Untangling Cloth

David Baraff Andrew Witkin Michael Kass

Pixar Animation Studios

Abstract

Deficient cloth-to-cloth collision response is the most serious shortcoming of most cloth simulation systems. Past approaches to cloth-cloth collision have used history to decide whether nearby cloth regions have interpenetrated. The biggest pitfall of history-based methods is that an error anywhere along the way can give rise to persistent tangles. This is a particularly serious issue for production character animation, because characters' bodies routinely self-intersect, for instance in the bend of an elbow or knee, or where the arm or hand rests against the body. Cloth that becomes pinched in these regions is often forced into jagged self-intersections that defeat history-based methods, leaving a tangled mess when the body parts separate. This paper describes a history-free cloth collision response algorithm based on global intersection analysis of cloth meshes at each simulation step. The algorithm resolves tangles that arise during pinching as soon as the surrounding geometry permits, and also resolves tangled initial conditions. The ability to untangle cloth after pinching is not sufficient, because standard cloth-solid collision algorithms handle pinches so poorly that they often give rise to visible flutters and other simulation artifacts during the pinch. As a companion to the global intersection analysis method, we present a cloth-solid collision algorithm called collision flypapering, that eliminates these artifacts. The two algorithms presented have been used together extensively and successfully in a production animation environment.

1 Introduction

Five years ago, the use of dynamically-simulated cloth by production animation studios was a novelty. Today, it has become a necessity. In production animation, cloth behavior must be plausible but strict physical accuracy is not a requirement. In this regard, simulating *unconstrained* cloth is largely a solved problem[Terzopoulos et al. 1987; Terzopoulos and Fleischer 1988; Carignan et al. 1992; Breen et al. 1994; Provot 1995; Eberhardt et al. 1996; Baraff and Witkin 1998; DeRose et al. 1998; Meyer et al. 2001; Choi and Ko 2002], although innovative formulations and new algorithmic efficiencies will always be welcome [Ascher and Boxerman 2002].

In contrast, handling collision and contact continues to be a serious problem for production cloth simulation, especially since collision behavior must be almost perfect: out of tens of thousands of vertices on a garment mesh, a single vertex wiggling for a few frames can ruin an entire simulation run. To achieve visually flawless results, early adopters of cloth-simulation technology have resorted to extensive pre-simulation tweaking of simulation parameters, collision geometry, and animation, as well as a variety of post-simulation fix-up techniques. All this extra work has made cloth simulation very expensive and painful as an animation tool [Berney and Redd 2000].

Successful cloth simulation methods must deal with collisions between cloth and non-simulated objects like characters and props as well as collisions of cloth with itself. In the case of collisions between cloth and non-simulated objects, it is usually possible to determine whether or not a cloth point is inside a non-simulated object, and apply forces or impulses to keep the cloth point outside or eject it. Consequently, the collision problem has been treated one particle at a time using techniques that are very well known to the computer graphics community[Terzopoulos et al. 1987; Carignan et al. 1992; Volino et al. 1995; Baraff and Witkin 1998; Bridson et al. 2002].

The problem of handling collisions of cloth with itself, however, has proved more vexing. If it is discovered that a cloth surface intersects itself, fixing the situation is not a simple matter. Given a point on the cloth near an intersection, there is no way to determine locally and instantaneously whether or not the particle is on the "wrong" side and must be pushed back through.

All past approaches of which we are aware use history to decide if a particle is on the wrong side. Bridson et al. [2002], for example, recently described a history-based approach that takes great care to guarantee that each new state is intersection free, provided its predecessor state was also intersection free. Volino et al. [1995] describes a history-based approach which makes no such guarantee, but applies penalty forces to push cloth points in a direction intended to untangle the cloth-cloth collision.

The biggest pitfall of history-based methods is that an error anywhere along the way can introduce permanent tangles: if a cloth point is on the "wrong" side but the simulator thinks it isn't, the simulator will dutifully apply penalty forces to keep it on the wrong side forever. To avoid this difficulty, we have developed what we believe is the first collision algorithm to use global geometric analysis without history to undo tangles. As a consequence, if extraordinary circumstances cause a tangle, the simulator can usually resolve it and keep going.

Given the kind of guarantees that Bridson's method provides, the ability to untangle cloth geometry may seem unnecessary: if you start Bridson's algorithm with no intersections, it will maintain that invariant. A problem, however, is that if outside constraints force cloth to intersect, the method can never recover. In production animation, this happens all the time, when cloth becomes pinched between intersecting character geometries.

Although real solid objects never intersect each other, animated character's bodies do so routinely, exhibiting all sorts of pathological geometry as they move and deform (figure 1). Particularly common are deep interpenetrations as when a character's arm sinks into her side or knees bend sharply (figures 2 and 10). Many of these interpenetrations are intended by the animators, who are animating to the viewpoint of a particular camera, and so they cannot be avoided or corrected without placing unacceptable constraints and artistic limitations on the animation process. Techniques that automatically prevent or correct[Cordier et al. 2002] intersections remain

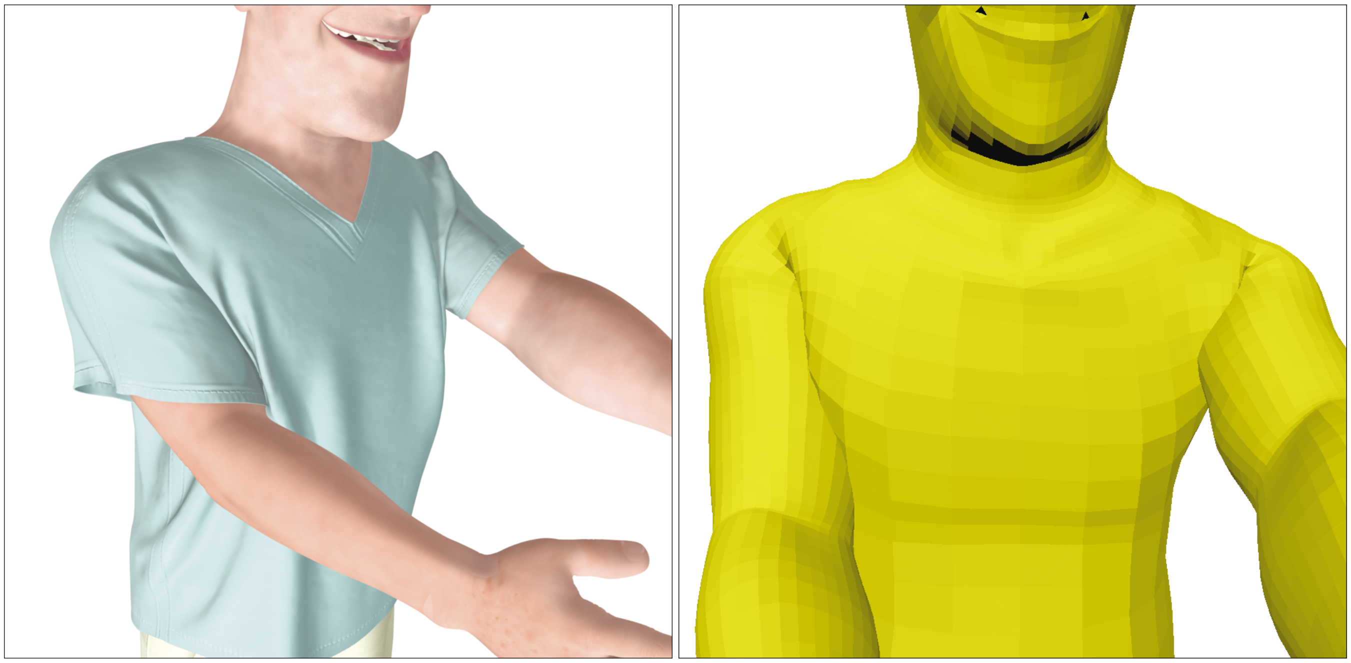

Figure 1: (right) Typical severe interpenetration of production CG characters. (left) Flypapering and GIA nonetheless deliver visually pleasing cloth simulation results.

