of PETSc by a filter-hook. This functionality allows us to register a custom hook function that is called within the CG procedure at appropriate places providing access to internal variables which can be modified within the hook function applying a filter procedure.

For generating frames, the computed positions have to be gathered on one node and at the same time permuted to the original application specific numbering. This gathering and permutation process is accomplished by PETSc vector-scatter collective communication primitives.

5. Performance Measurements

In this section we first describe the test scenarios we used to evaluate our approach. Then we discuss the results of our measurements.

5.1. Test Scenarios

We verified our approach with two test scenes.

In the first test we simulated a quadratic piece of cloth under the influence of gravity (see figure 4). The cloth has a size of one square meter and consists of 22,500 vertices and 44,402 triangles. In the rest state, the vertices form a uniform 150x150 grid, where the vertical and horizontal edges have a length of $\frac{2}{3}$ cm each, while the diagonal edges have a length of about 1cm (see the right image in figure 1). We think this is an appropriate discretization if we want to model small folds and wrinkles.

The cloth is fixed at three points and slides onto the floor. To treat the collisions we implemented a very simple collision detection and response. At the end of each time step the z coordinate of each vertex is compared to the floor height. If a collision is detected we correct the vertex position and velocity. Obviously, this is straightforward to parallelize. A parallel collision detection and response scheme for arbitrary objects is the subject of future work.

The second test scenario consists in a shirt which is fixed at two vertices (see figure 5). Here, the triangulation has no regular pattern, as can be seen in the left image in figure 1. The shirt consists of 35,024 vertices and 69,648 triangles.

Both simulations were computed at a time step of $h = 0.001$ and with a relative error tolerance (relative decrease in the residual norm) of 0.001 in the conjugate gradient method, leading to an average of 647 iterations per time step for the square cloth and of 1,115 for the shirt. The first simulation ran for a simulation time of 0.48s and the second for 0.36s. Intermediate results were collected from all processor nodes and written to files at a rate of 25 frames per second.

These two test scenes are quite demanding for the numerical solver because in both examples there occur high internal forces in the textiles (note that there is no post-correction of the edge lengths undertaken as in [Pro95], instead we use measured material parameters as described in [EKS03]) and nearly all the vertices are subject to large relative movements for the whole simulation.

Of course, the overall computation time depends heavily on the specific physical configuration of the animated scene and on the numerical precision that shall be obtained. Changing the stiffness of the materials, setting up a scene where stronger or weaker forces in the cloth occur or changing the time step or the conjugate gradient error tolerance all influence the number of conjugate gradient iterations that have to be done, and possibly the visual results.

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 program runs for each investigated setting. Figure 2 and Figure 3 show the results of the performance measurements for the cloth and the shirt scene. The time values given for one processor are based on a sequential version of our parallel simulator that employs sequential data structures and sequential arithmetic operations (PETSc is capable

