International Conference on Artificial Reality and Telexistence
Eurographics Symposium on Virtual Environments (2022)
J.-M. Normand and H. Uchiyama (Editors)

Progressive tearing and cutting of soft-bodies in high-performance virtual reality

M. Kamarianakis[1,2,4] A. Protopsaltis[3,4] D. Angelis[1,2,4] M. Tamiolakis[2,4] G. Papagiannakis[1,2,4]

[1] Foundation of Technology Hellas - ICS, Heraklion, Greece
[2] University of Crete, Dept. of Computer Science, Heraklion, Greece
[3] University of Western Macedonia, Kozani, Greece
[4] ORamaVR SA, Heraklion, Greece

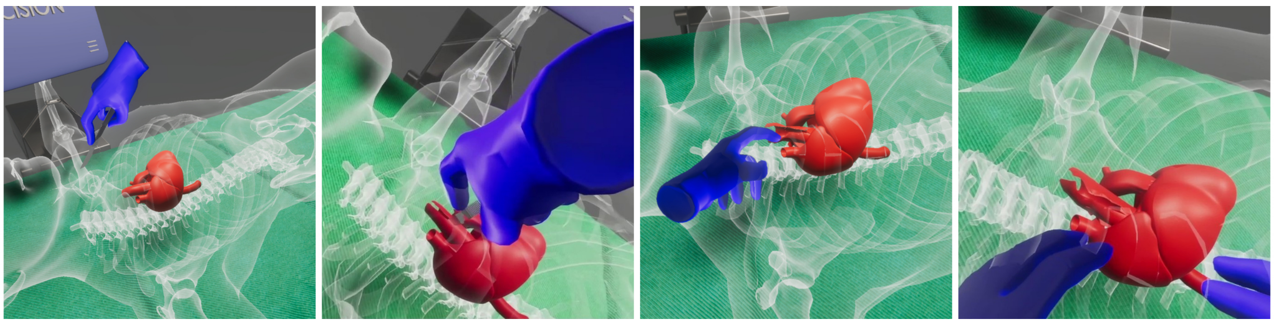

Figure 1: *A surgeon performing a highly-realistic tear operation on a pumping (animated/deformed while tearing) heart in VR. The operation is performed in real-time 10ms, while the 3D heart mesh is simulated and interacted as a soft-body. Further deformation on the soft tissues can be applied, being layered on top of a linear-blend skinned skeleton mesh.*

Abstract

We present an algorithm that allows a user within a virtual environment to perform real-time unconstrained cuts or consecutive tears, i.e., progressive, continuous fractures on a deformable rigged and soft-body mesh model in high-performance 10ms. In order to recreate realistic results for different physically-principled materials such as sponges, hard or soft tissues, we incorporate a novel soft-body deformation, via a particle system layered on-top of a linear-blend skinning model. Our framework allows the simulation of realistic, surgical-grade cuts and continuous tears, especially valuable in the context of medical VR training. In order to achieve high performance in VR, our algorithms are based on Euclidean geometric predicates on the rigged mesh, without requiring any specific model pre-processing. The contribution of this work lies on the fact that current frameworks supporting similar kinds of model tearing, either do not operate in high-performance real-time or only apply to predefined tears. The framework presented allows the user to freely cut or tear a 3D mesh model in a consecutive way, under 10ms, while preserving its soft-body behaviour and/or allowing further animation.

CCS Concepts

• ***Computing methodologies*** → *Mesh geometry models; Virtual reality;* • ***Mathematics of computing*** → *Mesh generation;*

1. Introduction

Since their inception, rigged animated models [MtLTM88] have become a major research topic in real-time computer graphics. Experts have been experimenting with various animation and deformation techniques, pushing the boundaries of realism and real-time performance. As the industry of Virtual, Augmented Reality (VR, AR) rapidly grows, the term of full user-immersion is being researched extensively. Fully-immersive virtual reality systems mainly aim to enable users to experience and perceive the virtual environments as real [PP20]. To maintain user immersion at all times, these VR systems must produce and project a high number of frames per second, which implies that the computational latency for each frame should be minimal. In this regard, increasingly more complex and optimized algorithms are being developed. Sophisti-

DOI: 10.2312/egve.20221275

cated computer graphics tools involve the ability to perform cuts, tears and drills on the surface of a skinned model [BSM*02, KP21]. Such algorithms are aiming to increase user immersion and to be used as sub-modules of even more complex operations. However, their scale-up for the extreme real-time conditions of virtual reality environments utilizing mobile, all-in-one un-tethered head-mounted displays (HMDs), remains an active field of research.

The need to interact in a shared virtual environment with other participants in the upcoming metaverse pushes the envelope for more realistic deformation simulations that lead to more complex techniques and interaction paradigms. In the physical world, certain deformable objects, e.g., soft or hard tissues, are deformed naturally when external forces are applied on them. To preserve immersion and avoid the uncanny valley in VR, the rigid object's physical behavior needs to be replicated in VR too [TPBF87, MMCK14, PLK*18]. One way to accomplish this is via the so-called *soft-body mesh deformation* [PZL*20], a suite of algorithms that essentially dictates how the vertices of a mesh should affect one another when an external force is applied anywhere on the surface of the model.

Performing interactive cuts on a model is not something new; However, most techniques are not suitable for applications requiring high frame-rates as they are based on finite-element methods. Moreover, implemented cuts in such applications are in most cases constrained: camera, model or user degrees of freedom, i.e. the user cannot freely cut anywhere on the model; a set of predefined cuts and their animations are usually produced and placed in the virtual environment by VR designers or artists, and each one is played when triggered by the user's specific and constrained actions.