too expensive and unreliable for production use.

Conventional cloth/solid collision techniques do not handle intersecting solids well. Cloth points that become sandwiched in areas of intersection get pulled in various directions, producing extreme stretch and shear, as well as tangling. Although the pinched areas themselves are hidden from view by the body, the effects are visible and disastrous: the cloth elastic forces that result from the stretch and shear can create visible artifacts such as large-scale flutter, and with conventional history-dependent techniques the tangles persist for the rest of the shot.

In order to handle this type of collision, we have developed a collision response technique we call *collision flypapering* which yields yields temporally smooth cloth behavior in regions of body intersection. Flypapering eliminates nearly all visible artifacts during the intersection by carefully controlling the motion of any trapped or pinched cloth points.

While flypapering avoids visual artifacts during severe character interpenetrations, it cannot guarantee that the affected cloth will be free from self intersections after the interpenetration is over. As a consequence it cannot be used effectively with an algorithm like Bridson's. It must be used in concert with an algorithm which allows recovery from tangled states. We have developed such an algorithm, which we regard as the main contribution of the paper. It is a cloth/cloth collision technique that analyzes intersections globally but instantaneously, rather than locally with history. The algorithm performs a *global intersection analysis* (GIA) of the interacting cloth meshes, characterizing the current intersection state in order to guide the cloth back to an untangled state when intersections occur.

2 Fixing Cloth Self-Intersections

All cloth simulators we know of handle cloth-to-cloth collisions by introducing repulsive forces or impulses between interacting portions of cloth. If two regions of cloth get too close, the repulsive forces attempt to prevent them from passing through each other. This approach works very well in relatively unconstrained situations where the forces are generally able to maintain the invariant that the cloth never self-intersects.

Bridson et al. [2002]'s recent work shows impressive results for crumpling cloth where the invariant is actually guaranteed. They use a combination of repulsive impulses and forces to establish and maintain consistent velocities between nearby cloth particles. In highly constrained situations, however, repulsive forces are far less effective. Bridson et al. note that even their algorithm cannot prevent a self-intersecting state from occurring in the case of severe pinching contact by solids. With constraining geometry typical of production character animation there is no guarantee, in fact no expectation that repulsive forces alone can prevent self intersections. One must therefore confront the problem of untangling a cloth surface after it has passed through itself. Repulsive forces need to be disabled, or better, replaced by attractive forces, in areas that have interpenetrated. This means that interpenetrated regions must first be detected.

Volino et al. [1995] recognize this fundamental problem, and describe a method that applies either attractive or repulsive forces to nearby surface elements based on a decision about the pair's orientation—whether they are on the "right" or "wrong" side of each other. Orientations are first assigned using local geometry and the history of nearby vertex/face and edge/edge pairs. Specifics of the assignment algorithm are not given except to say that it involves detecting crossing of the elements. The authors state that the local technique cannot handle intersected initial conditions or "a situation that has become inconsistent after some severe simulation trouble" [Volino et al. 1995]. To deal with these situations they describe a further technique in which colliding pairs are grouped into connected components and a single orientation assigned to the group based on statistics of the locally assigned orientations.

Volino et al. showed impressive results for cloth-cloth collision geometries, such as a falling ribbon, that were not strongly constrained by surrounding solid collision geometry. While visually complex, simulations with modest external forces and weakly constraining collision objects are far easier to handle from a collision standpoint than the crumpling with tight contact that our method was designed to handle. We would not expect Volino et al.'s method to succeed in the presence of severe pinching by surrounding solid objects. During pinching, cloth often becomes so jagged that the measured normals on which Volino et al. rely are all but meaningless. In addition, we would expect the reliability of the local history-based orientation assignments to degrade as the duration of interpenetration increases. In effect, the method would lose its memory.

The conclusion we draw from examining the methods of Bridson et al. and Volino et al., is that carrying collision history through extended, forced interpenetration events is a very difficult problem. Rather than trying to solve it, we present a method that makes no use of history, but instead draws on global topological analysis of intersections. While it does not uniquely determine a solution, this global analysis constrains the local attract/repel decisions strongly enough that we can solve the problem with high reliability.

3 Simulation Method

The underlying dynamics of our cloth simulation system largely follow that of Baraff and Witkin[1998] with elements from DeRose et al. [1998]. Cloth is modeled as a triangle mesh of particles, with stretch and shear forces formulated per mesh triangle, and bend forces formulated per pair of edge-adjacent triangles. Companion damping forces and external forces are present as well. The system is time-stepped using a single-step implicit Euler method.

Each cloth/solid contact is handled by directly enforcing a "hard" one-dimensional constraint on the contacting cloth particle; these hard constraints override any other forces in the system and always enforce a desired position and velocity (normal to the contact plane) for a particle in one simulation step. We use Baraff and Witkin's[1998] projection method which enforces constraints as part of the linear equation solver which time-steps the system forward.

When a cloth point is pinched between multiple surfaces, we employ a method called "flypapering." Instead of constraining one dimension of a particle's freedom, as we do during contact with a single surface, the collision flypapering method completely dictates a flypapered particle's position and velocity. These positions and velocities are calculated to yield realistic-appearing cloth behavior while allowing for graceful recovery when pinching ceases.

Collision flypapering is described in detail in section 4. Our global intersection analysis (GIA) method for cloth/cloth collision is described in section 5.

4 Collision Flypapering

Intersection of solid objects, and in particular self-intersection, is the major difficulty in dealing with cloth/solid collisions. Figure 2a shows simulated pants being pinched near a character's knees as she squats down; a cut-away view (figure 2b) shows that the character's legs greatly intersect as her knees bend. Clearly, cloth caught near the knee will be forced to intersect through part of the leg as long as the knee is sharply bent.

Despite this physically unrealistic intersection, the cloth motion can still appear realistic. Our experience has been that realistic cloth motion in the presence of pinching is critically dependent on the ability of the pinched cloth particle to remain motionless whenever the solid surfaces they are pinched by are motionless. Additionally, pinched cloth needs to lie roughly midway between pinched objects, as in figure 2. Finally, pinched cloth should not undergo abrupt transitions when pinching starts, stops, or when the number of surfaces pinching the cloth changes.

