Interactive Detailed Cutting of Thin Sheets

Pierre-Luc Manteaux, Wei-Lun Sun, François Faure, Marie-Paule Cani, James F. O'Brien

Interactive Detailed Cutting of Thin Sheets

Pierre-Luc Manteaux[1] Wei-Lun Sun[2] François Faure[1] Marie-Paule Cani[1] James F. O'Brien[2]

[1]University Grenoble-Alpes, CNRS (LJK) and Inria * [2]University of California, Berkeley †

Abstract

In this paper we propose a method for the interactive detailed cutting of deformable thin sheets. Our method builds on the ability of frame-based simulation to solve for dynamics using very few control frames while embedding highly detailed geometry - here an adaptive mesh that accurately represents the cut boundaries. Our solution relies on a non-manifold grid to compute shape functions that faithfully adapt to the topological changes occurring while cutting. New frames are dynamically inserted to describe new regions. We provide incremental mechanisms for updating simulation data, enabling us to achieve interactive rates. We illustrate our method with examples inspired by the traditional Kirigami artform.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation;

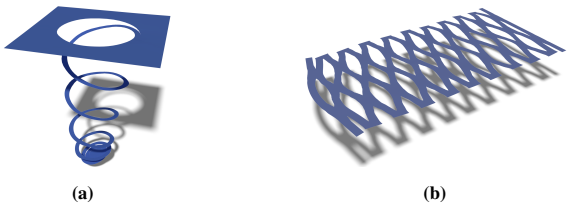
Keywords: physics-based animation, cutting, thin sheets, interactive

1 Introduction

Over the last three decades, physics-based animation methods have been proposed to simulate a wide range of phenomena. Substantial progress has been achieved in terms of efficiency and realism. As a result, physics-based animation has found applications in film, games, craft, teaching, and training.

Combining interactive user actions and detailed convincing animations is crucial for user experience in simulation and games. Unfortunately, computational contraints limit the fidelity that can be achieved with physics-based animation in interactive simulations. Often, the simulated objects lack detail compared to the rest of the virtual environment. Furthermore, operations that modify the structure of the simulated objects, such as cutting, maybe incompatible with faster simulation methods. When not prohibited, the latter generally exhibit strong limitations. Indeed, the level of sampling of a physically-based model usually depends on geometric complexity. Detailed cuts result in an increase of the sampling which directly impacts the performance. In practice, the number of samples is limited to ensure real-time performance. This limitation quickly prevents the user from applying detailed cuts.

In this work, we address the issue of enabling detailed cuts at interactive rates in thin sheets of deformable materials. Our method is able to capture detailed cuts while using a relatively low number of control nodes for the physically-based model. Our approach to decoupling the samplings of the physical geometric models, is to use a mesh-less simulation method called the frame-based model [Gilles et al. 2011]. In this method, the deformation field induced by animated frames is applied to the geometric model using skinning weights. As each frame can cover a large, detailed shaped region of the geometric mesh, only a few of them are typically required.



(a) (b)

Figure 1: *Progressive cutting of a spiral using only five control frames (a). Simulating complex deformations resulting from Kirigami cutting (b).*

To achieve user-driven cuts in a frame-based simulation, we allow cuts to be performed on the underlying mesh. We build a non-manifold grid that keeps track of the mesh topology at the simulation level, and allows us to incrementally adapt the frames regions of influence in order to represent the cut. Although remaining low, the number of frame node does increase during a cut. In particular, when a model is cut apart, at least one frame is needed to represent each disconnected component. Therefore, we detect crucial cutting events, enabling us to automatically insert new frames when and where they are needed. In order to reduce computations, we exploit the locality of the ongoing cutting gesture to incrementally update all the data used for the simulation.

Our contributions include (1) the building of a non-manifold grid to compute shape functions that faithfully represent the complex topology of the visual mesh while keeping a low number of control nodes, (2) the dynamic re-sampling of new frames into disconnected parts and (3) the incremental update of the simulation data that were concerned by the cut.

Our method can be used to simulate a wide variety of objects, such as stretchable cloth or pieces of paper. It features a very low number of frame nodes, high resolution mesh embedding, numerous and detailed cuts. Performance ranges from interactive to offline depending on the desired accuracy and complexity of the cuts.

2 Related Work

Cutting and fracture are both fascinating behaviors which can be simulated separately. In fracture, stress measurements predict how the material breaks. In cutting, the interaction with a tool define the cut path. For more details about cutting we refer the reader to the recent survey of Wu et al. [Wu et al. 2015]. Our review focuses on the modeling of topological changes in deformable models.