In this work, we propose a framework that allows the user to perform realistic tears, i.e., small cuts, on the surface of a model. Our algorithms are based on pure geometric operations on the surface mesh, and therefore are amenable to yield real-time results in VR, even in low-spec devices such as mobile VR head-mounted displays (HMDs). The significance of our work lies on the fact that in the current state-of-the-art, similar tears on a rigged 3D model in VR are predefined via linear-blend skinning animations, in order to allow them to playback in real-time. Our methods can be implemented in modern game engines such as Unity3D and Unreal Engine; convincing results are illustrated in the video accompanying this work (also, see Fig. 1). The specific calculations must be performed in real-time within a 10-20 ms to preserve user immersion. The ongoing research for increased realism in virtual environments heavily impacts educational and training applications, especially the ones regarding VR medical training (and beyond) [PKS*22].

2. Previous Related Work

[PO09] proposes a simplified version of previous FEM techniques for use in video-games and real-time simulations. They utilize a linearized semi-implicit solver and a well-mastered and optimized parallelized implementation on CPU of the conjugate gradient method. The adopted approach avoids re-meshing, by constraining the fracture on the faces of the simulation elements. It requires the duplication of vertices, while further introduces "splinters" that hide the produced artifacts. The embedded fracture model relies on maximum tensile stress criterion, element splitting according to a fracture plane, and local re-meshing to ensure a conforming mesh. This approach leads to a fast and robust fracture simulation for stiff and soft materials.

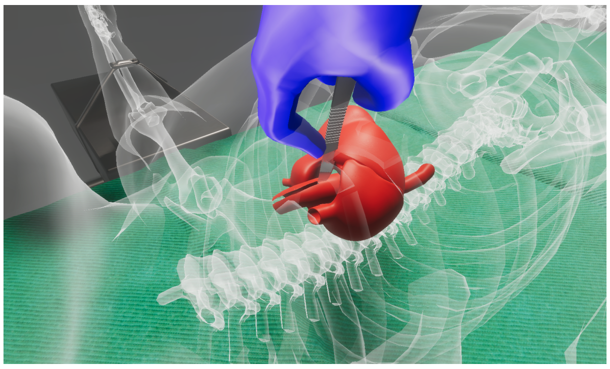
Figure 2: *A close up of a deformable heart being torn by a scalpel. The heart is simulated as a soft body using the proposed particle system.*

[MCS15] proposed a cutting algorithm, based on [SDF07], that allows arbitrary cracks and incisions of tetrahedral deformable meshes. In their work, the utilization of low resolution meshes assists the efficient simulation of the model, while preserving the surface detail by embedding a high-resolution material boundary mesh, for rendering and collision handling. The method allows the accurate cutting of high-resolution embedded meshes, arbitrary cutting of existing cuts, and progressive cuttings during object deformation. The utilized algorithm is based on the virtual node algorithm, that duplicates elements intersecting with the cutting geometry, rather than splitting them. The extended algorithm allows arbitrarily generalized cutting surfaces at smaller scales than tetrahedron resolution, and improves the shortcoming of the original algorithm, that restricted one cut per face and did not handle degenerate cases. The algorithm is based on embedding cracks in virtual elements, which limits the accuracy of the crack propagation computations. In this work several offline progressive cutting use cases were simulated using the proposed algorithm.

Aiming to model physical object cutting behavior, [HQZ*22] proposes an algorithm for highly realistic virtual cutting simulation, showing the contact effect before the cutting occurs, that considers deformable objects' fracture resistance. It utilizes a versatile energy-based cutting fracture evolution model, based on Griffith's energy. It introduces a tailored cut-incision evolution scheme that constraints the cutting tool's interaction with the deformable object, by evaluating the stage at which the cutting starts. To allow the surface indentation prior to cutting, the adapted model uses a material-aware scheme to generate the appropriate realistic and consistent behavior of the cutting tool, and the visual indentation deformation of the object. The designed framework is based on the co-rotational linear FEM model to support large deformations of soft objects and also adopt the composite finite element method (CFEM) to balance between simulation accuracy and efficiency. Additionally, it handles the collision and cut incorporation in the same way as the current FEM-based cutting methods using hexahedral elements. The experimental results show that realistic cutting

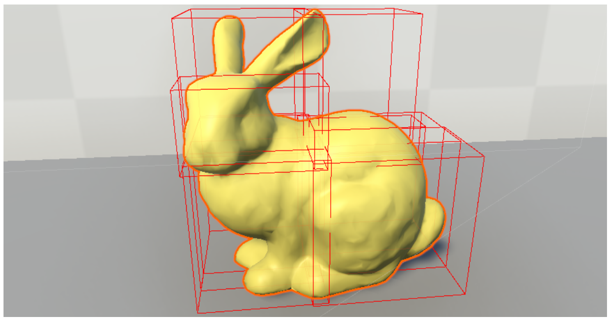

Figure 3: *The segmentation of a bunny model into mesh-sections defined by axis-aligned bounding boxes. In this case, all boxes contain the same number of vertices. Mesh-section overlapping is required to handle tearings that involve faces contained in multiple sections.*

simulations of various deformable objects with various materials and geometrical characteristics that introduce small computational cost for desktop systems.

[LLKC21] proposes real-time tearing and cutting operations on deformable surfaces using local Cholesky factorization updates in the global pass of a projective dynamics solver. These updates assist the handling of the simulations with topology changes. This adopted approach involves addition of new vertices, topological changes, and re-meshing operations, and allows effective gradual and progressive updates, which is common in real-time physics based simulations.

