Workshop on Virtual Reality Interaction and Physical Simulation VRIPHYS (2011)
J. Bender, K. Erleben, and E. Galin (Editors)

Adding Physics to Animated Characters with Oriented Particles

Matthias Müller and Nuttapong Chentanez

NVIDIA PhysX Research

Abstract

We present a method to enhance the realism of animated characters by adding physically based secondary motion to deformable parts such as cloth, skin or hair. To this end, we extend the oriented particles approach to incorporate animation information. In addition, we introduce techniques to increase the stability of the original method in order to make it suitable for the fast and sudden motions that typically occur in computer games. We also propose a method for the semi-automatic creation of particle representations from arbitrary visual meshes. This way, our technique allows us to simulate complex geometry such as hair, thick cloth with ornaments and multi-layered clothing, all interacting with each other and the animated character.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.5]: Computational Geometry and Object Modeling, Physically Based Modeling—Computer Graphics [I.3.7]: Three-Dimensional Graphics and Realism, Animation—Simulation and Modeling [I.6.8]: Type of Simulation, Animation—

Keywords: natural phenomena, physically based animation

1. Introduction

An appealing and popular way to enhance the realism of computer games is to augment their animated world with physical effects. The major challenges with introducing physics to games are to make the simulations unconditionally stable, controllable and fast. This is why rigid body simulation is still the most prominent physical effect in today's games and the main part of most physics engines. Physics based water simulations have been introduced more recently. For reasons of stability and speed, these are mainly restricted to height fields driven by the wave equation and particle effects. For the same reasons, cloth simulations have most often been used to animate non-interacting objects such as flags or curtains.

However, the most prominent elements in a game are the characters. This is particulary the case for third person games where the player's avatar is permanently in the focus of the camera. Traditionally, characters have been purely animated entities because even small failures such as local cloth entanglements are immediately visible and destroy the illusion a realistic world. On the other hand, adding physically based secondary motion to characters is one of the most effective ways to enhance the realism of a game with physics.

In this paper we present a new stable and fast method to add physics to animated characters suitable for using in games. Our method is based on the recently introduced approach of Müller et al. [MC11] for the simulation of solids. They represent a deformable object with a set of oriented particles connected by an arbitrary connectivity structure. This representation allows the skinning of a visual mesh of independent resolution and geometry to the physical mesh. It also provides an accurate collision volume even for a moderate number of particles making collision handling fast.

In order to be used in connection with animated characters, we had to extend the the basic oriented particle method (OP) in various ways. The main extensions and contributions of this work are:

1. A way to incorporate animation information into the OP method. The simulation can be set to follow the animation strictly or to be influenced by it in a momentum conserving way.
2. A method to create regular particle meshes stably from

DOI: 10.2312/PE/vriphys/vriphys11/083-091

arbitrary visual meshes for the efficient simulation of complex geometry such as multi-layered clothing.

3. Two techniques to enhance the stability of the original OP method: continuous collision handling for particles and
4. the concept of surface particles.

2. Related Work

Our method applies deformable simulation to animated characters. Stand alone deformable body simulation has been popular since its introduction to the computer graphics community by Terzopoulos et al.. [TPBF87]. A detailed survey of the field which includes finite element, finite volume, mass springs, particles and grid based methods can be found in [NMK*05]. Our paper builds upon the oriented particle method [MC11], which is an extension of position based dynamics [MHR06]. We chose this method because it is unconditionally stable, easy to tune and fast enough for being used in computer games.

An important use case for our method is the simulation of multi-layered clothing and hair on animated characters. Cloth and hair simulations have received special attention from the community since they have a wide range of applications and are considerably different from the general deformable solid. Excellent surveys on both areas can be found in [JWL08] and [WBK*07] respectively. With only a few exceptions, researchers in computer graphics have used the visual mesh as simulation mesh for cloth simulation. In contrast, our method allows to attach multiple visual layers of cloth to a single simulation mesh thereby avoiding expensive collision detection between the layers.

The separate simulation of multiple layers of clothing has been explored in the context of collision detection and response for deformable objects as surveyed in [TKH*05]. Simulating multiple layers of cloth leads to the more general problem of cloth self-collision detection, which is still an open research problem with a large body of work. We only list a few examples here. To accelerate cloth collision detection, Curtis et al. [CTM08] proposed the use of representative triangles to reduce the number of triangle pairs to be tested for collision. Harmon et al. [HVS*09] handle multiple contacts asynchronously to resolve collision of layers of cloth correctly. Otaduy et al. [OTSG09] formulate the contact force computation among layers of cloth and deformable bodies as a linear complementarity problem which they show how to solve efficiently. Recently, Schvartzman et al. [SPO10] proposed a hierarchical method to accelerate cloth self-collision detection by exploring the regularity of local surface patches.

Perez et al. [PuG99] devised a specialized algorithm for the robust simulation of multi-layered garments. They pull each layer of clothing toward a specific iso-surface defined by blobs around animated skeleton. Another robust way to simulate layers of cloth was proposed by [WBH04]. They compute the positions of a subset of the cloth vertices in a given layer using barycentric interpolation plus some fixed normal offset from the triangles in the previous layer.