A first possibility consists in using the same model for physics simulation and visualization. Topological changes are then mostly modeled by remeshing operations. Simple and fast remeshing techniques such as element deletion or element splitting were proposed. The latter was used in the first simulation of brittle and ductile materials [O'Brien and Hodgins 1999], [O'Brien et al. 2002]. Methods

*{ pierre-luc.manteaux|françois.faure|marie-paule.cani }@inria.fr

†{ sunweilunjwilson|job }@berkeley.edu

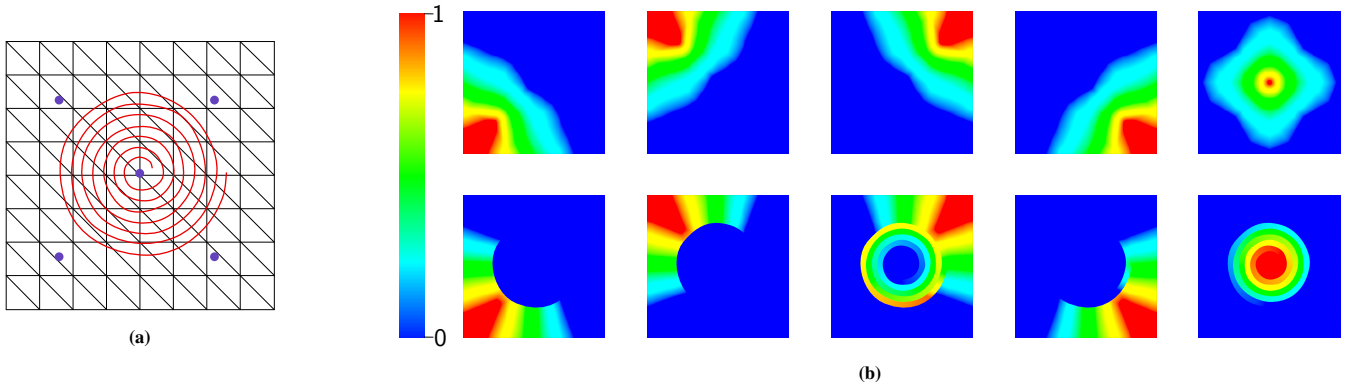


Figure 2: *Comparison between shape functions computed on a uniform grid and on a non-manifold grid. (a) The underlying mesh (black lines) is cut by spiral (red line) and sampled with five control frames (blue circle). (b) The shape functions for each of the frame with a uniform grid (top row) and with a non-manifold grid (bottom row). Values range from* 1 *to* 0 *and are respectively depicted from red to blue. We can observe that shape functions computed on the non-manifold grid strictly preserve the details and topology of the underlying mesh.*

that preserve element quality by local and global remeshing have also been developed. They recently lead to stunning results in the simulation of multi-layered paper tearing [Busaryev et al. 2013] and sheets tearing [Pfaff et al. 2014]. These methods cause the number of simulation nodes to vary over the course of a simulation, and this variation can be problematic in a realtime game context. By limiting the scope of remeshing predictable realtime performance can be achieved [Parker and O'Brien 2009]. An alternative to remeshing is to enrich elements with additional basis so that discontinuities can be represented. This is the core idea of the eXtended Finite Element Method (XFEM). It was successfully applied for offline cutting of discrete shells [Kaufmann et al. 2009].

A second possibility is to separate the visual model from the physics model, this is known as embedding. Numerous embedding techniques have been proposed. The virtual node method [Molino et al. 2004] embeds ill-shaped elements that arise after remeshing inside of well-shaped elements. This allows to robustly simulate detailed cuts [Wang et al. 2014]. However, the number of nodes increases substantially with the complexity of the cut. To reduce it, hierarchical methods were proposed and real-time cutting in medical applications has been achieved using composite finite element method [Wu et al. 2011]. Still, the number of nodes grows quickly with the number of cuts and remains limited to ensure interactive frame rate. Meshless methods avoid the problem of element quality. However, boundary and discontinuities require extra effort to be sharply represented. [Pauly et al. 2005] proposed to use visibility criterion to perform fracture. [Steinemann et al. 2009] used the visual model as a visibility graph to define nodes connectivity. Both methods rely on a dense sampling near the surface of the model and quickly impact performances as the number and the detail of cuts increases. There also have some work to carry complex materials [Nesme et al. 2009] and thin shells [Rémillard and Kry 2013] in hexahedra elements.

Embedding techniques have inspired our work. They allow interesting trade-off and show impressive cutting and fracture simulations. However, the relation between the resolution of the physical model and the visualization model remains very strong. Complex cuts result in a fast increase of the number of nodes. We want to reduce this connexion as much as possible. Complex topologies could be simulated with a very low number of nodes. Then, interactivity and intuitive control would be at hand.

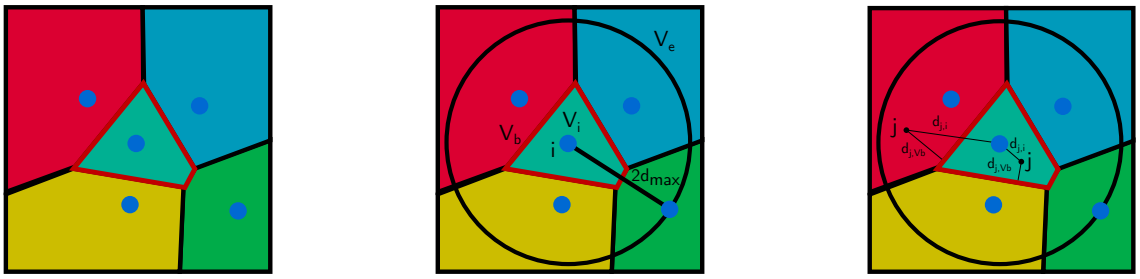
Few models have been proposed that simulate detailed deformable objects using a low number of nodes. Subspace simulations [Barbič and James 2005] compute a low basis of deformation modes in order to achieve real-time performance on detailed models. However, the low-basis is acquired after heavy precomputations. Interactive scenario could not handle the recomputation of the basis at each topological change. More recently, [Gilles et al. 2011] and [Faure et al. 2011] proposed a physics-based skinning technique, called the frame-based method. Highly detailed meshes can be embedded in very coarse simulations. The control nodes are affine frames and the deformation field is described by a linear blend skinning. Classical continuum mechanics is then used to solve for the dynamics. Skinning weights, also called shape functions, are built on linear interpolation using discrete voronoi regions. For each frame, they can represent a large region of influence with complex shape. Other advantages are discussed in [Faure et al. 2011]. Unfortunately, the current frame-based method does not allow the shape functions to reflect the topological changes of the embedded mesh.