3. Our Approach

Our methodology is based on the techniques of [KP21], where the authors describe simple cut and tear operations on a 3D mesh using basic geometric operations. Our optimized tearing module (see Section 3.1) allows for progressive uninterrupted tears i.e., the user can freely perform tears successively similar to a surgeon's tearing gesture. Furthermore, our cutting module (see Section 3.2) performs straight cuts, producing two (or more) fully deformable submeshes. Both modules operate on deformable meshes, using geometric algebra operations. To accomplish the so-called *soft-body mesh deformation*, we have developed a suitable particle decomposition on the model's vertices based on [NMK*06], where the model's vertices are clustered into groups, and physics *particles* are assigned on each group to handle forces and collisions (see Section 3.3). Upon mesh import, the particles are generated as described in Section 3.3.1, thus enabling soft-body behaviour in the original model. The pipeline used to properly simulate this behaviour in a modern game engine is provided in Section 3.3.2. After performing a tear (see Section 3.1) or cut operation (see Section 3.2) on the model, apart from the partial re-meshing that the model undergoes, a subsequent update of the nearby particles, involving affected vertices, is also required (see Section 3.3.3). This crucial step increases the realism of the torn model, as it allows proper visual simulation, such as deforming or animating, of the torn area. Finally, in Section 3.3.4, we propose an optional step towards optimizing the visual outputs of a torn soft-body model.

Via the proposed algorithms, we are able to perform real-time continuous tears on a soft-body model and update the underlying particle decomposition to obtain highly realistic results in VR. Our methods were designed with the lowest possible computational complexity to yield real-time results and high frame-rates in VR. Lastly, proper handling and weight assignment [KP21] to the tear-generated vertices allow us to tear not only rigid but also skinned models, where in the latter case, further animation is still feasible.

3.1. The Tear Algorithm

In order to achieve real-time tearing results, we have opted for basic geometric primitives, e.g., face-plane intersections and face ray-casting, as basic building blocks for our algorithms. This approach allows for fast identification of the faces affected by the tear.

In our implementation, the tear width is user defined. In non-zero settings, a destructive tear takes place: faces that fall in the tear-gap are completely or partially *clipped*, i.e., removed from the model. Partially clipped faces are calculated by their intersections with the tear-gap surrounding box which is defined by "connecting" consecutive bounding-boxes of single tear segments.

In case of a single tear segment, such a bounding box is aligned and bounded by the scalpel's endpoints in its final position and the scalpel's intersection with the model in its initial position; the width of the box is equal to the user defined tear width (see Fig. 5). As the user moves the scalpel, freely tearing the model, several scalpel's positions are sampled at specific time or distance intervals, defining multiple consecutive tears segments. In case of abrupt movements in the scalpel's trajectory, the algorithm forces extra sampling on the scalpel's position. To avoid jagged edges on the tearing path, the algorithm makes sure that consecutive bounding boxes do not overlap, by utilizing non rectangular bounding boxes instead (see Fig. 5).

To further optimize the performance of the tear algorithm, the mesh is segmented into smaller groups, called *mesh sections*. Each mesh section is defined as an axis aligned bounding box and contains groups of the mesh faces. This division of the mesh into smaller sections reduces significantly the Tear algorithm running time, as the affected mesh section is only a small subset of the entire mesh (see Fig. 3). The number and size of these sections are user defined.

Some comments on the steps of the tear algorithm are found below.

- In Line 5, the affected mesh sections are identified and searched for faces to be added to the search list S. This reduces running times, especially in complex models with a large number of vertices.
- In Line 9, a more sophisticated check that ensures a tested face T will remain unaffected in the torn model, and therefore, it can be removed from S for the next iteration, which further reduces the loop iterations in Line 6.
- The operation in Line 18, ensures that a properly triangulated mesh will be produced, i.e., a vertex on an edge will be connected to the opposite vertex in both adjacent faces (see Fig. 4).
- In Line 12, if the model is a soft-body (see Section 3.3), the particles map is also updated (see Section 3.3.3).

Algorithm 1 Tearing Algorithm

```
Input: Triangulated Mesh $M$, scalpel's position at specific time steps, Mesh Sections.
Require: Scalpel properly intersects $M$ at these time steps and that a tearing plane between timesteps is properly defined.
Output: The mesh resulting from $M$ getting torn by the scalpel.
 1: for each Two consecutive timesteps do:
 2:     Define a bounding box for the scalpel's tear segment.
 3:     Smoothen out the intersection of two bounding boxes by replacing a plane of the leading box by a the "touching" of the next one.
 4:     Determine the vertices of the mesh section(s) that fall into the bounding boxes, by performing multiple side tests of the planes defining the bounding boxes.
 5:     Define a search list $S$ containing all faces containing such vertices.
 6:     for each face $T$ in $S$ do
 7:         for each plane $\Pi$ of the first bounding box $B$ do
 8:             if $T$ intersects $\Pi$ then
 9:                 if The intersection points fall inside the band of the neighbouring planes of $\Pi$ then
10:                     Retriangulate the face into two smaller ones.
11:                     Keep only the smaller face(s) that lie outside $B$, thus clipping the mesh inside $B$.
12:                     Determine the normals/uvs of the intersection points via interpolation. In case of a rigged model, also determine the intersection point weight values from the nearby vertices.
13:                 end if
14:             end if
15:         end for each
16:     end for each
17:     for each Smaller face kept from previous loop do
18:         Do a second retriangulation pass, i.e., search its neighboring face and split it into two parts, if not split already.
19:     end for each
20: end for each
21: Update the mesh model and finalize the tear, by sending the cached final vertices and faces to the GPU buffer to properly update the mesh model.
```

