Applications of a Differentiable Physical Simulator Based on the Material Point Method

by

Michael Xu

Supervisor: David I.W. Levin
April 2020

B.A.Sc. Thesis

Division of Engineering Science
UNIVERSITY OF TORONTO

Contents

1 Introduction **4**

2 Related Work **5**

2.1 Material Point Method 5

2.1.1 MPM formulations 5

2.1.2 MPM topology 5

2.2 Animation-Control 5

2.2.1 External Force Control 5

2.2.2 Internal Force Control 6

3 MLS-MPM **7**

3.1 Particle-to-grid transfer (P2G) 7

3.2 Grid operations 7

3.3 Grid-to-particle transfer (G2P) 7

4 MPM as a Computation Graph **9**

4.1 Gradient computation 9

4.2 Gradient Descent 9

5 Deformation Gradient Control **11**

5.1 Deformation gradients: MPM vs FEM 11

5.2 Control Deformation Gradients 11

5.3 Elastic model 12

5.4 Position loss function 12

5.5 Mass loss function 13

5.6 Color loss function 13

6 MLS-MPM Gradients **15**

6.1 Variable Dependencies 15

6.2 Back-propagation initialization 16

6.3 Reverse G2P 17
6.4 Reverse Grid Operations 18
6.5 Reverse P2G 19

7 Results **23**
7.1 MPM mass rendering 23
7.2 Mass ejection 23
7.3 Convergence 24
7.4 Extending Animations 24
7.5 Color field results 25

8 Enhancing the Loss Function **27**
8.1 Higher target grid resolution 27
8.2 Penalty Grids 27

9 Limitations and Future Work **31**
9.1 Improving The Optimization Method 31
9.2 Improving the position loss function 31
9.3 Material Parameter Control 31
9.4 Mass control 32
9.5 Acknowledgements 32

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.

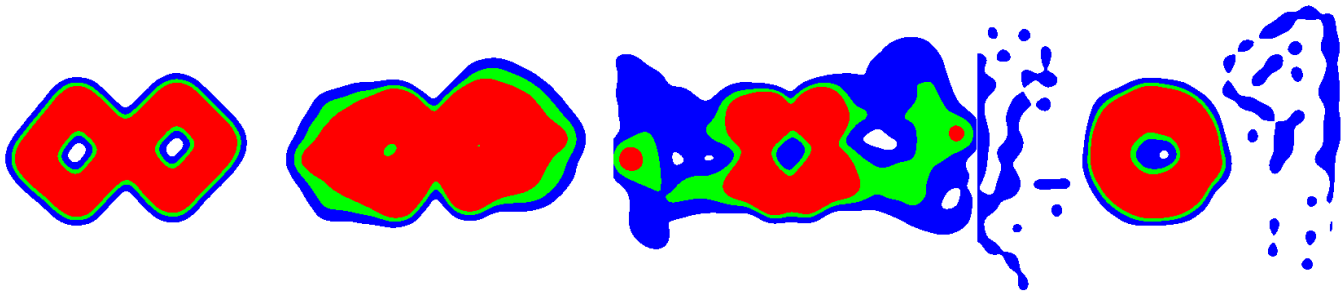

Figure 1: Our method based on deformation gradient control can produce animations of topologically changing materials. The MPM body starts out in its initial shape (left), then begins to deform its topology (middle left), changing topology and ejecting mass (middle right), until finally reaching the 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.

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 (MPM) 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.

In this thesis, we present a method for producing physics-based animations to control the movement of an object, morph an object into another shape or topology, and move the colors of an image around. We leverage the unique properties of MPM, namely the lack of a set topology for the simulated objects which allows for simple deformation control. We then present a handful number of results, and ways to improve our method.

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 MLS-MPM

We will use MLS-MPM [Hu et al., 2018a], a variant of MPM that is computationally more efficient while also being easier to implement. We will not go into detail the derivations behind MPM, the reader is encouraged to go to [Hu et al., 2018a] for MLS-MPM and [Jiang et al., 2016] for MPM in general. 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. The MLS-MPM simulation cycle is broken down into three steps:

3.1 Particle-to-grid transfer (P2G)

Particle properties are transferred to the grid in this step. In the standard case, these would be mass m_p and momentum $m_p \mathbf{v}_p^n$. Momentum is transferred using the Affine-Particle-in-Cell method (APIC) [Jiang et al., 2015] and a moving least squares force discretization [Hu et al., 2018a], weighted by a cubic B-spline kernel N.

