# GPU Based cloth simulation with Moving Humanoids

J. Rodriguez-Navarro
LABSID
Dept. Applied Mathematics
Universitat Politècnica de Catalunya
Barcelona-España
javier.rodriguez-navarro@upc.edu

M. Sainz
Developers technical group
Nvidia Corporation
msainz@nvidia.com

A. Susin
LABSID
Dept. Applied Mathematics
Universitat Politècnica de Catalunya
Barcelona-España
toni.susin@upc.es

## Abstract

In this paper we present a new real-time cloth simulation. The usual bottleneck in cloth simulation is collision detection, which becomes more difficult to solve, with good frame rates, when a complex geometry, like a human body, is involved. Recent collision image based methods, that use depth images to detect collisions, usually relays on CPU for collision correction. In our case we implement a GPU based simulation that takes care both of cloth simulation and body-cloth collisions. Our solution is based on a hierarchic depth map structure. A high frame rate is obtained with both structured and unstructured cloth meshes with thousands of particles.

## 1 Introduction

Cloth simulation can be considered as a particular case included at the more general framework of deformable models. A deformable model simulation requires the update of the model vertex positions at each time step according to some deformation law. This can be considered like a new object representation at each time step and was not possible take advantage of the old graphic cards for deformable simulations. Nowadays graphic cards incorporate programable capacity for their GPU (Graphics Processor Unity) and allows to read vertex positions from a texture using Vertex Texture Fetch (VTF) in a vertex program. As we will show, this can be used for achieving a high frame rate simulation on deformable models. Moreover, we have also introduced a new technic, the *quasi-feedback* method, that give us a criteria to ask about the status of the system during simulation on the GPU. This was only possible so far reading back to CPU with the corresponding decrease in the performance.

More precisely, we are facing the problem of cloth simulation on the GPU and the interaction between cloth and solid objects, essentially a human body. Collision detection is usually one of the bottleneck for cloth simulation because a huge number of computations are involved. When using the graphics card pipeline a new image based tool is available for collision detection. Depth map from appropriate point of views allow as to detect when a cloth particle is inside another object and a collision correction can be activated.

In order to have more versatility we have used both structured rectangular and unstructured triangle meshes for the cloth. In case of working only with structured rectangular meshes, the frame rate can be increased due to the known neighbor relationship which takes advantage of the internal texture codification. Unstructured triangle meshes are more general but, as we will explain, it will require an extra texture for taking account of vertex connectivity.

The paper is organized as follows, section 2 is devoted to previous work in the area, section 3 describes the cloth simulation model and its GPU implementation, section 4 is devoted to the collision problem and our solution using a hierarchic depth map. Results and conclusions are shown on last section.

## 2 Previous work on cloth simulation

Cloth simulation is a well known and widely studied computer graphics problem. We can roughly classify previous work according to the emphasis on modelling, simulation and computational efficiency. A good reference survey for the field is [14].

Early works are essentially devoted to the modelling aspects. Papers by Terzopoulos et al. [17], [18] where the first physically-based ones. They introduce cloth simulation for graphics community as a problem of deformable surfaces and used techniques from mechanical engineering like the finite element method and energy minimization. Other approaches on the dynamic modelling has been, the particle-based models from the works of Breen et al. [3] and Eberhardt et al. [6], the energy-based models from Carignan et al. [4] and Baraff and Witkin [2]. One of the most successful approach in modelling has been the mass-spring one, introduced by Provot [16].

One of the first, present and more successful work in cloth simulation is devoted to the MiraLab team leaded by N. Magnenat-Thalmann. Their contributions start in the early 80's and have reached many important results in all the above areas classification [13].

Like in other graphics topics, the increase in applications efficiency is related both with the chosen model and the hardware performance. Moreover, a trade-off between speed and precision has to be assumed, the work of Hauth and Etzmuss [9], and the one of Volino and Magnenat-Thalmann [20] discuss the convenience of numerical integrators.

