A Comparison of Different Techniques for Haptic Cloth Rendering

Chris Raymaekers, Lode Vanacken, Erwin Cuppens, Karin Coninx
Hasselt University, Expertise centre for Digital Media
and transnationale Universiteit Limburg
Wetenschapspark 2, B-3590 Diepenbeek, Belgium
{chris.raymaekers, lode.vanacken, erwin.cuppens, karin.coninx }@uhasselt.be

ABSTRACT

In haptic cloth rendering, a user can interact with a cloth simulation, while feeling forces that relate to the cloth topology. This can be realized by combining two existing techniques: cloth simulation and collision detection. However, several design choices must be made in order to have a realistic simulation while maintaining real-time rendering times.

This paper discusses the techniques needed for haptic cloth rendering and assesses which techniques are best suited in order to realize the haptic simulation.

1. Introduction

Haptic cloth rendering combines two existing technologies: cloth simulation, which calculates the behaviour of cloth in a virtual simulation and haptic rendering, which is used to make virtual objects touchable by means of a haptic device.

This paper describes how a cloth can be modelled, animated and hapticly rendered using various techniques. The next section describes how such a cloth can be represented. Section 3 discusses how the cloth can be animated using various integration techniques. A third issue to increase the realism of a cloth simulation is collision detection, which will be discussed in section 4. Section 5 will describe some techniques that can be used to render cloth topologies.

The above-mentioned techniques will be combined in section 6 in order to realize a haptic cloth rendering algorithm. Several different technologies that have been used are compared in order to find a good trade-off between realism and processing time.

Finally, we will draw some conclusions and discuss future work.

2. Cloth Simulation

Cloth simulation can be realized by means of two different techniques: Finite Elements and Mass Spring Systems. In this paper we will concentrate on cloth simulation using mass spring systems as this technique is better suited for a haptic simulation.

A first technique uses Finite Elements [1]. An important drawback of this method is its long computation time for detailed simulations.

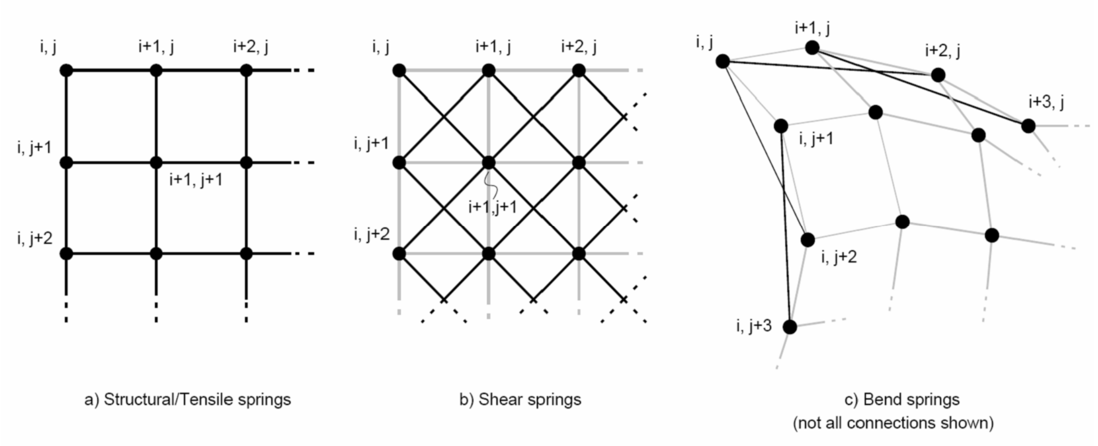

Figure 1: The three different types of springs

Since our goal is to combine cloth rendering with haptic rendering, the processing time has to be optimized. Therefore we will use another technique: Mass Spring Systems, which can obtain visually attractive results with a relative short computation time.
A Mass Spring System is a special type of particle system, consisting of several particles connected by springs. Each particle in the system moves under the influence of Newton's dynamics, based on several physical forces such as gravity, Hook's law, etc
In order to create a cloth topology, the particles are mostly connected by springs in a rectangular mesh [2].