$$m_i^n = \sum_i N(\mathbf{x}_i - \mathbf{x}_p^n) m_p \tag{1}$$

$$\mathbf{p}_i^n = \sum_p N(\mathbf{x}_i - \mathbf{x}_p^n)[m_p \mathbf{v}_p^n + (-\frac{3}{\Delta x^2} \Delta t V_p^0 \mathbf{P}_p^n \mathbf{F}_p^{nT} + m_p \mathbf{C}_p^n)(\mathbf{x}_i - \mathbf{x}_p^n)] \tag{2}$$

3.2 Grid operations

Grid velocity is computed by dividing grid momentum by grid mass.

$$\mathbf{v}_i^n = \frac{1}{m_i^n} \mathbf{p}_i^n \tag{3}$$

3.3 Grid-to-particle transfer (G2P)

$$\mathbf{v}_p^{n+1} = \sum_i N(\mathbf{x}_i - \mathbf{x}_p^n) \mathbf{v}_i^n \tag{4}$$

$$\mathbf{C}_p^{n+1} = \frac{3}{\Delta x^2} \sum_i N(\mathbf{x}_i - \mathbf{x}_p^n) \mathbf{v}_i^n (\mathbf{x}_i - \mathbf{x}_p^n)^T \tag{5}$$

$$\mathbf{F}_p^{n+1} = (\mathbf{I} + \Delta t \mathbf{C}_p^{n+1}) \mathbf{F}_p^n \tag{6}$$

$$\mathbf{x}_p^{n+1} = \mathbf{x}_p^n + \Delta t \mathbf{v}_p^{n+1} \tag{7}$$

The MPM algorithm is then just a P2G, grid update, and G2P operation in a loop.

4 MPM as a Computation Graph

4.1 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.

4.2 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. Our version of gradient descent uses a line search to make sure the loss function is decreasing. We also check for convergence based on the magnitude of the gradient. We note that looking into more sophisticated optimization methods may be worth it for future work.

Algorithm 1: MPM Spacetime Control

Given MPM point cloud, simulation parameters, and n = number of timesteps;

Set up spacetime computation graph;

for $i = 1;\ i \leq totaltemporaliterations; i++$ **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

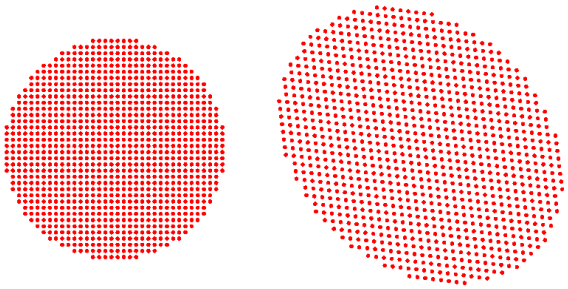

Figure 2: A solid MPM circle in its initial rest state (left), and its final rest state (right) after setting the deformation gradients to $\left(\begin{smallmatrix} 0.8 & 0 \\ 0.2 & 0.8 \end{smallmatrix}\right)$.

5 Deformation Gradient Control

5.1 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 6:

$$\mathbf{F}_p^{n+1} = (\mathbf{I} + \Delta t \mathbf{C}_p^{n+1})\mathbf{F}_p^n \qquad \text{(6 revisited)}$$

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.

5.2 Control Deformation Gradients

We add a control deformation gradient variable $\mathbf{F}_{pc}^n$ to each material point. This control variable is used in addition to the regular deformation gradient $\mathbf{F}_p^n + \mathbf{F}_{pc}^n$. The key difference is that it is not updated by the simulation, only the regular deformation gradient is as seen in equation 8. This control deformation gradient is instead going to be controlled by our optimization.

$$\mathbf{F}_p^{n+1} = (\mathbf{I} + \Delta t \mathbf{C}_p^{n+1})(\mathbf{F}_p^n + \mathbf{F}_{pc}^n) \tag{8}$$

5.3 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.

$$\mathbf{P}(\mathbf{F}) = \frac{\partial \Psi}{\partial \mathbf{F}} = 2\mu(\mathbf{F} - \mathbf{R}) + \lambda(J-1)J\mathbf{F}^{-T} \tag{9}$$

5.4 Position loss function

To automatically control deformation gradients, we introduce a particle position loss function in Equation 10.

$$L(\mathcal{C}) = \sum_{p=0}^{\mathbf{N}-1} \frac{1}{2} \|\mathbf{x}_p^n - \mathbf{x}_{pu}^n\|^2 \tag{10}$$

Where $\mathcal{C}$ is a spacetime control tensor of deformation gradients, and subscript u denotes that the value is generated from user-input. 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 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.