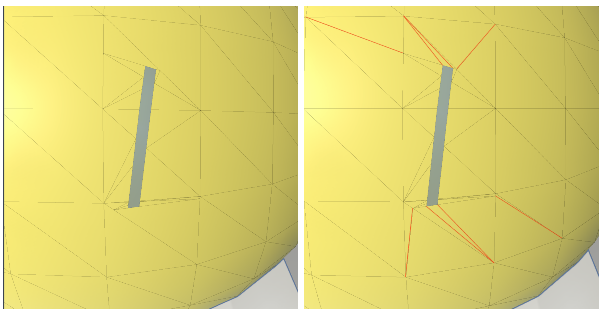

Figure 4: *An additional triangulation pass ensures an intersection point on an edge is properly connected to both vertices on the adjacent faces. Left figure depicts the result before this "second pass"; right figure shows, in red, the edges added in this step.*

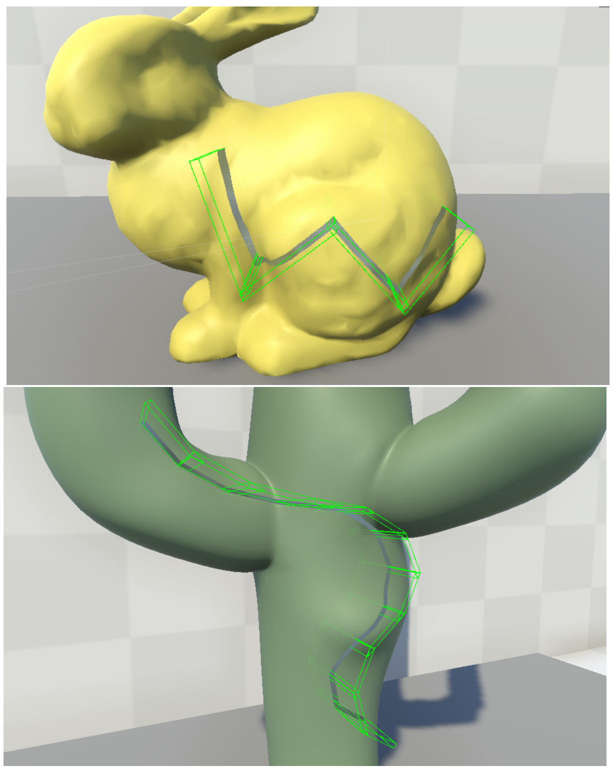

Figure 5: *Bounding boxes defined during a tear. The scalpel's position is sampled based on a time or distance threshold, and a sequence of over overlapping bounding boxes is created. A larger threshold is used in the bunny model, whereas a smaller one on the cactus; the length of the boxes changes proportionally to this threshold.*

3.2. The Cut Algorithm

The algorithm to perform a thorough straight cut on the mesh model is a simplified version of the respective single tear algorithm. Indeed, we define the cutting plane Π as the plane that goes through the following three points: the initial intersection point of the model mesh with the scalpel at a time step, and the scalpel's endpoints after a specific time step; notice that these three points should not be co-linear, otherwise the selected time step is altered.

As in tear algorithm, if the model is a soft-body (see Section 3.3), the particles map is also updated (see Section 3.3.3) during Line 4. After applying the cutting algorithm, each sub-model will lie on the same side of the cutting plane (see Fig. 6).

3.3. The Particle System

3.3.1. Generating the Initial Particles

The offline process followed to generate the initial particles is summarized in Algorithm 3.

Some remarks on the particle generation algorithm can be found below

- The Poisson Disk Sampling [BWWM10] in Line 1 ensures that

Algorithm 2 Cutting Algorithm

```
Input: Triangulated Mesh $M$, cutting plane $\Pi$.
Output: Two sub-meshes resulting from $M$ getting cut by the plane.
 1: for each Face $T$ in $M$ do
 2:     if $T$ intersects $\Pi$ then
 3:         Evaluate intersection points and add them to the $M$.
 4:         Determine the normals/uvs of the intersection point via interpolation. In case of a rigged model, also determine the intersection point weight values from the nearby vertices.
 5:         Retriangulate $T$ into three smaller ones, containing the intersection points.
 6:         Replace $T$ in the model with the smaller sub-faces.
 7:     end if
 8: end for each
 9: for each Face $T$ in $M$ do
10:     if Any vertex of $T$ lies on the positive side of $\Pi$ then
11:         Put $T$ and its vertices on the positive sub-mesh.
12:     else if Any vertex of $T$ lies on the positive side of $\Pi$ then
13:         Put $T$ and its vertices on the negative sub-mesh.
14:     end if
15: end for each
```

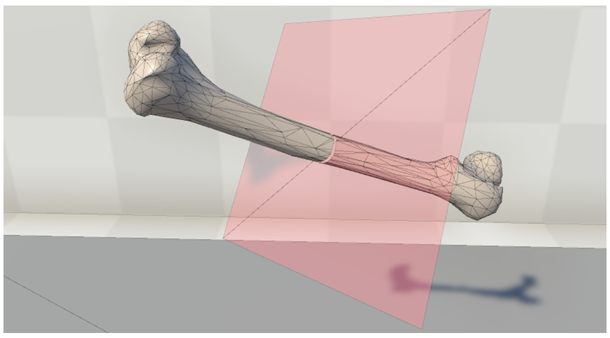

Figure 6: *A bone is cut. The cutting plane, defined by the scalpel, is denoted in red. The original bone model is dissected in two parts, each containing vertices lying on the same side of the plane.*

the selected points will be uniformly distributed on the mesh, maintaining sufficient distance between them.

