Mesh Partitioning for Parallel Garment Simulation

M. Hutter
Fraunhofer IGD
Fraunhoferstr. 5
64286 Darmstadt,
Germany
marco.hutter@igd.fraunhofer.de

M. Knuth
Fraunhofer IGD
Fraunhoferstr. 5
64286 Darmstadt,
Germany
martin.knuth@igd.fraunhofer.de

A. Kuijper
TU Darmstadt
Fraunhoferstr. 5
64286 Darmstadt,
Germany
arjan.kuijper@igd.fraunhofer.de

ABSTRACT

We present a method for partitioning meshes that allows a simple and efficient parallel implementation of different simulation methods. It is based on a generalization of the concept of independent sets from graph theory to sets of simulation elements. The general description makes it versatile and flexibly applicable in existing simulation systems. Every simulation method that formerly worked by sequentially processing a set of simulation elements can now be parallelized by partitioning the underlying set, without affecting the behavior of the simulated model.

Keywords

Computer Graphics, Animation, Simulation, Parallel Programming

1 INTRODUCTION

The simulation of garments has various fields of application, like computer-generated movies, computer games the virtual prototyping of garments. These applications cause a constant demand for increased visual realism, efficiency and accuracy of the underlying simulation models. Especially in engineering applications, increased efficiency is usually immediately expended for increasing the accuracy, e.g. by using meshes with a higher resolution.

With the advent of the multi-core era, parallelization started to become increasingly important in garment simulation. Like all other compute-intensive applications, garment simulation systems should be able to exploit the processing power of all available computing devices in order to scale with future hardware generations. Therefore, methods have to be developed that allow a flexible distribution of the workload among the computing devices and minimize the overhead for scheduling and synchronization.

In this paper, we give a general description for a strategy that allows us to partition the simulated elements of the mesh into multiple subsets in a way that allows an efficient simulation on multiple CPU cores or many GPU cores, without the need for explicit synchronization. The concept of independent sets from graph theory is generalized to simulation elements (for example, the primitives of the mesh) as well as sets of simulation elements.

The remainder of this paper is organized as follows: Section 2 gives an overview of the related work, focussing on the different methods for garment simulation and parallelization and the combination thereof. The concept for the mesh partitioning presented in this paper is detailed in Section 3. Section 4 shows the implementation and how the method has been evaluated and applied to different simulations, leading to the results that are summarized in Section 5. A discussion of the advantages and limitations is given in Section 6, and Section 7 gives a conclusion and an outlook for future work.

2 RELATED WORK

Many strategies for accelerating garment simulation have been developed since Weil [26] and Terzopoulus et al. [21] presented their first work about models for cloth simulation. Some of them focussed on the simulation model itself. The seminal work of Baraff and Witkin [1] aimed at increasing the stability and thus the possible time step size by using implicit integration. Other approaches model the cloth as a set of constraints in order to efficiently and robustly simulate the behavior of cloth, based on the work of Provot [16]. This usually refers to edge constraints, but has also been applied to triangle constraints in order to support the anisotropic behavior of cloth [24] or bending constraints [25]. Position-based dynamics for cloth simulation have been examined by [14]. Alternative approaches include Lagrange multipliers [7] or finite element methods [5]. A recent survey on

 ISBN 978-80-86943-71-8

cloth simulation literature can be found in [11], and more details on the subject in the context of garment simulation in [12].

General approaches for parallel cloth simulation have already been considered in [6], where a message-based method for the simulation of interacting particles is introduced. Parallelization methods for the solution of sparse linear systems with conjugate gradient methods have been developed for many engineering applications (see for example [2]), and applied to cloth simulation as well [17], [22]. The latter may rely on a decomposition of the simulated cloth into several subsets, for example, using mesh decomposition toolkits like Metis [9].

Other works partitioned the primitives of the simulation mesh into independent sets that can be treated in parallel. Such an approach for independent sets of particles has been described by Zara et al. [27]: Each processor receives a subset of the particles that the simulated object consists of, and computes the forces and state changes for this subset. In a dedicated computation step, the interactions between the particle subsets are handled. Specifically, a dependency graph is used for the computation of the final state.

For the parallel treatment of edges on many-core architectures, different approaches have been developed. Some authors employed sampling techniques and hierarchical solvers to perform an efficient simulation of cloth on the GPU, for example, Schmitt et al. [18]. Methods that are based on partitioning the set of primitives have also been developed (e.g. [28]) and are already implemented in the Bullet library [4]. Here, the set of edges is partitioned into "batches", where each batch is a set of edges, and no two edges of one batch have a common particle.

