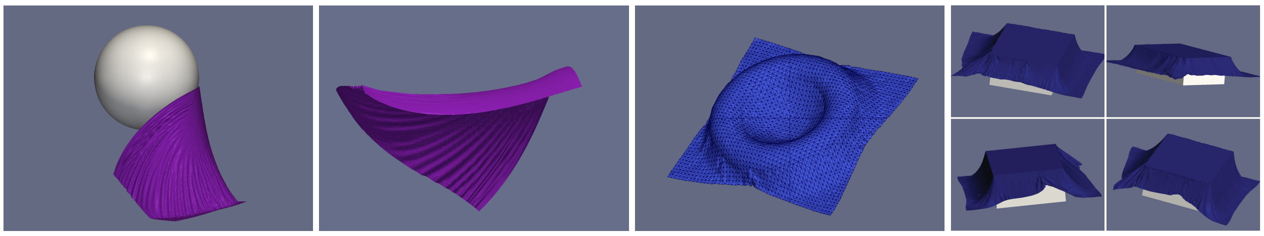
Cloth simulation using AABB hierarchies and GPU parallelism

Frizzi San Roman Salazar Bruno Brandoli Machado Alexander Ocsa Maria Cristina F. de Oliveira
Instituto de Ciências Matemáticas e de Computacão - ICMC
Universidade de São Paulo - USP



Abstract

Providing realistic, high-resolution and high-fidelity representation of motions ia essential in the cloth simulation problem. In order to make high resolution simulations tractable, several algorithms have been developed that manage cloth-object interactions efficiently through specialized data structures such as AABB trees. However, implementation restrictions on single CPU architectures impose certain limits on quality and performance in high-demanding simulations, motivating the study of new implementation techniques. In this paper we address several critical issues in high resolution cloth simulation, enabling us to represent and simulate intricate folds and wrinkles. We employ AABB hierarchies to optimize detection and response in cloth-object collisions. By employing a multi-processor approach on multi-threaded CPU and an emerging multi-core GPU-CUDA architecture, we quantitatively evaluate the workload and computational effort of the cloth simulation application. In addition to this quantitative performance evaluation on multi-processor architectures we illustrate the potential of our approach by presenting a variety of high-quality and high-resolution simulations of cloth behavior under different cloth-object interactions.

Keywords: Cloth simulation, cloth modeling, physically based modeling.

Author's Contact:

{frizzi,brandoli, aocsa, cristina}@icmc.usp.br

1 Introduction

Deformable model animation aims at producing realistic motions. Potential application problems include the simulation of muscle-body motion, flow of viscous liquids such as oil spill, and cloth simulation. Modeling cloth behavior has applications in cinematography [Selle et al. 2009], in the games industry and in clothing manufacture [Cordier et al. 2003]. Researchers from both the textile industry and computer graphics have joined efforts on the search for solutions to achieve high-fidelity cloth simulation [Hu et al. 2009].

The task is challenging because cloth behavior is very complex and highly non-linear. Faithful simulations require representing cloth's peculiar behavior regarding the variation of folds and wrinkles. Several approaches exist to handle this issue and a most popular one is based on physical principles [Hu et al. 2009]. It models cloth as a mass-spring system with deformable surfaces that are discretized into a polygonal mesh of nodes (particles, or mass-points). Nodes are connected by springs that define the elasticity, deformation and folds of the cloth [Zhibin and Zhanli 2006; Zhou et al. 2005]. The simulation solution involves defining motion equations over this deformable model, and performing several iterations to solve them. Therefore, a high computational time is typically required to ensure convergence and stability.

A plausible solution to solve the motion equations is to employ implicit integration methods [Zhou et al. 2005]. Parallelizing numerical computation in these methods is an attractive approach to substantially accelerate computation time. However, applying parallel techniques to cloth animation is challenging, because the solution has very fine granularity, with some steps in the simulation depending on results from previous steps.

Another aspect to consider in cloth animation is how to detect and treat collisions, which occur in two distinct ways: cloth models collide with objects in the scene, and self-collisions occur when cloth interacts with its own surface. Detection must occur in real time and the response must be displayed instantly. This detection has high computational cost, as a simple piece of cloth may be modeled by thousands of particles, and the collision detection algorithm must control a large number of geometric entities such as nodes, faces and edges. Many publications in the literature have considered how to accelerate this process [Mezger et al. 2002; Akenine-Moller et al. 2002; Maciel et al. 2007; Lv et al. 2007].

In this paper we address several cloth simulation issues. First, we reduce the number of iterations required for collision treatment through AABB hierarchies. As cloth is modeled as triangular meshes consisting of particles and springs, this hierarchical data structure was employed in order to reduce the number of intersection tests at primitives' levels. Second, we quantitatively analyze the computational workload of a cloth simulation application that employs a multi-processor approach on a multi-threaded CPU and on an emerging multi-core GPU-CUDA architecture. The multi-processor GPU-CUDA architecture enables high-performance since recent GPU models offer extremely high floating-point arithmetic throughput. Our efficient cloth simulation implementation relies on high memory bandwidth utilization and improved floating-point performance in the integration step. As observed in the experimental section (Section 6) frame rate generation is improved by up to 2.5. Finally, we report cloth simulation experiments from both a quantitative and qualitative perspectives producing high quality simulations for different mesh sizes and diverse scene settings.

The paper is organized as follows. Section 2 summarizes previous related work. Section 3 reviews important concepts on geometric and physical properties employed for cloth modeling. Section 4 describes different approaches to detect and treat collisions. Section 6 reports experimental results. Finally, conclusions are presented in Section 7.

2 Previous work

Cloth animation methods relate to the representation of objects and flexible surfaces. Simulations of cloth behavior may be achieved by employing geometrical and physical methods. Physical simulations require numerical methods to solve differential equations, and implicit integration methods have been proposed to compute particle positions in the simulation [Baraff and Witkin 1998]. Such methods are efficient in generating the simulation, albeit requiring a large number of iterations. Recent contributions employ semi-implicit integration methods as an alternative [Baraff et al. 2003;

Bridson et al. 2005; Zhou et al. 2005].

Moreover, there are techniques that reduce the number of algorithm iterations required for handling collisions. Van Den Berge employed bounding boxes for subdividing the three-dimensional space embedding an object [Van Den Bergen 1997]. Bounding boxes support optimization procedures such as searching, prior to employing algorithms to detect and resolve collisions. Selle et al. performed space segmentation using *AABB* trees, where parts of the partitioned object are aligned to the coordinate system [Selle et al. 2009]. Volino and Thalmann proposed an algorithm for hierarchical collision and self-collision detection and techniques to measure their orientation considering geometrical properties [Volino and Thalmann 2000]. Other authors employ data structures such as *Quadtrees* and *Octrees* for the same purpose [Gottschalk et al. 1996; Wieland et al. 2001].

