EUROGRAPHICS 2001/ A. Chalmers and T.-M. Rhyne
(Guest Editors)

Volume 20 (2001), Number 3

Fast Cloth Animation on Walking Avatars

T. Vassilev, B. Spanlang, Y. Chrysanthou

Department of Computer Science, University College London, United Kingdom

Abstract

This paper describes a fast technique for animating clothing on walking humans. It exploits a mass-spring cloth model but applies a new velocity directional modification approach to overcome its super-elasticity. The algorithm for cloth-body collision detection and response is based on image-space interference tests, unlike the existing ones that use object-space checks. The modern workstations' graphics hardware is used not only to compute the depth maps of the body but also to interpolate the body normal vectors and velocities of each vertex. As a result the approach is very fast and makes it possible to produce animation at a rate of three to four frames per second.

1. Introduction

The main objective of this work is to develop a fast technique for clothing and animating virtual humans. The final goal is to implement a system on the WEB, where customers will be able to upload the 3D virtual representation of their body, browse different pieces of clothing, try them on, see themselves walking wearing the clothing and buy if they are satisfied. The body model is acquired with a high-resolution 3D scanner, so it represents very accurately the body shape of a real person. Because of the accuracy of 3D scanning technology it will be possible not only to try on different types of clothes, but also to fit different sizes.

The system is based on an improved mass-spring model of cloth and a fast new algorithm for cloth-body collision detection. It reads as input a body file and a garment text file. The garment file describes the cutting pattern geometry and seaming information. The former can be derived from the existing apparel CAD/CAM systems, such as GERBER. The cutting patterns are positioned around the body and elastic forces are applied along the seaming lines. After a certain number of iterations the patterns are seamed, i.e. the garment is 'put on' the human body. Then gravity is applied and a body walk is animated.

The main contributions of this work are the velocity directional modification approach for dynamics and cloth-body hardware-assisted method for collision detection and response. The rest of the paper is organised as follows. Next section reviews previous work on cloth simulation and collision detection. Section 3 describes the cloth model and its improvement to cope with super-elasticity. Section 4 describes the collision detection algorithm in detail. Section 5 is dedicated to collision response. Section 6 gives results as images and comparison tables and section 7 concludes the paper and gives ideas about future work.

2. Background

2.1. Previous work in cloth simulation

Physically based cloth modelling has been a problem of interest to computer graphics researchers for more than a decade. First steps, initiated by Terzopoulos et al. [1,2], characterised cloth simulation as a problem of deformable surfaces and used the finite element method and energy minimisation techniques borrowed from mechanical engineering. Since then other groups have been formed [3 - 6] challenging the cloth simulation using energy or particle based methods. A more detailed survey on cloth modelling techniques can be found in the paper by Ng and Grimsdale[7].

Many of the approaches described above have a good degree of realism simulating the cloth but their common drawback is low speed. A relatively good result demonstrated by Baraff and Witkin [6] is 14 seconds per

frame for the simulation of a shirt with 6,450 nodes on a SGI R10000 processor. This is the main reason why these techniques cannot be applied to an interactive system.

Provot [8] used a mass-spring model to describe rigid cloth behaviour, which proved to be faster than the techniques described above. Its major drawback is the super-elasticity. In order to overcome this problem he applied a position modification algorithm to the ends of the over-elongated springs. However, if this operation modifies the positions of many vertices, it may elongate other springs. That is why this approach is applicable only if the deformation is locally distributed, which is not the case when simulating garments on a virtual body.

The method, described in this work, also exploits a mass-spring cloth model but it introduces a new approach to overcome its super-elasticity. Instead of modifying the positions of end points of the springs that were already over-elongated, the algorithm checks their length after each iteration and does not allow elongation more than a certain threshold modifying velocities of the corresponding vertices. This approach has been further developed and optimised for the dynamic case of simulating cloth on moving objects.

2.2. Mass-spring model of cloth

The elastic model of cloth is a mesh of $l \times n$ mass points, each of them being linked to its neighbours by massless springs of natural length greater than zero. There are three different types of springs: structural, shear, and flexion, which implement resistance to stretching, shearing and bending, correspondingly.