3 Overview

The goal of this work is to enable interactive detailed cutting of deformable thin sheets. The frame-based method exhibits some of the key features we are looking for: a very low number of nodes and a tunable separation between visual and physical models. We build on this framework and extend it to handle topological changes.

To transfer the cuts from the mesh to the frames, we continuously adapt the shape functions to the evolving mesh topology. This allows us to keep a constant number of nodes as long as there are no disconnected parts. In [Faure et al. 2011], the shape functions are computed on a uniform grid. The structure is simple and efficient. However, discontinuities that can be represented are very limited and strongly connected to the grid resolution. Instead, we build a non-manifold grid to compute topology-preserving shape functions (see Figure 2). The main idea is that cut cells are duplicated and store different connectivities. Therefore, grid resolution depends much less on the mesh topology while keeping all topological informations.

We detail the computation of the shape functions, motivate the use of a non-manifold grid and explain how to build it (Section 4). When several parts are disconnected, we need to make sure that they contain control frames. We present a simple method to detect those regions and re-sample them (Section 5). Even with such a low resolution physics model, it would be overkilling to update the whole system at each cut. Fortunately, cutting is often a local event.



(a) (b) (c)

Figure 3: *Illustrations of Voronoi shape function computation. (a) Starting from samples (blue circles), we build a Voronoi diagram using Dijkstra's shortest path algorithm. (b) Then, for each frame and its region V_i, we compute the maximum distance d_{max} to its Voronoi boundary V_b. We extend V_i to twice d_{max} which gives V_e. (c) Finally for each grid cell j in V_e we linearly interpolate using distance to the frame position and distance to V_b.*

We leverage this fact and propose simple strategies to incrementally update the different components of the simulation (Section 6). We illustrate our method in different scenarios (Section 7) and discuss limitations and future work (Section 8). We summarize our simulation loop in Algorithm 1 and detail our remeshing algorithm in appendix A.

Algorithm 1 Simulation loop

```
for each time step do
    perform a frame-based simulation step
    split the mesh along the cut
    embed the mesh in a non-manifold grid
    add new frames if required
    add new samples (collision, integration) if required
    compute shape functions on the grid
    incrementally update the samples
end for
```

4 Adaptive shape functions

In this section, we first summarize how Voronoi shape functions are traditionally computed. Then we detail why a non-manifold grid is necessary, how to build it and how to use it to compute the shape functions on complex topology.

4.1 Voronoi shape function

Let $w_i(x) : \Omega \rightarrow \mathcal{R}$ be the shape function for the i-th control frame, where Ω represents the domain. Starting from the Voronoi partition V of the set of control frames, we can independently compute w_i for each frame.

First, we compute the maximal distance d_{max} from the control node to its Voronoi boundary V_b. Then we extend its Voronoi region V_i to twice d_{max}. This gives a new region V_e which describes the final boundary of the shape function. Now, we can compute w_i inside V_e. We set w_i to be 1 at the frame position, 0 at the others and 0.5 on V_b. Finally, we linearly interpolate w_i between V_b, the frame position and the boundary of V_e. We detail the interpolation in Algorithm 2 and in Figure 3.

In practice, Voronoi diagram is computed using Dijkstra's shortest path algorithm on a grid in order to preserve geodesic distances. For each frame, the shape function is computed on the whole grid. As the grid resolution can be quite coarse, this is particularly fast. Negative values are clamped and weights are normalized to form a partition of unity. Then least-square approximation is performed to evaluate the shape function and its derivatives at specific position.

Algorithm 2 Shapefunction computation

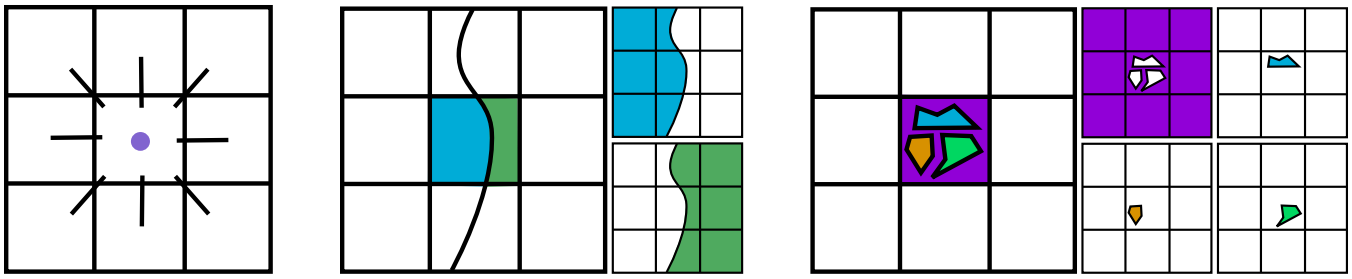
```
1:  procedure COMPUTE_SHAPEFUNCTION
2:      for each frame i do
3:          V_i ← Voronoi region of i
4:          V_b ← boundary of V_i
5:          d_max ← maximum distance to V_i boundary
6:          V_e ← extend V_i to 2.0 × d_max
7:          ▷ dist(A,B) is the geodesic distance between A and B
8:          for each grid cell j in V_e do
9:              if j is inside V_i then
10:                 w_i(j) = 0.5 ( 1 + dist(j,V_b) / (dist(j,V_b) + dist(j,i)) )
11:             else if j is inside V_e then
12:                 w_i(j) = 0.5 ( 1 − dist(j,V_b) / (dist(j,i) − dist(j,V_b)) )
13:             end if
14:         end for
15:     end for
16: end procedure
```

