Amorphous Objects Alive! (ESC499 Interim Report)

MICHAEL XU, University of Toronto, Canada
DAVID I.W. LEVIN, University of Toronto, Canada

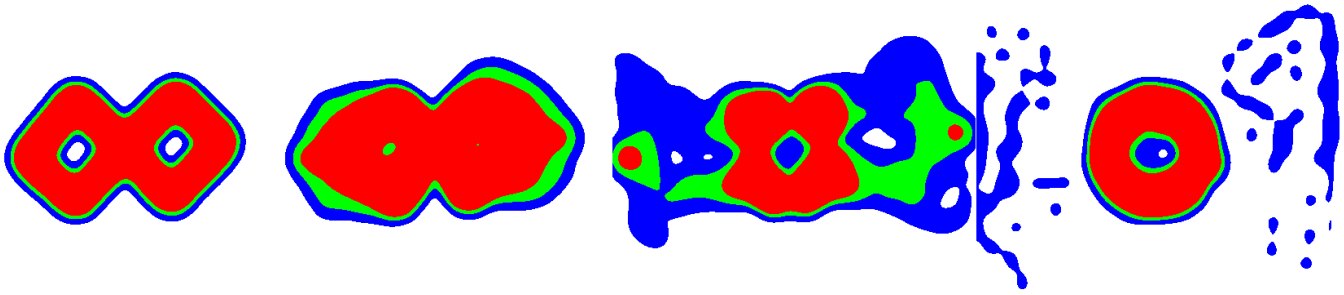

Fig. 1. Our method based on deformation gradient control can produce animations of topologically changing materials. The MPM body starts out in its initial shape (left), then begins to deform its topology (middle left), changing topology and ejecting mass (middle right), until finally reaching the target shape (right). The colors red, green, and blue represent areas of high, medium, and low mass. The blob decided that ejecting its own mass was the fastest way it could decrease its loss function.

ABSTRACT

We present a method of topologically morphing physical objects into user-defined shapes. We take advantage of the material point method's (MPM) ability to implicitly handle topological change. In addition, we define two different loss functions which can measure the "distance" between MPM bodies. We show that minimizing one of these loss functions by controlling deformation gradients can lead to interesting and novel animations. Beyond just target shapes, we show that there are many other parameters users can control to tune the animation to their liking.

CCS Concepts: • **Computing methodologies** → **Physical simulation**.

Additional Key Words and Phrases: material point method, animation control

ACM Reference Format:
Michael Xu and David I.W. Levin. 0000. Amorphous Objects Alive! (ESC499 Interim Report). *ACM Trans. Graph.* 00, 0, Article 0 (0000), 7 pages. https://doi.org/0000

1 INTRODUCTION

Adding physics to your animations is really hard without the help of a physical simulator. However, even with a physical simulator, a new problem arises, which is controlling the physical simulation in a desired way. Our goal is to take advantage of the new and hot material point method and emerging field of differentiable physical simulators to add new control methods to physics-based animation. Our focus is on controlling 2D material point method simulations in an artistic way, though many of the principals can be applied to 3D simulations.

Authors' addresses: Michael Xu, University of Toronto, Canada; David I.W. Levin, University of Toronto, Canada.

0730-0301/0000/0-ART0 $00.00
https://doi.org/0000

2 RELATED WORK

2.1 Material Point Method

2.1.1 MPM formulations. The material point method (MPM) emerged as a particle-in-cell method for solid mechanics [Sulsky et al. 1995]. MPM was first introduced to computer graphics when it was used to simulate snow for Disney's Frozen [Stomakhin et al. 2013]. Jiang et al. [Jiang et al. 2016] provided the first educational resource on MPM for computer graphics. Hu et al. [Hu et al. 2018a] reformulated MPM using a moving-least squares approach.

