Cloth Self Collision with Predictive Contacts

Chris Lewin

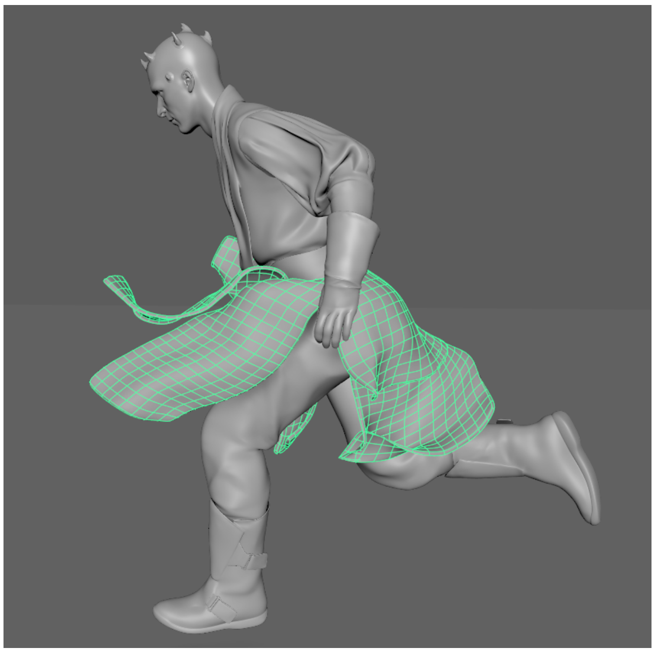

Figure 1: *A game character with self-colliding cloth, playing a violent animation. The cloth always returns to an acceptable pose even if the layers crash through each other momentarily. The total simulation cost is 0.4ms per 30Hz simulation step, using a single CPU thread.*

1 Introduction

As expectations of realism in games have increased in the last few years, cloth simulation has become a required feature for any game that wishes to be graphically competitive. Unlike simulations for film visual effects, real-time cloth simulations have to operate under severe resource constraints and look good under all possible user input. Given that cloth is often attached to characters that can move at superhuman speeds, the robustness requirements on the simulation can be quite restrictive. This invalidates many techniques used in high-end work and academia, and means that real-time cloth simulation is a somewhat different beast even when compared to gameplay rigid body simulation.

Typically, game simulations discretise cloth as a set of particles connected by soft constraints, whose motion is integrated through time using a combination of Verlet-type integration and a constraint solve using nonlinear projected Gauss Seidel (NPGS) iteration [Müller et al. 2007]. This approach can be derived from considering an implicitly integrated elastic system, and then simplifying the resulting method. Practically, what this approach means for us as designers of constraints is that to solve a constraint, one needs to define a constraint function $C(\mathbf{x})$ that measures how far away the constraint is from being satisfied, and be able to calculate the derivative of this function, $\frac{\partial C}{\partial \mathbf{x}}$.

Because of the tight resource budget afforded to cloth simulation and the low resolution at which we are able to simulate, we typically do not do very many solver iterations or simulate at a high frequency. This causes issues like artificial compliance (i.e. sagging cloth) that we can work around by introducing additional, global constraints. We also have various control constraints (different kinds of damping and spring forces, for instance) that the artist can use to keep the cloth under control. In combination, these different types of constraints can create an acceptable look for character clothing. However, one crucial missing piece is the collision interaction between the cloth and itself (or between one piece of cloth and another), which we call self collision.

2 Self Collision

Self collision of cloth is one of the trickiest phenomena to simulate in physics-based animation, for the following reasons:

1. Cloth primitives (particles or triangles) are small and can move quickly relative to their size.
2. The cloth is thin and two-sided, so in the absence of other information it's not possible to tell which side of one cloth piece another piece should be.
3. Cloth meshes deform during the simulation, unlike rigid bodies.

In high-end (Academic and VFX) work, these issues are usually dealt with using some kind of continuous collision detection (CCD) scheme. Efficient simulation of cloth with self collision in this field is still evolving, but generally relies on:

1. Performing CCD sweeps multiple times per simulation step.
2. Continuous collision queries between deforming primitive pairs (point-triangle and edge-edge).