The final decision for choosing one simulation model or another is mainly related with the final application field, for video-games or films the requirements are totally different. The work of Jacobsen [10] for the game industry has been the pioneer in introducing a dynamic model for the cloth together with the numerical integrator, the Verlet method, which is suitable for GPU implementations. The first GPU cloth implementation is due to S. Green [8], a structured rectangular mesh cloth and a solid sphere are simulated using Verlet method on the GPU. Only stretch forces has been simulated.

For dealing with collisions on the GPU image based methods are introduced by Vassilev et al. [19] for walking humanoids without dealing with occlusions. Other recent papers by Kolb et al. [11], [12] studies collisions with complex but static objects.

We use ideas from these papers to introduce our method for dealing with cloth colliding with a complex object like the human body in motion. We also solve the occlusion problem using many views from cameras positioned in fixed local coordinates. From the skinning correction applied during animation we simulate the human body as a non-rigid object and therefore, more precise collision detection is obtained.

## 3 Cloth Simulation model

As we pointed out in the previous section, there are many different physically-based methods for animation of cloths. In our case we are aiming a realistic result more than a real engineering simulation of fiber cloths. Moreover, we want to use the graphics hardware in order to obtain really fast simulations. There are other approaches not relayed on the GPU that obtain fast simulation [5] at the price of decomposing the cloth mesh in several parts and animate only some of them. They have obtained nice results but we are able to obtain similar ones without fixing any part of the cloth which makes our method more flexible and applicable to general garments without restrictions.

Our cloth simulation model is based on [10], from game industry, and the GPU implementation presented in [8]. We extend these previous works to a general triangulated mesh solving the difficulties arising when a different number of neighbors has to be considered for each vertex, and also to collisions with general complex objects in motion.

The usual Newton's dynamic equations are integrated using a stable Verlet numerical scheme which can be stated for each $i$-particle as

$$\mathbf{x}_{n+1}^{i} = \mathbf{x}_{n}^{i} + \varepsilon(\mathbf{x}_{n}^{i} - \mathbf{x}_{n-1}^{i}) + \mathbf{a}^{i}\Delta t^{2}. \qquad (1)$$

Here $\mathbf{x}_n^i$ stands for the present particle position and $\mathbf{a}^i$ its acceleration (essentially gravity). $0 < \varepsilon \leq 1$ is a drag term usually used to stabilize numerically the system. The simulation time step $\Delta t$ is the one used for advance in time.

Indeed only external forces are taken into account when computing the acceleration term. Internal forces are included in the simulation as a restriction of the motion [21], nowadays this approach is one of the best choices for GPU implementation.

The cloth model is based on a particle system where each vertex is a mass particle and the plane deformation properties of stretch, shear and bending are simulated as a constraint on the initial length of the edges of the mesh [16]. For structured meshes it is equivalent to consider the usual scheme shown in figure 1. One can only use stretch relations like in [8] but results are in general too elastic.

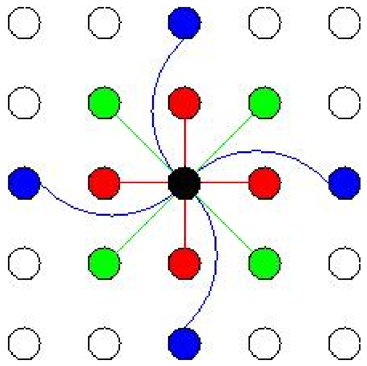

Figure 1: Planar deformation relations for structured meshes with stretch (red), shear (green) and bend (blue) springs.

For unstructured triangular meshes we simulate internal forces using connections between a vertex and all neighbor particles. For that, we have an auxiliary texture, *connectivity texture* used as a look-up table of connections and rest lengths.