Voronoi shape functions were designed in order to respect key properties that are particularly useful for physics-based animation [Faure et al. 2011] . First, they respect the Kronecker property, i.e $w_i(x) = \delta_i(x)$ where $w_i(x)$ is the shape function of node i, x is a spatial position and δ_i is Dirac function. Second, they form a partition of unity, i.e $\sum_i w_i(x) = 1$. Third, they are built to be as linear as possible in order to produce uniform deformations. Finally, they can easily be biased by material properties in order to represent heterogeneous material.

4.2 Non-manifold grid

As mentionned above, in [Faure et al. 2011], shape functions are computed on a uniform grid using Dijktra's shortest path algorithm to compute geodesic distance. Starting from a uniform grid with a 8-neighbor connectivity, we could reflect topological change by changing the connectivity of the cut cells. Then, when we re-compute shape functions, the topology would automatically be taken into account as we use geodesic distance.

Unfortunately, this strategy is very limited for uniform grid and would only work in simple cases. For instance, several cuts that intersect or that create disconnected components inside one cell could



(a) (b) (c)

Figure 4: *Illustrations of different possibilities for a non-manifold cell with eight connectivity (a). In (b), the cell is simply cut into two cells. Each duplicate of the cut cell has a specific connectivity that represent the cut topology. In (c), multiple disconnected components can be contained inside one cell. The cell is duplicated four times. Three of the duplicates have no connectivity. However they can embed complex geometry and then be simulated by adding new frames for each of the component. The fourth duplicate keeps its eight neighbors and remains independent from the three other.*

not be represented. Even without cut, small gaps that lie inside one cell could not be correctly represented. Geodesic distances would be false and the object would behave as if there were no cuts or gaps. Augmenting the resolution would not solve the problem. We would fight the same issue as previous methods. Our grid resolution would be highly dependent on the complexity of the topology and the geometry of the object. It would directly impact performances.

We want each grid cell to be able to represent the connectivities of the different disconnected components that lie in the cell. To do so, each cut cell is duplicated as many times as it contains disconnected parts. Each duplicate has a specific connectivity built from the material connectivity. This results in a data structure called *non-manifold grid* (see Figure 4).

Non-manifold grids are used by many other cutting methods to embed fine geometric details in coarse finite element simulations. However, we make a completely different use of it. Instead of duplicating control nodes as the cells are cut, thereby increasing their number and the computation time, we use the grid to adapt the shape functions to the evolving topology of the mesh. Most of the time, the number of nodes can remain constant while representing detailed geometry and multiple cuts.

There are several ways to compute this non-manifold grid. In our method, we start by embedding the mesh in a uniform grid. Mesh elements that overlap a grid cell are detected using intersections tests and are assigned to it. Then, for each grid cell, we use a flood fill algorithm to detect the disconnected parts of the mesh. This informs about how many duplicates need to be created for the cell. Finally, for each duplicate we establish its connectivity by comparing its geometry with the geometry of the neighbor cells duplicates. We summarize our method in Algorithm 3 and illustrate the main steps in Figure 5.

Algorithm 3 Non-manifold grid building

```
1:  procedure BUILD_NON_MANIFOLD_GRID(grid G, mesh M)
2:      BUILD_GRID_GEOMETRY(G,M)
3:      DUPLICATE_GRID_CELL(G)
4:      BUILD_GRID_CONNECTIVITY(G)
5:  end procedure
6:
7:  procedure BUILD_GRID_GEOMETRY(grid G, mesh M)
8:      for each cell i of G do
9:          Store overlapping element of M
10:     end for
11: end procedure
12:
13: procedure DUPLICATE_GRID_CELL(grid G, mesh M)
14:     for each cell i of G do
15:         C ← disconnected component of M in i
16:         for each component j of C do
17:             Duplicate the cell i
18:             Store j in the duplicate
19:         end for
20:     end for
21: end procedure
22:
23: procedure BUILD_GRID_CONNECTIVITY(grid G)
24:     for each cell i of G do
25:         N ← neighbor cells of i
26:         for each duplicate j of i do
27:             for each duplicate k in N do
28:                 if j and k shares geometry then
29:                     Create a link between j and k
30:                 end if
31:             end for
32:         end for
33:     end for
34: end procedure
```

5 Frame re-sampling

As long as no parts of the model are disconnected, our method allows to keep a constant number of control frames. However, when parts are disconnected, we need to sample it with at least one frame in order to simulate it.

