Efficient Cloth Model and Collisions Detection for Dressing Virtual People

Tzvetomir Ivanov Vassilev, Bernhard Spanlang, Yiorgos Chrysanthou
Dept of Computer Science, University College London,
Gower Street, London WC1E 6BT, United Kingdom
{T.Vassilev|B.Spanlang|Y.Chrysanthou}@cs.ucl.ac.uk

Abstract
This paper describes a fast technique for dressing virtual humans with different pieces of clothing. It exploits a mass-spring cloth model but applies a new velocity modification approach to overcome its super-elasticity. The algorithm for cloth-body collision detection and response is based on image-space interference tests, unlike most of the existing ones, which use object-space checks. The modern workstations' graphics hardware is used not only to compute the depth maps of the body contours but also to interpolate the body normal vectors. As a result the approach is very fast and makes it possible to sew a garment around a virtual human body in about a second.

Keywords: cloth simulation, physically based modelling, image-based collision detection

1 Introduction

The main objective of this work is to develop a fast technique for clothing 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 types of clothes, try them on, and buy them 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 detecting cloth-body collisions. It reads as input a body file and a text file of the garment's cutting patterns. The ideal solution would be to read output text files from the existing clothing CAD/CAM systems such as GERBER, which produce standard DXF or HPGL files describing the cutting pattern geometry and seaming information. Since we did not have such files at hand, for the first version standard drawing packages were used to create example patterns. Their outlines were exported in a text file and the necessary seaming information typed in. In order to create a rectangular topology mesh necessary for the cloth model, the cutting patterns are processed by a meshing subroutine. After the cutting patterns have been produced and meshed, they 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 shirt is 'put on' the human body. Then several more iterations are performed to drape the cloth.
The rest of the paper is organised as follows. Next section reviews previous work on cloth simulation and collision detection. Sections 3 and 4 describe the cloth model and its improvement to cope with super-elasticity. Section 5 describes the collision detection algorithm in detail. Section 6 is dedicated to collision response. Section 7 gives results as images and comparison tables and section 8 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 challenging the cloth simulation using particle [3, 4] or energy [5, 6] 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 the 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 much 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 has modified the positions of many vertices, it may have elongated 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.

2.2 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 hierarchy [6, 11]. Objects are grouped hierarchically according to proximity rules and a bounding box is pre-computed for each object. The collision detection is then performed analysing bounding box 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 for object interference in CAD/CAM systems [13], in dental practise [14], etc., but never for cloth-body collision detection and response.
This paper exploits an image-space approach to collision detection and response. It uses the workstation graphics hardware not only to compute the *z*-buffer depth maps but also to interpolate the normal vectors on the human body. 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 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.
Let $\mathbf{p}_{ij}(t)$, $\mathbf{v}_{ij}(t)$, $\mathbf{a}_{ij}(t)$, where i=1,…, l and j=1,…, n, be correspondingly the positions, velocities, and accelerations of the mass points at time t. The system is governed by the basic Newton's law:

$$\mathbf{f}_{ij} = m\,\mathbf{a}_{ij}, \qquad (1)$$

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 the 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 $T_0 \approx \pi\sqrt{m/K}$. In fact our experiments showed that the numerical solving of equations 3 is stable when

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

where K is the highest stiffness in the system.

4 Constraining super-elasticity

The major drawback of the mass-spring cloth model is that it is very elastic. This means that the deformation rate for some springs is too high and as a result the cloth stretches under its own weight, something that does not normally happen to real cloth.

4.1 Increasing stiffness

One way to avoid the super-elastic effect is to increase the stiffness of the springs. For the same external forces (in our case gravity and damping) the elongation rate should be lower for stiffer springs. This approach, however, has an obvious drawback. Equation 4 shows that increasing the stiffness coefficient K will require decreasing the time step Δt. This means that for the same animation time the number of iterations will be greater and the algorithm will be more time consuming. Another drawback is that even if the stiffness is increased, the resulting material behaves more like rubber rather than cloth.