Length constraints are satisfied using a relaxation iterative procedure. At each time step and for each of the mesh edges, if the present length is different than the original (or the rest) one, the two particles on this edge are displaced a percentage (stiffness constant) of this difference in the edge direction. Usually the stiffness constant, $k_s$, is 0.5. Thus, if we denote by $d_{ij} = \| \mathbf{x}_n^i - \mathbf{x}_n^j \|$ and we assume, for instance, that $d_{ij} > l_{ij}$ being $l_{ij}$ the rest length between particles $i$ and $j$, then the corrected positions will be

$$\begin{aligned} \mathbf{x}_n^i &= \mathbf{x}_n^i - k_s * (d_{ij} - l_{ij}) * \mathbf{v}_{ij}, \\ \mathbf{x}_n^j &= \mathbf{x}_n^j + k_s * (d_{ij} - l_{ij}) * \mathbf{v}_{ij}. \end{aligned} \tag{2}$$

Here, $\mathbf{v}_{ij}$ stands for the unit vector at the edge direction from particle $\mathbf{x}_n^j$ to particle $\mathbf{x}_n^i$.

For a direct solution concerning all particles proper correction, a global linear system must be builded taken into account all the connection edges. This can be done in a quite fast way [1] but it is not the best choice again for GPU implementation. Instead of building a huge matrix for solving the position correction problem one can use an iterative method more appropriated for sparse systems like the one involved here. Sparsity is inherent to cloth simulation because each particle is connected with few neighbor vertex compared to the total number of particles in the mesh. As we will show in the next section, the GPU implementation of deformation restrictions is equivalent to solve this linear system using a Gauss-Seidel iterative scheme (see [15]) without need of an explicit global matrix. After several iterations of the same procedure, the system will converge to the rest length for all edges.

The number of iterations were fixed in all previous implementations of such an iterative process in GPU so far. We have implemented a new technic, the *quasi-feedback* method, to stop an iterative process saving a lot of useless iterations. This method consists in a mechanism to obtain a value from the GPU that reports of the system status. In our case we want to stop iterations when the deformation of the cloth is not significant. For that we render the cloth discarding the points whose displacement are less than a suitable threshold, we call the `ARB_occlusion_query` extension and it gives the number of samples that are less than this threshold. We stop the iterative process when no correction is needed. A full description of this technic is beyond the scope of this paper and it will be published somewhere. The convergence is almost guarantied unless huge accelerations are acting in the simulation. Using this relaxation procedure unrealistic cloth enlargements (see figure 2) are avoided.

Cloth particles are stored in a texture 2D and the topology do not change in any step simulation. We resort to `NV_vertex_program3` extension which allows to fetch vertex from a texture in the vertex program. So we remove the read back to CPU obtaining a big performance increase. In the same vertex program we compute the normal of each vertex, averaging neighbor normal faces as usual, for lighting the cloth.

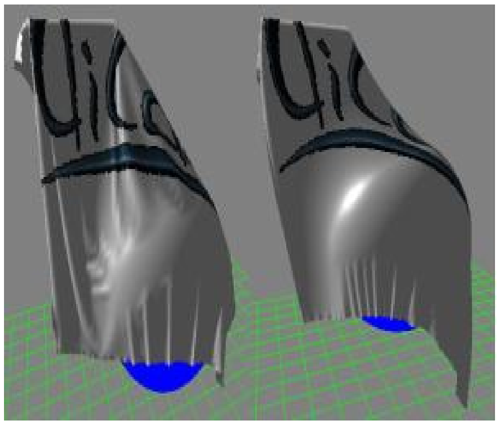

Figure 2: Unrealistic cloth enlargements (right) according to the number of iterations in the motion restriction procedure. Left image with 80 iterations, right with 10 iterations.

### 3.1 Simulation on the GPU

We will explain next the details for the GPU implementation of our simulator. The way we solve the collision problem is postponed to the next section.

Our goal is to run the complete dynamical simulation on the GPU and we use CG language for this propose. In [8] a structured mesh is used for modelling a piece of rectangular cloth. Taking advantage of this simple neighbor structure only three textures are used for storing the past, present and new mesh positions.

