Extracting Surface Geometry from Particle-Based Fracture Simulations

Chakrit Watcharopas[1,3](✉), Yash Sapra[2], Robert Geist[1], and Joshua A. Levine[1]

[1] Clemson University, Clemson, USA
cwatcha@clemson.edu
[2] McMaster University, Hamilton, Canada
[3] Kasetsart University, Bangkok, Thailand
chakrit.w@ku.ac.th

Abstract. This paper describes an algorithm for fracture surface extraction from particle-based simulations of brittle fracture. We rely on a tetrahedral mesh of the rest configuration particles and use a simple, table-lookup approach to produce triangulated fracture geometry for each rest configuration tetrahedron based on its configuration of broken edges. Subsequently, these triangle vertices are transformed with a per particle transformation to obtain a fracture surface in world space that has minimal deformation and also preserves temporal coherence. The results show that our approach is effective at producing realistic fractures, and capable of extracting fracture surfaces from the complex simulation.

1 Introduction

To animate solids undergoing fracture, computer graphics researchers frequently use physically-based simulation techniques. These techniques can be grouped into two types of approaches: finite element methods or particle-based methods. While both approaches have advantages and disadvantages, one major distinction between the two is how the domain of interest is discretized for simulation. In finite element methods, a computational mesh is used to represent the solid and naturally provides a description of the fracture surface using a subset of mesh elements. Particle-based methods instead use only a vertex set to represent the solid, and thus a representation of the fracture surface needs to be recovered through a post-processing technique.

This work focuses on a new approach for extracting fracture surfaces from particle-based simulations of brittle fracture. Our approach takes some inspiration from mesh-based approaches, in that we build a tetrahedral mesh of the initial particle set in rest configuration. As the particles move during simulation, we dynamically update information on this mesh. In any given time step, we use a marching tetrahedra approach to locally extract a fracture surface in the rest space configuration. Each triangle on this fracture surface is then mapped into the world space using a best fit linear transformation.

© Springer International Publishing Switzerland 2015
G. Bebis et al. (Eds.): ISVC 2015, Part I, LNCS 9474, pp. 82–91, 2015.
DOI: 10.1007/978-3-319-27857-5_8

The result is a simple, easily parallelizable approach to extracting fracture surface geometry that is practical to implement on GPUs and surprisingly effective. Under the assumption of rigidity, our experiments show this technique can be employed in a variety of complex simulations. Building the input tetrahedral mesh is effectively no more overhead than producing the input point cloud (our approach leverages standard algorithms, such as Delaunay triangulations). The resulting fracture geometry enables new visualizations of particle-based fracture, and it also provides some flexibility in defining where the fracture surface occurs.

2 Related Work

Some of the first approaches for animating fracture involved using deformable models for sheets and cloth [1] and spring-mass networks for 3d solids [2]. O'Brien and colleagues showed that the finite element method could be effective for both brittle [3] and ductile [4] fracture. More recent work has improved the computational efficiency, using quasi-static analysis [5,6] and simplifications of the finite element method for realtime [7]. In these approaches, meshing becomes a concern, especially remeshing near the fracture [8]. Mesh-free methods offer alternatives that avoid remeshing [9–11].

Our work employs the peridynamics-based method of Levine et al. [10] for fracture simulation, but we provide a new solution for fracture surface extraction. Instead of building a predefined piece of geometry for each particle, we extract the surface from a single input mesh. Hirota and colleagues employ a similar approach, but they require that elements must disconnect when a spring breaks (compare Fig. 4 [9] to our Fig. 2). We make no such requirements, which leads to a more complicated case table but further decouples the simulation from the geometry. This also avoids the need to build an implicit-function for the geometry, as is commonly done when skinning particles for fluid simulations [12–14]. Skinning typically builds smooth surfaces that are appropriate for fluids, but struggles to model the sharp features that occur during fracture.