2.1.2 MPM topology. The main advantage of using a particle-in-cell method such as MPM is its implicit handling of topological change, as opposed to mesh based methods such as Finite Element Method (FEM) where remeshing and mesh distortion can become a significant computational problem. Wang et al. [Wang et al. 2019] explored methods for explicitly handling and tracking topological change in MPM. Continuum damage and fracture mechanics have also been applied to MPM [Wolper et al. 2019], producing impressive fracture simulations. In this work, we have decided to stick to using MPM's implicit topological handling due to ease of implementation.

2.2 Animation-Control

2.2.1 External Force Control. McNamara et al [McNamara et al. 2004] use a combination of Gaussian wind forces to control the animation of fluids and gases. They also introduce the idea of using mass sources the control of level-set fluids. Gentle external forces have also been used for real-time interactive control of deformable objects [Barbič and Popović 2008].

2.2.2 Internal Force Control. The principal of control for elastically deformable characters by only creating internal energy was introduced by Coros et al. [Coros et al. 2012]. When control forces

are generated only from an internal elastic potential, momentum is automatically conserved. This avoids problems of external control methods, where motion can seem non-physical and unrealistic. However, their formulation faces the limits of a finite element mesh such as strict topology. Hu et al. used actuator stresses to control soft robotics simulated in MPM [Hu et al. 2018b], but these examples did not include any topology change.

3 MPM AS A COMPUTATION GRAPH

3.1 MPM Algorithm

We use MLS-MPM since it is faster, easier to implement, and easier to take gradients from than traditional MPM. We refer to Hu et al. [Hu et al. 2018a] for an in depth introduction to MLS-MPM. We will also follow MPM in graphics notation, that is using subscripts i, j, k when referring to grid nodes and subscripts p, q, r when referring to particles.

3.2 Gradient computation

We can view one timestep of MPM as a computation graph, where the nodes are both the material points and the grid nodes. The connections between material points and grid nodes are made if they are within range via the shape functions used in the MPM simulation. We can view this computation graph as one layer of a network of layers, each layer corresponding to a timestep. Thus, by using the chain rule, the gradient of one variable can be taken with respect to any backward dependent variable. We refer to the appendix of Hu et al. [Hu et al. 2018b] which provides many of the gradients we need.

3.3 Gradient Descent

Now that we have a method for computing accurate gradients, we can perform any gradient-based optimization method. We write in pseudocode how to perform gradient descent on MPM to optimize a given loss function with respect to some control parameters in Algorithm 1. Note that we can save computation time by controlling parameters on a reduced number of control timesteps. We define a "temporal iteration" as one pass of gradient descent on all the control timesteps.

Algorithm 1: MPM Spacetime Control

```
Given MPM point cloud, simulation parameters, and n =
  number of timesteps;
Set up spacetime computation graph;
for i = 0; i < total temporal iterations; i++ do
    for each control timestep do
        Run Forward Simulation;
        Compute Loss;
        Compute Gradients w.r.t. control parameters of
          current timestep;
        Perform Gradient Descent;
    end
end
```

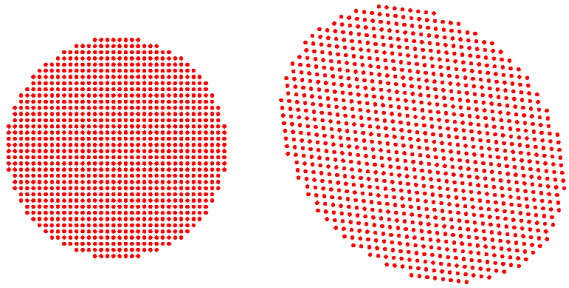

Fig. 2. A solid MPM circle in its initial rest state (left), and its final rest state (right) after setting the deformation gradients to $\left(\begin{smallmatrix} 0.8 & 0 \\ 0.2 & 0.8 \end{smallmatrix}\right)$.