A second use case for our method is the simulation of the skin of an animated character as a deformable layer. The most popular way to attach a triangle mesh to an animated skeleton is linear blend skinning [MTLT88]. Here, each vertex is transformed by a weighted average of the bones transformations. This simple approach works well in most cases but produces artifacts near highly bent joints. Several new methods have been proposed to alleviate this problem. Wang and Phillips [WP02] extend the scalar weights to matrices and learn them automatically from training animations. Merry et al. [MMG06] introduce the concept of *animation spaces*. They compute the positions of the vertices as a linear combination of generalized coordinates of the bind pose positions and bone transformations. Kavan et al. [KCZO08] introduced dual quaternion skinning to reduce the artifacts caused by linearly blending transformation matrices.

All these methods are kinematic and therefore not directly aware of self-collisions of the skin. This problem can be solved by simulating the skin as a deformable object attached to a skeleton as we do. Most methods embed the surface into a volumetric mesh and solve the elasticity equations using finite elements or finite differences ([Cap04], [IKCC08], [CJ10] and [MZS*11]). Galoppo [Gal08] uses simulation textures on the surface of a character. These textures implicitly define a simulation mesh on which the elasticity equations are solved. For a detailed review of simulation based skinning methods we refer the reader to [Gal08].

3. Simulation with Oriented Particles

Since our method is based on the OP method we will first recap the basics of this simulation approach. For a more detailed description see [MC11]. In a first step, simulation particles are placed on a given visual mesh. These particles are connected to form a simulation mesh. In what follows, the term *vertex* always refers to the vertices of the visual mesh, while the term *particle* is used for nodes of the simulation mesh, i.e. the simulation particles. Along with each position vector $\mathbf{p}$, oriented particles also store a unit quaternion $\mathbf{q}$ describing their orientations. In contrast to a basic particle system, both quantities are simulated, i.e. each particle moves and rotates. The orientation $\mathbf{q}$ is used in several ways. First, particles can be represented by anisotropic shapes. As in [MC11], we use ellipsoids. This way, the geometry of the visual mesh can be approximated more accurately by the particles for collision handling (see Figure 3). In addition, the unique rigid transformation T mapping the rest state $(\bar{\mathbf{p}}, \bar{\mathbf{q}})$ to the current state $(\mathbf{p}, \mathbf{q})$ can be used to skin the vertices to nearby particles.

Let $\bar{\mathbf{x}}_1 \dots \bar{\mathbf{x}}_{N_v}$ be the positions of the vertices in the undeformed bind pose and let N_p be the number of particles.

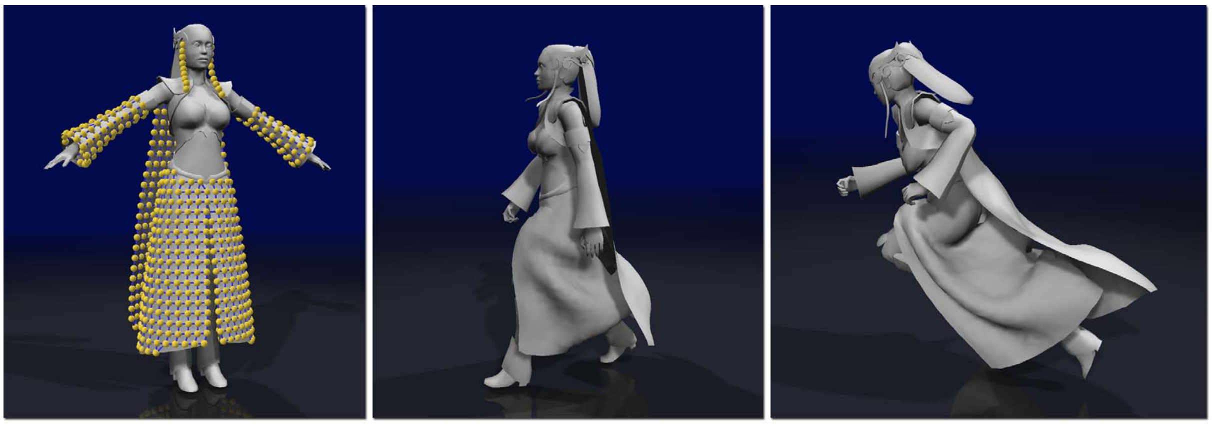

Figure 1: *Left: regular simulation meshes created with our method on separate parts of the visual mesh. Middle and right: snapshots of a walking and running animation with hair, cape and skirt interacting with each other and the body. Model courtesy of Simultronics.*

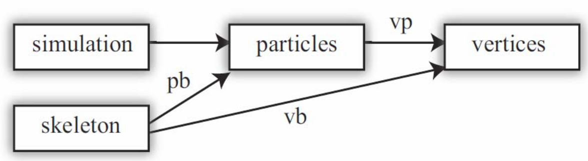

Figure 2: *The upper row shows the basic OP flow: the particles are driven by the simulation while the vertex positions are computed from the particle positions via skinning. With a rigged mesh the particles can either be driven by simulation or the skeleton, while the vertices can be skinned either to the particles or directly to the skeleton.*

To move the visual mesh along with the simulation mesh, skinning weights $w_{i,j}^{vp} \in [0\dots 1]$ and particle references $r_{i,j}^{vp} \in [1,2,\dots,N_p]$ are defined for every vertex i. The superscript *vp* indicates vertex-particle skinning. The deformed coordinates of the vertices are updated using

$$\mathbf{x}_i = \sum_j w_{i,j}^{vp}\, T_{r_{i,j}^{vp}}(\bar{\mathbf{x}}_i), \tag{1}$$

where T_j is the rigid transformation of particle j. Up to this point, we have summarized the basic OP method. In this setting, the particles move freely and the vertices follow them via skinning. Objects simulated in this way are passive like pillows or curtains. Our goal is to bring deformable objects to life by combining passive simulation with active animation.

