# Field-based approaches for the collision-free animation of layered and dynamic clothing

Thomas Buffet

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2021IPPAX008

Thèse de doctorat

# Field-based approaches for the collision-free animation of layered and dynamic clothing

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à l'Ecole polytechnique

École doctorale n°626 Ecole doctorale de l'Institut Polytechnique de Paris (ED IP Paris)
Spécialité de doctorat : Informatique

Thèse présentée et soutenue à Palaiseau, le 13 janvier 2021, par

**THOMAS BUFFET**

Composition du Jury :

| | |
|---|---|
| Tamy Boubekeur<br>Professeur, Telecom Paris (LIX), Adobe 3D & Immersive | Président |
| Maud Marchal<br>Maitre de conférences, HDR, Université de Rennes, INSA (IRISA) | Rapportrice |
| Sylvain Lefèbvre<br>Directeur de recherche, INRIA (Loria) | Rapporteur |
| Loïc Barthe<br>Professeur, Université Toulouse III Paul Sabatier (IRIT) | Examinateur |
| Etienne Vouga<br>Assistant Professor, University of Texas, Austin (ICES) | Examinateur |
| Cédric Zanni<br>Maitre de conférences, Université de Loraine, Mines Nancy (Loria) | Examinateur |
| Damien Rohmer<br>Professeur, Ecole polytechnique (LIX) | Directeur de thèse |
| Marie-Paule Cani<br>Professeur, Ecole polytechnique (LIX) | Co-directrice de thèse |

# Table of Contents

# Introduction

Fig. 1: Examples of dressed characters from video-game or movie production.

When animating a scene in most industrial production, a special care is given to the modeling of contacts between objects. In the real world, dynamic objects collide with each other, and different phenomena -such as bouncing or friction- impact their dynamics and their appearance. However, in 3D graphics, objects are not natively following such behavior. Indeed, the notion of contact between virtual objects is not trivial, and they can perfectly visually penetrate each other which causes visual artifacts and will also be wrong in term of the subsequent dynamics of the scene. In that case, such objects should be detected and processed as *in collision*. As such, a whole research area has been to *handle collisions* between virtual objects. This include the *detection* of actual or impending collision between objects, and a way to actually *correct* or *prevent* them, while trying to mimic real world behavior.
In this topic, three main types of objects can be differentiated : rigid , volumetric bodies, deformable volumetric bodies and deformable surfaces such as thin-shell or clothes (we do not consider the case of strands, which are even more intricate to consider and are handled by very specific method, in the context of hair interaction for instance). For the first type, while it brings its own challenge, handling collision is made easier by the fact that these objects can be defined by only three sets of parameters : their shape, their position and their orientation. For instance, two rigid spheres of radius $r$ and of respective centers $c_1$ and $c_2$ will be colliding if and only if $||c_2 - c_1|| < 2r$. Collision with or between deformable volumes becomes harder to solve as far more parameters are to be taken into account : the shape of such body changes during simulation, becoming

a set of interconnected parameters. However, collisions are still handled pretty easily thanks to the fact that these objects are volumetric : it can often be determined easily if a part of an object is actually inside another volumetric one, and using some kind of penetration depth, some global displacement -usually including some deformation to adequately model contact- can be performed.
While collisions of deformable surfaces with volumetric bodies can take advantages of some of these symplifying assumptions, collisions between two deformable surfaces do not benefit from them : there is no easy way to tell what side of a surface something is located, penetration depth is ill-defined, and we often have to resort to more tedious approaches to actually handle the collision of such objects. In this work, I focus on the collision of clothes, which fall in this last category of objects. More precisely, I will restrain to clothes modeled by triangle meshes as it is one of the most used representation.
Three main issues have been tackled in the past decades regarding collision handling for clothes. The first issue is efficient collision detection between deformables surfaces. As mentioned above, no or only few simplifying assumptions exist for such objects, and existing methods rely mostly on one scheme : iterating over each pair of triangles in the scene and perform elementary tests between them to decide if they are intersecting. Because this is computationnaly expensive, these methods also rely on dedicated routines to actually reduce the number of pairs to test, to which a whole branch of research has dedicated itself. The second issue to be noted is the one of the order of correction : because naïvely trying to correct an intersection might worsen another, of even create a new one, special care has to be given to this matter. It is often dealt with using iterative method, along with some kind of fail-safe if no satisfactory correction has been found in a few iterations. Finally, the last issue is the one of modeling the contact between the clothes in a plausible way : not only the collisions need to be solved, but in order to correspond to how real clothes behave when in contact, phenomena such as friction also need to be modelled.
In this work, I focus on the first two of these issues : we propose two approaches to the intersection-free animation of clothes, using volumetric fields (including both scalar field and vector field). We stand appart from existing methods as no primitive tests are performed over the triangles of the objects. As such, while our methods also scale on other parameters, they scale only linearly with the number of triangles in the scenes. Moreover, we do not split the handling of collision between some detection and then some correction, and handle it instead in a single-pass, without having to care about any order of resolution.
I will discuss in more details the existing techniques in the chapter 1, first introducing the collision handling technique for deformables meshes in section 1.1, and how both scalar field and vector field have been used in 3D modeling and animation in section 1.2. The next two chapters will then be focused on our contributions :

- The first method is dedicated to the static untangling of layers of garments using scalar fields. Garments meshes are first approximated by an implicit surface, defined as the iso-surface of a scalar-field. Then, those surfaces are combined using a n-ary operator we defined specifically for this problem. Finally, each garment's geometry

is projected on a new position, ensuring that intersections between them has been solved and replaced by a contact situation. Chapter 2 will be dedicated to this method.

- The second work adresses the dynamic animation of clothes. We handle collision by embedding the clothes inside a vector field discretized in a 3D grid which is filled with the local velocities of the vertices of the cloth. Then, we solve for a least-square problem constraining the velocities to be divergence-free, along with smoothing constraint, which modify the trajectories of the vertices of the clothes in order to prevent collision when needed. Chapter 3 will focus on this approach.

**Publications**

**Vector field based collision avoidance between arbitrary number of dynamic shapes**
In preparation
Thomas Buffet, Damien Rohmer, Loïc Barthe, Marie-Paule Cani

**Implicit Untangling : A Robust Method for Modeling Layered Clothing**
*ACM Transaction On Graphics (TOG) 2019, Special issue SIGGRAPH 2019*
Thomas Buffet, Damien Rohmer, Loïc Barthe, Laurence Boissieux, Marie-Paule Cani

**Untangling Layered Garments : An Implicit Approach**
*ACM Symposium on Computer Animation (SCA) 2018, Poster*
Thomas Buffet, Damien Rohmer, Marie-Paule Cani

**Une approche implicite pour l'animation de vêtement**
*Journées du Groupe du Travail en Modélisation Géométrique (GTMG) 2018*
Thomas Buffet, Damien Rohmer, Marie-Paule Cani


Fig. 2: The most frequent examples of cloth in Computer Animation are the garments worn by virtual characters. The mesh of this dress (courtesy of Laurence Boissieux) contains tens of thousands of vertices.

# Chapter 1

# Related work

In this chapter, we will first cover works related to the problem of collision handling between cloths. We will then focus on the usage of volumetric field in 3D modeling and animation.

## 1.1 Handling collisions for cloth animations

We will review how collision processing has been performed for clothes in the past few years. We will first discuss some fundamental notions about collision detection and response used by the vast majority of methods, and will then focus more precisely on two different problems: the prevention of collision when an initial intersection-free configuration is available, and if not, its correction using only the knowledge of the current configuration.

### 1.1.1 Fundamental notions

#### Discrete triangle/triangle intersection test

As mentioned in the introduction, detecting collision in a classical way is done with pairwise tests over the triangles of the scene. More precisely, in order to decide if two triangles are intersecting, one shall check if any of the edges of one triangle intersect the other triangle. This can be done by first computing the intersection of an edge with the plane containing the triangle, and then checking if this intersection is in fact contained inside the triangle by computing its barycentric coordinates and check where they fall. Iterating over all edges of the two triangles is necessary in order not to miss any collision (cf fig 1.1).

Some optimizations are possible: for instance, one can note that when checking if an edge intersects a triangle, a particular discarding condition is to first check if both vertices at the end of the edge are on opposite side of the plane of the triangle. This amounts to two dot-products, which already improves the computation compared to the naïve approach in case of non-intersection. Nonetheless, the full-test of intersecting triangles remains a succession of several geometric tests, and as such can be considered as a non-trivial matter. Overall, such computation can get particularly heavy especially if all possible pairs of triangles have to be tested : let $n$ be the total number of triangles,

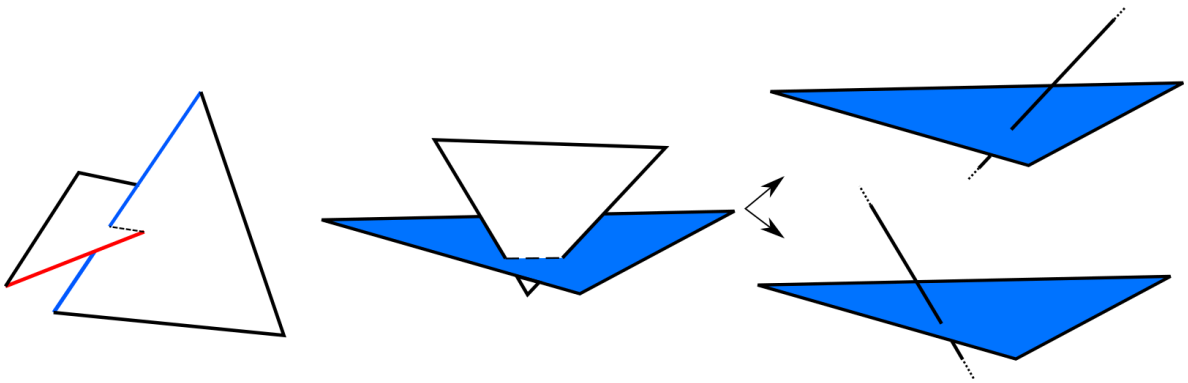

Fig. 1.1: Two different types of intersection of triangles. On the right, the second one is decomposed into the two line/triangle tests that detected it.

testing all possible pairs amounts to a quadratic $O(n^2)$ algorithmic complexity. Because cloth simulations typically counts tens or hundred of thousands of triangle, this is not tractable. In fact, a lot of effort has been put into being able to reduce the number of pair of triangle to test, as discussed in the next section.

#### Acceleration schemes

In order to reduce the number of triangle pairs to be tested, so called *broad phases* have been developped. Taking place before the *narrow phase* of checking the intersection between a set of pairs of triangles, the broad phase aims at discarding pairs that are "clearly" not intersecting, based on different heuristics. A lot of these are based upon bounding-volumes and bounding volume-hierarchies (cf Fig 1.2). These hierarchies are typically tree-structures whose leaves bound primitives (in our case, triangles) and whose progressively coarser levels bound aggregate, enabling groups of primitives to be quickly tested against other groups, without having to iterate over each primitive [KHM+98].

In addition to this, other and more specific schemes have been considered (1.3). For instance, Volino et al. [VCMT95] noted that if curvature of a cloth is locally-low, then local triangles cannot intersect each other, effectively discarding some local pairs when testing for self-collisions. [WLH+13] fires rays from a set of observers defined beforehand, and detect invert-oriented triangles (improved, adapted to clothes and implemented on GPU over [WLW+14] and [WC14]). Tang et al. [TCYM08] first improved the curvature based scheme, and introduced the notion of *orphan-set* to drastically reduce the number of tests between adjacent triangles. [GRLM03] uses visibility computations to provide *potentially colliding sets* based on the rendering of the scene.

Altogether, such broad culling phases have been a prevalent research area, as it is vital for the animation to effectively reduce the number of primitive tests. Typically, while

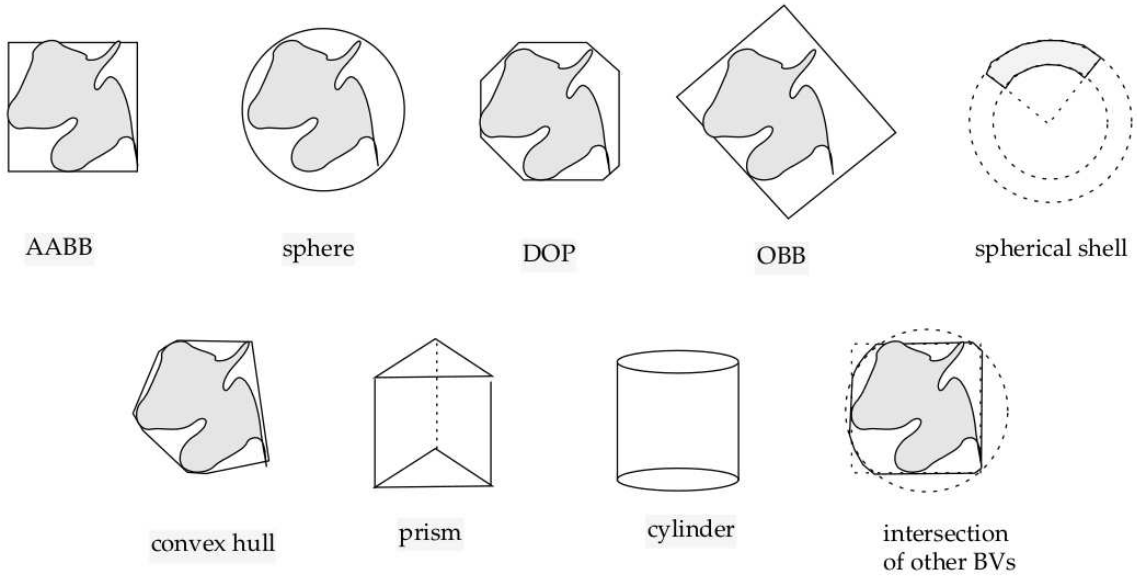


Fig. 1.2: Different bounding volume (from [FH05])

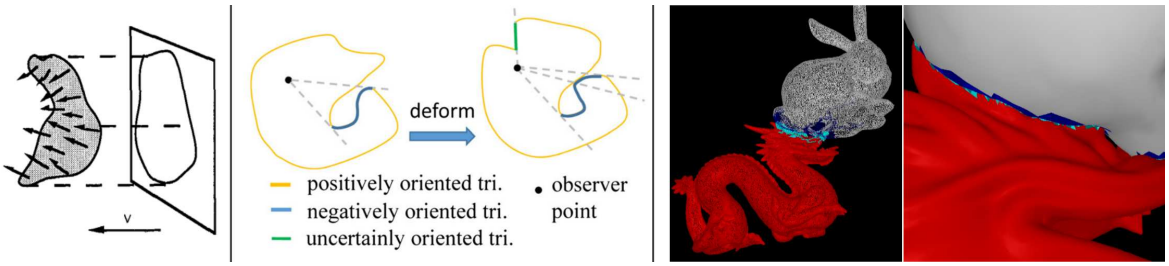


Fig. 1.3: Exemple of culling phase : curvature based (left, from [VCMT95]), radial-view culling(middle, from [WC14]), and rendering based (right, from [GRLM03])

the algorithmic complexity effectively achieved varies with the method, and is some time not trivially derivable, it is safe to assume that using such acceleration method actually reduces the collision detection complexity from $O(n^2)$ to $O(nlog(n))$, which is the actual complexity when "simply" using bounding volume hierarchies. For the rest of this section, whenever a test between two triangles is mentioned, it should be kept in mind that these two triangles have passed the broad phase and have been selected to be tested.

### 1.1.2 Preventing collision

A sub-part of the collision handling problem is the one of preventing the intesections between objects : given an intersection-free state, one shall guarantee that at the end of the next time-step of animation, the clothes are still intersection-free. From frame to frame, such guarantee ensures that a full animation can then be computed without intersection at any time.

#### The "response" scheme

To this end, one of the most popular pipeline is the one described in [BFA02], and I will describe it in more details as a lot of methods appearing afterwards are actually based on this one.
In this method, a time-step is explicitely splitted into two half time-steps: a simulation part -in which the internal law of the clothes along with external forces are integrated- and a collision handling part. Here is the full algorithm as described in the paper :
Starting from time $t = 0$, and with cloth position $x^0$ and velocity $v^0$,

- (1) Select a collision time step size $\Delta t$ and set $t^{n+1} = t^n + \Delta t$
- (2) Advance to candidate positions $\bar{x}^{n+1}$ and velocities $\bar{v}^{n+1}$ at time $t^{n+1}$ with the cloth internal dynamics
- (3) Compute the average velocity $\bar{v}^{n+\frac{1}{2}} = (\bar{x}^{n+1} - x^n)/\Delta t$
- (4) Check $x^n$ for proximity, then apply repulsion impulses and friction to the average velocity to get $\tilde{v}^{n+\frac{1}{2}}$
- (5) Check linear trajectories from $x^n$ with $\tilde{v}^{n+\frac{1}{2}}$ for collisions, resolving them with a final midstep velocity $v^{n+\frac{1}{2}}$
- (6) Compute the final positions $x^{n+1} = x^n + \Delta t v^{n+\frac{1}{2}}$
- (7) If there were no repulsions or collisions, set $v^{n+1} = \bar{v}^{n+1}$

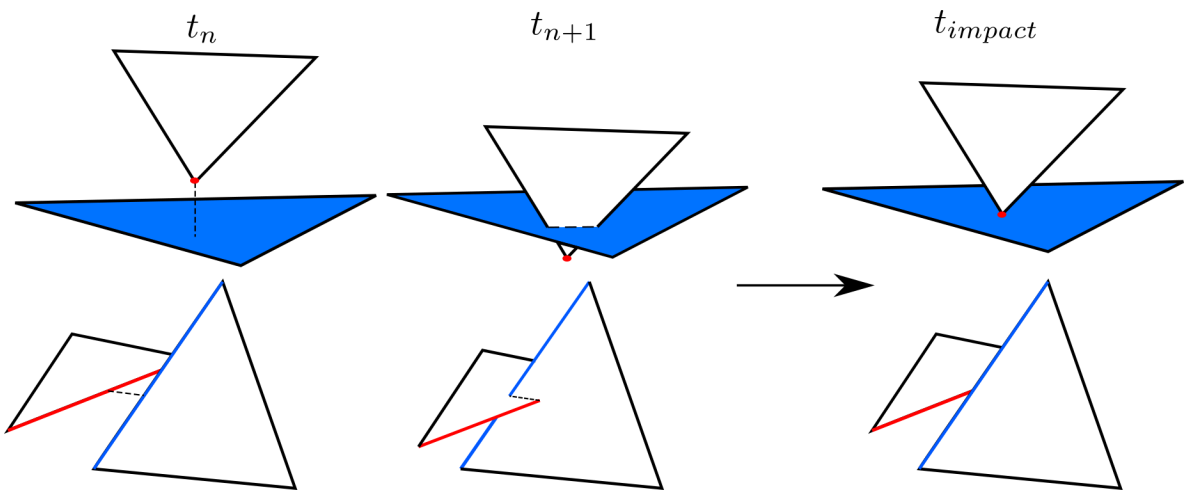


Fig. 1.4: CCD aims at detecting collision by finding the exact time of impact between primitives.

- (7b) Else, advance the midstep velocity $v^{n+\frac{1}{2}}$ to $v^{n+1} = \bar{v}^{n+1}$ throug an implicit integration.

The collision handling part is divided among the parts (4) to (7b). Future collisions are prevented using proximity queries between pair of triangles. Two types of queries are performed: vertex/face queries and edge/edge queries. If primitives happens to be too close to each other, repulsive forces are used to prevent their collisions. However, collisions might have already been caused by the "naïve" time-stepping performed in (1). Therfore, pairs of triangles are tested in step (5) using Continuous Collision Detection (or CCD). Opposed to the Discrete Intersection Detection mentioned earlier, Continuous Detection aims at computing the exact time at which the collision took place, using the knowledge of the previous position of the primitives (cf 1.4). [Pro97] showed that it amounts to solve a third degree polynomial equation, which is typically an order of magnitude more costly than simply testing for discrete collision [YMJ+17], but gives valuable information about the collision.

After CCD, colliding pairs are given a velocity impulse at (6). Both the repulsion forces and those impulses are then integrated over the half-time step that is left. However, because correcting a collision gives no guarantee that no other collisions are created, this whole collision handling process (namely, the computation of an acceleration structure based on bounding volume hierarchy, a culling phase, proximity tests and CCD) is iterated a few-times, hoping for convergence. As it happens, the repulsion forces are able on their own to prevent a lot of the collisions, but from the experiments reported by the authors some tedious cases are problematic and need a lot of iterations to converge. To circumvent this problem, they use a fail-safe called Rigid Impact Zone (or RIZ). Introduced in [Pro97], this fail-safe catches problematic primitives, for which a few iterations are not enough, and agglomerate them into clusters of primitives spatially close to each others. Then, each cluster is treated as a rigid-body for the whole time-step, for which the velocity and

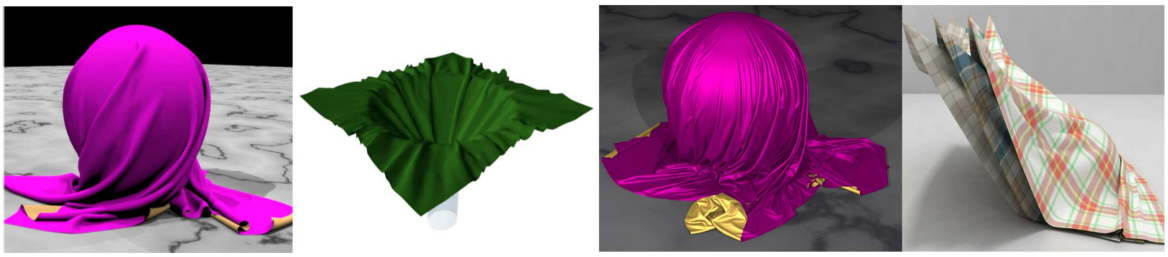

Fig. 1.5: The response scheme introduced by [BFA02] and improved by [HVTG08], [SSIF09] and [LTT+20] robustly handle collisions and friction between clothes.

the angular momentum derive from the velocities at half-the time step of the vertices included in the cluster. This aims at modeling extreme friction behavior caused by the proximity of the triangles in such region, while also guaranteeing that collisions will be resolved in finite time.
This method proved to be a robust way of dealing with collisions, and the "two half-time steps" paradigm became well used in both industry and research.

The collision response part was improved by [HVTG08], and more precisely the Rigid Impact Zone. In this work, a new-formulation based on projection along some constraint gradient enabled for the rigid impact zone to allow sliding or shearing motion, along with the translation and rotation that was already present. This proved efficient to more accurately model friction behavior.
The response scheme introduced by [BFA02] was also the base of the work of Selle et al. [SSIF09]. One of their contribution is to provide a parallelizable expression of the method, achieving the same kind of result in a much faster way. The other is to explicitely compute any inversion of orientation between pairs of triangle between two-time steps, effectively basing themselves on the history of the clothes.
Finally, following on the parallelization process, Li et al. [LTT+20] (succeeding to [TWL+18] and [TWT+16]) efficiently adapted the algorithm to be parallelized over multiple GPUs. Their works contains a GPU-parallelizable formulation for the update of the Bounding Volume Hierarchy used to cull pair of triangles away, an algorithm to assemble their *Contact-aware Matrix* and solve the system associated to it in parallel, and the efficient workload distribution between processes. They achieved results of the same quality than previous work, but at interactive framerate of several frames per seconds.

#### Volume based methods

Based on the two half-time steps paradigm of [BFA02], Sifakis et al. [SMT08] kept continuous collision detection but replaced the collision response part with a volume preserving impulse. This is motivated by the observation of the air behavior between two colliding thin layer, that can act as a cushion.
They identify so-called *stencils*, or quadruplet of vertices, involved in a collision at half

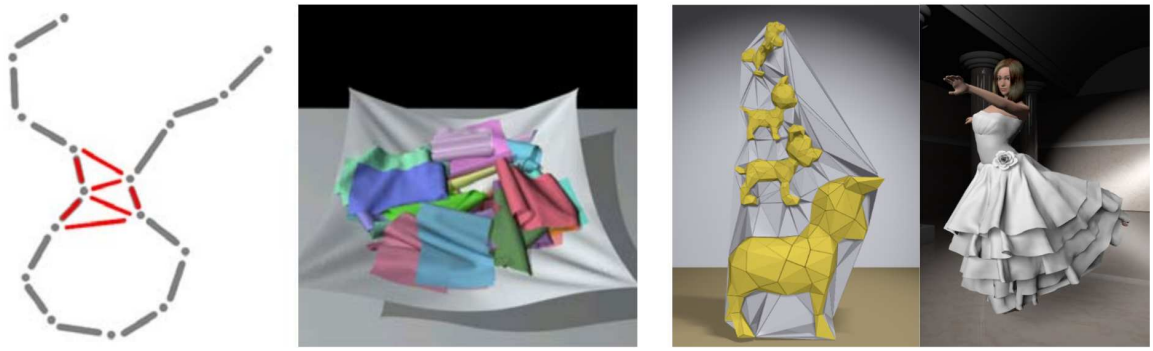

Fig. 1.6: Volume based method such as [SMT08] (left) and [MCKM15] (right) tetrahedrize the space between the objects.

the time step. This quadruplet gives a tetrahedra on which they derive formulas based on pressure to compute an impulse preserving its volume. They obtain a set of non-linear equations solved using iterative solver. Through this method, they found that Rigid Impact Zone fail-safe was no longer needed.
Air meshes are another structure involving tetrahedra [MCKM15]. This work is based on position based dynamics [MHHR07], and express both the cloth internal laws and the handling of collisions as constraints only depending on the positions of vertices. These constraints are then solved sequentially through many Gauss-Seidel iterations, thus effectively mixing the cloth dynamics and the collision processing step (at the opposite of the collision scheme discussed earlier). Contrary to the previous method, this time a full tetrahedrisation of the scene is computed and maintained throughout the simulation. Each of the tetrahedras keep track of a possible inversion of its volume, which converts into a constraint over its 4 vertices. The method is in the continuity of position based approaches: a non-physicial and approximate way to deal with the simulation - though achieving plausible results - in exchange for a pretty low cost of computation.

#### Asynchronous methods

Harmon et al. ([HVS+09]) adapted algorithms from robotics or from rigid body dynamics. This work adapts Kinetic Data Structure (KDS,[BGH99]) for use with Asynchronous Variational Integrator (AVI, [LMO03]) to provide robust collision handling. The paradigm of asynchrony is the following : each object (in our case, each triangle) moves at its own rythm, following a queue of events sorted in *causal order*. When no events -such as collision- occurs during a large period of time, this allows to use high time-step as oversampling time is a waste of computation power. On the other hand, collision regions are handled through a succession of micro-time steps, ensuring robustness and correctness of the result. As acknowledged in the paper, reducing the computational time was not a goal in this work, and their result took hours to make for rather simple simulation (tens of thousands of vertices, and rather short simulations), but the visual realism of their results was astonishing (see the tied knot, figure 1.7).

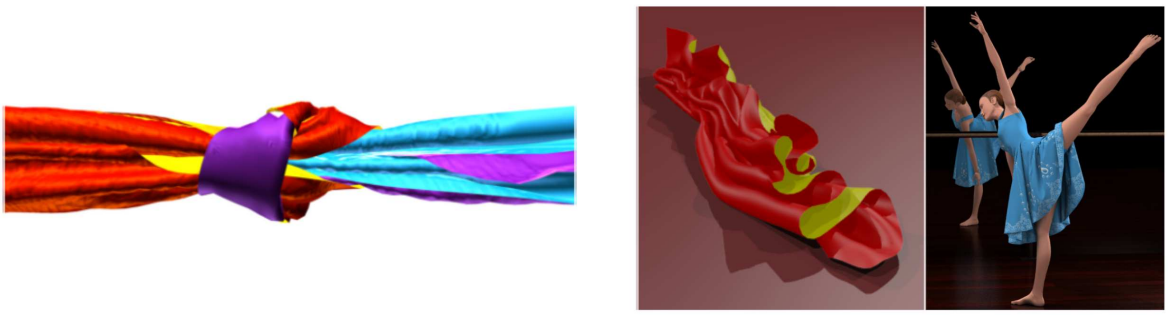

Fig. 1.7: Asynchronous (left, from [HVS+09]) and implicit solvers (right, from [OTSG09] and [LDN+18]) methods focus on hyper-realistic result by extending rigid body mechanics to deformable objects in order to handle contact response.