4 DEFORMATION GRADIENT CONTROL

4.1 Deformation gradients: MPM vs FEM

Deformation gradients are computed from tetrahedral rest-states in FEM [Sifakis and Barbic 2012]. MPM, on the other hand, numerically integrates deformation gradients using Equation 1:

$$\mathbf{F}_p^{n+1} = \left(\mathbf{I} + \Delta t \sum_i \mathbf{v}_i^{n+1} \nabla \omega_{ip}^n\right)^{\mathsf{T}} \mathbf{F}_p^n \tag{1}$$

This method of calculating deformation gradients has its disadvantages, such as introducing numerical plasticity. On the other hand, MPM bodies can be expanded, contracted, and sheared easily just by modifying deformation gradients, as seen in Fig. 2. This treatment of deformation gradients gives us a convenient tool for controlling plastic deformations (shape change) of MPM bodies.

4.2 Elastic model

We use the hyperelastic constitutive model of fixed-corotational elasticity for our simulations for its simplicity [Stomakhin et al. 2012], [Jiang et al. 2016]. Note that hyperelastic materials do not experience any plastic deformation normally. Deformation gradient control is what introduces plastic deformation, which produces shape and topology change.

4.3 Position loss function

To automatically control deformation gradients, we introduce a particle position loss function in Equation 2.

$$\mathbf{E}(\mathbf{C}) = \sum_{p=0}^{\mathbf{N}-1} \frac{1}{2} \|\mathbf{x}_p^n - \mathbf{x}_{pc}^n\|^2 \tag{2}$$

Where $\mathbf{C}$ is a spacetime control tensor of deformation gradients, and subscript c denotes that the variable is user-defined. Using this loss function, we are able to generate both deformation and movement based results, shown in Fig. 3.

The limitations of this loss function are that the target shape must be defined as a point cloud, where each point maps to a point in the control point cloud. Complications arise when the target shape can't be represented by an affine transformation of the control point cloud. Deciding how to map the control points can become a complicated transportation problem, which we currently do not

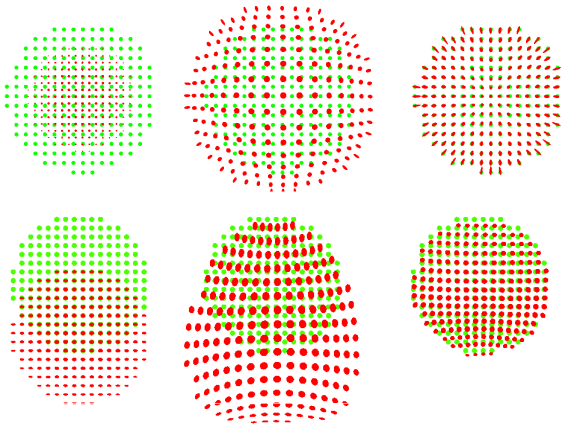

Fig. 3. Control examples using the position loss function (2). (Top) an mpm circle expanding in a zero gravity environment. (Bottom) an mpm circle jumping in a gravity environment. The red particles are the MPM points we are controlling, while the green particles represent the target positions of the points. The deformation gradients of the particles have also been mapped to the circles to form ellipses.

address. Another limitation with defining target points is that the control points have only one desired final destination. An animator may be more interested in the collective MPM body deforming into the target shape, disregarding where each individual control point ends up.

4.4 Mass loss function

To alleviate the aforementioned problems of the position loss function, we define a mass loss function on the MPM grid nodes in Equation 3.

$$\mathbf{E}(\mathbf{C}) = \sum_{i=0}^{\mathbf{M}-1} \frac{1}{2} \|m_i^n - m_{ic}^n\|^2 \tag{3}$$

This function allows us to easily define a target shape. We use image files to create the target point cloud, which in turn is used to create the target mass grid. The complicated transportation problem from the position loss function is solved in the mass loss function using the MPM grid. This allows us to create topology changing examples, for example Fig. 1.

5 RESULTS

We demonstrate a variety of topology changing examples in Fig. 4.

5.1 MPM mass rendering

We use the same MPM grid node basis functions to sample each pixel on the rendered screen using a fragment shader. These sampled pixels represent MPM particles in the continuum that MPM attempts to represent. We use the mass of the sampled MPM pixel to color the pixels, where large masses are red, medium masses are green, small masses are blue, and zero or nearly-zero masses as white.