4. Combining Simulation and Animation

Since we work with animated characters, we use rigged meshes. In addition to a regular mesh, a rigged mesh contains, a skeleton, bone weights $w_{i,j}^{vb} \in [0\dots 1]$ and bone references $r_{i,j}^{vb} \in [1,2,\dots,N_b]$, where N_b is the number of bones

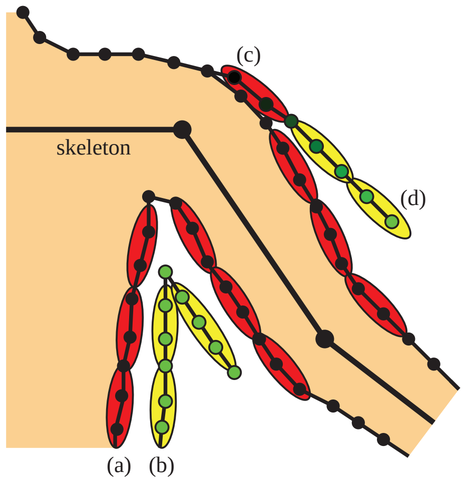

Figure 3: *2D cross section through a simulation of a shirt on a character. The inputs are a visual mesh for the skin (a) and a visual mesh for the shirt (b) with skinning information. Oriented particles are placed on the visual meshes. The yellow particles are simulated and the red particles animated, following the skeleton. Animated particles are used to attach the simulation mesh (c) or as collision primitives (all others). The vertices of the visual meshes are marked to be skinned to either the skeleton (black) or the particles (green) with smooth transitions as between vertices (c) and (d).*

of the skeleton and the superscript *vb* indicates vertex-bone skinning. Vertices are skinned to the skeleton via

$$\mathbf{x}_i = \sum_j w_{i,j}^{vb}\, B_{r_{i,j}^{vb}}(\bar{\mathbf{x}}_i), \tag{2}$$

where B_j is the rigid transform of the bone j.

To combine simulation and animation, we need a way to animate the particles and make them follow the motion of the skeleton. There is a simple way to do this. With the two sets of skinning references, we can derive references for skinning the particles to the bones of the skeleton. The links and weights from the particles to the bones are given by

$$(r_i^{pb}, w_i^{pb}) = \left\{ (r_{j,k}^{vb}, w_{j,k}^{vb} \cdot w_{j,l}^{vp}) : r_{j,l}^{vp} = i \right\}, \tag{3}$$

where the superscript *pb* indicates particle-bone skinning. In words, particle i gets attached to all bones that vertex j is attached to, if vertex j is attached to particle i. Since several vertices attached to a given particle can be attached to the same bone, the list defined in Equation (3) can potentially contain multiple references to the same bone. We lump all these references to a single entry while summing up all the corresponding weights. To speed up skinning, we only keep the k entries with the largest weights (we use $k = 4$ in all examples) and normalize the weights so they sum up to one. All three skinning maps discussed above are depicted in Figure 2.

With the particle-bone skinning information, particles can either be simulated or animated. More precisely, the rigid transformations of the particles can be either updated by the solver or dictated by the skeleton via

$$T_i = \sum_j w_{i,j}^{pb} B_{r_{i,j}^{pb}}. \tag{4}$$

We let the user decide which particles should be animated and which should be simulated. Figure 3 shows a typical use case. Here we are given a character with skin and a shirt as visual meshes rigged to a skeleton. We want to simulate only part of the shirt and make sure it collides correctly against the arm. This can be achieved by creating particles on the shirt and the skin in the simulated region and attaching specific particles to the skeleton. In Figure 3, attached particles are shown in red and particles driven by the simulator in yellow. The particle near the shoulder (c) was made animated in order to attach the simulated particles to the character. Therefore, this red particle and all yellow particles need to be connected. The other red particles have been placed on the skin mesh to ensure the yellow particles do not penetrate the character. No connectivity is needed between them because they strictly follow the animation.

An indirect way to animate the vertices would be to skin them to animated particles (see Figure 2). However, the accuracy of this two step process depends on the placement of the particles and the density of the simulation mesh. Also, particles would have to be placed everywhere on the visual mesh. We therefore use direct vertex-bone skinning to animate the visual mesh in non-simulated regions. The user paints smooth per-particle weights between zero and one to define blending between the positions resulting from skinning to particles or skinning to the skeleton. In Figure 3 the black vertices are skinned to the skeleton while the green ones follow the motion of the particles. The visual mesh between locations (c) and (d) shows a smooth transition between the two types of skinning.

Figure 4: *Baron Münchhausen pulls himself and the horse he is sitting on out of a swamp by his own hair. (Drawing by Theodor Hosemann, public domain).*

4.1. Momentum Conserving Animation

In the method for combining simulation and animation described above, the positions and orientations of animated particles are determined by the skeletal animation alone. This way, the character as a whole moves strictly according to a pre-defined path, a path that does not need to be physically possible. For instance, the character might stand still in open air, not touching the ground.