Both of these techniques are very expensive and the present work represents our attempt to avoid doing as many of these calculations as possible.

3 Collision with Contacts

Before continuing with self collision, let's discuss an easier problem: preventing cloth from moving through a static collider like a sphere, capsule or box. One common way to do this is to simply use discrete collision detection and resolution: After integrating the system velocities forward, we check whether any cloth point is inside a collider and if it is, we push it out by the shortest route. This works well if the system time step is small relative to the velocities of the colliding objects, but this is almost never true in games. The most obvious artefact of this approach is the well known bullet-through-paper effect, where fast moving cloth will simply never intersect with the collider and will instead pass through it.

This is annoying enough, but there is a more pernicious problem which can even affect cloth that is mostly stationary with respect to its colliders. This is that other constraints in the system can move the cloth arbitrarily far in one timestep even if the cloth has no velocity at all, and depending on what order we solve the constraints in, the cloth might appear one side of a collider or the other. Iterating the constraint solves within one timestep can help with this, but cannot eliminate the problem. Furthermore, iterating on a discrete collision constraint means repeating the collision detection process, which is extremely expensive. So this method is highly flawed.

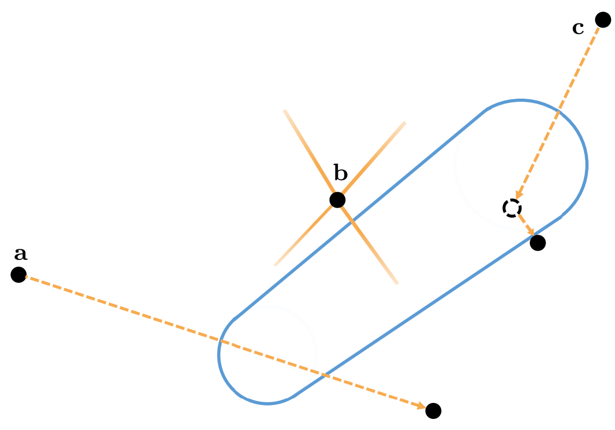

Figure 2: *Problematic cases for discrete collision detection between a particle and a capsule collider. Case* $\mathbf{a}$*: the particle has a velocity that takes it straight through the collider. Case* $\mathbf{b}$*: The particle has no velocity but is connected to constraints that may pull it through the collider. Case* $\mathbf{c}$*: The particle's velocity places it the wrong side of the collider, and it is pushed outwards in the wrong direction.*

In our solver, the way we avoid these issues is by using a predictive contact approximation. At the beginning of the simulation timestep, we have a configuration of the cloth and colliders that we assume is 'known good', i.e., the cloth is the correct side of the colliders. Then, we do a single semi-continuous collision detection step that finds points that satisfy any of the following conditions:

1. Points that are inside colliders.
2. Points that are near to colliders (within some user-defined epsilon that we call the *static collision radius*, or SCR).
3. Points that have mutual velocity with a collider that will cause them to be within the SCR of it.

For combinations of point and collider that satisfy these conditions, we create a contact constraint. This is simply an infinite plane, with a plane normal equal to the separating direction between the point and collider at the start of the frame. Solving the collision constraint then involves simply pushing the point out of the half-space behind the plane, which is an extremely cheap operation.

Using contacts thus fixes two problems: First, they are cheap to solve, which means we can do more solver iterations without making the simulation expensive. This helps to resolve situations where collisions are in opposition to other constraints (such as a cape being dragged over the body). Second, it doesn't matter how far through the contact plane the point is pushed; as long as the system of constraints is feasible and enough solver iterations are done, it will always end up the same *side* of the collider as it was at the beginning of the frame. This can cause some simulation artifacts if the point travels very far tangentially to the plane, but it does solve completely the tunnelling issues with discrete collision.