4.2 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.

4.3 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-given 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.
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 in 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 the algorithm sets them to zeros. If this is applied to all strings, the stretching components of the velocities are removed and in this way further stretching of the cloth is not allowed. As the results show this approach works fine not only for local but also for global deformations.

5 Collision detection and response using depth and normal maps

Collision detection is one of the crucial parts in fast cloth simulation. In each iteration of the simulation a check for collision with the scanned 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. Conventional methods have to search through a dedicated collision check data structure just to find a face on the body that might collide with a vertex of the garment. Even more checks need to be done if collision tests for polygon edges and faces are made. Collision response computations usually are performed for each occurring collision. Our goal was to find a simple, efficient and scalable solution. We decided to implement an image based collision detection method, which was first used by Shinya and Forque [12]. Using this technique it is possible to tell whether a point collides only by comparing the depth value of the garment point with the according depth information of the body stored in depth maps. In our approach collision response information is gathered from the body's normal maps. Depth and normal maps are created by using two projections, one of the front and one of the back of the 3D model. For rendering the maps we place two orthogonal cameras at the center of the front and the back face of the models bounding box. To get accurate depth values the camera's far clipping plane is set to the depth of the bounding box and the near clipping plane is set to zero. Both cameras point at the centre of the bounding box. This is illustrated in Figure 2. Since we currently work with static models, the map generation is only done once at the beginning of the simulation. For animated models we would need to render the maps at each animation step. However, this is a general problem with animation. If we would use an OS approach we would also have to update the structure for each movement of the body model.

5.1 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 by using OpenGL's read-buffer function. The *z*-buffer contains floating-point 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. When reading the *z*-buffer values, they are converted into geometry space using

$$\begin{aligned} back &: depthmap(X,Y) = z_{\max} - zbuffer(X,Y) * bboxdepth + tol, \forall (X,Y) \\ front &: depthmap(X,Y) = z_{\min} + zbuffer(X,Y) * bboxdepth - tol, \forall (X,Y) \end{aligned}, \tag{5}$$

where $z_{\min}$ and $z_{\max}$ are correspondingly the minimum and maximum z values of the bounding box and *tol* is a tolerance value explained in details later. Figure 3 shows example depth maps.

5.2 Generating the normal maps

Before the two renderings for generating depth and normal maps, we substitute the color (*Red*, *Green*, *Blue*) value of each vertex of the model with the coordinates (n_x, n_y, n_z) of its normal vector **n**. After rendering the frame buffer contains normal information of the surface. Since the OpenGL color 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 color fields using

$$\begin{pmatrix} Red \\ Green \\ Blue \end{pmatrix} = 0.5\mathbf{n} + \begin{pmatrix} 0.5 \\ 0.5 \\ 0.5 \end{pmatrix}. \tag{6}$$

By turning on the interpolated shading option we 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 the memory gives us a smooth normal map. Conversion from (*Red*, *Green*, *Blue*) space into the normals space is done by

$$\mathbf{n} = 2\begin{pmatrix} Red \\ Green \\ Blue \end{pmatrix} - \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}. \tag{7}$$

Figure 4 shows example normal maps.

5.3 Testing for collisions

After retrieving the depth and normal 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

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

$$X_{back} = \left(1 - \frac{x + \frac{bboxheight}{2}}{bboxheight}\right) mapsize; \quad X_{front} = \left(\frac{x + \frac{bboxheight}{2}}{bboxheight}\right) mapsize. \tag{8}$$

First the z value is used to decide which maps 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

$$\begin{aligned} back &: z < depthmap(X_{back}, Y) \\ front &: z > depthmap(X_{front}, Y) \end{aligned}, \tag{9}$$

If a collision occurred, we retrieve the normal vector from the normal map indexed by the same coordinates (X, Y) used for the collision check to respond.