The approaches for parallelizing cloth simulation are not restricted to the simulation of the intrinsic behavior of the cloth like the time integration and computation of internal forces. Several authors developed methods for parallelizing other elements of the cloth simulation process, particularly the detection and treatment of collisions, contact and friction between the cloth and other objects. Selle et al. [19] proposed an improved, parallel collision detection method similar to that of Bridson et al. [3], where the particles of the cloth mesh are distributed among several processors. An architecture for parallelizing the whole cloth simulation pipeline, including time integration, collision detection and collision response was proposed by Tang et al. [20]. They define *streams* of data for the state of the simulated garment, consisting of positions and velocities, as well as for the collision- and collision response information. These streams are processed in parallel, solely on the GPU, in order to achieve a high speedup compared to CPU based implementations or hybrid methods that rely on a transfer of the simulation- or collision data between the CPU and the GPU memory space.

Our focus in this paper is on the parallelization of the material simulation which is usually summarized as the computation of internal forces or constraints and the time integration.

Most of the approaches for parallelizing these steps that have been presented so far make assumptions about the structure of the simulated object that make a practical application of these approaches difficult. These assumptions refer to limitations regarding the shape of the simulated cloth, and particularly the mesh structure. For example, many of the aforementioned approaches rely on the mesh consisting of regular triangles or quadrilaterals.

The partitioning strategies that have been presented for general or irregular meshes are usually specific for a certain kind of mesh primitive: They subdivide the sets of vertices, edges or faces, based on specific rules for the particular primitive type.

The approaches that try to abstract from the primitive type and apply general partitioning methods to the whole simulation mesh require an explicit synchronization at the borders of the subsets of the mesh.

Our goal is to overcome these limitations, and therefore we present a method that allows a simple parallelization of simulations that formerly consisted of a sequential processing of mesh primitives. It consists of a versatile partitioning strategy that does not require explicit synchronization between the partition elements and may be applied to arbitrary meshes with arbitrary primitives.

3 CONCEPT

The majority of the cloth simulation methods referred to in the previous section operate on a triangle- or quadrilateral mesh that represents the geometry of the cloth. Depending on the actual simulation model, different primitives of this mesh are relevant for the actual computation that is to be performed.

The following subsections will give a general description of possible representations of the cloth model, and show how this description may serve as the basis for our partitioning method. The conditions that must hold for the parallel simulation of cloth on the CPU or the GPU are explained. We will show how to partition the simulation model so that these conditions are satisfied, and how the existing simulation methods may be applied to the partitioned simulation model.

3.1 Simulation model state

We describe the simulation model in its most general from to contain a set of *simulation elements* M. These simulation elements are usually constructed from certain primitives of the mesh. For example, the vertices

 ISBN 978-80-86943-71-8

of the mesh may be converted into a set of particles, where each particle p_i contains a position $\mathbf{x}_i$, a velocity $\mathbf{v}_i$, a mass m_i, and either an acceleration $\mathbf{a}_i$, or alternatively, a force $\mathbf{f}_i$ where $\mathbf{f}_i = m_i\mathbf{a}_i$. For particle-based simulation methods like that of Zara et al. [27], these particles may already be the actual simulation elements. For other approaches, like classical, edge-based mass-spring models or the position based approach of Muller et al. [14], the simulation elements are the springs or distance constraints that are imposed by the edges of the simulation mesh or inserted between two opposite particles of two triangles that share a common edge in order to model a bending resistance. For triangle-based simulation methods like that of Volino et al. [24] or the 2D strain limiting approach described in [25], each triangle is a simulation element. When the bending resistance of the cloth is computed from the angle between the normals of adjacent triangles, as in [25], or the in-plane deformation of two adjacent triangles as in [23], one pair of triangles may also be a simulation element. Other simulation models may involve even higher-order simulation elements like tetrahedrons.

The similarity of the term "simulation elements" to the term *finite elements* is not a coincidence: In both cases, the elements can be considered as the smallest elements of a decomposition of the problem. These elements are the basic building blocks of a computational model of the simulated object. Thus, the simulation elements are associated with a representation of the simulation model state. This state is the actual basis for the computation that is performed for each simulation element. In many cases, a simulation element m_i corresponds to set of particles that it consists of: $m_i = \{p_{i_0}, ..., p_{i_n}\}$. For example, the simulation model state of an edge in a mass-spring model is given by the positions, velocities and accelerations (or forces) that are stored in the respective particles.

3.2 Sequential processing

Given a simulation model state based on a set of simulation elements M, we assume that the simulation algorithm in its most abstract and generic form can be written as in Algorithm 1.

```
Input is the set of simulation elements M
for all simulation elements m_i ∈ M do
    perform computation for m_i
```

Algorithm 1: Sequential algorithm

The computation that is performed for each simulation element may, for example, be the computation of a force for one structural spring of a mass-spring-model, or position corrections for the particles in an edge- or face-based strain limiting approach.

Our goal is now to partition the simulation elements so that the same algorithm can be run in parallel, regardless of the type of the simulation elements.