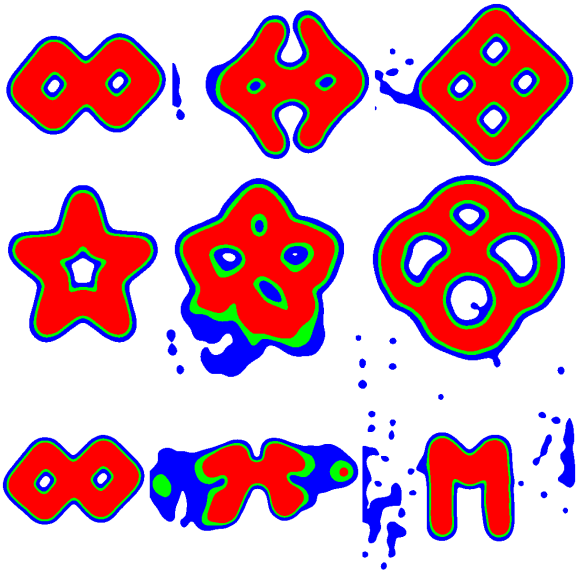

Fig. 4. Control examples using the mass loss function (3). These have been rendered using the mass rendering technique. (Top) genus 2 to genus 4, (middle) genus 1 to genus 4, (bottom) genus 2 to genus 0.

5.2 Mass ejection

Note that in some cases, the optimization decides that ejecting mass is the fastest way to move toward the target shape (decreasing the loss function). This can produce interesting and visually pleasing animations. A more drastic example of mass ejection can be seen in Fig. 5.

5.3 Convergence

To speed up computation, the majority of optimizations produced in this figure are not run to convergence. Running the optimization to convergence in this case means having an infinite amount of temporal iterations, only ending the optimization after it fails to find a sufficient decrease within an entire temporal iteration. Converged animations match the target shape more accurately than non-converged animations, as seen in Fig. 5. This presents a trade-off in computation speed versus target shape matching accuracy. Luckily, some artists would prefer that the object doesn't match the target shape too accurately, which would let them take the advantage of computation speed without losing much in return.

5.4 Extending Animations

It is difficult to determine how many time steps the simulation should run for. If it is too short, the object may not have enough time to morph into the target shape. If it is too long, the optimization will take a long time. A simple trick is to just extend the animation. The idea is to run an optimization to convergence, then take the final point cloud from the optimization and use that as an initial point cloud for another optimization using the same target shape. Not only can this technique be used to give objects an extra arbitrary amount of time to morph into a target shape, but it can also just

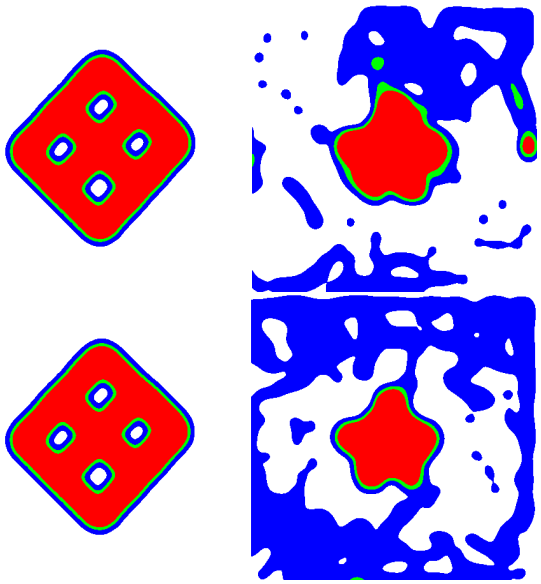

Fig. 5. Initial and final states of a genus 4 object turning into a star. (Top) Non-converged optimization. (Bottom) Converged optimization. Notice how the converged optimization produces a sharper star, although it has more mass ejected. However, the mass missing the target shape is mostly of low density (blue) in the converged example, while the mass missing the target shape is mostly high density (red) in the non-converged example.