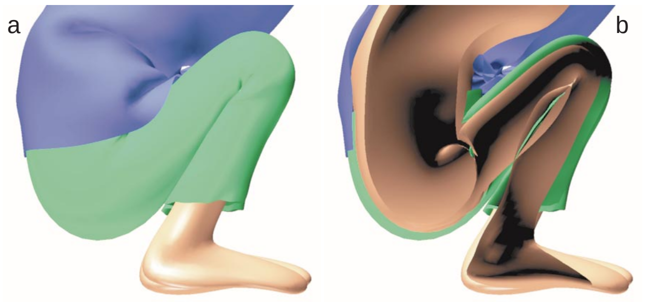

Figure 2: (a) An extreme pose resulting in strongly pinched pants. (Note: the cloth is deliberately low-resolution; this is an early preproduction example, not a finished result.) (b) Closeup cut-away view shows cloth flypapered between the legs. Note the extreme intersection between legs near the knees; collision flypapering produces a response resulting in the cloth neatly lying midway between the intersecting leg areas.

4.1 Approaches That Do Not Work

Since we already use hard constraints to enforce contact with a cloth particle and a single solid surface, we considered extending this approach to handle pinching behavior. Given a set of pinching surfaces, we attempted to distill the multiple contacting surfaces into a single planar constraint, approximately midway between the surfaces. We found, however, that even for pinching between a pair of solids, the opposing surface normals were rarely opposite enough for this to be effective; moreover, there was a great lack of consistency in the constraints generated for neighboring pinched particles. This led to unnatural distortions of the pinched cloth, with pinched particles still free to move in a plane. The resulting particle movements caused the planar constraints on a particle to vary from step to step, even if the pinched solid objects were motionless. In other words, everything wiggled.[1]

We next considered abandoning constraints and simply using standard repulsive forces to push particles away from surfaces. This has intuitive appeal, since it naturally handles any number of pinching solid surfaces; we hoped the generated repulsive forces would balance, leaving cloth roughly midway between solid surfaces without any computational effort at all. This approach failed as well; the cloth/solid repulsive forces were left to compete with all other forces in the system (and most notable with the cloth/cloth forces). The result was again a system which often wiggled even when the solid objects were motionless.

4.2 Flypapered Particles

If we simply consider the case of cloth particles pinched between motionless solid objects, an obvious solution presents itself: pinched particles should not move. The rationale for this is that the friction forces generated by the pinch are a match for any other forces acting on the cloth, and adjust to eliminate any motion. From an algorithmic standpoint, enforcing a hard constraint which "flypapers" these particles so that they remain in place is trivial.

The problem, of course, is defining the behavior for flypapered particles when the solid objects that flypaper them are in relative motion. Suppose that a cloth particle came into contact with a single "sticky" solid object. If the cloth particle adhered to the solid object, we could define a constraint on the particle as follows. Let S denote the solid object surface and let $W_S(\mathbf{b}, t)$ map points $\mathbf{b}$ in

[1]Adding insult to injury, the instigators of the wiggles—the pinched particles—tended to be hidden from view by the solid objects. Thus, the wiggling pinched particles were not directly seen, but they induced wiggles in their neighbors. Tracking down the actual culprits became complicated.

some body-space coordinate system to world-space coordinates at time t. The mapping from world space back to S's body space is $W_S^{-1}(\mathbf{w}, t)$ where $\mathbf{w} \in \mathbb{R}^3$ is a point in world space. If the cloth particle adhered to S at time t_0, then the particle remains attached to S at future times t by requiring that the particle's position $\mathbf{p}(t)$ satisfy

$$\mathbf{p}(t) = W_S(\mathbf{b}, t) \qquad \text{where} \qquad \mathbf{b} = W_S^{-1}(\mathbf{p}(t_0), t_0).$$

Note that we only need to construct these mappings locally, that is, near $\mathbf{p}(t_0)$.

However, a flypapered particle needs to be attached to multiple solid objects, which may be in relative motion. At time t_0, we flypaper a particle with position $\mathbf{p}(t_0)$ to multiple solid surfaces S_1 through S_n by first defining body-space coordinates $\mathbf{b}_i$ relative to each surface S_i near $\mathbf{p}(t_0)$. We also define "goal" positions $\mathbf{g}_i(t)$ for the particle relative to each surface at time t. The body-space and goal positions are simply computed as

$$\mathbf{b}_i = W_{S_i}^{-1}(\mathbf{p}(t_0), t_0) \qquad \text{and} \qquad \mathbf{g}_i(t) = W_{S_i}(\mathbf{b}_i, t); \tag{1}$$

note in particular that $\mathbf{g}_i(t_0) = \mathbf{p}(t_0)$ for all i. Given these definitions, a collision flypapered particle's position at time t_1 is constrained by enforcing

$$\mathbf{p}(t_1) = \frac{1}{n}\sum_i \mathbf{g}_i(t_1). \tag{2}$$

Several properties regarding this constraint on $\mathbf{p}(t_1)$ are immediately apparent. First, if none of the surfaces are in motion, $\mathbf{p}(t_1)$ is constrained to be equal to $\mathbf{p}(t_0)$ since each $\mathbf{g}_i(t_1) = \mathbf{p}(t_0)$. More strongly, if the pinching surfaces move together as if they were a single rigid body, the pinched particle moves as it was firmly attached to this rigid body. Thus, equation (2) satisfies the property that motionless pinching objects yield motionless flypapered cloth particles.

Equation (2) also tends to position flypaper particles roughly midway between pinching surfaces. To see this, consider a single pair of surfaces S_1 and S_2 that pinch a particle, and imagine that S_1 is motionless while S_2 moves inward toward S_1. Then $\mathbf{g}_1(t_1) = \mathbf{g}_1(t_0) = \mathbf{p}(t_0)$ which means that

$$\mathbf{p}(t_1) = \frac{\mathbf{g}_1(t_1) + \mathbf{g}_2(t_1)}{2} = \frac{\mathbf{p}(t_0) + \mathbf{g}_2(t_1)}{2}$$

which, since $\mathbf{g}_2(t_0) = \mathbf{p}(t_0)$, yields

$$\mathbf{p}(t_1) - \mathbf{p}(t_0) = \frac{\mathbf{p}(t_0) + \mathbf{g}_2(t_1)}{2} - \mathbf{p}(t_0) = \frac{\mathbf{g}_2(t_1) - \mathbf{g}_2(t_0)}{2}.$$

In other words, the particle inherits exactly half of S_2's motion. Over time, the flypapered particle tends to lie midway between S_1 and S_2.