3.3 Partitioning simulation elements

The basic idea of partitioning the simulation elements into independent sets that we describe in the following sections is conceptually similar to the computation of the "edge batches" by Zeller et al. [28], and has already been implemented in [4] or [8]. These descriptions and implementations are currently restricted to edges as the simulation elements, with the intention to perform the simulation on the GPU. Extending this idea to arbitrary simulation elements is straightforward, and thus will not be described here in detail. Instead, we will focus on the extension of this approach to *sets* of simulation elements.

3.4 Independent sets

In graph theory, an *independent set* is a set of vertices where no two vertices are connected by one edge. Two simulation elements can be considered as being *adjacent* when they share any data. That is, when the computation of the new state of the simulation model for one simulation element modifies or depends on the state of the other simulation element. Particularly, in many cases it can be said that two triangles or edges are adjacent if they share a common particle. An adjacency graph for the simulation elements may thus be defined as a graph $G = (V,E)$, where the vertex set V of the graph contains the set of simulation elements, and the edge set E of the graph is defined as the set of 2-tuples of simulation elements that are adjacent.

A generalization of the concept of independent sets allows us to define two *sets* of simulation elements as being adjacent when any simulation element of one set is adjacent to any element of the other set. The set of simulation elements M may therefore be divided into several subsets, as discussed in Section 4, yielding a partition $P(M) = \{M_0, ..., M_{n-1}\}$. These subsets are tested for pairwise adjacency. Let A be the set of tuples of adjacent sets (M_i, M_j), then $G = (P(M), A)$ is an adjacency graph on which a vertex coloring can be computed in order to determine a set of independent sets of sets of simulation elements.

3.5 Vertex coloring

A *vertex coloring* is a partition of the vertex set of a graph so that each element of the partition is an independent set. More formally, a c-coloring of the graph is a set $R = \{R_0, ..., R_{c-1}\}$ where each $R_i = \{v_{i_0}, v_{i_1}, ...\}$ is an independent set, each corresponding to one color.

The smallest number of colors needed to color a graph is called the *chromatic number* of the graph, and the corresponding coloring is called an *optimal coloring*.

 ISBN 978-80-86943-71-8

Computing such an optimal coloring is an NP-complete problem, and thus not feasible for larger graphs, even as a preprocessing step.

Instead of computing the chromatic number of the graph, one can compute a coloring of the graph with an optimal *coloring number*, as defined by [15]. An ordering of the vertices that results in an optimal coloring number c_{opt} can be computed using the *smallest-last coloring* algorithm introduced by [13]. This coloring number is bounded by the maximum degree $d_{\max}$ of any vertex in the graph. A simple greedy strategy thus allows us to compute a coloring with a number of colors that is bound by $d_{\max}$ in linear time.

The overall process of computing independent sets of sets of simulation elements based on a coloring of the adjacency graph is summarized in Figure 1.

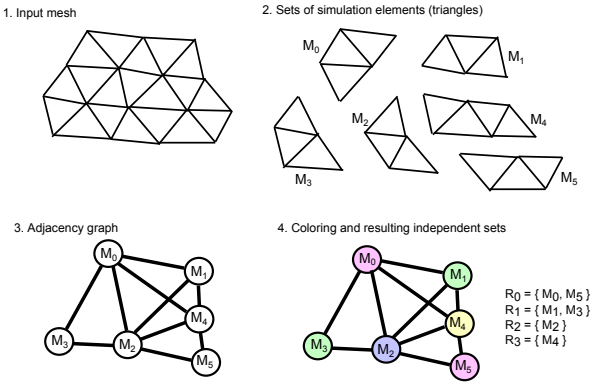

Figure 1: Process of the computation of independent sets of sets of simulation elements.

3.6 Dispatching to CPUs or GPUs

Within the taxonomy of parallel techniques, we focus on the architectures using a shared address space. These may refer to multi-core architectures, namely CPUs with two or more cores, or to many-core architectures like GPUs with hundreds or thousands of cores.

The conditions that must hold for the parallel processing of simulation elements are the same for the CPU- and the GPU-case: In both cases, the simulation elements that are processed concurrently may not be adjacent. However, multi-core CPUs are capable of task-parallel processing, whereas GPUs are usually wide SIMD implementations that are tailored for data-parallelism. That means that multiple CPU cores can concurrently process the elements of an independent set of sets of simulation elements, whereas a GPU is better suited for concurrently processing the elements of an independent set of simulation elements. This difference is sketched in Figure 2.