Cutting a single input mesh allows us to use marching methods for fast surface extraction. Our approach is similar in spirit to the well-known Marching Cubes algorithm [15]. Marching cubes focuses only on the representation of manifolds that are level sets, but when fracture occurs it is challenging to model the collection of surfaces as a single level set. Conceptually, our problem is closer to a multi-material representation. Nielson and Franke [16] first proposed a technique for calculating a separating surface in a tetrahedron where its vertices are classified into various classes. Bronson and colleagues [17] extend this idea to lattice cleave multiple material domains. A key difference in our approach is that in any given tetrahedron, an edge can be broken but might still be in the same connected "material" for fracture, necessitating new cases to be developed herein.

3 Algorithm Overview

We use a discretized model of the solid object in the form of tetrahedral mesh where its vertices are the particles we will simulate. To compute a fracture surface, we run a particle simulation, producing time steps $k \in 0 \ldots N$, where each time step tracks the positions of particles and whether any mesh edge is "broken" during simulation. For each time step k, we march through all tetrahedra in the input mesh and perform two operations:

1. look up the case associated with the broken edge pattern (more details in Sect. 3.3) to obtain a set of triangles per tetrahedron, and
2. transform each triangle's vertices (more details in Sect. 3.4)

3.1 Surface Modeling

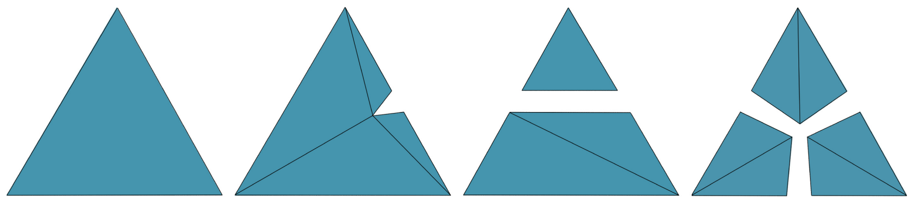
In peridynamics, like many spring-mass simulations, particles are simulated, move around, and may break *bonds* connecting them. We use bonds to describe pairs of particles that interact during the simulation, which by design are a superset of the mesh edges. When a bond that also exists as an edge in the input tetrahedral mesh is broken (i.e. during fracture), we consider that edge to be broken. Our conceptual surface model is that each tetrahedron goes through states of transitions in the rest space. The transition is categorized by the number of broken edges in the tetrahedron. Since a tetrahedron has 6 edges, there are $2^6 = 64$ possibilities of how edges of a tetrahedron can be broken. Collapsing cases by removing rotational symmetries, we can group these possibilities into 11 cases, described as subcases of 7 transitional states – Case 0 through Case 6 – indicating the number of broken edges.

3.2 Face Topologies

Our 11 cases are best described by first examining the possible configurations for a triangular face of a tetrahedron. The set of face topologies that we use is shown in Fig. 1. Using the number of broken edges on a tetrahedral face, we design the four types of face topologies, F0 through F3. The F0 face topology is used when no broken edge has occurred on a tetrahedral face. This topology may make a transition to the F1 when one edge of the tetrahedral face becomes broken, and so on. Note that in the figure we intentionally make the edge breaks wider in the F1, F2, and F3 topologies so that these broken edges can be seen easily. Nevertheless, in general these edge cut-points are placed exactly in the middle of edges and face cut-points are placed inside the face in the rest position, waiting to be transformed to the world position.