In order to extend our simulations to a wider range of cloth meshes, we consider more general unstructured meshes using both triangular and rectangular elements. These are the kind of meshes you can usually obtain from a modeler software tool. Thus, we are forced to take into account the different number of neighbors for each vertex and stored it into a connectivity texture used for address proposes.

We can not use dynamic allocation for textures and this forces to choose a different storing strategy. Instead of forcing a new connectivity texture to take into account the different number of neighbors for each vertex (*skyline storage*), we choose to store the neighbor indexes for each vertex using a fixed amount of positions (*maximum size band storage*). For each mesh, we define the parameter $N_{max}$ by the maximum number of neighbors that has any vertex in the mesh and we use this fixed number of texture elements for storing the indexes of the present neighbors. Thus, the size of the needed texture can be determined when the cloth mesh is loaded. We have to establish a trade-off between the way to store the associated indexes and maximum size of available textures. By one hand, we have stored in a texture the positions of the simulated cloth mesh, called *positions texture* (texPos). To get a position which is located in the index $(i, j)$ of the position texture, it is enough to call the CG function `f4tex2D(texPos, float2(i,j))`. On the other hand, we know the neighbors of a fixed vertex , $p_{ij}$, located at index $(i, j)$ in the *texPos*. As a special case, if we want to simulate 1-dimensional deformable objects, the connectivity becomes 2, then we can generate a connectivity texture RGBA where we can store at the index $(i, j)$ the index of the neighbors of this vertex. If $N_{max} \leq 4$, we can make an analog procedure, in this case we put the index of neighbors in each coordinate as follows: if we suppose that the indexes in texPos are $(i_k, j_k)$, $k = 1, \ldots, 4$ and the size of texPos is $(w, h)$, then the value of the coordinate $k$ in the index $(i, j)$ of index texture is $i_k * w + j_k$. The following cg-sentences allow to recover the index of texPos,

```
float  idx1D =
       sampler2D(texConn, T.xy);
float2 idx2D ;
idx2D.x = frac(idx1D/h)*w;
idx2D.y = idx1D/h;
```

It may also be possible to eliminate `frac` instruction by using the repeating-tiled addressing mode (`GL_REPEAT`). This reduces the conversion to a single assembly instruction, otherwise this may not work on some texture configuration ([7]).

In our case, we have connectivity $N_{max} = 16$. We create a connectivity texture RGBA of size $(w * 16, h)$. The indexes of the neighbors of $p_{ij}$ are represented by $(i*16, j)$, $(i*16+1, j), \ldots, (i*16+15, j)$. As the number of connection is not constant, but less than 16, we first initialize the connectivity texture to $-1$ marking when no more neighbors exists. We put at the coordinates *rg* the index of the neighbors and reserve the coordinate *b* to put the rest length of the associated mesh edge. At the coordinate *a* we store the diagonal mesh distance

$\sqrt{2} * b$.

We want to remark that, because we are using texture 2d, we have previously normalized the indexes to $[0,1]$.

Onces the position was stored in the corresponding texture, one pass through a Verlet fragment program updates to a new position. After that we iterate, using a constraint fragment program, a convenient number of times to ensure the correct final length between particles. And finally, as it will be explained bellow, collision detection has to be checked.

## 4 Dealing with Collisions

In computer animations, in general, there is always a bottleneck: the collision problem. One has to take into account interaction between different objects in a scene and this can be a very tedious problem. The shape of the objects is one of the most important features in the collision detection. The first CG cloth implementation [8] uses only a sphere as a solid body for cloth collision and this makes the collision detection easy. Only a distance computation between the center of the sphere and the cloth particles is needed. This can be computed on the GPU at really high frame rates. In general, for basic geometry primitives like spheres or aligned boxes one can use simple distance functions and solve this problem easily.

For dealing with a more complex object, like a human figure, an image-based hardware accelerated approach is one of the best choices. We follow a depth map based approach similar to [11], [12] but we extended it to collision with moving objects. Moreover, using present depth information, we are not forced to consider the human body as a set of rigid bodies, but we can consider it as a deformable surface, taken into account the skinning correction at the vertex near the joints and also consider muscle shape deformations.