#### Constraint-based implicit solvers

Also borrowing from rigid-body simulation, [OTSG09] implemented an implicit formulation for contact-handling. In the case of rigid-body, this kind of method achieves robust and efficient collision process, but suffers a complexity explosion when extended to deformable objects, as the degrees of freedoms involved and the number of contacts grow exponentially. They use continuous collision detection, then generate a set of non-linear constraints, some of them insuring non-penetration while others model friction between objects. They then iteratively solve for all the constraints simultaneously, making the method tractable for deformable objects by interlacing relaxation steps.
[LDN+18] notes that the approximations used in this method introduce artificial anisotropic motion for objects in contact, and aims at correcting such behavior. They adaptively refine the mesh (using a method similar to [NSO12]) in order for a new vertex to be situated exactly at every contact-point (while the method previously consider contact point being inside triangles or on edges). They then derive the constraints accordingly (for instance introducing a *pin constraint*), and propose a solver specifically designed for the task.
As for the asynchronous one, this family of methods builds on the robustness of rigid-body mechanics, and adds Coulomb-friction approximation to achieve realistic looking results, at the cost of pretty low framerates. (Several seconds per frame for pretty "simple" meshes composed of only thousands of vertices).

#### Discussions

We presented the main methods dealing with the *prevention* of collisions. Starting from an intersection-free configuration, they all compute some acceleration schemes enabling them to cull away un-necessary tests. They then perform some kind of CCD to identify problematic sets of triangles, and then deal with collision response using a variety of method, modeling friction and preventing inter-penetration in different ways. The

problem of the order of correction is dealt with by either :

- using an iterative method and some fail-safe if no fast-convergence is achieved (this is the case of the "collision scheme" methods, or of the volume-based ones);
- or by expressing and solving simultaneous constraints using rigid-body mechanics methods adapted for this problem (eg. based on asynchrony or implicit solvers).

However, these methods tackle only a sub-part of cloth intersection problem, as they need to start from a situation in which no collision are occuring, else no CCD is possible. Next sections describes methods designed for handling the case of already intersecting or self-intersecting clothes.

### 1.1.3 The untangling problem

The previous approaches start from an intersection-free configuration, and prevent the intersection at each frame of the animation. However, such a configuration is not always available. First, at the beginning of an animation, meshes might be already intersecting. Moreover, external constraints (suh as the limbs of a character colliding themselves and pinchiong the clothes between them) might cause un-preventable intersections between the clothes.
This leads to an ill-defined problem : given an intersection and no prior knowledge about the position of the meshes, one should choose which part of the objects are actually *on the wrong side* of the other one. Also, there are potentially infinite ways to correct an intersection, and while the knowledge of previous state would help defining some repulsion direction, here some other heuristic needs to be used to infer such decision. Those problems might be the reason why this issue has not been tackled as much as the more general prevention of collisions.
While multiple heuristics have been used in various applications, only a few generic approaches have been developed, and we cover them in this section.

#### Global Intersection Analysis

The first method has been introduced by Baraff et al. [BWK03], and is based on *Global Intersection Analysis*, or GIA. This time using only discrete intersection detection between pairs of meshes, and also between each mesh and itself, they agglomerate the geometric intersecting path of the triangles into intersection paths. They differentiate 4 types of intersections :

- the intersection can follow a simple, "circular" intersection path;

- it can follow an intersection path that goes through a loop-vertex (applies only to intersection of a mesh with itself;
- it can follow a path that ends at the boundaries of the meshes;
- or it can follow a path that might not end at the boundaries of the cloth, and that do not loop around.

As depicted in Fig. 1.8, the first three of these intersections trivially partition the meshes into two parts. This is a necessary condition for the rest of the method to work, and because the last type of intersection does not partition the mesh, it is left as a non-handled case in this work. For intersections partitioning the clothes, a flood-fill algorithm is used to color the vertices of the meshes and to compute the area of each part of the partition. From this, the authors are able to define an *inside* region as being the region that is on the wrong side of the other cloth (or of itself): they choose the regions with the smallest area. This is motivated by the fact that, during a simulation with time-steps that are likely to be small, only small interference should happen, or at least small with respect to the side of the mesh.
If an intersection path is of the first or third type, it traces two distinct correspondent closed paths on the mesh or meshes involved (cf figure 1.8, (a) for example). From this correspondance, two inside regions are correspondant if the closed pathes bounding them are correspondant. In that case, one of the regions will be colored in $white_i$ while the other one is colored in $black_i$, with $i$ denoting one specific intersection (it can be noted that a vertex can have any number of colors). For a region defined by an intersection path of the second type (which contains loop-vertices), the region is colored in red. Afterward, all is left in the algorithm is to effectively correct the intersections. To do so, forces are applied at all vertices. Taking a vertex p and a nearby triangle T :

- if p is colored in black (resp. white) and all the vertices of T are colored in white (resp. black), then attractive forces are applied between p and T;
- if p and all the vertices of T are red, then no force is applied, and the those 4 vertices are able to move freely;
- if none of the above apply, then repulsive forces are applied between p and T.

The conjonction of the first and third conditions is supposed to effectively correct intersections that belong to the first and third type of intersections (with no loop-vertices). The third condition alone helps correct the intersections with loop-vertices, with the second condition being motivated by the fact that it is ambiguous to decide where the red vertices should move.
As acknowledge by the authors, there is no guarantee for their iterative scheme to converge, but they could not find counter-examples, and achieved good results overall.
The method was further enhanced by [WLG] who treated the last case of intersections

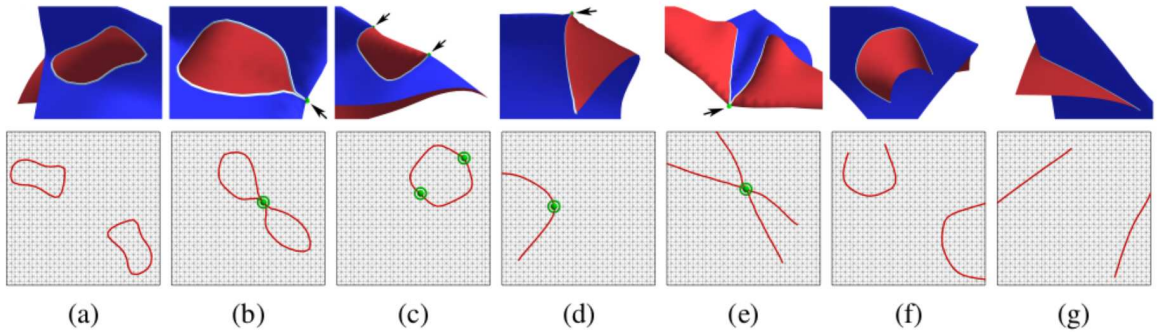


Fig. 1.8: GIA relies on categorizing types of possible intersections, and use the fact that most of them partition the clothes into inside and outside region. Here, (a), (b) and (c) trivially partition the clothes as noted in [BWK03], while (d), (e), (f) and (g) were handled in [WLG] (image from [WLG])

left un-handled by [BWK03]. To do so, they actually make the number of different intersections grow up to 7, with one of them being a degenerate case, and 3 of them being variants of the originally un-handled cases. They also rely on the definition of inside regions in which vertices are attracted to other ones. They notice that in 2 of the 3 variants unhandled by [BWK03], one can actually define an inside region and still apply the algorithm.

While the method in [BWK03] originally introduced attractive forces between all neighboring candidate primitives, this time a more intricate approach is developped. It is motivated by the fact that, using CCD, we are able to compute the exact point at which a vertex has crossed another primitive. This time, the information is not available, except for the set that lies on the intersection path : looking at this in a "CCD-fashion", those points are actually at the exact time of intersection. A sampling of the intersection path is used to obtain boundary conditions for a RBF fitting [BB03]. This gives them a smooth function mapping each point of a region to a point in its corresponding region, and attractive forces are set between these vertices and their associated point. For the last sub-case, for which no inside region can definitely be defined, they basically push the intersection towards the boundaries of the clothes to solve the problem.

#### Intersection contour minimization

The second impactful scheme is introduced in [VMT06]. Here, the intersection contour between the meshes is also considered, albeit in another way. They derive how displacement of the vertices surrounding an intersection influence the intersection path length. They note that reducing this intersection curve's length can be seen as the minimization of an energy. They do so by inducing small displacements of the vertices along its gradient. While each vertex follows its own local gradient, they note that in some cases, it is

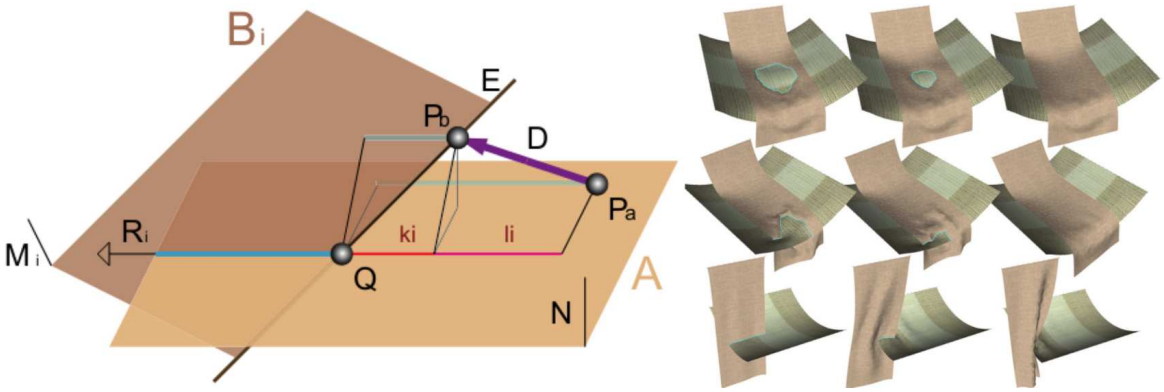


Fig. 1.9: Intersection contour minimization derives a gradient for the length of the intersection contour, and performs gradient descent to minimize it (from [VMT06]).

far more effective to compute a *global* gradient that each vertex follows, which they call a *global* scheme. All this derivation can be adapted to force-based approaches, and thus be implemented inside a pipeline for cloth animation.

Compared to the GIA, this method does not partition the set of possible intersections, and does not rely on the partitioning of the clothes into region. It instead deals with any intersection the same way. However, intersection contour minimization suffers from the expression of the problem as an optimization when dealing with local minima, that they cannot escape from.

Finally, it can be noted that [YMJ+17] implemented a method for dealing with cloth intersection using only discrete collision detection, in which they unify all of the above (GIA and contour minimization) into a single pipeline.

### 1.1.4 Discussions

We presented the main method dealing with static intersection handling for clothes, or *untangling*. Given a tangled configuration, heuristic such as Global Intersection Analysis or Intersection Contour Minimization are used to untangle the meshes. Along with their own problems, these method also suffer from classical drawbacks due to the use of forces to pull appart clothes: these forces need to be tuned, often complemented with additional parameters, which makes the final result hard to control.

Overall, the method exposed in Section 1.1.2 (prevention of intersection) and 1.1.3 (untangling of clothes initially intersected) share similarities. All of these methods test triangle primitives against each others for intersection: because this would typically be of quadratic complexity, a broad phase is performed to cull away unnecessary tests, typically

reducing the complexity of almost an order of magnitude. Moreover, the problem of order of correction is dealt with using iterative method. , and fail-safe when it fails to produce good results in a reasonable amounts of iterations, or using complex and computationnaly heavy procedures derived from rigid-body mechanics.

In this work, we propose two methods tackling the problem of collision handling between clothes which bypass the need of a broad phase and of tests on pairs of primitives for collision detection. Interestingly, we follow the idea of approach such as the volumetric one (see Section 1.1.2). However, while these methods use volumetric structures as a tool, we propose natively volumetric approaches by embedding the clothes in a scalar field for the first approach, and in a vector field for the second one. In next sections, we cover some related works about the use of such types of fields to solve modeling and animation problems in Computer Graphics.

## 1.2 Volumetric fields in 3D modeling and animation

In this section, we briefly focus on the specifities of two type of volumetric field used in modeling and animation : scalar fields and vector fields.

### 1.2.1 Scalar fields and implicit surfaces

A scalar field f can be defined as

$$f : \mathbb{R}^3 \to \mathbb{R}$$

ie function mapping each point in space to a real-value. Given a sufficiently smooth scalar field $f$, and a real value $iso$, an implicit surface $S_{f,iso}$ can be defined as :

$$\{p \in \mathbb{R}^3 | f(p) = iso\}$$

If $f$ is at least $C^1$, one can also consider $\nabla f$ as a vector field directed toward (or away from, depending on the sign of $f - iso$) the implicit surface $S$. Also, for any point $p$ on $S$, $\frac{1}{||\nabla f(p)||}\nabla f(p)$ is actually a normal vector to the surface at $p$.
A typical example of implicit surface is the the definition of a sphere of radius $r$ and of center $p_0$ : let f be defined as $f : p \to ||p - p_0|| - r$, then the aforementioned sphere is the implicit surface defined as $f = 0$.
By convention, two types of implicit surfaces, and of scalar field, are usually used :

- if the scalar field is actually zero everywhere except inside a given boundary, then its image is an interval, usually mapped back to the interval $[0, 1]$, and the $iso$ value usually considered is 0.5. These field are often named *soft-objects* or *metaballs*.
- if not, then the field value typically grows toward infinity as the distance to the surface grows. The $iso$ value considerd in that case will usually be 0. Those field are often refered to as *R-function*.

In practice, the use of one or the other type of field depends on the needs : the first one are bounded, often computationnaly cheaper, and benefits from easy blending as discussed in the next sections; the second one are usually more expensive to compute, but give useful information about the distance to their iso-surface.

#### Collision detection

Implicit surfaces are well known for their ability to ease collision detection. Given a point $p$, the sign of $f(p) - iso$ gives valuable information about the position of $p$ with respect to $S_{f,iso}$. Indeed, we can first note that this surface actually partitions space into two regions

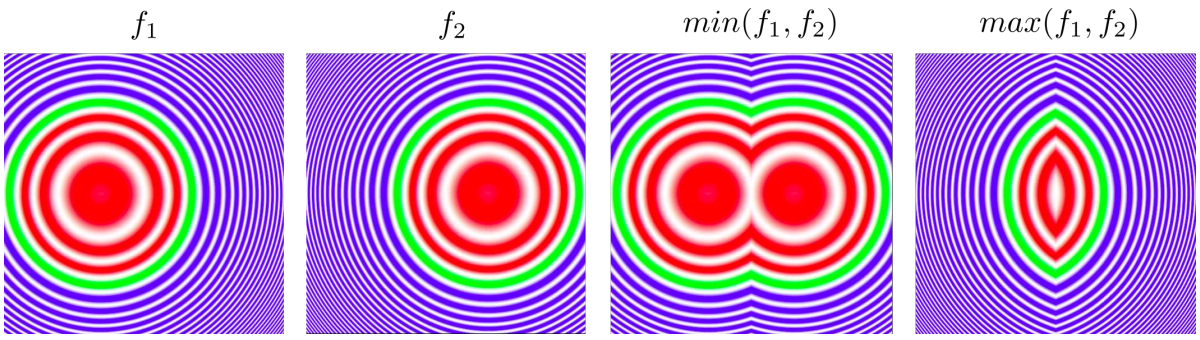


Fig. 1.10: First and second pictures, two implicit circle in green defined as the isolevel of $f_1$ and $f_2$. In red (resp. blue) negative (resp.positive) level-sets of these field. Third and fourth picture, new field have been defined as the minimum (resp. the maximum) of the two initial fields, actually outlining the sharp union (resp. sharp intersection) of the two disk bounded by the intial circles.

: one region in which $f - iso < 0$, and one in which $f - iso > 0$, with the implicit surface for which $f - iso = 0$ being the interface between these two. By convention, we can define the volume *inside* the surface $S_f$ (not always bounded) as the set of point for which $f - iso <= 0$. Then, by definition, the point $p$ is inside the volume if $f(p) - iso <= 0$. This gives a simple test enabling to decide if a point is inside a volume, which does not exist for the volume defined by a closed mesh.

**Composition using closed-form blending operators**

Implicit surfaces are also known for their ability to seamlessly blend into more complex surfaces using simple operators. For instance, given two R-function $f_1$ and $f_2$, a point $p$, and the field $g : p \rightarrow min(f_1(p), f_2(p))$, by definition $p$ will be inside the volume of the implicit surface associated to g if $g(p) < 0$, which is equivalent to $f_1(p) < 0$ *or* $f_2(p) < 0$ : as such, the interior of $S_{g,0}$ is the exact volumetric union of the interior of $S_{f_1,0}$ and $S_{f_2,0}$, as shown in figure 1.10. One can actually describe the same way the intersection of the two surfaces as the maximum of the two associated field. For soft-objects, such behavior can also be observed when interverting the max and min operators.
In general, given two scalar fields $f_1$ and $f_2$, and a binary *operator* $\mathrm{O} : \mathbb{R}^2 \rightarrow \mathbb{R}$, another field $g$ can be created by considering $g : p \rightarrow O(f_1(p), f_2(p))$. For the sake of simplicity, this notation can often be simplified as $g = O(f_1, f_2)$. The min and max operator are the first and most simple examples of analytical composition operator. An obvious shortcoming is that they produce *sharp* union (or intersection) : even if the initial fields $f_1$ and $f_2$ were $C^1$, the resulting field is not smooth everywhere anymore. This is a problem when dealing with subsequent composition, as in CSG modeling. For soft-objects, the operator $+$ can play the role of a smooth union, and Ricci's super elliptic operator [Ric73] actually controls the level of smoothness thanks to an extra parameter (cf. the operator $O_R$ in

$$O_R : (f_1, f_2) \rightarrow \sqrt[n]{f_1^n + f_2^n}$$
$$O_P : (f_1, f_2) \rightarrow f_1 + f_2 + \sqrt{f_1^2 + f_2^2} + \frac{a_0}{1+(\frac{f_1}{a_1})^2+(\frac{f_2}{a_2})^2}$$

Fig. 1.11: Closed form parameterized operators that enable the transition between sharp and smooth union.

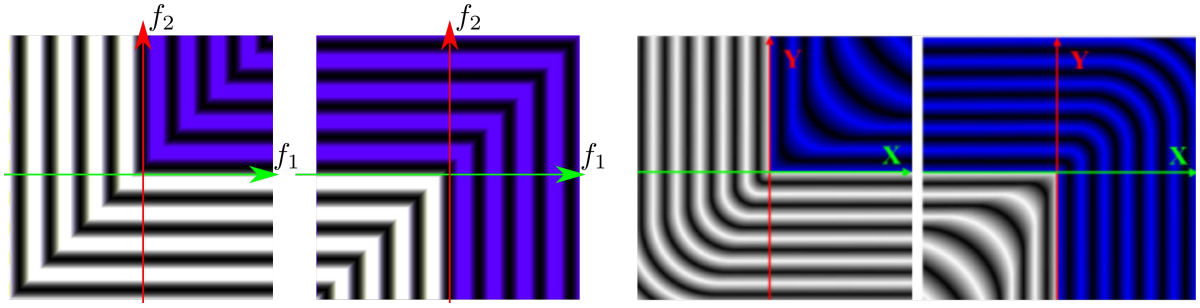


Fig. 1.12: First and second pictures: max and min operators represented in the implicit space by their isolevels. Third and fourth picture : the smooth versions obtained in [BDS+03]. The blue values indicate positive value of the operators, while the white values indicate negative values. While the axis of the figure from [BDS+03] are labeled X and Y, they actually refer to field values.

Figure 1.11). An equivalent formulation for the smooth blending of R-function, along with all the CSG related operations (union, intersection and difference), is proposed in [PASS95a] (cf the operator $O_P$ in figure 1.11). However, the use of extra parameters make these operators intricate to use.

**Building operators using the implicit space**

In order to overcome this issue, methods were developped to allow intuitive building of binary composition operators using the so-called *implicit space*. Introduced by Hoffmann et al. [HH85], each coordinate in this space corresponds to a field, and binary composition operators are visualized thanks to their level-set (cf. Figure 1.12).

An interesting property of operators defined in this space is the following : let $O$ be a binary operator represented in the implicit space, $f_1$ and $f_2$ two R-functions, $g = O(f_1, f_2)$, $S_{f_1,0}$ and $S_{f_2,0}$ associated implicit surface, and let the volume inside the implicit surface be defined as the set of points for which the fields are negative, as a convention. Then, the position of the *iso*-value of $O$ in the implicit space actually dictates the position of the implicit surface $S_{g,iso}$ in the world space. For instance, in Figure 1.10, the 0 iso-value of the *min* operator lies between the blue and white region, more precisely :

- on $f_1 = 0$ if $f_2 > 0$;

- on $f_2 = 0$ if $f_1 > 0$.

This is in fact how the analytical *min* operator is supposed to behave : only the part of the surfaces $S_{f_1,0}$ and $S_{f_2,0}$ that are outside of the other volume are kept in the resulting surface.
This new point of view on composition operator allows to build new operators by directly operating in the implicit space. For instance, Barthe et al. blend implicit lines [BDS+03] or deform existing operators [BWDG04] in order to define novel composition. They thus obtained operators such as a smooth union and a smooth intersection operators, which maintain the $C^1$ continuity of the combined fields everywhere except on the surface (cf Fig 1.12).

#### N-ary composition

In fact, the composition of scalar fields is not limited to binary composition. Some operators trivially extend to N parameters (max/min, the sum, or the super-elliptic blend), and other N-ary methods include blending with range control [dWv09, HL03, Hsu18], set theoretic operations [PASS95a], or extended convolution operators for topology control [ZBQC13].
To the best of our knowledge, no *field-space*-based definition of N-ary operator existed so far (we will introduce one in chapter 2).

#### Contact modeling

Leveraging on the fact that implicit surfaces ease collision detection, and on the modularity of their composition, some work used them to accurately model the contact between objects. While Cani proposed a closed form solution [Can93], it has since then been given more controls thanks to the usage of the implicit space. For instance, Vaillant et al. [VGB+14] define the 0.5 iso-level of their operators as to encapsulate a contact surface between two intersecting implicit surfaces. One of their operators is depicted on figure 1.13, on the far left. Its 0.5 isovalue is depicted in yellow, purple and green : it is in fact the union of three sets of values :

- $\{(f_1, f_2)/f_1 = 0.5 = \& \ f2 < 0.5\}$, which represents the initial surface $S_{f_1,0.5}$ where it does not intersect the other initial implicit surface $S_{f_2,0.5}$;
- $\{(f_1, f_2)/f_2 = 0.5 = \& \ f1 < 0.5\}$, which represents the initial surface $S_{f_2,0.5}$ where it does not intersect the other initial implicit surface $S_{f_1,0.5}$;
- a third set representing the contact surface. This set is located in the region where $f_1 > 0.5$ and $f_2 > 0.5$, which intuitively places the contact surface between the two initial intersecting surfaces, only where they are actually intersecting.

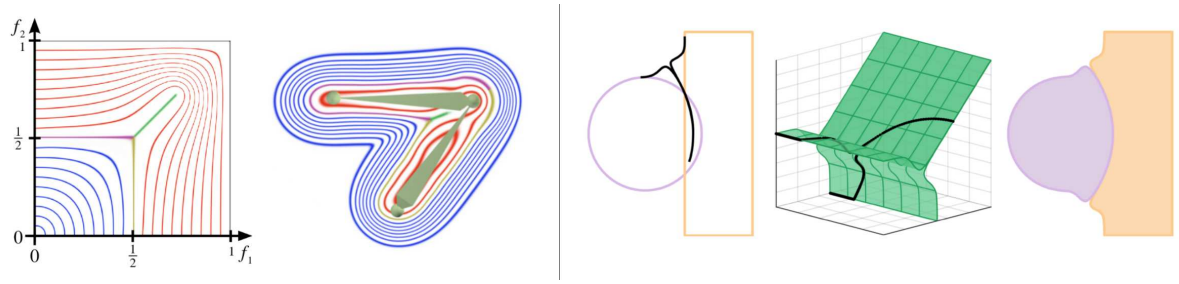

Fig. 1.13: Contact is directly modeled as an isolevel in the field space by Vaillant et Al. ([VGB+14],left) and Angles et Al. ([ATW+17], right)

In [ATW+17], this isovalue is sketched by the user in the world space, and interactively converted in the field space as an operator (Fig. 1.13, right).

### Other applications of implicit surfaces

Interestingly implicit surfaces have also been used for other types of applications such as

- the use of convolution surfaces and their extension to scale invariant integrale surfaces (SCALIS)[BS00],[ZBQC13],[ZGC15],[FSHZ19] to overcome the limitations of simple soft-objects;
- controlling blending location by enabling it only where needed [PASS95b, dWv09, BBCW10] or using composition operators that depend on the orientation of the fields [Roc89], [GBC+13],[CGB13]
- the simulation of highly deformable and non-constant topology objects, such as fluids [SS03],[Kim10] or magnetic substances [NZWC20], where the implicit surface represents the interface between phases.

### Discussions

Scalar field enable to define closed surfaces with a built-in volumetric model. This allows to easily define an interior and exterior regions, thus offering an easy way to detect potential collisions. The contact between objects can be modeled through composition operators, and its form and location can be tweaked and parameterized in the field space. However, as far as we know none of the aforementioned operators deal with the problem we tackle in chapter 2, that is the untangling of any number N of nested implicit surfaces. Moreover, most of the methods for contact modeling only operate on two fields by the mean of binary operators (eventually completemented by a third parameter which is not a field), while we propose a N-ary operator, for which the state of the art is rather succint. This operator is defined in the field space. Moreover, while previous method offered

numerical method to compute and store their operator, we propose a closed-form formula for its definition.

### 1.2.2 Divergence-free vector fields

A vector field $u$ can be defined in all generality as

$$u : \mathbb{R}^3 \to \mathbb{R}^p$$

ie function mapping each point in space to a $p$-dimensional value, with $p$ usually equal to 2 or 3. We will assume $p = 3$ in the remainder of this section, as we are mostly dealing with 3D modeling and animation.

#### Divergence of a vector field

The divergence of a vector field $u$ is defined in cartesian coordinates as

$$\nabla.u = \frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y} + \frac{\partial u_z}{\partial z}$$

using the $\nabla$ operator $\nabla = (\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z})$. An interpretation of its value is given by the *divergence theorem* : the divergence at one point p is the limit of the integral of the flux going in and out of a volume around p as the volume shrinks to 0. If the vector field represent a velocity field, $\nabla.u(p)$ is positive (resp. negative) if the matter advected by the field is compressing (resp. dilating) around p. In particular, $\nabla.u = 0$ if the velocity field represents an incompressible flow.

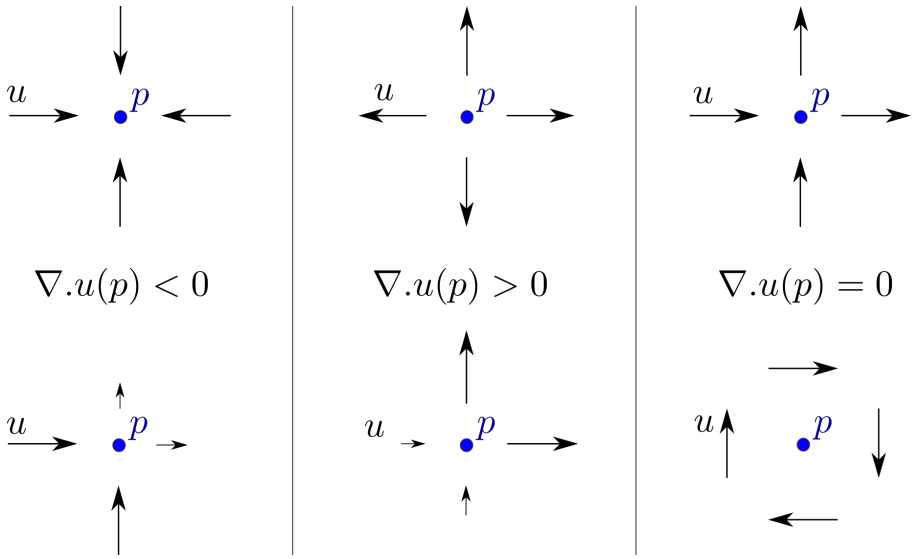


Fig. 1.14: Different divergence configurations for a vector field u. Top left, p is a *sink*. Middle top, p is a *source*. Bottom right: note the swirl motion typically observed in incompressible fluid simulation.

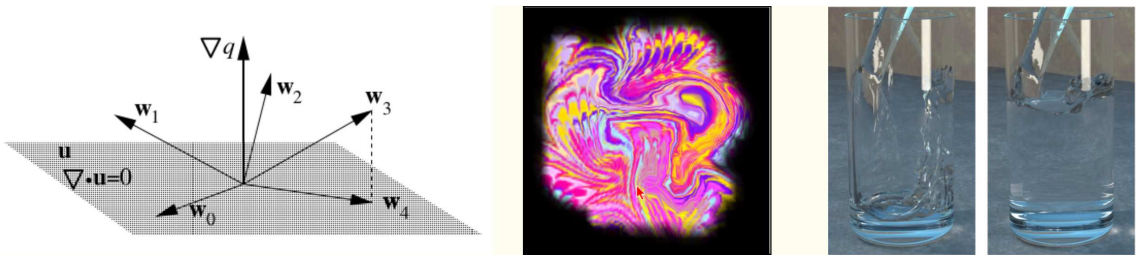


Fig. 1.15: The Chorin projection method (left, image from [Sta99]) is used to obtained vector field modeling incompressible behavior in fluids such as gas (middle, [Sta99]) or water (right, [EMF02]).