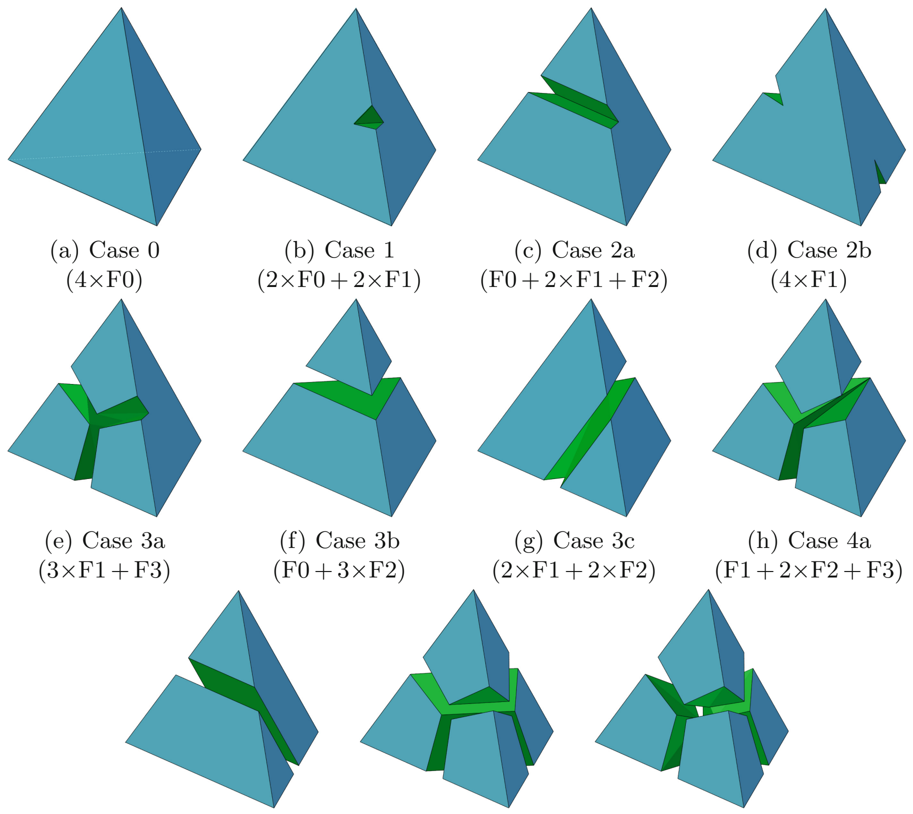
3.3 Case 0 to Case 6

Figure 2 shows all the possible tetrahedral topologies. These can be grouped into a collection of cases based on the number of broken edges.



(a) F0 (b) F1 (c) F2 (d) F3

Fig. 1. Face topologies



(a) Case 0 (4×F0) (b) Case 1 (2×F0 + 2×F1) (c) Case 2a (F0 + 2×F1 + F2) (d) Case 2b (4×F1)

(e) Case 3a (3×F1 + F3) (f) Case 3b (F0 + 3×F2) (g) Case 3c (2×F1 + 2×F2) (h) Case 4a (F1 + 2×F2 + F3)

(i) Case 4b (4×F2) (j) Case 5 (2×F2 + 2×F3) (k) Case 6 (4×F3)

Fig. 2. Generic geometry for the 7 transitional cases with their subcases.

Case 0: Its tetrahedral topology is depicted in Fig. 2a; all four faces have the face topology of F0. For this case, we only generate a boundary surface, without an interior surface. For other cases, the interior surfaces are shown.

Case 1: The topology of this case is illustrated in Fig. 2b where only one edge is broken. As seen in the figure, the four faces of the tetrahedral topology become the following: the back and the bottom faces have the F0 face topology, and the left and the right faces both have the F1 face topology.

Case 2: Two edges are broken. There are two cases for where the two broken edges may appear on a tetrahedron: (a) the two broken edges are on the same face, and (b) the two broken edges are *not* on the same face.

An example of Case 2a is shown in Fig. 2c. When occurring on the same face, these two broken edges create a crack path across the face of the tetrahedron. Its topology contains one F0 face, two F1 faces, and one F2 face. For Case 2b, the topology is different. A crack only appears on each broken edge, as exhibited in Fig. 2d; all four faces have the topology of F1.