3 Cloth Modeling

Computer animation was initially performed on models of rigid objects, and only later implementations were extended to handle articulated objects. Nowadays a variety of modeling strategies exist for both rigid and deformable objects. Animating deformable objects requires geometric models that accommodate the change of shape over time. In computer graphics this may be achieved with numerical methods (finite element or boundary), physical models of curves, surfaces or deformable solids.

There is a growing interest on physical models capable of improving realism in depicting deformable objects. This section describes a strategy that models cloth as polygonal meshes where mesh vertices and edges are associated with a physical mass-spring model. Numerical methods are then employed to simulate the cloth object with a good degree of realism.

3.1 Polygonal meshes

A common solution for modeling cloth is to represent it as a mesh described by a two-dimensional array of nodes M. Each matrix element represents the coordinates x,y and z of a point on the cloth surface. The physical simulation of a flexible mesh considers the concept of elastic springs joining these nodes.

Such a model assumes forces being applied to mass-points nodes of the mesh, generating new node positions to ensure its balance whenever required. Many contributions in cloth animation are based on this approach, as such meshes are easily manipulated and very effective for describing deformations [Zhou et al. 2005; Zhibin and Zhanli 2006; Selle et al. 2009].

The cloth model must be repeatedly evaluated at successive steps during an animation. Therefore, modeling meshes with too much detail becomes impractical due to the high computational cost imposed. Typically, mesh detail is traded-off for simulation performance, thus penalizing the representation of features typical of real cloth.

Subdivision meshes may be employed to obtain smoother cloth behavior at a low-performance penalty. Selle et al. apply subdivision rules to generate new edges (called bending springs). Given a simple polygonal mesh, they can improve mesh detail [Selle et al. 2009]. Figure 1 exemplifies the outcome of a subdivision and Figure 2 illustrates its application to a cloth model.

3.2 Physical Properties

Realistic animations are defined by dynamic properties that model the behavior of cloth in terms of how the particles of its describing mesh interact with external and internal forces. This approach has been employed in several contributions from the literature [Zhou et al. 2005; Zhibin and Zhanli 2006; Selle et al. 2009; Hu et al. 2009].

In general, computer animations consider four kinds of forces: gravity, elasticity, curvature and restrictions. The resulting force F_r at each grid point is defined by:

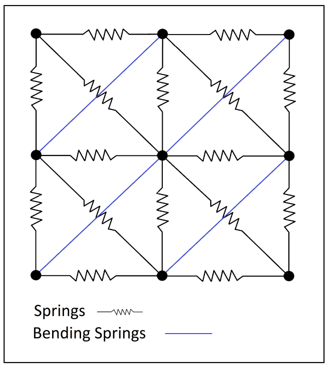
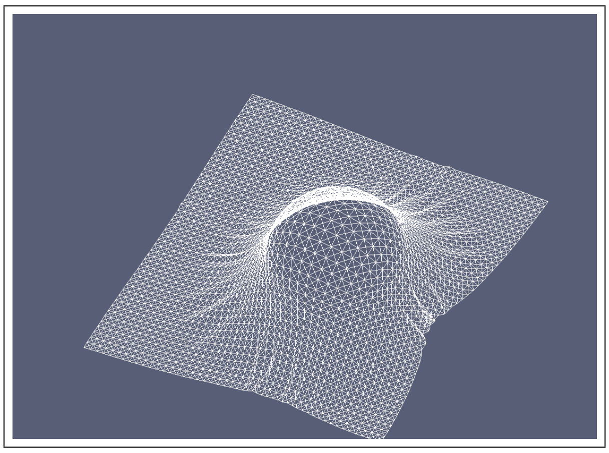
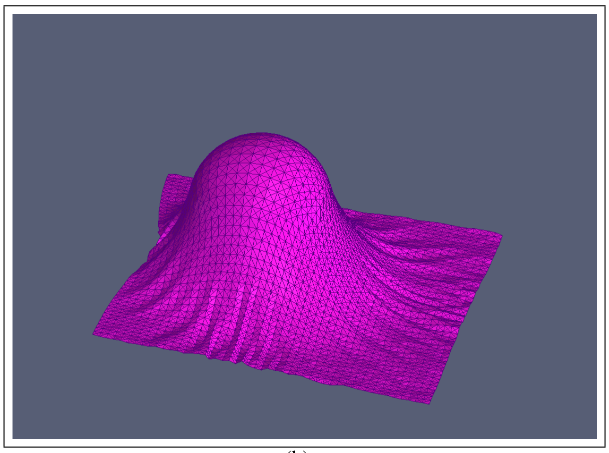


Figure 1: *Mesh subdivision strategy for increased softness and realism in cloth modeling.*





(b)

Figure 2: *Example of mesh subdivision for cloth simulation. Figure 2(a) shows the polygonal mesh and Figure 2(b) shows a rendering of the same object.*

$$F_r = F_{gravity} + F_{elasticity} + F_{curvature} + F_{restrictions} \quad (1)$$

However, the model is not necessarily restricted to these forces. For example, to consider wind strength – or any other force of interest – it is sufficient to include an additional component F_{wind} in Equation 1.

Gravity force: This may be computed for each particle using:

$$F_{gravity} = m.g \quad (2)$$

where $F_{gravity}$ is the resulting force acting upon the body, m is the particle's mass and g denotes the acceleration of gravity at Earth's surface (approximately $10m/s^2$).

Elasticity force: Each particle P_{ij} in matrix M_{ij} connects to its neighbors by elastic springs, as shown in Figure 3. The force applied by the elastic spring on a point of mass m obeys Hook's law:

$$F_{spring} = -k_m.(P - P_r) \tag{3}$$

where k_m is the elastic spring constant (modified to consider only displacements and not deformations), P denotes the particle's coordinates and P_r the particle's coordinates when at rest. So, elasticity strength may be defined as the sum of the spring forces acting on particle P:

$$F_{elasticity} = \sum F_{spring} \tag{4}$$

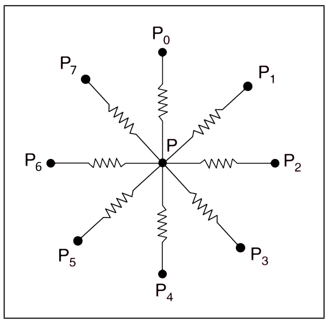


Figure 3: *Elasticity force at a point P_{ij} is modeled as the sum of elastic spring forces from its neighbors.*

The strength of bending or folding

This force determines the extent to which a surface may be bent and folded, and is computed to generate the motion of a particle as a function of the forces acting upon it from its neighbors. This effect is simulated by placing angular springs among the neighbors of particle P.