We start by detecting empty regions i.e lists of connected cells that are not influenced by any frame. This is done using a flood fill algorithm on the grid containing the shape functions values. These empty regions are then sampled using a farthest sampling algorithm. Finally, the samples are uniformly distributed by applying several Lloyd relaxation steps. For now, the number of frames which are sampled is user-defined but we would like to investigate for setting it automatically (see Section 8).

As a cut progresses, it may happen that only one frame influences a large region. Then this region can only express affine motion. Depending on the material properties, the size and the shape of the region, this can result in unconvincing behaviours. For rigid materials this is not a problem but for soft material this can quickly become unrealistic. We propose a simple strategy to solve some of

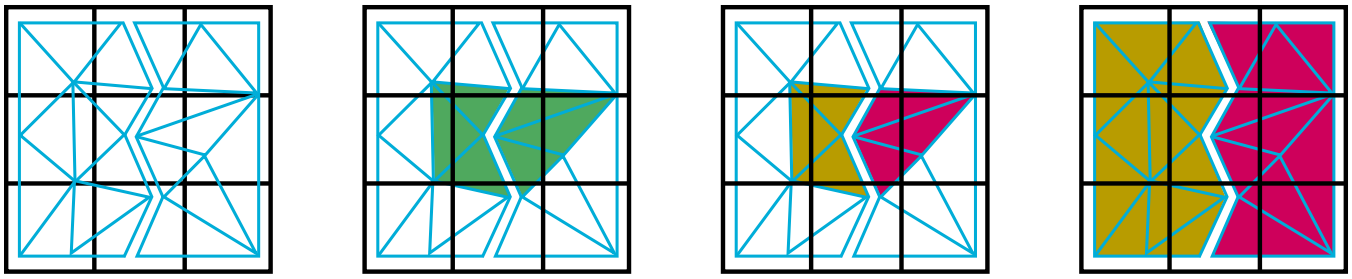


Figure 5: *We describe the building of the non-manifold grid for the center cell of the grid. (a) The mesh is embedded in a uniform grid. (b) First, we store the overlapping geometry in the cell. (c) Then we detect disconnected parts using a flood fill algorithm. (d) Finally the cell is duplicated. For each duplicate, we look for other duplicates that share geometry and establish its connectivity.*

these cases. For each frame, we look for regions where the shape function value is above a user-defined threshold w_{max}. Then if the volume of the region is above a maximal volume threshold v_{max}, we uniformly re-sample the region. This strategy allows to detect large regions which are mostly influenced by only one frame and are the most likely to need re-sampling. For now, w_{max} and v_{max} are user-defined.

As regions of influence are very large, the popping artefacts induced by adding instantaneously one additional frame can be noticeable. In order to reduce them we propose a simple strategy. Once the position of the new frame in the undeformed, material space has been chosen, we use the previous deformation field to interpolate its new position, orientation and velocity.

6 Incremental update

The domain and the shape functions continuously change during cutting. Therefore, all the simulation data that are related to the domain or the shape functions need to be updated at each time step a cut occurs. Fortunately, cutting is often a local phenomenon. We exploit this locality to incrementally update only what is necessary and therefore save substantial computational time.

In our case, there are several simulation components that need to be updated. The first of this component contains the integration points that compute deformation gradients and transfers internal forces to the control frames. Then there is the collision component, a simple set of points, that transfers external forces to the control frames. Finally, there is the mesh that we visualize whose vertices positions are interpolated from the frame positions. Each of this component can have its own resolution. Their data are computed from the control frames using interpolation. This layer-based organization allows to separate the resolutions of the physical simulation, the interactive model and the visual rendering to achieve a good trade-off between realism and performance.

In the following sections we describe the mechanisms we used to incrementally update the different components of the simulation.

6.1 Re-sampling

As for the frames, we always need to have at least one collision node and one integration point inside each part of the model. Otherwise, we cannot compute deformations or interact with these parts of the model. Usually, there are much more collision nodes and integration points than frames. Instead of adding new points only when we detect new empty regions, we perform a few Lloyd relaxation steps at each time step to always keep a uniform sampling of the domain. In a progressive cut scenario, only a small number of samples will need to be updated at each time step and will result in an efficient incremental update. However, if disconnected parts are created from a cut, we apply the re-sampling strategy discussed in Section 5. We detect the disconnected parts using a flood fill algorithm and uniformly re-sample them.

6.2 Integration point update

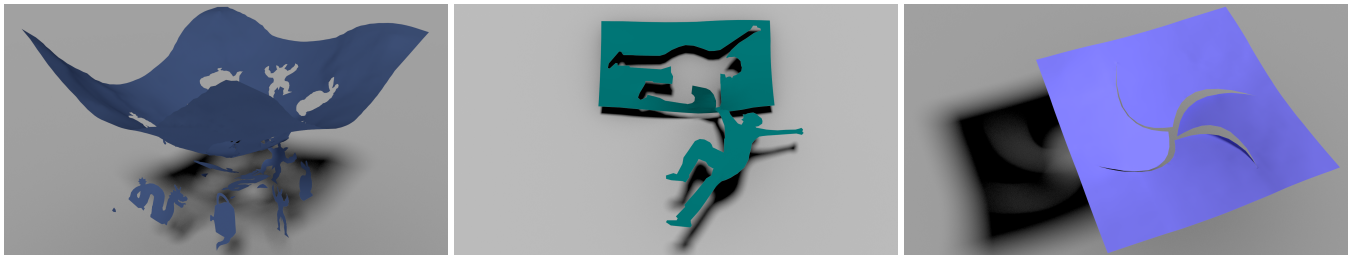
Integrations points are used to compute deformation gradients and transfer internal forces to the frames. To do so, each integration point are interpreted as a small volume of the domain and carries a position, a region's volume and the volume moments. As soon as a cut occurs, the region's volume of integration points close to the cut will change and it becomes necessary to update these integration points. This can be easily done by storing an explicit description of the region of the integration point i.e a list of cells. If the cut goes through one of these cells then we update the integration point data.