While equation (1) seems to imply that the simulator needs history—a memory of the body-space coordinate vectors $\mathbf{b}_i$ when pinching was initiated at time t_0—collision flypapering is better implemented without such history. In particular, we deliberately recompute the vectors $\mathbf{b}_i$ at each time step. This is an improvement, because over time, the initial body-space coordinates when flypapering was first initiated become irrelevant. It matters only that for a given current state of particles and solid objects, the particles move when the solid objects do and stop when they stop. Additionally, by ignoring history, flypapered cloth particles are not affected adversely if the number of solid object surfaces participating in the flypapering changes abruptly.

4.3 Nonuniform Weighting

Equation (2) defines flypapering by giving equal weight to all solid object surfaces. We can select among a range of behaviors by modifying the constraint imposed by equation (2) to be instead

$$\mathbf{p}(t_1) = \sum_i \alpha_i \mathbf{g}_i(t_1).$$

where α_i are a set of n nonnegative weights satisfying $\sum \alpha_i = 1$.

Taken to an extreme, we might choose $\alpha_1 = 1$ and set all other α_i to zero. In this case, when flypapered, a particle will exactly track the motion of surface S_1, and will ignore the motion of the other surfaces. When might we do this? Consider a character with pants, skidding on the floor. If we wish the pants to remain glued to the legs, but *only* when the pants are pinched between the legs and the floor, then we choose weights so that the pants track exactly with the legs, while ignoring the floor. (The reverse, in which the pants stick to the floor while the character slides out of them, is typically not desirable.)

A more moderate scenario occurs when we wish some particles to track one collision surface closely, while being somewhat influenced by another. For a character wearing a shirt and rubbing her stomach, we might want the cloth to stay 98% stuck to the torso and follow the hands at only a 2% value; but if we want the hand rubbing to appear more vigorous, we could set the weights so that the shirt follows the torso at 90% and the hands at 10%. Of course, the weighting values can vary from particle to particle; for example, if the character is wearing gloves, the gloves would want to track the fingers closely, and pay little attention to other parts of the body. Figure 9 shows the result of identical animated character motions with different flypapering weightings.

4.4 Pinch Determination

The actual determination that a cloth particle needs to be flypapered is straightforward for the most part; if a particle is in close proximity to multiple distinct surfaces flypapering is in general warranted. However, given that a single solid object surface can deform so as to create pinches with itself (for example, under the knees in figure 2) the matter does require some attention.

A single solid surface should induce flypapering if the surface folds back on itself so that two distinct portions of the surface (with reasonably opposing surface normals) contact a cloth particle. Unfortunately, any sort of bump in the character's skin, or "thin" feature, such as a finger, involves some portion of the skin surface being near another portion, with each surface's inward normal being nearly opposite. Yet cloth that merely touches a finger, or slides over a bump in the skin should not be flypapered. Fortunately, the global intersection analysis (GIA) described in section 5 gives us a simple way of distinguishing between these two cases.

Consider figure 3. Suppose cloth particle x is found to be inside a solid object, near distinct surface regions A and B, although closer to A than to B. Applying the GIA algorithm to the solid object's triangle mesh tells us whether or not to flypaper: if portions of the mesh near x on surface A are reported to be intersecting the mesh near x on surface B, then the two surfaces have intersected in the neighborhood of x. The situation is then as shown in the lower right of figure 3, with particle x being pinched between the two intersecting surface regions A and B. In this case, x should be flypapered.

However, if the GIA algorithm reports no such intersections then the situation is as shown in the lower left of figure 3. Here, particle x is merely in contact with a surface whose surface normal varies rapidly, and there is no reason to make use of flypapering. Instead, the usual contact constraint is applied to the particle.

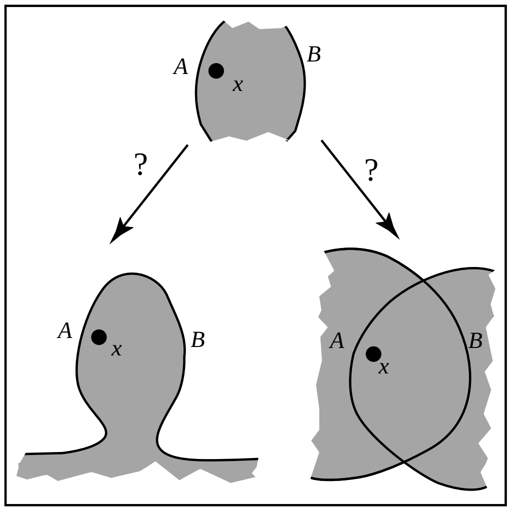

Figure 3: A particle *x* that is interior with respect to both *A* and *B* is either being pinched between intersecting surface (the situation in the lower right) or is merely near a "thin" feature (lower left).

5 Global Intersection Analysis

5.1 Synopsis

The details of our cloth/cloth collision algorithm are split between this section and section 6. Before diving into the details, a short synopsis of the entire method is helpful. Consider figure 4a which shows an intersection curve (in red) between two meshes. To disentangle the two meshes, we want the white vertices on the top mesh (see the exploded view, figure 4b) to pass back through the white vertices of the bottom mesh. If these were cloth meshes, we would want the two sets of white vertices to exchange attractive forces. However, particles on the two meshes that are *not* surrounded by the intersection path should exchange standard repulsive forces.

Similarly, if we have multiple regions of intersection (figure 4c) then we need to set up multiple correspondences. The vertices bounded by the green paths exchange attractive forces to disentangle, as do the vertices bounded by the green paths.

One complication however is that there may be multiple ways of untangling. We can eliminate the intersection in figure 4a either by moving the top of the sphere down through the sheet, or the bottom up through the sheet; we could even move the sheet inside the sphere! The reason for the ambiguity is that the intersection curve on each mesh partitions that mesh into two distinct regions. In figure 4b we have marked the smaller of each region with the white vertices. While this is of course an arbitrary choice, it is the obvious choice for cloth simulation. We are on very safe ground choosing the smaller regions as the interpenetrated ones, since intersections that arise during simulation are generally tiny compared to the sizes of the meshes. (Put another way, this decision will only be wrong if the time step is ridiculously large, i.e. if an intersection is noticed only after it has grown so large that the smaller region is the only unintersected portion of the mesh.)

We summarize the GIA process and then consider some details:

1. Find intersection curves between pairs of meshes (and a mesh with itself).

2. For each curve found, use a flood-fill algorithm to color both sides of the intersection curve, and keep the smaller size. (This is done on both of the intersecting meshes, for each curve).

3. Hand off the sets of corresponding vertices to the dynamics engine, so that it can decide whether or not proximate cloth vertices should attract or repel.