5.4 Estimating the error of discretisation

Considering the fact that most of the modern workstation 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

$$\Delta z = \frac{bboxdepth}{2^{24}} < \frac{100}{2^{24}} < 6.10^{-6}\, cm\,. \tag{10}$$

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

$$\Delta x = \Delta y = \frac{bboxheight}{mapsize} \approx \frac{160 \text{ to } 180}{mapsize} cm\,, \tag{11}$$

where we consider the average person to be 160 to 180 *cm* high. This means that we have control over the error in x and y direction varying the size of the maps. The preliminary tests showed that a reasonable value for the error tolerance in equation 5 is *tol*=0.5 *cm*. The need of this tolerance is due to the numerical discretised integration of Newton's equations and it depends on the time step Δt. Since we add this error tolerance to the front and back of the body, but not to the sides, the first version of the collision detection algorithm worked well for the front and back, but not so well for the sides of the body, which was expected. A solution to this problem is to add one layer of pixels around the body in the image maps, which represents the error tolerance for the sides. However, in the world coordinate system this tolerance must be equal to the tolerance (*tol*=0.5 *cm*) for the front and back. So, $\Delta x = \Delta y = 0.5$ and then *mapsize*=320 to 360 pixels.

6 Resolving collisions

After a collision was detected, the algorithm should 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 Moore and Wilhelms [17], which proved to be very fast.

Let **v** be the velocity of the vertex **p** colliding with the surface **s** (Figure 5). The surface normal at the point of collision is denoted by **n**. If $\mathbf{v}_t$ and $\mathbf{v}_n$ are the tangent and normal components of **v**, then the resultant velocity can be computed as:

$$\mathbf{v}_{res} = C_{fric}\mathbf{v}_t - C_{refl}\mathbf{v}_n, \quad (12)$$

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

A similar approach can be implemented to detect and find the responses not only to collisions vertex-body, but also face-body. 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 12, 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.

7 Results

The algorithms were implemented on a SGI workstation (R12000 processor) using the Open Inventor library for rendering the images. Results of dressing a female and a male body with different pieces of clothing are shown on Figure 6 and Figure 7 respectively. The sleeveless shirt, skirt and dress are constructed of two patches with a number of vertices 272, 168, 512 per patch respectively. The trousers are built out of four patches of 150 vertices each. The sleeves on the second shirt are two separate patches of 136 vertices each.

In order to compare the efficiency of our collision detection algorithm with the existing ones, we implemented a classic bounding boxes approach using OS interference tests. The body is subdivided in five parts: torso, two arms and two legs. For each part a quad tree of bounding boxes is constructed with the leaves being the faces on the body with their normal vector. 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 section 0. The next two tables summarise the times measured when conducting performance tests. They do not include the set-up time necessary to read the input files, to generate the depth and the normal maps or to build the bounding boxes hierarchy.

Table 1 shows the results of applying both techniques to sew a shirt and a dress. The IS approach converges much faster, not only because its time per iteration is shorter, but it also produces a smaller number of iterations. These numbers of vertices were chosen, because they were the minimum necessary to produce visually pleasing results for the bounding boxes approach. A smaller number produces 'holes' on the garments. This is due to the fact that we check only for vertex-face collision. Face-face collision tests could also be implemented, but this will slow it down even further.

Table 1. Comparing a traditional OS collision detection to the IS one

Product	CD algorithm	Number of cloth vertices	Number of iterations	Total time in sec	Time per iteration in ms
Shirt	**IS based**	1800	82	1.3	15.8
	OS based	1800	149	4.6	30.8
Dress	**IS based**	3840	130	4.3	33.0
	OS based	3840	150	7.8	52.0

As Table 2 shows, due to the face-body collision detection, implemented in our IS algorithm, it is possible to reduce quite a lot the number of vertices on the cloth and still produce visually pleasing results, as shown in Figure 6 and Figure 7. The table indicates the minimum number of vertices and their corresponding times necessary to dress a human body with four different pieces of clothing: a skirt, long trousers, a sleeveless shirt and a dress.