6.3 Local weights update

Weights and derivatives are interpolated from the grid to positions of the different samples : collision nodes, integration points and mesh vertices. At each cut, we need to update these values. In an interactive context, we cannot afford to perform interpolation for all these samples. Once again, we leverage the fact that a cut is very often a local event, sometimes progressive, and will impact only a small fraction of the different samples. Our idea is to perform incremental update of weights and derivatives by detecting the low number of samples that were impacted by the cut. At each time step, if a cut was performed, we compare the new shape functions with the previous ones and detect the grid cells which have been impacted by the cut. All the samples that are contained or are neighbors of these cells need to be updated. In the end, even if we have control frames that covers large regions of the domain compared to classical simulations, simulation data that need to be updated remains spatially local.

7 Results

We illustrate our method in a variety of simulations where a piece of paper undergoes progressive scripted cuts. As we use several layers of samples (frames, collision nodes, integration points), choosing a good trade off between accuracy and performance is essential. In all the examples, we used the minimum number of samples we could without compromising visual results (see Table 1). Frame re-sampling was required to simulate disconnected parts. However it appears that no additional collision nodes or integration points were



(a) (b) (c)

Figure 6: *(a) Several highly detailed shapes are cut in a deformable sheet. Each disconnected part is automatically re-sampled with additional control frames. (b) Simulation of a highly detailed cut that falls under gravity and remains attached to the main part by a thin piece of paper. (c) Two cuts intersect to form a vortex shape. This illustrates the abilities of the non-manifold grid to handle multiple intersecting cuts.*

Name	Grid Size	#frame		#vertices		#collision	#integration	Lowest FPS		
		Initial	Final	Initial	Final			Before Cutting	During Cutting	After Cutting
Spiral (Fig. 1a)	40×40	5	5	81	2111	200	200	60	14.4	60
Kirigami (Fig. 1b)	68×68	47	47	4225	7453	600	800	11.3	3.2	10.9
Patchwork (Fig. 6a)	50×50	5	12	4225	8253	200	200	60	6.8	45
Vortex (Fig. 6c)	68×68	5	5	4225	4889	200	200	45	7.2	35.7
FallingGuy (Fig. 6b)	100×50	10	10	289	861	500	500	60	8.2	60

Table 1: *Resolution of the different components of the simulation and timings.*

Name	Percentage of update for a cutting step				
	%grid cell	%shape function cell	%vertices	%collision nodes	%integration points
Spiral (Fig. 2a)	0.07	28.1	61.5	27.2	41.3
Kirigami (Fig. 1b)	1.06	15.9	17.8	15.8	20.3
Patchwork (Fig. 6a)	0.02	2.78	4.33	2.55	7.15
Vortex (Fig. 6c)	0.08	10.3	12.8	9.2	24.2
FallingGuy (Fig. 6b)	0.09	5.84	11.4	5.77	14.9

Table 2: *Percentage of updated data in a cutting time step. We averaged the percentage for the whole cutting time. We notice that even if very few grid cell are affected, it implies important changes on the shape functions and the samples that are associated to these values.*

required. We can deduce that our relaxation strategy is sufficient to keep the object uniformly sampled along the simulation.

All our examples run at interactive frame rate during the whole simulation (see Table 1). Frame rates were collected on a twelve-core 3.20 GHz Intel Xeon CPU with 15.6 GB RAM. We highlight that our implementation is not optimized and there are still a lot of room for improvements that we did not have time to implement. Full animated results are shown in the accompanying video.

Figure 1a shows a long spiral cut in a sheet of paper simulated with only 5 frames. The shape functions of the frames faithfully represent the cut as shown in Figure 2.

To illustrate that our method can handle multiple cuts and still simulate complex deformations, the creation of a Kirigami is shown in Figure 1b. 48 cuts are performed and it only required 47 frames and 400 integration points to produce a plausible behavior.

Detailed cuts can be performed and separated components can be handled as shown in Figure 6a. In a cloth sheet, we progressively cut bunny, teapot, dragon and armadillo shapes. Each time a new object is completely cut, it is automatically re-sampled with additional frames.

As we explained, the non-manifold grid can represent an arbitrary number of connectivity in one cell. This is particularly useful in order to represent intersecting cuts as shown in Figure 6c.

We noticed that even if a cut only concern a few grid cells, the number of data to re-compute is much more important. This comes from the fact that each frame can cover a large region and changes arising from a local cut can be important. Fortunately, our incremental update mechanisms allows to save numerous unnecessary computations as shown in Table 2.