Let $\mathbf{p}_{ij}(t)$, $\mathbf{v}_{ij}(t)$, $\mathbf{a}_{ij}(t)$, where i=1,…, l and j=1,…, n, be respectively the positions, velocities, and accelerations of the mass points at time t. The system is governed by the basic Newton's law:

$$\mathbf{f}_{ij} = m\,\mathbf{a}_{ij}, \tag{1}$$

where m is the mass of each point and $\mathbf{f}_{ij}$ is the sum of all forces applied at point $\mathbf{p}_{ij}$. The force $\mathbf{f}_{ij}$ can be divided in two categories.

The **internal forces** are due to the tensions of the springs. The overall internal force applied at the point $\mathbf{p}_{ij}$ is a result of the stiffness of all springs linking this point to its neighbours:

$$\mathbf{f}_{int}(\mathbf{p}_{ij}) = -\sum_{k,l} k_{ijkl}\left(\overrightarrow{\mathbf{p}_{kl}\mathbf{p}_{ij}} - l^0_{ijkl}\frac{\overrightarrow{\mathbf{p}_{kl}\mathbf{p}_{ij}}}{\left\|\overrightarrow{\mathbf{p}_{kl}\mathbf{p}_{ij}}\right\|}\right), \tag{2}$$

where k_{ijkl} is the stiffness of the spring linking $\mathbf{p}_{ij}$ and $\mathbf{p}_{kl}$ and l^0_{ijkl} is the natural length of the same spring.

The **external forces** can differ in nature depending on what type of simulation we wish to model. The most frequent ones will be:

- Gravity: $\mathbf{f}_{gr}(\mathbf{p}_{ij}) = m\mathbf{g}$, where $\mathbf{g}$ is the gravity acceleration;
- Viscous damping: $\mathbf{f}_{vd}(\mathbf{p}_{ij}) = -C_{vd}\mathbf{v}_{ij}$, where C_{vd} is a damping coefficient.

For more information on how to model wind see Provot [8].

All the above formulations make it possible to compute the force $\mathbf{f}_{ij}(t)$ applied on point $\mathbf{p}_{ij}$ at any time t. The fundamental equations of Newtonian dynamics can be integrated over time by a simple Euler method:

$$\left|\begin{array}{l}\mathbf{a}_{ij}(t+\Delta t) = \dfrac{1}{m}\mathbf{f}_{ij}(t)\\ \mathbf{v}_{ij}(t+\Delta t) = \mathbf{v}_{ij}(t) + \Delta t\,\mathbf{a}_{ij}(t+\Delta t)\\ \mathbf{p}_{ij}(t+\Delta t) = \mathbf{p}_{ij}(t) + \Delta t\,\mathbf{v}_{ij}(t+\Delta t)\end{array}\right., \tag{3}$$

where Δt is a chosen time step. More complicated integration methods, such as Runge-Kutta [16], can be applied to solve the differential equations. This however reduces the speed significantly, which is very important in our case. The Euler Equations 3 are known to be very fast and give good results, when the time step Δt is less than the natural period of the system $T_0 \approx p\sqrt{m/K}$. In fact our experiments showed that the numerical solving of Equations 3 is stable when

$$\Delta t \le 0.4p\sqrt{\frac{m}{K}}, \tag{4}$$

where K is the highest stiffness in the system.

2.3. Collision detection cloth-body

Collision detection and response prove to be the bottleneck of dynamic simulation algorithms that use highly discretised surfaces. So, if it is necessary to achieve a good performance then an efficient collision detection algorithm is essential. Most of the existing algorithms for detecting collisions between the cloth and other objects in the scene are based on geometrical object-space (OS) interference tests. Some apply prohibitive energy field around the colliding objects [1], but most of them use geometric calculations to detect penetration between a cloth particle and a face of the object together with optimisation techniques in order to reduce the number of checks.

The most common approaches are voxel or octree subdivision [9]. The space is subdivided either into an array of regular voxels or into a hierarchical tree of octants and the detection is performed exploring the corresponding structure. Another solution is to use a bounding box (BB) hierarchy [6, 11]. Objects are grouped hierarchically according to proximity rules and a BB is pre-computed for each object. The collision detection is then performed analysing BB intersections in the hierarchy. Other techniques exploit proximity tracking [10] or curvature computation [11] to reduce the big number of collision

checks, excluding objects or parts which are impossible to collide.