3.7 Parallel cloth simulation

In order apply the sequential Algorithm 1 in parallel to multiple sets of simulation elements, we propose Algorithm 2: In a precomputation step, the simulation

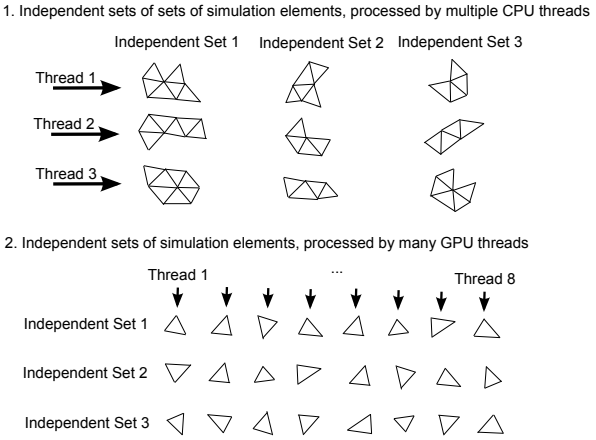

Figure 2: Sketch of the different schemes for dispatching sets of simulation elements to the CPU or simulation elements to the GPU

elements are created from the input mesh. The set M of simulation elements is partitioned into $P(M) = \{M_0, ..., M_{n-1}\}$ as described in Section 4. The independent sets $R = \{R_0, ..., R_{c_{\text{opt}}-1}\}$ are computed based on the adjacency graph. Each $R_i = \{M_{i_0}, M_{i_1}, ...\}$ is a set of sets of simulation elements, and these sets are not adjacent to each other. Thus, the elements of each R_i may be processed in parallel, each with the sequential Algorithm 1.

Input is the set of independent sets of sets of simulation elements R
for all independent set $R_i \in R$ **do**
  for all subsets of simulation elements $M_{i_j} \in R_i$ in parallel **do**
    apply Algorithm 1 to M_{i_j}

Algorithm 2: Parallel algorithm

It is important to note that, according to the construction of the set R, there is no data shared between the elements of sets M_{i_a} and M_{i_b} that are processed in parallel. They do not have any common simulation elements. And particularly, according to the definition of being "independent", the *state* of the simulated model that is represented with each simulation element from M_{i_a} is completely unrelated to the state that is represented with a simulation element from M_{i_b}.

4 IMPLEMENTATION

The concept of computing independent sets of simulation elements has been implemented for different cloth simulation methods. An example implementation of the basic concept using edge constraints has been shown in [8]. Within the *Future Fashion Design* project the concept has been generalized and integrated into a commercial garment simulation system, where a large variety of simulation schemes are combined. This involves edge

springs, edge strain constraints, triangular springs, triangle strain constraints, bending edges and pairs of triangles that are used to compute bending forces or bending constraints, and various combinations thereof.

For the implementation of a parallel cloth simulation on a multi-core CPU based on independent sets of sets of simulation elements, we create the simulation elements from the input mesh. In our current implementation, this may either be

- The edges of the underlying mesh
- Bending edges between opposite particles of pairs of triangles with a common edge

or

- The triangles of the mesh
- Pairs of triangles with a common edge

each being used as springs or constraints.

Once these simulation elements have been created, different strategies may be employed to split these elements into multiple subsets. Many of these strategies are conceptually similar to that for the construction of a bounding volume hierarchy (see for example the work of Klosowski [10]): The sets of simulation elements may be constructed bottom-up, by combining adjacent elements into groups, or top-down, by splitting the initial set into smaller subsets. For the latter approach, there are several choices for selecting the split axes and the number of subsets that is created in each step. The goal usually is to obtain a balanced hierarchy - which corresponds to subsets of approximately equal sizes in our case.

In our test scenes, the cloth is initially given in a rest state in the xy-plane. A very simple splitting strategy for this case is the recursive subdivision: A bounding box is computed for the set of simulation elements M. Then the set is split at the bounding box center along the x- or/and y-axis, yielding 2 or 4 new sets, respectively. This subdivision step is repeated on the resulting sets, until a stopping criterion is met, yielding a partition $P(M) = \{M_0, ..., M_{n-1}\}$.

There are several possible stopping criteria for the recursive subdivision: The recursion may either stop when the number of simulation elements in each set falls below a certain threshold, or when a certain number of partition elements is reached. This is a tuning parameter that will be detailed in Section 5.

Figure 3 shows the result of the computation of the independent sets of sets of simulation elements, for one pattern of a garment, using triangles as the simulation elements. It shows the full pattern, as well as the respective independent sets $R_0, ..., R_3$, each with a different color.

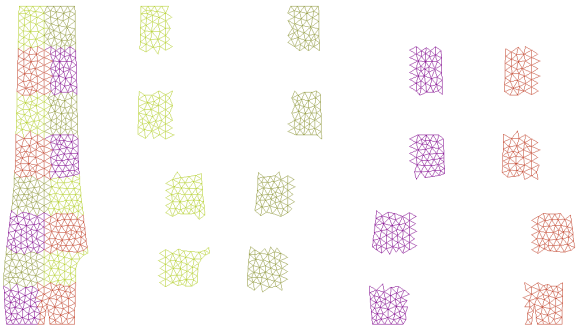

Figure 3: A coloring on the graph whose nodes correspond to sets of simulation elements of one pattern yields a set of independent sets of sets of simulation elements.