8 Discussion

We presented a novel method to simulate highly detailed cuts with a sparse set of control nodes which allows interactive frame rates. This approach can be seen as a reduced simulation that handles topological changes without requiring expensive pre-computations. Of course, our work is not without limitations and and we see interesting directions for future work.

First, as very few frames are used, one cut may generate large changes in the weight distribution and produce popping artefacts that cannot be avoided using our interpolation strategy. This is particularly noticeable when simulating soft materials and can be seen in some of the examples of our accompanying video. Strategies proposed by [Narain et al. 2013] and [Tournier et al. 2014] in the context of adaptive simulations could be used to limit this problem.

Secondly, for large deformations, the surface can look bumpy. There are several reasons for this problem. Linear blend skinning

produces well known artefacts that could be solved using a better skinning approach such as dual quaternion skinning. Also, the shape functions derivatives are discontinuous and this is particularly noticeable during high deformations. One could easily change the shape functions and still use the non-manifold grid to depict the topology.

Finally, our implementation is far from being optimal. Currently the non-manifold grid and the shape functions are re-computed from scratch at each cut. We could enjoy a dynamic acceleration structure to incrementally update our non-manifold grid. Shape functions could also be incrementally updated. Finally there are several parts of our method that could enjoy parallelization such as samples interpolation.

In future, we first plan to extend our work to 3D. The implementation of our current non-manifold grid would require a tetrahedron representation of the object. We would like to investigate the method of [Rémillard and Kry 2013] to build this structure only from the object surface. We think that the frame-based framework can be used to produce interactive detailed fracture simulation. The main challenge is to accurately compute stress tensors which are then used to determine fracture direction. Instead of using a dense sampling of frames and integration points to compute the stress tensors, we would like to combine a low resolution stress tensor measurement with procedural detail generation as in the work of [Chen et al. 2014] and [Lejemble et al. 2015]. Finally, we would like to investigate advanced sampling strategies in order to automatically determine how many frames are required for a given region. This would involve the material property, the size and the shape of the region that needs to be sampled.

Acknowledgements

This work was partially funded by the advanced grant no. 291184 EXPRESSIVE from the European Research Council (ERC-2011-ADG_20110209) and the France Berkeley Fundation.

References

BARBIČ, J., AND JAMES, D. L. 2005. Real-time subspace integration for st. venant-kirchhoff deformable models. *ACM Trans. Graph. 24*, 3 (July), 982–990.

BUSARYEV, O., DEY, T. K., AND WANG, H. 2013. Adaptive fracture simulation of multi-layered thin plates. *ACM Trans. Graph. 32*, 4 (July), 52:1–52:6.

CHEN, Z., YAO, M., FENG, R., AND WANG, H. 2014. Physics-inspired adaptive fracture refinement. *ACM Trans. Graph. 33*, 4 (July), 113:1–113:7.

FAURE, F., GILLES, B., BOUSQUET, G., AND PAI, D. K. 2011. Sparse Meshless Models of Complex Deformable Solids. *ACM Transactions on Graphics 30*, 4 (July), Article No. 73.

GILLES, B., BOUSQUET, G., FAURE, F., AND PAI, D. 2011. Frame-based Elastic Models. *ACM Transactions on Graphics 30*, 2 (Apr.), Article No. 15.

KAUFMANN, P., MARTIN, S., BOTSCH, M., GRINSPUN, E., AND GROSS, M. 2009. Enrichment textures for detailed cutting of shells. *ACM Trans. Graph. 28*, 3 (July), 50:1–50:10.

LEJEMBLE, T., FONDEVILLA, A., DURIN, N., BLANC-BEYNE, T., SHRECK, C., MANTEAUX, P.-L., KRY, P. G., AND CANI, M.-P. 2015. Interactive procedural simulation of paper tearing with sound. In *Proceedings of the Eigth International Conference on Motion in Games*, ACM, New York, NY, USA, MIG '15.

MOLINO, N., BAO, Z., AND FEDKIW, R. 2004. A virtual node algorithm for changing mesh topology during simulation. *ACM Trans. Graph. 23*, 3 (Aug.), 385–392.

NARAIN, R., PFAFF, T., AND O'BRIEN, J. F. 2013. Folding and crumpling adaptive sheets. *ACM Transactions on Graphics 32*, 4 (July), 51:1–8. Proceedings of ACM SIGGRAPH 2013, Anaheim.

NESME, M., KRY, P., JERABKOVA, L., AND FAURE, F. 2009. Preserving Topology and Elasticity for Embedded Deformable Models. *ACM Transactions on Graphics 28*, 3 (Aug.), Article No. 52.

O'BRIEN, J. F., AND HODGINS, J. K. 1999. Graphical modeling and animation of brittle fracture. In *Proceedings of ACM SIGGRAPH 1999*, ACM Press/Addison-Wesley Publishing Co., 137–146.

O'BRIEN, J. F., BARGTEIL, A. W., AND HODGINS, J. K. 2002. Graphical modeling and animation of ductile fracture. In *Proceedings of ACM SIGGRAPH 2002*, ACM Press, 291–294.

PARKER, E. G., AND O'BRIEN, J. F. 2009. Real-time deformation and fracture in a game environment. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 156–166.

PAULY, M., KEISER, R., ADAMS, B., DUTRÉ, P., GROSS, M., AND GUIBAS, L. J. 2005. Meshless animation of fracturing solids. *ACM Trans. Graph. 24*, 3 (July), 957–964.