In certain scenarios we want the animation of the skeleton to only influence the body in a physically possible way. This means that the animation should not be able to change the position of the center of mass of the body. In other worlds, a character should not be able to pull himself out of a swamp by his own hair, as legendary Baron Münchhausen allegedly did (Figure 4). To achieve this, we first compute the positions of the animated particles as before. However, instead of overwriting the current particle positions, we perform a shape match first [MHT05]. More precisely, let $\mathbf{p}_i$ be the current positions of the set of animated particles and $\mathbf{a}_i$ their positions given by the animation, i.e. skeletal skinning. We first compute the moment matrix

$$\mathbf{A} = \sum_i m_i (\mathbf{p}_i - \bar{\mathbf{p}})(\mathbf{a}_i - \bar{\mathbf{a}})^T, \tag{5}$$

where m_i are the particle masses, $\bar{\mathbf{p}} = \sum_i m_i \mathbf{p} / \sum_i m_i$ and $\bar{\mathbf{a}} = \sum_i m_i \mathbf{a} / \sum_i m_i$. Then, instead of overwriting $\mathbf{p}_i$ with $\mathbf{a}_i$, we overwrite it with $\mathbf{R}(\mathbf{a}_i - \bar{\mathbf{a}}) + \bar{\mathbf{p}}$, where the rotation matrix $\mathbf{R}$ is the rotational part of the polar decomposition of $\mathbf{A} = \mathbf{RS}$. We overwrite the orientation $\mathbf{q}_i$ with $\mathbf{r}\,\bar{\mathbf{q}}_i$, where $\mathbf{r}$ is the unit quaternion that corresponds to the rotation matrix $\mathbf{R}$.

This technique was used in the simulation of the monster shown in the two shots on the right in Figure 10.

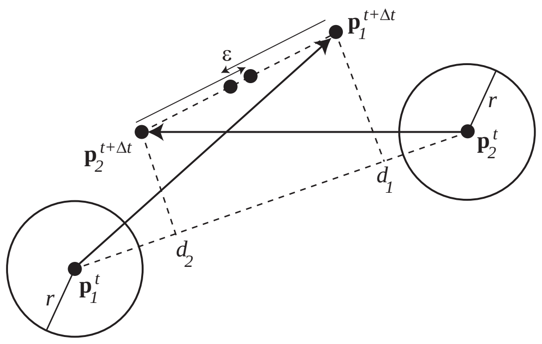

Figure 5: *Approximate continuous collision handling for two spherical particles with radius r. If, during the time step Δt, the distance between two particles becomes smaller then $2r$, we check whether their projected positions on the line $\mathbf{p}_1^t - \mathbf{p}_2^t$ got inverted. If so, we flip their positions along the line $\mathbf{p}_1^{t+\Delta t} - \mathbf{p}_2^{t+\Delta t}$ to conserve momentum.*

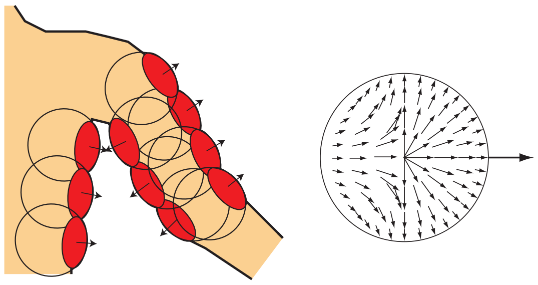

Figure 6: *Surface particles have their first axis aligned with the mesh normals. For collision, they are replaced by a sphere tangent to the ellipsoid at the extremal point of this axis for robust collision handling. The knowledge of the outward direction can also be used to modify the direction in which colliding particles are projected out of the sphere to prevent entanglements.*

5. Collision Stabilization

The ellipsoid representation of particles allows a more accurate approximation of the shape of the visual mesh. On the other hand, particles can be arbitrarily thin. The thinner the particles, the higher the chance of missing collisions. For this reason we restrict the aspect ratio of the ellipsoids to be smaller than 2 as [MC11] propose. Collision detection is even more challenging in the presence of animated components because the prescribed trajectories of animated particles can contain arbitrary velocities and accelerations. Therefore, we extend the basic OP method with methods to stabilize collision handling.

There are two main types of collision: collision between the simulated components and collision of simulated against animated parts. In the case of a character with simulated clothing, the first type corresponds to cloth self collision and the second type to collision of the clothing against the character's skin. In most cases, correct self collision handling is not as essential as collision handling against the animated character. For instance, a soft layer representing a belly rarely collides with itself. In games, the parameters and the shape of the clothing are often tuned such that self collisions occur rarely such as with capes or coats. Another example is the hair simulation shown in Figure 11, where intersections among the hair strands are hardly visible but penetrations with the body would be disturbing. Therefore we suggest a simple method to improve self collision handling which does not avoid self collisions completely but works in most cases. Also, if self collisions were perfectly handled, entangled states would never be resolved spontaneously. In contrast, robust collision handling against the animated character is essential and omnipresent. We thus propose an additional method to stabilize character collisions.