- In terms of modern game engines (e.g., Unity3D), we would describe the particles spawned in Line 2 as a GameObject with a spherical collider and a rigidbody to handle physics forces.
- To properly simulate soft-bodies, we apply a typical spring mass approach [NMK*06], with some modifications (e.g. the inside pressure is not used to calculate the movement of each particle). In Line 3, each particle gets connected to its anchor point's position vertex, via a spring, thus ensuring that the particles will always tend to return to their initial anchor position upon displacement. (see Fig. 7)
- In Line 7, the vertices assigned to the particle P are the ones that will be affected by P's potential displacement, with a weight inversely proportional to their distance from the P's spawn position, usually based on a sigmoid function.
- In Line 12 we may determine each particle's adjacent particle neighbours. This grid of particles (see Fig. 8) will eventually act as a set of control points that will enable soft-body deformations; upon moving a particle, all neighbouring particles will also be partially displaced, and the affected vertices will yield the desired effect. A particle's displacement due to the movement of an adjacent particle is inversely proportional to the distance of their anchor points.

Algorithm 3 Particle Generation Algorithm

```
Input: Triangulated Mesh $M$, user defined particle radius $d$ and particle-to-particle distance $\delta$.
Output: Particle map for $M$.
 1: Perform a Poisson Disk Sampling, on the vertices of $M$.
 2: Spawn spherical particles centered at the selected vertices, which are denoted as anchor points.
 3: The particle and its anchor are connected via a spring.
 4: for each Particle $P$ do
 5:     for each Vertex $V$ of the mesh do
 6:         if $V$ lies inside the sphere, centered at $P$'s anchor point, with radius $d$. then
 7:             Assign the vertex $V$ to the particle $P$ and determine the weight influence of $P$ on $V$.
 8:         end if
 9:     end for each
10:     for each Other particle $Q$ do
11:         if $Q$'s anchor point lies inside the sphere, centered at $P$'s anchor point, with radius $\delta$. then
12:             Denote $Q$ and $P$ as neighbouring particles and determine their in-between influence
13:         end if
14:     end for each
15: end for each
```

The connections between particles and vertices affected or neighbouring particles, along with the respective influence weights, are referred to as the *particle map.*

3.3.2. Particle Simulation

The particle simulation is performed almost natively by modern game engines, such as Unity3D, as it involves the same mechanics with joint animation, i.e., both frameworks (particle simulation & joint animation) include control points, and vertices with weights assigned to them.

Particularly, for static meshes, only the particles' anchor positions need to be updated. In each simulation update, Unity3D automatically calculates forces and collisions, and applies the position changes for each particle, and the corresponding weighted displacement on the assigned vertices. On the other hand, skinned meshes involve additional steps in each simulation update. Initially, the particle's anchor position is calculated based on the pose of the model at the specific time step. As the anchor points are essentially vertices of the mesh, the animation equation is applied to obtain these positions. Subsequently the particle's adjacency is updated, by re-applying Line 4 of Section 3.3.1; as the position of anchor points might have been altered, the particles' adjacent neighbours may have changed, based on the δ distance threshold. Finally, the

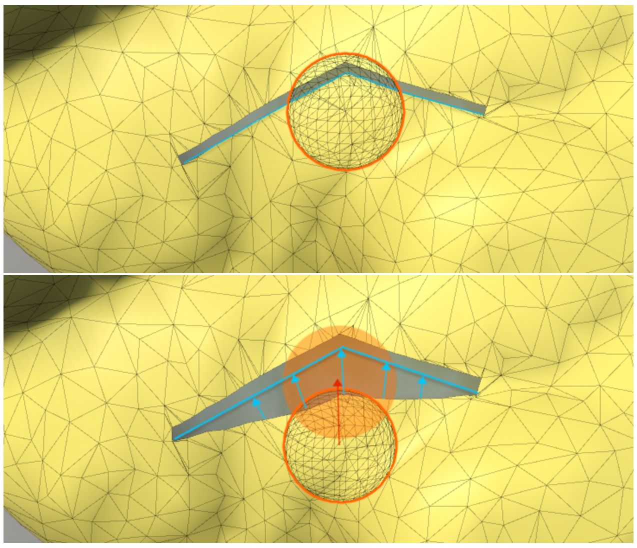

Figure 7: *As a particle (orange circle) is displaced from its initial position (Top), it gains a velocity that tends to return it there (red arrow), simulating elasticity on all affected vertices (Bottom). Blue arrows represent the resulting tendency of the vertices to return to their initial position.*

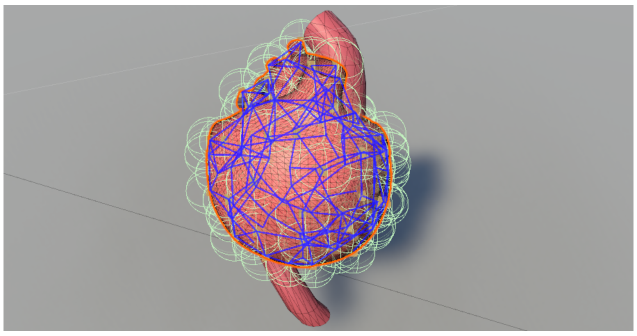

Figure 8: *The particle system layer on top of a heart model. The spheres illustrate the particles and their interconnections (in blue lines).*

same mechanics with a static mesh are applied, i.e., determine the position of the particle and consequently of the affected vertices. Specifically, the final global position f_i of the i-th vertex of the model is determined, by evaluating $\boxed{f_i = \sum_{j \in J(i)} w_{i,j} D_j v_i}$, where

- f_i and v_i are the final and initial homogeneous coordinates of the i-th vertex,
- $J(i)$ contains the indices of the particles that affect the i-th vertex,
- $w_{i,j}$ is the corresponding influence factor between the j-th particle and the i-th vertex, and
- D_j is the 4x4 matrix corresponding to the displacement of the j-th particle from its anchor, in global coordinates.

We obtain the final displacement D_j, either directly, i.e., the user moved a specific set of particles, or indirectly, via model-user interaction in VR, e.g., the user squeezed the model. In the latter case, the game engine's physics component is responsible to evaluate the displacement D_j of the particles, by calculating the forces and collisions involved, at runtime. Under the physics engine hood, the forces applied to particles displace them from their anchor points. As a consequence, their velocity is altered to be proportional to the respective displacement, always pointing to the initial position, thus simulating elasticity.