Table 2. Minimum times necessary to dress a virtual body with the listed products

Product	Number of cloth vertices	Number of iterations	Total time in sec	Time per iteration in ms
Skirt	336	66	0.206	3.1
trousers	560	69	0.334	4.8
Shirt	544	55	0.268	4.8
Dress	1024	88	0.784	8.9

Figure 8 illustrates how computational time changes with increasing the number of vertices on the cloth. As one can see, the dependence is linear which indicates the low $O(n)$ complexity of the algorithm. Because of the nature of the method its speed does not depend on the number of vertices on the body, that is why such performance tests were not conducted.

8 Conclusions and Future work

A fast system for dressing virtual people has been presented in this paper. The body surface is acquired through a 3D scanner, which makes the approach very accurate and allows users to try on different sizes of clothing and see how they fit them.

The system uses a mass-spring cloth model with velocity modification methods for both coping with the super-elasticity of the original model and resolving responses to collisions. The method exploits a new image-space based algorithm for detecting collisions cloth-body, which proved to be more efficient than existing ones. As a result the speed is good enough for a real time simulation of draping garments on 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 and read the collision positions directly from the frame buffer or the stencil buffer. This will also provide full cloth-body interference detection rather than just for selected cloth points as it is now. However, this will also require a modification of the collision response algorithm.

So far we conducted experiments only on a static human body. Our future plans are to implement dynamic simulation of garments on moving virtual actors using the presented approach, evaluate its efficiency and see how it compares to other existing techniques.

9 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.
17. Moore M and Wilhelms J. Collisions detection and response for computer animation. *Computer Graphics (Proc. SIGGRAPH* 1988); **22** (4):289-298.

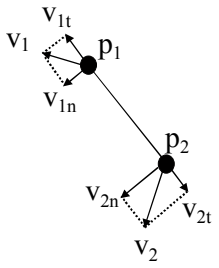

Figure 1. Velocity modification for over-elongated springs

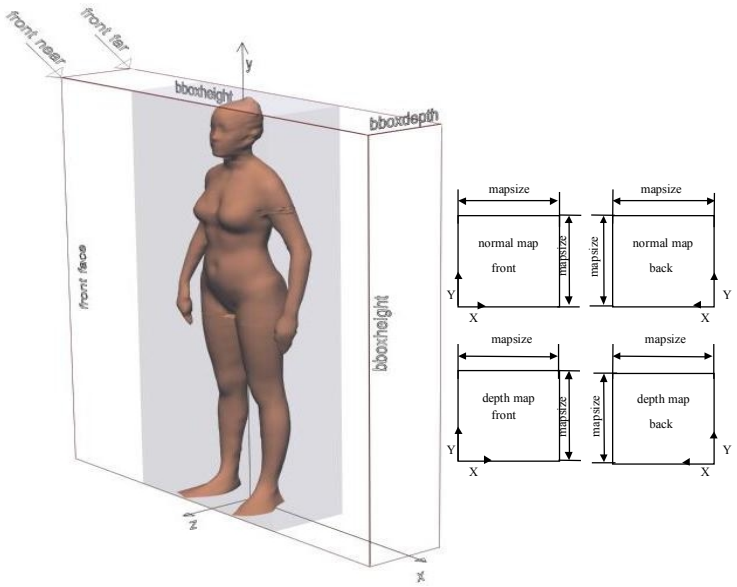

Figure 2. Front camera clipping planes and maps

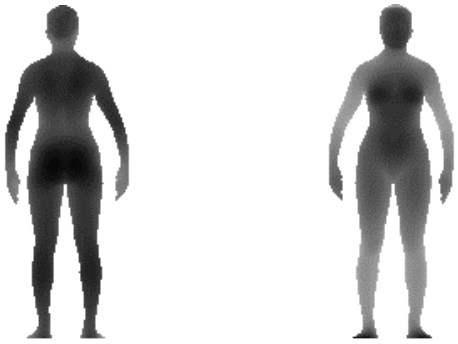

Figure 3. Depth map: back (left) and front (right)