For example, Figure 3 shows the springs formed by P_0, P and P_4; P_1, P and P_5; and so on. While at rest, the three points define a line, and the angle defined by them is 180°. These springs are considered to have a customizable elasticity coefficient, called k_c.

It is also necessary to consider the **opposite forces** acting on the model, such as friction. Friction is the force resisting the relative motion of solid surfaces and its direction is defined as the opposite direction of the speed of the object's movement. Its magnitude is proportional to the magnitude of the normal force generated by contact between rough surfaces. The force due to friction is computed by:

$$F_{friction} = \mu.N \tag{5}$$

where μ is the friction coefficient and N is the normal.

3.3 Integration Methods

Simulation models employ numerical integration techniques to compute both the physical forces applied to the particles of the cloth model and the particle's displacements considering a determined speed. Common approaches in animation are explicit integration [Provot 1995; Zhibin and Zhanli 2006; Bayraktar et al. 2007], implicit integration [Baraff and Witkin 1998; House and Breen 2000; Kang et al. 2000; Baraff et al. 2003] and semi-implicit integration methods [Müller et al. 2004; S. Alzamora et al. 2008; Selle et al. 2009].

Each mesh particle is associated with a pair of parametric coordinates (u,v) on the cloth surface. They are useful to evaluate the tension and transverse deformation exerted on the cloth at each instant, and may be employed for texture mapping the cloth model. Each particle has a variable position in the three-dimensional space. Implicit Backward Euler integration is employed to integrate forces, compute the updated particle positions and dynamically simulate the cloth. The advantage of this method is its improved stability for cloth simulation. It is defined by Equation 6:

$$X(t + \Delta t) = X(t) + f(X(t + \Delta t))(\Delta t) \tag{6}$$

We substitute $f(X(t + \Delta t))$ by a linear approximation over the Taylor series basis, defining ΔX as $X(t + \Delta t) - X(t)$:

$$\Delta X = f(X(t) + \Delta t))(\Delta t) \tag{7}$$

Approximating $f(X(t) + \Delta t))$:

$$\Delta X = f(X(t) + f^{'}(X(t))\Delta X))(\Delta t) \tag{8}$$

Operating and isolating ΔX:

$$\Delta X = \Big(\frac{1}{\Delta t}I - f(X(t))\Big)^{-1} f(X(t)) \tag{9}$$

where I is the identity matrix. Solving Equation 6 requires computing the linear system described by Equation 9 and solving the integration in the form $X(t + \Delta t) = X(t) + \Delta X$.

Implicit methods require additional calculations, as defined in Equation 9, being therefore computationally expensive. In the simulation, each particle p_{i+1} depends only on its previous state p_i and on the particles directly connected to it, which enables resorting to parallelization to reduce computational cost [Hughes et al. 2007].

4 Collision Detection and Response

4.1 Collision Detection

A collision detection algorithm identifies whether two or more objects from the scene intersect each other. The goal is to avoid overlap and collision between objects, and generate the folds and deformations in the cloth. The algorithm must indicate whether there is a collision, at which time in the animation it will happen and the position of the objects that will come into contact [van den Bergen 2003]. Each object in the scene may collide with any other and an algorithm must compare each pair of objects, primitive-to-primitive. Therefore, a time varying simulation of cloth composed of many primitives with deformation properties is very costly.

Bounding volumes [van den Bergen 2003] are spatial data structures aimed at reducing the number of checks made by the validation and collision treatment algorithm [Millington 2007]. In the following we discuss how to accomplish this.

4.1.1 Bounding Volumes

A bounding volume is a closed volume that completely contains a given object or a part of it. If two objects have bounding volumes that do not collide the objects within them are not in contact. Bounding volumes should satisfy the following properties [Millington 2007]:

- Have the minimum size required to encapsulats the object.
- Require minimum computational time for the overlapping checks.
- Require low memory usage for representing the data structure.
- Require low computational cost in searching for collisions.

Several bounding volume representations exist [Ericson 2004], we shall mention three of them in this paper: bounding spheres, axis aligned bounding boxes (AABB) and oriented bounding boxes (OOB). A bounding sphere is represented by its center and radius and, consequently, overlap computation becomes a simple task of checking whether the distance between the centers of two spheres is less than the sum of their radiuses. Bounding boxes, on the other hand, are represented by one central point and a set of distances from this point to the edges of the box (half the total length of the box in the corresponding direction). Bounding boxes can be either axis aligned or object oriented. The choice of the bounding volume depends on the shape of the enclosed object, as depicted in Figure 4.

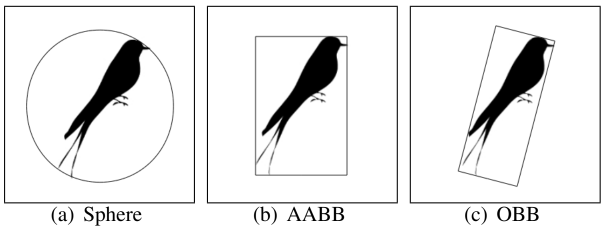


Figure 4: *Bounding volumes delimiting the object (bird). Figure 4(a) illustrates a bounding sphere representation; similarly, AABB and OBB volumes are shown in Figures 4(b) and 4(c), respectively. Ideally a bounding volume for collision detection should adhere to the object as tightly as possible.*

AABBs volumes are three-dimensional rectangular boxes characterized by being aligned to the coordinate system axes. The overlap test is computationally quick and only requires a direct comparison of different coordinate values [Ericson 2004]. Usually, an AABB is represented by the maximum and minimum values for each coordinate axis. Therefore, two AABBs only intersect if, and only if, there is overlap on the three axes. Algorithm 1 describes the procedure for testing overlap.

Algorithm 1: Overlap test

```
Input  : AABB A, AABB B
Output: $true$ if an overlap occurs. $False$ otherwise.
begin
    for $i = 1$ to $3$ do
        if A.maximum[i] $<$ B.minimum[i] or
        A.minimum[i] $>$ B.maximum[i] then
            return $false$;
    return $true$
end
```

4.1.2 Bounding volume hierarchy

A bounding volume hierarchy is a spatial data structure in which nodes store bounding volumes. The present work employs hierarchies based on AABBs. The bounding volume is defined by its center, its axes (oriented in 3D space) and its extension (length on each axis). Building the AABB requires computing a minimal and a maximal points from each triangle's vertex in 3D space [Ericson 2004]. An AABB hierarchy is obtained by a recursive top-down subdivision of the represented object. At each recursion step an AABB is computed for a smaller set of primitives, and the set of primitives is subdivided with respect to a chosen partition plane until each subset contains a single primitive element, as illustrated in Figure 5. Therefore, a set of n primitives has n leaf nodes and $n - 1$ internal nodes [Van Den Bergen 1997]. The tree holds a computational cost of $O(nlog(n))$ in case of AABBH primitives are distributed uniformly, which n is the number of primitives in the model [Van Den Bergen 1997].