Several techniques exist to simulate the physical properties of the cloth. The most used technique makes use of three different types of springs [3,4], as can be seen in Figure 1:

- structural springs model the resistance to stretching
- shear springs hinder the shearing of individual particles on the cloth surface
- bend springs model the resistance to bending

3. Integration Techniques

In order to simulate movements of the cloth topology, described in the previous section, the position and velocity of the different particles have to be calculated. This is realized through an integration technique.
This section discusses several existing integration techniques. Starting from this overview we will choose the integration technique that is best suitable for haptic rendering in terms of the trade-off between computation speed and physical correctness.

3.1. *Ordinary Differential Equations*

Each of the particles in the cloth's topology is a moving object and therefore its properties should be integrated each timestep. In order to represent the movement properties of a particle we use an initial value problem. These problems can be solved using a first order Ordinary Differential Equation (ODE)[1]:

$$\dot{\mathbf{x}} = f(\mathbf{x}, t) \qquad (1)$$

[1] Please note that the Newton's laws are second order; using a simple conversion we obtain two first order ODE's.

Solving an ODE can be achieved using analytic and numerical methods. As numerical methods are better suited for simulations, we will discus these methods in the remainder of this section.

3.2. *Explicit Integration*

The simplest and most known integration techniques are those based on Taylor series. Using a truncated Taylor series and an approximation for higher order terms we can easily add more accuracy to the simulation. The simplest explicit integration technique, explicit Euler; uses only 2 terms of the Taylor series:

$$\mathbf{x}(t+h) = \mathbf{x}(t) + h\dot{\mathbf{x}}(t) \qquad (2)$$

This technique requires only little computation time, but can lead to an inaccurate and even an unstable simulation.
The accuracy can be increased by introducing more terms: the Explicit Midpoint technique uses 3 terms and Runge-Kutta uses a variable number of terms. The Runge-Kutta with 5 terms (also called Runge-Kutta of 4th order) is the most used variant and provides the best trade off between computational power and expected accuracy [5].

3.3. *Implicit Integration*

Contrary to explicit techniques, where the velocity at the current timestep is used to compute the new position of the particle at the next timestep, implicit techniques take a step back in time and compute the velocity at the position in the new timestep.
An example is Backward Euler [2]:

$$\mathbf{x}(t+h) = \mathbf{x}(t) + h\dot{\mathbf{x}}(t+h) \qquad (3)$$

As this technique uses a linear system, which leads to an increased stability, much larger timesteps can be taken. However, the computation time is much larger than those of explicit techniques.

3.4. *IMEX Integration*

The springs in a cloth can both be stiff (high spring stiffness constant) as non-stiff (low spring stiffness constant). Stiff springs lead more easily to high forces, which cause inaccurate and unstable simulations.
IMEX integration techniques try to combine the advantages of the above-mentioned techniques. The ODE is subdivided in a non-stiff part which is integrated using an explicit technique and a stiff part which is integrated using an implicit technique.

Using this subdivision of an ODE, more computational power is given to stiff equations which are problematic for a stable simulation. At first only the structural springs were integrated implicitly [6] as they usually have much higher stiffness constants. But this is dangerous: as soon as any shear or bend spring becomes stiff, the simulation would diverge. Boxerman et al. [7] split the ODE at run-time. Using a stability criterion, they decide which springs to handle as non-stiff or as stiff.

3.5. Verlet Integration

Verlet integration is often used in molecular dynamics, but can also be used for cloth simulation [8]. The basic Verlet scheme, uses the sum of two Taylor series around the current timestep and then is rearranged to provide the position at the next timestep:

$$\mathbf{x}(t+h) = 2\mathbf{x}(t) - \mathbf{x}(t-h) + \frac{\mathbf{f}(t)}{m}h^2 + O(h^4) \qquad (5)$$

The accuracy of this integration technique is $O(h^4)$ which is only one order below Runge-Kutta of the 4th order but has the advantage that it only evaluates **f** once instead of four times.

A drawback of the basic Verlet scheme is the fact that it doesn't take the velocity into account. The velocity at the current timestep can be calculated when it is needed during the simulation, but this computed velocity will lack one timestep behind. To solve this problem, velocity Verlet and leapfrog Verlet are introduced. In these two schemes, the velocities are computed at the midpoint between the current and the next timestep and are then used to compute the position at the next timestep.