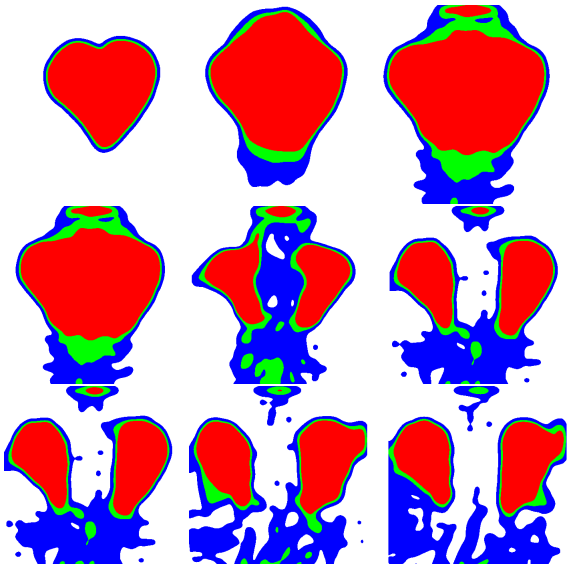

Fig. 6. Initial and final states of a heart breaking apart in an extended animation. Each row represents one optimization, and each new row uses the final point cloud from the previous row as its initial point cloud. All rows are optimizing toward to the same target shape. The first row shows that the optimization wasn't given enough time to break the heart apart. The second row gave the object the time it needed to break apart. In the final row, you can see that the optimization already got as close to the target shape as it will get, so it is just controlling deformation gradients to sustain the animation.

extend an animation where an object needs to sustain a target shape. An object would need some extra deformation gradient control to sustain a target shape since the loss function we minimize does not care about the final velocities or deformation gradients, which means that if the simulation was continued, the object could follow its velocity and deformation to morph into something that is not the target shape. An example of extending animations can be seen in Fig. 6.

5.5 Penalty Grids

If mass ejection is undesired, we can add an extra penalty to mass on nodes outside of the target shape. However, sometimes mass ejection is required for the object to match the target shape, even when the optimization is run to convergence, as seen in Fig. 7.

6 LIMITATIONS AND FUTURE WORK

We created a control framework which automatically modifies the deformation gradients of material points. This effectively creates plastic deformation, which when combined with MPM's implicit topology handling, can produce a multitude of interest shape and topology morphing examples. Using a physical simulation method like MPM also constrains the morphing to be physically accurate. This can produce the mass ejection effect, as seen in Fig. 1 and Fig. 4.

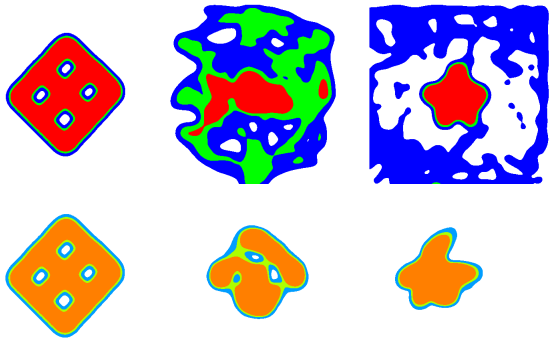

Fig. 7. A genus 4 object morphing into star. (Top) Converged optimization without using a penalty grid. (Bottom) Converged optimization using a penalty grid. Note that the object with the penalty grid wasn't able to create the star shape as accurately. This is unexpected result actually makes sense, since taking away mass ejections can inhibit the objects ability to morph.

6.1 Improving the position loss function

The position loss function has difficulty in creating target shapes, since there needs to be a one-to-one mapping between target points and control points. One method we can try is giving users the option to define their target points by manually deforming their control points using a cage-based skinning technique. Another option is to use more complicated techniques in optimal transport. Regardless of how we do it, there will still be the limitation of control points only having one target position, a restriction that the mass loss function

does not have.

6.2 Improving the mass loss function

In the case of the mass loss function, the amount of detail the target shapes can have depends on the grid resolution. Higher grid resolutions lead to more computational time, which can slow down the optimization considerably. One method we can try is to use a finer target grid, but keep using a coarse simulation grid.