The dedicated reader may have noticed above that we used the term 'semi-continuous collision detection' above. This is what we call an approach to collision detection that seeks to answer the third point above - finding whether points have mutual velocity with a collider that will put them within a certain radius of it - without doing extremely expensive calculations. For instance, working out the closest separation between a point and a capsule collider that has both linear and angular velocity is actually very expensive. What we do in this case, and for every kind of primitive we support in

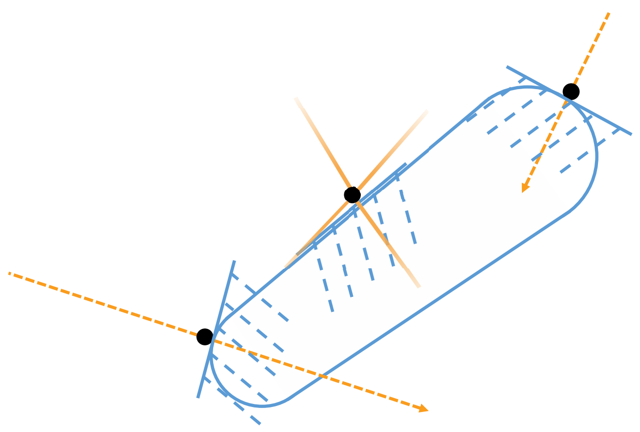

Figure 3: *Collision resolution with predictive contacts, for the problematic cases enumerated above. In each case we create an infinite half-space contact constraint which keeps the particles the correct side of the collider.*

cloth collision, is to take the discrete separating direction at the beginning of the time step and ignore changes to it over the interval. We simply look at the mutual velocity of the cloth point and the closest point on the collider *at the beginning of the timestep*, and decide based on their mutual velocity along the discrete separating direction whether they are likely to come close enough to generate a contact. This approach is no more expensive than the discrete collision detection calculations, but is much more robust. Spurious contacts will be generated by this approach, but they are much less noticeable than the cloth being pulled through colliders and thus we tolerate them.

The purpose of this long aside was to introduce the ideas of contact approximation and semi-continuous collision detection, which we apply to cloth self collision in the following sections.

4 Cloth Self Collision with Contacts

Since discrete collision detection between cloth and colliders gives acceptable behaviour and is easily implemented, it can be tempting to try this approach when doing self collision. This will work when the time step is small enough, but one crucial issue is lack of radius in the cloth. If the cloth has to be very thin for the clothing to look good when simulated, then the timesteps required for all collisions to be captured will be infeasibly small. It's for this reason that CCD is frequently used. Instead, we will construct an algorithm based on predictive contact detection and resolution.

As in the case of cloth versus colliders, we will find situations where individual cloth primitives are *likely* to come into contact with each other during the time step, and create constraints to keep them the correct side of one another. In this way we will hope to keep the cloth globally free of intersections. We support two kinds of cloth self collision: Point-point and full mesh (point-triangle and edge-edge). Point-point collision is relatively cheap and the continuous collision detection problem is easy to solve exactly. However, there is an obvious problem when the density of points in the cloth is low. Since this is almost always true in game cloth simulation, we have primarily invested our efforts into full-mesh collision. We will present all three types of constraints here.

4.1 Point-Point Contact

Consider two particles with positions $\mathbf{p}_a, \mathbf{p}_b$, displacements[1] $\mathbf{d}_a, \mathbf{d}_b$ and radius r. We want to know whether the particles will pass within the static collision radius r_{sc} of one another, and if so we will create a constraint to keep them on the same side of each other as they began the frame. We can easily solve this problem exactly by finding the distance of closest approach between the lines $(\mathbf{p}_a,\ \mathbf{p}_a + \mathbf{d}_a)$ and $(\mathbf{p}_b,\ \mathbf{p}_b + \mathbf{d}_b)$, and then creating a contact if that distance is less than $r_{sc} + 2r$. If required, we could work out the time of impact (ToI) between the two particles but this is not useful for our algorithm.

Once we know a collision or near-miss is likely to take place, we must create a constraint to prevent penetration. A natural choice for this constraint is to simply ape the point-collider contact and store a plane with normal $\hat{\mathbf{n}}$ along which the particles may not penetrate. The constraint condition is then:

$$\begin{aligned} C(\mathbf{p}_a, \mathbf{p}_b) &= \hat{\mathbf{n}} \cdot (\mathbf{p}_a - \mathbf{p}_b) - 2r \geq 0 \\ \frac{\partial C}{\partial \mathbf{p}_a} &= \hat{\mathbf{n}} \\ \frac{\partial C}{\partial \mathbf{p}_b} &= -\hat{\mathbf{n}} \end{aligned} \tag{1}$$

As long as the constraint is solved sufficiently, this ensures the particles will stay the correct side of one another. We can make an arbitrary choice about how this normal plane is defined; we simply choose the offset between the two particles at the beginning of the frame (i.e. $\hat{\mathbf{n}} = \text{normalize}(\mathbf{p}_b - \mathbf{p}_a)$). This can cause some artifacts with friction, but we find them tolerable.