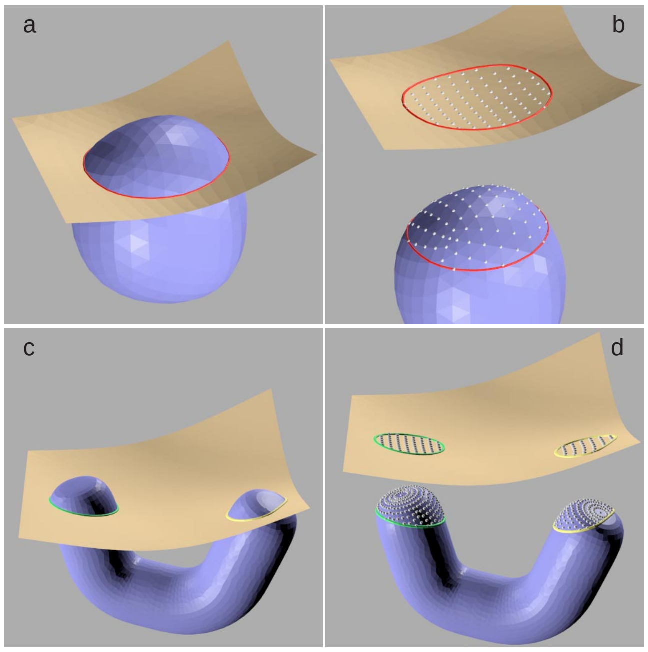

Figure 4: (a) Intersecting meshes with intersection path marked in red. (b) Exploded view. (c) Two separate intersections between a pair of meshes. (d) Exploded view.

5.2 Meshes and Intersections

The meshes we consider in this paper are triangle meshes with manifold geometry. The "curve" of intersection between two such triangle meshes is formed by pairwise intersections of triangles from the two meshes (figure 5). Robustly computing intersections between curved surfaces has been a long-standing concern in computer graphics and is in general quite difficult. For triangle meshes however, it is a rather trivial problem.

Generically, a pair of intersecting triangles generates a line segment of intersection. This line segment begins and ends where a mesh edge intersects a face. We assume that meshes are in general position; that is, no mesh vertex lies exactly in the plane of any non-neighboring mesh face, and no mesh edge intersects any non-neighboring mesh edge (over both its own mesh and all other meshes). We probablistically guarantee general position by adding a small amount of random noise to all vertex positions, and performing computations using standard double-precision arithmetic. Freedom from degeneracy makes the computation of the intersection path simple: finding line segments involves no special geometric cases, since there is no degeneracy, and the geometric operations are trivial (computing line/plane interesections and point/plane distance evaluations).

The only special circumstance not excluded by general position is the situation in figure 5 where the intersection begins at a vertex v shared by the two triangles and ends at an edge/face intersection. We call such a vertex v a *loop vertex* (for reasons that will become clear shortly). Note that this particular case concerns a triangle mesh with self intersections, since triangles A and B are in the same mesh. To emphasize that the intersection curve is describable by finitely many line segments, we will use instead the term *intersection path*. We precompute a hierarchical bounding box tree

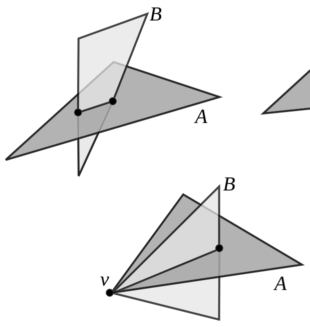

Figure 5: Possible triangle intersections.

[Bridson et al. 2002; Gottschalk et al. 1996] for each cloth mesh at the simulation and update the bounds during simulation to quickly find all edge/triangle intersections.

Ignoring the issue of mesh boundaries momentarily, intersections between a pair of meshes are easily classified. The meshes may not intersect, the meshes may intersect once (figure 4a) or they may intersect multiple times (figure 4c). A mesh may also intersect itself; however, self-intersections come in two very different forms. The self-intersection may result in two closed intersection paths, as shown in figure 6. This occurs when the mesh deforms so that two distinct regions of the mesh intersect, and is essentially the same case as figure 4, except that the two regions happen to be on the same mesh. In contrast however figure 7 shows a self-intersection that results in only one closed intersection path. In this case, the mesh has deformed so that a single region has been folded on top of itself, forming a loop. The intersection path in figure 7 originates from and terminates in a loop vertex (hence the name). Note that this particular case *cannot* arise for smooth (i.e. C_1) surfaces because it would require the folded mesh to become nondifferentiable at the loop vertices. As a result, we have not encountered a description of this case in any previous literature. [2]

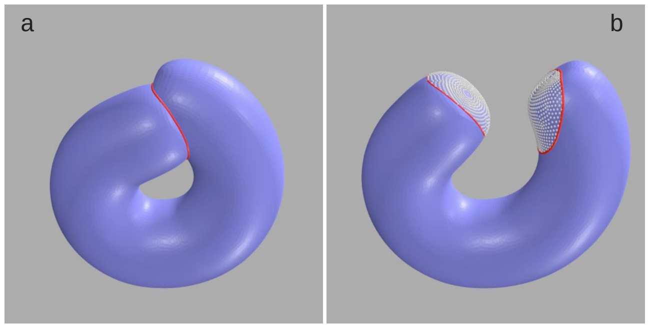

Figure 6: (a) Self-intersection. (b) By unbending the mesh somewhat, two distinct intersection paths are seen.

5.3 Finding Intersection Paths

With the above in mind, the analysis of a pair of meshes M_1 and M_2 (with $M_1 = M_2$ when checking for self-intersections) begins as follows. We start by tracing an intersection path between M_1 and M_2. Intersection paths are found by first checking all mesh edges of mesh M_1 to see if they intersect any triangle faces of M_2 and vice versa. Standard hierarchical bounding volume schemes are used to find all such intersections quickly, and this is computationally inexpensive. Assuming that there are face/edge intersections, there is at least one intersection path. Choosing an arbitrary edge/face intersection as a starting point, we begin to walk along the intersection path in a consistent direction, consuming edge/face intersections until we arrive back at the initial edge/face intersection. At this point, we have completely explored this particular intersection path. If there are any remaining unvisited edge/face intersections, we choose one such intersection as a new starting point and trace another path. This continues until all intersection paths have been found. For meshes with boundaries (for example, the sheet in figure 4a), an intersection path can begin or end on a mesh boundary. If the traversal of the path encounters a boundary, it simply completes the path by returning to the starting edge/face intersection and walking the path in the opposite direction.

[2]It is not surprising that this case has not been encountered before; intersection of triangular meshes is a trivial result. Most research on surface intersections involves C_1 surfaces. Patrikalakis [1993] gives a survey of work and results in this area; also, see Krishnan and Manocha [1997] for a basic characterization of the types of intersection usually studied.

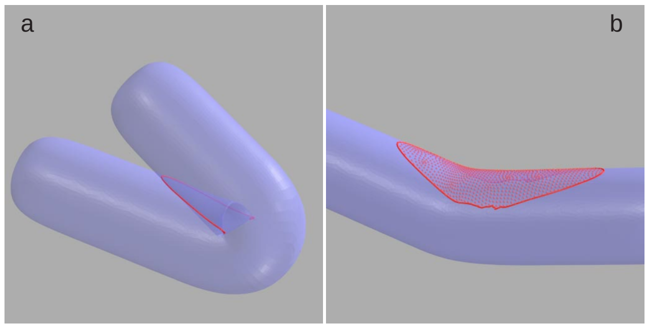

Figure 7: (a) A severely bent arm, with self-intersection. (b) Unbending the arm to better see the intersection path, reveals that the intersection path consists of only a single closed path.