6.3 Penalizing velocity and deformation gradients

To make the morphing less volatile, we can decide to add penalties on the final velocities and deformation gradients of the material points. This can allow animators to use stiffer materials without them causing a numerical explosion in the MPM simulation.

6.4 Material Parameter Control

Another step we could take is material parameter optimization. Different material parameters will produce different animations. These can be controls the animator uses, or the animator can decide they want the MPM blob to alter its own material parameters for the best optimization.

6.5 Color in the Loss Function

To improve the artistic freedom given to animators, we can try to add a color term to the loss function. With this we may be able to generate interesting animations such as the paint in a painting moving around to form another painting.

6.6 Mass control

Finally, we can also experiment with material mass optimization. If the material could alter its own mass, it could create areas of high mass to push on. There could also be many different ways for the MPM blob to use mass optimization that we cannot think of. However, we will need to be careful when using this with the mass loss function. We may need to define a more general grid-based loss function for this technique.

ACKNOWLEDGMENTS

The authors would like to thank the members of the DGP lab for their valuable comments and helpful suggestions.

7 APPENDIX: SOFTWARE

The animations and figures for this paper were produced using a C++ program that uses OpenGL for rendering. GLM (OpenGL Mathematics Library) was used for implementing most linear algebra structures and operations. ImGui (Immediate mode GUI library) was used to create the graphical user interface. The main/novel feature of the program is the MPM spacetime deformation gradient control, but the program can also simulate and interact with MPM objects in real time by making use of the GPU via OpenGL Compute Shaders. The program can also save and load MPM point clouds, while also synthesizing MPM point clouds from images. The code is located

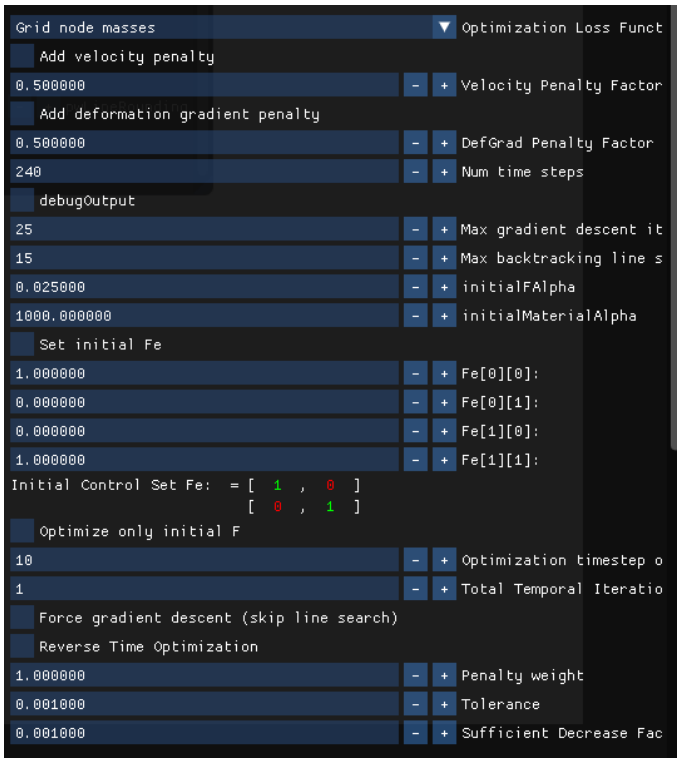

Fig. 8. The GUI containing the optimization parameters users can choose to play with.

at https://github.com/mshoe/MPM_Geometry (currently a private repository).

7.1 General and Spacetime Data structures

The program has data structures for general MPM point clouds and MPM background grids, but uses larger data structures for the MPM point clouds and grids used in spacetime control (for storing gradients). Due to this, users will need to create their point clouds with the general MPM data structures, and then designate them to be used in spacetime control.

7.2 Spacetime optimization parameters

After an initial point cloud and a target point cloud is set, the user can play with a wealth of different parameters, as seen in Fig. 8.

7.3 Modifying rendering methods

After an optimization is complete, users can change the renderings between points and the grid mass rendering, as seen in Fig. 9.