4.2 Point-Triangle Contact

Consider a point $\mathbf{p}$ with displacement $\mathbf{d}$ and a triangle $(\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2)$ with displacements $(\mathbf{d}_0, \mathbf{d}_1, \mathbf{d}_2)$. We wish to know whether the point is likely to come close enough to the triangle that we need to generate a contact. If we try to solve this problem exactly, we quickly discover that we have to compute the roots of a nasty sixth-order polynomial in order to find the time of closest approach between the two primitives. Even if we treat the triangle as having zero radius, we still have to solve a cubic polynomial to find out whether the point and triangle will pass through each other during the time interval. These approaches are both too expensive for real-time simulation. Instead, we use an approximation based on discrete distance between the primitives at the beginning of the frame:

1. Find the closest point on the triangle $\mathbf{p}_c$, and the displacement of that point $\mathbf{d}_c$ using barycentric interpolation.
2. Project the displacements $\mathbf{d}$ and $\mathbf{d}_c$ along the discrete separating direction $\hat{\mathbf{n}} = \text{normalize}(\mathbf{p}_c - \mathbf{p})$, and compare the closest approach distance with the radii and SCR to decide whether to create a contact.

This approximation is conceptually like a linearisation of the sextic polynomial representing distance between the objects over time, and by increasing the SCR we can make it more and more conservative.

Once we know a collision or near-miss is likely to take place, we must create a constraint to prevent penetration. An obvious candidate in this case is to simply constrain the point to lie the same side of the triangle as at the start of the frame. This can be encoded using only a single bit, and thus a point-triangle contact contains no floating-point data: only indices of the point & triangle, and the sidedness of the contact. The constraint condition is:

$$C(\mathbf{p}, \mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2) = \hat{\mathbf{n}} \cdot (\mathbf{p} - \mathbf{p}_0) - 2r \geq 0 \tag{2}$$

where $\hat{\mathbf{n}} = \pm\text{normalize}(\text{cross}(\mathbf{p}_1 - \mathbf{p}_0,\ \mathbf{p}_2 - \mathbf{p}_0))$ depending on the sidedness of the triangle. The constraint derivatives are:

$$\begin{aligned} \frac{\partial C}{\partial \mathbf{p}} &= \hat{\mathbf{n}} \\ \frac{\partial C}{\partial \mathbf{p}_1} &= (\mathbf{p}_2 - \mathbf{p}_0) \times \mathbb{N}\hat{\mathbf{n}} \\ \frac{\partial C}{\partial \mathbf{p}_2} &= (\mathbf{p}_1 - \mathbf{p}_0) \times \mathbb{N}\hat{\mathbf{n}} \\ \frac{\partial C}{\partial \mathbf{p}_0} &= (\mathbf{p}_1 - \mathbf{p}_2) \times \mathbb{N}\hat{\mathbf{n}} - \hat{\mathbf{n}} \end{aligned} \tag{3}$$

where $\mathbb{N} = \frac{d\hat{\mathbf{n}}}{d\mathbf{n}} = (\mathbb{I} - \hat{\mathbf{n}}\hat{\mathbf{n}}^T)/|\mathbf{n}|$ is the geometric stiffness matrix.