**Divergence constraint in 3D modeling and simulation**

Constraining the divergence of a velocity field to zero has been extensively done to model incompressible material. The most known example would be the simulation of incompressible fluids. Chorin [CM79] first introduced the *projection method* which relies on the *Helmhotz-Hodge decomposition* : any vector field $w$ can be decomposed into

$$w = u + \nabla(q)$$

with $u$ a divergence-free velocity-field, $q$ a scalar field and $\nabla(q)$ its gradient (not to be mistaken with $\nabla.q$). Multiplying both side of the equation by the $\nabla$ operator thus gives $\nabla.w = \Delta q$ with $\Delta q$ the laplacian -divergence of the gradient- of q. This is a Poisson equation that can be solved to obtain q. Finally, $u$ is obtained with $u = w - \nabla(q)$, and represents the divergence-free projection of $w$. This scheme has been used to obtain divergence-free velocity fields in fluid simulation in works such as [Sta99], [EMF02] or [NNC+20]. However, other methods exist : for instance, using the fundamental property that $\nabla.\nabla \times u = 0$, Bridson et al [BHN07] used the curl of a Perlin noise to generate divergence-free turbulence.

While such constrained vector field has been prominently used in fluid simulation, it is not limited to this domain. For instance in modeling, Angelidis et al. [ACWK04] composed severeal *swirling sweepers*, showing that they form a deformation field that preserves volume, and thus with null divergence. Their method thus provided a modeling tool conserving volume of material. Clebsch decomposition [Kot91] is used in [vFTS06] to decompose a deformation field into two scalar field, and a divergence-free deformation field is obtained by the cross-product of their gradients. This forces the field to locally preserve volume, which prevents self-collision during deformation. Such decomposition is also used in [AS07] to adapt a deformation field to volume-preserving skinning. Recently, Eisenberger et al. [ELC18] introduced a basis of divergence-free vector field with which they create deformation fields to both interpolate and find correspondances between shapes.
Divergence-free vector field have been used in simulation of other body than fluids as well: Irving et al. [ISF07] solved for a pressure term, the gradient of which they used to correct

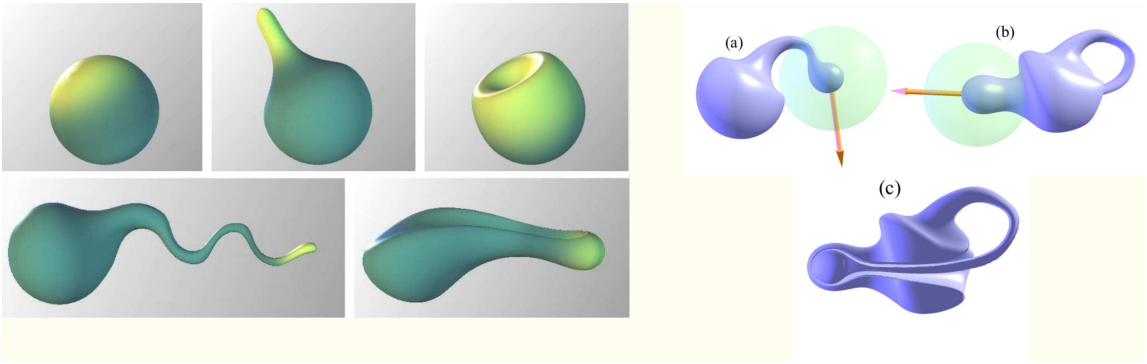


Fig. 1.16: Divergence-free deformation fields are used to deform initial shape while preserving its volume (left, [ACWK04]) or to prevent self-collision (right, [vFTS06]).

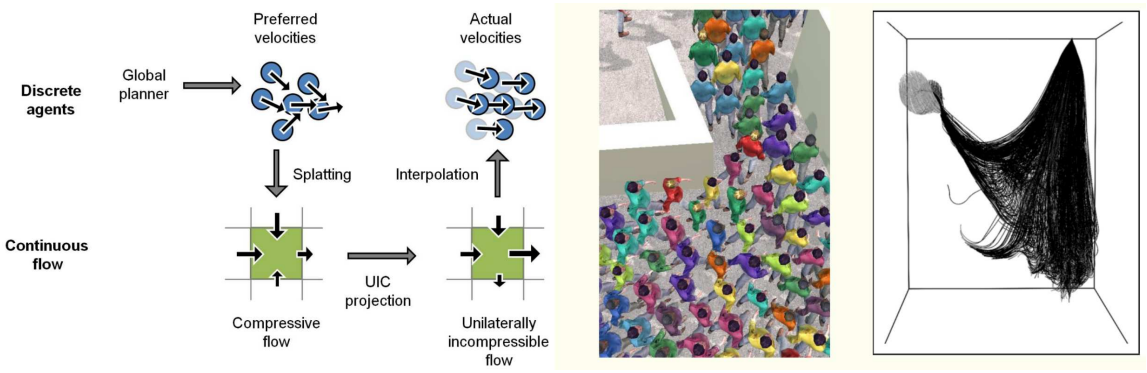


Fig. 1.17: Narain et al. (left and middle,[NGCL09]) and McAdams et al. (right, [MSW+09]) constrain an existing velocity field to be divergence-free in order to avoid collision between people in a crowd, or between strands of hair.

their velocity field, and perform volume conserving simulation of deformable volumetric body. Diziol et al. [DBB11] used the divergence theorem on the surface of a closed mesh, and derived a position based constraint that they minimize in order to preserve volume during animation. The idea of avoiding collision of [vFTS06] can also be encountered in works like [NGCL09] in which Narain et al. use a linear complementarity problem on the density of people to find a divergence-free vector field for their crowd not to collide with itself. McAdams et al. [MSW+09] constrained the divergence of their velocity field to avoid that strands of hair cross each other during a time-step, thus preserving the global volume of hair.

### 1.2.3 Discussions

Vector-fields have been extensively used to model deformation fields or velocity fields. Constraining the divergence of a vector field to be 0 makes it conserve the volume of the material it advects. This allows to model the behavior of continuous, incompressible

matter such as fluid, but also of discretely modelised matter which can aggregate but should not collide (such as hair or people in a crowd). This has not been applied to cloths yet : one can argue that volumetric null-divergence might be overshooting for cloths which are mostly 2-dimentional objects. However, we believe that approaches such as the one of McAdams et al. [MSW+09] can be adapted to clothes. Beside, considering that clothes are volumetric body where they aggregate is the idea behind the Rigid Impact Zone mentioned in Section 1.1, which furthermore reinforces this intuition.
In Chapter 3, we propose an adaptation of the method of Narain et al. [NGCL09] and McAdams et al. [MSW+09] to thin, surfacic bodies such as clothes. They are embedded in a 3D velocity field that we make locally divergence-free in order to prevent collision when necessary, while maintaining the initial velocity when not.

# Chapter 2

# Implicit Untangling: Using scalar fields to untangle layers of clothes

Fig. 2.1: Cloth individually fitting a mannequin.

In this chapter, we tackle a specific problem of untangling. In real world, garments are often worn in multiple layers, which impact both their visual appearance and their dynamics. In order to mimics such behavior, virtual characters need to be dressed with multiple layers as well. Given a 3D avatar, and a pre-existing wardrobe composed of well-sized garments modeled by triangular meshes, one might want to try to manually fit several garments to the character by applying rigid transformation to them. However, chances are high to obtain a tangled configuration , that is a configuration in which meshes are intersecting each other. Correcting these intersections would require local deformations to be applied on the meshes, which are particularly complex to set using interactive tools when multiple layers are intersecting. Indeed, each mesh layer could be finely tuned and added on top of the other ones in a specific configuration, but this approach would be particularly tedious and time-consuming as the number of layers increases.

Fig. 2.2: When worn altogether, the meshes heavily intersect each others.

Intersecting garments can typically be encountered by an artist trying to dress a character given a set of garment meshes, or can be seen in games when selecting garments to dress an avatar. Intersecting starting configuration are problematic as most cloth simulators need a configuration without intersections to start an animation, or just to provide guarantees that the meshes shall remain intersection-free. This issue is often circumvented by only proposing a set of garments with simple or specific shapes which cannot intersect other layers, or even in considering only a single layer of cloth that solely visually mimics the appearance of superimposed garments. A naive solution would be to consider each layer acting as a rigid body constraint for the one on top of it. An iterative algorithm based in this idea could consists in pushing the first layer out of the body, and then recursively consider a new body as being the union of the old body and the previous layer to correct subsequent layer by also pushing them away. While this approach would ensure a result without collision, the relative influence of the outer layer would not be taken into account on the inner one: a leather jacket would not be flattening lighter

Fig. 2.3: Our method is able, from a set of garments possibly exhibiting deep interpenetration, to compute an untangled state, which takes the relative rigidity and thicknesses of layers into account and is animation ready.

layers underneath it.
The method we propose to solve this problem uses an intermediate representation of the meshes as implicit surfaces, along with a co-variant field. The intersecting implicit surfaces are combined using a new family of N-ary composition operators, specially designed for untangling layers. These operators are parameterised by the relative influence of each garment on the other which allows to model deformations caused by the superposition of layers of clothes. Garment meshes are finally projected to the deformed and collision-free implicit surfaces, while trying to best preserve triangles and avoiding the loss of details (see Section 2.1).
Our method stands appart from existing ones as no primitive test between triangle of the intersecting meshes are necessary to untangle them. As such, given a fixed number of layers, our method achieves linear complexity in term of the number of triangles in the scene, allowing for the use of high-resolution meshes. Additionally, no arbitrary order of correction is fixed, and all the corrections are done in one go, without having to resort to iterative method to ensure correction. Finally, the operators we created are defined for any number of layers, allowing for the stacking of high number of garments, while methods in the litterature often restricted themselves to 2 or 3 layers simultaneously interacting.

## 2.1 Overview

We aim at untangling an arbitrary number of intersecting garments. The key insight is to take benefits from the ability of volumetric representations to model contact through the composition of scalar fields. To achieve this, garments need to be converted to some implicit representation, while contact modeling by iso-surface composition needs to be extended to the untangling of an arbitrary number of nested surfaces.

### 2.1.1 Notations and input

In this chapter, the implicit surfaces $S_i$ we consider are closed surfaces defined as the 0-iso-surface of a scalar field $f_i$. By convention, we define the interior of the volume within $S_i$ by $\{p \in \mathbb{R}^3 \mid f_i(p) < 0\}$. This is the convention used by HRBFs (Hermite Radial Basis Functions) [Wen05] on which we will build on. They enable a user to reconstruct closed implicit surfaces from a set of sample points and the associated normal directions.

We consider a set of $N$ predefined garment models given by their input meshes, which are already wrapped over an input character body. The user specifies the desired order between these garments, where layer 1 is the closest to the body and layer $N$ is worn above the other layers.

Virtual *thickness* values $t_i$ are specified for each layer, enabling the tuning of the minimal distance between the centered meshes representing each garment, while *weights* $w_i > 0$, acting as a stiffness parameter, enable the tuning of their relative influence when deformed during contact. Small values of $w_i$ mimic very flexible material that tends to match the geometry of other surfaces in contact regions, while large values mimic stiffer layers that tend to keep their shape. Note that the body is handled as a fully rigid layer $w_i = +\infty$. The deformation we want to output for each garment is to account for the combined effect of all the other layers (above and under) both in terms of thicknesses and weights.

### 2.1.2 Processing pipeline and challenges

Our processing pipeline, depicted in fig. 2.4, includes a prepossessing step—namely implicit approximation of all garments in a library, which can be done independently from a specific set-up. At run time, the user selects layers, sets their parameters, and untangles them. More precisely, for each mesh vertex, gradient descent and tangential relaxation are interleaved to deform each garment towards a target implicit surface, computed by applying an untangling operator to the input fields. Let us detail the main problems to be solved.

**Implicit representation of garments:** While implicit surface reconstruction from sample points is a solved problem for closed surfaces such as the character's body, extending it to open surfaces with boundaries such as garments is a challenge. In particular, the new representation needs to generalize the notion of *inside* and *outside* to these open surfaces, to be able to detect the interpenetration regions between layers, in which corrections need to be applied through the field composition operator.

Therefore, our first contribution, presented in Section 2.2, is a general solution for approximating open surfaces with boundaries using a double implicit representation: we combine the field $f_i$ which embeds the mesh among iso-surfaces with a *co-variant field* $h_i$, computed from the open borders of the mesh. This enables us to define the region where a garment layer could cause intersections, accounting for the fact that it does not cover the full 0-iso-surface of $f_i$.

**Untangling operators for nested implicit surfaces:** Once the user sets a new configuration with a character and some ordered layers of garments from the library, implicit composition is used to untangle the associated fields, generating contact surfaces through deformation in the regions where garments interpenetrate (see fig. 2.4-middle). Here, we are not looking for a single composition operator but for $N$ operators $\mathcal{O}_i$, parameterized by the layers weights $w_i$, and able to generate $N$ new fields $\hat{f}_i$ whose respective 0-iso-surfaces coincide in the previously intersecting regions while remaining intersection-free elsewhere.

We introduce a closed-form solution for these operators, enabling us to apply them whatever the number of colliding layers and their relative rigidities. Section 2.3 first explains how these operators can be defined in the general case of N layers, before detailing it for the specific case of two and three layers.

**Using the deformed fields for updating the garments:** In our case, the untangling composition is not to be applied to some abstract, nested implicit surface, but to meshes representing garments. Similarly to Vailllant et al [VGB+14], output meshes are computed by displacing vertices of the original input meshes along the gradient of their respective field $\nabla\hat{f}_i$ until reaching the zero value of $\hat{f}_i$.

Several changes however need to be made: first, the meshes should not lie on the same 0-iso-surface in contact regions, but at some offset distance from each other, to avoid visual artifacts and allow subsequent physically-based animation. Secondly, some tangential relaxation is required to ensure that the local mesh deformation remains minimal while well approximating the 0-iso-surface of $\hat{f}_i$. Indeed, over-elongated triangles could cause geometric intersection even while implicit layers do not intersect, and could lead to unstable subsequent simulations.

Section 2.4 explains our practical solution for applying the deformation to meshes, which involves an extended composition operator accounting for the required thicknesses

$t_i$ of each layer to guarantee nested target surfaces, and a new relaxation method that we interleave with gradient descent to reduce tangential distortions.

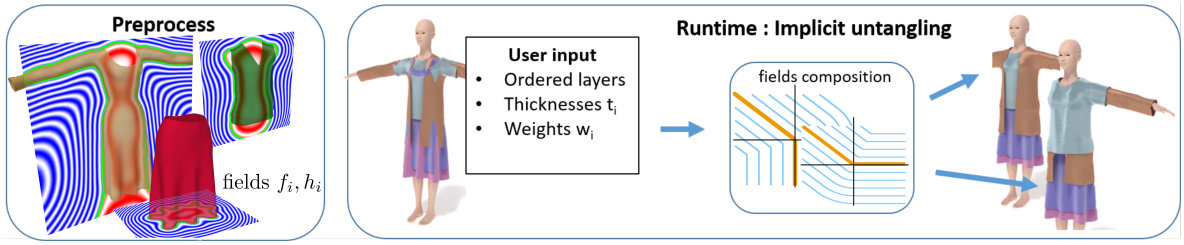


Fig. 2.4: Processing pipeline: Implicit reconstruction of all garments are computed as a preprocess (left). At run-time, the user selects a number of garments, sets their order, thicknesses and weights (center). The associated fields are composed using our new untangling operators, leading to a variety of possible results after mesh projection (right). For instance, the T-shirt (low weight during deformation) can either be worn below or above the thicker leather jacket.

## 2.2 Implicit approximation for garments

While we use standard HRBFs to approximate the character's body, we need to extend implicit reconstruction to the case of open surfaces with boundaries for handling garments. The main challenge is to characterize regions where the garment might interpenetrate with others during field composition, used to test for interpenetration: indeed, using the full 0-iso-surface of $f_i$ to approximate a cloth layer would not work, since extra collisions could then be wrongly detected, leading to unwanted deformations of the other layers, such as in the region below the jacket and t-shirt in Fig. 2.5. The key idea to achieve this is to use a second implicit volume, defined by a *co-variant field* $h_i$.

More precisely, the input mesh is first sampled using Poisson dart-throwing. Samples are used to compute a HRBF approximation $f_i$ ([Isk02] , [MGV11]), such that for each pair point/normal $(p_s, n_s)$ in the sample, $f_i(p_s) = 0$ and $\nabla f_i(p_s) = n_s$. The implicit surface defined by $\{p \in \mathbb{R}^3 \mid f_i(p) = 0\}$ is then close to the initial mesh anywhere near one of its vertices. As in former works combining meshes and implicit modeling [VBG+13], each mesh vertex keeps track of its exact iso-value in the reconstructed field $f_i$, so that no detail is lost (eg. thicker parts such as seam-lines or pockets would be adequately reconstructed after deformation). Note that in practice, we pre-store the values of $f_i(p)$ and $\nabla f_i(p)$ in a grid. Evaluating $f_i$ and its gradient at any spatial position is then efficiently approximated using a tri-linear interpolation. Due to the continuity of the HRBF model, the 0-iso-surface of $f_i$ is a closed surface. In order to be able to discard parts where interactions with the other garments should not be considered, we also compute a co-variant field $h_i$, aimed at capturing the region "inside borders". While the use of two scalar fields was already introduced in order to reconstruct open surfaces from incomplete point-set

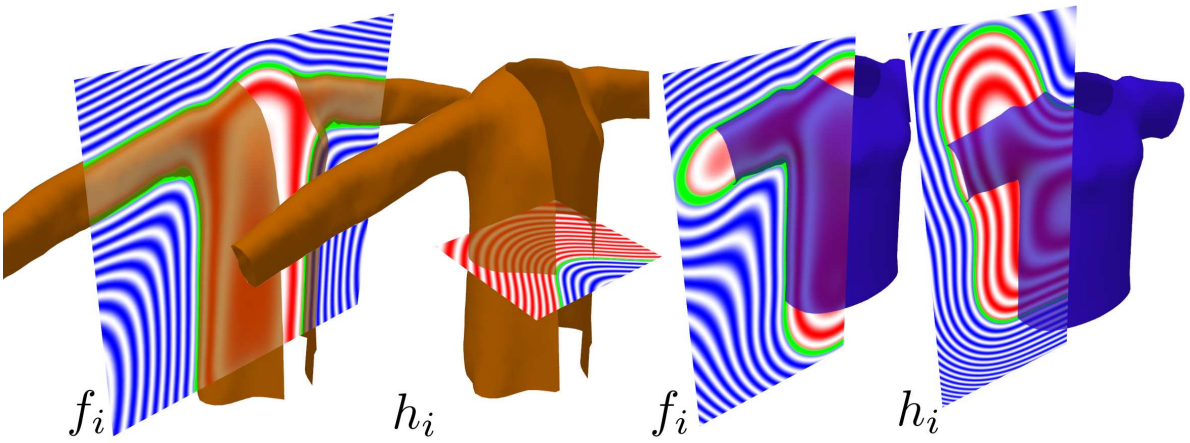


Fig. 2.5: From left to right: a jacket and a T-shirt, with the associated field $f_i$ and co-variant field $h_i$. The iso-values of each field are depicted on a 2D plane set to intersect the model, where positive isovalues are in blue, negative ones in red, and the 0-iso-value in green. Note that the negative part of the covariant field (in red) enables us to characterize the part of space where a garment might interact with other layers. This enables to discard, for instance, the rounded part at the bottom of $f_i$'s 0-iso-surface for the T-shirt.

acquisition using evolving level-sets [SH04], we propose a new efficient formulation for the field $h_i$, based on HRBF, specifically adapted to take advantage of meshes with boundaries . More precisely, the HRBF is computed from the open-boundary samples $(p_{s'}, n_{s'})$, where the gradient $\nabla h_i(p_{s'}) = n_{s'}$ is set to the unit vector both orthogonal to the open-boundary curve and in the tangent plane with respect to the surface (see Fig.2.5). The region where $h_i(p) > 0$ is called the *zero-influence* region of the layer, meaning that it should not be considered for processing interactions with this layer. Conversely, the *influence region of a layer* is defined by $h_i(p) < 0$.

In consequence, the *inside* of the garment, which should be used to test for collision with outer cloth layers is defined as $\{p \in \mathbb{R}^3 | f_i(p) < 0 \ \& \ h_i(p) < 0\}$ while the *outside* of the garment, where collisions with inner cloth layers are detected, is given by $\{p \in \mathbb{R}^3 | f_i(p) > 0 \ \& \ h_i(p) < 0\}$.

## 2.3 Untangling nested implicit surfaces

In this section, we describe our implicit untangling operator in the general case of $N$ implicit surfaces defined as the 0-iso-surfaces of fields $(f_i)_{i\in[1,N]}$. Without loss of generality, we consider in the following that $i$ refers to their nesting order. The application to garments defined by both a field and a co-variant field is presented in Section 2.4. The objective is to define a set of closed-form composition operators enabling the defor-

mation of each of the surfaces so that contact is modeled in regions where they previously interpenetrated. Untangling is performed by replacing each field $f_i$ by $\hat{f}_i$ such that $\forall p \in \mathbb{R}^3, \hat{f}_i(p) = \mathcal{O}_i(f_1(p), .., f_N(p))$, where $\mathcal{O}_i$ is a composition operator i.e. a function from $\mathbb{R}^N$ to $\mathbb{R}$, and where each of the resulting isosurfaces $\{p \in \mathbb{R}^3 \mid \hat{f}_i(p) = 0\}$ does not intersect any of the other ones. To simplify notation, we consider the $f_i$, ie. functions applied to a position in space and returning $f_i(p) \in \mathbb{R}$, as a set of independent real variables. This enables us to define the N-dimensional vector $f = (f_1, \ldots, f_N) \in \mathbb{R}^N$, lying in the so-called *fields-space*, and to write the operator as $\mathcal{O}_i(f)$. Inspired from Angles *et al.* [ATW+17], the operators are built in two steps: First, we build a *desired* zero-set $Z_i$ in fields-space, i.e. in $\mathbb{R}^N$ in our case. Secondly, we build $\mathcal{O}_i$ from $Z_i$ such that $\mathcal{O}_i(f)$ is the signed Euclidean distance between $f$ and $Z_i$, thus ensuring that $Z_i$ is the zero-set of $\mathcal{O}_i$.
Finally, applying this operator to a position $p$ in 3D space, consists in evaluating $\mathcal{O}_i(f(p))$, where $f$ is now considered as a function of the input position $p$.

In the following, we present the construction of the zero sets $Z_i$, first by defining their general closed-form expression in the $N$-dimensional case (sec. 2.3.1), and then by explaining their intuitive construction in the special cases $N = 2$ (sec. 2.3.2), $N = 3$ (sec. 2.3.3) .

### 2.3.1 General formulation in the N-dimensional case

Let us consider a given layer $i$. We define its zero-set $Z_i$ as the union of $d = i(N - i + 1)$ sub-spaces $(H_i^{b,c})_{b \in [0,i-1], c \in [0,N-i]}$.

$$\forall i \in [1, N]\,,\ Z_i = \bigcup_{b,c} H_i^{b,c}\ . \tag{2.1}$$

Each subspace $H_i^{b,c}$ is itself defined by a hyperplane with normal vector $n_i^{b,c} \in \mathbb{R}^N$, and $N - 1$ additional inequalities expressed as linear combinations of the $(f_i)$, formally described using a matrix $M_i^{b,c}$.

$$\begin{aligned} &\forall i \in [1, N],\ \forall b \in [0, i-1],\ \forall c \in [0, N-i], \\ &\quad H_i^{b,c} = \left\{ f \in \mathbb{R}^N \;\middle|\; n_i^{b,c} \cdot f = 0 \ \text{ and } \ M_i^{b,c} f > 0 \right\}\ . \end{aligned} \tag{2.2}$$

As further detailed in the next sections, $H_i^{b,c}$ corresponds to a subset of $\mathbb{R}^N$ that expresses the interactions between the layer $i$ and all other layers within the interval $[i - b, i + c] \setminus i$. Intuitively, the equality constraint (related to the normal $n$) shifts the layers to be coincident rather than interpenetrating, and the inequality constraints (related to the matrix $M$) select whether this shift should be applied based on what interpenetrations are present.

The vector $n_i^{b,c}$ is defined as

$$\forall j \in [1, N],\ n_i^{b,c}[j] = \begin{cases} w_j & \text{if } j \in [i-b, i+c] \\ 0 & \text{otherwise,} \end{cases} \tag{2.3}$$

where the $w_j$ are the weights associated with each layer, and acting as stiffness parameters during the untangling process. Note that $n_i^{b,c}$ can be multiplied by a scalar value without changing the zero-set.

The matrix $M_i^{b,c}$ of size $(N-1) \times N$ defining the inequality constraints on $H_i^{b,c}$ can be expressed using four squared triangular blocks, respectively, $A$ of size $(i-b-1)^2$, $B$ of size $b^2$, $C$ of size $c^2$, $D$ of size $(N-i-c)^2$. These matrices have the following expression, when layer $i$ is in collision with its $b$ inner, and $c$ above layers.

$$M_i^{b,c} = \begin{pmatrix} A & 0 & 0 & 0 & 0 \\ 0 & B & 0 & 0 & 0 \\ 0 & 0 & 0 & C & 0 \\ 0 & 0 & 0 & 0 & D \end{pmatrix}, \text{ with} \tag{2.4}$$

$$A = \begin{pmatrix} w1 & w2 & \dots & w_{i-b-1} \\ 0 & w2 & \dots & w_{i-b-1} \\ & & \ddots & \vdots \\ 0 & \dots & 0 & w_{i-b-1} \end{pmatrix} \quad B = \begin{pmatrix} -w_{i-b} & 0 & \dots & 0 \\ \vdots & \ddots & & \\ -w_{i-b} & \dots & -w_{i-2} & 0 \\ -w_{i-b} & \dots & -w_{i-2} & -w_{i-1} \end{pmatrix}$$

$$C = \begin{pmatrix} w_{i+1} & w_{i+2} & \dots & w_{i+c} \\ 0 & w_{i+2} & \dots & w_{i+c} \\ & & \ddots & \vdots \\ 0 & \dots & 0 & w_{i+c} \end{pmatrix} \quad D = \begin{pmatrix} -w_{i+c+1} & 0 & \dots & 0 \\ \vdots & \ddots & & \\ -w_{i+c+1} & \dots & -w_{N-1} & 0 \\ -w_{i+c+1} & \dots & -w_{N-1} & -w_N \end{pmatrix}$$

Matrices B and C represent the conditions for layer $i$ and its direct neighbors to be in collision, while matrices A and D represent the absence of collision between layer $i$ and the other layers. Note that each line of these matrices is defined up to a positive scaling factor, since $M_i^{b,c}$ is used to express the N-1 conditions $M_i^{b,c} f > 0$.

### 2.3.2 Case of two layers ($N = 2$)

To give some intuition, let us explain in more details the construction of the zero-sets in the simple case of two layers $N = 2$, and show the correspondences with the variables previously defined.

Let $f_1$ and $f_2$ be the two fields representing the two implicit surfaces to be nested, where $f_1$ corresponds to the inner one and $f_2$ to the outer one. Let us detail the creation of the zero-set $Z_1$, which drives the creation of the operator $O_1$ (see Fig. 2.6 for a visual illustration in field-space). Let us consider a point $p \in \mathbb{R}^3$ on the inner implicit surface such that $f_1(p) = 0$. We can then consider two cases:

1. $f_2(p) < 0$: $p$ is inside the outer layer. This is a *legal position* in field-space, ie. an expected configuration compatible with the absence of collision. In such case, the iso-surface from layer 1 defined by $f_1 = 0$ should remain unchanged around $p$.

2. $f_2(p) > 0$: $p$ is outside the outer layer which is an illegal position, meaning that there is some intersection between the two layers. Intuitively, in this case we would want the inner iso-surface to be pushed inward by the outer one to solve such collision. We choose to correct the iso-surface by making it lie on $w_1 f_1 + w_2 f_2 = 0$. Indeed, this expression can be written as $f_1 = -\frac{w_2}{w_1} f_2 < 0$: following the conventions expressed in Section 3, the resulting surface is in fact placed inward with respect to $f_1 = 0$. As depicted in Fig.2.7, modifying the relative weights $(w_1, w_2)$ moves the corrected iso-surface continuously from $f_1 = 0$ to $f_2 = 0$.