Case 3: This case may lead to three subcases. The first subcase (3a) arises when all three broken edges are on the same face (as seen in Fig. 2e). One face topology of this case becomes the F3 topology, which has the "Y" crack pattern on the face. Another subcase (3b) is where all three edges emanating from a single vertex are broken. Under this circumstance, the tetrahedron breaks into two pieces, separating that one vertex from the other vertices. Figure 2f illustrates this subcase. A third subcase (3c) exists when two adjacent faces of the tetrahedron have two broken edges on each face, and the other two faces only have one broken edge. Figure 2g illustrates this subcase.

Case 4: There are two subcases for this situation. The first subcase (4a) is similar to Case 3b where three broken edges happen to share the same vertex. This subcase extends to have one more broken edge on the tetrahedron (Fig. 2h). Its topology contains one F1 face, two F2 faces, and one F3 face. The scenario for the second subcase (4b) is comparable to a cutting plane slicing through a tetrahedron as shown in Fig. 2i. The tetrahedron is separated into two pieces.

Case 5: Fig. 2j depicts this scenario. The broken edges cut a tetrahedron into three separate pieces. Since only one edge is not yet broken (appears as the bottom-back edge in the figure), the back and bottom faces of the tetrahedron have the face topology of F2, while the other two faces have the topology of F3.

Case 6: An illustration of this case is shown in Fig. 2k. The tetrahedron breaks into four disjoint pieces, separating four vertices of the tetrahedron completely. The face topologies of this case are all F3.

3.4 Surface Transformation

The goal of our surface transformation is to make sure that the transformation of the fracture surface from the rest space to the world space will not lead to any gaps or discontinuities in the output surface. Our approach guarantees that any two triangles adjacent in the rest space will stay adjacent in the world space. We assign every triangle vertex of the fracture surface to a transformation from only a single particle. Instead of applying the transformation per tetrahedron, which could create cracks in the output surface, we apply the transformation per tetrahedral vertex. By doing this we establish a one-to-one relationship between simulated particle and triangle vertex of the fracture surface. This may cause some minimal stretching and deformation to the fracture surface, but the result is guaranteed to be crack-free.

After the fracture surface is computed in rest space, we use particle positions from dynamic simulation to find appropriate transformation for the fracture surface. Using Procrustes superposition [18,19], we compute a rigid transformation for each simulated particle. During the dynamic simulation, we use the information on particle bonds to obtain particle connectivity. The direct connectivity from particle i to its neighbors is used to calculate transformation for the particle i. From the connectivity between these particles, we compute the initial particle position in rest space $\mathbf{P}$ and the current particle position in world space $\mathbf{Q}$. Instead of setting transformation from the centroids of $\mathbf{P}$ and $\mathbf{Q}$, we translate with the positions of particle i in rest space and world space, respectively. We then compute the transformation from translated $\mathbf{P}$ to translated $\mathbf{Q}$ using Procrustes superposition. If particle i no longer has any neighbors, we translate its position and use the rotation matrix from the last simulation time step with neighbors.

4 Results

In this section, we present the results produced by our technique. We evaluate the technique by creating four experimental setups. The first experiment is a solid marble sphere flying into a concrete Stanford Armadillo as shown in Fig. 3. The model of the sphere has roughly twice the particle density of the Armadillo. Upon the impact, the sphere propelled the Armadillo backward and caused the body to break apart and the limbs (right arm and tail) to separate from the body.

The second experiment is a projectile shot through a glass plate. This experiment is similar to one described in [10]. Nevertheless, we didn't selectively weaken particle bonds in the glass plate in order to predetermine the fracture pattern. The result in Fig. 4 captures a characteristic spider pattern on the fracture surface. In addition, Fig. 4c shows separately the interior and the boundary triangles of the frame 23. The interior fractures appear at the top of the figure, while the boundary surface is presented at the bottom. Although, from the outside, some regions of the glass surface appear untouched, there were fractures that occurred inside these regions, and our method was able to capture these interior fractures.