After a deformation on the cloth model results from an iteration step, the AABBH rebalancing property is employed to maintain primitives at the leaf nodes, thus avoiding a reconstruction of the tree. Operations are applied only to leaf nodes, at a cost $O(n)$. Leaf nodes are connected into a linked list so that they may be traversed directly, without visiting non-leaf nodes, similarly to a B^+tree data structure [van den Bergen 2003].

4.1.3 Segment-Triangle Intersection

Computing segment-triangle intersection is important since, once a a possible collision is detected on a sector, methods for collision detection between primitives must be applied. Moller proposed an efficient algorithm that performs the calculations in 3D, employing the parametric triangle equations to find the intersection point [Akenine-Moller et al. 2002]. This method may be improved by applying the plane parametric equations and a simple cross product.

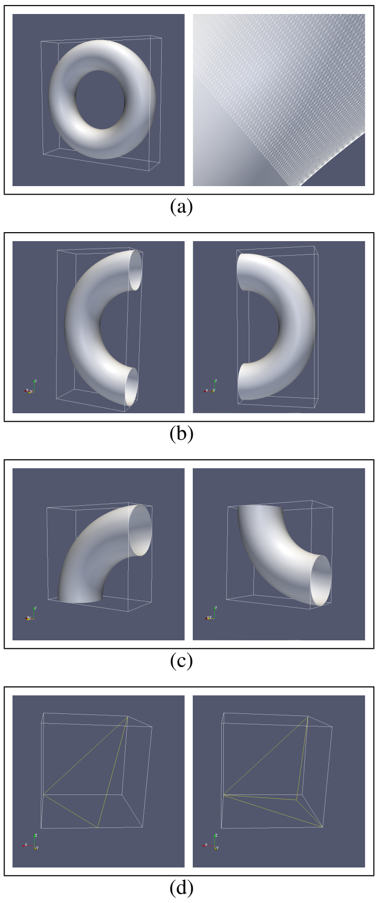


Figure 5: *Subdivision's representation of an object with AABB hierarchy. Figure 5(a) shows the first level of the AABB tree and it is bounding box. The goal is to divide the major axis of the outcome bounding box, associated with its orthogonal plan. Figures 5(b) and 5(c) show the levels 2 and 3, respectively. The final result is encapsulated by a polygon as shown in Figure 5(d).*

Although this variation of Moller's algorithm does not improve precision of the collision computation, it reduces the number of calculations required to determine the collision point. It employs the barycentric coordinates of the triangle to determine whether an intersection point belongs to the triangle or whether it is just part of its containing plane.

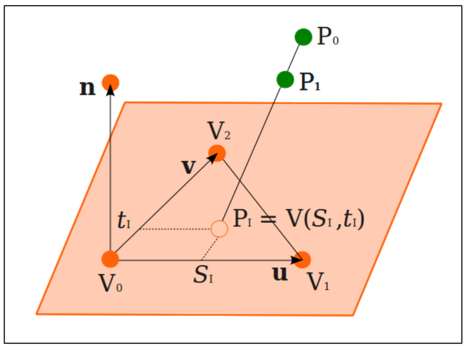


Figure 6: *Example of segment-triangle intersection.*

Let us consider a segment S from point P_0 to P_1 and a triangle T with vertices V_0, V_1 and V_2, as in Figure 6. Triangle T is in plane P with normal vector $\vec{n} = (V_1 - V_0) \times (V_2 - V_0)$. To get an intersection between S and T, we must first get an intersection between S and P. If S and P do not intersect, then neither do S and T. However, if an intersection exists between the segment and the plane, one must determine whether the intersection point belongs to the triangle in order to get a valid intersection.

4.1.4 Plane-Triangle Intersection

Let us consider a triangle T defined by vertices P_0, P_1 and P_2 contained on a plane p_1 with normal vector $\vec{n_1}$, and a plane p_2 with normal vector $\vec{n_2}$. If p_1 and p_2 are not parallel they intersect on a line L. If T intersects p_2 the intersection will be a line segment contained in line L. If T and p_2 do not intercept each other, then the three triangle vertices are on the same side of the plane. On the other hand, if T and P_2 do intersect, one of the points in T must be on one side of the plane and the other two on the opposite one. This may be verified by inspecting the sign of the distance computed from the point to the plane. Suppose that P_0 is on one side of p_2 and that P_1 and P_2 are on the other side. Then the two segments P_0P_1 and P_0P_2 intersect p_2 at two points I_1 and I_2 which are on the intersection line of p_1 and p_2. Segment I_1I_2 is the intersection between triangle T and the plane p_2 [Akenine-Moller et al. 2002]. Figure 7 illustrates this type of intersection.

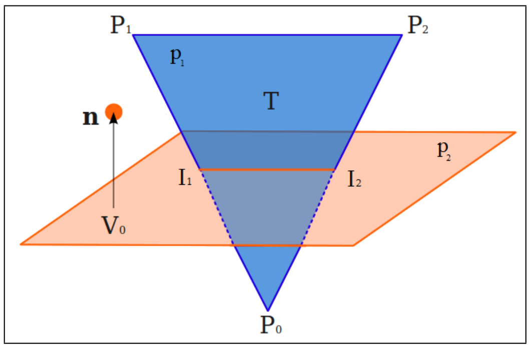


Figure 7: *Example of plane-triangle intersection.*

4.1.5 Triangle-Triangle Intersection

Finally, it is necessary to check the triangle-triangle intersection to avoid possible penetrations undetected by the segment-triangle algorithm, thus ensuring the cloth does not enter into any object. This process is simpler and faster than the segment-triangle intersection, and consists in finding the triangles' plane equations and checking if an intersection exists between them. Consider two triangles T_1 and T_2, each of them lying on a plane, p_1 and p_2, respectively. Their intersection must be on a line of intersection L between the two planes. Let the intersection between T_1 and p_2 be $R_1 = I_{11}I_{12}$, and the intersection between T_2 and p_1 be $R_2 = I_{21}I_{22}$. If either R_1 or R_2 is empty (that is, one triangle does not intersect the plane of the other), then T_1 and T_2 do not intersect. Otherwise their intersection is equal to the intersection between the two segments R_1 and R_2 on line L [Akenine-Moller et al. 2002]. The overlap test is performed with the six segments from the triangles and finding one overlapped pair is sufficient to detect a collision. Figure 8 shows this type of intersection.