PFAFF, T., NARAIN, R., DE JOYA, J. M., AND O'BRIEN, J. F. 2014. Adaptive tearing and cracking of thin sheets. *ACM Transactions on Graphics 33*, 4 (July), xx:1–9. To be presented at SIGGRAPH 2014, Vancouver.

RÉMILLARD, O., AND KRY, P. G. 2013. Embedded thin shells for wrinkle simulation. *ACM Trans. Graph. 32*, 4 (July), 50:1–50:8.

STEINEMANN, D., OTADUY, M. A., AND GROSS, M. 2009. Splitting meshless deforming objects with explicit surface tracking. *Graph. Models 71*, 6 (Nov.), 209–220.

TOURNIER, M., NESME, M., FAURE, F., AND GILLES, B. 2014. Velocity-based Adaptivity of Deformable Models. *Computers and Graphics 45* (Dec.), 75 – 85.

WANG, Y., JIANG, C., SCHROEDER, C., AND TERAN, J. 2014. An Adaptive Virtual Node Algorithm with Robust Mesh Cutting. In *Eurographics/ ACM SIGGRAPH Symposium on Computer Animation*, The Eurographics Association, V. Koltun and E. Sifakis, Eds.

WU, J., DICK, C., AND WESTERMANN, R. 2011. Interactive high-resolution boundary surfaces for deformable bodies with changing topology. In *Proceedings of 8th Workshop on Virtual Reality Interaction and Physical Simulation (VRIPHYS) 2011*, 29–38.

WU, J., WESTERMANN, R., AND DICK, C. 2015. A survey of physically based simulation of cuts in deformable bodies. *Computer Graphics Forum*.

A Remeshing

As stated previously, the mesh is only used for visualization. Simulation robustness will not be determined by its elements' quality.

Therefore we used an extremely simple remeshing algorithm. The input are the mesh and a polyline that represents the cut. We start by remeshing along the polyline so that the mesh conforms with it. Then we duplicate the mesh vertices along this polyline to create the crack. The whole procedure is summarized in algorithm 4 and illustrated in Figure 7. The remeshing part uses vertex insertion and edge split operations (see Figure 8). The splitting part only uses vertex split operation (see Figure 9).

Algorithm 4 Remeshing Algorithm

procedure CUT_ALONG_SEGMENT(Segment S, mesh M)
  INSERT_SEGMENT(S, M)
  $\tilde{P} \leftarrow$ edges corresponding to S
  SPLIT_ALONG_POLYLINE($\tilde{P}$,M)
end procedure

procedure INSERT_SEGMENT(Segment S, Mesh M)
  $\tilde{S} \leftarrow$ subdivide S at intersection with M edges
  for each point i of $\tilde{S}$ **do**
    $E \leftarrow$ closest edge to i
    $V \leftarrow$ closest vertex to i
    $F \leftarrow$ closest triangle to i
    if distance(E,i)$< \epsilon_{edge}$ **then**
      Split E at i
    else if distance(V,i)$< \epsilon_{vertex}$ **then**
      Snap i to V
    else
      Split F at i
    end if
  end for
end procedure

procedure SPLIT_ALONG_POLYLINE(Polyline P, Mesh M)
  for each vertex V of P **do**
    Split triangles arround V according to P
  end for
end procedure

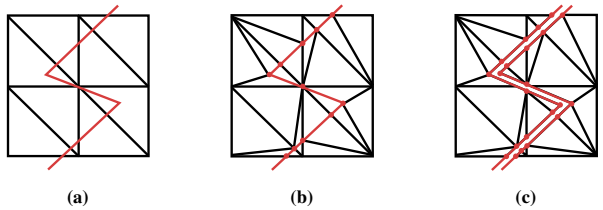


Figure 7: *Illustration of our remeshing algorithm. (a) For remeshing, we start from an input mesh and a polyline that represents the cut. (b) First we re-mesh along the polyline so that the mesh is conform with the cut. (c) Then we split the mesh vertices along the polyline.*

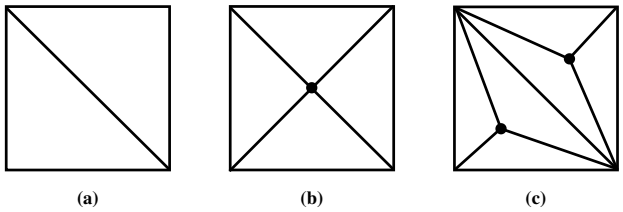


Figure 8: *Illustrations for edge splitting and vertex insertions. (a) The input mesh. (b) After edge splitting. (c) After vertex insertions.*

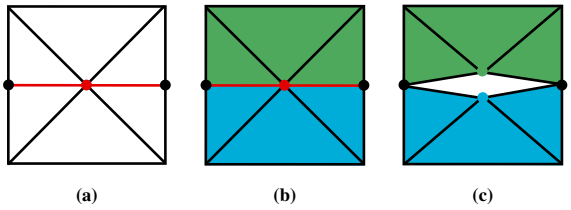


Figure 9: *Illustrations for the vertex splitting operation. (a) A mesh which is conform with the polyline (in red). (b) We start by assigning each triangle arround the vertex to split to one side of the polyline. (c) We duplicate the vertex and modify each of the triangles accordingly to its side.*