The zero-set $Z_1$ can therefore be expressed as the following subset of $\mathbb{R}^2$:

$$Z_1 = \big\{ f = (f_1, f_2) \in \mathbb{R}^2 \mid \begin{array}{l} (f_2 < 0 \ \& \ f_1 = 0) \ || \\ (f_2 > 0 \ \& \ w_1 f_1 + w_2 f_2 = 0) \big\} \ . \end{array} \tag{2.5}$$

This relation can be rewritten under the general form described previously as:

$$\begin{cases} Z_1 = H_1^{0,0} \bigcup H_1^{0,1} \\ H_1^{0,0} = \{ f \in \mathbb{R}^2 \mid n_1^{0,0} \cdot f = 0 \text{ and } M_1^{0,0} f > 0 \} \\ H_1^{0,1} = \{ f \in \mathbb{R}^2 \mid n_1^{0,1} \cdot f = 0 \text{ and } M_1^{0,1} f > 0 \} \ , \end{cases} \tag{2.6}$$

with $n_1^{0,0} = (1, 0), n_1^{0,1} = (w_1, w_2)$, and $M_1^{0,0} = (0, -1), M_1^{0,1} = (0, 1)$. As explained previously, vector $n$ and the lines of matrix $M$ can be multiplied by positive scalars, and can equivalently be defined to match with the general notation of the equations (2.3) and (2.4), as $n_1^{0,0} = (w_1, 0), n_1^{0,1} = (w_1, w_2)$, and $M_1^{0,0} = (0, -w_2), M_1^{0,1} = (0, w_2)$, respectively.

In these cases, $H_1^{0,0}$ and $H_1^{0,1}$ represent half-lines in the 2D fields-space. $M_1^{0,0} = (0, D)$, with $D = (-w_2)$ (no collision with the single outer layer), while the other matrices A, B, C are empty. Similarly, $M_1^{0,1} = (0, C)$ with $C = (w_2)$ (expressing collision with the single outer layer). $Z_2$ is defined in a symmetric way. For a point $p$ lying on the outer implicit surface, for which $f_2(p) = 0$, two similar cases can arise:

1. $f_1(p) > 0$, denoting a legal position for which the second implicit surface should remain unchanged;

2. $f_1(p) < 0$, denoting an illegal position for which we set the new iso-surface to lie on $w_1 f_1 + w_2 f_2 = 0$, which can be rewritten as $f_2 = -\frac{w_1}{w_2} f_1 > 0$. The corrected iso-surface then lies outward the input one, which models the action of the inner implicit surface pushing the outer one away.

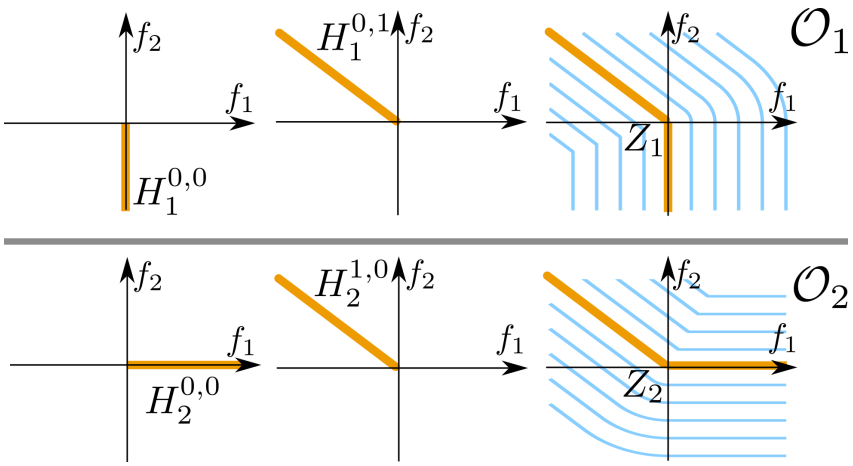


Fig. 2.6: N=2 with fixed $w_1$ and $w_2$: Left and middle: Visualisation of the subparts forming $Z_i$. Right: Visualisation of $Z_i$ and some iso-lines of $O_i$, computed as the distance to $Z_i$. Notice how $H_1^{0,1}$ and $H_2^{1,0}$ are similar, leading eventually to the contact between the two corrected iso-surface.

This leads to the following definition for the zero-set:

$$\begin{cases} Z_2 = H_2^{0,0} \bigcup H_2^{1,0} \\ H_2^{0,0} = \{f \in \mathbb{R}^2 \mid n_2^{0,0} \cdot f = 0 \text{ and } M_2^{0,0} f > 0\} \\ H_2^{1,0} = \{f \in \mathbb{R}^2 \mid n_2^{1,0} \cdot f = 0 \text{ and } M_2^{1,0} f > 0\} \ , \end{cases} \tag{2.7}$$

with $n_2^{0,0} = (0,1)$, $n_2^{1,0} = (w_1, w_2)$, $M_2^{0,0} = (1,0)$ and $M_2^{1,0} = (-1,0)$. Using the convention from Section 2.3.1, this can be re-expressed as $n_2^{0,0} = (0, w_2)$, $n_2^{1,0} = (w_1, w_2)$, $M_2^{0,0} = (w_1, 0)$ and $M_2^{1,0} = (-w_1, 0)$.

As we can check, both operators push points formerly on one of the implicit surfaces, but in the intersection region, to the same iso-surface $w_1 f_1 + w_2 f_2 = 0$, which untangles the two volumes and creates a contact surface instead. Note that the weights $w_1$ and $w_2$ do not need to sum to one: only their relative value is important, since the contact surface they define can be re-written as $f_1 + \frac{w_1}{w_2} f_2 = 0$. Playing with their relative value enables us to tune the position of the contact surface anywhere between $f_1 = 0$ (layer 1 not changed, ie. behaving rigidly during untangling) and $f_2 = 0$ (layer 2 not changed, ie. behaving rigidly, as illustrated in Figure 2.7).

### 2.3.3 Case of three layers ($N = 3$)

The case of three layers is slightly more complex. Indeed, a collision between layer 1 and layer 3 should not be handled directly without taking into account the influence of layer 2 in-between. To solve such inter-dependence, we consider first the collision of layer 2 and 3 independently from layer 1, and set, in case of collision, an intermediate

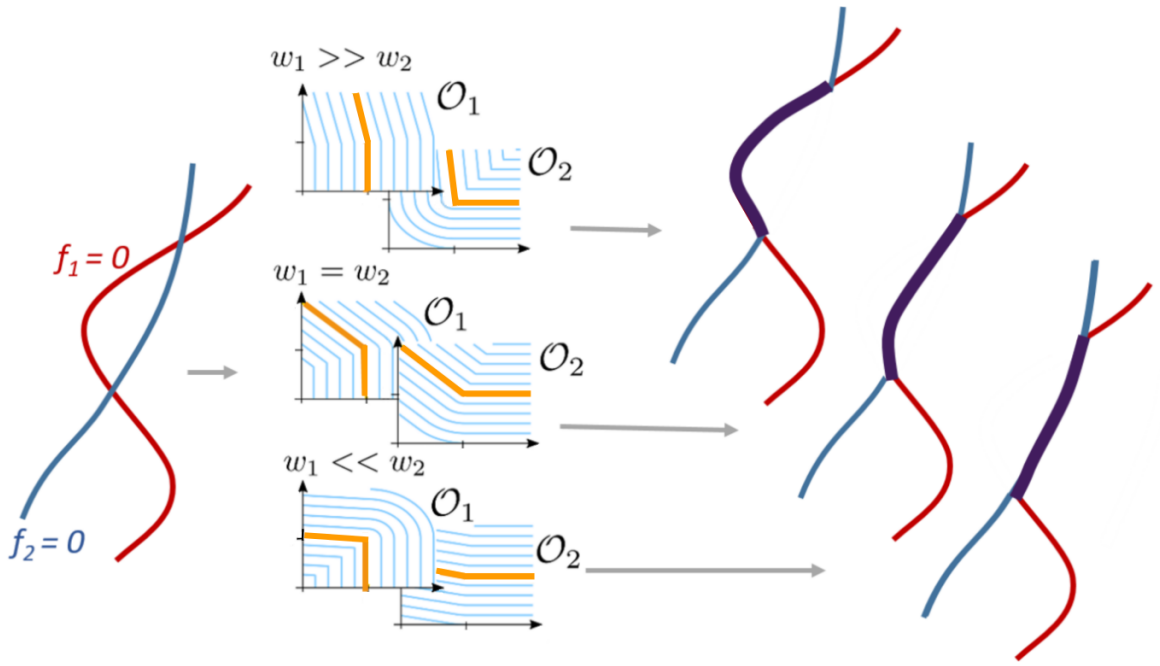


Fig. 2.7: Case of two layers. From left to right: two intersecting implicit surfaces in cross section, where the red one should be kept at the right; view of $Z_i$ and iso-lines of the operators $O_i$ in field-space ; untangled results. Modifying the orientation of the half-lines by tuning the ratio between $w_1$ and $w_2$ brings the contact surfaces after untangling from the former position of the blue layer to the one of the red layer (right, top to bottom)

corrected iso-surface defined as $w_2 f_2 + w_3 f_3 = 0$. Then the collision is handled between the intermediate iso-surface and layer 1.

To define $Z_1$, we are left with three cases when classifying the possible interpenetration states at a point $p$ on the first layer such that $f_1(p) = 0$:

1. No intersection occurs if $f_2(p) < 0$ (layer 2 is above, see Fig. 2.8.a ), and $w_2 f_2(p) + w_3 f_3(p) < 0$ (layer 3, after intermediate correction, is above, see Fig. 2.8.b ). In this case, we keep layer 1 non-deformed by maintaining the zero-set on $f_1 = 0$.

2. Only layer 2 is intersecting layer 1 if $f_2(p) > 0$ and $f_3(p) < 0$ (Fig. 2.8.c). In this case, similarly to $N = 2$, we chose to model coherently the corrected zero-set using $w_1 f_1 + w_2 f_2 = 0$.

3. Layers 2 and 3 are both interacting with layer 1 when $f_3(p) > 0$ ($p$ is outside layer 3) and $w_2 f_2(p) + w_3 f_3(p) > 0$. In this case, even if layer 2 is not intersecting layer 1 in its initial configuration, it is pushed by layer 3, and the resulting intermediate iso-surface ends up in collision with layer 1. (see Fig. 2.8 (e)). In this case, we set the corrected 0-set of layer 1 to $w_1 f_1 + w_2 f_2 + w_3 f_3 = 0$

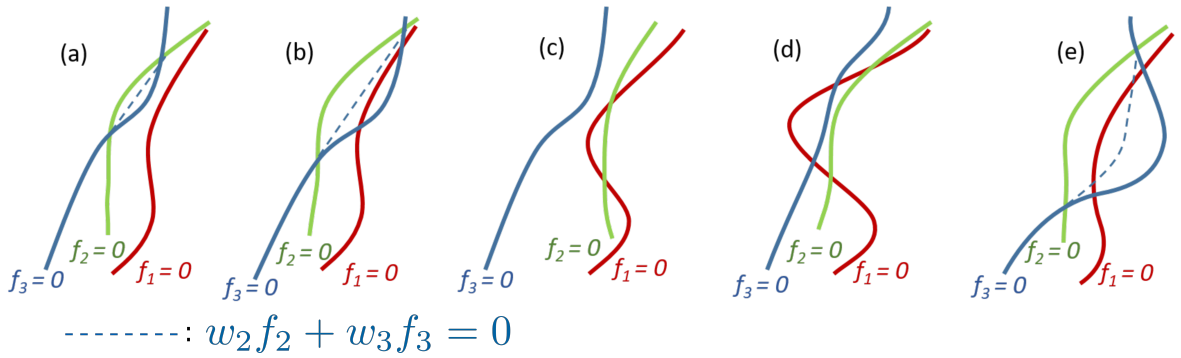


Fig. 2.8: Partial view of three implicit layers in cross section. The inner side is at the right of $f_1$, and the outer side is on the left of $f_3$. The dashed line represents, in case where it is needed, the intermediate surface profile between layer 2 and 3 represented by $w_2 f_2 + w_3 f_3 = 0$. (a) and (b): No interaction with layer 1; (c): layer 1 intersects only layer 2; (d) and (e): All three layers are interacting (note: the corrected state between 1 and the other layers is not represented).

These three cases turn into the 3 sub-spaces forming $Z_1$:

$$\begin{cases} Z_1 = H_1^{0,0} \bigcup H_1^{0,1} \bigcup H_1^{0,2} \\ H_1^{0,0} = \{f \in \mathbb{R}^3 \mid n_1^{0,0} \cdot f = 0 \text{ and } M_1^{0,0} f > 0\} \\ H_1^{0,1} = \{f \in \mathbb{R}^3 \mid n_1^{0,1} \cdot f = 0 \text{ and } M_1^{0,1} f > 0\} \\ H_1^{0,2} = \{f \in \mathbb{R}^3 \mid n_1^{0,2} \cdot f = 0 \text{ and } M_1^{0,2} f > 0\} \ , \end{cases} \tag{2.8}$$

with $n_1^{0,0} = (1,0,0)$, $n_1^{0,1} = (w_1, w_2, 0)$, $n_1^{0,2} = (w_1, w_2, w_3)$, and

$$M_1^{0,0} = \begin{pmatrix} 0 & -1 & 0 \\ 0 & -w_2 & -w_3 \end{pmatrix}, \ M_1^{0,1} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix}, \ M_1^{0,2} = \begin{pmatrix} 0 & w_2 & w_3 \\ 0 & 0 & 1 \end{pmatrix} .$$

Using the convention from Section 2.3.1 where 1 is replaced by the corresponding weight, this can be rewritten as:
$n_1^{0,0} = (w_1, 0, 0)$, $n_1^{0,1} = (w_1, w_2, 0)$, $n_1^{0,2} = (w_1, w_2, w_3)$, and

$$M_1^{0,0} = \begin{pmatrix} 0 & -w_2 & 0 \\ 0 & -w_2 & -w_3 \end{pmatrix}, \ M_1^{0,1} = \begin{pmatrix} 0 & w_2 & 0 \\ 0 & 0 & -w_3 \end{pmatrix}, \ M_1^{0,2} = \begin{pmatrix} 0 & w_2 & w_3 \\ 0 & 0 & w_3 \end{pmatrix}$$

In this case, we have $M_1^{0,0} = \begin{pmatrix} 0 & D \end{pmatrix}$ with $D = \begin{pmatrix} -w_2 & 0 \\ -w_2 & -w_3 \end{pmatrix}$, $M_1^{0,1} = \begin{pmatrix} 0 & C & 0 \\ 0 & 0 & D \end{pmatrix}$ with $C = (w_2)$ and $D = (-w_3)$, and $M_1^{0,2} = \begin{pmatrix} 0 & C \end{pmatrix}$ with $C = \begin{pmatrix} w_2 & w_3 \\ 0 & w_3 \end{pmatrix}$.

While the constraints associated to layer 3 can be derived symmetrically to the ones we just described for layer 1, the set of constraints to set up for the second layer are

slightly different. Indeed, layer 1 and 3 can be directly compared to layer 2 without requiring the consideration of any intermediate corrected surface. We therefore identify four possibles cases:

1. No collision: $f_1 > 0$ and $f_3 < 0$, leading to the isosurface $f_2 = 0$.
2. Collision with layer 1 only: $f_1 < 0$ and $f_3 < 0$ leading to the isosurface $w_1 f_1 + w_2 f_2 = 0$.
3. Collision with layer 3 only: $f_1 > 0$ and $f_3 > 0$ leading to the isosurface $w_2 f_2 + w_3 f_3 = 0$.
4. Collision with layer 2 and 3: $f_1 < 0$ and $f_3 > 0$ leading to the isosurface $w_1 f_1 + w_2 f_2 + w_3 f_3 = 0$.

Using the previous formulations, vector $n_i^{b,c}$ and matrices $M_i^{b,c}$ are defined as follows:

$$n_2^{0,0} = (0, w_2, 0),\ n_2^{1,0} = (w_1, w_2, 0),$$
$$n_2^{0,1} = (0, w_2, w_3),\ n_2^{1,1} = (w_1, w_2, w_3),$$

$$M_2^{0,0} = \begin{pmatrix} w_1 & 0 & 0 \\ 0 & 0 & -w_3 \end{pmatrix},\ M_2^{1,0} = \begin{pmatrix} -w_1 & 0 & 0 \\ 0 & 0 & -w_3 \end{pmatrix},$$
$$M_2^{0,1} = \begin{pmatrix} w_1 & 0 & 0 \\ 0 & 0 & w_3 \end{pmatrix},\ M_2^{1,1} = \begin{pmatrix} -w_1 & 0 & 0 \\ 0 & 0 & w_3 \end{pmatrix}.$$

and

$$\begin{cases} Z_2 = H_2^{0,0} \bigcup H_2^{1,0} \bigcup H_2^{1,0} \bigcup H_2^{1,1} \\ H_2^{0,0} = \{f \in \mathbb{R}^3 \mid n_2^{0,0} \cdot f = 0 \text{ and } M_2^{0,0} f > 0\} \\ H_2^{1,0} = \{f \in \mathbb{R}^3 \mid n_2^{1,0} \cdot f = 0 \text{ and } M_2^{1,0} f > 0\} \\ H_2^{0,1} = \{f \in \mathbb{R}^3 \mid n_2^{0,1} \cdot f = 0 \text{ and } M_2^{0,1} f > 0\} \\ H_2^{1,1} = \{f \in \mathbb{R}^3 \mid n_2^{1,1} \cdot f = 0 \text{ and } M_2^{1,1} f > 0\}. \end{cases} \tag{2.9}$$

Again, note that the equalities $n_1^{0,1} = n_2^{1,0}$ and $n_1^{0,2} = n_2^{1,1}$ allow to model the same contact iso-surface between, respectively, layer 1 interacting with layer 2, and all layers 1, 2, 3 interacting.
As in the case of two layers, the relative values given to the weights can be used to tune the amount of deformation applied to each layer in interpenetration situations. For instance, in the deep intersection case when all three surfaces interact and end up on $w_1 f_1 + w_2 f_2 + w_3 f_3 = 0$, this contact surface can be pushed towards $f_1 = 0$ ($w_1 \gg w_2 \,\&\, w_3$), $f_2 = 0$ ($w_2 \gg w_1 \,\&\, w_3$), or $f_3 = 0$ ($w_3 \gg w_1 \,\&\, w_2$).

The case $N > 3$ can be further developed by considering more intermediate layers for higher values of $N$, to reach the formulation given in Section 2.3.1. To provide a better insight on the construction of the matrices in the general case, $N = 4$ is given some insight in the next subsection.

### 2.3.4 General evaluation of the operator

The operator $\mathcal{O}_i$ can be evaluated at any position $f \in \mathbb{R}^N$ as the signed Euclidean distance between $f$ and $Z_i$. To evaluate this distance, we compute, as an intermediate step, the closest point $f_i^{b_0,c_0}$ such that

$$(b_0, c_0) = \min_{b,c} ||f - f_i^{b,c}||,$$

where $f_i^{b,c}$ is the closest point to $f$ on $(H_i^{b,c})$.

#### Computing closest point in the implicit space

The algorithm to compute efficiently $f_i^{b,c}$ for any $(b, c)$ is as follows. Given $f \in R^N$ and $(b, c)$, we want to compute the closest point $f_i^{b,c} \in H_i^{b,c}$ to $f$. This closest point can lie on different subspaces of $H_i^{b,c}$. For $N = 2$, each $H_i^{b,c}$ is a half-line stopping at $(0, 0)$, and $f_i^{b,c}$ can either lie on the half-line, or be $(0, 0)$.
For $N = 3$, each $H_i^{b,c}$ is a plane bounded by two half-lines described by two vectors $m_1$ and $m_2$ which are the lines of the matrix $M_i^{b,c}$, such that $\forall f, f \in H_i^{b,c} \Leftrightarrow m_1.f > 0, m_2.f > 0 \; and \; n_i^{b,c}.f = 0$. Therefore, $f_i^{b,c}$ can lie in 4 different locations:

1. "inside" the plane;
2. on the extremity of the plane such that $m_1.f = 0$;
3. on the extremity of the plane such that $m_2.f = 0$;
4. on the corner of the plane such that $m_1.f = 0$ and $m_2.f = 0$.

In fact, there are $2^{N-1}$ possible locations for $f_i^{b,c}$. Rather than explicitly testing each of the subspaces, which would lead to an exponential complexity in the number of layers, we use the following algorithm of quadratic complexity. Note that, in our tests, even for a small number of layers, our algorithm performs faster than the naive exhaustive test.

The first step is to use an adapted basis for $R^N$ in order to lower the number of tests. The algorithm 1 returns such a base. In this base, we can easily partition $R^N$ into $2^{N-1}$ sub-spaces. Each sub-space correspond to one of the possible location for $f_i^{b,c}$. We can determine to which space $f$ belongs with only $N - 1$ tests over the $v_i$. The base and the associated sub-spaces are illustrated in figure 2.9.

**ALGORITHM 1:** Computing an adequate basis of $R^N$

**Data:** $(n_i^{b,c}, (m_j)_{j\in[1..N-1]}$the lines of $M_i^{b,c})$
**Result:** $\mathcal{V}$ basis of $R^N$
**for** *j=1 .. N-1* **do**
  Compute $v_j$ as a solution of unit norm of the system :
    $n_i^{b,c} \cdot x = 0$
    $\forall l \neq j, m_l \cdot x = 0$
  **if** $v_j \cdot m_j < 0$ **then**
    $v_j \leftarrow -v_j$
  **end**
**end**
**return** $\mathcal{V} = (v_j)_j$

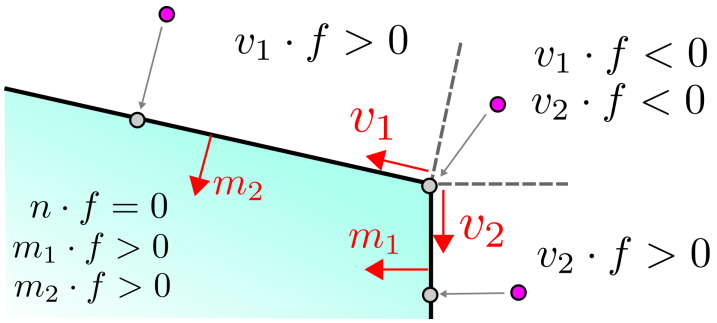


Fig. 2.9: Representation of an $H_i^{b,c}$ for N=3. The vectors $v_i$ are adapted to simple test to know where the projection of any f will lie.

This basis and its inverse are computed once for every $H_i^{b,c}$ when the weights $(w_i)_i$ are fixed.
Then, $f_i^{b,c}$ is computed using the Algorithm 2 below, which aims at re-orthogonalizing the adapted basis computed in Algorithm 1, in a well chosen order relative to the position of $f$ compared to $H_i^{b,c}$, which is obtained through the tests illustrated on figure 2.9 (for sake of readability, we use the same notation for a basis $\mathcal{V}$ and the matrix whose columns are the basis vectors):

**ALGORITHM 2:** Computing projection over an $H_i^{b,c}$

**Data:** $\mathcal{V}$, $\mathcal{V}^{-1}$, $f$
**Result:** $f_i^{b,c}$
$d = \mathcal{V}^{-1} f$ the decomposition of f into $\mathcal{V}$
I=[0] a set in which there is only 0
**for** *i=1..N-1* **do**
  **if** $d_k < 0$ *And* $v_k \cdot f < 0$ **then**
    Push k into I
  **end**
**end**
Reorder $\mathcal{V}$ into $\mathcal{V}_2 = (\ (v_i)_{i \notin I},\ (v_i)_{i \in I})$
$\mathcal{V}_3 = GramSchmidt(\mathcal{V}_2)$
$p = size(I)$
$\mathcal{V}_4$ = the sub-family consisting of the p last vectors of $\mathcal{V}_3$
c=$\mathcal{V}_4^{-1} f = {}^t\mathcal{V}_4 f$
$f_i^{b,c} = f - \mathcal{V}_4 c$
Return $f_i^{b,c}$

As explained in section 5.4, this algorithm is iterated over the $i(N-i-1)$ hyperplanes to find all the $f_i^{b,c}$ and compute $f^\star$. The algorithmic complexity of Gram-Schmidt's re-orthogonalization method (given below for sake of completeness) being $O(N^2)$, so is the complexity of Algorithm 2. Therefore, the complexity of computing $O_i(f)$ amount to $O(N^3)$.
Note that the order of the vectors given as input to the re-orthogonalization influences the result basis, hence the re-ordering of the basis in algorithm 2.

**ALGORITHM 3:** Gram-Schmidt Orthogonalisation

**Data:** $\mathcal{B} = (b_i)_{i \in [1,N]}$
**Result:** $\mathcal{O}$ orthogonal basis
$o_1 = b_1$
**for** *i=2 .. N* **do**
  $o_i = b_i$ **for** *j=1 .. i-1* **do**
    $o_i = o_i - (o_i \cdot b_j) b_j$
  **end**
  $o_i = o_i / ||o_i||$
**end**
Return $\mathcal{O} = (o_i)_{i \in [1,N]}$

**Evaluation of $\mathcal{O}_i$**

The final signed distance is then computed as:

$$\mathcal{O}_i(f) = \text{sign}(n_i^{b_0,c_0} \cdot f)\ ||f - f_i^{b_0,c_0}|| \ . \tag{2.10}$$

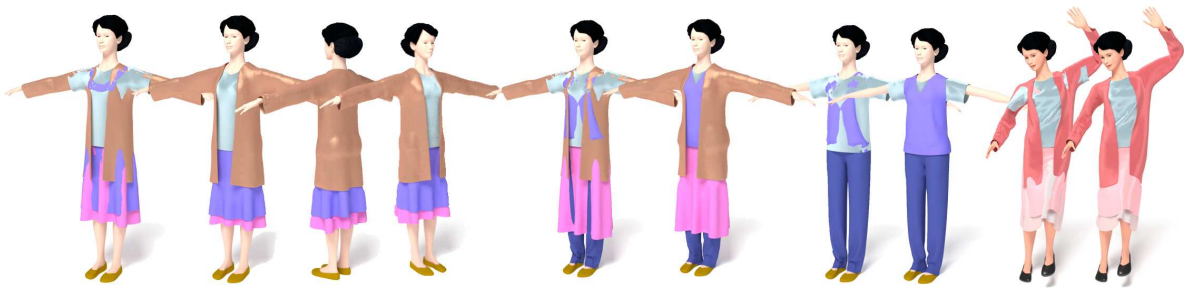

Fig. 2.10: Various untangled clothing on top of the same mannequin body (the initial colliding configuration is shown on the left of each example). Note that the resulting silhouette is strongly influenced by the worn layers. Right: application of our method on a different pose.

Once $f_i^{b_0,c_0}$ is known, we can note that computing the gradient of the operator $\nabla\mathcal{O}_i = (\partial\mathcal{O}_i/\partial f_j)_{j\in[1,N]}$ (which will be used in Section 2.4.3) is straightforward. Indeed, $\mathcal{O}_i$ is the Euclidean distance function, therefore its gradient at position $f$ is given by:

$$\nabla\mathcal{O}_i(f) = \text{sign}(n_i^{b_0,c_0} \cdot f)\ (f - f_i^{b_0,c_0})/\|f - f_i^{b_0,c_0}\|. \tag{2.11}$$

## 2.4 Application to untangling garment meshes

In this Section, we present the extension of field composition to open surfaces using the co-variant fields introduced in Section 2.2, the modification of the operators to take cloth thickness into account, and the computation of the resulting deformation of garment meshes.

### 2.4.1 Using co-variant fields to detect active layers

In practice, garments on top of a mannequin are only locally nested. For instance a jacket and trousers may only be overlapping around the hips of the character. As a result, detected inter-penetrations between the associated implicit surfaces should be discarded, except within the overlapping region.

More precisely, we define the *active layers* at a point $p$ in space as the set of garment layers $j$ whose influence region includes $p$, ie. such that $h_j(p) < 0$ (see Section 2.2). This enables us to compute the corrected 0-iso-surface of a given layer $i$ — giving the deformation to be applied to a mesh point $p$ of layer $i$, while only considering the other locally-active layers $j$.

In the implicit space $F$, canceling-out the effect of a layer $j$ consists in not taking its corresponding field $f_j$ into account within the computation of the operator, while keeping

the nesting order unchanged. Thus, the operator must be applied at each point $p$ on the subset of fields $(f_j)_{j\in\mathcal{J}(p)}$, with $\mathcal{J} = \{j \in [1, N] \mid h_j(p) < 0\}$. The corrected field for garment $i$ at $p$ is thus given by:

$$\hat{f}_i(p) = \mathcal{O}_i\left((f_j(p))_{j\in\mathcal{J}(p)}\right). \tag{2.12}$$

Note that the closed-form solution for our operators enable us to seamlessly switch to this lower-dimensional implicit space. In the remainder of this section, for sake of simplicity, we re-number the $M$ active layers as $p$ within $[1...M]$.