4.2 Collision Response

Once a collision has been detected between two particles, their position must be modified to proceed with the animation. Bridson et al. proposed a strategy for updating the speed of particles using the formulae of friction to compute their new position [Bridson et al. 2005]:

$$V_n = (C_n)^2.P_n \tag{10}$$

$$V_t = P_v - V_n \tag{11}$$

$$P_v = -V_n + \mu V_t \tag{12}$$

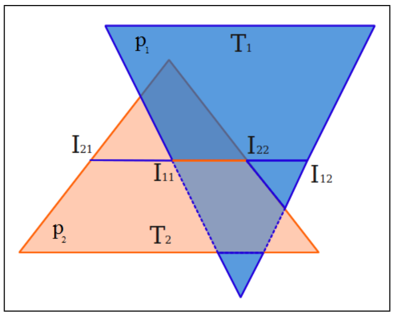


Figure 8: *Example of triangle-triangle intersection.*

where V_n is the normal's speed of the collision, P_n is the normal at the particle, C_n is the surface normal at the collistion point, V_t is the tangential velocity, μ is the friction coefficient and P_v is the particle speed.

5 Implementation details

Graphics Processing Units (GPUs) are optimized for computationally-intensive and highly parallel operations and as such provide an interesting platform for computationally demanding applications. The release of software development kits such as the NVIDIA-CUDA has encouraged their use as a computational unit to offload the CPU in application domains other than graphics.

The CUDA programming model consists of both host (CPU and cache) and device (GPU and video memory) functions. The former are written in nearly standard C/C++, executed on CPU, and used to call the latter, written in annotated C and executed on GPU so as to accelerate highly parallel and computationally intensive tasks. In the cloth simulation problem, single CPU architectures impose limits on quality and performance for high demanding simulations. Resorting to GPU programming is one approach to overcome these performance limitations.

As discussed in Section 3 cloth is simulated as a network of springs where each point is connected to a set of neighboring points, defining a large mass-spring system of connected nodes. It worth noticing that a large number of mass-spring simulations can be handled in parallel. Nonetheless, problems involving neighborhood information do not show good data locality properties and hence are not easily mapped to massive parallel architectures. To solve this problem we split the system into particle blocks, as depicted in Figure 9, which can be dynamically assigned to threads. Zara et al. proposed a similar strategy but implemented on a PC cluster. As they shown this strategy is suited to irregular problems where the computation task can not be split into small and equally weighted computation processes [Zara et al. 2002].

In our implementation the cloth's grid vertices are stored in the video card memory, which can be directly accessed and modified with CUDA. Hence, one may retrieve information relative to many particles simultaneously with no need of moving data between host and device. At a time instant t, positions, velocities and accelerations of all particles are stored in three different arrays. Employing a simple GPU matrix data structure allows this information to be efficiently retrieved, and the forces exerted by springs on the connected points are computed from Hook's law, as discussed in Section 3.2. By splitting each of these three arrays into several blocks, each block as a shared object, computation is performed on several particles simultaneously. These arrays split into shared blocks are processed separately, and local interactions are processed individually. Block-to-block interactions are processed by parallel threads, each one dealing with a pair of blocks, as illustrated in Figure 9.

6 Experiments

Our purpose is to effectively model the dynamic behavior of the cloth under different scenes and objects. In this section we

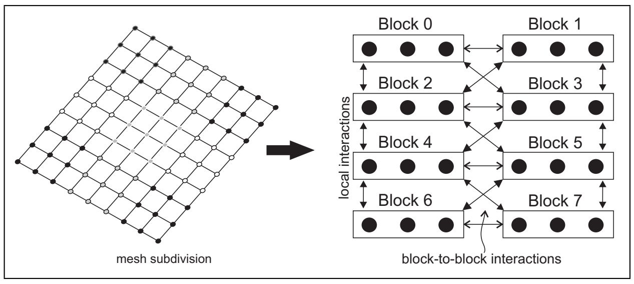


Figure 9: *The particle system split as a particles blocks.*

present comprehensive cloth simulation experiments considering both a quantitative and a qualitative perspectives. The experiments consider three objects: a cuboid, a sphere and a toroid. We assess different cloth model sizes and evaluate their visual outcome, considering two alternative implementations, namely on CPU and GPU. We also show that using multi-threaded programming on GPU and CPU increased the speed of the implicit integration method, and consequently, the whole frame generation rate. All implementations are in C++ and core implementation adopts standard POSIX thread library on CPU and NVIDIA CUDA on GPU. Finally, Paraview [Law et al. 2001] was employed for scene visualization and animation.

6.1 Experimental Setup

Experiments were performed on a computer Core2Quad 3.0GHz with 8 GB RAM, video card Quadro FX 3800, 1024 MB, featuring a 192-core NVIDIA® CUDA™ parallel computing architecture. Table 1 lists the processor and compiler specifications employed in the experimental setup.

Processor	Type	# of Cores	# of Threads	RAM	Compiler
Intel Core2Quad 3GHz	CPU	4	2	8 GB	g++ 4.4.1
Intel Core2Quad 3GHz	CPU	4	4	8 GB	g++ 4.4.1
Intel Core2Quad 3GHz	CPU	4	8	8 GB	g++ 4.4.1
NVIDIA FX 3800	GPU	192	768	1 GB	nvcc 3.1

Table 1: *Compiler and processor specifications employed in the experiments.*

Table 2 summarizes the three cloth models considered, each with a different grid resolution, number of particles and number of springs.

Name	Resolution	# Particles	# Springs
Mesh 1	50 x 50	2.500	9.702
Mesh 2	100 x 100	10.000	39.402
Mesh 3	200 x 200	40.000	158.802

Table 2: *Experiments performed with three distinct cloth models (different grid resolution and mesh sizes).*

6.2 Qualitative Performance

A qualitative performance analysis is accomplished by generating several animations and renderings of scenes composed of the cloth and rigid objects, considering a variety of complex interactions among cloth and objects. Three scene geometries are considered: in the first scene a cloth is thrown over the surface of a cube; in the second the cloth is thrown over the surface of a sphere and in the third scene it interacts with the surface of a toroid. These experiments employed cloth models at three different resolutions: 50 x 50, 100 x 100 and 200 x 200. Representative results are illustrated in Figures 10, 11, 12, 13, 14, 15 and 16. Figures show that the cloth behavior is adaptable to any object. Better realism and resolution are achieved when employing larger meshes. Figure 13 illustrates the cloth falling and then crashing into a smooth ball.