Recently new techniques were developed, based on image-space (IS) tests [12 - 15]. These algorithms use the graphics hardware to render the scene and then perform checks for interference between objects based on the depth map of the image. In this way the 3D problem is reduced to 2.5D. As a result of using the graphics hardware these approaches are very efficient. However, they have been mainly used to detect rigid object interference [12, 15], in CAD/CAM systems [13], in dental practise [14], but never for cloth-body collision detection and response.

This paper exploits an image-space approach to collision detection and response. Its main strength is that it uses the workstation graphics hardware not only to compute the depth maps, necessary for collision detection, but also to generate maps of the normal vectors and velocities for each point on the human body. The latter are necessary for collision response. As a result the algorithm is very fast and the detection and response time do not depend on the number of faces on the human body.

3. Velocity directional modification

The major drawback of the mass-spring cloth model is its "super elasticity". This is due to the fact that the springs are "ideal" and they have unlimited linear deformation rate. As a result the cloth stretches even under its own weight, something that does not normally happen to real cloth.

3.1. Position modification

Provot [8] proposed to cope with the super-elasticity using position modification. His algorithm checks the length of the springs after each iteration and modifies the positions of the ends of the spring, if it exceeds its natural length with more than a certain value (10% for example). This modification will adjust the length of some springs but it might over-elongate others. So, the convergence properties of this technique are not clear. It proved to work for locally distributed deformations but no tests were conducted for global elongation.

3.2. Velocity modification

The main problem of the position modification approach is that first it allows the springs to over-elongate and then tries to adjust their length modifying positions. This of course is not always possible because of the many links between the mass points. Our idea was to find a constraint that does not allow any over-elongation of springs.

The algorithm works as follows. After each iteration it checks for each spring whether it exceeds its natural length by a pre-defined threshold. If this is the case, the velocities are modified, so that further elongation is not allowed. The threshold value usually varies from 1% to 15% depending on the type of cloth we want to simulate.

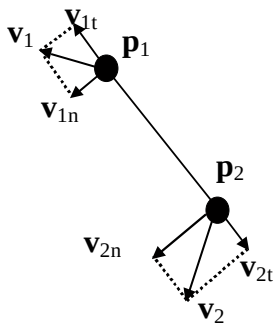

Figure 1: *Velocity modification for over-elongated springs*

Let $\mathbf{p}_1$ and $\mathbf{p}_2$ be the positions of the end points of a spring found as over-elongated, and $\mathbf{v}_1$ and $\mathbf{v}_2$ be their corresponding velocities (see Figure 1). The velocities $\mathbf{v}_1$ and $\mathbf{v}_2$ are split into two components $\mathbf{v}_{1t}$ and $\mathbf{v}_{2t}$, along the line connecting $\mathbf{p}_1$ and $\mathbf{p}_2$, and $\mathbf{v}_{1n}$ and $\mathbf{v}_{2n}$, perpendicular to this line. Obviously the components causing the spring to stretch are $\mathbf{v}_{1t}$ and $\mathbf{v}_{2t}$, so they have to be modified. In general $\mathbf{v}_{1n}$ and $\mathbf{v}_{2n}$ could also cause elongation, but their contribution within one time step is negligible. There are several possible ways of modification:

- Set both $\mathbf{v}_{1t}$ and $\mathbf{v}_{2t}$ to their average, i.e.

$$\mathbf{v}_{1t} = \mathbf{v}_{2t} = 0.5(\mathbf{v}_{1t} + \mathbf{v}_{2t}). \quad (5)$$

- Set only one of them equal to the other, but how to decide which one to change at the current simulation step?

Our experiments showed that Equation 5 is good enough for the static case, i.e. when the cloth collides with static objects. So if we want to implement a system for dressing static human bodies, Equation 5 will be the obvious solution, because it produces good results and is the least expensive. For the dynamics, however, the way of modifying velocity proved to have an enormous influence on the cloth behaviour. For example Equation 5 gives satisfactory results for relatively low rate of cloth deformations and relatively slow moving objects. In faster changing scenes, it becomes "clumsy" and cannot give a proper respond to the environment.