### 2.4.2 Taking cloth thickness into account

Applying the operators on values $f = (f_1, \ldots, f_M)$ leads to 0-iso-surfaces that are in exact contact when inter-penetrations are corrected. While this property was desirable to design our untangling operator, we actually aim at modeling cloth layers that may have a non-negligible physical thickness, and that will anyway need to be located on different surfaces for launching an animation. In the following, we include the required void space between garment layers aimed at avoiding ill-conditioned simulation within the notion of "thickness" of a cloth layer.

A naive approach to handle thickness would be to leave some geometric gaps between garment meshes and their target implicit surfaces, during the process of meshes projection to the associated implicit surfaces. Unfortunately, this could result in new collisions with neighboring layers due to this extra thickness. In contrast, our approach relies on directly integrating thickness values $(t_i)_{i\in[1,N]}$ within the expression of the operators, allowing to seamlessly and robustly handle collisions between thick layers.

Let us consider that the cloth surface, and thus the iso-surfaces defined by $\mathcal{O}_i(f) = 0$, is centered within the associated, thick cloth layer. As a result, layer $i$ should remain at a minimal distance of $t_i/2+t_{i+1}/2$ from layer $i+1$, at a minimal distance $t_i/2+t_{i+1}+t_{i+2}/2$ from layer $i+2$, etc.

We model this effect by applying offsets in the implicit space which convert into adequate displacements of the iso-surfaces in the 3D space. More precisely, applying an offset to a field $f_j$ leads to a geometric displacement of the layer $j$ along the gradient of the field $\nabla f_j$. In our case, $f_j$ is computed as a HRBF having, by construction, a unit gradient norm on the sampled surface points, and thus, at first approximation, $\simeq 1$ in the neighborhood of the 0-isosurface. Therefore applying a small offset $\delta$ in the implicit space leads to displacement of the layer $j$ along its normal by a length $\simeq \delta$.

Thanks to this property, we take into account the offset on the $M$ active layers with respect to the current layer $i$ by applying a change of variables before applying the operator. More precisely, we replace $\mathcal{O}_i(f)$ in Equation (2.12) by $\mathcal{O}_i(\tilde{f})$, with $\tilde{f} =$

$(\tilde{f}_1, \ldots, \tilde{f}_M)$ defined as:

$$\tilde{f}_j = \begin{cases} f_j & if\ i = j \\ f_j + \frac{t_i}{2} + \frac{t_j}{2} + \sum_{k=i+1}^{j-1} t_k & if\ j > i \\ f_j - \frac{t_i}{2} - \frac{t_j}{2} - \sum_{k=i-1}^{j+1} t_k & if\ j < i. \end{cases} \tag{2.13}$$

### 2.4.3 Projecting mesh vertices to their iso-surface

The modified operators we just presented are the ones used for deforming the meshes towards their corrected, untangled configuration. This is done by interleaving two relaxation processes at each vertex of the mesh: *gradient relaxation* consists in moving points along the gradient $\nabla \hat{f}_i$, toward their original iso-value, while *tangential relaxation* tends to make them slide along the iso-surfaces of $\hat{f}_i$ so that the distortion of mesh triangles is minimized.

Gradient relaxation is computed using Newton iterations using the gradient value given by

$$\nabla \hat{f}_i = \sum_j \frac{\partial \mathcal{O}_i}{\partial f_j} \nabla f_j \quad , \tag{2.14}$$

where the $\nabla f_j$ are obtained using trilinear interpolation of field values pre-stored in a grid and $\frac{\partial \mathcal{O}_i}{\partial f_j}$ is computed from the closest point $f_i^{b_0,c_0}$ as mentioned in Section 2.3. In practice, we use a step length $= 0.3$.

Tangential relaxation is inspired from As Rigid As Possible deformations (ARAP) [SA07]. The key adaptation to our case is to compute per-edge rotations and length changes: the rotation computed for each edge is the minimal rotation around its center that makes it tangential to the field, and the additional length change is a symmetric displacement of the vertices enabling the edge to restore its original length. These are used as target displacements for the two vertices defining the edge. At each tangential relaxation step, each mesh vertex applies an average of the displacements assigned the the adjacent edges.

Interleaving this second relaxation process with the more standard gradient descent enables-us to avoid over-elongated or inverted triangles, ensuring that the corrected iso-surface will be well approximated in deformed regions.

## 2.5 Results and discussion

### 2.5.1 Implementation

All times measured in this chapter were taken on a standard laptop computer with an Intel quad Core i7 CPU, clocked at 3.1GHz with 32GB of RAM. Our software uses up to

1Gb of RAM memory at run-time for the presented examples (including the storage of all fields and their gradients stored in uniform grid of size $256 \times 256 \times 64$) As our approach treat each vertex independently during mesh deformation described in Section 2.4, we use OpenMP to trivially parallelize our code. The rendered images were computed off-line using Blender and 3DSMax renderer for the animations.

### 2.5.2 Qualitative results

Several results of our method are depicted in Fig. 2.3 and Fig. 2.10, showing it can be used to model a variety of layered clothing while ensuring collision-free states. We note that although aimed at providing a collision-free configuration for the garments, our method is also able to generate a quite plausible initial configuration, enabling the user to test the look of the virtual character even before launching animations.

Fig. 2.11-left illustrates a change of layer order between a rigid jacket and a flexible t-shirt. Note how the strong rigidity of the jacket influences the visible silhouette of the t-shirt when the latter is above. Another example of exchange of layer order is shown in Fig. 2.3-middle between a t-shirt and a trouser. Fig. 2.11-right shows the action of the mannequin body which is modeled as a layer of infinite rigidity and has visible action around the hips.

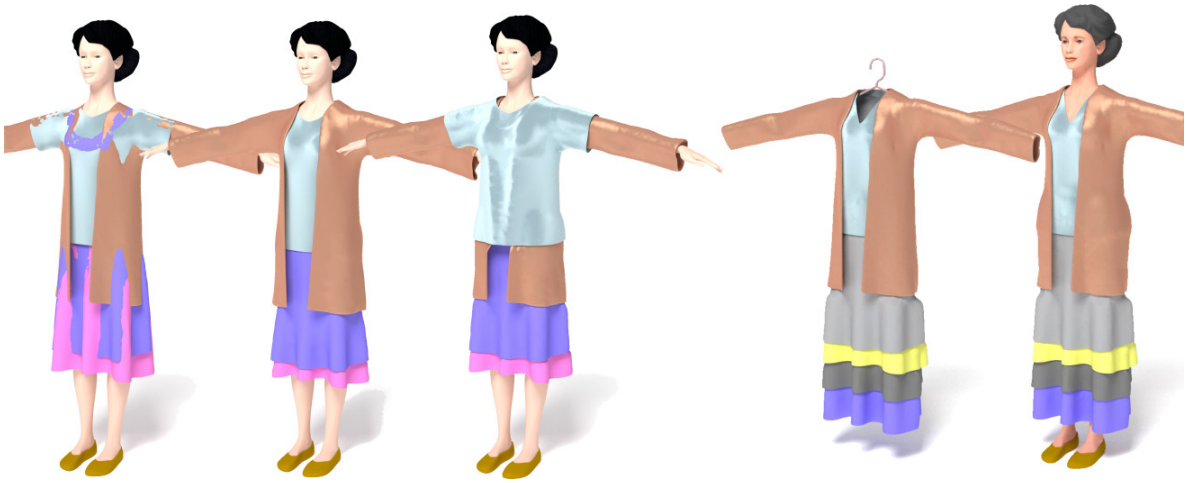

Fig. 2.11: Influence of the interior layers on the visible silhouette. Left: Exchange between a rigid leather jacket and a flexible t-shirt. Right: Result with and without the mannequin body.

We tested our approach on the extreme case of a character wearing 9 layers (including the body) and show the result in Fig. 2.12. This validates the robustness of the method in high dimensional fields space, with strongly tangled initial configuration, and deep inter-penetration. Vertical and top-to-bottom cuts are provided to illustrate the well behaved collision-free geometry of all internal layers, in comparison to the initial state.

As explained previously, user defined weights $w_i$ can model the relative influence between layers when collision is corrected. Changing these weights allows us to tune the relative amount of deformation of the layers from fully rigid to fully flexible. Fig. 2.13 shows a horizontal cut through the dresses of the model shown in Fig. 2.3-left when weights are modified.

Finally, as shown in Fig. 2.3-right, our untangled model can be directly plugged in as the initial condition of common cloth simulators to be animated without requiring any manual modification on the surface geometry. Note that the animated version of this model is provided in the accompanying video.

Fig. 2.12: Untangling an extreme configuration made of nine initially colliding layers. Cuts along layers are shown on both the initial input (left) and on the untangled result (right). A horizontal cut is shown across the dress in bottom, while the left-most and right-most cuts are performed on a 45° corner and zoomed-in to check that the resulting surfaces are fully exempt of collisions.

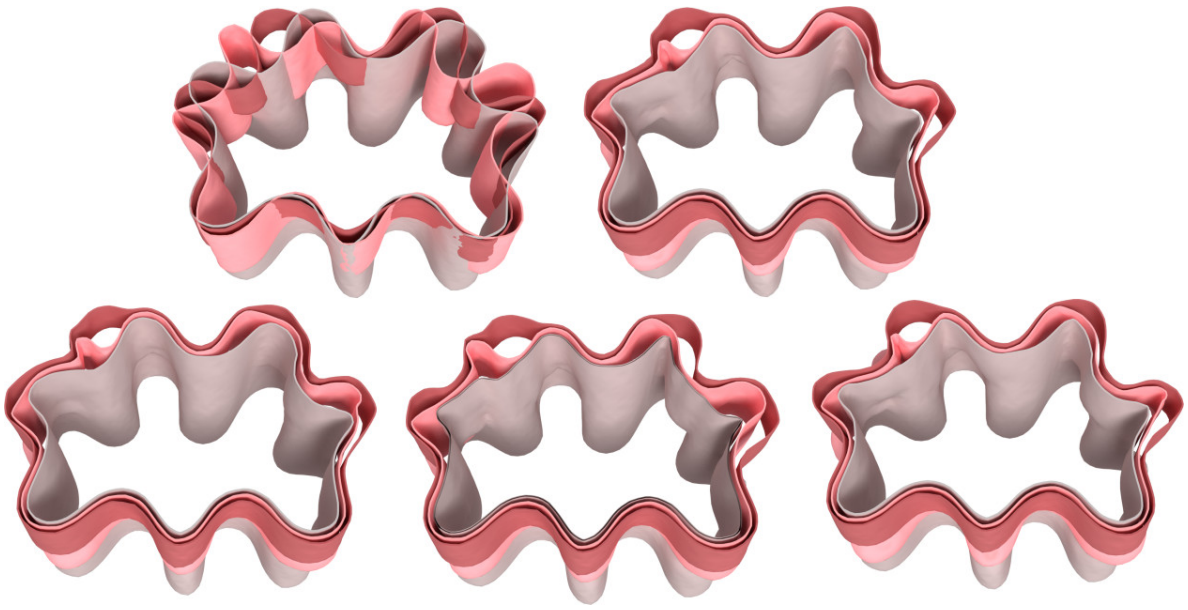

Fig. 2.13: Horizontal cut through the red dress layers shown in Fig. 2.3-left. Top-left: initial colliding configuration. Top-right: untangled configuration when all layers have the same weights. Bottom: layers weights are set, respectively from left to right, to $(w_1, w_2, w_3) = (2.5, 1, 1)$, $(w_1, w_2, w_3) = (1, 2.5, 1)$, $(w_1, w_2, w_3) = (1, 1, 2.5)$. Note how the most-rigid layer mostly-keeps its original shape and deforms the surrounding ones.

### 2.5.3 Quantitative results

The overall number of operations to untangle a model is $O(nKN^3)$, where $N$ is the number of cloth layers, $n$ is the number of vertices, and $K$ is the number of steps required in the iterative deformation. Indeed, the cubic complexity is brought by the field evaluation, while this evaluation has to be performed for every vertex until converging toward the 0-iso-surface. For all our examples we have $K \leq 15$, while $N$ is at most 9. As a result for a constant number of layers and iterations, our untangling algorithm is linear with respect to the number of vertices. Time variation with respect to $N$ in shown in Table 2.1. Table 2.2 shows our computation time for different cases and validates the roughly linear dependency of the computation time with respect to $n$. We can also note that, in practice, timings strongly depend on the number of vertices in collisions that need to be corrected. Therefore deep penetration of multiple layers are more computationally costly that correcting slight superficial ones. Table 2.3 provides a measure of the error in the user-defined layer thickness due to the approximation detailed in Section 2.4.2. Lastly, the precomputation of the fields for each garment takes less than 4 seconds for all our examples.

Table 2.1: Runtime in seconds with respect to the number of layers and of vertices, for two examples of increasing complexity. The 3 left (resp. right) columns correspond to the example depicted Fig. 2.3-middle (resp. Fig. 2.12) on which we added layers one by one. On the right, notice how deep penetrations involving the 3 first layers cause the runtime to drop for the entire example, while on the left the penetrations are superficial and are resolved in a few iterations.

| N | #Vertices n | Time(s) | N | #Vertices n | Time(s) |
|---|---|---|---|---|---|
| 3 | 6460 | 0.12 | 3 | 7751 | 1.04 |
| 4 | 9658 | 0.20 | 4 | 9200 | 1.29 |
| 5 | 15543 | 0.36 | 5 | 13673 | 2.35 |
| 6 | 18740 | 0.87 | 6 | 16871 | 3.45 |
| 7 | 19672 | 1.45 | 7 | 22453 | 4.54 |
| | | | 8 | 23385 | 6.80 |

Table 2.2: Each pair of lines correspond to the same example, on which we subdivided the meshes two times. Line one and two : example Fig. 2.3-middle, with one less layer. Line three and four : example Fig. 2.3-right. Line 5 and 6 : example Fig. 2.11-second picture. We can note that our method exhibits a linear complexity with respect to the number of vertices.

| | N | #Vertices1 | #Vertices2 | #Vertices3 |
|---|---|---|---|---|
| | 3 | 5062 | 20062 | 79816 |
| *Time* | | *0.22 s* | *0.76 s* | *2.71 s* |
| | 4 | 7502 | 29635 | 117785 |
| *Time* | | *0.28 s* | *1.06 s* | *3.56 s* |
| | 5 | 10694 | 42315 | 168275 |
| *Time* | | *0.57 s* | *2.1 s* | *7.15 s* |

### 2.5.4 Limitations

Although our method tends to generate plausible configurations in most cases — thanks to the proper handling of relative weights, and thicknesses — it stops considering the influence of a layer as soon as a point is out of the associated influence region. While this is not a problem with respect of getting correct collision-free configurations, this results in an overly distorted shape for the underlying layers, such as we can see in Fig. 2.14 for the parts of the skirts immediately below the jacket. If our method is an initialization before animation this is not a problem, but if our results are to be directly used as an illustration, some local relaxation is required, as shown in Figure 2.14 (right).

To achieve a better level of plausibility, our method would also need to consider the fact that garments deform isometrically with the associated 2D pattern: even is the cloth is slightly extensible, they tend to fold rather than compress. Consequently, folds should be added to the new shapes of inner layers, when they get compressed by a stiffer layer

Table 2.3: Measures of error between user defined thickness value and computed one. Note that the error increases with the thickness value and local curvature of the surface.

| Mesh | $\delta_{target}$ | $\delta_{obtained}$ | Relative error |
|---|---|---|---|
| Jacket | 0.20 | 0.1983 | 0.85% |
| PullV | 0.09 | 0.0901 | 0.11% |
| TShirt | 0.04 | 0.0406 | 1.5% |
| Petticoat | 0.04 | 0.0421 | 5.3% |

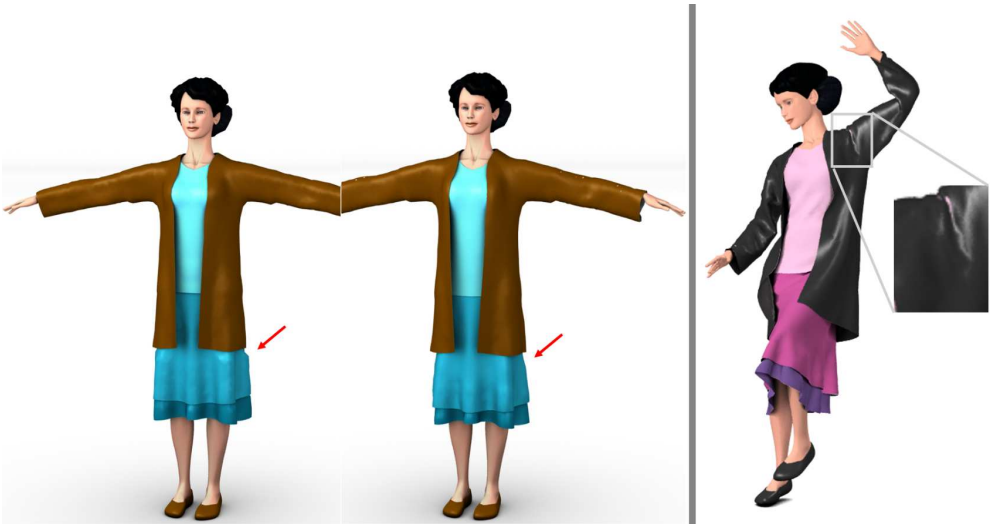

Fig. 2.14: Left : limitations. The salient distortion that occurs near the border of the influence zone of a top layer (left) can be attenuated using relaxation (center), if our results are to be directly used as plausible shapes. Right : Even when launched in collision-free states, standard cloth simulators often fail to generate collision-free motion for layered garments.

on top. This could be done by using inspiration from Rohmer et al. [RPC+10], which makes use of an implicit model for folds. The latter could be integrated as an additional displacement within our operator.

In addition, our method suffers from a few failure cases. Indeed, it only applies when a well-defined nesting between different cloth layers can be defined. This is not the case for a single, self-intersecting garment (see Fig. 2.15-left). In such case, the self-intersecting cloth surface cannot be reconstructed as some zero-sets of implicit fields to be untangled, which prevent the use of our method. Enabling implicit untangling to be applied on local surface patches computed on the fly would be a nice extension of our method, since it could allow us to handle the challenging case of self-collision states [AVGT12].

Another possible failure case is illustrated in Fig. 2.15-right, where a shorty is worn on top of a long dress. In this case, the nesting order is not captured globally by the field $h_i$, which leads to a final result where collision still occurs. Extending the definition of

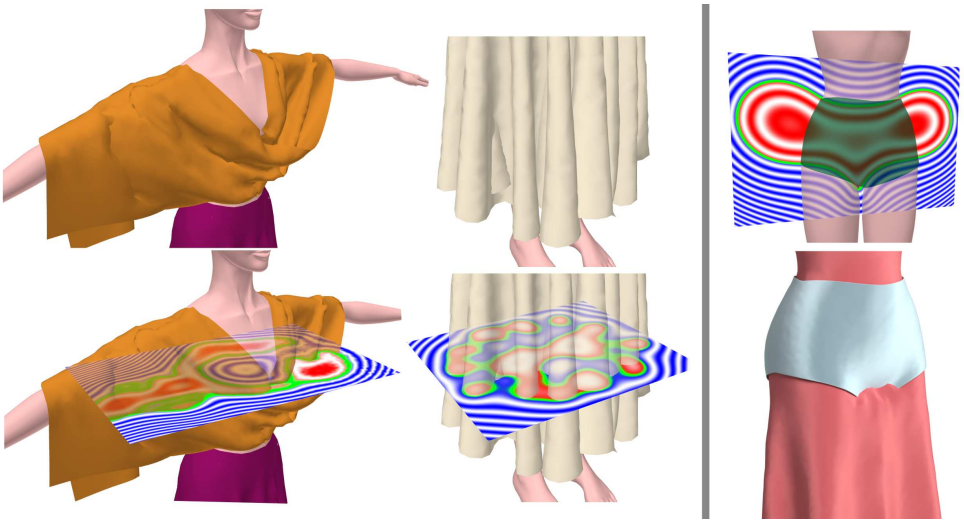

Fig. 2.15: Failure cases: on the left, a shawl folded back onto itself and a large skirt in deep intersection with itself cause the implicit reconstruction to fail: no clean 0-isosurface can be defined, which makes our approach inadequate for processing such self-collision case. On the right, the co-variant field $h_i$ computed for a shorty does not capture the influence zone of the shorty between the legs, leading the shorty to be ignored as an outer layer when the vertices of the dress are processed in this zone.

$h_i$, possibly taking into account user indications, could also be handled as a future work.

Lastly, being able to apply our untangling method at each step of an animation would be highly desirable, since cloth simulation engines may not be able to maintain collision-free states during highly dynamic motion, even if we provide one to start with. While applying our current method is feasible, it would require reconstructing the field and co-variant field of each garment at each frame, leading to a few tenths of seconds of computational time. Taking temporal coherence into account to allow us a more efficient reconstruction over time would be an interesting direction for future work.

### 2.5.5 Extension to animation

We studied a possible adaptation of the method to speed up the evaluation of the field for animation purposes during the internship of Yohann Kazoula that I supervised. Because one of the computational bottleneck remains in the computations of the HRBF, we aimed at circumventing it: given a point p, fetching the value of each $f_i(p)$ in the grid they are stored in is fast, but requires the computation and storage of each HRBF beforehand. While this was seen as precomputation in the scope of static untangling, it needs to be done each frame during animation which induces large computational overhead. Computing instead a signed distance to the garment $i$ is slower than only fetching for a value, but does not require heavy computation beforehand -or at least not as heavy as for the

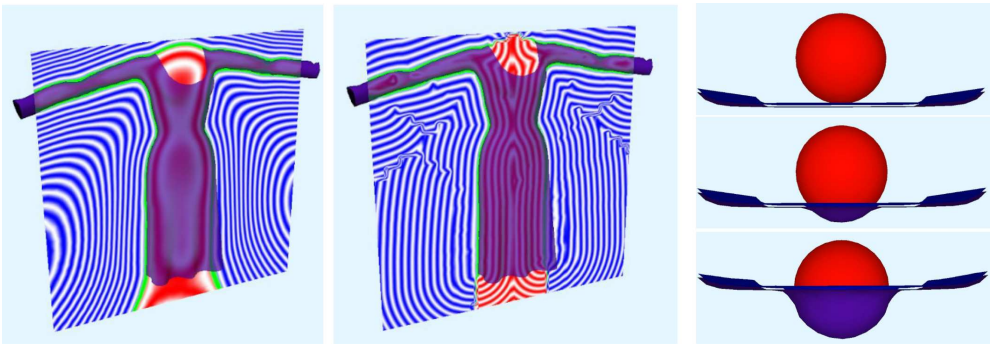

Fig. 2.16: On the left, the difference between the HRBF reconstruction of a dress, and the signed distance field computed in our extension.On the right, a few starting frames of a ball falling through 2 sheets of clothes.

HRBF. For this reason, we chose to replace HRBFs with signed distance fields computed only where needed.
Given a point $p$, computing the distance $d_i(p)$ of $p$ to the mesh $i$ is done by computing $close_i(p)$ the closest point to $p$ that lies on the mesh, and computing the distance to this point. Then, the signed distance $g_i(p)$ is obtained by the relative normal position of p with respect to the mesh. We used an uniform grid structure, in order not to loop on all the triangles of the mesh.
We kept the computation of the co-variant fields $g_i$ as they are, using HRBF functions, because they are less expensive to compute than the approximating fields $f_i$, and because we focused on the replacement of the approximating field in this extension.
We computed a small animation of a kinematic ball going through two sheets of cloth (figure 2.16). Computing a few frames using distance field to untangle the layers each frame took around 4 time less than recomputing and storing the reconstructing field as HRBF each frame (1.5 frames per second instead of 0.4 frames per second). While still outperformed by state of the art methods, this showcases how one of our bottleneck lies in the use of HRBF. In order to go further, a replacement for the covariant field should also be considered, along with better optimizations (such as hierarchical acceleration structure instead of an uniform grid). Moreover, our method would still not natively handle self-collisions, and a work-around would still be needed. As mentioned below, we decided to take another direction presented in Chapter 3.

## 2.6 Conclusion

We described a solution for untangling layered garments using scalar field and implicit surfaces. Our method allows to robustly convert penetration states into collision-free contact regions, while taking into account both a thickness parameter for cloth layers and their relative rigidity. The configuration we generate for the dressed character can serve as valid initial state for launching simulations as layers are guarantee to not intersect,

and are even separated by a user defined distance modeling the thickness of the clothes. Moreover, since relative cloth rigidity can be considered, this configuration remains close to the rest state in contact regions. This enables our method to be used for quickly generating plausible static shapes of garments that take any number of layers of underwear into account.
As discussed in section 2.5.4 and briefly prototyped in section 2.5.5, one obvious and useful extension would be to directly extend this method to animation. At each frame, one would :

- compute the approximating field and the co-variant field for each cloth;
- and then project, the clothes toward new-isosurfaces generated to correct possible intersections between the layers.

However, the previously described shortcoming are actually preventing this approach and should be adressed first. In fact, the computation of all the HRBF amounts to solve several dense linear systems, which is computationally prohibitive for interactive animation applications. As discussed previously, replacing these fields with distance fields coupled with basic optimizations already gives better performances.
While this issue is only related to performance, the use of HRBF -or other smooth implicit surfaces- to represent garments actually introduces another issue which conceptually prevents a straightforward extension to animation. Indeed, as discussed in Section 2.5.4, some configurations of clothes are not easily compatible with the global notion of inside and outside (see fig 2.15), including -in particular- the case of self-collision. Self collision is very likely to happen during physically-based animation of cloths and should therefore necessarily be accounted for. Mitchell et al. [MASS15] proposed a dedicated multivariate non manifold implicit surface model to handle self-collision. Orthogonally to our approach, this work is restricted to self-collision response and cannot handle intersection caused by different layers as it lacks the notion of interior/exterior. Ideally, both types of collisions should be handled using a single, unified approach.
For these reasons, we propose to introduce an unified methodology able to avoid both collision between different layers and objects as well as self-collision during an animation. The approach described in the next chapter will be based on the use of divergence-free vector fields, describing this time a velocity in space instead of a single scalar. This representation will notably be easily coupled with the use of a physically-based animation where the simulated velocities will serve as input for the vector field to be computed.

# Chapter 3

# Vector field based collision avoidance between an arbitrary number of dynamic shapes

The recent approaches on interactive cloth simulations are able to efficiently animate detailed cloth surfaces: as explained in Chapter 1.1 these approaches, carefully optimized to be computed on high-end GPU [TWL+18, LTT+20], are based on exhaustive per-triangle collision detection and response using the following pipe-line: implicit time integration on vertex positions, continuous collision detection, and impact zone solver. These approaches can now achieve impressive results on a few interacting cloth layers, but are not well suited to scale toward a large number of highly interacting cloth surfaces, such as for instance, modeling garments in a washing machine where multiple garments are clashing back and forth toward each other. Indeed, these approaches correct collisions once they already occurred, and are limited by the complexity of the collision constraints they have to solve to retrieve an untangled configuration. Typically deep collision in complex scenarios may not be fully solved, and can then lead to a locking situation where triangles in collision remain interlocked at future animation steps. In this chapter, we show that the use of a field-based approach can robustly handle an animation with an arbitrary number of surfaces sliding on each other instead of intersecting.

In contrat with the standard pipe-line based on collision detection and response, our method does not solve existing collisions but rather avoids them beforehand, and can even run with existing collisions without being trapped in locking situations for the rest of the animation. Our method relies on the use of a vector field modeling a single velocity for a given position in space. This vector field is computed to match the simulated cloth velocity at their specific surface positions, but is constrained to be globally divergent free in space, or even to have positive divergence when surfaces are close-by and are moving toward each other. The vertices of the surfaces embedded in this field are displaced along the newly computed velocity, thus modeling a collision free deformation and, possibly, a separation behavior.

The use of divergent-free vector fields to guide shape motions is very generic as it is independent of both the types of objects and the simulation, which is solely used to provide velocity constraints. As long as the field resolution is computed accordingly, this approach can handle, in a unified way, an arbitrary number of interacting surfaces moving and sliding back and forth towards each other, as well as avoiding self-intersection.

Furthermore, the velocity field resolution can be chosen independently of the number of vertices of the surfaces embedded in it. Therefore the computational time of the approach is mostly independent of the mesh resolution and the number of layers. While projecting the entire velocity field directly onto a divergent free constraint, similarly to a incompressible fluid model [Sta99, EMF02], allows to guarantee an intersection free behavior, the general motion of the embedded surfaces would be highly impacted, even in regions that are not in a quasi-collision state. In addition, deformable surfaces like garments are sparsely defined in space and do not need to be embedded in a dense velocity field. These two reasons lead us to propose a solution enabling to locally adjust such sparse velocity field towards null divergence.