5.1. Self Collision Handling

To reduce self penetrations we propose the continuous collision handling method depicted in Figure 5. This step is performed only once per time step, after the prediction step and before the solver handles static collisions. Its aim is to make sure that particles that collide at some point during the time step to not pass through each other so that subsequent static collision handling projects them in the right directions. Let us assume we have two spherical particles with radius r traveling linearly from position $\mathbf{p}_1^t$ to position $\mathbf{p}_1^{t+\Delta t}$ and from $\mathbf{p}_2^t$ to $\mathbf{p}_2^{t+\Delta t}$ during a time step. Figure 5 depicts this situation. We first test whether the particles collide as some point during the time step, i.e. if

$$\min_{s\in[0\ldots1]} ||\mathbf{p}_1^t + s(\mathbf{p}_1^{t+\Delta t} - \mathbf{p}_1^t) - \mathbf{p}_2^t - s(\mathbf{p}_2^{t+\Delta t} - \mathbf{p}_2^t)|| < 2r.$$

If this is the case we further test whether they pass through each other. We do this by checking whether their projections on the line $\mathbf{p}_2^t - \mathbf{p}_1^t$ are flipped, i.e. whether

$$(\mathbf{p}_2^t - \mathbf{p}_1^t)\cdot\mathbf{p}_1^{t+\Delta t} > (\mathbf{p}_2^t - \mathbf{p}_1^t)\cdot\mathbf{p}_2^{t+\Delta t}. \quad (6)$$

If the positions are flipped, we move to particles to undo the inversion as follows:

$$\mathbf{p}_1^{t+\Delta t} \leftarrow \mathbf{p}_1^{t+\Delta t} + (\mathbf{p}_2^{t+\Delta t} - \mathbf{p}_1^{t+\Delta t})(1+\varepsilon)\frac{w_1}{w_1+w_2} \quad (7)$$

$$\mathbf{p}_2^{t+\Delta t} \leftarrow \mathbf{p}_2^{t+\Delta t} + (\mathbf{p}_1^{t+\Delta t} - \mathbf{p}_2^{t+\Delta t})(1+\varepsilon)\frac{w_2}{w_1+w_2}, \quad (8)$$

where $w_1 = 1/m_1$, $w_2 = 1/m_2$ and ε a small number. It is important that the particles are moved along their connecting line in order to conserve momentum. We do not have to resolve the collision completely because this is done by the solver via static collision handling. We just have to make sure that the positions do not get flipped. For the same reason we simply replace ellipsoids by their circumspheres.

5.2. Surface Particles

To keep the number of particles low a volumetric object is ideally represented by particles on the surface only. For a more accurate approximation of a smooth surface these particles typically have the shape of flat, thin ellipsoids as shown in the left image of Figure 6. However, with only a thin layer of ellipsoids the chances of missing collisions are high. In addition, once particles pass through the surface layer they get trapped inside the object which results in entanglements.

To alleviate these problems we let an artist mark certain particles as *surface particles* and treat them specially. The orientation information of particles is used in various ways by the basic OP method. We add an additional function to this list. If the ellipsoids are created as described in [MC11], they are aligned with the geometry of the surface and one axis points out of the object as shown in Figure 6. We identify this axis as the one that is best aligned with the average of the triangle normals in the neighborhood. By making this axis the first column in the orientation matrix, we always know the way out of objects during the simulation.

This information is used in two ways: to enlarge the collision volume and to modify the directions in which colliding particles are moved. We increase the collision volume of a particle by replacing the ellipsoid with a sphere that is tangent at the extremal point on the outward axis (see Figure 6). More precisely, let a,b,c be the principal radii of a specific ellipsoid and the outward axis aligned with the x-axis. The principal curvature radii at $[a,0,0]^T$ are $r_y = \frac{b^2}{a}$ and $r_z = \frac{c^2}{a}$. We replace the ellipsoid by a sphere with radius $R = \frac{r_y+r_z}{2}$ centered at $[a-R,0,0]^T$. Potentially R can become arbitrarily large. However, by restricting the aspect ratios of the ellipsoids such that $a,b,c \in [\frac{1}{\sigma}r,\dots r]$, R is bounded by $R \leq \sigma r$, where r is the global particle radius. As mentioned above, we use $\sigma = 2$.

To make sure particles do not get trapped inside objects we modify the direction in which other particles are projected out of a surface particle, as shown in the left image of Figure 6. First, the vector to the closest point on the surface of the collision sphere is computed. If the component of this vector along the outward axis is negative, we flip it. This yields the smooth field shown in the image. Colliding particles are then moved using this modified field. Note that simply moving colliding particles out of the collision sphere along the outward axis would yield unnatural drift on the surface.

As Figure 6 shows, the collision sphere of a surface particle typically overlaps with many other particles. To make sure that these overlaps are not registered as collisions by the solver at run time, we filter out pairs that already intersect in the rest state.