After conducting numerous tests we came up with the following solution. We introduce a vector called a "directional vector", which is computed as:

$$\mathbf{v}_{dir} = \mathbf{v}_{grav} + \mathbf{v}_{object}, \quad (6)$$

where $\mathbf{v}_{object}$ is the velocity of the object which the cloth is colliding with, and $\mathbf{v}_{grav}$ is a component called "gravitational velocity" computed as $\mathbf{v}_{grav} = \mathbf{g}\Delta t$. The directional vector gives us the direction, in which higher spring deformation rates are most likely to appear at the current step of simulation, and in which the cloth should resist to modifications. The components of the directional vector are the sources, which will cause

cloth deformation. In our case they are two, gravity and velocity of the moving object, but in other environments there might be other sources which have to be taken into account, such as wind for example.

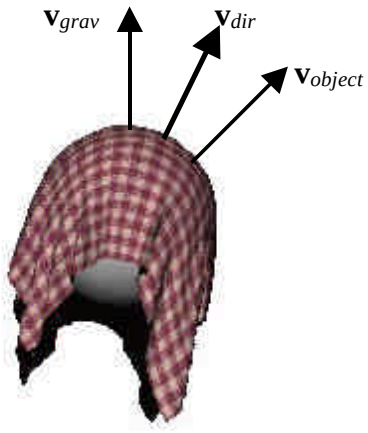

Figure 2. *Introducing directional vector*

Once the directional vector has been determined, the velocities are modified in the following way. Let $\mathbf{p}_{12} = \mathbf{p}_2 - \mathbf{p}_1$ be the spring directional vector and α be the angle between $\mathbf{p}_{12}$ and $\mathbf{v}_{dir}$. The cosine of α can be easily computed as a scalar product of the two vectors.

- If the spring is approximately perpendicular to the directional vector $\mathbf{v}_{dir}$ ($|\cos\alpha|<0.3$) then modify both velocities using Equation 5;
- Else set the velocity of the rear point (considering the directional vector) equal to the front one, so that it can "catch up" with the changing scene. If $\cos\alpha>0$ the rear point is $\mathbf{p}_1$ else it is $\mathbf{p}_2$.

If this is applied to all springs, the stretching components of the velocities are removed and in this way further stretching of the cloth is not allowed. In addition the "clumsiness" of the model is eliminated and it can react adequately to moving objects. As the results show this approach works for all types of deformation: local or global, static or dynamic.

4. Collision detection

Collision detection is one of the crucial parts in fast cloth simulation. At each simulation step, a check for collision with the human model has to be performed for each vertex of the garment. If a collision between the body and a cloth point occurs, the response of that collision also needs to be calculated. In our system we implemented an image-space based collision detection approach. Using this technique it is possible to find a collision only by comparing the depth value of the garment point with the according depth information of the body stored in the depth maps. We went even further and decided to use the graphics hardware to generate the information needed for collision response, that is the normal and velocity vectors of each body point. This can be done by encoding vector co-ordinates (x, y, z) as colour values (R, G, B). Depth, normal and velocity maps are created using two projections: one of the front and one of the back of the model. For rendering the maps we place two orthogonal cameras at the centre of the front and the back face of the body's BB. To increase the accuracy of the depth values, the camera far clipping plane is set to the far face of the BB and the near clipping plane is set to near face of the BB. Both cameras point at the centre of the BB. This is illustrated in Figure 3. The maps are generated at each animation step, although if the body movements are known, they can be pre-computed.