The key idea of our method is to efficiently localize and set adapted constraints on the divergence such that they only act in the vicinity of nearby surfaces where collisions may happen, while automatically adapting to the proximity and velocity between surfaces. Our contribution consists in extending the use of divergent free, and divergent constrained, vector fields to arbitrary collision avoidance between deformable surfaces guided by a simulation. To this end, we first propose a local shape density expressed as a scalar field, estimating, in a unified way, surface proximity, ie. proximity between different surfaces as well as self-proximity. Second, we propose a spatial localization mechanism enabling to detect where collisions are likely to happen from proximity and velocity clues, and where the velocity should therefore be modified. This localization computation does not require any geometrical intersection computation, and is only computed from the density and velocity field. Third, we propose a set of linear constraints to be applied on the surface velocities to avoid collision and handle friction. Thanks to the linearity of the constraints, the solution can be expressed as a sparse least square problem and efficiently solved, while remaining robust to challenging situations with several rapidly moving garments.

## 3.1 Method Overview

Our method aims at avoiding collision between deformable surfaces representing garments in modifying their velocity at every time step of an animation.
Let us suppose that the surfaces are represented as triangular meshes. Each mesh is associated with a physically-based simulation that computes a velocity for each vertex. These velocities will be used as a target velocity, given as input to our method. Note that the simulation does not need to consider any collision nor self-collision between objects, and can therefore be computed separately on each garment. Furthermore, the remainder of our method is independent from the choice of simulation model and solver, which can be adapted to the type of garments we wish to represent. In our case, we considered an elastic simulation solved using Position Based Dynamics [MHHR07].
From the input our current vertex positions and target velocities along all surfaces, we build two discrete fields on a 3D grid:

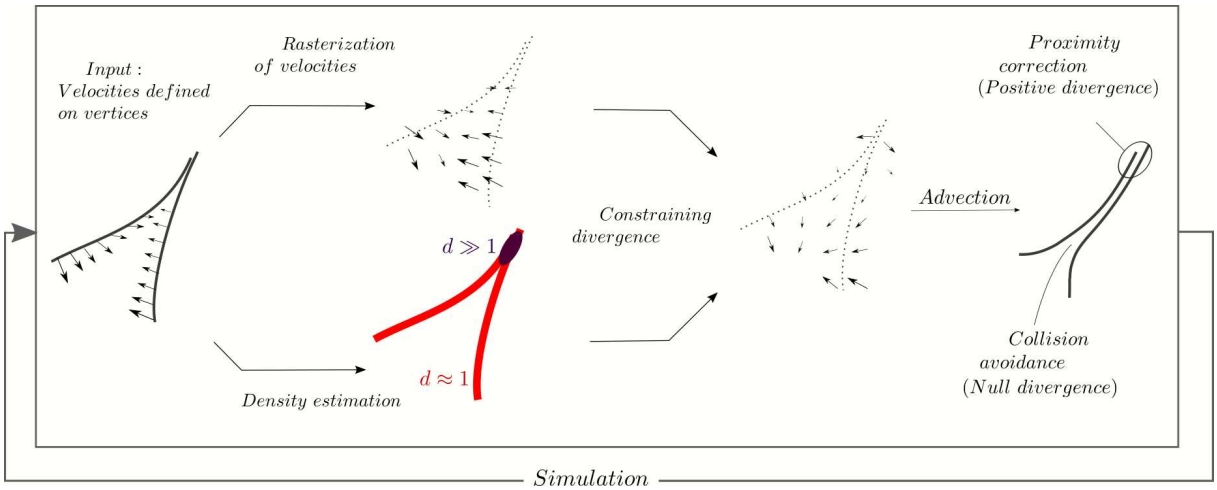


Fig. 3.1: Our method takes as input meshes with velocities at their vertices, and computes a vector field constrained to avoid imminent collision and correct proximity. The meshes are then advected by this vector field.

- First, a discrete vector field computed in rasterizing the simulated per-vertex velocities onto the grid. This field is initially solely defined in the discrete neighborhood of the garment surfaces.

- Second, a discrete density field, ie. a scalar field whose value indicates the proximity between different surfaces, or between a folded surface and itself (see Section 3.2).

The next, and most important, step of our approach consists in modifying the velocity field. To this end, a set of linear constraints are expressed on each voxel of the grid, and are locally adapted depending on the value of the density field. More precisely, when the density value indicates than two or more surfaces are approaching, the divergence of the velocity field is constrained to be null, or even set to a positive value to prevent the collision and separate these surfaces. The discrete velocity field is modified globally in solving the resulting, sparse linear system. This step is described in more details in Section 3.3.
The final step is to assign to each vertex a new velocity, interpolated from the grid of corrected velocities. Each vertex is then advected along its velocity, and a new simulation step is performed from which we restart our collision algorithm.
The general pipeline of our method is inspired from [MSW+09] and [NGCL09]. It differs in that our approach handles surfaces, while they respectively focus on curves and particules. More precisely, we adapted the rasterization of velocities and the measure of density to handle surfaces such as clothes.

## 3.2 Defining discrete velocity and density field

In the following, we consider a cubic Cartesian grid indexed by $(i, j, k)$ indices and set to be sufficiently large to cover the entire domain where the animated surfaces are allowed to move. Calling $h$ the edge-length of a voxel, the 3D position of a grid cell can therefore be expressed as $g_{i,j,k} = h\,(i, j, k)$ assuming the first voxel is at the origin. Note that only sparse information is stored for both velocities and density fields. Therefore only non-zero values are explicitely stored.

### 3.2.1 Rasterizing the velocity

Given a set of triangular meshes with prescribed velocity at each vertex, we rasterize this velocity in the neighboring cells of the triangle. To this end, we inspired from the method from McAdams et al. [MSW+09]. However, while their approach is defined for curves assumed to have globally coherent speed (the speed of the hair), we extend the approach to work on triangles exhibiting possibly large speed differences (since they may belong to different pieces of clothes).
Let us consider a triangle indexed by $n$. We note $p_{n,(i,j,k)}$ the closest point from $g_{i,j,k}$ on the triangle $n$. The velocity $v_{i,j,k}$ stored at $g_{i,j,k}$ is computed as

$$v_{i,j,k} = \frac{\sum w_{n,(i,j,k)} v_{n,(i,j,k)}}{\sum w_{n,(i,j,k)}} \tag{3.1}$$

where

- $v_{n,(i,j,k)}$ is the velocity at position $p_{n,(i,j,k)}$ on the triangle n, computed as the interpolation of the vertex velocities of the triangle using barycentric coordinates.
- $w_{n,(i,j,k)}$ is a weight modeling the influence of $v_{n,(i,j,k)}$ over the distant position $g_{i,j,k}$.

Following the definition of McAdams et al. [MSW+09] we consider a linearly decreasing influence with respect to the distance between the voxel position and the triangle. However, we also consider the triangle speed in order to propagate the triangle influence further away when its speed is high.
In practice, we compute the integer value $l_n = 1 + ceil(\frac{dt}{h}||v_{n,(i,j,k)}||)$ for the triangle n, representing the number of consecutive neighboring grid cells that are influenced by the triangle. Then the weight is expressed as $w_{n,(i,j,k)} = max(\frac{(2l_n-1)}{2}h - d(p_{n,(i,j,k)}, g_{(i,j,k)}), 0)$, with $d$ the euclide an distance. We can finally define for any voxel $(i, j, k)$ a weight as $w_{i,j,k} = \sum_n w_{n,(i,j,k)}$.

**Kinematic obstacles:** Rigid volumetric kinematic obstacles are handled by rasterizing their local velocities at their discrete boundaries, ie. the first voxels directly outside their volume, weighted by a virtually infinite value. In practice, this enables to ignore velocities of surrounding clothes in those voxels, and impose the speed of the kinematic rigid bodies.

### 3.2.2 Rasterizing density

We introduce a discrete density field $d_{i,j,k}$ indicating proximity between different surfaces, or between a surface and itself. This field is used in the next section to localize possible collision regions, as well as guiding the divergence value of the velocity field.
We compute this density field such that its value be equals to 0 in empty regions, 1 on the position of a single surface, and greater than one when several surface parts are close to each other. Note that a collision between two surfaces lead to a value of 2 at the intersection position. This field should also take into account the notion of garment thickness in its definition, and should be fast to compute on the grid.

The general idea is to sum the contributions of oriented density samples scattered on the mesh surfaces.
To this end, we precompute a Poisson sampling of radius $rad$ on the garment surfaces. As garment surfaces do not exhibits large variation of lengths through the animation, these samples are only computed once and stored as barycentric coordinates on their corresponding triangles. In our scenarios we empirically found that $rad = h\sqrt{3}$ provides a sufficiently dense sampling without arbitrarily increasing the computational cost.
Then each sample $m$ is associated with a density function $D_m(p)$ at position $p$. $D_m$ is expressed as an exponentially decreasing function of the distance between $p$ and the sample, with an anisotropy expressed in the normal direction $n$ of the surface:

$$D_m(p) = exp(-||SR(p - p_m)||^2) \tag{3.2}$$

where $p_m$ is the position of the sample, R is the rotation matrix such that $Rn = (0, 0, 1)$, and $S = diag(\sigma_t, \sigma_t, \sigma_n)$ is a diagonal matrix expressing the anisotropy of the density. We choose $\sigma_t = \sqrt{3/2}h$ to ensure that the density of a single, non-self-colliding garment, does not exceed 1. And $\sigma_n = thick\frac{1}{\sqrt{(-ln(0.5))}}$ modeling the thickness of the cloth.
The final density d is computed as the sum of all the contributions $d(p) = \sum_m D_m(p)$, and is stored in the discrete grid $d_{i,j,k} = d(g_{i,j,k})$.

## 3.3 Collision avoidance using a constrained velocity field

We detail in this section how we compute the new velocity field from the input one and the density field. The clothes are then advected in this field for collision avoidance.

### 3.3.1 Localizing the constraints

As explained in the introduction, our method applies different constraints locally depending on the evaluated risk of collision and density of interaction between surfaces. This

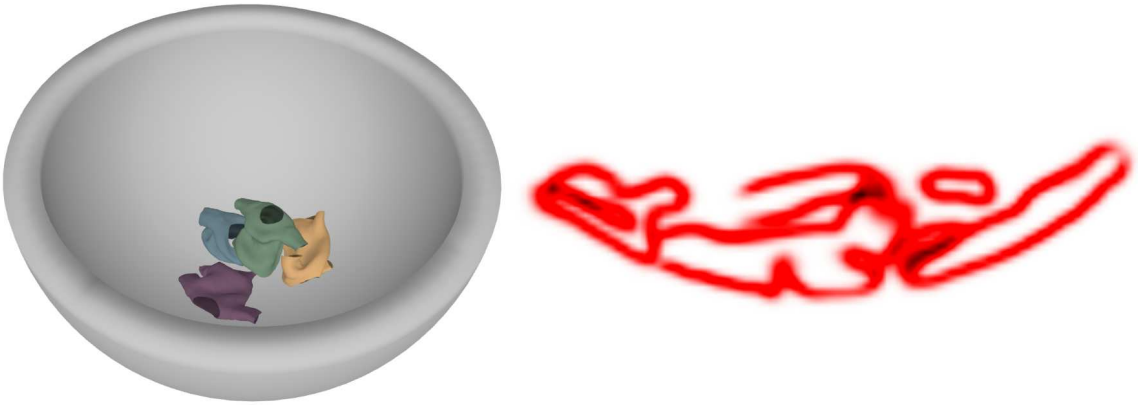

Fig. 3.2: On the left, a few meshes at the bottom of a bowl. On the right, a 2D view of the density, visualized on a plane cutting the scene. Red value represent a density close to 1, while darker shade represent density superior to 1.

localisation is expressed as a subset of the grid, and is fully defined from the input velocity field $v_{i,j,k}$ and density field $d_{i,j,k}$.
Note that in the following, we will use the same notation for continuous operators such as div, and their discrete counterpart expressed using finite difference.
We consider in our approch three regions noted $V_0$, $V_1$ and $V_2$ as follows :

- $V_0$ corresponds to the union of the neighborhood of the clothes surfaces and of the voxels on their trajectories. Using the notation of Section 3.2, it can be defined as the set of voxels for which the weight $w_{i,j,k}$ is non-zero:

$$V_0 = \{(i,j,k) | w_{i,j,k} \neq 0\} \tag{3.3}$$

- $V_1 \subset V_0$ corresponds to the set of voxels where a possible collision may occur. It first contains the voxels in which density is superior to 1, as this indicates nearby surfaces. Moreover, we want to allow the separation of surfaces : several surfaces might naturally move away from each other without any action from our method. In that case, defining a constraint on the divergence of the velocity would introduce an artificial bond between them.
  To further consider regions where surfaces are currently approaching toward each-other, we add the condition that the initial velocity has a negative divergence, thus leading to

$$V_1 = \{(i,j,k) \in V_0 \mid d_{i,j,k} \geq 1 \; or \; \nabla . v_{i,j,k} \leq 0\} \tag{3.4}$$

- $V_2 \subset V_0$ corresponds to the set of voxels in the neighbors of surfaces on which friction should be applied. We typically consider the discrete neighborhood of rigid obstacles.

### 3.3.2 Velocity constraints

Given the three possible localization, we are now looking to express a new velocity $\tilde{v}_{i,j,k}$ that preserves, at best, the initial simulated velocity $v_{i,j,k}$ while avoiding possible collisions. To compute this velocity efficiently we consider three linear constraints on $\tilde{v}_{i,j,k}$.
First, we consider a general *anchoring constraint* used to maintain the initial velocity of the garment surfaces when no other constraint apply. This constraint is applied in all the voxels on the trajectory of the clothes, along with their direct neighborhood, namely $V_0$ :

$$\forall (i,j,k) \in V_0,\ \tilde{v}_{i,j,k} = v_{i,j,k} \tag{3.5}$$

The second constraint aims at setting a specific value for the divergence of the velocity. This value shall be null in order to avoid collision, or positive in order to force surfaces to separate from each other. This constraint is applied in the region $V_1$:

$$\forall (i,j,k) \in V_1, \nabla.\tilde{v}_{i,j,k} = s_{i,j,k} \tag{3.6}$$

with $s_{i,j,k} = max(0, h(d_{i,j,k} - 1)/dt)$ acting as a virtual *source of velocity* which increases linearly with the density. In particular, when $d_{i,j,k}$ approaches 2, meaning that a collision is about to happen, the divergence term is similar to applying an impulse of magnitude $h/dt$, effectively aiming at separating objects by 1 voxel during the time step dt.
Finally, the last constraint is used to model friction at the neighborhood or kinematic obstacles. As mentioned in Section 3.2, the voxels in $V_2$ store the rasterized velocity of their neighboring kinematic objects. We use a null-laplacian constraint in order to smoothly transition to these velocities when approaching these voxels :

$$\forall (i,j,k) \in V_2, \Delta.\tilde{v}_{i,j,k} = 0 \tag{3.7}$$

In order to better avoid collision with kinematic object, we weighted the *anchor constraint* in these voxels. In practice, this amounts to multiply both side of the anchor constraint by a weight $w_{kinematic}$. In our example, we typically fixed $w_{kinematic} = 5$.
The conjunction of these constraint gives an overconstrained sytem of linear equations over the $\tilde{v}_{i,j,k}$, that we solve in the least square sense using a conjugate gradient algorithm.

### 3.3.3 Length-preserving velocity advection

The last step of our algorithm is to advect the vertices of the mesh along the newly computed velocity field $\tilde{v}_{i,j,k}$.

To avoid unwanted length distortions of the garment surfaces during the advection process, we compute it in several substeps – about 10 in our experiments. Each subset interleaves a small vertex displacement along its velocity computed as a tri-linear interpolation of $\tilde{v}_{i,j,k}$, with a constraint of edge-length preservation. In our case, the edge-length constraint is solved using a similar Position Based Dynamics approach that we used to initialize the velocity, but in considering in this case very few iterations over the length constraints.

## 3.4 Results and discussion

We present in this section the results of our approach applied to various cloth surfaces animations.

### 3.4.1 Qualitative results

We first illustrate the results of our approach on a 2D animation in Fig. 3.3 where two curves are initially moving toward each other. As shown in Fig. 3.3-right, applying our local divergence contraints on the velocity fields allows to make the curves deform to respond to upcoming collision, rather than merely crossing each other.
Fig.3.4 shows a 3D simulation where two squared pieces of cloth fall under gravity on each other between various spherical rigid obstacles. Note that our method handle collision avoidance between a surface and the rigid sphere, the plat ground, or the other sheet in a unified way using velocity constraints at the grid level. The animation ends-up in a stable state where the two cloths lies on top of each other.

As our approach computes collision (and self-collision) avoidance in a local and unified way, it can handle 3D scenes containing a large number of deformable surfaces in contact, or even constantly moving and sliding on each other.
We present two examples of such challenging animations in Fig.3.5 and 3.6. Fig.3.5 describes up to 60 t-shirts falling on a rotating bowl that periodically change of rotation direction. This scene involves complex motions and shape deformation that can be noticed in the last image and does not suffer from locking or numerical divergence issues. In addition, the efficiency of the linearly depends on the number of triangles, and is almost independent from the complexity of the apparent contacts between the surfaces.
Fig.3.6 illustrates 60 T-shirts in a "washing-machine" represented by the surrounding rotating half-cylinder. In this case, the fast motion of the cylinder induces fast local velocity on the deformable surfaces, and thus lots of chock-like motions toward each others. in-

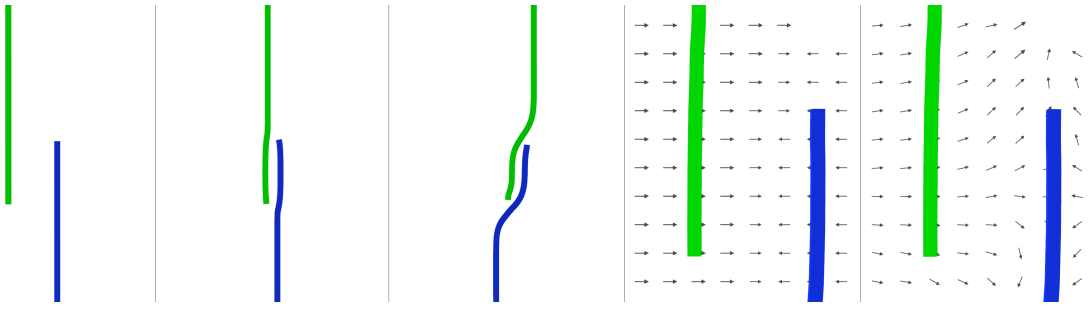

Fig. 3.3: Left: Three frames of a 2D animation where the green and blue curves are initially moving toward each other. Right: Close-up of the animation with a display of the velocity field before, and after, application of our constraints. the result is assymetric because the right cloth has initially a higher velocity than the left one.

teractions.

**Presence of collisions** We can note that, although limiting them, our approach does not guarantee to be fully free from collision and self-collision throughout the animation. First, our constraints on the velocity are solved in a least-square sense, therefore the solution may still exhibit negative divergence values locally. Second, advection is only computed using discrete time steps, so vertices integrates errors with respect to streamline paths. Third, and most importantly, the divergence constraint is discretized over the voxels of the grid. This leads to a distance of a few voxels maintained between the clothes. However, if the voxel size is small with respect to the size of the edges of the cloth meshes, the length preservation contraint interleaved with the advection mentioned in Section 3.3.3 may resault in displacements spanning one or several voxels.
While our approach cannot guarantee the absence of collision, the overall computation is fully independent from the existence of collision between the embedded surfaces. This is actually a benefit of our method and a key element of its robustness, enabling to handle very complex scenes with lots of intricated surfaces, where collisions would be very hard, or time consuming, to explicitly avoid and handle. In fact, the same process that causes collision -the relaxation step mentioned above- is responsible for their resolution as the simulation goes on, and surfaces in collision then continue their motion, guided by their individual simulated velocity and the mesh-based length preservation constrained during the advection, without necessarily locking to each other, nor diverging as the least-square problem remains well conditioned. Contrary to other approaches based on explicit geometric collision computation, our approach is thus able to seamlessly animate embedded surfaces that possibly collide. In several examples, and as illustrated in Fig.3.7, one may observe that shallow collisions between surfaces may occur during a few time step, but then seamlessly resolve themselves thanks to the continuation of their respective motion.

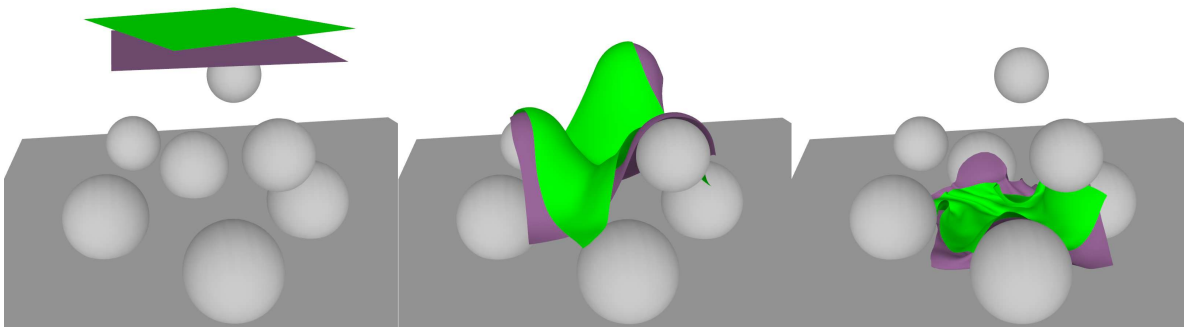

Fig. 3.4: Two pieces of cloth falling on rigid obstacle and converging toward a stable final rest state.

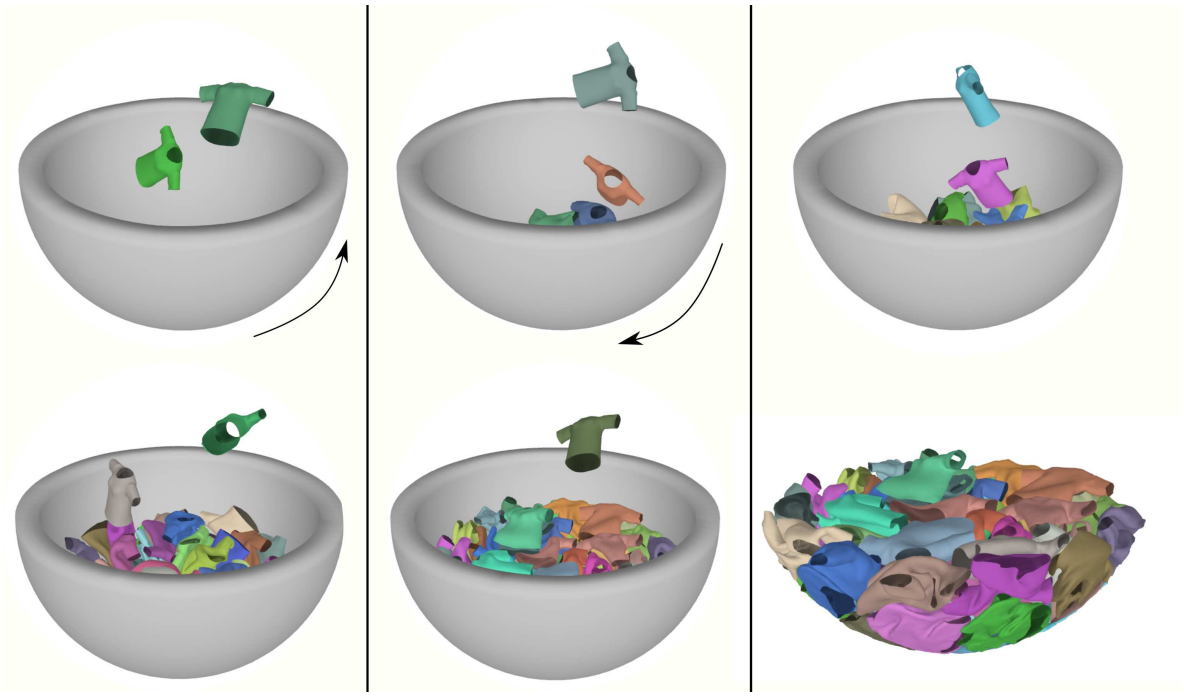

Fig. 3.5: 60 T-shirts agglomerating on top of each other in a rotating bowl. Each T-Shirt is spawned every 45 frames. Bottom left : a zoom without rendering the bowl.

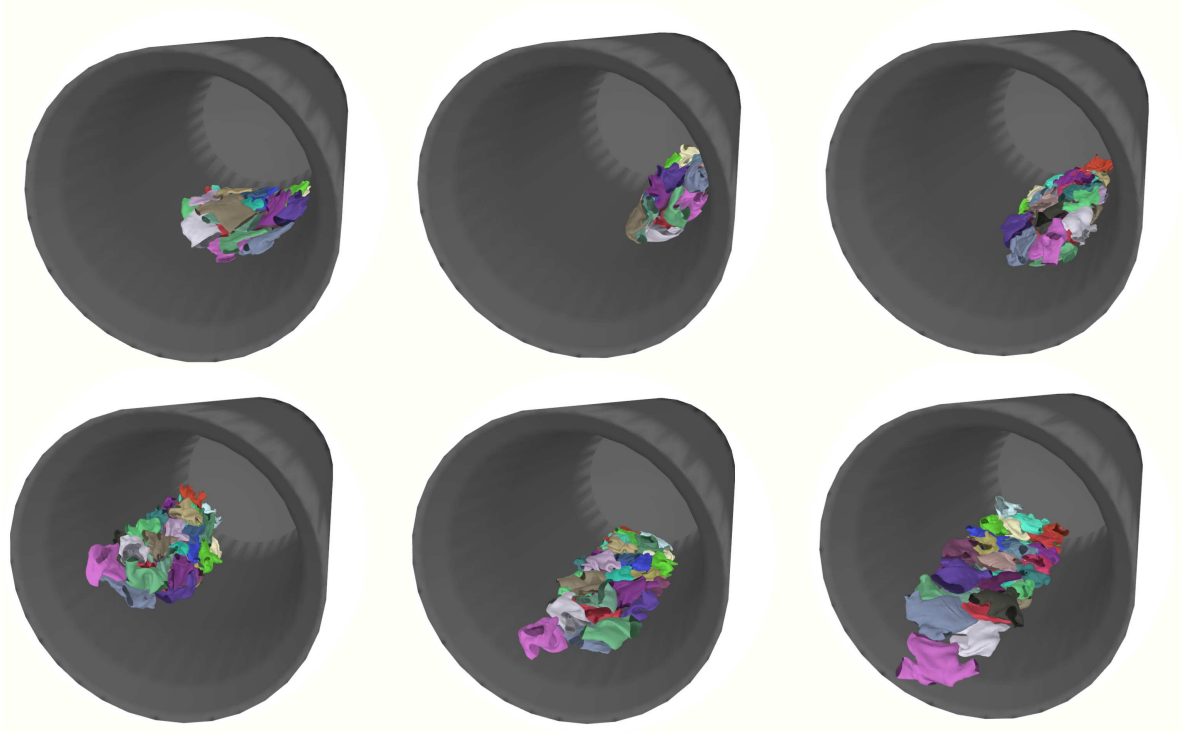

Fig. 3.6: 60 agglomerated T-Shirts are dropped inside rotating cylinder modeling a washing-machine.

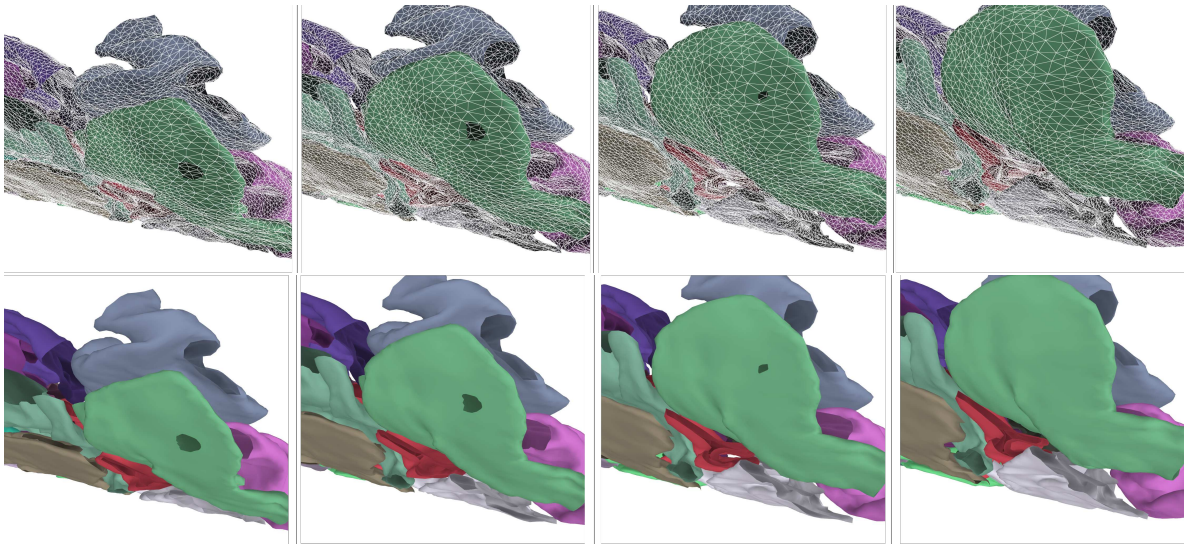

Fig. 3.7: Zoom on a mesh of figure 3.6 during a few frames of simulation. Self-intersection does locally occur, but disappears after a few frames.