3.6. Inverse Dynamics

In the previous sections we gave a short overview of possible integration techniques that could be used for cloth simulation. One of the most important aspects in any simulation is stability. Inverse dynamics can increase the stability drastically and can be combined with any integration technique. There are two possible methods; the first one repositions the particles after the integration in order to retain the rest length of the springs or deviate a given amount from it [3]. While the second one changes the velocities [9].

4. Collision Detection

In order to increase the realism of the cloth simulation it is necessary that the cloth interacts with other objects in the scene. For example, when the cloth is put on a table, it should drape the table instead of falling through. Also, when a user touches the cloth with the virtual pointer, the cloth should move according to the movements of the pointer.

This behaviour can be supported by means of collision detection. In most cases, collision detection algorithms consist of two phases: the broad phase and the narrow phase. The *broad* phase efficiently decreases the number of possibly colliding objects. The *narrow* phase quickly detects the areas of an object that possibly collides and checks these areas for collision.

Since we are especially interested in finding the intersection of two objects, we will only concentrate on the narrow phase of the collision detection algorithm. Two techniques that can be used to detect collision in the narrow phase are bounding volume hierarchies and spatial subdivision. Both will be presented more in detail in the remainder of this section.

4.1. Bounding Volume Hierarchies

The Bounding Volume Hierarchy (BVH) is one of the most efficient data structures for collision detection and it is mostly used with non-deformable objects.

The main idea behind a BVH is a tree structure where the primitives of the object are recursively subdivided until some leaf criterion is obtained. In most applications this criterion is 1 primitive per leaf. A BVH gets pre-computed but if we want to use a BVH with deformable objects (e.g. a cloth), then we need to rebuild or repair the BVH. Rebuilding will take too long, up to a few seconds. But repairing the BVH can be performed relatively fast. A lot of types of bounding volumes exist, such as sphere, Axis-Aligned Bounding Box (AABB) and Object-aligned Bounding Box (OBB). An ideal bounding volume encloses the primitive as close as possible and can be checked for intersection in a very fast manner. If the BVH has to be repaired, the enclosing of a primitive by the bounding volume also has to be calculated in little time.

A BVH can be constructed top-down, bottom-up or through insertion [10]. The top-down technique is the easiest and most widely used, the group of primitives are subdivided in sets of primitives using heuristics [11, 12]. The number of sets that are used to subdivide the primitives into defines the arity of the BVH. A

BVH of higher arity is faster for deformable objects due to the BVH having less nodes that need repairing and lower recursion depth [13, 14].

To test collision among objects or a self-collision of an object, we run through the BVH's top-down and test recursively pairs of nodes. If two nodes are intersecting, then the primitives of those leafs are checked for intersection. If one node is a leaf and the other is an internal node, then the leaf is checked against all children of the internal node. If both nodes are internal, then the number of collision tests is minimized by checking the internal node with the smallest volume against the children of the internal node with the biggest volume [12, 11].

The BHV can be repaired in a bottom-up and in a top-down manner. Larsson et al. [13] use a hybrid technique, in which the upper half is repaired bottom-up. When during collision detection an unrepaired node is reached, this node and its children are repaired top-down.

Brown et al. [15] propose a bottom-up technique in which they use a priority queue. The nodes are sorted on depth in the tree and the priority queue is initialized with all leafs that contain deformed primitives. As long as the queue is not empty, the top element (node) is updated and its parent is inserted into the queue. Using this update mechanism, recursion is eliminated and all leafs are only updated once.

4.2. *Spatial Subdivision*

Another collision detection technique subdivides the objects in space using a hierarchy. Two possibilities exist: a subdivision for the complete space [16] or the space around the object is subdivided and every object has its own subdivision hierarchy [17]. In this section, we will discuss a technique, which utilizes the first possibility: Optimized Spatial Hashing (OSH).

OSH will divide the space implicitly into small grid cells (uniform grid) and employs a hash function to map 3D grid cells to a hash table. This is not only memory efficient, but also provides flexibility, since this allows for handling potentially infinite regular spatial grids with a non-uniform or sparse distribution of object primitives.

The algorithm runs in two phases. In the first phase, the vertices are classified against the grid cells and added to the hash table. In the second phase, the primitives are hashed. A primitive can intersect multiple grid cells, as can be seen in Figure 2.