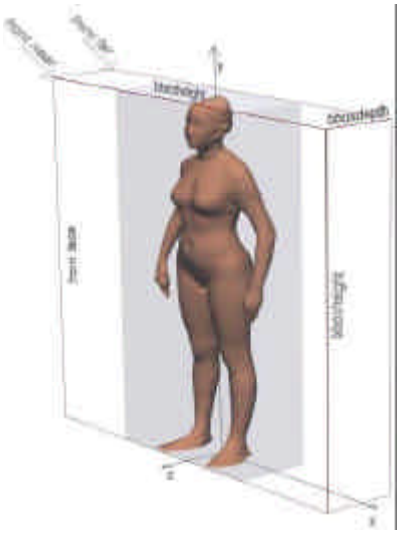

Figure 3. *Front camera clipping planes*

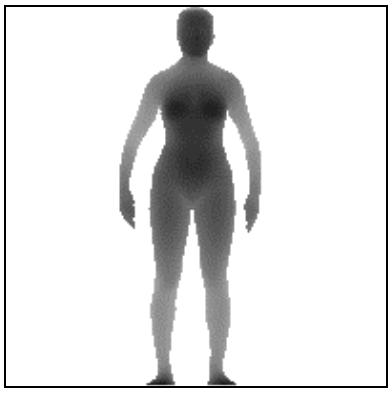

Figure 4: *Front depth map: darker shades are near brighter ones are far*

4.1. Generating the depth maps

When initialising the simulation we execute two off-screen renderings to retrieve the depth values, one for the front and one for the back. The z-buffer of the graphics hardware is moved to main memory using OpenGL's buffer-read function. The z-buffer contains floating-point

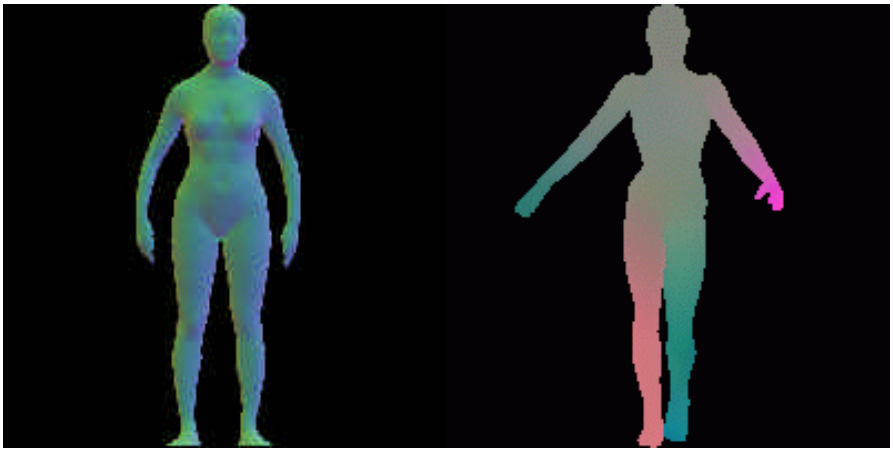

Figure 5*: *Colour maps*
Left - Normal map: colours indicate different direction
Right - Velocity map: colours show direction and magnitude

values from 0.0 to 1.0. A value of 0.0 represents a point at the near clipping plane and 1.0 stands for a point at the far clipping plane.

Figure 4 shows an example depth map.

4.2. Generating the normal maps

During the two renderings for generating the depth, the normal maps are also computed. We substitute the (*Red*, *Green*, *Blue*) value of each vertex of the 3D model with the coordinates (n_x, n_y, n_z) of its normal vector **n**. In this way the frame-buffer contains the normal of the surface at each pixel represented as colour values. Since the OpenGL colour fields are in a range from 0.0 to 1.0 and normal values are from –1.0 to 1.0 the coordinates are converted to fit into the colour fields using

$$\begin{pmatrix} Red \\ Green \\ Blue \end{pmatrix} = 0.5\mathbf{n} + \begin{pmatrix} 0.5 \\ 0.5 \\ 0.5 \end{pmatrix}. \tag{7}$$

We turn on the interpolated-shading option and allow the graphics hardware to interpolate between the normals for all intermediate points. Using OpenGL's read-buffer function to move the frame buffer into main memory gives us a smooth normal map. Conversion from (*Red*, *Green*, *Blue*) space into the normals space then is done using

$$\mathbf{n} = 2\begin{pmatrix} Red \\ Green \\ Blue \end{pmatrix} - \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}. \tag{8}$$

Figure 5* (left) shows an example normal map.

4.3. Generating the velocity maps

Similarly to the rendering of the normal maps, we substitute the (*Red*, *Green*, *Blue*) value of each vertex of the 3D model with the coordinates (v_x, v_y, v_z) of its velocity **v**. Since the velocity coordinate values range from –*maxv* to +*maxv*, they are converted to fit into the colour fields using

$$\begin{pmatrix} Red \\ Green \\ Blue \end{pmatrix} = 0.5\mathbf{v} + \begin{pmatrix} 0.5/maxv \\ 0.5/maxv \\ 0.5/maxv \end{pmatrix}. \tag{9}$$

Again the interpolated-shading option is enabled to allow the graphics hardware to interpolate the velocities for all intermediate points. The conversion from (*Red*, *Green*, *Blue*) space into the velocity space is computed

$$\mathbf{v} = maxv(2\begin{pmatrix} Red \\ Green \\ Blue \end{pmatrix} - \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}). \tag{10}$$

Figure 5* (right) shows an example velocity map.