**Influence of the positive divergence** As previously noted, the divergence constraint is only approximated in our approach. To further enforce near-by surfaces to move away from each other, we apply a positive divergence term $s_{i,j,k}$ in Equation 3.6. This term helps to compensate for non-perfect collision avoidance, and can be understood as the effect of "insuflating artificial air velocity" between the surfaces. Fig. 3.8 compares the results obtained when using only $s_{i,j,k} = 0$ (enforcing divergence-free only), with respect to the use of our locally positive divergence. We can note that this term largely limits intersections between surfaces and leads to a less compact set of surfaces with thin air layers between them.

**Thickness parameterization** The thickness parameter exposed in Section 3.2 can be used to model a thickness attribute for each cloth surface in increasing the nearest distance between surfaces with increasing *thick* value. This effect is illustrated in Figure 3.9 on a 2D-simulated scenario. Because this also increases the distance for self-collision, this does also model in a way the rigidity of more thick clothes.

### 3.4.2 Computational time

All times measured in this chapter were taken on a standard laptop computer with an Intel quad Core i7 CPU, clocked at 3.1GHz with 32GB of RAM. Our software uses up to 1Gb of RAM memory at run-time for the presented examples (mostly due to the rasterizing of velocities and densities inside a sparse 3D grid). We used OpenMP to trivially parallelize rasterization and advection. We used the Eigen library for all matrix related operation, and especially its Least Square Conjugate Gradient implementation to solve

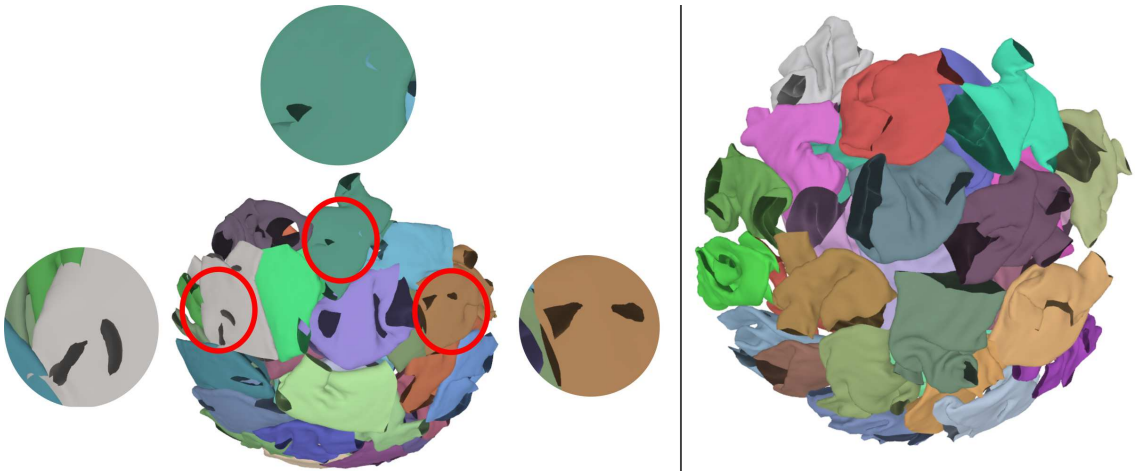

Fig. 3.8: The use of our positive divergence term (eq. 3.6) helps to prevent collision between packed surfaces (right) compared to a simple null-divergence approximation (left).

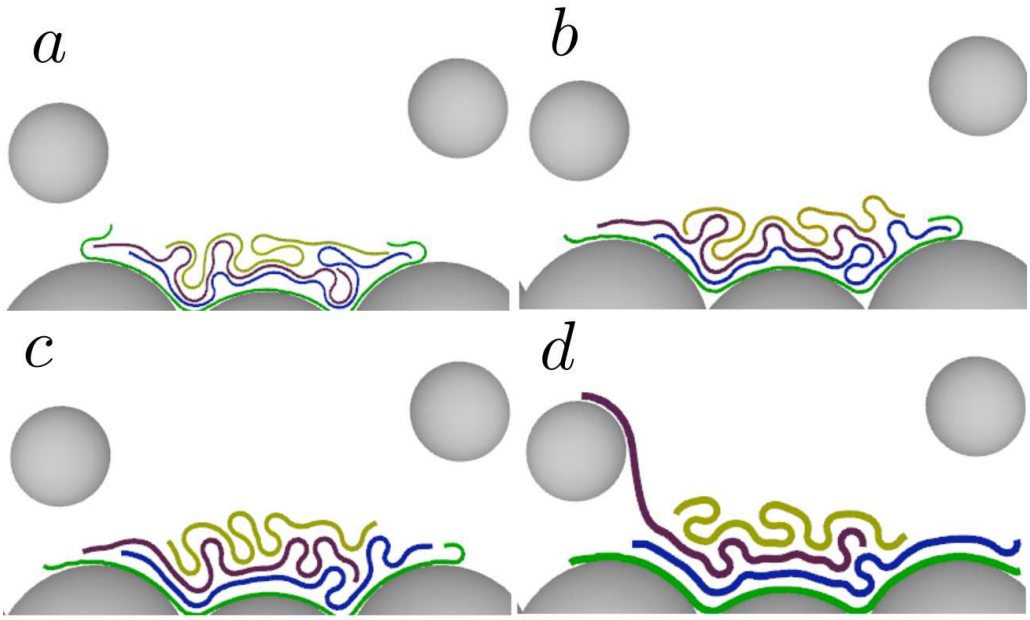


Fig. 3.9: A 2D animation illustrating the influence of an increasing thickness parametr. Note the increasing space between the layers.

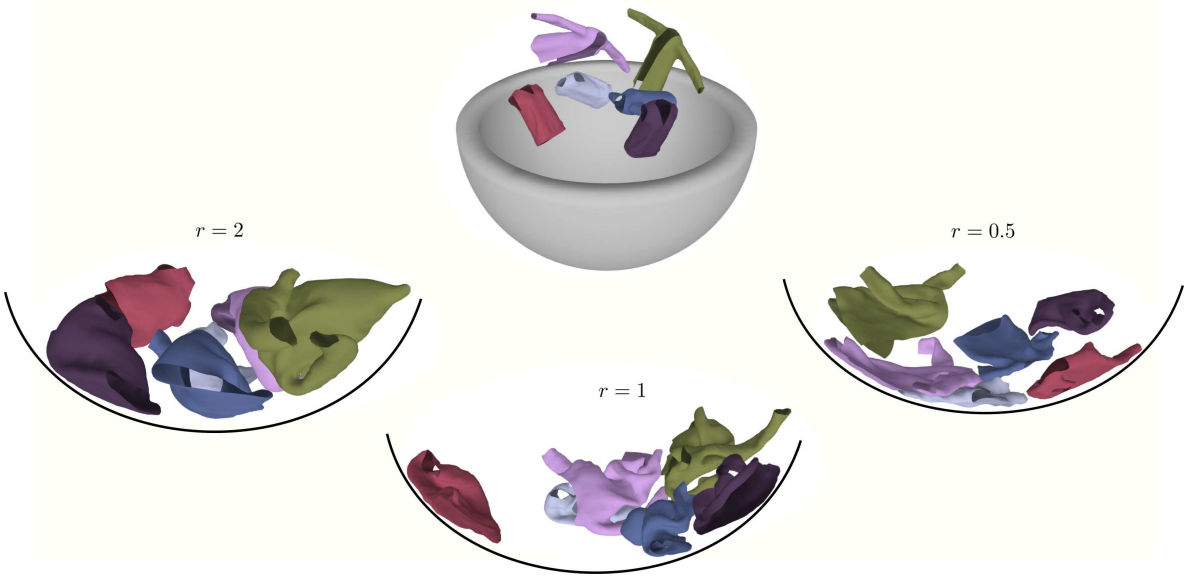


Fig. 3.10: A few garments are dropped inside a bowl. Bottom left, middle and right : three results after a few frame of simulation using different voxel size (and thus different ratio r between the voxels size and the mean size of edges of the meshes). Note how high r rigidifies the clothes and maintains large distances between them.

our least square formulation.

One of the most important parameter on which depends the computation time of our method is the number of voxels in which a velocity has been rasterized : increasing this number also increases the size of the linear system to be solved.
This number of voxels is not directly controlled, but through the size of the voxels used in a scene $h$: if the grid we used was dense, the number of voxels would typically increase with $\frac{1}{h^3}$. However, only the voxels with non-null $w_{i,j,k}$ (using the notation of Section 3.2) are actually considered in the method. Because those voxels are located around the surfaces and on their trajectory, their number actually increases with $\frac{1}{h^p}$ with $2 < p < 3$. Figure 3.10 shows an experiment we did to evaluate this relationship, and the related increase in computation time: given fixed meshes, we define $e$ as the mean edge-length of the triangles composing the meshes. Then, we define $r = \frac{h}{e}$ the ratio between the size of the voxels and $e$. We simulated an animation involving these meshes using different values of $h$ (and so different values of $r$). We counted the mean number of voxels during each simulation: as showcased in Figure 3.11, this number varies between 5000 for $r = 2.8$, to 430k for $r = 0.2$, effectively varying almost proportionally to $\frac{1}{h^2}$.
We also measured the mean computational time per frame for each simulation. As expected, this time also increases quadratically as the size of the voxels diminishes, increasing their number.

The other parameter influencing the computation time is the number of triangles.

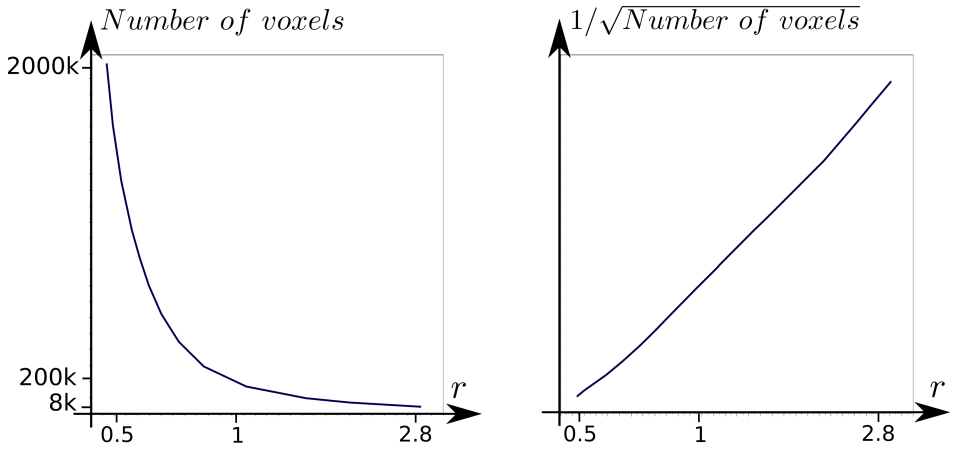


Fig. 3.11: Graph showing the increasing of voxels when decreasing their size in the scene of figure 3.10.

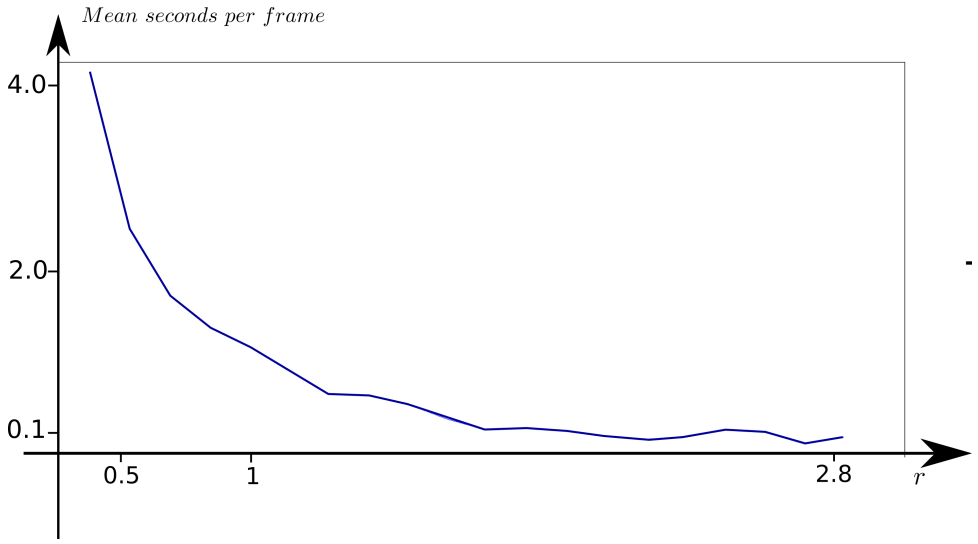


Fig. 3.12: Mean running time of the simulation of Figure 3.10 in seconds per frame with respect to the ratio r between the mean size of edges of the clothes and the size of the cell h. As the size of the voxel decreases, the computational time quadratically increases. A minimum time is reached for high r as a minimal rasterization is done for every triangle, regardless of voxel size.

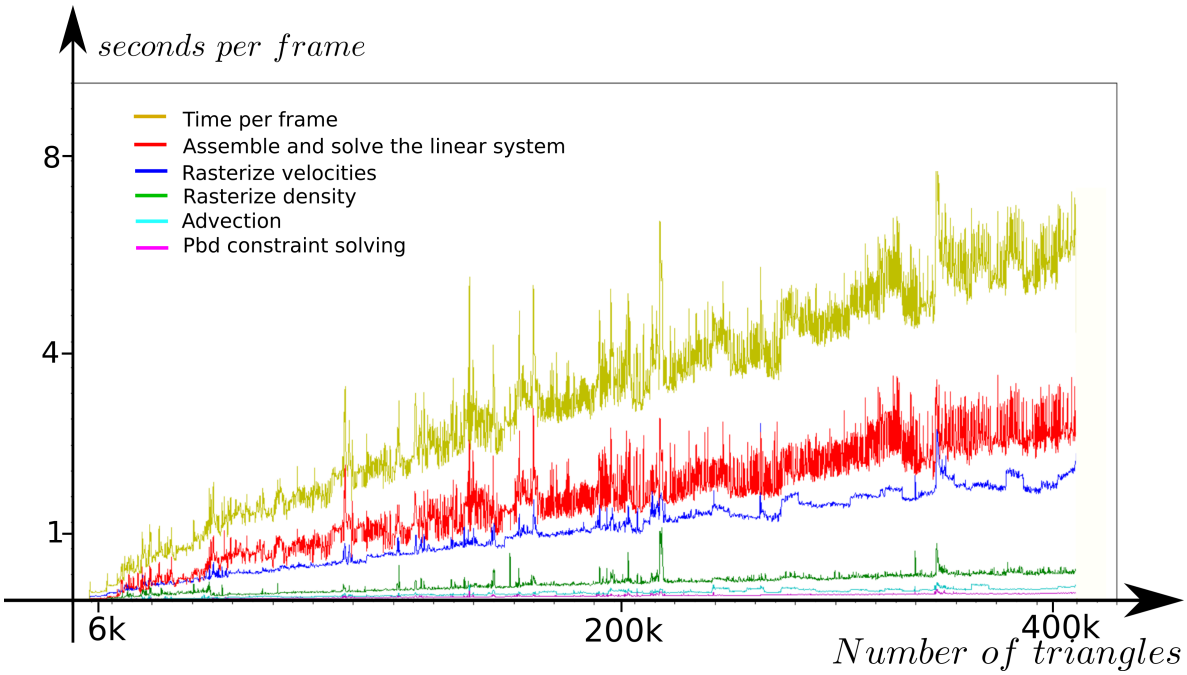


Fig. 3.13: Running time in seconds per frame for the example of Figure 3.5. The T-shirts are spawned throughout the simulation, which increases the number of triangles by a constant value each time. With a constant cell size h, our method exhibits empirical linear complexity with respect to the number of triangles.

Given a fixed voxel size $h$, we show in Figure 3.13 that the computational complexity of our method scales linearly with the number of triangles. Our typical examples involving 2 surfaces of about 5k triangles each using a grid size of 0.2 - we typically choose $r = h/e = 1$ in our examples - were computed at about 50 to 100 ms per frame. More complex simulations such as the bowl of Figure 3.5 or the washing machine (Figure 3.6) totalling 350 to 400 thousands triangles went up to 8 seconds per frame. The time spent assembling and solving the linear system of constraints typically represents half of this time, with the other half being mostly shared between the rasterization of velocities and of densities.
For comparison sake, Li et al. [LTT+20] used multiple GPU to perform between 0.2 to 0.9 second per frame on average on example ranging from 500k to 1.6M triangles. Their result however depends heavily on the complexity of the contact involved, and they perform better the lesser the surfaces are close to each other (and thus the lesser they are prone to intersect each other), while ours depends mostly on the number of triangle and the grid resolution.

### 3.4.3 Discussion and extensions

As explained previously, and shown in Figure 3.10, our method does not provide a collision free-animation in all situations. While our approach usually handles well shallow-intersections, deep interpenetrations, which can happen when voxels size are too small with respect to the edges of the surfaces, might lead to surfaces that will remain tangled for the remainder of the animation.
Experimentally, the best trade-off between collision avoidance, visual quality, and efficiency seems to be reached when the grid cell size is roughly similar to the triangle edge-length. Indeed, taking large grid cells allows to achieve small computational time, and will even lead to fewer collisions, but the effect of the discretized velocity field will be more clearly visible. Our current approach empirically requires an interval of at least three voxels wide between the two surfaces, to act on their relative speeds and avoid interpenetrations (as can be clearly seen in Figure 3.3). Using large grid cell therefore leads to surfaces that stop far away from each other, and which will look thicker and stiffer than expected (shown in Figure 3.10).
Symmetrically, considering small grid cells will quadratically increase the computational time. In addition, while the vector field has more degrees of freedom to avoid large space between surfaces, it also leads to larger local displacements. These displacements then need to be corrected for length preservation in the advection step, which turns out to generate new intersections. In the future, adding extra smoothing constraints in the system should however be able to improve this behavior.

#### Directional divergence constraints

Related to this limitation, we can also propose, and already started to experiment, the use of anisotropic divergence constraints. While divergence-free vector fields are naturally

well suited for isotropic behavior such as incompressible liquids, garment like surfaces have differents behaviors in their orthogonal and tangential directions. They typically bulge in their orthogonal direction, and slide, but cannot extend nor compress in their tangential one.
Our idea is to better capture such surface behavior in introducing the notion of directional divergence expressed as

$$div((v.d)d) \tag{3.8}$$

where $d$ is an oriented normalized vector field.
First, considering the constraint $div((v.n)n) = s$, where $n$ is given by the local normal of the surfaces, may improve the quality of our result in avoiding vortices within the field, while still ensuring collision avoidance in their orthogonal direction (cf Figure 3.14). Our initial test with such constraints showed very promising results as collision avoidance quality was preserved as much as with the isotropic behavior, but led to faster least-square convergence (15% improvement), possibly associated to simpler constraints with less degrees of freedom. In addition, surfaces considered to be in contact slides on each other without additional numerical friction, initially caused by the isotropy of the classical divergence constraint. Still, we could observe some discretization artifacts due to orientation directions that were not aligned with the axis of the grid, leading to unexpected deformation of the surface even when no external force was applied. In addition, when collision still occured between two surfaces, the possible sudden inversion of normal directions introduced large variations in the velocity field. Therefore, this method may require specific adaptation in the future to blend between directional and isotropic divergence.
Second, considering constraints oriented along the tangent planes of the surfaces could help modeling length-preserving behavior of the surfaces. Let us consider a 1D curve with tangent vector $t$. Applying the constraint div( (v.t) t)=0 is similar to constraining the length of the curve to remain constant. Extending this constraint in a surface could be expressed by the use of two as-orthogonal-as-possible tangential vector fields t1 and t2, and then simultaneously using two directional constraints div((v.t1)t1)=s1 and div((v.t2)t2)=s2. While such fields may not always be easy to compute and can lead to opposite constraints, another possibility would be to consider a single constraint on the tangent plane div(v-(v.n)n)=s, which would lead to a relaxed constraint targeting only area preservation.

#### Unified approach for collision handling

Another extension we further started to explore is the use of our generic approach to handle collision between other types of shapes beyond garment-like deforming surfaces. In Figure 3.15 we illustrate the use of our approach for rigid objects.
While rigid objects can be simulated individually using standard rigid body models, the key idea is that our approach can handle both deformable surfaces and rigid objects

within the same framework. In the illustrated case in Figure 3.15, collision between the rigid bodies is fully handled using the divergence constraint of their velocities, while their rigid behaviors are enforced through the use of a shape-matching algorithm [MHTG05]. Similarly to garments, collision between the shapes can still occur and should be further investigated to ensure that the objects can separate after a few steps.

#### Improving efficiency

Another improvement could focused toward improving the computational cost of our method. The use of multi-grid methods [Wes95] should allow to iteratively solve for the low-frequency solution in a coarse grid, and then use these intermediate results to compute high-frequency features at higher resolution. This approach would not only allow accelerating convergence, but possibly allow a finer control on the spatial frequency of the velocity field on more defined grids. Specific implementation in the case of sparse grids is however still in progress.
Finally, while our computational cost and artifacts are mostly limited by the grid resolution and associated discretization, using adapted continuous basis functions of divergence-free vector fields, such as the one proposed in the recent work from Eisenberger et al.[ELC18], could be of great interest. Their use in our context still remains an open question, but they could ultimately allow fast and concise representation of low frequency solutions, while providing an exact-continuous representation compatible with artifact-free derivative operators.

## 3.5 Conclusion

In this chapter, we presented a method based on vector fields to avoid collision between any number of deformable meshes. The velocities of the clothes are rasterized within a grid, along with a measure of their spatial density. The divergence of the resulting velocity field is then constrained to a locally null value to preserve the air volume between close enough surfaces and avoid collision, or to a positive value to separate too close objects. This allows to simulate any number of meshes potentially interacting with each other in challenging cases. While the method presented in Chapter 2 was not able to handle self-collisions, and was thus only applicable to static intersections of differents meshes, the method presented in this chapter handles in a unified way collisions and self-collisions by the mean of a generalized density estimation.

The actual precision of the method is, however, limited by the use of a discrete rasterization of the velocity field in a 3D grid, as well as a least square solution that cannot guarantee the lack of collision. These limitations generates artifacts when the grid size is respectively too large, or too small, with respect to the triangle edge length. Several improvements have, however, been proposed and remain currently under investigation to

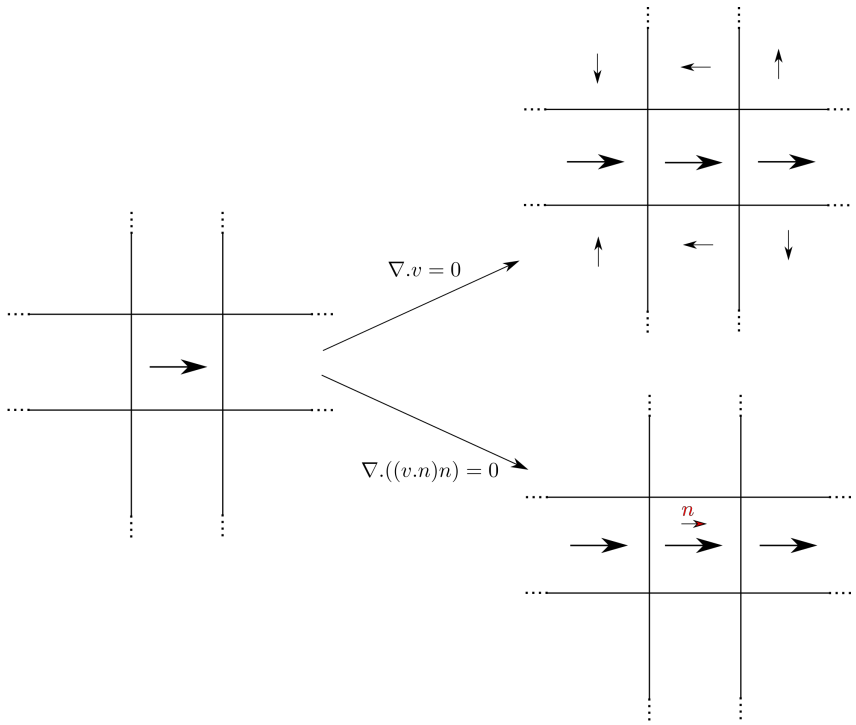


Fig. 3.14: On the left, a simple anchor constraint on the middle voxel. On the right, the result when adding a divergence constraint: on the top result, classic null divergence is wanted, which result in vortices appearing around the middle voxel. On the bottom, directional divergence is used and no vortex appear.

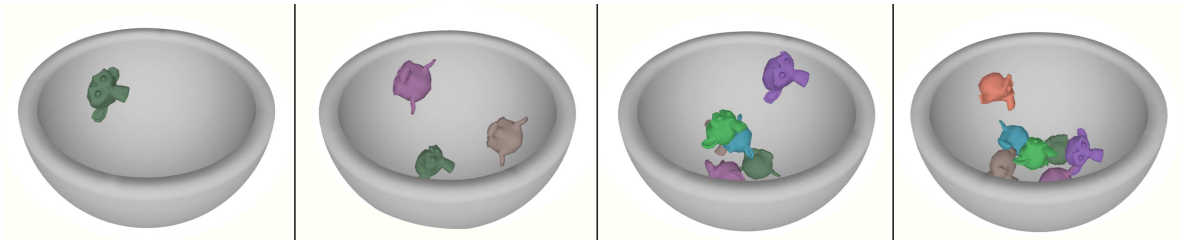

Fig. 3.15: Simulating 7 rigid monkey heads with our method by applying shape matching between each frame.

limit such artifacts and allow to alleviate the dependence to the grid resolution.
In addition, while the divergence of the velocity field has only been used to avoid intersection, its use along specific directions may be able to capture length preserving behavior, allowing to integrate this constraint within the unified framework, and without additional cost.

While our algorithm brings interesting theoretical computational complexity compared to standard methods, our actual timings remains roughly an order of magnitude slower than state of the art collision correction. Indeed, mesh-based approaches have been highly and iterativelly optimized through years, to reach dedicated multi-GPU implementations [LTT+20]. On the other hand, our prototype was not optimized for efficiency, and is running on CPU only. Due to their natural local structure that can be easilly parallelized, field based approaches should however be very good candidates for highly efficient GPU friendly implementations, and optimized implementation of vector-field based collision handling would also be an interesting avenue for future work.

# Conclusion

In this work, we proposed two approaches to the intersection-free animation of clothes. These methods stand apart from existing ones through the use of volumetric fields as the main structure to handle the collisions. Standards methods usually split the detection of impending or occured collisions, and their prevention or correction. In contrast, the use of volumetric field allows our methods to unify these two steps, bypassing at the same time the need for using various acceleration structures and the need of defining an arbitrary order of correction, affecting results.
Chapter 2 proposed a method dedicated to the static untangling of layers of garments using scalar fields. Deep initial intersections are solved thanks to a novel implicit representation for open surfaces such as clothes, which gives information about the depth of collision between the layers. Such representation allows to replace intersections by contact surfaces : inspired by contact modeling approaches such as [VGB+14] or [ATW+17], we extend them by proposing a N-ary composition operator modeling contact between an arbitrary number of subsequent layers. This provides an intersection-free starting point for a simulation, which is often needed in order to compute a valid animation. However, the use of scalar fields used to define the geometry of the shapes is associated to two limitations when applied to garment animation. First, the sudden change of geometry applied as collision correction does not integrate simulation parameters such as surface velocity. Second, self-collisions cannot be easily detected on a single isosurface, which would blend in collision areas. This led us to chose another approach to tackle the intersection-free animation of clothes, this time using a vector field.
This new approach is discussed in Chapter 3. We build on the process used in [NGCL09] or [MSW+09], which is to constrain a vector-field modeling the velocity of the objects in the scene, and adapt it to be used on deformable surfaces. The velocity field, initialized from a cloth simulation, is modified using dedicated linear constraints applied to its divergence and depending of a local density measurement. The volume between the garment surfaces is then either preserved or inflated in order to prevent collisions during the animation.