The third experiment is a vase dropping onto a table. This experiment shows the effectiveness of per particle transformation over per component transformation. By using the information on how edges are broken inside the tetrahedral mesh, as opposed to which connected component defines the fracture surface, we can compute fractures between pieces of the component that are still attached to each other. Figure 5 shows the result of the experiment, where Fig. 5b displays the simulated particles in world space at frame 70. The particles with the same color belong to the same connected component. As the figure shows, the majority of the vase body at this frame is still in the same component. We compare the per particle transformation shown in Fig. 5c with the per component transformation shown in Fig. 5a. The per component transformation fails to separate

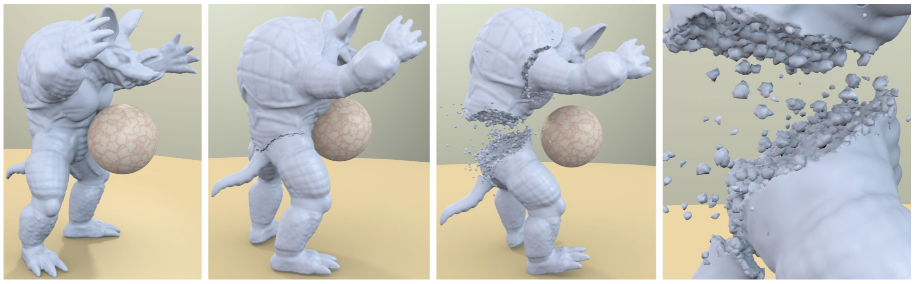

(a) frame 110 (b) frame 120 (c) frame 140 (d) Close-up

Fig. 3. A solid sphere is flying into the Stanford Armadillo.

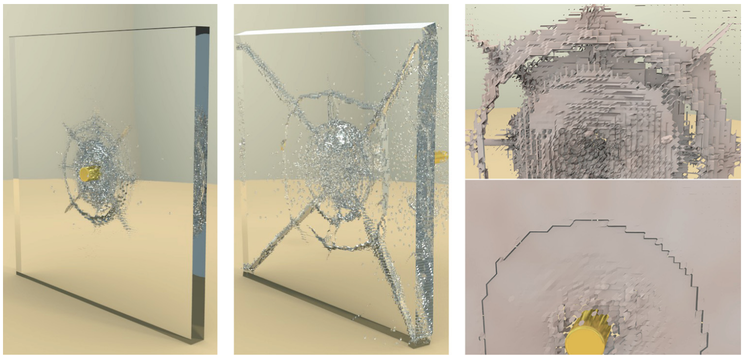
the vase's body and the right handle, producing an inconsistency with the particle simulation. Using the per particle transformation, we see the fracture pieces of the vase are transformed to practically the same locations of the simulated particles in the world space.

Another advantage of using per particle transformation is that we can avoid the temporal discontinuities seen in the case of per component transformations. This problem occurs when a connected component at one frame breaks into multiple components in the following frame. In such situations, each component acquires a new transformation that may be completely different, leading to sudden changes in the transformation. This situation creates so-called popping artifacts, as described in [10]. We perform this per particle transformation in parallel on the GPU, and the added expense is negligible over per component transformations.

The fourth experiment is a hand chopping a stack of plates, demonstrating a complex simulation of multiple object interactions (Fig. 6). The fracture patterns occurred inside the plates in the rest position (from the top to the bottom plates) are also shown in Fig. 6b. The top plate appeared to have more fractures caused by the direct impact from the hand.