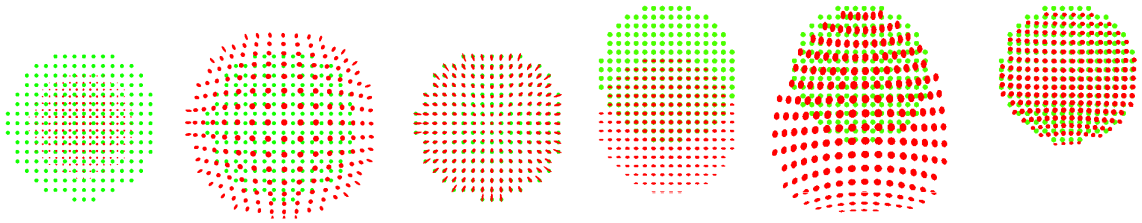

Figure 3: Control examples using the position loss function (10). (Top) an mpm circle expanding in a zero gravity environment. (Bottom) an mpm circle jumping in a gravity environment. The red particles are the MPM points we are controlling, while the green particles represent the target positions of the points. The deformation gradients of the particles have also been mapped to the circles to form ellipses.

5.5 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 11.

$$L(\mathcal{C}) = \sum_{i=0}^{\mathbf{M}-1} \frac{1}{2} \|m_i^n - m_{iu}^n\|^2 \tag{11}$$

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.

5.6 Color loss function

Another interesting prospect with deformation gradient control is animating changing colors fields. We introduce a new color vector $\mathbf{c}_p$ to the material points, and a color-mass $\mathbf{c}_i$ vector to the grid nodes. The color-mass vector on the grid nodes is calculated just like momentum is:

$$\mathbf{c}_i^n = \sum_p N(\mathbf{x}_i - \mathbf{x}_p^n) m_p^n \mathbf{c}_p^n \tag{12}$$

With this we can define a color-loss function:

$$L(\mathcal{C}) = \sum_{i=0}^{\mathbf{M}-1} \frac{1}{2} \|\mathbf{c}_i^n - \mathbf{c}_{iu}^n\|^2 \tag{13}$$

With this new loss function, we can attempt to create animations where the colors of an object are rearranged.

6 MLS-MPM Gradients

Here we show the gradients used for our mass-loss function. The majority of these gradients with respect to a general loss function were derived in [Hu et al., 2018b]. The main change we make is the use of the control deformation gradient term $\mathbf{F}_{pc}^n$. We also choose to use cubic B-spline kernels N instead of quadratic B-spline kernels. Finally, we introduce a drag coefficient γ in the P2G momentum equation to improve stability of the simulation and also add more realism.

6.1 Variable Dependencies

The forward simulation comes from the following equations:

$$\mathbf{P}_p^n = \mathbf{P}_p^n(\mathbf{F}_p^n + \mathbf{F}_{pc}^n) \tag{14}$$

$$m_i^n = \sum_i N(\mathbf{x}_i - \mathbf{x}_p^n) m_p \tag{15}$$

$$\mathbf{p}_i^n = \sum_p N(\mathbf{x}_i - \mathbf{x}_p^n) \left[m_p \mathbf{v}_p^n (1 - \Delta t\gamma) + (-\frac{3}{\Delta x^2} \Delta t V_p^0 \mathbf{P}_p^n (\mathbf{F}_p^n + \mathbf{F}_{pc}^n)^T + m_p \mathbf{C}_p^n)(\mathbf{x}_i - \mathbf{x}_p^n) \right] \tag{16}$$

$$\mathbf{v}_i^n = \frac{1}{m_i^n} \mathbf{p}_i^n \tag{17}$$

$$\mathbf{v}_p^{n+1} = \sum_i N(\mathbf{x}_i - \mathbf{x}_p^n) \mathbf{v}_i^n \tag{18}$$

$$\mathbf{C}_p^{n+1} = \frac{3}{\Delta x^2} \sum_i N(\mathbf{x}_i - \mathbf{x}_p^n) \mathbf{v}_i^n (\mathbf{x}_i - \mathbf{x}_p^n)^T \tag{19}$$

$$\mathbf{F}_p^{n+1} = (\mathbf{I} + \Delta t \mathbf{C}_p^{n+1})(\mathbf{F}_p^n + \mathbf{F}_{pc}^n) \tag{20}$$

$$\mathbf{x}_p^{n+1} = \mathbf{x}_p^n + \Delta t \mathbf{v}_p^{n+1} \tag{21}$$

The forward variable dependency is thus:

$$\mathbf{x}_p^{n+1} \longleftarrow \mathbf{x}_p^n, \mathbf{v}_p^{n+1} \tag{22}$$

$$\mathbf{v}_p^{n+1} \longleftarrow \mathbf{x}_p^n, \mathbf{v}_i^n \tag{23}$$

$$\mathbf{C}_p^{n+1} \longleftarrow \mathbf{x}_p^n, \mathbf{v}_i^n \tag{24}$$

$$\mathbf{F}_p^{n+1} \longleftarrow \mathbf{F}_p^n, \mathbf{C}_p^{n+1}, \mathbf{F}_{pc}^n \tag{25}$$