In the former case, where the user chooses to displace a specific set of particles, we may evaluate the displacement of all particles, by taking into consideration that the particles are interconnected via a spring-like system. Thus, we may determine the displacement D_j of a particle indirectly, moved by its neighbouring particles movements via $\boxed{D_j = \sum_{k \in K(j)} W_{j,k} D'_k}$, where

- $K(j)$ contains the indices of the particles that are connected to the j-th vertex,
- D_j is the final displacement of the j-the particle, which was not displaced directly, but indirectly, due to the adjacency with the k-th vertex,
- D'_k is the 4x4 matrix corresponding to the displacement of the k-th particle that is displaced by the user directly,
- $W_{j,k}$ is the corresponding influence factor between the j-th and the k-th particle, a value inversely proportional to their in-between distance.

3.3.3. Updating the Particles After a Tear or a Cut

After a tear or a cut operation, it is important to update the particle map, in order to preserve the realism of the soft-bodies. This map is updated by adding or removing vertices, as well as modifying the particle connections with the assigned vertices or other neighbouring particles. To produce physically correct deformation results, simple directives are introduced, e.g., vertices belonging to opposite sides of a tear, although close enough, cannot belong to the same particle (see Fig. 9).

Below we provide an overview of the algorithm used to perform the particle update during a tear operation.

Algorithm 4 Particle Update Algorithm

```
1: Assign intersection points introduced by the tear operation to
   particles, as in Line 7 of the Particle Generation Algorithm.
2: If any tear bounding box intersects the segment connecting two
   neighbour particles, these particles are no longer considered
   neighbours.
3: for each Particle $P$ do
4:     Remove any vertex $V$ affected by $P$ from its influence list if
       $V$ and the anchor point of $P$ lie on different sides of any tear
       plane.
5: end for each
```

Regarding Line 4, this simple-to-describe objective is one of the most important and challenging primitives, in order to avoid potential artifacts. Special focus was given to intersection vertices that were introduced close to the connection of two consecutive tears segments, i.e., in the intersection of two bounding boxes. To properly identify whether a vertex lies on the opposite side of the anchor point with respect to a *tear plane*, i.e., the plane splitting the

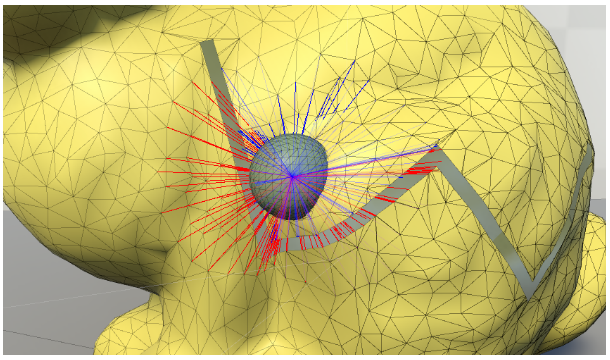

Figure 9: *A spherical particle's (gray) anchor point (i.e., center) is connected with segments to all initially affected vertices, before the tear operation. A red segment indicates that the corresponding vertex is no longer affected after the tear, as a tear bounding box intersects with it. A blue segment, indicates that the vertex is still affected after the tear.*

bounding box in half, containing the scalpel's endpoints, we had to consider both the current and the previous tear segment. Additionally, to provide correct results in lower running times, for a given particle P, we only considered checking against tear planes that corresponded to sufficiently close bounding boxes. To be more specific, if the vertex, lying on a bounding box (i.e., an intersection point introduced during the tear operation), closest to the anchor point of P, is not affected by P, then the corresponding check of the particle against the corresponding tear plane may be omitted.

A similar but simpler methodology is followed after a cut operation on the mesh. In that case, the particle clustering map is updated by removing all vertex-particle or particle-particle connections where the corresponding connection segment intersects the cutting plane.

3.3.4. Adding More Particles for Optimized Tear Animation

The method proposed for a progressive tear operation on a mesh model with a subsequent update of the existing particles, yields highly realistic results. However, the described particle system still misses to model the absolute physical behavior of human tissue. To model such realistic behavior, the reaction of human tissue after a tear operation would be to animate and slightly open up the wound. To achieve this type of animation, we consider an auxiliary set of newly created particles around the tear slit (see Fig. 10). These new particles will be assigned to all vertices that took part in the two triangulation passes. A slight displacement of the new particles' anchors in a direction normal but away from the tear segments makes the animation possible.

However, the insertion of additional particles leads not only to increased realism, but also increased running times.

4. Results and Discussion

We experimented our method using various mesh models; some representative use cases are shown in Figs. 1, 11 and 2. In all cases, further cut/tear operations, soft-body deformations and/or model animations are possible. Tables 1 and 3 contain the time required to perform the algorithms for tear and cut respectively. Every presented running time is the average time of 10 or more runs, obtained using a Windows 11 PC equipped with AMD Ryzen 7 5800H at 3.2GHZ, 16GB RAM and an Nvidia RTX 3060 (6GB RAM) graphics card. The proposed method was implemented exclusively using non-parallel CPU computations. Our methods are also partially implemented in a VR medical training application [ZKK*21], running on a modern game engine performing identical results and producing real-time frame-rates, suitable for desktop but also for VR immersive systems. In our experiments, an HTC Vive Pro tethered HMD was used to properly validate that a satisfying immersive quality of experience (QoE) on the user's end was achievable using our framework; this is illustrated in the video accompanying this work.