4.3 Edge-Edge Contact

True continuous collision for deforming fat lines is essentially equivalent to the point-triangle case, and it is similarly expensive to solve the problem exactly. So we take a similar approach to the point-triangle case. Consider two lines $(\mathbf{p}_a, \mathbf{p}_b)$ and $(\mathbf{p}_c, \mathbf{p}_d)$, with displacements $(\mathbf{d}_a, \mathbf{d}_b)$ and $(\mathbf{d}_c, \mathbf{d}_d)$. We find the closest points on the two lines and their parameters (α, β) on those lines, i.e., $\mathbf{p}_\alpha = \text{lerp}(\mathbf{p}_a, \mathbf{p}_b, \alpha)$ and $\mathbf{p}_\beta = \text{lerp}(\mathbf{p}_c, \mathbf{p}_d, \beta)$. Then we find the displacement of these points at the beginning of the frame: $\mathbf{d}_\alpha = \text{lerp}(\mathbf{d}_a, \mathbf{d}_b, \alpha)$ and $\mathbf{d}_\beta = \text{lerp}(\mathbf{d}_c, \mathbf{d}_d, \beta)$. Then, as before, we project these displacements onto the discrete separation vector $\hat{\mathbf{n}} = \text{normalize}(\mathbf{p}_\beta - \mathbf{p}_\alpha)$ and compare the separation and displacement speeds with the radii and SCR to work out whether the edges may come into close proximity.

Assuming we want to generate a contact, we have to decide what quantities to actually constrain. Unlike triangles, edges have no natural orientation[2]. We must therefore, like in the point-point case, commit to having some quantities in the contact description that are frozen at the beginning of the frame. We choose a simple model for the edge-edge contact where it is exactly equivalent to the point-point contact, except the points being constrained are the virtual points $\mathbf{p}_\alpha$ and $\mathbf{p}_\beta$ some distance along each line. We must therefore store $\hat{\mathbf{n}}$, α and β along with the topology in order to be able to solve the contact. There is only one extra layer of indirection in this contact compared to the point-point case, so the projection is quite simple:

$$\begin{aligned} C(\mathbf{p}_a, \mathbf{p}_b, \mathbf{p}_c, \mathbf{p}_d) &= (\mathbf{p}_\alpha - \mathbf{p}_\beta) \cdot \hat{\mathbf{n}} - 2r \geq 0 \\ \frac{\partial C}{\partial \mathbf{p}_a} &= -(1 - s)\hat{\mathbf{n}} \\ \frac{\partial C}{\partial \mathbf{p}_b} &= -s\hat{\mathbf{n}} \\ \frac{\partial C}{\partial \mathbf{p}_c} &= (1 - t)\hat{\mathbf{n}} \\ \frac{\partial C}{\partial \mathbf{p}_d} &= t\hat{\mathbf{n}} \end{aligned} \tag{4}$$

4.4 Broadphase

We now understand the detection and resolution of contacts for point-point and full mesh self collision. However, testing every pair

[1] Here we use the term 'displacement' to mean the velocity multiplied by the timestep, i.e, the total movement over the timestep if the particle were not constrained.

[2] Although one may be able to derive an orientation from the local mesh neighbourhood, we have not yet explored this possibility.

of points or triangles against one another would be prohibitively expensive, so we must use an acceleration structure to reduce the number of primitive comparisons. We use a simple Axis Aligned Bounding Box (AABB) tree where each leaf node bounds a primitive (point or triangle) extruded along its displacement, and expanded by the Static Collision Radius. We can then find all intersections between the leaf nodes and perform contact generation on the resulting pairs of primitives.

4.5 Rep-Tri

One issue encountered when using full-mesh self collision is that the triangles are the base primitive used by the broadphase, but the actual contacts are between shared features (points, faces and edges) of the triangles. This means that without special treatment, we will potentially generate many identical contacts and waste processing power. Representative Triangles (Rep-Tri) is a simple way to filter out all these duplicate contacts. In this scheme, we greedily assign shared features (vertices and edges) to one of the triangles between which they are shared. For each triangle, we can encode which of its vertices and edges it owns using just six bits. Then, we can test these bits when looping over the feature pairs and use them to cull duplicate contacts. This provides a substantial speed boost for a very modest memory cost.

4.6 Regions

So far we have discussed self collision in purely local terms - as pairs of intersecting primitives. However, in a typical production cloth mesh there will be many parts of the mesh for which self collision is not relevant, and many parts for which the inter-collision between bits of cloth may be relevant while the collisions inside one part may not. It is thus a waste to simulate self collision on the entire cloth mesh. We provide a system for setting up regions of the cloth and determining what kind of self collision interaction they have with themselves and other regions.