Each R_i contains sets of simulation elements, namely $R_i = \{M_{i_0}, M_{i_1}, ...\}$. According to the construction of these sets, the simulation model state of any simulation element in M_{i_a} does not affect or depend on the simulation model state of any M_{i_b} for $a \neq b$, so the elements of R_i may be be processed in parallel using Algorithm 2.

5 RESULTS

We integrated our mesh partitioning method into an existing garment simulation system in order to verify its practical applicability. The system uses many different kinds of simulation elements, which are combined in order to perform an isotropic or an anisotropic material simulation.

5.1 Simulation types

The isotropic simulation is *edge-based*. The simulation elements are different kinds of edges:

- Edge Forces (**EF**) are springs that correspond to the edges of the triangle mesh
- Edge Constraints (**EC**) are distance constraints for the edges of the triangle mesh
- Edge Bending Constraints (**EBC**) are distance constraints that are inserted between opposite particles of two triangles that share one edge, to model bending resistance (these constraints are of the same type as the Edge Constraints)

The anisotropic simulation is *triangle-based*. The simulation elements are triangles and pairs of triangles:

- Triangle Forces (**TF**) are spring forces that are computed based on the deformation of the triangles of the mesh
- Triangle Bending Forces (**TBF**) are spring forces that are computed for pairs of triangles that share one edge, which are used in the computation of the bending resistance

 ISBN 978-80-86943-71-8

- Triangle Constraints (**TC**) are constraints enforcing the triangles of the mesh to be in their rest shape

5.2 Benchmark scenes

In order to evaluate the speedup that can be achieved with our method, we used two different test scenes: The first one, shown in Figure 4, is an artificial scene of a large circular piece of cloth that is draped on a torus. The cloth consists of 45k particles, 95k triangles and 155k edges.

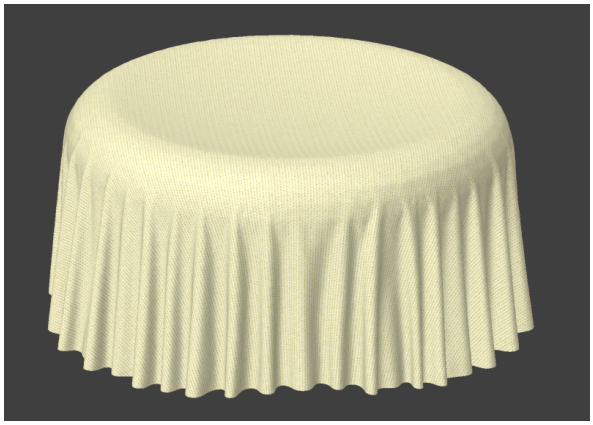

Figure 4: Benchmark scene 1: A cloth with 45k particles, 95k triangles and 155k edges, draped over a torus

The second benchmark is a realistic shirt worn by an artificial character, as shown in Figure 5. The shirt consists of 10 patterns, with 60k particles, 116k triangles and 176k edges in total.

Figure 5: Benchmark scene 2: A shirt consisting of 10 patterns, with 60k particles, 116k triangles and 176k edges in total

The current focus of our research is on *interactive* virtual prototyping applications. Therefore, our benchmarks consist of meshes with a resolution that still allows an interactive simulation. We ran several artificial benchmarks with different mesh resolutions, and the *speedup* that was achieved was largely independent of the actual mesh resolution.

5.3 Benchmark setup

The simulation system is implemented in Java. The tests have been run on an Intel Xeon 2.4GHz, with 6 cores (12 virtual), 24 GB RAM, Windows 7 / 64bit, using the Java JRE 7u11. Each benchmark run consisted of starting the simulation with a time step of 1ms, and running for a predefined number of steps until the cloth approached a rest state.

5.4 Benchmark results

For both test cases, the default number of subsets that was created during the recursive subdivision was 128 (also see the section about the *Benchmark Parameters* below). The first test scene was run for 600 steps, and the second scene was run for 200 steps. The following tables contain the average duration for the computations for each simulation element type in the last 25 time steps. The durations are given in milliseconds.

Note that these timings do *not* include miscellaneous computation steps like the collision detection or rendering updates. They only refer to the core of the simulation which is parallelized using our partitioning method. The computation that is performed for each subset of simulation elements is exactly the same as for the serial version. Thus, the overall behavior and drape of the garment is not noticeably affected by the parallelization.