Experiments of [HQZ*22] showed partial cutting simulations of various deformable objects, called fractures, which correspond to our tear operations. The method produces highly realistic virtual cutting simulations considering the deformable object's fracture resistance. The scalpel cutting seems to be a little inaccurate with respect to the visualized cut. In terms of computational results, the method introduces a relatively low overhead on desktop stations while no experimentation is mentioned on demanding frame-rate systems such as VR or embedded within game engine pipelines. The experiments show (see Table 2) that for medium sized models

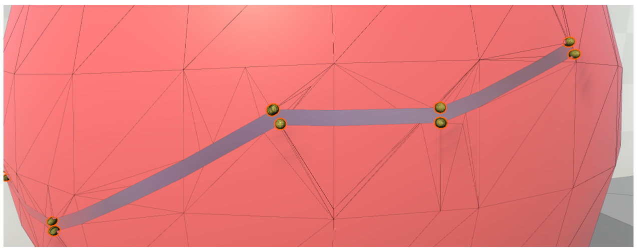

Figure 10: *Location of particles (yellow spheres) additionally spawned during a progressive tear towards improving visual results. A movement of these particles away from the tear achieves higher realism and a smoother animation of the tear opening up.*

Table 1: *Running times required to tear a sphere, a bunny and a heart model.*

Characteristics	Sphere	Bunny	Heart
Number of vertices	515	2527	9747
Number of faces	768	4968	18336
Number of particles	191	179	224
Operation	Running times per tear segment		
Perform Tear	0.36 ms	3 ms	2.54 ms
Update particles	0.39 ms	2.01 ms	0.87 ms
Disconnect Particles	0.91 ms	1.25 ms	2.63 ms
Calculate BoneWeights	0.90 ms	3.81 ms	11.04 ms
Update Mesh	0.07 ms	0.24 ms	0.76 ms
Total Time	3.25 ms	11.19 ms	18.65 ms

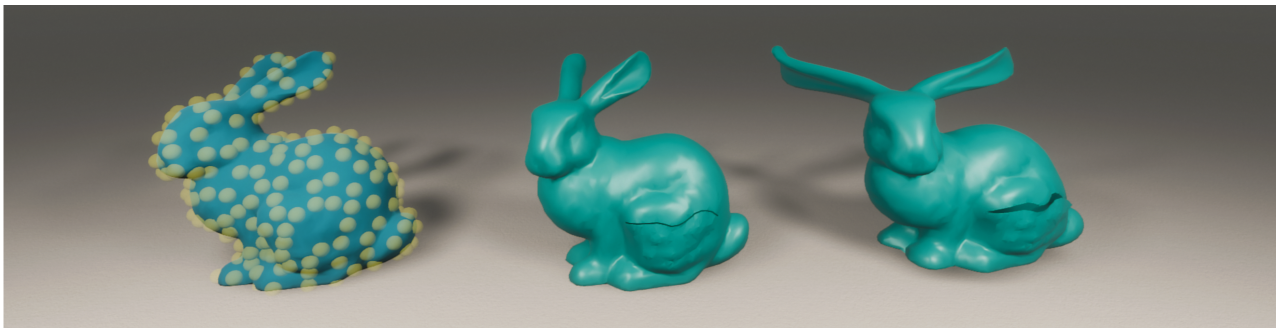

Figure 11: *(Left) The vertices of a 3D model are clustered in particles, to allow soft-body characteristics. (Middle) A continuous tear is performed on the model. (Right) The particles of the torn model are updated, allowing further proper soft-body deformations.*

Table 2: *Comparison of ours tear method with the one presented in [HQZ*22]. Models of similar complexity were used. OUR denotes that the proposed framework was applied.*

Model	Faces	Running Time
Horse	4266	10.14 ms
Bunny (OUR)	4968	11.19 ms
Cuboid	18128	52.77 ms
Heart (OUR)	18336	18.65 ms

Table 3: *Running times required to cut a bone, a bunny and a small cactus model.*

Characteristics	Bone	Bunny	Small Cactus
Number of vertices	516	2527	2976
Number of faces	983	4968	3000
Cut Operation			
Intersection Points	64	356	186
Running times	12 ms	17.29 ms	13.49 ms

the method's results are close to our results, while in larger models our method is much faster.

The work of [MCS15] provide cutting results for various models with no information on their mesh resolution. The experimentation of the method was run on two different desktop platforms. The running times of the method are bound by the utilization of a Newton-Raphson iterative solution scheme. In that regard, the method was not experimented in VR or game engine environments.

The [LLKC21] provide real-time tearing and cutting operations on deformable surfaces. This method is mainly experimented on cloth models which differ significantly from surgical-like tear operations. The simulation involves a local/global solve of projective dynamics with the pre-computed factorization, and the factor modification process. The produced results are indeed satisfying for desktop systems, but not for VR, as the time for a cloth cut is 49ms in total.

5. Conclusions & Future Work

We have presented an algorithm that allows a user to perform unconstrained consecutive tears on a rigged model in VR, while preserving its ability to be deformed as a soft-body. Since our method is geometry-based, it does not require significant GPU/CPU resources, it is amenable to work in real-time VR even for low-spec devices, making it suitable for mobile VR (currently, testing is in progress). We expect that it will eventually pave the way to alter the modern landscape of such VR interactions, where similar operations are mostly predefined. Also, most state-of-the-art methods including physically-correct methods (e.g. Finite Element Methods) cannot be used as they require significant computing resources and/or produce low fps results, unsuitable for mobile VR applications. The proposed framework is already implemented in the MAGES SDK, running on Unity3D, publicly available for free.