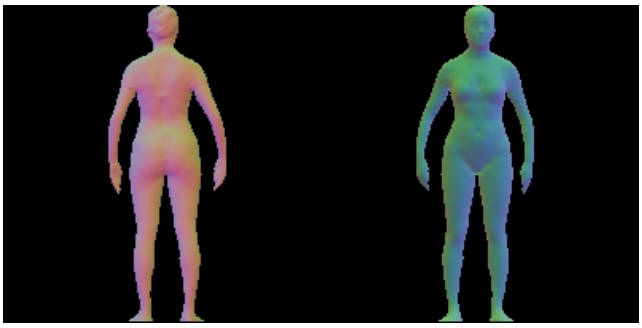

Figure 4. Normal maps: back (left) and front (right)

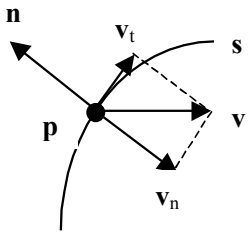

Figure 5. Resolving collisions manipulating velocities

Figure 6. Female body dressed in: dress (left), outfit of shirt and skirt (right)

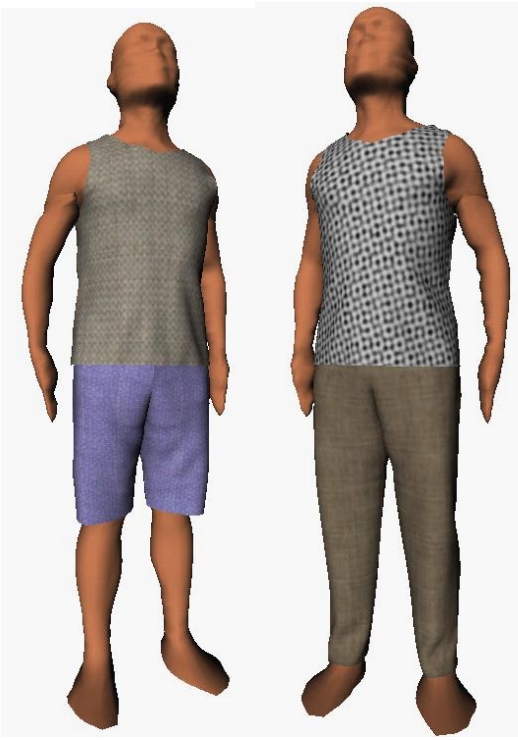

Figure 7. Male body dressed in: vest and shorts (left) and shirt and trousers (right)

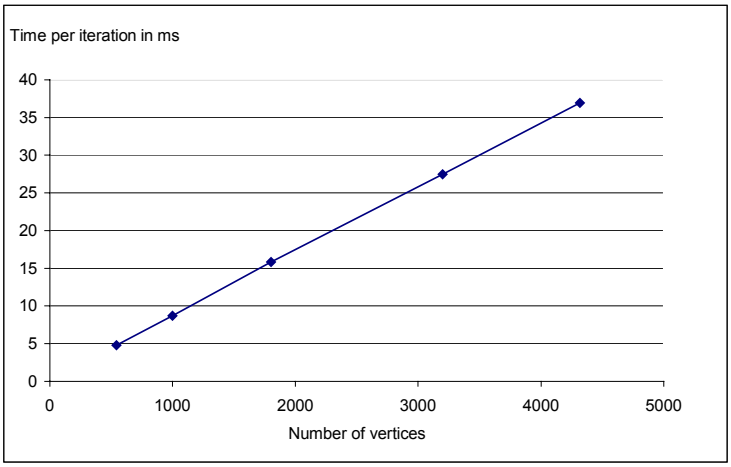

Figure 8. Computing time per iteration versus number of vertices on the shirt