N	EF	EC	EBC	Total
1	9.06	104.16	50.76	163.99
2	3.79	53.78	26.56	84.14
4	2.51	29.95	15.99	48.47
6	2.63	24.24	13.80	40.68
8	2.52	23.20	13.85	39.58
10	2.48	22.87	13.49	38.86
12	2.22	22.92	13.99	39.14

Table 1: Computation time for the first benchmark scene, in milliseconds, using the edge-based simulation. *N* = Number of threads, *EF* = Edge Forces, *EC* = Edge Constraints, *EBC* = Edge Bending Constraints, as detailed in 5.1

The diagrams 6 and 7 show the total speedup of the computation for all simulation elements in the edge-based and the triangle-based simulation method, respectively, depending on the number of threads.

It can be seen that the speedup increases nearly linearly with the number of threads, up to the number of

 ISBN 978-80-86943-71-8

N	TF	TBF	TC	Total
1	9.56	47.85	270.23	327.65
2	5.93	22.48	132.38	160.81
4	2.88	11.49	99.75	114.13
6	2.54	9.74	56.52	68.81
8	2.27	9.16	47.48	58.92
10	2.24	8.95	48.54	59.74
12	2.17	7.94	47.97	58.09

Table 2: Computation time for the first benchmark scene, in milliseconds, using the triangle-based simulation. *N* = Number of threads, *TF* = Triangle Forces, *TBF* = Triangle Bending Forces, *TC* = Triangle Constraints, as detailed in 5.1

N	EF	EC	EBC	Total
1	10.45	175.08	98.50	284.03
2	5.37	103.44	88.11	196.93
4	3.52	50.82	32.70	87.04
6	2.97	42.14	27.83	72.94
8	2.94	40.44	27.80	71.19
10	2.16	40.60	29.46	72.23
12	2.11	39.37	26.75	68.24

Table 3: Computation time for the second benchmark scene, in milliseconds, using the edge-based simulation. *N* = Number of threads, *EF* = Edge Forces, *EC* = Edge Constraints, *EBC* = Edge Bending Constraints, as detailed in 5.1

N	TF	TBF	TC	Total
1	10.09	51.79	565.72	627.61
2	5.96	27.53	220.35	253.85
4	3.55	15.66	120.66	139.88
6	3.04	11.59	101.32	115.96
8	2.54	13.40	97.09	113.04
10	2.71	11.90	91.16	105.78
12	2.75	11.18	88.55	102.48

Table 4: Computation time for the second benchmark scene, in milliseconds, using the triangle-based simulation. *N* = Number of threads, *TF* = Triangle Forces, *TBF* = Triangle Bending Forces, *TC* = Triangle Constraints, as detailed in 5.1

available physical cores. Thus, our method allows us to achieve a significant speedup for all kinds of simulation elements, and scales well with an increasing number of physical cores.

5.5 Benchmark parameters

As mentioned in Section 4, there are several degrees of freedom for the computation of the initial subdivision of the set of simulation elements. The recursive subdivision that we use in our implementation allows two different stopping criteria: Either depending on the size of the resulting subsets, or depending on the number of subsets. For our analysis, we focus on the latter, because for our test cases, the number of simulation elements of the individual test scenes as well as the number of CPU cores of the target machine are known beforehand, whereas in other application scenarios, a criterion based on the size of the subsets may be more appropriate.

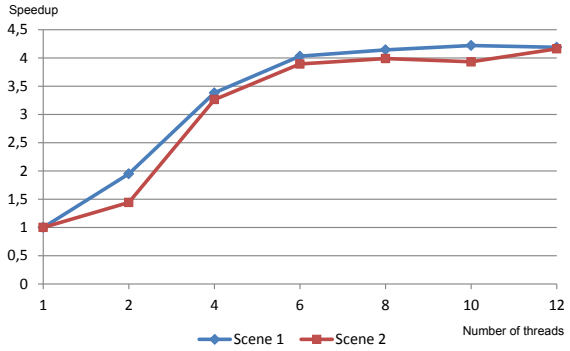

Figure 6: Speedup for the edge-based simulation method depending on the number of threads on a 6-core machine

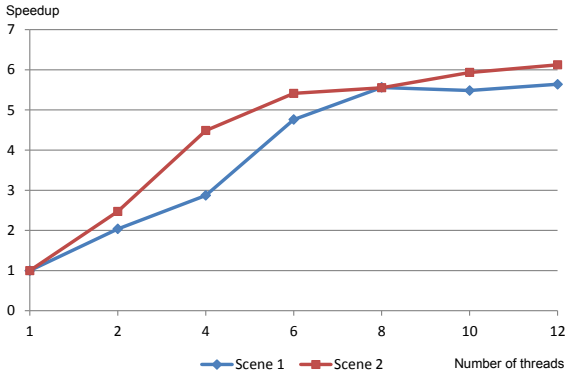

Figure 7: Speedup for the triangle-based simulation method depending on the number of threads on a 6-core machine

The number of subsets that are created with the subdivision process affects the dispatching and processing in two ways: A larger number of smaller subsets (which results in larger independent sets) allows a more fine-grained scheduling and makes it possible to distribute the workload more evenly among the available cores. On the other hand, when the number of subsets is too large compared to the number of elements they contain, the overhead for dispatching them will increase and might degrade the overall performance. However, we found that for large meshes, the overall performance did not change significantly when the number of subsets was between 8 and 64 times the number of available cores, as illustrated in Figure 8.