For moving models, the work of [19] is a first step, but they use only two depth maps, front and back, for the whole figure. Therefore, when some occlusions arises, for instance when the arm passes in front of the chest, no possible information about collisions can be extracted from these depth maps. To overcome this problem we consider several detailed views, in fact, we use a front and back view for each of the anatomical sets of our humanoid. Based on the hierarchic structure of the humanoid, already used for animation purposes, we place two fixed cameras in local coordinates (front and back) for each anatomical set, see figure 3. During motion, the position of the cameras will be always the same with respect to the associated set and the occlusion problem can be solved. In our approach we consider 14 anatomical sets, we choose head, torso, and 3 segments for each arm and leg. This give us a maximum total amount of 28 different depth maps (see figure 3) that will be processed two by two as we explain in what follows.

### 4.1 GPU collision detection

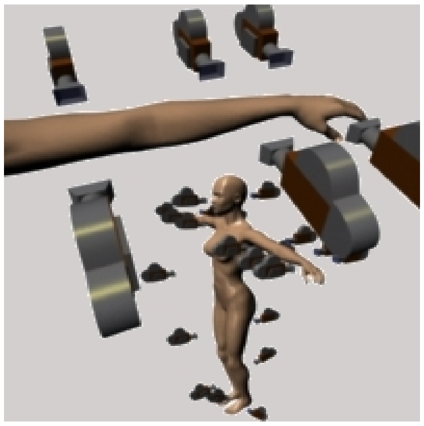

Figure 3: Superposed image of virtual camera positions associated to human body segments. A closeup for the left arm.

For each particle cloth we should check when there are interpenetration with the human body. Since it is divided in different parts, we associate at the beginning two orthographic camera, front (FC) and back (BC) cameras to each part. The rotation and translation which are applied to the part of body at the present frame is also applied to the FC and BC, so we keep the point of view and handle each part independently to remove possible occlusions with other different body parts. Then, in each time step, to check the interpenetration of a cloth particle into the deformable body, we express the particle coordinates in the corresponding local coordinate system for each part of the body. We have to verify that the particle belongs to the inner part of the camera volume of viewing. Finally, we detect an interpenetration when the depth of the particle is less than the depth of the back and front part. This process is called **depth-test**, see the algorithm below.

**Algorithm 4.1** Depth test

```
for each body part P do
    pLocal1 = mul(P.MVPfront, p);
    pLocal2 = mul(P.MVPback, p);
    if P.OBB contains pLocal then
        float4 d1 = tex2D(front, pLocal1.xy);
        float4 d2 = tex2D(back, pLocal2.xy);
        if pLocal1.z < d1.a && pLocal2.z < d2.a then
            Interpenetration.
        end if
    end if
end for
```

To improve the performance we use only one texture RGBA to save the normal map and the depth map associated to each anatomic body set. For this purposes, we use a frame buffer object to render it offscreen. Pbuffer could be another option to do this, but neither it does not allow render to texture directly with Vertex Array Range (VAR) extension or Vertex Buffer Object (VBO) extension, nor fetch the texture in the vertex program. In both cases the performance of the simulation decreases considerably.

In the following vertex program we compute the depth value and save, at the coordinate texture, the values of the normal vector and the depth value. No color coordinates are used to prevent interpolation on the values.

```
vf30 vp_depthnorm(
        float4 P:POSITION,
        float3 N:NORMAL,
        uniform float4x4 MVP)
{
 vf30 Out;
 Out.HPOS = mul(MVP, P);
 Out.TEX0.xyz = N;
 Out.TEX0.w   = 0.5*Out.HPOS.z+0.5;
 return Out;
}
```

The vertex progam is followed by a fragment program that transforms the texture coordinate in color coordinate.