7.4 Real-time Physics Playground

The program also comes with a real-time interactive physics playground. In this mode, MPM is parallelized on the GPU using compute shaders. The user can interact with the objects by creating a gravitational pull where there mouse is. The user can also play with many different visualization features, one of which is shown in Fig. 10.

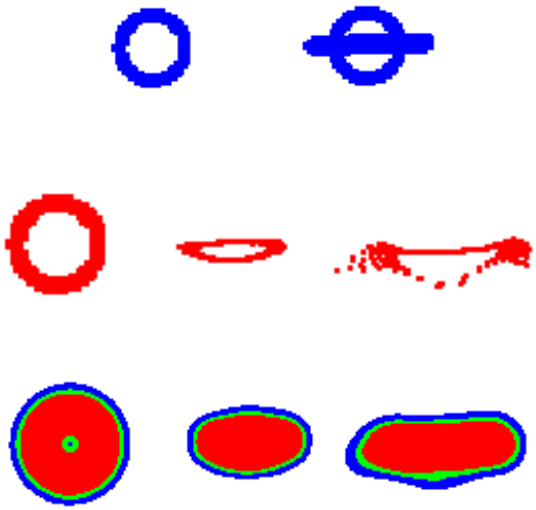

Fig. 9. (Top) (left) Initial MPM point cloud, (right) target point cloud visualized on top of initial point cloud. (Middle) The produced animation, visualized via points. (Bottom) The produced animation, visualized via grid density.

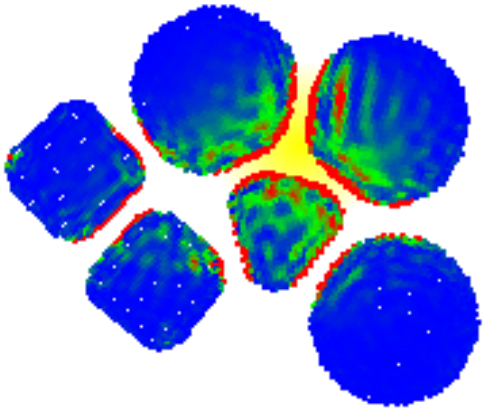

Fig. 10. A user playing a collection of shapes while visualizing their elastic potential energies. The yellow glow represents where their mouse is creating the gravitational pull.

Apart from elastic materials, the user can also play with snow, or at least an approximation of snow in MPM as seen in Fig. 11. The user can also modify the material parameters of the objects they are playing with in real-time to make whatever kind of substance they desire, as seen in Fig. 11.

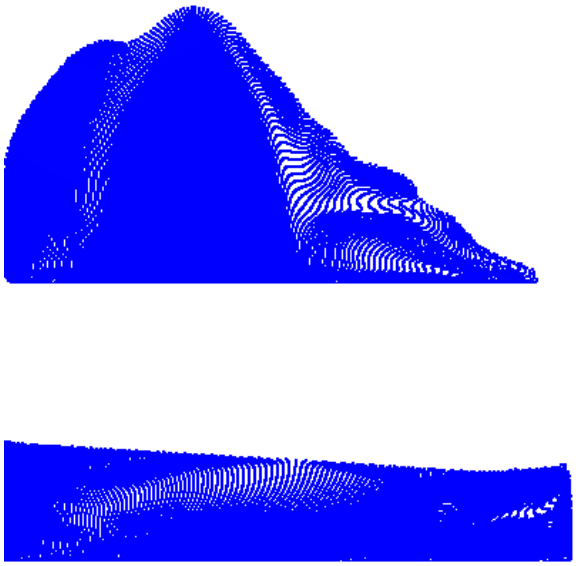

Fig. 11. (Top) MPM snow. (Bottom) The same snow block, but after its shear modulus was reduced, making it a more slimy substance.