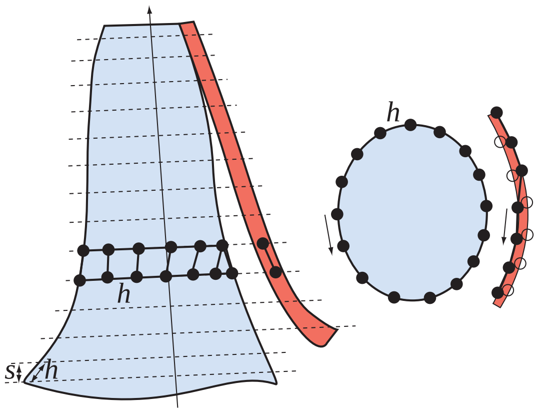

Figure 7: *Vertical and horizontal cross sections of a coat and cape. Particles (black dots) are placed on the cut lines of the meshes with horizontal planes. A particle is only created if there are no close particles already placed on the mesh. If desired, edges (thick black lines) are created between subsequent particles on a cut line and between particles on neighboring horizontal planes.*

6. Mesh Generation

The authors of [MC11] distributed simulation particles randomly on the visual mesh. For clothing we found that regular simulation meshes yield better results then meshes with irregular structure. Therefore, we devised a specialized mesh generation method with the following features:

- It works with triangle soups but utilizes connectivity information if available
- It creates a quad-mesh-like distribution of particles and connectivity but without the guarantee of being manifold
- It creates single-layered simulation meshes for thin multilayered visual meshes.
- The method works automatically but yields better results if the artist applies it to subregions of the meshes separately (see Figure 1).

Artists often create their visual meshes without physical simulations in mind so being able to handle arbitrary triangle meshes robustly is important. On the other hand, the OP method does not pose any constraints on the connectivity structure of the simulation mesh, which gives this algorithm more freedom. Figure 7 shows the basic steps of approach. The method can be applied to all visual meshes in the model at once, like the coat and the cape. Ideally, the artist works on sub regions or sub meshes one at a time and stitches the parts together by hand if connectivity is needed. In case the algorithm is used to place collision particles only, like the red particles in Figure 3, particles alone without edges are sufficient.

The algorithm itself works as follows. The main parameter provided by the artist is a spacing distance h. First, we per-

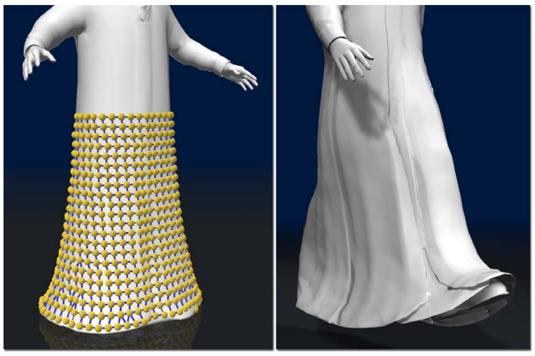

Figure 8: *Left: our method creates a single layered simulation mesh for a two-sided visual mesh. Right: simulation of the thick cloth. Model courtesy of CCP.*

form a principal component analysis on the selected triangles and choose the axis along the largest dimension. We then walk along this axis from bottom to top. The step size s is determined by multiplying h with the average inclination of the triangles w.r.t. to the axis on a given level (see Figure 7). At each level we consider the plane perpendicular to the axis and all the triangles intersecting it. These triangles are first unmarked. Then, for each unmarked intersecting triangle we construct the intersection line of the mesh with the plane by starting at the triangle and walking along the mesh in both directions. All triangles visited during the process are marked. The result is a piecewise linear chain of segments. In the image on the right of Figure 7 there are two intersection lines, i.e. the borders of the blue and red areas. To create the particles we start at the beginning of each chain or at an arbitrary position if the chain is circular. In Figure 7 the start positions and traversal directions are indicated with the two arrows. While walking along the line we place simulation particles with spacing h but only if there is no particle on the same level closer than distance h to the new position. Positions for which this criterion is not met are shown as empty circles in Figure 7. If a newly created particle has a predecessor, we connect it by an edge. If not, we connect it with the closest particle on the level if their mutual distance is smaller than h. In case of the coat, the resulting simulation mesh is simply a connected circle. For the cape, a single line is generated even though the cross section is volumetric, which is what we want in this case.

To connect the particles vertically, each particle on level i is connected to the closest particle on level $i-1$ if the distance between them is smaller than h. If there are more particles on level $i-1$ than on level i, some particles on level $i-1$ do not get connected to level i particles. Therefore, the particles on level $i-1$ are also connected to the closest particles on level i if the edge does not yet exist.

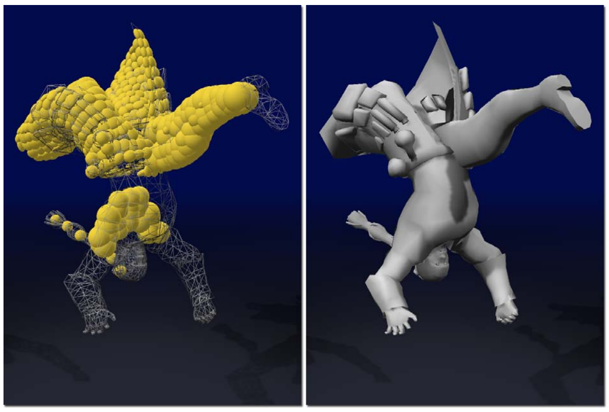

Figure 9: *Simulation of clothing with ornaments. Surface particles on the legs prevent penetrations during the fast motion of a somersault. Model courtesy of Simultronics.*

7. Results

All our examples were simulated in real time on a single core of an Intel Core i7 CPU at 3.1 GHz and skinned and rendered on an NVIDIA GeForce GTX 480 GPU. The timings are summarized in Table 1.