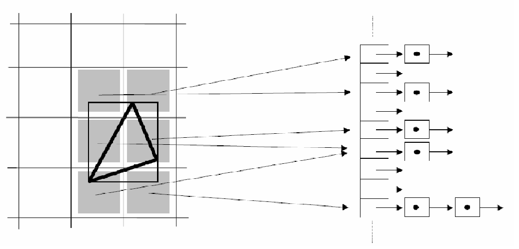

Figure 2: Optimized Spatial Hashing

A discussion about the optimal parameters for OSH can be found in [16].

5. Haptic Rendering

Haptic rendering computes the appropriate forces in order to create the illusion of physical contact using a haptic device.

In this section we will discuss some techniques for the rendering of deformable objects. Since these techniques are often based on rendering methods for rigid objects, two of these techniques are shortly elaborated on [18].

In Penalty Based Methods a repelling force is generated according to the penetration depth into an object. This technique has several problems: pop-through of thin objects and force discontinuities when the penetration direction isn't uniquely defined. In order to solve these problems, Constraint Based Methods were introduced by Zilles et al [19]. A representative object, often called the Surface Contact Point (SCP), is introduced in the virtual environment to replace the haptic device pointer (see Figure 3). A force is then calculated that draws the pointer towards the SCP.

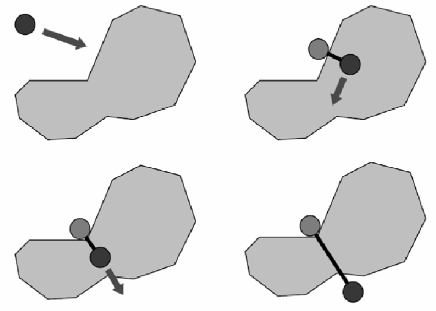

Figure 3: Constraint Based Method: the lighter circle is the representative object (SCP).

As haptic rendering of deformable objects is performed together with a costly physical simulation of the object, an adaptation of the haptic rendering algorithm is not uncommon. We will give an overview of adaptations that are proposed.

Mark et al. [20] propose to use an intermediate model that is updated in the simulation and used during the haptic rendering. They propose a very simple model: a plane. This way forces can be calculated very fast using a penalty based method but some problems do arise. As the plane needs to change according to the movement of the user, discontinuities can occur (e.g. on a sharp edge).

Instead of using an intermediate model to represent the deformable object in the haptic loop, a local model can be used. Here only a part of an object is used in the haptic loop. The haptic loop treats this part as a rigid object. Deformations are computed in a physical simulation. Mendoza et al. [21] propose to attach a virtual bar to the haptic pointer and use the OpenGL picking technique to compute the local model [22]. This way movement is restricted in bar-wise direction but their method is proposed for a laparoscopic simulator where it works fine.

A last approximation structure is the Forcegrid introduced by Mazella et al. [23]. The Forcegrid is an uniform grid and functions as a buffer structure. In this grid approximate forces are stored, which are forwarded to the haptic device using an interpolation function. The Forcegrid is filled up during the simulation. At the beginning the grid has zero-forces and during the simulation a collision detection module checks for contact with an object and puts an appropriate force in the grid. For this reason, it takes a few milliseconds before the user starts to feel any forces. Unfortunately this technique also suffers from some of the problems mentioned earlier.

6. Haptic Cloth Rendering

Haptic cloth rendering can be realized by combining the techniques that were previously discussed in this paper.

This section discusses the design choices of our haptic cloth rendering implementation and evaluates the used techniques. For this purpose, we have developed an evaluation technique for haptic algorithms [24]. In this evaluation, one or more users interact with an object, using a reference algorithm. At each haptic loop, the haptic pointer's position and velocity is recorded. Afterwards, all algorithms are executed using the saved positions and velocities as input. This ensures that all algorithms receive the same input and can be compared in a fair manner.

6.1. Topology

As mentioned earlier, we will use a Mass Spring System to model a cloth. A few simple forces which can act on a Mass Spring System are implemented: gravity and viscous drag.

The cloth itself has some properties which can be selected and adjusted: As we use a rectangular surface, the height and width of the cloth can be chosen together with the number of the particles along this height or width. Furthermore, the cloth's mass can be chosen, and is uniformly divided across the particles.