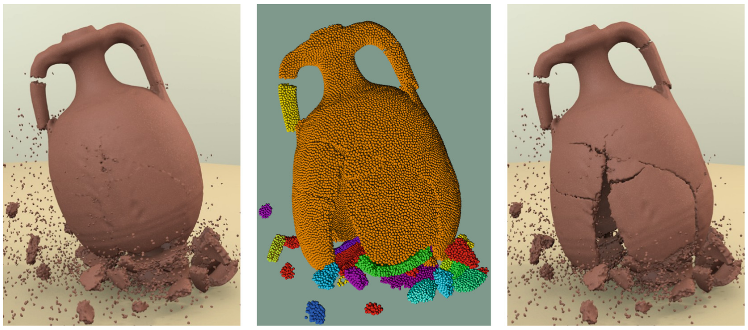
5 Discussion

In Sect. 3.2, we identified a set of face topologies and used them throughout our experiments. The face topology of F3 (three broken edges occurring on the face topology) in this set has some cut-points inside the face, and they can hamper a smooth topology transition from the F2 topology to the F3 topology, since the face cut-points begin to appear in the F3 while no face cut-point exists in the F2. An alternative, second set of the face topologies can help prevent the recreation of geometry when this topology transition occurs. A comparison between the previous and the new F3 topologies is shown in Fig. 7. Although the first set has more cut-points and preserves object mass, the second set has a simpler topology which contains 3 triangles instead of 6, and can also provide the smooth topology transition.



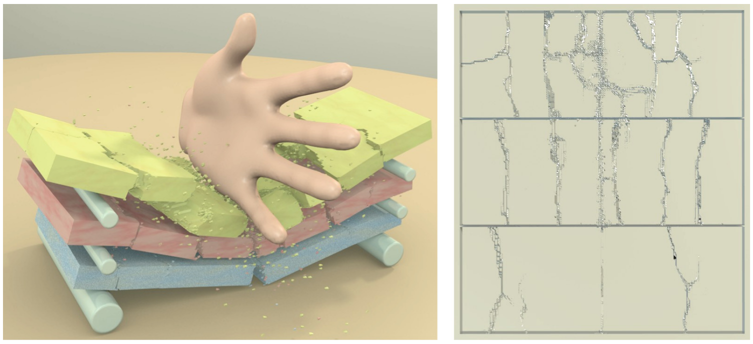
(a) frame 23 (b) frame 35 (c) Inner & Outer surfaces

Fig. 4. A projectile is shot through a glass plate.



(a) Per component (b) frame 70 (c) Per particle

Fig. 5. A vase is dropped on a table.



(a) frame 60 (b) Fractures inside plates in the rest position

Fig. 6. A stack of plates is smashed by a hand.

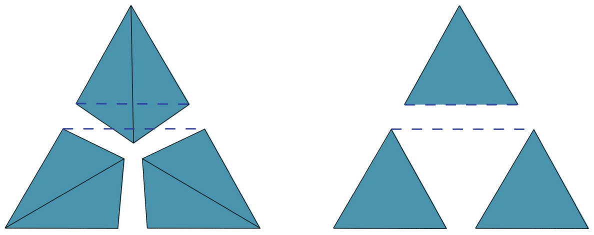

(a) First set (b) Second set

Fig. 7. The F3 face topology of the first and second sets.

Our fracture geometry extraction approach has some limitation. Since our transformation for the fracture geometry does not perform collision detection, we rely on a best fit linear transformation, which assumes rigidity. As a result, if the collision is not carefully evaluated in the simulation, or if the simulation is not rigid, the geometry can have self-intersection. Nevertheless, in practice, we have never encountered this problem. We believe that our approach may tolerate some ductility in fracture, but the simulated tetrahedral mesh may deform significantly. Therefore, we may need a more sophisticated approach than per particle transformations.

We currently place cut-points in the middle of broken edges, which can limit artistic control of the fracture appearance. For example, a planar cut through the material is difficult to obtain if it does not align with the mesh. We plan to extend our approach to allow placing cut-points more effectively so that manipulation of roughness or smoothness on the fracture cuts becomes feasible.

Similar to many marching algorithms, our algorithm can create a large number of triangles. One of our cases will produce as many as 48 triangles for a single tetrahedron. We are currently investigating adaptive approaches that allow us to coarsen the mesh in regions that do not require detailed geometry.