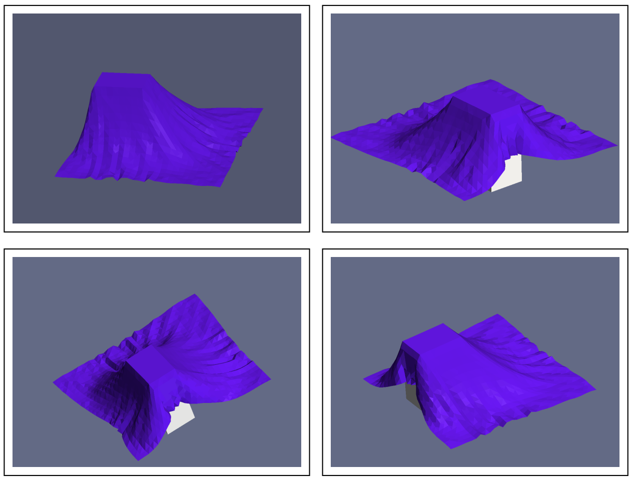


Figure 10: *Cloth simulation with a mesh with resolution 50 x 50 and 2.500 triangles. This scene illustrates interaction of the cloth with a cube and the floor.*

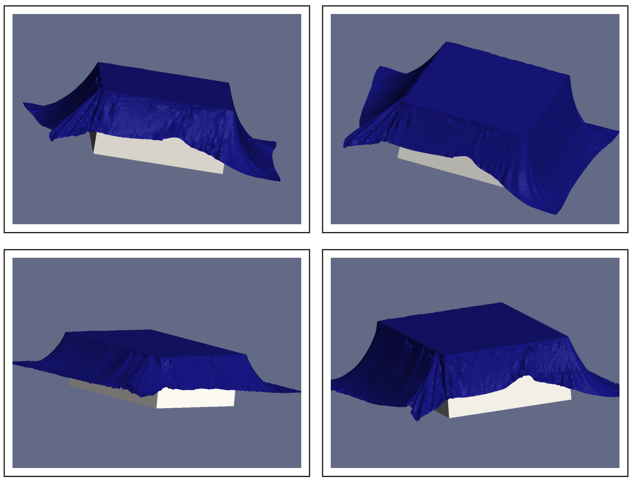


Figure 11: *Cloth simulation with a 100 x 100 resolution mesh and 10.000 triangles, illustrating interaction of the cloth with a cube.*

6.3 Quantitative Experiments

In high-resolution cloth simulations employing the implicit integration method the floating-point operations are the most expensive computations. In this section we investigate the performance of the implicit integration step in the proposed solution by considering different numbers of threads running on CPU and GPU. Notice that in all cases the algorithms running on the GPU-CUDA produced identical results to those running on CPU.

Performance is measured as the relative speedup, expressed by $(\frac{time\_CPU}{time\_GPU})$. Figure 17 and Table 3 summarize the execution times of the complete frame generation processes on both GPU and CPU.

The frame generation process demands higher computational capability as mesh model sizes increase. On GPU the proposed parallel program achieves higher speedup on larger problem sizes. In the workload characterization analysis presented in Section 3.3 we highlighted the intense computational demand of the integration method. GPUs provide optimal arithmetic operation capability when SIMD operations are fully pipelined – which does not happen when handling small model sizes. Moreover, on handling larger mesh sizes one can start additional threads to hide memory access

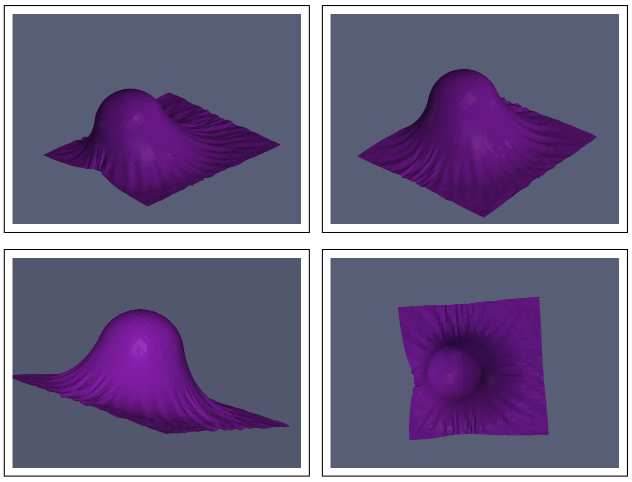


Figure 12: *Cloth simulation with a 50 x 50 resolution mesh and 2.500 triangles. The sphere has a radius of 6 units and the cloth is resting on the floor.*

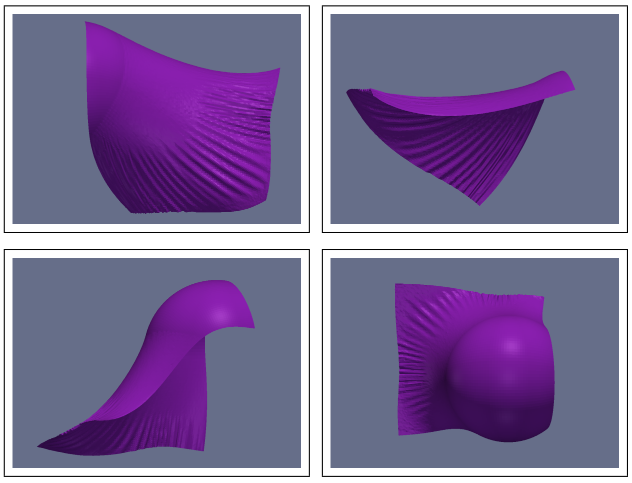


Figure 13: *Cloth simulation using a mesh model with 10.000 triangles. In this scene the sphere was moved and detached from the floor.*

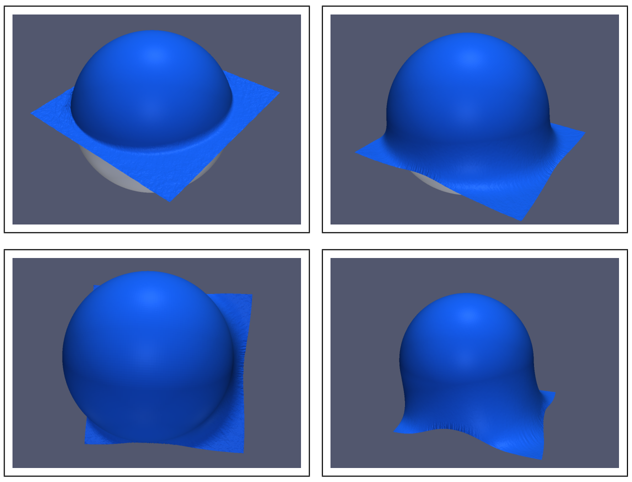


Figure 14: *Cloth simulation using a mesh with 40,000 triangles, cloth interacting with a sphere.*

latency. Therefore, it is reasonable to argue that the proposed GPU implementation scales well with the problem size.