If the intersection region is as shown in figure 7 then the intersection path both begins and ends with a loop vertex. (Because of the nondegeneracy assumption, the only way for the path to reverse direction and begin to double-back on itself in three space is by a loop vertex, as shown in figure 5. As shown in figure 7a, when folded the path appears to have two ends, which are in reality the two locations where the path reverses direction. Thus, there are two loop vertices.) Assuming that the search for the intersection path starts on the path between the two loop vertices, we will walk the path until we terminate at one loop vertex, return to the start, and walk the path until the other loop vertex is reached. The two searches are pieced together to form one path that begins and ends at a loop vertex. In the case of a mesh with boundaries, the path can have one loop-vertex if the path begins and ends on a mesh boundary.

5.4 Mesh Coloring

After each intersection path is found we perform mesh coloring, using a standard flood-fill algorithm, to decide which vertices are the interior of the intersection path. As we noted earlier, it is topologically ambiguous which side we should perform the flood-fill on, so we elect to choose the smaller region as the interior. This requires that we perform two flood fills, measure the area of the mesh visited by both, and declare the smaller such region the interior. Visiting the entire mesh for each intersection path (there can be many) is needlessly expensive however. A simple speedup is to perform both flood fills simultaneously, advancing each one vertex at a time and then continuing with the other. Which ever finishes first has explored the smaller region. Of course, this assumes that mesh triangles are approximately the same size. If not, each vertex encountered represents some amount of the surface explored. The flood fills are alternated when the active fill exceeds the area found by the inactive fill.

Note that boundary/boundary intersections between meshes can result in intersection paths that are not closed *and* do not terminate on a boundary. Such intersection paths do not partition the mesh; this is detected by the two flood fills attempting to fill the same portion of the mesh. Cloth/cloth intersections that define a curve (but not an area) of intersection are problematic. Our current system forms no strategy for actively trying to untangle boundary intersections; this is clearly an area calling for additional development.

Suppose that we are currently examining the ith intersection path. If this intersection path arises from a situation as depicted in either figure 4b or figure 6b, then the intersection path in space traces two distinct closed paths on the mesh or meshes involved. We arbitrarily color the interior vertices of the first path with color "black$_i$" while coloring the interior vertices of the second path with color "white$_i$." Remember that if the intersection is a self-intersection that yields only a single region as in figure 7, when unfolded the path bounds a closed region of the mesh. In this case, all the vertices in the interior of this region are marked as having color "red$_i$." As more intersection paths are discovered, a vertex may be found to belong to multiple regions; accordingly, a vertex is allowed to have an arbitrary number of colors.

All vertices colored black$_i$ have intersected through the corresponding vertices colored white$_i$ and vice versa; the vertices colored red$_i$ have passed through themselves to form a looping self-intersection of the mesh. The colors assigned to the vertices are the output of the GIA method; in the next section we describe how to make use of this coloring information to actively untangle cloth.

6 Cloth-to-Cloth Collision Response

Given the information provided by GIA, the final implementation of our cloth-to-cloth collision-response algorithm is very simple. Once the intersections between cloth meshes have been found using GIA, we handle cloth/cloth collisions by introducing interaction forces between a cloth particle p and a nearby cloth triangle T.

The GIA data determine the interaction forces as follows:

1. If for some i the particle p has been colored black$_i$ and all three particles in the cloth triangle T have been colored white$_i$ then p and T inhabit cloth regions that intersected one another. Accordingly, p and T are attracted so as to remove the intersection. Symmetrically, the same occurs if p has been colored white$_i$ and T has been colored black$_i$.

2. If p has been colored red$_i$, for some i, and any particle in T has been colored red$_i$, then those two particles are on a portion of cloth which has self-intersected in the manner of figure 7. Neither attraction nor repulsion is applied (neither is necessarily correct)—instead, this section of the cloth is free to move through itself in any manner. Typically other sections of the cloth which do not fall into this category will move to pull the cloth back apart.

3. If none of the above hold, then p and T inhabit regions of the cloth which have not intersected. A repulsive interaction force is applied.

Note that a mesh which self intersects, yielding white and black colored vertices will exert forces to repair itself; it is only self intersections of a mesh which lead to "red" coloring that lead to passive behavior on the part of the mesh. Self intersections leading to "red" coloring as in figure 7 often happen when cloth is pinched in bent elbows, knees, and under arms. Because each one these intersections forms a single connected region on the same mesh, it is difficult to apply non-conflicting interaction forces that untangle the cloth. In particular, it is manifestly ambiguous which portions of such an intersection region need to move back through each other to untangle. By doing nothing, we prevent the cloth from snagging on itself; forces on the rest of the cloth tend to quickly untangle the mesh as soon as an opportunity presents itself (figure 10e).

Clearly, there are no proofs or guarantees to be had with the above algorithm. Whenever one is forced into an intersection, there will be a period of time during which the mesh must attempt to untangle. While approaches which guarantee correct behavior such as Bridson et al.'s [2002] are naturally more appealing from a theoretical perspective, they place serious conditions on the input animation and solid surfaces. Given that solid meshes with serious intersections are currently a fact of life, algorithms which do their best to repair the "damage," when given the chance, seem to be the only practical choice at present.

7 Implementation and Results

The cloth/cloth and flypapering algorithms described in this paper were used extensively for dynamics simulations of cloth and fur in Pixar/Disney's *Monsters, Inc.*. The vast majority of the simulations run for this movie delivered acceptable results with little to no "tweaking"; in fact, most resimulations were for artistic reasons, and not due to simulation defects. Note that to control costs, it is vital in a production environment to produce an acceptable result with a very small number of simulation runs. The video examples for this paper are all the results of simulations that ran to completion the first time, without any adjustment of simulation parameters or input. Running time is also critical; however, the additional cost of using flypapering and GIA is negligible. For a garment with 18K vertices in a typical shot, this translates into an additional cost of less than 0.5 seconds per frame of animation, running on a 2Ghz Pentium-4 processor.

Computational geometry algorithms are sometimes difficult to implement from the standpoint of adjusting tolerances and dealing with boundary cases. The methods described in this paper, happily, require no such adjustment. In fact, the only continuum decisions made are based on point/plane evaluations, where only the sign of the result matters. The assumption of general position guarantees that each of these evaluations will be nonzero.

Figures 1, 8, 9, and 10 show simulation output making use of GIA and collision flypapering. In figure 8a, a shirt is tortured without making use of GIA until it is in a tangled and pinched state. When GIA is turned on in figure 8b the shirt immediately untangles. Figure 9 shows Boo rubbing her stomach with different flypapering weights applied to her hands. The same animated motion yields a distinct variety of shirt motions. Figure 10 shows Boo pulling her arm in tightly against her body until a large amount of intersection occurs; however, the resulting cloth motion does not betray any cloth intersections from the camera's viewpoint.