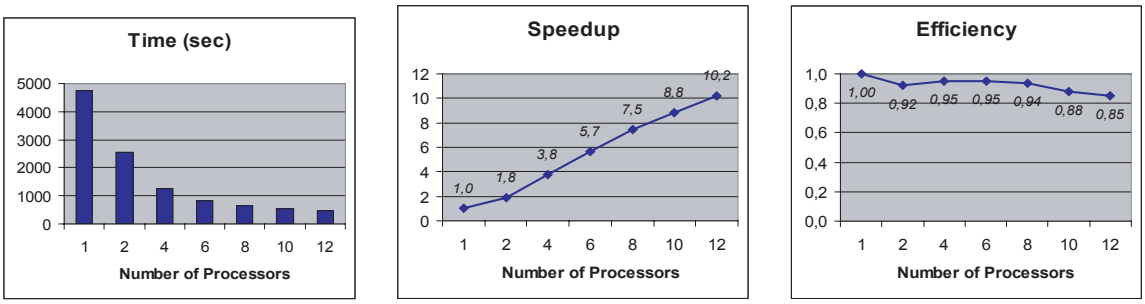

Figure 2: *Results of performance measurements of the cloth scene.*

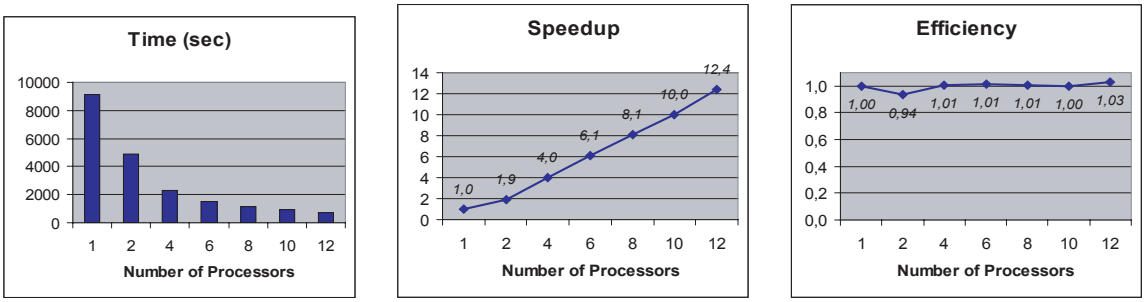

Figure 3: *Results of performance measurements of the shirt scene.*

to automatically choose sequential primitives at run-time if only one processor is available by means of its polymorphic architecture). The results show that we achieve a high level of parallel efficiency preserving a high level of scalability in both test scenarios. For the shirt scene super-linear speedups could be observed. In data parallel applications the main source of super-linear speedups are memory cache effects. With an increasing number of processors the data working-set of each individual processor becomes smaller, often resulting in an increased cache hit rate.

6. Conclusions and Future Work

We presented a parallel cloth simulation engine that is capable of substantially improving the computational performance of cloth animations. In particular, we described a parallel realization of an implicit integration scheme for cloth animations on distributed memory architectures. The performance measurements show that the employed methods scale well, indicating that parallel techniques are a promising approach to achieve high computational performance for high resolution cloth models.

In the future, we will add parallel algorithms for collision detection (using bounding volume hierarchies) and response. It seems to be an interesting question how to optimize the mesh partitioning for self-collisions of the cloth.

With the possibility to handle very large numbers of triangles due to parallelization, it will also be interesting to experimentally evaluate the influence of the discretization level and simplifications of the physical models and numerical methods on the realism in cloth simulations.

References

[AB03] ASCHER U., BOXERMAN E.: On the modified conjugate gradient method in cloth simulation. *The Visual Computer* (2003).

[BBG*01] BALAY S., BUSCHELMAN K., GROPP W. D., KAUSHIK D., KNEPLEY M., MCINNES L. C., SMITH B. F., ZHANG H.: PETSc home page. http://www.mcs.anl.gov/petsc, 2001.

[BBG*02] BALAY S., BUSCHELMAN K., GROPP W. D., KAUSHIK D., KNEPLEY M., MCINNES L. C., SMITH B. F., ZHANG H.: *PETSc Users Manual*. Tech. Rep. ANL-95/11 - Revision 2.1.5, Argonne National Laboratory, 2002.

[BGMS97] BALAY S., GROPP W. D., MCINNES L. C., SMITH B. F.: Efficient management of parallelism in object oriented numerical software libraries. In *Modern Software Tools in Scientific Computing* (1997), Arge E., Bruaset A. M.,, Langtangen H. P., (Eds.), Birkhauser Press, pp. 163–202.

[BMF03] BRIDSON R., MARINO S., FEDKIW R.: Simulation of Clothing with Folds and Wrinkles.

In *Proc. SIGGRAPH Symposium on Computer Animation* (2003).

[BW98] BARAFF D., WITKIN A.: Large Steps in Cloth Simulation. In *Computer Graphics (Proc. SIGGRAPH)* (1998), pp. 43–54.

[BWK03] BARAFF D., WITKIN A., KASS M.: Untangling Cloth. In *Computer Graphics (Proc. SIGGRAPH)* (2003).

[CK02] CHOI K.-J., KO H.-S.: Stable but Responsive Cloth. In *Computer Graphics (Proc. SIGGRAPH)* (2002), pp. 604–611.

[CK03] CHOI K.-J., KO H.-S.: Extending the Immediate Buckling Model to Triangular Meshes for Simulating Complex Clothes. In *Eurographics Short Presentations* (2003), pp. 187–192.

[CMT02] CORDIER F., MAGNENAT-THALMANN N.: Real-time Animation of Dressed Virtual Humans. *Computer Graphics Forum* (2002).

[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).

[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.

[HES03] HAUTH M., ETZMUSS O., STRASSER W.: Analysis of Numerical Methods for the Simulation of Deformable Models. *The Visual Computer* (2003).

[KCCL01] KANG Y.-M., CHOI J.-H., CHO H.-G., LEE D.-H.: An efficient animation of wrinkled cloth with approximate implicit integration. *The Visual Computer 17*, 3 (2001).