4.4. Testing for collisions

After retrieving depth, normal and velocity maps testing for and responding to collisions can be done very efficiently. If we want to know whether a point (*x*, *y*, *z*) on the cloth collides with the body, the point's (*x*, *y*) values need to be converted from world coordinate system into the map coordinate system (*X*, *Y*) as shown

$$Y = \frac{y * mapsize}{bboxheight};$$

$$X_{back} = \left(1 - \frac{x + \frac{bboxheight}{2}}{bboxheight}\right) mapsize; \tag{11}$$

$$X_{front} = \left(\frac{x + \frac{bboxheight}{2}}{bboxheight}\right) mapsize.$$

First the z value is used to decide which map to use. If the z value is lower than 0 the back-map is used otherwise the front-map is taken. The corresponding z value of the depth map is compared with the z value of the pixel's coordinates using

$$back: z < depthmap(X_{back}, Y)$$
$$front: z > depthmap(X_{front}, Y) \quad (12)$$

If a collision occurred, we retrieve the normal and velocity vectors from the colour maps indexed by the same coordinates (X, Y) used for the collision check. These vectors are necessary to compute a collision response.

4.5. Estimating the error of discretisation

Considering the fact that most of the modern workstations use a 24 bit z-buffer and that $bboxdepth$<100 cm for an average person, then the following estimation applies for discretisation error in z

$$\Delta z = \frac{bboxdepth}{2^{24}} < \frac{100}{2^{24}} < 6.10^{-6} cm\ . \quad (13)$$

This is more than enough in our case, bearing in mind that the discretisation error of the 3D scanner is of the order of several millimetres. The errors in x and y are equal and can be computed as

$$\Delta x = \Delta y = \frac{bboxheight}{mapsize} \approx \frac{160\ \text{to}\ 180}{mapsize} cm\ , \quad (14)$$

where we consider the average person to be 160 to 180 cm tall. This means that we have control over the error in x and y direction varying the size of the maps. However bigger map size also means bigger overhead, as buffer retrieval times will be higher. A reasonable trade-off is Dx=Dy=0.5 cm, so $mapsize$=320 to 360 pixels.

5. Resolving collisions

After a collision has been detected, the algorithm has to compute a proper response of the whole system. Our approach does not introduce additional penalty, gravitational or spring forces; it just manipulates the velocities, as proposed by Eberhardt et al [4], which proved to be very efficient. However, we had to do some modifications for the dynamic case.

Let $\mathbf{v}$ be the velocity of the point $\mathbf{p}$ colliding with the object $\mathbf{s}$ and let $\mathbf{v}_{object}$ be the velocity of this object (Figure 6). The surface normal at the point of collision is denoted by $\mathbf{n}$. First, the relative velocity between the cloth and the object has to be computed as $\mathbf{v}_{rel} = \mathbf{v} - \mathbf{v}_{object}$. If $\mathbf{v}_t$ and $\mathbf{v}_n$ are the tangent and normal components of the relative velocity $\mathbf{v}_{rel}$, then the resultant velocity can be computed as:

$$\mathbf{v}_{res} = C_{fric}\mathbf{v}_t - C_{refl}\mathbf{v}_n + \mathbf{v}_{object}, \quad (15)$$

where C_{fric} and C_{refl} are a friction and a reflection coefficients, which depend on the material of the colliding objects.

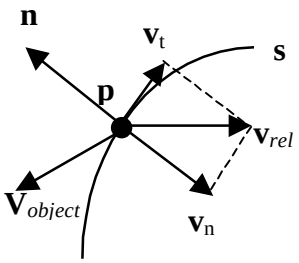

Figure 6: *Resolving collisions manipulating velocities*

A similar approach can be implemented to detect and find the responses not only to vertex-body, but also to face-body collisions. For each quadrangle on the cloth we compute its midpoint and velocity as an average of the four adjacent vertices. Then we check for a collision of this point with the body and if so, compute its response using Equation 15, and apply the same resultant velocity to the surrounding four vertices. If there is more than one response for a vertex, then an average velocity is calculated for this vertex. As the results section shows, this approach helps to reduce significantly the number of vertices, which speeds up the whole method.