Scene	Particles	Triangles	Simple	Final
Cape girl	1400	12k	40	35
Monster	130	40k	250	45
Robe	780	7k	60	50
Hair	220	17k	300	40
Somersault	700	7k	90	50

Table 1: *Frame rates (in frames per second) of the demo scenes. The first column lists the number of simulation particles used, the second the number of triangles of the visual mesh, the third the frame rate including skinning of the visual mesh with simple shading and the fourth the frame rate including final rendering.*

To simulate the hair, skirt and cape of the girl shown in Figure 1 we created simulation meshes on several parts of the model using our method. To demonstrate physical motion of the skin itself we added particles in the chest region (not shown in the Figure) for discreet breast movement. There are a total of 1400 simulation particles. All simulated parts interact with each other and collide against the body. This scene runs at 40 fps including skinning and rendering of the 12k triangles.

We used the animated monster shown in Figure 10 to demonstrate the different behavior between direct and momentum conserving animation. The first two images show the monster following a physically impossible path through the air. The two images on the right show the behavior of the monster when the same animation is applied in a momentum conserving way. This time, the monster stays on the floor as expected in a physical world.

Figure 8 shows a robe modeled with a two sided visual mesh

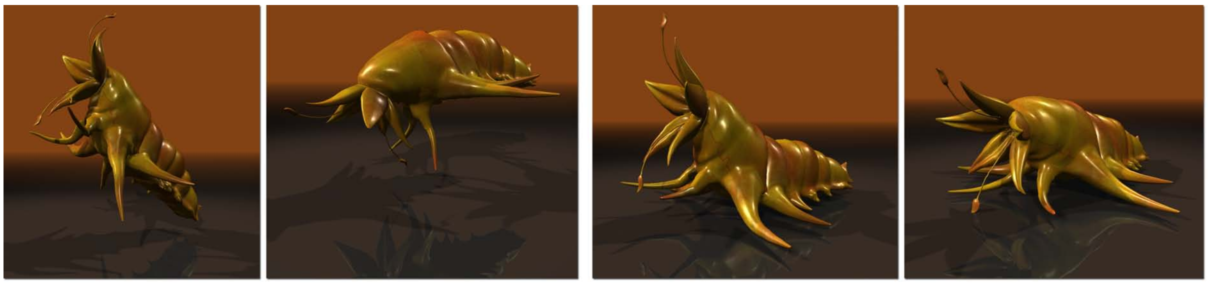

Figure 10: *Left: a monster following a physically impossible path through the air. Right: the same animation applied with momentum conservation.*

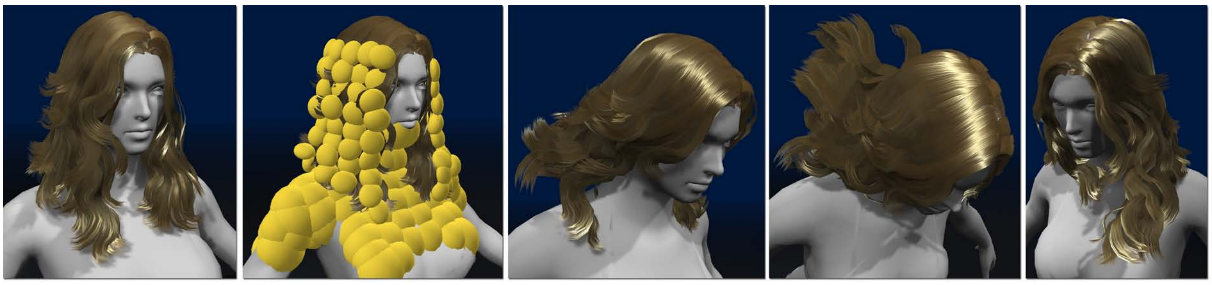

Figure 11: *Simulating complex hair geometry with 220 simulation particles for dynamics and collision handling. Model courtesy of CCP.*

to give the impression of thick cloth. With our mesh generation method we created a regular, single layered simulation mesh. This way, the solver does not see the complex visual geometry. The visual mesh simply moves with the simulation mesh via skinning. This scene comprising 780 simulation particles runs at 60 fps.

We also applied our technique to the simple hair animation shown in Figure 11. In a game, the time available for simulating the hair of a character is very limited. To make this simulation fast, we only use 100 simulated particles and skin multiple hair strands to a single particle chain. A nice side effect of this choice is that the hair remains volumetric because the vertical distances between the mesh sheets are kept constant. For collision handling we placed another 120 animated particles on the body. With simple shading we achieved a frame rate of 300 fps.

The last scene sown in Figure 9 demonstrates the robustness of our approach. To prevent the clothing from penetrating the legs even in the fast motion of a somersault, we placed surface particles on the body of the character. In total we used 700 particles on a visual mesh of 7k triangles resulting in a simulation frame rate of 90 fps.

8. Conclusion and Future Work

We have presented a method to simulate parts of an animated character. The method extends the oriented particle approach in various ways. We showed how to transfer animation data from the visual mesh to simulation particles for collision handling and to attach the simulation mesh to the animated character. We also presented ways to improve the stability of collision handling needed for fast moving characters. Finally, we proposed a mesh generation method that works robustly with arbitrary triangle surfaces. Our method works robustly with characters taken from real games as various test scenes have shown.

Our method is well suited to simulate complex geometry and unstructured visual meshes. However, for simple, single layered cloth where the simulation and visual meshes match one to one, skinning – one of the main features of the oriented particle approach – is not needed. In those cases, other methods such as mass spring systems are better suited because their computational cost per particle is lower.