In order to be able to perform collision detection, each particle has a normal attached to it. These normals are also used for the visible representation, using smooth (Gouraud) shading. Figure 4 shows an example of a cloth, a flag of our research institute. The upper corners of the flag are fixed. The other particles are under the influence of gravity.

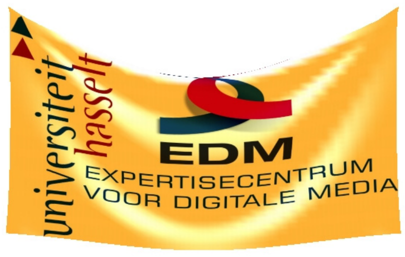

Figure 4: Result of the cloth simulator using smooth shading and texturing

6.2. Integration

As cloth simulations are stiff systems, where the solutions have no problem with an oscillatory behaviour, Verlet integration is probably the best solution according to the taxonomy of Hauth [1].

We have implemented several techniques in order to validate this: Euler, Midpoint and Runge-Kutta 4th, together with the 3 Verlet schemes, basic, leapfrog and velocity.

We found that leapfrog Verlet had the best results with a simulation speed that was almost equal to Explicit Euler. Mathematically leapfrog Verlet is the second most stable/accurate integration technique compared to the tech-

niques implemented here, Runge-Kutta 4th is the most stable.
We also implemented inverse dynamics using the technique of Provot [3]. This technique made the cloth more stable, but also made it less flexible and movable. We therefore chose leapfrog Verlet for our implementation.

6.3. Haptic rendering

The haptic cloth rendering was realized using HAL[2], a haptic library we developed in our lab [25]. We integrated the cloth simulator as a separate library into HAL.
To realize the haptic rendering we implemented the techniques discussed in section 4 and used them with Constraint Based Methods discussed in section 5. Afterwards we evaluated these techniques.
Our implementation has a BVH with two possible bounding volumes: a sphere and an AABB. The arity of the tree can be either two (binary tree) or four (quad tree). During the simulation, the tree was repaired using a modification of the algorithm of Brown et al. [15]. Instead of using a priority queue, sorted on depth in the tree, we sort the priority queue on unique identifiers (ID) given to the nodes in a preprocess step. This ID is distributed along the tree by running through the tree in breadth-first and giving every node an increasing number. Using these IDs every parent will have a lower number than its children and every node will have a lower number than its right neighbour.
Furthermore, when updating a node, the last added parent's ID is saved. As the right neighbour of the last treated node is always treated next, we can test if the parent is already added to the priority queue. This reduces the number of times that internal nodes are added to the priority queue. To initialize the priority queue we need to know which leafs of their bounding volume has become invalid. Instead of traversing the tree top-down we add all leafs in an array and use that array to eliminate recursion, which reduces the time needed to repair the tree, as can be seen in Table 1.

#nodes	array	binary	quad
162	0.116	0.153	0.141
722	0.652	0.922	0.799
1682	1.722	2.2277	1.99
3042	2.871	3.914	3.346
4802	4.537	5.4	5.15

Table 1 Comparison of optimized and two top-down node repair selection techniques techniques (times in ms)

[2] http://edm.uhasselt.be/software/hal/

The results of Table 1 were compared using paired student t-tests. We found that all differences are statistical significant ($p<0.01$). We can thus conclude that the optimized array is the best solution.
Furthermore, we tested the number of nodes that needed repairing and the time it took to repair them all. In Table 2, we evaluated sphere and AABB bounding volume hierarchies with an arity of two and four. We can deduce that a sphere BVH invalidates less frequently in our simulations than an AABB and thus needs less updates. We can also note that a higher arity has less computation time because there are always fewer nodes to update as recursion is eliminated. These findings are again confirmed using paired student t-tests ($p<0.01$).
As a comparison, we also tested the Optimized Spatial Hashing algorithm. This is developed for collision detection between deformable objects, but was adjusted to be used with haptic rendering. Since only collision with the haptic device pointer has to be calculated, we only hash the triangles instead of also hashing the vertices, thus removing the first phase of the original algorithm. This algorithms is however significantly slower than the other algorithms ($p<0.01$).
The updating or hashing is executed during the physical simulation, while the result is further used in the haptic rendering. For haptic rendering we implemented the algorithm of Ruspini et al. [26], combined with the force shading algorithm of Morgenbesser et al. [27]. For the collision detection step, we used a naive algorithm tests for collision against all triangles as reference algorithm. The results of this test are summarized in Table 3. We break up the average timings in loops where collision is found and loops where no collision is found as this can make a large difference in the result. As expected, the query times for OSH algorithm are almost constant.