Furthermore, the way we have started needs to store 28 (two for every body segment) depth maps, however the fragment program nowadays accepts only 16 texture as a maximum. To solve this problem we store for each anatomical set, the front and back depth map in the same texture reducing this number to 14 textures. Another possibility could be to handle the textures one by one, i.e., generate the normal and depth map of one body part and check the interpenetration, but in this case the application performance decreases considerably.

As it could be notice, we verify each particle collision with all parts of the human body. If the cloth is really in contact with too many parts, this approach could be very efficient. However, if the cloth has only collision with a small number of parts, this is not suitable. In this case, we only select parts of the body such that the cloth has interpenetrated in theirs oriented bounding box OBB. We remark that the volume of viewing of the camera associate to each body part fits in the OBB of its part. So when we display the cloth from FC or BC, if there are not interpenetrations between the cloth and the OBB of its associated part, the number of drawn samples are zero. To know the number of drawn samples we call the function `glGetQueryObjectuivARB` of `ARB_occlusion_query` extension. One can improve the performance of this method using an approach equivalent to a multiresolution strategy. It is not necessary render the completed cloth mesh, we can maintain two list of index positions, one of them, the complete list, and the other, a reduced list, containing an up third level grouping one of every eight particles. The reduction of the list depends on the dimension of the particle system. In our case, we have reduced the list until no gain in latency and no loss accuracy have been found.

### 4.2 Collision correction

When the front and back checks return penetration a collision correction is needed. Here we found a new problem, the particle position has to be changed and for that, the normal direction to the body at the collision point has to be known. We solved this problem in a similar way of [11], [12] based again in the graphics hardware. The idea is to use an implicit model representation of the body skin, which consists essentially in storing together depth and body's normal information as it has been explained above. When we render the corresponding human subset from the point of view of the local cameras. At the beginning we have stored a normal for each vertex of the human body. During animation we have to recompute every normal according to the corresponding motion matrix and skinning weight, see figure 4. The vertex program `vp_depthnorm` must be modified conveniently to introduce this features. For the sake of clarity we do not have incorporate all the details of this shader.

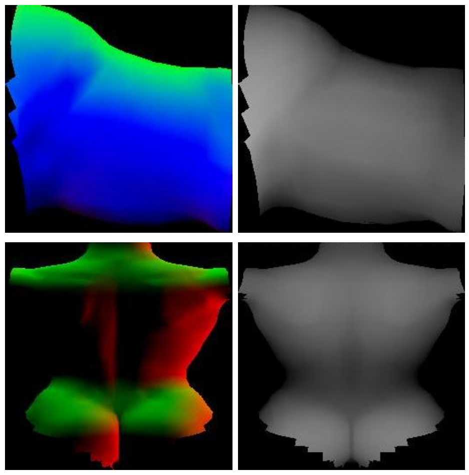

Figure 4: Two Normal-Depth textures corresponding to Front-Arm view (up) and Back-Torso view (down).

When a cloth particle $p_c$ is detected inside an anatomical set, two points of the corresponding depth maps are associated to this particle, the front $p_f$ and back $p_b$ points. The amount of real penetration $RP$ is computed from the scaled differences

$$RP_f = z_s(d1.a - pLocal1.z)$$

and

$$RP_b = z_s(d2.a - pLocal2.z), \qquad (3)$$

respectively. The variables $pLocal1$ , $pLocal2$, $d1$ and $d2$ have been defined in algorithm 4.1 and $z_s$ is the scale parameter passing from camera distance to real world distance. These quantities are both positives in case of penetration and the stored normal at the closest depth point (see figure 5) is used to correct its position outside of the body.

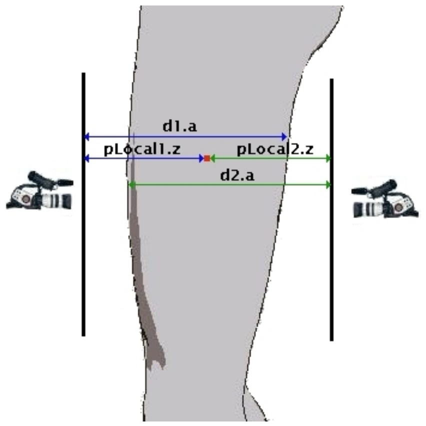

Figure 5: Image based collision correction using front and back cameras for the left upper-leg segment.