Of course, even with flypapering and GIA there are limits to what the simulator will tolerate. Solid mesh intersections in which a limb passes halfway through another limb will pull the cloth off the character. Similarly, insufficient cloth resolution (when cloth triangles become large compared with the diameter of a limb) can also produce bad results. As noted in section 5.4, boundary/boundary intersections (e.g. cuff-to-cuff intersections between sleeves) are still not handled particularly well in our current implementation. Cloth simulation for animation, even with the improvements described in this paper, still takes skill and experience to achieve high-quality results. However, the collision response algorithms described by this paper tolerate substantially worse solid mesh intersections than any other simulation systems that the authors are aware of. In combination, GIA and flypapering handle the extreme pinching typical of production character animation, while avoiding worrisome visual artifacts such as wiggling cloth and persistent tangles.

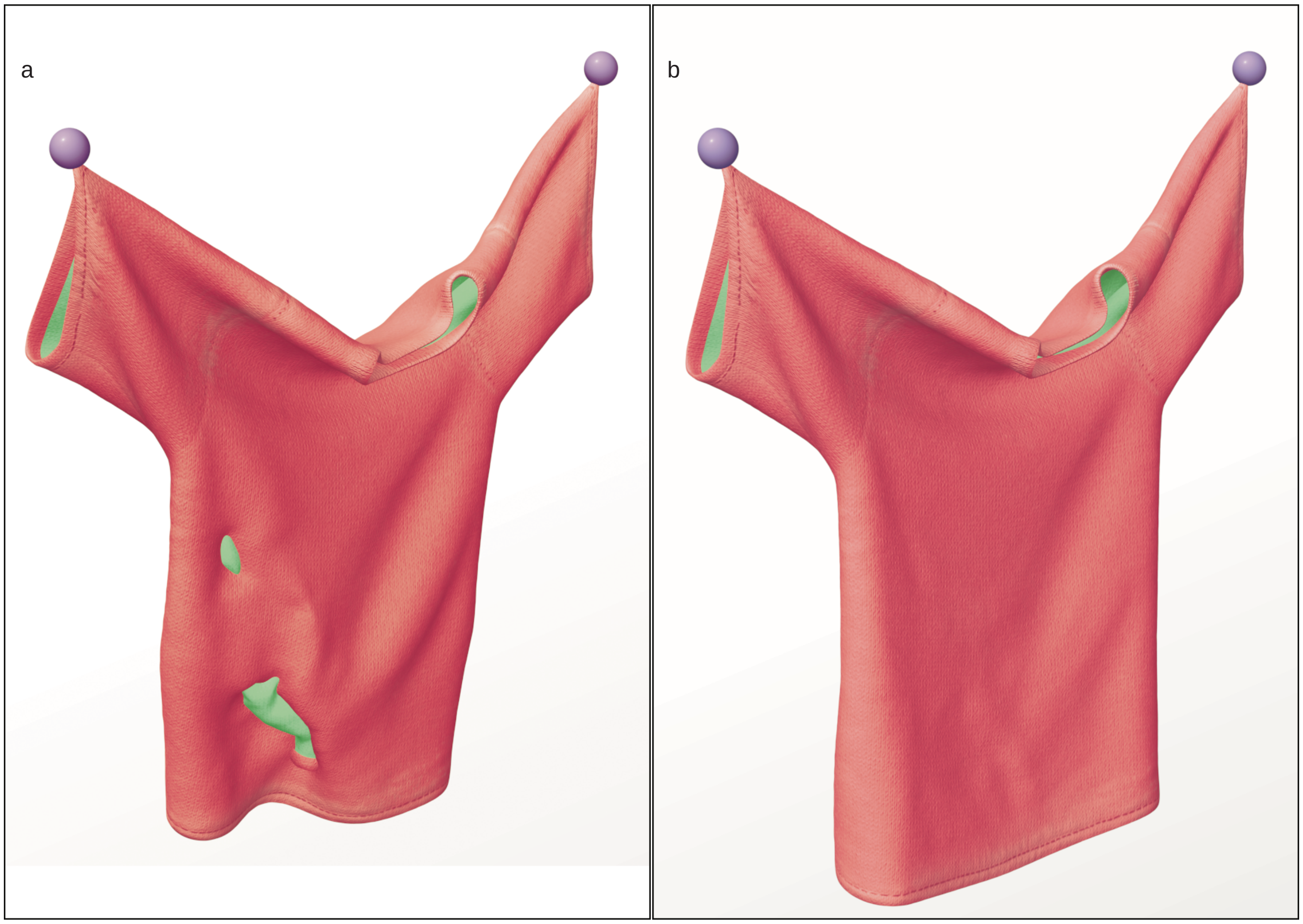

Figure 8: (a) Tortured shirt without benefit of GIA to disambiguate intersections. (b) Starting with the initial conditions in (a), GIA is turned on and the shirt immediately recovers.

References

ASCHER, U., AND BOXERMAN, E. 2002. On the modied conjugate gradient method in cloth simulation. *(submitted to) The Visual Computer*.

BARAFF, D., AND WITKIN, A. 1998. Large steps in cloth simulation. *Computer Graphics* (*Proc. SIGGRAPH*), 43–54.

BERNEY, J., AND REDD, J. 2000. *Stuart Little*. SIGGRAPH Course Notes, ACM SIGGRAPH, ch. Costumes.

BREEN, D., HOUSE, D., AND WOZNY, M. 1994. Predicting the drape of woven cloth using interacting particles. *Computer Graphics* (*Proc. SIGGRAPH*), 365–372.

BRIDSON, R., FEDKIW, R., AND ANDERSON, J. 2002. Robust treatment of collisions, contact, and friction for cloth animation. *Computer Graphics* (*Proc. SIGGRAPH*), 594–603.

CARIGNAN, M., YANG, Y., MAGENENAT-THALMANN, N., AND THALMANN, D. 1992. Dressing animated synthetic actors with complex deformable clothes. *Computer Graphics* (*Proc. SIGGRAPH*), 99–104.

CHOI, K., AND KO, H. 2002. Stable but responsive cloth. *Computer Graphics* (*Proc. SIGGRAPH*), 604–611.

CORDIER, F., VOLINO, P., AND THALMANN, N. 2002. Integrating deformations between bodies and clothes. *The Journal of Visualization and Computer Animation 12*, 1, 45–53.

DEROSE, T., KASS, M., AND TRUON, T. 1998. Subdivision surfaces in computer animation. *Computer Graphics* (*Proc. SIGGRAPH*), 85–94.

EBERHARDT, B., WEBER, A., AND STRASSER, W. 1996. A fast, flexible, particle-system model for cloth draping. *IEEE Computer Graphics and Applications 16*, 52–59.

GOTTSCHALK, S., LIN, M., AND MANOCHA, D. 1996. OBB-Tree: A hierarchical structure for rapid interference detection. *Computer Graphics* (*Proc. SIGGRAPH*), 171–180.

KRISHNAN, S., AND MANOCHA, D. 1997. An efficient surface intersection algorithm based on lower-Dimensional formulation. *ACM Transactions on Graphics 16*, 1 (Jan.), 74–106. ISSN 0730-0301.