Our tests showed that the velocity collision response did not always produce satisfactory results. For example, when heavy cloth was simulated there were penetrations in the shoulder areas. In order to make the collision response smoother, an additional reaction force was introduced for each colliding point on the cloth (seeFigure 7).

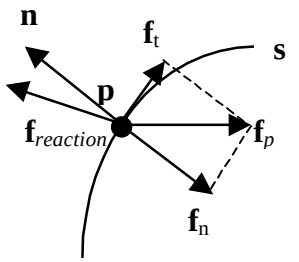

Figure 7: *Introducing reaction forces*

Let $\mathbf{f}_p$ be the force acting on the cloth vertex $\mathbf{p}$. If there is a collision between $\mathbf{p}$ and an object in the scene $\mathbf{s}$, then $\mathbf{f}_p$ is split in its two components: normal ($\mathbf{f}_n$) and tangent ($\mathbf{f}_t$). The object reaction force is computed

$$\mathbf{f}_{reaction} = -C_{fric}\mathbf{f}_t - \mathbf{f}_n, \quad (16)$$

where the first component is due to the friction and depends on the materials.

6. Results

The algorithms were implemented on a SGI workstation (R12000 processor) and on a modern PC (Pentium III, 1 GHz) using the Open Inventor library for rendering the images.

Figure 8: *Tablecloth draping over a moving sphere*

Figure 8 shows the results of a tablecloth draping over a moving sphere. The left image depicts the original mass-spring model with no restriction to elasticity. The cloth is too elastic and the model cannot respond fast enough to the moving sphere. The right image is obtained by applying the elasticity restriction approach with velocity directional modification as described in Section 3. The elasticity threshold is 5%.

In order to compare the performance of our collision detection algorithm with the existing ones, we implemented a classic BB approach using OS interference tests. The body is subdivided into five parts: torso, arms and legs. For each part a quad tree of BBs is constructed with the leaves being the faces on the body with their normal vectors. The algorithm checks for collisions between a vertex of the cloth and a face on the body and computes the response using the technique described in the previous section.

	shirt		dress	
	IS	OS	IS	OS
No of vertices	1800	1800	3840	3840
Time/iteration (ms)	15.8	30.8	33.0	52.0
CD&R time (ms)	7.6	22.5	15.1	34.2
CS time (ms)	8.2	8.3	17.9	17.8
CD&R time (%)	48.1	73.0	45.7	65.7
CS time (%)	51.9	27.0	54.3	34.3

Table 1: *Comparison between a traditional OS collision detection and the IS one for two products*

Table 1 shows the results of applying both techniques to simulate a shirt and a dress on a SGI workstation. The times shown do not include the rendering of the maps for the IS method and the generation of the BB tree for the OS method. CD&R stands for "collision detection and response", and CS means "cloth simulation". As one can see the IS approach improves the speed of the collision detection and response with a factor of 2 to 3. The table compares the results only for a static body since our object space algorithm was implemented only for this case. This means that in dynamics additional time will be spent for computing the velocity of each body point. In our IS algorithm this is done by rendering the velocity map and the graphics hardware interpolates the velocities for the intermediate points. For the OS algorithm, however, these velocities need to be computed at each frame and interpolated when a collision occurs, so we expect that the advantages of our technique will be even more obvious in the dynamic case.

	Skirt	Dress with sleeves
No of vertices	672	2300
Sec/frame	0.244	0.294
Frames/sec	4.1	3.4

Table 2: *Frames per second for animating two different pieces of clothing*

Figure 9 illustrates two different frames of a walking body shown from two different viewpoints. Additional videos are supplied demonstrating walking female body dressed in different pieces of clothing: skirt, trousers, dress and shirt. The code was compiled on a state of the art PC, 1 GHz, 512 MB RAM and a graphics acceleration card GeForce2. Table 2 shows the results for a skirt and a dress with sleeves. The simulation ran at a speed of 3 to 4 frames per second (250 to 300 ms per frame), depending on the complexity of the garment. Note that at each frame, apart from rendering the maps, cloth simulation and collision detection and response, we also need to read the open inventor file, describing the body mesh, and to render the scene graph on the screen. The technique can be speeded up further, if the maps for each frame are pre-computed and stored in advance, so that we do not have to generate them at each frame. This however will require a lot of memory and will restrict it to pre-defined animation.