In the future we plan to improve the mesh generation method to detect principal parts of the input mesh and stitch the resulting simulation meshes together automatically. We also work on a parallelization using CUDA and integration into a game engine.

References

[Cap04] Capell S.: *Interactive Character Animation Using Dynamic Elastic Simulation*. PhD thesis, University of Washington, 2004. 2

[CJ10] Clutterbuck S., Jacobs J.: A physically based approach to virtual character deformation. In *ACM SIGGRAPH 2010 talks* (2010). 2

[CTM08] Curtis S., Tamstorf R., Manocha D.: Fast collision detection for deformable models using representative-triangles. In *Proceedings of the 2008 symposium on Interactive 3D graphics and games* (New York, NY, USA, 2008), I3D '08, ACM, pp. 61–69. 2

[Gal08] Galoppo N.: *Animation, Simulation, and Control of Soft Characters using Layered Representations and Simplified Physics-based Methods*. PhD thesis, UNC, 2008. 2

[HVS*09] Harmon D., Vouga E., Smith B., Tamstorf R., Grinspun E.: Asynchronous Contact Mechanics. *SIGGRAPH (ACM Transactions on Graphics)* (Aug 2009). 2

[IKCC08] Irving G., Kautzman R., Cameron G., Chong J.: Simulating the devolved: Finite elements on walle. In *ACM SIGGRAPH 2008 talks* (2008). 2

[JWL08] Jiang Y., Wang R., Liu Z.: A survey of cloth simulation and applications. In *Computer-Aided Industrial Design and Conceptual Design, 2008. CAID/CD 2008. 9th International Conference on* (nov. 2008), pp. 765 –769. 2

[KCZO08] Kavan L., Collins S., Zara J., O'Sullivan C.: Geometric skinning with approximate dual quaternion blending. vol. 27, ACM Press, p. 105. 2

[MC11] Müller M., Chentanez N.: Solid simulation with oriented particles. In *ACM Transactions on Graphics (Proc. SIGGRAPH 2011) (to appear)* (2011). 1, 2, 4, 6

[MHR06] Müller M., Hennix B. H. M., Ratcliff J.: Position based dynamics. *Proceedings of Virtual Reality Interactions and Physical Simulations* (2006), 71–80. 2

[MHT05] Müller M., Heidelberger B., Teschner M.: Meshless deformations based on shape matching. In *Proc. SIGGRAPH 2005* (2005), pp. 471–478. 4

[MMG06] Merry B., Marais P., Gain J.: Animation space: A truly linear framework for character animation. *ACM Trans. Graph. 25* (October 2006), 1400–1423. 2

[MTLT88] Magnenat-Thalmann N., Laperrière R., Thalmann D.: Joint-dependent local deformations for hand animation and object grasping. In *Proceedings on Graphics interface '88* (Toronto, Ont., Canada, Canada, 1988), Canadian Information Processing Society, pp. 26–33. 2

[MZS*11] McAdams A., Zhu Y., Selle A., Empey M., Tamstorf R., Teran J. M., Sifakis E.: Efficient elasticity for character skinning with contact and collisions. In *ACM Trans. Graph.* (August 2011), vol. 30, pp. 37:1–37:12. 2

[NMK*05] Nealen A., Müller M., Keiser R., Boxerman E., Carlson M.: Physically based deformable models in computer graphics. *Eurographics 2005 state of the art report* (2005). 2

[OTSG09] Otaduy M. A., Tamstorf R., Steinemann D., Gross M.: Implicit contact handling for deformable objects. *Computer Graphics Forum (Proc. of Eurographics) 28*, 2 (apr 2009). 2

[PuG99] Pérez-urbiola R. E., G. I. R.: Multi-layer implicit garment models. In *Shape Modeling International Proceedings* (1999), pp. 66–71. 2

[SPO10] Schvartzman S. C., Pérez L. G., Otaduy M. A.: Star-contours for efficient hierarchical self-collision detection, 2010. 2

[TKH*05] Teschner M., Kimmerle S., Heidelberger B., Zachmann G., Raghupathi L., Fuhrmann A., Cani M.-P., Faure F., Magnenat-Thalmann N., Strasser W., Volino P.: Collision detection for deformable objects. *Computer Graphics Forum 24*, 1 (2005), 61–81. 2

[TPBF87] Terzopoulos D., Platt J., Barr A., Fleischer K.: Elastically deformable models. In *Computer Graphics (SIGGRAPH '87 Proceedings)* (1987), pp. 205–214. 2

[WBH04] Wong W. S.-K., Baciu G., Hu J.: Multi-layered deformable surfaces for virtual clothing. In *Proceedings of the ACM symposium on Virtual reality software and technology* (New York, NY, USA, 2004), VRST '04, ACM, pp. 24–31. 2

[WBK*07] Ward K., Bertails F., Kim T.-Y., Marschner S. R., Cani M.-P., Lin M.: A survey on hair modeling: Styling, simulation, and rendering. *IEEE Transactions on Visualization and Computer Graphics (TVCG) 13*, 2 (Mar-Apr 2007), 213–34. To appear. 2

[WP02] Wang X. C., Phillips C.: Multi-weight enveloping: least-squares approximation techniques for skin animation. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation* (New York, NY, USA, 2002), SCA '02, ACM, pp. 129–138. 2