Acknowledgments. This material is based upon work supported by the National Science Foundation under Grant Nos. IIS-1314757 and CNS-1126344. The School of Computing at Clemson University is an NVIDIA GPU Research Center and an NVIDIA GPU Teaching Center. Thanks to David Leubke, Cliff Woolley, and Chandra Cheij of NVIDIA for their support in providing hardware and technical consultation. We also thank Benafsh Husain for reviewing an early draft of this manuscript. The hand model was created by user monatsend and distributed on www.blendswap.com.

References

1. Terzopoulos, D., Fleischer, K.W.: Modeling inelastic deformation: viscolelasticity, plasticity, fracture. Comput. Graph. **22**(4), 269–278 (1988)
2. Norton, A., Turk, G., Bacon, R., Gerth, J., Sweeney, P.: Animation of fracture by physical modeling. Visual Comput. **7**, 210–219 (1991)

3. O'Brien, J.F., Hodgins, J.K.: Graphical modeling and animation of brittle fracture. In: SIGGRAPH, pp. 137–146 (1999)
4. O'Brien, J.F., Bargteil, A.W., Hodgins, J.K.: Graphical modeling and animation of ductile fracture. ACM Trans. Graph. **21**, 291–294 (2002)
5. Müller, M., McMillan, L., Dorsey, J., Jagnow, R.: Computer animation and simulation 2001. In: Magnenat-Thalmann, N., Thalmann, D. (eds.) Real-time Simulation of Deformation and Fracture of Stiff Materials. Eurographics, pp. 113–124. Springer, Vienna (2001)
6. Bao, Z., Hong, J.M., Teran, J., Fedkiw, R.: Fracturing rigid materials. IEEE Trans. Visual. Comput. Graphics **13**, 370–378 (2007)
7. 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, pp. 165–175. ACM (2009)
8. Koschier, D., Lipponer, S., Bender, J.: Adaptive tetrahedral meshes for brittle fracture simulation. In: Proceedings of the 2014 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, Eurographics Association (2014)
9. Hirota, K., Tanoue, Y., Kaneko, T.: Simulation of three-dimensional cracks. Visual Comput. **16**, 371–378 (2000)
10. Levine, J., Bargteil, A., Corsi, C., Tessendorf, J., Geist, R.: A peridynamic perspective on spring-mass fracture. In: Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation. (2014)
11. Pauly, M., Keiser, R., Adams, B., Dutré, P., Gross, M., Guibas, L.J.: Meshless animation of fracturing solids. ACM Trans. Graph. (TOG) **24**, 957–964 (2005). ACM
12. Akinci, G., Ihmsen, M., Akinci, N., Teschner, M.: Parallel surface reconstruction for particle-based fluids. Comp. Graph. Forum **31**, 1797–1809 (2012)
13. Bhattacharya, H., Gao, Y., Bargteil, A.W.: A level-set method for skinning animated particle data. In: Symposium on Computer Animation, pp. 17–24 (2011)
14. Yu, J., Turk, G.: Reconstructing surfaces of particle-based fluids using anisotropic kernels. ACM Trans. Graph. (TOG) **32**, 5 (2013)
15. Lorensen, W.E., Cline, H.E.: Marching cubes: a high resolution 3d surface construction algorithm. In: ACM siggraph computer graphics, vol. 21, pp. 163–169. ACM (1987)
16. Nielson, G.M., Franke, R.: Computing the separating surface for segmented data. In: IEEE Proceedings on Visualization 1997, pp. 229–233 (1997)
17. Bronson, J., Levine, J., Whitaker, R., et al.: Lattice cleaving: a multimaterial tetrahedral meshing algorithm with guarantees. IEEE Trans. Visual. Comput. Graphics **20**, 223–237 (2014)
18. Kabsch, W.: A discussion of the solution for the best rotation to relate two sets of vectors. Acta Crystallogr. A **34**, 827–828 (1978)
19. Twigg, C.D., Kačić-Alesić, Z.: Point cloud glue: constraining simulations using the procrustes transform. In: Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation, pp. 45–54 (2010)