	Binary tree				Quad tree				
	sphere		AABB		sphere		AABB		OSH
#trian-gles	#up-dates	ms	#up-dates	ms	#up-dates	ms	#up-dates	ms	ms
162	263	0.307	323	0.408	190	0.285	247	0.378	1.368
722	1191	1.759	1443	2.429	832	1.1531	1063	2.257	6.071
1682	2506	4.680	3355	7.429	1887	4.442	2675	7.216	13.185
3042	3185	7.211	5341	13.569	3162	6.360	3865	11.205	23.760
4802	6433	13.650	9401	22.686	4284	12.907	6727	20.729	39.998

Table 2 Comparison of average repair times for different representations

	Reference algorithm		OSH		Sphere (arity 2)	
#triangles	CD	NCD	CD	NCD	CD	NCD
126	0.103	0.084	0.025	0.009	0.026	0.010
576	0.752	0.735	0.024	0.013	0.029	0.012
3042	2.782	2.745	0.026	0.009	0.031	0.013
	Sphere (arity 4)		AABB (arity 2)		AABB (arity 4)	
#triangles	CD	NCD	CD	NCD	CD	NCD
126	0.026	0.010	0.023	0.009	0.024	0.008
576	0.028	0.012	0.025	0.009	0025	0.013
3042	0.029	0.013	0.027	0.010	0.027	0.011

Table 3 Comparison of average collision detection times for different representations

For the BVHs, the query times slowly increase when the number of triangles increases. A Paired student t-test indicates that the differences are statistically significant ($p<0.01$). For a small number of triangles, the sphere BVH is the best, while OSH is better for a large number of triangles

6.4. Discussion

Combining the results from this section, one can conclude that for haptic cloth simulation a sphere BVH with arity four is the best performing algorithm. This is caused by the fact that the differences in update times are much higher then the haptic rendering times.
A few problems still arise in our implementation. During the haptic rendering force discontinuities can arise since the cloth topology is changed outside the haptic loop. As the user pushes against the cloth, vibrations are introduced which cannot be eliminated using interpolation between forces.

7. Conclusion and Future Work

In this paper we discussed haptic cloth rendering, a combination of two existing technologies. Both technologies are very computationally expensive. Therefore a number of techniques were assessed in a formal evaluation. From this evaluation, we can conclude that a BVH with a sphere as bounding volume and an arity of four is the best for our cloth simulation.
One problem that still has to be solved concerns the force discontinuities. We believe that this can be realized using two cloth representations in the haptic loop: the newly calculated cloth and the previously calculated cloth. The force on the haptic pointer should be calculated on both cloths and integrated, resulting in a smooth transition from one version of the cloth to the next.
Furthermore, no self-collisions are currently calculated. The different representations should also be evaluated on this test.

8. Acknowledgements

Part of the research at EDM is funded by EFRO (European Fund for Regional Development), the Flemish Government and the Flemish interdisciplinary institute for broadband technology (IBBT).

9. References

1. Hauth M (2003) Numerical Techniques for Cloth Simulation. In: Clothing Simulation and Animation, Siggraph 2003 Course #29.
2. Baraff D, Witkin A (1998) Large Steps in Cloth Simulation. In: Proceedings of ACM SIGGRAPH 98, pp 43-54.