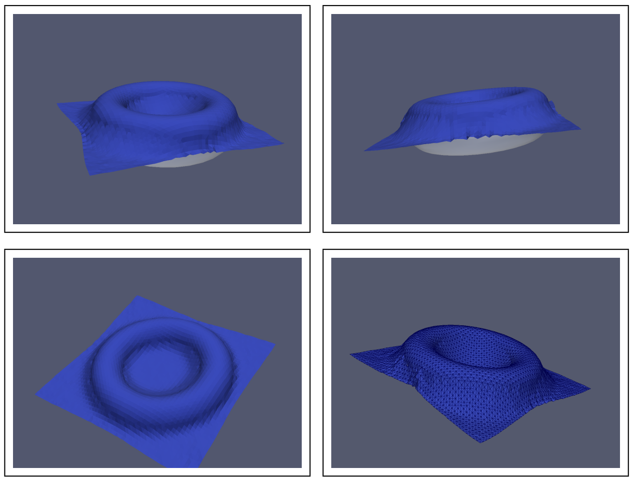


Figure 15: *Cloth simulation using a mesh model with 2.500 triangles, interacting with a toroid.*

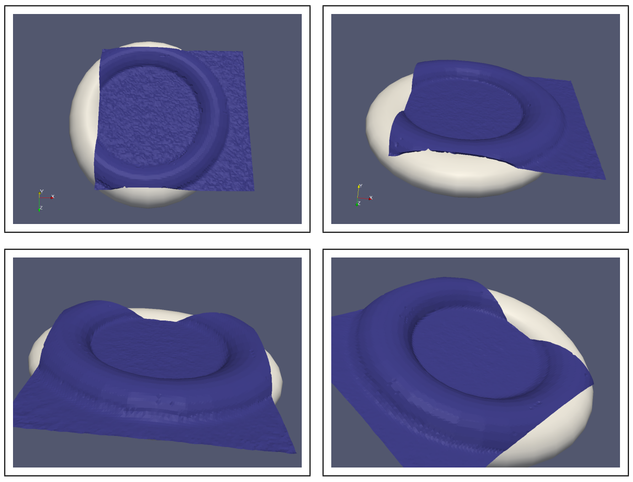


Figure 16: *Cloth simulation with a 100 x 100 resolution mesh and 10.000 triangles, illustrating interaction of the cloth with a toroid.*

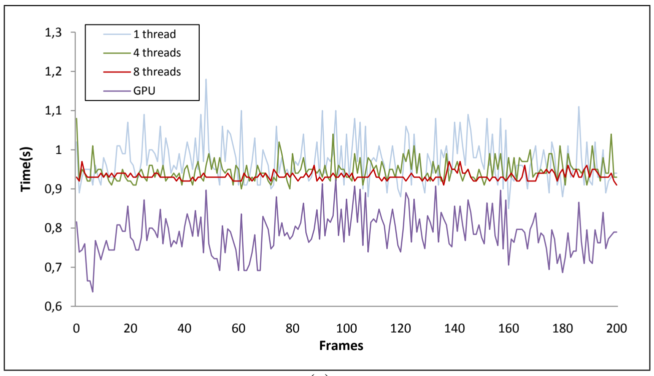
Nome	GPU (s)	CPU (s)	Speedup
Mesh 1	0.79	0.97	1.24
Mesh 2	4.18	10.45	2.50
Mesh 3	74.84	158.46	2.12

Table 3: *Comparison of average execution times on GPU and CPU for different cloth mesh sizes.*

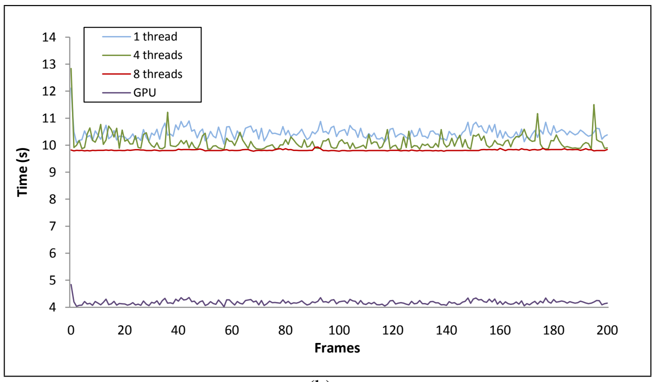
7 Conclusions

For decades simulation of physical objects has been object of study in computer graphics, resulting in a dramatic evolution of the techniques employed. Deformable objects such as fabrics, cloth, or rubber balls may now be effectively modeled with techniques that incorporate physical properties of object behavior, described in terms of interaction with external and internal forces. Simulations attempt to render realistic animations of fabrics and cloth with appearance and characteristics similar to the real world. In approaches that adopt computationally intensive implicit integration methods to solve the simulation model, achieving realism and high-throughput rates on large models is still an issue.

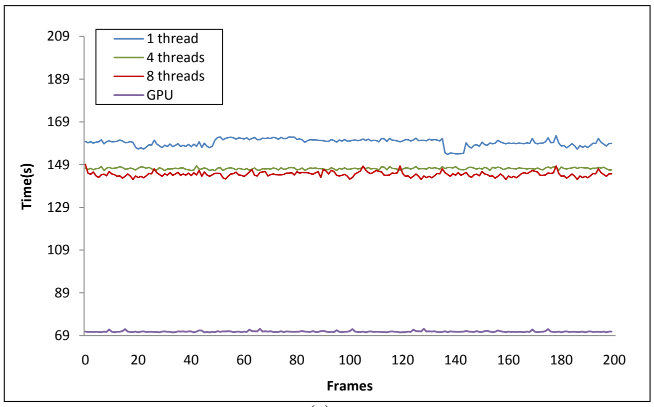
We addressed this problem in this paper. Collision treatment is improved by employing AABB hierarchies, and motion generation is accelerated by employing suitable integration methods efficiently implemented on multi-threaded CPU and on an emerging multi-core GPU-CUDA architectures. Our experiments have shown



(a)



(b)



(c)

Figure 17: *Comparing execution times for different numbers of threads running on CPU and GPU. Figures 17(a), 17(b) and 17(c) show meshes with resolution 50 x 50, 100 x 100 and 200 x 200, respectively.*

that using threads, both on CPU and GPU, ensures adequate performance even for larger cloth models. Therefore, it is reasonable to argue that it is now possible to handle high resolution cloth simulations in real-time with efficient implementations on GPU and multi-core architectures.