REFERENCES

Jernej Barbič and Jovan Popović. 2008. Real-time Control of Physically Based Simulations Using Gentle Forces. *ACM Trans. Graph.* 27, 5, Article 163 (Dec. 2008), 10 pages. https://doi.org/10.1145/1409060.1409116

Stelian Coros, Sebastian Martin, Bernhard Thomaszewski, Christian Schumacher, Robert Sumner, and Markus Gross. 2012. Deformable Objects Alive! *ACM Trans. Graph.* 31, 4, Article 69 (July 2012), 9 pages. https://doi.org/10.1145/2185520.2185565

Yuanming Hu, Yu Fang, Ziheng Ge, Ziyin Qu, Yixin Zhu, Andre Pradhana, and Chenfanfu Jiang. 2018a. A Moving Least Squares Material Point Method with Displacement Discontinuity and Two-way Rigid Body Coupling. *ACM Trans. Graph.* 37, 4, Article 150 (July 2018), 14 pages. https://doi.org/10.1145/3197517.3201293

Yuanming Hu, Jiancheng Liu, Andrew Spielberg, Joshua B. Tenenbaum, William T. Freeman, Jiajun Wu, Daniela Rus, and Wojciech Matusik. 2018b. ChainQueen: A Real-Time Differentiable Physical Simulator for Soft Robotics. *CoRR* abs/1810.01054 (2018). arXiv:1810.01054 http://arxiv.org/abs/1810.01054

Chenfanfu Jiang, Craig Schroeder, Joseph Teran, Alexey Stomakhin, and Andrew Selle. 2016. The Material Point Method for Simulating Continuum Materials. In *ACM SIGGRAPH 2016 Courses (SIGGRAPH '16)*. ACM, New York, NY, USA, Article 24, 52 pages. https://doi.org/10.1145/2897826.2927348

Antoine McNamara, Adrien Treuille, Zoran Popović, and Jos Stam. 2004. Fluid Control Using the Adjoint Method. *ACM Trans. Graph.* 23, 3 (Aug. 2004), 449–456. https://doi.org/10.1145/1015706.1015744

Eftychios Sifakis and Jernej Barbic. 2012. FEM Simulation of 3D Deformable Solids: A Practitioner's Guide to Theory, Discretization and Model Reduction. In *ACM SIGGRAPH 2012 Courses (SIGGRAPH '12)*. ACM, New York, NY, USA, Article 20, 50 pages. https://doi.org/10.1145/2343483.2343501

Alexey Stomakhin, Russell Howes, Craig Schroeder, and Joseph M. Teran. 2012. Energetically Consistent Invertible Elasticity. In *Proceedings of the 11th ACM SIGGRAPH / Eurographics Conference on Computer Animation (EUROSCA'12)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 25–32. https://doi.org/10.2312/SCA/SCA12/025-032

Alexey Stomakhin, Craig Schroeder, Lawrence Chai, Joseph Teran, and Andrew Selle. 2013. A Material Point Method for Snow Simulation. *ACM Trans. Graph.* 32, 4, Article 102 (July 2013), 10 pages. https://doi.org/10.1145/2461912.2461948

Deborah Sulsky, Shi-Jian Zhou, and Howard L. Schreyer. 1995. Application of a particle-in-cell method to solid mechanics. *Computer Physics Communications* 87, 1 (1995), 236 – 252. https://doi.org/10.1016/0010-4655(94)00170-7 Particle Simulation Methods.

Stephanie Wang, Mengyuan Ding, Theodore F. Gast, Leyi Zhu, Steven Gagniere, Chenfanfu Jiang, and Joseph M. Teran. 2019. Simulation and Visualization of Ductile Fracture with the Material Point Method. *Proc. ACM Comput. Graph. Interact. Tech.* 2, 2, Article 18 (July 2019), 20 pages. https://doi.org/10.1145/3340259

Joshuah Wolper, Yu Fang, Minchen Li, Jiecong Lu, Ming Gao, and Chenfanfu Jiang. 2019. CD-MPM: Continuum Damage Material Point Methods for Dynamic Fracture Animation. *ACM Trans. Graph.* 38, 4, Article 119 (July 2019), 15 pages. https://doi.org/10.1145/3306346.3322949