[KK96] KARYPIS G., KUMAR V.: *Parallel Multilevel k-way Partitioning Schemes for Irregular Graphs*. Tech. Rep. 036, Minneapolis, MN 55454, May 1996.

[KSFS03] KECKEISEN M., STOEV S. L., FEURER M., STRASSER W.: Interactive Cloth Simulation in Virtual Environments. In *Proc. IEEE Virtual Reality* (2003).

[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).

[MDDB01] MEYER M., DEBUNNE G., DESBRUN M., BARR A.: Interactive Animation of Cloth-like Objects in Virtual Reality. *The Journal of Visualization and Computer Animation 12*, 1 (2001), 1–12.

[Pro95] PROVOT X.: Deformation Constraints in a Mass-Spring Model to Describe Rigid Cloth Behavior. In *Graphics Interface '95* (1995), pp. 147–154.

[RRZ00] ROMERO S., ROMERO L. F., ZAPATA E. L.: Fast Cloth Simulation with Parallel Computers. In *Euro-Par* (2000), pp. 491—499.

[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.

[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).

[ZFV02] ZARA F., FAURE F., VINCENT J.-M.: Physical Cloth Animation on a PC Cluster. In *Fourth Eurographics Workshop on Parallel Graphics and Visualisation* (2002).

[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 4: *The first test scene: a piece of cloth consisting of 22500 vertices is pinned at three points and dragged down by gravity until it slides onto the floor.*

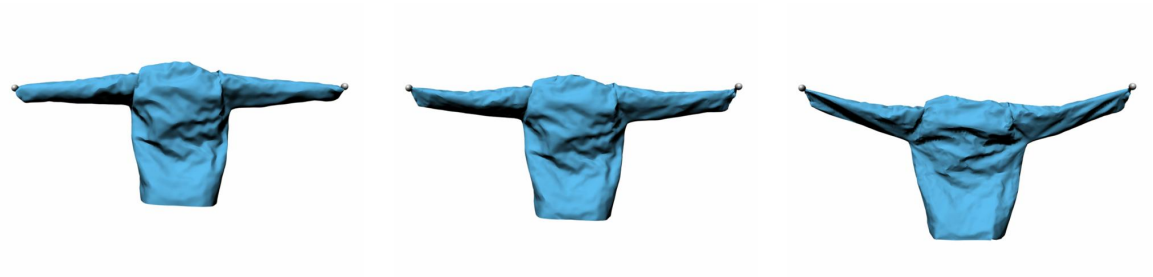

Figure 5: *The second test scene: a shirt consisting of 35024 vertices is pinned at two points and dragged down by gravity.*

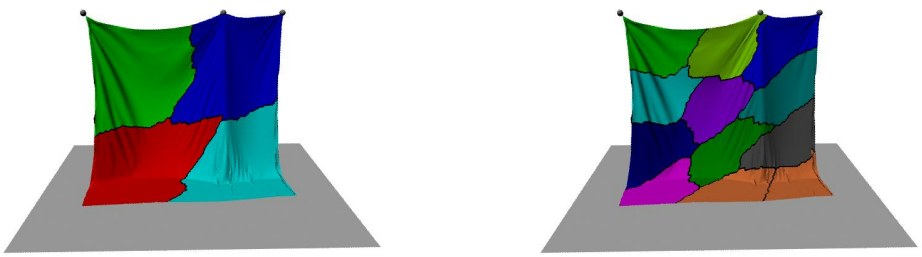

Figure 6: *The piece of cloth partitioned for 4 and 12 processors (colors are assigned randomly).*

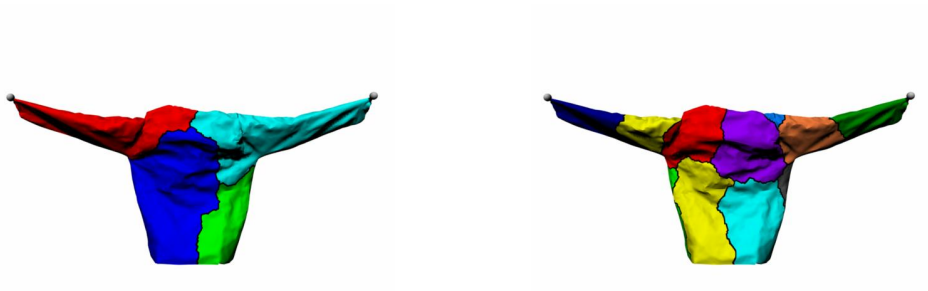

Figure 7: *The shirt partitioned for 4 and 12 processors (colors are assigned randomly).*