For example, in our first benchmark scene, the set of 95k triangles is subdivided into 128 subsets. The graph coloring yields 5 independent sets, each containing approximately 26 subsets, each on average containing 744 elements.

 ISBN 978-80-86943-71-8

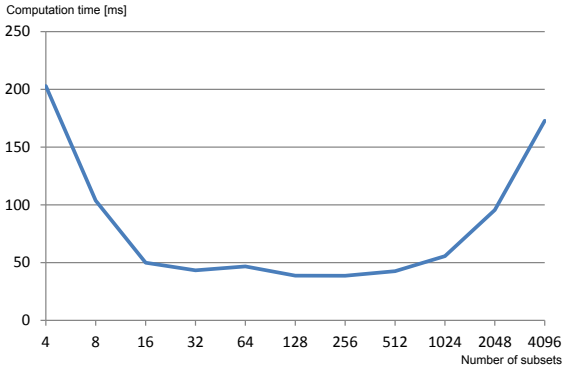

Figure 8: Dependency of the running time on the number of subsets that are created during the recursive subdivision. The test was performed with scene 1, using the edge-based simulation, and running with 8 threads. The time is given in milliseconds.

6 DISCUSSION

The main advantage of the partitioning scheme presented here is that it may be applied to all kinds of simulation elements, and may easily be integrated into existing simulation applications. A given implementation of an algorithm that sequentially processes simulation elements, as sketched in Algorithm 1, may easily be converted into a parallel version according to the scheme sketched in Algorithm 2, after partitioning the simulation elements as described in this paper.

We consider it as important to point out that the proposed method does not require an *explicit* synchronization among the elements that are processed in parallel, since they are *independent* by construction. Other methods divide the simulated mesh into several parts and treat these parts in parallel, but still require a synchronization at the border of these parts. When the parallelism should be increased, by increasing the number of parts that are created, then the synchronization overhead grows accordingly. This is not the case for our method.

It is not obvious how a similar partitioning scheme may be applied to an algorithm that involves data structures of a more global nature — for example, matrices that contain constraint information that has been derived from the structure of the whole mesh, and which is used as the input for SLE solvers. Another challenge may be topological changes of the underlying mesh, which may make it necessary to update the internal data structures. Since one of our goals is general applicability of the method, these aspects will also be addressed in our future work.

7 CONCLUSION AND FUTURE WORK

The description of the implementation in this paper focussed on parallel simulation for multi-core CPUs. Similar concepts have been developed for a many-core GPU-based simulation. The increasing number of different computing devices, ranging from CPUs and GPUs to specialized DSPs, will make it necessary to tailor applications for heterogeneous computing. The methods presented here may be one step towards this goal: It is possible to create independent sets of sets of simulation elements for the CPU, as well as independent sets of simulation elements for the GPU. Once there is a common execution- and memory model for both device types, it could be possible to split the workload that is imposed by a large simulation mesh into smaller workloads whose structure is appropriate each of the available computing devices, respectively. That means that one part of the simulation mesh may be treated efficiently by several CPU cores, whereas another part is treated by multiple GPUs.

Another aspect that is to be examined is the combination of the presented approaches with changes in the topology of the underlying mesh. This may refer to small and local changes that may happen when sewing or otherwise modifying patterns, as well as to large and global changes that may happen in a simulation that adaptively changes the mesh resolution. In this case, simulation elements are removed or created on the fly, and the sizes and structure of formerly independent sets may change so that a recomputation or adaption of these sets may be necessary. For this purpose, we are examining data structures that allow a description of the structure of independent sets in form of a hierarchy that may be updated efficiently in response to changes in the topology.

Acknowledgment : The research leading to these results has received funding from the European Commission, Seventh Framework Programme (FP7/2007-2013) under grant agreement 285026.

8 REFERENCES

[1] David Baraff and Andrew Witkin. Large steps in cloth simulation. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '98, pages 43–54, New York, NY, USA, 1998. ACM.

[2] Alécio Pedro Delazari Binotto, Christian Daniel, Daniel Weber, Arjan Kuijper, André Stork, Carlos Eduardo Pereira, and Dieter W. Fellner. Iterative sle solvers over a cpu-gpu platform. In *HPCC*, pages 305–313. IEEE, 2010.

[3] Robert Bridson, Ronald Fedkiw, and John Anderson. Robust treatment of collisions, contact and friction for cloth animation. *ACM Trans. Graph.*, 21(3):594–603, July 2002.

[4] Erwin Coumans. Game physics simulation, 1999.