If the distance between $RP_f$, $RP_b$ is less than an advisable tolerance an average normal is taken. This, according to the shape of body segments, can happens only in the boundaries of the depth maps. Time step simulation prevents from big penetrations and therefore, the average normal is usually a good approximation of the actual normal of the body surface.

## 5 Experiments and Results

The simulation example consists in a cloth falling on a walking woman, see figure 6. We have chosen this simulation because the cloth is not fixed to any part of the body and occlusions between different body parts are present, for instance, forearm and torso. The mesh used for the woman is composed of 17523 vertex, 17539 quadrangles and 2789 triangles, which is stored in a vertex buffer object, and the motion is read from a standard biovision format file. Skinning weights values were calculated only once according to the animation and stored in a skinning file. The maximum number of iterations in the relaxation algorithm is 40, but using the quasi-feedback method, at the beginning when there are a little deformation, only 2-5 iterations are needed.

On table 1 we present the results for different cloth dimensions. The values of the table give the number of frames per second for each render, including simulation. First column represents

|  | fixed iter(20/40) | quasi-feedback |
|---|---|---|
| $64 \times 64$ | 105/86 | 88-124 |
| $128 \times 128$ | 63/43 | 44-90 |
| $256 \times 256$ | 22/14 | 14-42 |

Table 1: Frame rate results for different cloth dimensions considering all body parts.

|  | fixed iter(20/40) | quasi-feedback |
|---|---|---|
| $64 \times 64$ | 90-144/77-108 | 107-200 |
| $128 \times 128$ | 54-77/41-50 | 40-133 |
| $256 \times 256$ | 20-24/14-15 | 14-55 |

Table 2: Frame rate results for different cloth dimensions considering only body parts which can collide.

the simulation times with a fixed number of iterations, 20/40. And the second one gives a minimum and maximum frame rate when the quasi-feedback method is activated.

On table 2 the same results are shown when the oriented bounding box are used as a previous filter to select the possible body parts for collision. In this case, first column show also the minimum and maximum frame rate achieved during animation.

These results are obtained using a desktop computer with a Pentium IV 3.0 Ghz processor, and a nVidia Geforce 6800 GT graphics card.

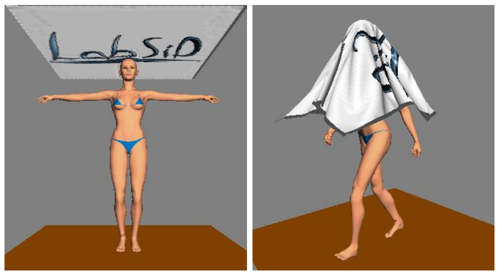

Figure 6: Two screenshots corresponding to the final simulation of the cloth falling on the walking woman.

## 6 Conclusions and Future Work

We have presented a new method for simulating virtual cloth for real-time animations. Our method is running completely on the GPU using the last features of the present graphics cards. Collision correction bottleneck is overcome by means of image based depth-map methods. A high increase of the simulation speed is due to cut off the unnecessary computations. A new technic for getting a feedback from the GPU that allows interrupt an iterative process has been introduced.

For cloth simulation we use structurated meshes in our results, although we can also simulate a cloth defined by an unstructurated mesh. In this case, a small decrease of the frame rate is appreciable due to an extra call to the connectivity texture. However, for unstructured meshes the internal deformation forces can not be simulated using mass-spring connections, if you want to avoid too large iterations, and therefore, loosing real-time performance. When a small number of iterations are used for general cloth meshes, an unrealistic elongation of the cloth is produced. We are working in other different models to solve this problem on the GPU. This will allow us to dress a virtual humanoid using any cloth generated previously with a modeler software. With our present model we are limited to garments formed only by rectangle meshes.