In the future, we intend to further optimize our framework to work in a fraction of the current running times by utilizing GPU compute and geometry shaders, taking advantage of the parallel pipeline they offer. So far, the performance overhead by our framework during a VR session, involving high-complexity models, is minimal and in most cases negligible, due to the user's mental preparation time between actions. As tear operations are especially useful for VR medical training scenarios, we would like to explore our algorithm's adaptation to the collaborative needs of multi-user scenarios of such applications. Furthermore, to provide the best possible user experience without getting in the uncanny valley [ZPL*20], we plan to develop an offline FEM model reflecting the ground truth of natural deformation behavior of the human body, that will assist the evaluation process of our method's realism. Lastly, we would like to investigate the utilization of deep learning for the optimal identification of best suited clusterings, based on the model, and the action(s) the user intents to perform.

Acknowledgments

The project was partially funded by the European Union's Horizon 2020 research and innovation programme under grant agreements No 871793 (ACCORDION) and No 101016509 (CHARITY).

References

[BSM*02] Bruyns C. D., Senger S., Menon A., Montgomery K., Wildermuth S., Boyle R.: A survey of interactive mesh-cutting techniques and a new method for implementing generalized interactive mesh cutting using virtual tools. *The journal of visualization and computer animation 13*, 1 (2002), 21–42. 2

[BWWM10] Bowers J., Wang R., Wei L.-Y., Maletz D.: Parallel Poisson disk sampling with spectrum analysis on surfaces. *ACM Transactions on Graphics (TOG) 29*, 6 (2010), 1–10. 4

[HQZ*22] He S., Qian Y., Zhu X., Liao X., Heng P.-A., Feng Z., Wang Q.: Versatile cutting fracture evolution modeling for deformable object cutting simulation. *Computer Methods and Programs in Biomedicine 219* (June 2022), 106749. doi:10.1016/j.cmpb.2022.106749. 2, 7, 8

[KP21] Kamarianakis M., Papagiannakis G.: An all-in-one geometric algorithm for cutting, tearing, and drilling deformable models. *Advances in Applied Clifford Algebras 31* (Jul 2021), 58. 2, 3

[LLKC21] Li J., Liu T., Kavan L., Chen B.: Interactive cutting and tearing in projective dynamics with progressive cholesky updates. *ACM Transactions on Graphics (TOG) 40*, 6 (2021), 1–12. 3, 8

[MCS15] Mitchell N., Cutting C., Sifakis E.: Gridiron: an interactive authoring and cognitive training foundation for reconstructive plastic surgery procedures. *ACM Transactions on Graphics (TOG) 34*, 4 (2015), 1–12. 2, 8

[MMCK14] Macklin M., Müller M., Chentanez N., Kim T.-Y.: Unified particle physics for real-time applications. *ACM Transactions on Graphics (TOG) 33*, 4 (2014), 104. 2

[MtLTM88] Magnenat-thalmann N., Laperrire R., Thalmann D., Montréal U. D.: Joint-dependent local deformations for hand animation and object grasping. In *In Proceedings on Graphics interface '88* (1988), pp. 26–33. 1

[NMK*06] Nealen A., Müller M., Keiser R., Boxerman E., Carlson M.: Physically based deformable models in computer graphics. In *Computer graphics forum* (2006), vol. 25, Wiley Online Library, pp. 809–836. 3, 5

[PKS*22] Papagiannakis G., Kamarianakis M., Sauter T. C., Chalmers A., Lasenby J., Di Lernia D., Greenleaf W.: Editorial: New Virtual Reality and Spatial Computing Applications to Empower, Upskill and Reskill Medical Professionals in a Post-Pandemic Era. *Frontiers in Virtual Reality 3* (2022). 2

[PLK*18] Papagiannakis G., Lydatakis N., Kateros S., Georgiou S., Zikas P.: Transforming medical education and training with vr using m.a.g.e.s. In *SIGGRAPH Asia 2018 Posters* (New York, NY, USA, 2018), SA '18, Association for Computing Machinery. 2

[PO09] Parker E. G., O'Brien J. F.: Real-time deformation and fracture in a game environment. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2009), pp. 165–175. 2

[PP20] Protopsaltis A., Papagiannakis G.: *Virtual Reality: A Model for Understanding Immersive Computing*. Springer International Publishing, Cham, 2020, pp. 1–4. URL: https://doi.org/10.1007/978-3-319-08234-9_165-1, doi:10.1007/978-3-319-08234-9_165-1. 1

[PZL*20] Papagiannakis G., Zikas P., Lydatakis N., Kateros S., Kentros M., Geronikolakis E., Kamarianakis M., Kartsonaki I., Evangelou G.: Mages 3.0: Tying the knot of medical vr. In *ACM SIGGRAPH 2020 Immersive Pavilion* (New York, NY, USA, 2020), SIGGRAPH '20, Association for Computing Machinery. 2

[SDF07] Sifakis E., Der K. G., Fedkiw R.: Arbitrary cutting of deformable tetrahedralized objects. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2007), pp. 73–80. 2

[TPBF87] Terzopoulos D., Platt J., Barr A., Fleischer K.: Elastically deformable models. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques* (1987), pp. 205–214. 2

[ZKK*21] Zikas P., Kamarianakis M., Kartsonaki I., Lydatakis N., Kateros S., Kentros M., Geronikolakis E., Evangelou G., Apostolou A., Catilo P. A. A., Papagiannakis G.: *Covid-19 - VR Strikes Back: Innovative Medical VR Training*. Association for Computing Machinery, New York, NY, USA, 2021. 7

[ZPL*20] Zikas P., Papagiannakis G., Lydatakis N., Kateros S., Ntoa S., Adami I., Stephanidis C.: Scenior: An Immersive Visual Scripting system based on VR Software Design Patterns for Experiential Training. *The Visual Computer 36*, 10-12 (10 2020), 1965–1977. arXiv: 1909.05719. URL: http://arxiv.org/abs/1909.05719, doi:10.1007/s00371-020-01919-0. 8