We have shown through this work that the use of volumetric fields share a common set of advantages.
First, they encode properties locally in space, which leads to computations that are independant from the discretization of the shape they embed. For instance, collision between isosurfaces encoded with their scalar field rely on the knowledge of interior/exterior which is a simple query of the field value, while surface moving towards each other can be detected from the divergence of the associated velocity field. As a result, we were able to

propose algorithms with computational complexity scaling linearily with the number of triangles, while raw collision detection would require quadratic complexity, and therefore be non applicable without some spatial sorting structure.
Second, they offer a robust model for handling intersections between objects. While the use of classical mesh-based methods needs dedicated treatment for special cases (different types of triangle intersection, extreme cases of intersection with geometry at the same location), using a volumetric representation often allows to alleviate those problems. Moreover, adding an arbitrary number of objects is done seamlessly, without having to care about an arbitrary order of correction. Finally, volumetric approaches do not suffer from numerical divergence caused by force-based approaches, or by degenerated primitives.
Third, those approaches are pretty generic. While we use some volumetric representation to represent deformable surfaces –either static as in Chapter 2 or dynamic in Chapter 3, we are free to include any kind of object in our method (albeit potentially necessitating a modification in the method, as for the case of rigid-bodies mentioned in Section 3.4.3).

However, these field-based approaches come with their own limitations. One of these is a trade-off between computational time and precision: either their computation is expensive in order to obtain closed-form field, arbitrarly precise but also costly to evaluate (as with HRBF in Chapter 2), or the choice is made to obtain a discrete representation, typically faster to compute but parameterized by its resolution. Our methods are a good example of this property: in Chapter 2, we analytically compute scalar fields encoding the contact between several surfaces. Their evaluation are quite expensive, but they accurately model the contact constraint we were aiming at. In Chapter 3, this time our vector field is not analytically computed, but instead is discretized over a grid. This makes the different constraints we want the vector field to follow pretty straight-forward to express and enforce, and also makes the resulting field evaluation computationally cheap. However this comes at the cost of features heavily dependant of the resolution of the discretization (such as the minimal distances between clothes, visual rigidity etc.).
Another limitation is that, in general, volumetric fields lack local control of results. Adding local control over simulation results is an interesting challenge that has been tackled by several works for specific problems. Examples related to scalar fields in modeling include [GBC+13] for implicit surfaces blending based on gradient, or [ZBQC13] for detail-preserving blending between skeleton-based implicit surfaces. In the case of vector fields used in animation, details are often handled through hybrid models mixing lagrangian modelisation (mesh, particule system etc.) with a vector field. As an example, we can cite how the hair in [MSW+09] is subject to geometrical intersection correction even though individual hair strands have been advected by a supposedly divergence-free vector field, or -in our method- how we interleave advection with length-preserving constraints solved on the meshes themselves: this showcases how vector fields have difficulties to handle the problematic of collision handling on their own.

We encountered specific problems in our research. In Chapter 2, the most important

one was the lack of self-collision handling. This effectively led us to use another type of field in Chapter 3. However we believe that a lot can still be done on that matter. First, self-collison has been tackled in the case of level-set surfaces by [MASS15], which proves that this is a solvable problem. Second, the advances in composition operators over the past few years are promising in showing that these operators can be created with high flexibility. While several works already modeled contact between different surfaces using a variety of composition operators, trying to model contact between a surface and itself using some *transformation* operator could be an interesting attempt to tackle this problem.
In Chapter 3, we presented some avenues for improvement concerning modeling of behavior such as edge-length preservation, directly inside the vector-field. However, the most problematic issue in our opinion is how our result are far from the state of the art in term of "minimal distance" between the surfaces. This is directly linked to the use of a discrete grid to store the velocity field, and a really interesting challenge would be to try to find some analytical formulation of the problem. As mentioned in Section 3.4.3, a basis for divergence-free vector fields is used for example in [ELC18], and we do think that trying to adapt their method to the case of collision-free animation would be interesting.

# Bibliography


[ACWK04] Alexis Angelidis, Marie-Paule Cani, Geoff Wyvill, and Scott King. Swirling-sweepers: Constant volume modeling. In *ACM SIGGRAPH 2004 Sketches*, SIGGRAPH '04, page 40, New York, NY, USA, 2004. Association for Computing Machinery. 25, 26

[AS07] Alexis Angelidis and Karan Singh. Kinodynamic skinning using volume-preserving deformations. In Dimitris Metaxas and Jovan Popovic, editors, *Eurographics/SIGGRAPH Symposium on Computer Animation*. The Eurographics Association, 2007. 25

[ATW+17] Baptiste Angles, Marco Tarini, Brian Wyvill, Loïc Barthe, and Andrea Tagliasacchi. Sketch-based implicit blending. *ACM Trans. Graph.*, 2017. 22, 34, 75

[AVGT12] Samantha Ainsley, Etienne Vouga, Eitan Grinspu, and Rasmus Tamstorf. Speculative parallel asynchronous contact mechanics. *ACM Trans. Graph., Proc. ACM SIGGRAPH Asia*, 31(6), 2012. 51

[BB03] Martin D. Buhmann and M. D. Buhmann. *Radial Basis Functions*. Cambridge University Press, USA, 2003. 15

[BBCW10] Adrien Bernhardt, Loïc Barthe, Marie-Paule Cani, and Bryan Wyvill. Implicit Blending Revisited. *Computer Graphics Forum*, 29(2), May 2010. 22

[BDS+03] Loc Barthe, N A Dodgson, M A Sabin, B Wyvill, and V Gaildrat. Two-dimensional Potential Fields for Advanced Implicit Modeling Operators. *Computer Graphics Forum*, 22(1):23 – 33, 2003. 20, 21

[BFA02] Robert Bridson, Ronald Fedkiw, and John Anderson. Robust treatment of collisions, contact and friction for cloth animation. *ACM Trans. Graph., Proc. ACM SIGGRAPH*, 21(3):594–603, 2002. 8, 10

[BGH99] Julien Basch, Leonidas J Guibas, and John Hershberger. Data structures for mobile data. *Journal of Algorithms*, 31(1):1 – 28, 1999. 11

[BHN07] Robert Bridson, Jim Houriham, and Marcus Nordenstam. Curl-noise for procedural fluid flow. *ACM Trans. Graph.*, 26(3):46–es, July 2007. 25

[BS00] Jules Bloomenthal and Ken Shoemake. Convolution surfaces. *ACM SIGGRAPH Computer Graphics*, 25, 11 2000. 22

[BWDG04] LOIC BARTHE, BRIAN WYVILL, and ERWIN DE GROOT. Controllable binary csg operators for soft objects. *International Journal of Shape Modeling*, 10(02):135–154, 2004. 21

[BWK03] David Baraff, Andrew Witkin, and Michael Kass. Untangling cloth. *ACM Trans. Graph., Proc. ACM SIGGRAPH*, 22(3):862–870, 2003. 13, 15

[Can93] Marie-Paule Cani. An implicit formulation for precise contact modeling between flexible solids. In *ACM SIGGRAPH*, pages 313–320, 1993. 21

[CGB13] Florian Canezin, Gaël Guennebaud, and Loïc Barthe. Adequate Inner Bound for Geometric Modeling with Compact Field Function. *Computer & Graphics (proceedings of SMI 2013)*, 37(6):565–573, July 2013. 22

[CM79] A.J. Chorin and J.E. Marsden. *A Mathematical Introduction to Fluid Mechanics*. A Mathematical Introduction to Fluid Mechanics. Springer-Verlag, 1979. 25

[DBB11] R. Diziol, J. Bender, and D. Bayer. Robust real-time deformation of incompressible surface meshes. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '11, page 237–246, New York, NY, USA, 2011. Association for Computing Machinery. 26

[dWv09] Erwin de Groot, Brian Wyvill, and Huub van de Wetering. Locally restricted blending of blobtrees. *Computers and Graphics*, 33(6):690 – 697, 2009. 21, 22

[ELC18] Marvin Eisenberger, Zorah Lähner, and Daniel Cremers. Divergence-free shape interpolation and correspondence. *CoRR*, abs/1806.10417, 2018. 25, 72, 77

[EMF02] Douglas Enright, Stephen Marschner, and Ronald Fedkiw. Animation and rendering of complex water surfaces. *ACM Trans. Graph.*, 21(3):736–744, July 2002. 25, 56

[FH05] Charbel Fares and Ar Hamam. Collision detection for rigid bodies: A state of the art review. In *GraphiCon*, 2005. 7

[FSHZ19] Alvaro Javier Fuentes Suárez, Evelyne Hubert, and Cédric Zanni. Anisotropic convolution surfaces. *Computers and Graphics*, 82:106–116, 2019. 22

[GBC+13] Olivier Gourmel, Loic Barthe, Marie-Paule Cani, Brian Wyvill, Adrien Bernhardt, Mathias Paulin, and Herbert Grasberger. A gradient-based implicit blend. *ACM Trans. Graph.*, 32(2), 2013. 22, 76

[GRLM03] Naga K. Govindaraju, Stephane Redon, Ming C. Lin, and Dinesh Manocha. Cullide: Interactive collision detection between complex models in large environments using graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, HWWS '03, page 25–32, Goslar, DEU, 2003. Eurographics Association. 6, 7

[HH85] Christoph M. Hoffmann and John E. Hopcroft. Automatic surface generation in computer aided design. Technical report, USA, 1985. 20

[HL03] P. C. Hsu and C. Lee. Field functions for blending range controls on soft objects. *Proc. of Eurographics, Computer Graphics Forum*, 22(3):233–242, 2003. 21

[Hsu18] P.-C. Hsu. K-ary implicit blends with increasing or decreasing blend ranges for level blend surfaces. *Journal of Advances in Information Technology*, 2018. 21

[HVS+09] David Harmon, Etienne Vouga, Breannan Smith, Rasmus Tamstorf, and Eitan Grinspun. Asynchronous contact mechanics. New York, NY, USA, 2009. ACM. 11, 12

[HVTG08] David Harmon, Etienne Vouga, Rasmus Tamstorf, and Eitan Grinspun. Robust treatment of simultaneous collisions. *ACM Trans. Graph., Proc. ACM SIGGRAPH*, 27(3), 2008. 10

[ISF07] Geoffrey Irving, Craig Schroeder, and Ronald Fedkiw. Volume conserving finite element simulations of deformable models. *ACM Trans. Graph.*, 26(3):13–es, July 2007. 25

[Isk02] Armin Iske. Scattered data modelling using radial basis functions. *Tutorials on Multiresolution in Geometric Modelling. Mathematics and Visualization*, 2002. 32

[KHM+98] J. T. Klosowski, M. Held, J. S. B. Mitchell, H. Sowizral, and K. Zikan. Efficient collision detection using bounding volume hierarchies of k-dops. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):21–36, 1998. 6

[Kim10] Byungmoon Kim. Multi-phase fluid simulations using regional level sets. In *ACM SIGGRAPH Asia 2010 Papers*, SIGGRAPH ASIA '10, New York, NY, USA, 2010. Association for Computing Machinery. 22

[Kot91] P. R. Kotiuga. Clebsch potentials and the visualization of three-dimensional solenoidal vector fields. *IEEE Transactions on Magnetics*, 27(5):3986–3989, 1991. 25

[LDN+18] Jie Li, Gilles Daviet, Rahul Narain, Florence Bertails-Descoubes, Matthew Overby, George Brown, and Laurence Boissieux. An Implicit Frictional Contact Solver for Adaptive Cloth Simulation. *ACM Transactions on Graphics*, 37(4):1–15, August 2018. 12

[LMO03] A. Lew, J. Marsden, and M. Ortiz. Asynchronous variational integrators. *Arch. Rational Mech. Anal.*, 167:85–146, 2003. 11

[LTT+20] Cheng Li, Min Tang, Ruofeng Tong, Ming Cai, Jieyi Zhao, and Dinesh Manocha. P-cloth: Interactive complex cloth simulation on multi-gpu systems using dynamic matrix assembly and pipelined implicit integrators, 2020. 10, 55, 70, 74

[MASS15] Nathan Mitchell, Mridul Aanjaneya, Rajsekhar Setaluri, and Eftychios Sifakis. Non-manifold level sets: A multivalued implicit surface representation with applications to self-collision processing. *ACM Trans. Graph.*, 34(6), October 2015. 54, 77

[MCKM15] Matthias Müller, Nuttapong Chentanez, Tae-Yong Kim, and Miles Macklin. Air meshes for robust collision handling. *ACM Trans. Graph.*, 34(4), 2015. 11

[MGV11] Ives Macedo, Joao Paulo Gois, and Luiz Velho. Hermite radial basis functions implicits. *Comput. Graph. Forum*, 30:27–42, 2011. 32

[MHHR07] Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. Position based dynamics. *J. Vis. Comun. Image Represent.*, 18(2):109–118, April 2007. 11, 56

[MHTG05] Matthias Müller, Bruno Heidelberger, Matthias Teschner, and Markus Gross. Meshless deformations based on shape matching. *ACM Trans. Graph.*, 24(3):471–478, July 2005. 72

[MSW+09] Aleka McAdams, Andrew Selle, Kelly Ward, Eftychios Sifakis, and Joseph Teran. Detail preserving continuum simulation of straight hair. *ACM Trans. Graph.*, 28(3), July 2009. 26, 27, 57, 58, 75, 76

[NGCL09] Rahul Narain, Abhinav Golas, Sean Curtis, and Ming C. Lin. Aggregate dynamics for dense crowd simulation. In *ACM SIGGRAPH Asia 2009 Papers*, SIGGRAPH Asia '09, New York, NY, USA, 2009. Association for Computing Machinery. 26, 27, 57, 75

[NNC+20] R. Nakanishi, F. Nascimento, R. Campos, P. Pagliosa, and A. Paiva. Rbf liquids: An adaptive pic solver using rbf-fd. 2020. 25

[NSO12] Rahul Narain, Armin Samii, and James F. O'Brien. Adaptive anisotropic remeshing for cloth simulation. *ACM Trans. Graph.*, 31(6), November 2012. 12

[NZWC20] Xingyu Ni, Bo Zhu, Bin Wang, and Baoquan Chen. A level-set method for magnetic substance simulation. *ACM Trans. Graph.*, 39(4), July 2020. 22

[OTSG09] Miguel A. Otaduy, Rasmus Tamstorf, Denis Steinemann, and Markus Gross. Implicit contact handling for deformable objects. *Computer Graphics Forum*, 28(2):559–568, 2009. 12

[PASS95a] A. Pasko, V. Adzhiev, A. Sourin, and V. Savchenko. Function representation in geometric modeling: concepts, implementation and applications. *The Visual Computer*, 11(8):429–446, 1995. 20, 21

[PASS95b] Alexander Pasko, Valery Adzhiev, Alexei Sourin, and Vladimir Savchenko. Function representation in geometric modeling: concepts, implementation and applications. *The Visual Computer*, 11:429–446, 08 1995. 22

[Pro97] Xavier Provot. Collision and self-collision handling in cloth model dedicated to design garments. In Daniel Thalmann and Michiel van de Panne, editors, *Proceedings of the Eurographics Workshop on Computer Animation and Simulation 1997, Budapest, Hungary, September 2-3, 1997*, Eurographics, pages 177–189. Springer, 1997. 9

[Ric73] A. Ricci. Constructive geometry for computer graphics. *Computer journal*, 16(2), 1973. 19

[Roc89] A. P. Rockwood. The displacement method for implicit blending surfaces in solid models. *ACM Trans. Graph.*, 8(4):279–297, October 1989. 22

[RPC+10] Damien Rohmer, Tiberiu Popa, Marie-Paule Cani, Stefanie Hahmann, and Sheffer Alla. Animation Wrinkling: Augmenting Coarse Cloth Simulations with Realistic-Looking Wrinkles. *ACM Transactions on Graphics, Proc. SIGGRAPH Asia*, 29(5):157, 2010. 51

[SA07] Olga Sorkine and Marc Alexa. As-rigid-as-possible surface modeling. In *Proc. Symposium on Geometry Processing*, 2007. 46

[SH04] J. E. Solem and A. Heyden. Reconstructing open surfaces from unorganized data points. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, volume 2, pages II–II, June 2004. 33

[SMT08] Eftychios Sifakis, Sebastian Marino, and Joseph Teran. Globally coupled collision handling using volume preserving impulses. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '08, page 147–153, Goslar, DEU, 2008. Eurographics Association. 10, 11

[SS03] J. A. Sethian and Peter Smereka. Level set methods for fluid interfaces. *Annual Review of Fluid Mechanics*, 35(1):341–372, 2003. 22

[SSIF09] Andrew Selle, Jonathan Su, Geoffrey Irving, and Ronald Fedkiw. Robust high-resolution cloth using parallelism, history-based collisions, and accurate friction. *IEEE TVCG*, 15(2), 2009. 10

[Sta99] Jos Stam. Stable fluids. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '99, page 121–128, USA, 1999. ACM Press/Addison-Wesley Publishing Co. 25, 56

[TCYM08] Min Tang, Sean Curtis, Sung-Eui Yoon, and Dinesh Manocha. Interactive continuous collision detection between deformable models using connectivity-based culling. In *SPM '08: Proceedings of the 2008 ACM symposium on Solid and physical modeling*, pages 25–36, New York, NY, USA, 2008. ACM. 6

[TWL+18] Min Tang, Tongtong Wang, Zhongyuan Liu, Ruofeng Tong, and Dinesh Manocha. I-Cloth: Incremental collision handling for GPU-based interactive cloth simulation. *ACM Trans. Graph. Proc. ACM SIGGRAPH Asia*, 37(6), 2018. 10, 55

[TWT+16] Min Tang, Huamin Wang, Le Tang, Ruofeng Tong, and Dinesh Manocha. CAMA: Contact-aware matrix assembly with unified collision handling for GPU-based cloth simulation. *Computer Graphics Forum, Proc. Eurographics*, 35(2), 2016. 10

[VBG+13] Rodolphe Vaillant, Loïc Barthe, Gaël Guennebaud, Marie-Paule Cani, Damien Rohmer, Brian Wyvill, Olivier Gourmel, and Mathias Paulin. Implicit skinning: Real-time skin deformation with contact modeling. *ACM Trans. Graph.*, 32(4), 2013. 32

[VCMT95] Pascal Volino, Martin Courchesne, and Nadia Magnenat Thalmann. Versatile and efficient techniques for simulating cloth and other deformable objects. In *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, pages 137–144, New York, NY, USA, 1995. ACM. 6, 7

[vFTS06] Wolfram von Funck, Holger Theisel, and Hans-Peter Seidel. Vector field based shape deformations. *ACM Trans. Graph.*, 25(3):1118–1125, July 2006. 25, 26

[VGB+14] Rodolphe Vaillant, Gäel Guennebaud, Loïc Barthe, Brian Wyvill, and Marie-Paule Cani. Robust iso-surface tracking for interactive character skinning. *ACM Trans. Graph.*, 33(6), 2014. 21, 22, 31, 75

[VMT06] Pascal Volino and Nadia Magnenat-Thalmann. Resolving surface collisions through intersection contour minimization. In *ACM Trans. Graph., Proc. ACM SIGGRAPH*, 2006. 15, 16

[WC14] Sai Keung Wong and Yu-Chun Cheng. Gpu-based radial view-based culling for continuous self-collision detection of deformable surfaces. *The Visual Computer*, 32, 12 2014. 6, 7

[Wen05] Holger Wendland. *Scattered Data Approximation, Cambridge University Press*. 2005. 30

[Wes95] P. Wesseling. Introduction to multigrid methods. Technical report, 1995. 72

[WLG] Martin Wicke, Hermes Lanker, and Markus Gross. Untangling cloth with boundaries. 14, 15

[WLH+13] Sai-Keung Wong, Wen-Chieh Lin, Chun-Hung Hung, Yi-Jheng Huang, and Shing-Yeu Lii. Radial view based culling for continuous self-collision detection of skeletal models. *ACM Trans. Graph.*, 32(4), July 2013. 6

[WLW+14] Sai-Keung Wong, Wen-Chieh Lin, Yu-Shuen Wang, Chun-Hung Hung, and Yi-Jheng Huang. Dynamic radial view based culling for continuous self-collision detection. In *Proceedings of the 18th Meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '14, page 39–46, New York, NY, USA, 2014. Association for Computing Machinery. 6

[YMJ+17] Juntao Ye, Guanghui Ma, Liguo Jiang, Lan Chen, Jituo Li, Gang Xiong, Xiaopeng Zhang, and Min Tang. A unified cloth untangling framework through discrete collision detection. *Computer Graphics Forum*, 36(7), 2017. 9, 16

[ZBQC13] Cédric Zanni, Adrien Bernhardt, Maxime Quiblier, and Marie-Paule Cani. SCALe-invariant Integral Surfaces. *Computer Graphics Forum*, 2013. 21, 22, 76

[ZGC15] C. Zanni, M. Gleicher, and M.-P. Cani. N-ary implicit blends with topology control. *Comput. Graph.*, 46, 2015. 22

# Annexe : Résumé en Français / French summary

Gérer de manière robuste et efficace les collisions entre des surfaces 3D déformables en animation reste un challenge, d'autant plus lorsque ces surfaces sont potentiellement en contact sur de larges zones, comme c'est le cas pour les vêtements. Par conséquent, les avatars virtuels ne portent souvent qu'une seule couche de vêtements. En plus de limiter les possibilités créatives, ceci nuit à la modélisation de phénomènes physiques comme la friction entre les couches, et nuit donc aussi au réalisme de la scène.

Le principal challenge existant lors de la coexistence de plusieurs couches de vêtements en animation est la gestion (détection et correction) des collisions entre elles. Les méthodes classiques usuelles reposent sur les maillages de chaque vêtement, en testant différentes paires de triangles pour une éventuelle intersection géométrique.
Une fois les primitives en intersections identifiées, ce qui représente déjà un travail conséquent auquel s'est attelé tout un pan de la recherche„ il reste à résoudre la collision, on identifie alors deux grands type de solutions : prévenir l'intersection, ou la résoudre une fois qu'elle est apparu. Les premières méthodes se servent de l'histoire de la simulation pour déduire certaines informations (comme, par exemple, les positions relatives préalables des objets) pour ensuite parvenir à corriger la simulation actuelle et empêcher la collision. Ces méthodes sont inutilisables dans le cas d'une simulation qui commencerait avec des intersections.
La seconde famille de méthode contourne ce problème en n'utilisant pas d'informations antérieures à la situation de collision, et se contentent de critères géométriques et d'heuristiques adaptées pour la résoudre. Cependant, ces méthodes reposent souvent sur l'utilisation de forces et/ou de procédé itératif, et souffrent des inconvénients classiques liés à leur usage : des paramètres additionnels qui influent sur le résultat obtenu, ainsi qu'une convergence vers un résultat stable pas toujours garanti.
Dans ce manuscrit, nous proposons deux méthodes alternatives aux approches classiques, se reposant cette fois-ci sur l'usage de champ tridimensionnel.

En particuliers, notre première méthode fait usage de champs scalaires. Ceux-ci sont connus dans le milieu des modèles 3D dans lequel ils sont utilisés pour modéliser des surfaces implicites. Ces surfaces peuvent ensuite être combinées aisément, pour -par exemple- résoudre des situations de collisions et modéliser un contact exact entre deux objets. Les méthodes existantes se focalisent cependant sur des objets volumiques, tandis que les vêtements sont des surfaces fines. De plus, les méthodes de combinaisons de

surface sont pour la plupart limitées à deux surfaces, tandis que nous nous intéresserons à un nombre quelconque de couches de vêtements.
Notre méthode, intitulé “Implicit untangling”, ou démêlage implicite, se décompose en trois étapes :

- Une reconstitution implicite de chaque couche de vêtements, à l'aide de champs de type HRBF;
- La combinaison des surfaces implicites ainsi obtenues, qui fournit de nouvelles surfaces implicites modélisant le contact aux endroits ou d'éventuelles intersections sont détectés. Cette combinaison s'effectue grâce à des opérateurs N-aires que nous avons inventé pour résoudre ce problème;
- La projection des maillages sur leur surfaces implicites corrigées respectives, ce qui corrige effectivement les intersections entre vêtements.

Bien que les temps de calculs évoluent de manière cubique avec le nombre de couches de vêtements, ceux-ci évoluent aussi en complexité linéaire avec le nombre de triangles des maillages étudiés (contre une complexité quadratique pour des algorithmes classiques sans structures d'accélération). Cette méthode permet par ailleurs de modéliser le contact exact entre un nombre arbitraire de vêtements portés. De plus, les intersections sont résolues d'une traite, sans avoir à itérer sur des paires d'objets comme le feraient des méthodes itératives. Cependant, les temps de calcul élevés de la première étape empêche d'appliquer cet algorithme à chaque pas d'une simulation pour effectivement corriger les intersections d'une animation entière en temps réel. Nous pouvons tout de même obtenir une un état initial sans collision, dans l'optique d'appliquer des méthodes de prévention d'intersections.

Notre deuxième méthode se range dans cette catégorie, et repose sur l'usage de champ vectoriel à divergence nulle. Représentant mathématiquement l'advection d'une matière incompressible, ceux-ci ont naturellement essentiellement été utilisés dans le milieu de la simulation de fluide, mais aussi en modélisation, ou plus particulièrement en simulation de foule, pour empêcher la collision de ses membres. Nous nous inspirons de ce type de méthode et les adaptons à l'animation de vêtements : nous discrétisons l'espace dans lequel se trouvent les vêtements, et y créons un champ vectoriel représentant la vitesse des différents objets, pour enfin y résoudre un système linéaire contraignant la divergence de ce champ de vitesse. Bien que limitée par la résolution de la grille discrète utilisée, notre méthode possède une complexité empiriquement linéaire en terme du nombre de triangles des vêtements, et empêche effectivement la collision des vêtements durant leur simulation.

ECOLE DOCTORALE

**Titre :** Approches basées champ pour l'animation de couches de vêtements sans collision

**Mots clés :** Gestion de collision, Animation de vêtments, Surfaces Implicites, Champ scalaire, Champ de vecteur

**Résumé :** Gérer de manière robuste et efficace les collisions entre des surfaces 3D déformables en animation reste un challenge, d'autant plus lorsque ces surfaces sont potentiellement en contact sur de larges zones, comme c'est le cas pour les vêtements. Par conséquent, les avatars virtuels ne portent souvent qu'une seule couche, ce qui nuit à la modélisation de phénomènes comme la friction entre les vêtements, et limite aussi les possibilités créatives. Dans ce manuscrit, nous proposons deux approches alternatives à la détection et la gestion de collisions classique, basé maillage, en nous basant sur l'utilisation de champ volumiques.
Plus précisément, nous présentons deux méthodes, la première utilisant une représentation par surface implicite des couches de vêtements pour gérer d'éventuelles intersections statiques, et la seconde plongeant les vêtements dans un champ de vecteur modélisant leur vitesse.
Premièrement, nous présentons une méthode de démêlement statique de vêtement. Cette méthode se base sur une réprésentation intermédiaire des vêtements en tant que surfaces implicites -iso-niveau d'un champ scalaire- ouvertes. Pour N couches de vêtements, les N surfaces implicites associées sont combinés à l'aide d'opérateur N-aire que nous avons créé pour ce problème. Nous obtenons N nouvelles surfaces, telles que les intersections entre les surfaces initiales ont été remplacés par des zones de contact.
Deuxièmement, nous proposons une méthode utilisant un champ de vecteur pour l'animation sans collisions d'un nombre quelconque de couches de vêtements. A chaque pas de temps et après simulation, la vitesse des vêtements est convertie en un champ de vecteurs discret, dont nous contraignons localement la divergence pour éviter les collisions entre des surfaces advectés par ce champ.

**Title :** Field-based approaches for the collision-free animation of layered and dynamic clothing

**Keywords :** Collision handling, Cloth animation, Implicit surfaces, Scalar field, Vector field

**Abstract :** While real world garments are commonly worn as several layers on top of each other, 3D virtual garment animation in current entertainment production are often simplifying this modeling in considering only a single cloth layer on top of the virtual mannequin. Such simplification may limit the creative possibilities, as well as failing to model key physical aspects related to friction between cloth layers. Indeed, robustly and efficiently handling collision avoidance between 3D surface layers in contact, possibly exhibiting complex shapes and dynamic behaviors, remains a challenge in Computer Graphics.
This Phd proposes an alternative approach to mesh-based collision handling. More precisely, we propose two contributions using respectively, an implicit surface representation for static garment layer untangling, and a vector field embedding for the animation of dynamic layers.
We first present a static method for untangling layers of garments that do not require the knowledge of a previous collision-free state. Our method relies on an intermediate representation of garments as open implicit surfaces embedded in a scalar field. The implicit surfaces associated with each layer are combined using a new family of N-ary composition operators, specially designed for untangling layers. The N-resulting implicit surfaces correspond to untangled surfaces where collision is replaced by contact.
Secondly, we propose a vector-field-based approach that allows collision free garment animation in interleaving fast simulation steps with volume based velocity optimization. The simulation step is used to compute a per-vertex speed without consideration for collision. These speeds are converted as a grid-based velocity field which is then optimized to locally enforce constraints on its divergence , thus avoiding collision between layers as well as self-collision. This is done while preserving as much as possible their simulated speed in the other regions. The optimization is based on an efficient least-square representation and leads to a velocity field that can robustly handle a large number of animated garments, as well as allowing a unified framework for the animation of collision-free dynamic objects.

**Institut Polytechnique de Paris**
91120 Palaiseau, France