MEYER, M., DEBUNNE, G., DESBRUN, M., AND BARR, A. 2001. Interactive animation of cloth-like objects in virtual reality. *The Journal of Visualization and Computer Animation 12*, 1, 1–12.

PATRIKALAKIS, N. 1993. Surface-to-surface intersections. *IEEE Computer Graphics and Applications 13*, 1, 89–95.

PROVOT, X. 1995. Deformation constraints in a mass-spring model to describe rigid cloth behavior. In *Graphics Interface*, Graphics Interface, 147–155.

TERZOPOULOS, D., AND FLEISCHER, K. 1988. Deformable models. *Visual Computer 4*, 306–331.

TERZOPOULOS, D., PLATT, J., BARR, A., AND FLEISCHER, K. 1987. Elastically deformable models. *Computer Graphics* (*Proc. SIGGRAPH*) *21*, 205–214.

VOLINO, P., COURCHESNE, M., AND MAGNENAT THALMANN, N. 1995. Versatile and efficient techniques for simulating cloth and other deformable objects. *Computer Graphics* (*Proc. SIGGRAPH*), 137–144.

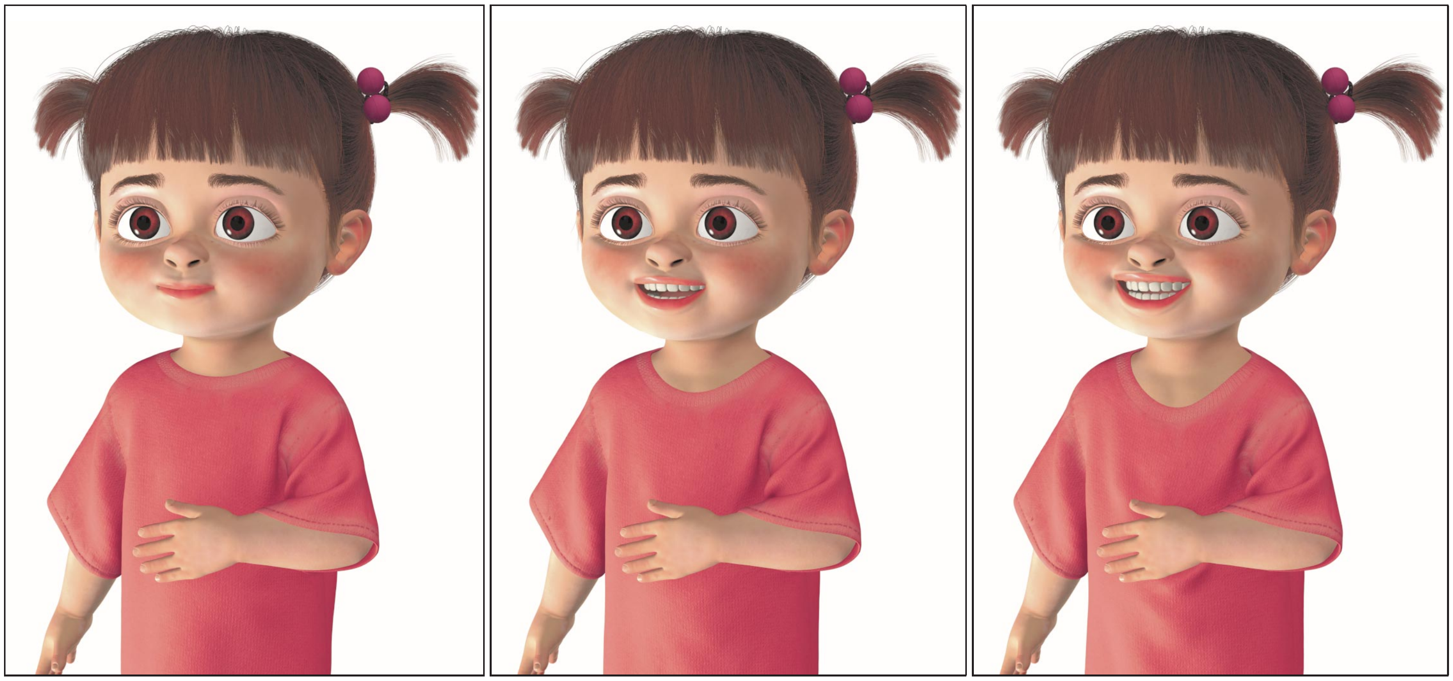

Figure 9: The same animated hand motion but with different flypapering weightings on the hands. From left to right, the hands are weighted at $\alpha = 10\%$, $\alpha = 50\%$ and $\alpha = 100\%$.

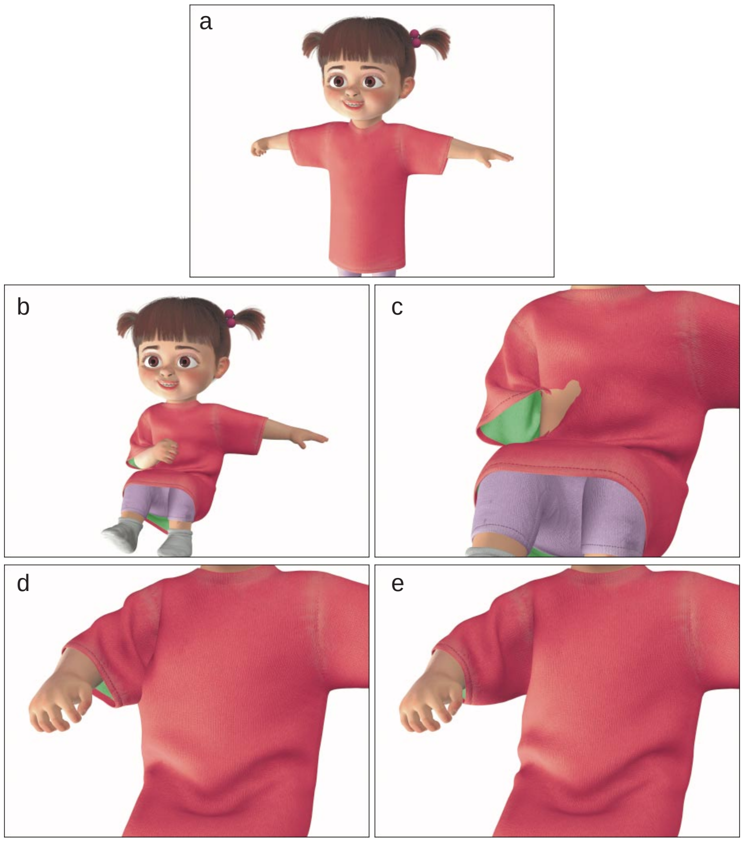

Figure 10: (a) Starting pose. (b) Arm moves in tightly. (c) Close-up view of (b) with right arm invisible. Note how the arm position forces cloth to intersect both itself and the body. (d) Without GIA, a cloth/cloth intersection persists as the arm pulls out, snagging the sleeve. (e) The same frame as (d), but using GIA, the cloth doesn't snag as the arm pulls out.