Figure 9: *Animating dressed people*

7. Conclusions and Future work

An efficient technique for dynamic cloth simulation has been presented. It uses a mass-spring model with a new velocity directional modification method for coping with the super-elasticity of the original model. New image-based method for cloth-body collision detection and response has been presented. The graphics hardware is used not only to produce depth maps but also to interpolate normal vectors and velocities of the body, which are used for collision response. As a result the speed is good enough for an almost real time simulation of garments on walking virtual humans. One further possible step to improve the quality and speed of the collision detection algorithm is to render the cloth at each iteration step and read the collision positions directly from the frame buffer or the stencil buffer. The current system does not implement cloth-cloth collision detection and response. Future work will explore the possibilities of applying an image-based approach to cloth-cloth collisions too.

8. Acknowledgements

This work is a part of the Foresight LINK award project "3D Centre for Electronic Commerce" at the University College London. Many thanks to Laura Dekker and Ioannis Douros for supplying the 3D body models and to Prof. Philip Treleaven and Franco Tecchia for their help and ideas.

References

1. Terzopoulos D, Platt J, Barr A and Fleischer K. Elastically Deformable Models. *Computer Graphics (Proc. SIGGRAPH* 1987*)*; **21** (4): 205-214.

2. Terzopoulos D and Fleischer K. Deformable Models. *Visual Computer* 1988; **4**:306-331.

3. Breen D E, House D H and Wozhny M J. Predicting the drape of woven cloth using interacting particles. *Computer Graphics (Proc. SIGGRAPH* 1994*)*; **28**:23-34.

4. Eberhardt B, Weber A and Strasser W. A fast, flexible, particle-system model for cloth-draping. *IEEE Computer Graphics and Applications* 1996; **16**:52-59.

5. Carignan M, Yang Y, Magnenat-Thalmann N and Thalmann D. Dressing animated synthetic actors with complex deformable clothes. *Computer Graphics (Proc. SIGGRAPH* 1994*)*; **28**:99-104.

6. D. Baraff and A. Witkin, Large Steps in Cloth Simulation, *Computer Graphics (Proc. SIGGRAPH* 1998*)*; 43-54.

7. Ng N H and Grimsdale R L. Computer graphics techniques for modelling cloth. *IEEE Computer Graphics and Applications* 1996; **16**:28-41.

8. Provot X. Deformation constraints in a mass-spring model to describe rigid cloth behaviour. *Proceedings of Graphics Interface* 1995; 141-155.

9. Badler N I, Glassner A S. 3D object modelling. Course note 12, Introduction to Computer Graphics. *SIGGRAPH* 1998; 1-14.

10. Pascal V, Magnenat-Thalmann N. Collision and self-collision detection: efficient and robust solution for highly deformable surfaces. *Sixth Eurographics Workshop on Animation and Simulation* 1995; 55-65.

11. Provot X. Collision and self-collision detection handling in cloth model dedicated to design garments. *Proceedings of Graphics Interface* 1997; 177-189.

12. Shinya M and Forque M. Interference detection through rasterization. *Journal of Visualization and Computer Animation* 1991; **2**:131-134.

13. Rossignac J, Megahed A, Schneider B O. Interactive inspection of solids: cross-section and interferences. *Computer Graphics (SIGGRAPH* 1992*)*; **26(2)**: 353-360.

14. Myszkowski K, Okunev O, Kunii T. Fast collision detection between complex solids using rasterizing graphics hardware. *The Visual Computer* 1995; **11 (9)**: 497-512.

15. Baciu G, Wong W S, Sun H. RECODE: an image-based collision detection algorithm. *The Journal of Visualization and Computer Animation* 1999; **10 (4)**: 181-192.

16. Press W H, Flannery B P, Teukolsky S A and Vetterling W T. *Numerical Recipes in C: the Art of Scientific Computations*. Cambridge University Press 1992.