References

AKENINE-MOLLER, T., MOLLER, T., AND HAINES, E. 2002. *Real-Time Rendering*. A. K. Peters, Ltd., Natick, MA, (page 66USA.

BARAFF, D., AND WITKIN, A. 1998. Large steps in cloth simulation. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 43–54.

BARAFF, D., WITKIN, A., AND KASS, M. 2003. Untangling cloth. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, ACM, New York, NY, USA, 862–870.

BAYRAKTAR, S., GUDUKBAY, U., AND OZGUC, B. 2007. Practical and realistic animation of cloth. In *3DTV Conference, 2007*, 1 –4.

BRIDSON, R., MARINO, S., AND FEDKIW, R. 2005. Simulation of clothing with folds and wrinkles. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Courses*, ACM, New York, NY, USA, 3.

CORDIER, F., SEO, H., AND MAGNENAT-THALMANN, N. 2003. Made-to-measure technologies for an online clothing store. *IEEE Comput. Graph. Appl. 23*, 1, 38–48.

ERICSON, C. 2004. *Real-Time Collision Detection (The Morgan Kaufmann Series in Interactive 3-D Technology) (The Morgan Kaufmann Series in Interactive 3D Technology)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

GOTTSCHALK, S., LIN, M. C., AND MANOCHA, D. 1996. OBBTree: a hierarchical structure for rapid interference detection. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 171–180.

HOUSE, D. H., AND BREEN, D. E., Eds. 2000. *Cloth modeling and animation*. A. K. Peters, Ltd., Natick, MA, USA.

HU, X., BAI, Y., CUI, S., DU, X., AND DENG, Z. 2009. Review of cloth modeling. In *Computing, Communication, Control, and Management, 2009. CCCM 2009. ISECS International Colloquium on*, vol. 4, 338–341.

HUGHES, C. J., GRZESZCZUK, R., SIFAKIS, E., KIM, D., KUMAR, S., SELLE, A. P., CHHUGANI, J., HOLLIMAN, M., AND CHEN, Y.-K. 2007. Physical simulation for animation and visual effects: parallelization and characterization for chip multiprocessors. *SIGARCH Comput. Archit. News 35*, 2, 220–231.

KANG, Y.-M., CHOI, J.-H., CHO, H.-G., AND PARK, C.-J. 2000. Fast and stable animation of cloth with an approximated implicit method. In *Computer Graphics International, 2000. Proceedings*, 247 –255.

LAW, C. C., HENDERSON, A., AND AHRENS, J. 2001. An application architecture for large data visualization: a case study. In *PVG '01: Proceedings of the IEEE 2001 symposium on parallel and large-data visualization and graphics*, IEEE Press, Piscataway, NJ, USA, 125–128.

LV, M. Y., LI, F. M., TANG, Y., AND BI, W. H. 2007. A fast self-collision detection method for cloth animation based on constrained particle-based model. In *DMAMH '07: Proceedings of the Second Workshop on Digital Media and its Application in Museum & Heritage*, IEEE Computer Society, Washington, DC, USA, 140–145.

MACIEL, A., BOULIC, R., AND THALMANN, D. 2007. Efficient collision detection within deforming spherical sliding contact. *IEEE Transactions on Visualization and Computer Graphics 13*, 3, 518–529.

MEZGER, J., KIMMERLE, S., AND ETZMUβ, O. 2002. Progress in collision detection and response techniques for cloth animation. In *PG '02: Proceedings of the 10th Pacific Conference on Computer Graphics and Applications*, IEEE Computer Society, Washington, DC, USA, 444.

MILLINGTON, I. 2007. *Game Physics Engine Development (The Morgan Kaufmann Series in Interactive 3D Technology)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

MÜLLER, M., KEISER, R., NEALEN, A., PAULY, M., GROSS, M., AND ALEXA, M. 2004. Point based animation of elastic, plastic and melting objects. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 141–151.

PROVOT, X. 1995. Deformation constraints in a mass-spring model to describe rigid cloth behavior. In *Graphics Interface 1995*, Canadian Human-Computer Communications Society, 147–154.

S. ALZAMORA, G. G., PONCE ATENCIO, Y., AND ESPERANÇA, C. 2008. Simulation of deformable bodies based on tetrahedral meshes and shape matching. In *Proceedings of the VII Games and Digital Entertainment Symposium - Computing Track*, 108 – 114.

SELLE, A., SU, J., IRVING, G., AND FEDKIW, R. 2009. Robust high-resolution cloth using parallelism, history-based collisions, and accurate friction. *IEEE Transactions on Visualization and Computer Graphics 15*, 2, 339–350.

VAN DEN BERGEN, G. 1997. Efficient collision detection of complex deformable models using AABB trees. *J. Graph. Tools 2*, 4, 1–13.

VAN DEN BERGEN, G. 2003. *Collision Detection in Interactive 3D Environments (The Morgan Kaufmann Series in Interactive 3D Technology).* Morgan Kaufmann.

VOLINO, P., AND THALMANN, N. M. 2000. Implementing fast cloth simulation with collision response. In *CGI '00: Proceedings of the International Conference on Computer Graphics*, IEEE Computer Society, Washington, DC, USA, 257.

WIELAND, F., CARNES, D., AND SCHULTZ, G. 2001. Using quad trees for parallelizing conflict detection in a sequential simulation. In *PADS '01: Proceedings of the fifteenth workshop on Parallel and distributed simulation*, IEEE Computer Society, Washington, DC, USA, 117–123.

ZARA, F., FAURE, F., AND VINCENT, J.-M. 2002. Physical cloth simulation on a pc cluster. In *EGPGV '02: Proceedings of the Fourth Eurographics Workshop on Parallel Graphics and Visualization*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 105–112.

ZHIBIN, L., AND ZHANLI, L. 2006. Dynamic cloth animation in virtual environments. In *IV '06: Proceedings of the conference on Information Visualization*, IEEE Computer Society, Washington, DC, USA, 761–765.

ZHOU, C., ZHU, H., JIN, X., AND FENG, J. 2005. Efficient and simple cloth animation. In *CAD-CG '05: Proceedings of the Ninth International Conference on Computer Aided Design and Computer Graphics*, IEEE Computer Society, Washington, DC, USA, 483–488.