3. Provot X (1995) Deformation Constraints in a Mass-Spring Model to Describe Rigid Cloth behavior. In: Graphics Interface ’95, pp 147-154.
4. Lander J (1999) Devil in the Blue Faceted Dress: Real-time Cloth Animation. In: Game Developer Magazine.
5. Volino P, Magnenat-Thalmann N (2001) Comparing Efficiency of Integration Methods for Cloth Simulation. In: Proceedings of the 19th Computer Graphics International Conference (CGI-01), pp 265-274.
6. Eberhardt B, Etzmuß O, Hauth M (2000) Implicit-Explicit Schemes for Fast Animation with Particle Systems. In: Proceedings of the Eurographics Workshop on Computer Animation and Simulation 2000 (CAS 2000).
7. Boxerman E, Ascher U (2004) Decomposing Cloth. In: Eurographics/ACM SIGGRAPH Symposium on Computer Animation, pp 153-161.
8. Jakobsen T (2001) Advanced Character Physics. In: Proceedings of GDCONF 2001.
9. Vassilev T, Spanlang B, Chrysanthou Y (2001) Fast Cloth Animation on Walking Avatars. Computer Graphics Forum 20(3), pp 260-267.
10. Zachmann G, Langetepe E (2003) Geometric Data Structures for Computer Graphics. In: Proceedings of ACM SIGGRAPH.
11. Quinlan S (1994) Efficient Distance Computation between Non-Convex Objects. In: Proceedings of the IEEE International Conference On Robotics and Automation, pp 3324-3329.
12. Van Den Bergen G (1997) Efficient collision detection of complex deformable models using AABB trees. J. Graph. Tools 2(4), pp 1-13.
13. Larsson T, Akenine-Möller T (2001) Collision Detection for Continuously Deforming Bodies. In: Eurographics 2001, Short Presentations, pp 325-333.
14. Mezger J, Kimmerle S, Etzmuß O (2003) Hierarchical Techniques in Collision Detection for Cloth Animation. Journal of WSCG 11(2), pp 322-329.
15. Brown J, Sorkin S, Bruyns C, Latombe J-C, Montgomery K, Stephanides M (2001) Real-Time Simulation of Deformable Objects Tools and Application. In: Computer Animation 2001, pp 228-236.
16. Teschner M, Heidelberger B, Müller M, Pomeranets D, Gross M (2003) Optimized Spatial Hashing for Collision Detection of Deformable Objects. In: Proceedings of the Conference on Vision, Modeling and Visualization 2003 (VMV-03), pp 47-54.
17. Ganovelli F, Dingliana J, O’Sullivan C (2000) BucketTree: Improving Collision Detection between Deformable Objects. In: Spring Conference on Computer Graphics (SCCG ’00), pp 156-163.
18. SIGGRAPH-ACM publication (1999) Haptics: From Basic Principles to Advanced Applications. Course Notes for SIGGRAPH '99 #38.
19. Zilles C B, Salisbury J K (1996) A Constrained-Based God-object Method For Haptic Display. In: IROS '95: Proceedings of the International Conference on Intelligent Robots and Systems-Volume 3.
20. Mark W R, Randolph S C, Finch M Van Verth J M, Taylor R M II (1996) Adding Force Feedback to Graphics Systems - Issues and Solutions. In: SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, pp 447-452.
21. Mendoza C A, Laugier C (2001) Realistic Haptic Rendering for Highly Deformable Virtual Objects. In: Virtual Reality, pp 264-270.
22. Lombardo J-C, Cani M-P, Neyret F (1999) Real-time Collision Detection for Virtual Surgery. In: Proceedings of the Computer Animation (CA '99), pp 82-90.
23. Mazzella F, Montgomery K, Latombe J-C (2002) The Forcegrid - A Buffer Structure for Haptic Interaction with Virtual Elastic Objects. In: Proceedings of the IEEE International Conference on Robotics and Automation.
24. De Boeck J, Raymaekers C, Coninx K (2005) A Method for the Verification of Haptic Algorithms. Preproceedings 12th International Workshop on Design, Specification and Verification of Interactive Systems (DSVIS ’05), pp 85-96.
25. Raymaekers C, De Boeck J, Coninx K (2005) An Empirical Approach for the Evaluation of Haptic Algorithms. In: Proceedings of First Joint Euro-Haptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems (WorldHaptics 2005), pp 567-568.
26. Ruspini D C, Kolarov K, Khatib O (1997) The haptic display of complex graphical environments. In: SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques, pp 345-352.
27. Morgenbesser H B, Srinivasan M A (1996) Force Shading for Shape Perception in Haptic Virtual Environments. In: Touch Lab Report 4. RLE TR-606.