## 7 Acknowledgments

We would like to thank Prof. Renato Pajarola, currently at Univ. of Zürich, for his support at the beginning of our research at the Univ. of California, Irvine. This work has been supported by a collaboration grant CCEGP-03/04 and CICYT grant TIN-2004-08065-C02-01.

## References

[1] Baraff D., Linear-Time Dynamics Using Lagrange Multipliers. Computer Graphics Proceedings, Annual Conference Series, 96: 137–146, 1996.

[2] Baraff, D. and Witkin, A. (1998). Large steps in cloth simulation. Computer Graphics Proceedings, Annual Conference Series, 98: 3–54, 1998.

[3] Breen, D., House, D. and Wozny, M. . Predicting the drape of woven cloth using interacting particles. Computer Graphics Proceedings, Annual Conference Series, 94: 365–372, 1994.

[4] Carignan, M., Yang, Y., Thalmann, N. M., and Thalmann, D. Dressing animated synthetic actors with complex deformable clothes. Computer Graphics Proceedings, Annual Conference Series, 92: 99–104, 1992.

[5] Cordier F., Seo H., and Magnenat-Thalmann N., Made-to-Measure Technologies for an Online Clothing Store. IEEE Computer graphics and applications, 38–48, January 2002.

[6] Eberhardt B., Weber A. and Strasser W. A fast, flexible, particle-system model for cloth draping. IEEE Computer Graphics and Applications, 16: 52–59, 1996.

[7] Lefohn A., Kniss J. and Owens J. Implementing Efficient Parrallel Data Structures on GPUs. In GPU gems 2: Programming Techniques for High-Performance Graphics and General-Purposes Computation. Edited by Matt Pharr, pp. 512–545. Addison Wesley, 2005.

[8] Green S., Nvidia developers 2003. http://developer.nvidia.com/object/demo_cloth_simulation.html .

[9] Hauth M. and Etzmuss O. A high performance solver for the animation of deformable objects using advanced numericla methods. Computer Graphics Forum. Vol. 20 num. 3, 1–10, 2001.

[10] Jacobsen T., Advanced Character Physics. Proc. Game Developers Conference'01, 1–10, 2001.

[11] Kolb A., John L., Volumetric model repair for virtual reality applications. In Eurographics short presentations 2001, Univ. of Manchester, 249–256, 2001.

[12] Kolb A., Latta L. and Rezk-Salama C., Hardware-based simulation and collision detection for large particle systems. In proc. Graphics Hardware'04, T. Akenine-M$\ddot{o}$ller, M. McCool Ed., 1–9, 2004.

[13] MiraLab, University of Geneve. http://miralabwww.unige.ch/ .

[14] Ng H.N., Grimsdale R.L., Computer graphics techniques for modeling cloth. IEEE Computer Graphics and Applications 16: 28–41, 1996.

[15] Press W.H., Flannery B.P., Teukolsky S.A., and Vetterling W.T. Numerical Recipes. Cambridge University Press, 1986.

[16] Provot, X. Deformation constraints in a mass-spring model to describe rigid cloth behaviour. In Proceedings of Graphics Interface 95, 141–155, 1995.

[17] Terzopoulos D., Platt J.C. and Barr A.H. Elastically deformable models. Computer Graphics (Proc. SIGGRAPH), 21: 205–214, 1987.

[18] Terzopoulos and K. Fleischer. Deformable models. Visual Computer, 4:306–331, 1988.

[19] Vassilev, T., Spanlang, B., Chrysanthou, Y. Fast Cloth Animation on Walking Avatars. Computer Graphics Forum, vol 20 num. 3, 1–8, 2001.

[20] Volino P. and Magnenat-Thalmann N., Comparing Efficiency of Integration Methods for Cloth Animation, Proceedings of Computer Graphics International (CGI), IEEE Press, 265-274, 2001.

[21] Witkin A., Baraff D. and Kass M., Physically-based Modeling. SIGGRAPH Course notes. 2001.