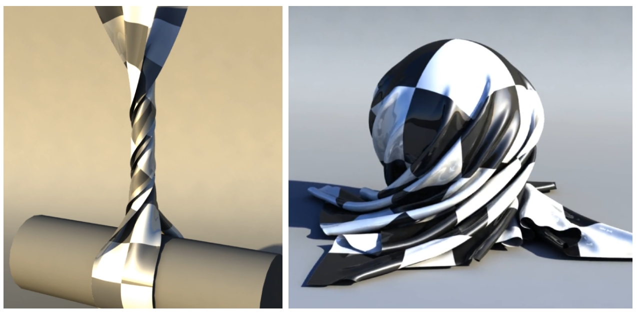

Figure 4: *Classic self collision tests using our algorithm. Left: A collider twists a tape into a knot (20ms per 30Hz frame) Right: Cloth falls onto a rotating sphere (1s per 30Hz frame).*

4.7 Results of Basic Algorithm

Using this basic algorithm, we can simulate cloth undergoing fairly sedate deformations as long as the thickness and static collision radius are generous. This can allow us to simulate most situations with reasonable quality. However, due to the fact that we are not using fully continuous collision detection, this method has difficuly dealing with quick movement and thin clothing. Increasing the static collision radius can compensate somewhat for this, but the number of spurious contacts generated can become very large and this can negatively affect behaviour and efficiency. So this basic algorithm is not actually terribly suited for videogame simulations, where inputs to the simulation can be very badly behaved.

5 Situational Tricks and Hacks

To remedy this situation, we turn to the traditional games-industry medicine of situational tricks and hacks.

5.1 Layered Clothing

Many clothing configurations that can benefit from self collision simulation are composed of layers. In this case, we may have global information that, for instance, one layer of a flamenco skirt should always rest on top of another layer. In this case, we can simply hardcode the sidedness of the point-triangle contacts between these layers using this pre-authored information. This means that even if a contact is missed one frame, and the meshes become entangled, they will be able to disentangle themselves on subsequent frames. This eliminates the main issue that plagues self collision: that failures in collision detection cannot be corrected in subsequent frames.

5.2 Contact Caching

Do we actually have to detect any collisions at runtime? For certain kinds of clothing, the answer is actually *no*. In clothing consisting of layers, the contact set will not change very much from frame to frame. In this case, we can simply compute a set of contacts once and then solve them every frame. We can do this at load time or in the pipeline, and thus avoid doing any collision detection at all at runtime. The drawback to this approach is that we will end up solving some constraints that would not have been generated if we had run collision detection, which can cause changes in behaviour when the layers move away from their starting pose. However, for some commonly used situations in game cloth simulation, this is not much of a drawback.

5.3 Results with tricks and hacks

With a combination of layered clothing and contact caching, we are able to simulate self-colliding cloth in a large subset of realistic situations at a cost that remains reasonable.

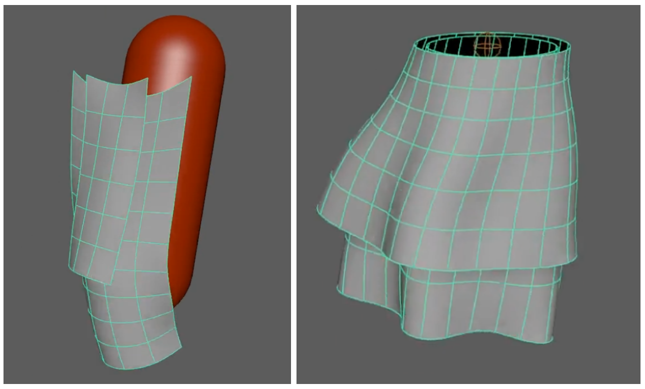

Figure 5: *Game-like self collision situations using layered clothing hacks and contact caching. Left: Two simple flaps rest against a leg collider (0.1ms per 30Hz frame). Right: Layered skirt (0.2ms per 30Hz frame).*

References

MÜLLER, M., HEIDELBERGER, B., HENNIX, M., AND RATCLIFF, J. 2007. Position based dynamics. *J. Vis. Comun. Image Represent. 18*, 2 (Apr.), 109–118.