[5] Olaf Etzmuss, Joachim Gross, and Wolfgang Strasser. Deriving a particle system from contin-

uum mechanics for the animation of deformable objects. *IEEE Transactions on Visualization and Computer Graphics*, 9(4):538–550, October 2003.

[6] D. House and D. Breen. Particles: A naturally parallel approach to modeling. In *Proceedings of 3rd Symposium on the Frontiers of Massively Parallel Computation*, pages 150–153, 1990.

[7] Donald House, Richard W. Devaul, and David E. Breen. Towards simulating cloth dynamics using interacting particles. *International Journal of Clothing Science and Technology*, 8:75–94, 1996.

[8] Marco Hutter. JOCL cloth simulation demo, 2011.

[9] George Karypis and Vipin Kumar. Parallel multilevel k-way partitioning scheme for irregular graphs. In *Proceedings of the 1996 ACM/IEEE conference on Supercomputing (CDROM)*, Supercomputing '96, Washington, DC, USA, 1996. IEEE Computer Society.

[10] James T. Klosowski, Martin Held, Joseph S.B. Mitchell, Henry Sowizral, and Karel Zikan. Efficient collision detection using bounding volume hierarchies of k-dops. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):21–36, 1998.

[11] James Long, Katherine Burns, and Jingzhou (James) Yang. Cloth modeling and simulation: A literature survey. In Vincent G. Duffy, editor, *HCI (17)*, volume 6777 of *Lecture Notes in Computer Science*. Springer, 2011.

[12] Nadia Magnenat-Thalmann. *Modeling and Simulating Bodies and Garments*. Springer Publishing Company, Incorporated, 1st edition, 2010.

[13] David W. Matula and Leland L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *J. ACM*, 30(3):417–427, July 1983.

[14] Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. Position based dynamics. In César Mendoza and Isabel Navazo, editors, *VRIPHYS*, pages 71–80. Eurographics Association, 2006.

[15] A. Hajnal P. Erdos. On chromatic number of graphs and set-systems, 1966.

[16] Xavier Provot. Deformation constraints in a mass-spring model to describe rigid cloth behavior. In *IN GRAPHICS INTERFACE*, pages 147–154, 1995.

[17] L.F. Romero, L. F. Romero, E.L. Zapata, Sergio Romero, Luis F. Romero, and Emilio L. Zapata. Fast cloth simulation with parallel computers. In *In Euro-Par 2000*. Springer-Verlag, 2000.

[18] Nikolas Schmitt, Martin Knuth, Jan Bender, and Arjan Kuijper. Multilevel cloth simulation using gpu surface sampling. In *Virtual Reality Interactions and Physical Simulations (VRIPhys)*, Lille, France, November 2013. Eurographics Association.

[19] Andrew Selle, Jonathan Su, Geoffrey Irving, and Ronald Fedkiw. Robust high-resolution cloth using parallelism, history-based collisions, and accurate friction. *IEEE Trans. Vis. Comput. Graph.*, 15(2):339–350, 2009.

[20] Min Tang, Ruofeng Tong, Rahul Narain, Chang Meng, and Dinesh Manocha. A gpu-based streaming algorithm for high-resolution cloth simulation. *Computer Graphics Forum*, 32(7):21–30, 2013.

[21] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. Elastically deformable models. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '87, pages 205–214, New York, NY, USA, 1987. ACM.

[22] Bernhard Thomaszewski, Simon Pabst, and Wolfgang Blochinger. Exploiting parallelism in physically-based simulations on multi-core processor architectures. In *EGPGV*, pages 69–76, 2007.

[23] Pascal Volino and Nadia Magnenat-Thalmann. Simple linear bending stiffness in particle systems. In *Symposium on Computer Animation*, pages 101–105, 2006.

[24] Pascal Volino, Nadia Magnenat-Thalmann, and François Faure. A simple approach to nonlinear tensile stiffness for accurate cloth simulation. *ACM Trans. Graph.*, 28(4), 2009.

[25] Huamin Wang, James F. O'Brien, and Ravi Ramamoorthi. Multi-resolution isotropic strain limiting. *ACM Transactions on Graphics*, 29(6):156:1–10, December 2010. Proceedings of ACM SIGGRAPH Asia 2010, Seoul, South Korea.

[26] Jerry Weil. The synthesis of cloth objects. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '86, pages 49–54, New York, NY, USA, 1986. ACM.

[27] Florence Zara, François Faure, and Jean-Marc Vincent. Parallel simulation of large dynamic system on a PCs cluster: Application to cloth simulation. *Int. J. Comput. Appl.*, 26(3):173–180, March 2004.

[28] Cyril Zeller. Cloth simulation on the gpu. In *ACM SIGGRAPH 2005 Sketches*, SIGGRAPH '05, New York, NY, USA, 2005. ACM.

 ISBN 978-80-86943-71-8