$$\mathbf{p}_i^n \longleftarrow \mathbf{x}_p^n, \mathbf{C}_p^n, \mathbf{v}_p^n, \mathbf{P}_p^n, \mathbf{F}_p^n, \mathbf{F}_{pc}^n \tag{26}$$

$$\mathbf{v}_i^n \longleftarrow \mathbf{p}_i^n, m_i^n \tag{27}$$

$$\mathbf{P}_p^n \longleftarrow \mathbf{F}_p^n, \mathbf{F}_{pc}^n \tag{28}$$

For back-propagation, the reversed variable dependency is:

$$\mathbf{x}_p^n \longrightarrow \mathbf{x}_p^{n+1}, \mathbf{v}_p^{n+1}, \mathbf{C}_p^{n+1}, \mathbf{p}_i^n, m_i^n \tag{29}$$

$$\mathbf{v}_p^n \longrightarrow \mathbf{p}_i^n, \mathbf{x}_p^n \tag{30}$$

$$\mathbf{v}_p^{n+1} \longrightarrow \mathbf{p}_i^{n+1}, \mathbf{x}_p^{n+1} \tag{31}$$

$$\mathbf{v}_i^n \longrightarrow \mathbf{v}_p^{n+1}, \mathbf{C}_p^{n+1} \tag{32}$$

$$\mathbf{F}_p^n \longrightarrow \mathbf{F}_p^{n+1}, \mathbf{P}_p^n, \mathbf{p}_i^n \tag{33}$$

$$\mathbf{F}_{pc}^n \longrightarrow \mathbf{F}_p^{n+1}, \mathbf{P}_p^n, \mathbf{p}_i^n \tag{34}$$

$$\mathbf{C}_p^{n+1} \longrightarrow \mathbf{F}_p^{n+1} \tag{35}$$

$$\mathbf{C}_p^n \longrightarrow \mathbf{p}_i^n \tag{36}$$

$$\mathbf{p}_i^n \longrightarrow \mathbf{v}_i^n \tag{37}$$

$$m_i^n \longrightarrow \mathbf{v}_i^n \tag{38}$$

$$\mathbf{P}_p^n \longrightarrow \mathbf{p}_i^n \tag{39}$$

$$\tag{40}$$

6.2 Back-propagation initialization

To run back-propagation, the forward simulation needs to be completed first. After the forward simulation is completed, we can start from the final timestep (N_t) grid and compute

$\frac{\partial L}{\partial m_i^{N_t}}$ for each grid node:

$$L(\mathbf{C}) = \sum_{i=0}^{\mathbf{M}-1} \frac{1}{2} \| m_i^{N_t} - m_{ic}^{N_t} \|^2 \tag{11 revisited}$$

$$\implies \frac{\partial L}{\partial m_i^{N_t}} = m_i^{N_t} - m_{ic}^{N_t} \tag{41}$$

We can then compute $\frac{\partial L}{\partial \mathbf{x}_p^{N_t}}$ for each material point in the final time step:

$$m_i^{N_t} = \sum_i N(\mathbf{x}_i - \mathbf{x}_p^{N_t}) m_p \tag{15 revisited}$$

$$\implies \frac{\partial L}{\partial \mathbf{x}_p^{N_t}} = \frac{\partial L}{\partial m_i^{N_t}} \frac{\partial m_i^{N_t}}{\partial \mathbf{x}_p^{N_t}} \tag{42}$$

$$= m_p \sum_i \frac{\partial L}{\partial m_i^{N_t}} \frac{\partial N(\mathbf{x}_i - \mathbf{x}_p^{N_t})}{\partial \mathbf{x}_p^{N_t}} \tag{43}$$

We then set the other initial gradients that we will be using. Note that $\mathbf{F}_p^n$ and $\mathbf{C}_p^n$ do not affect m_i^n, their effects are only seen in future timesteps. Also, due to how we are running the MPM algorithm, our order of computations, and our choice of notation, $\mathbf{v}_p^n$ does not affect $\mathbf{x}_p^n$ when we are at timestep n. $\mathbf{v}_p^{n+1}$ however affects $\mathbf{x}_p^{n+1}$, when we are at timestep n.

$$\frac{\partial L}{\partial \mathbf{v}_p^{N_t}} = \mathbf{0} \tag{44}$$

$$\frac{\partial L}{\partial \mathbf{F}_p^{N_t}} = \mathbf{0} \tag{45}$$

$$\frac{\partial L}{\partial \mathbf{C}_p^{N_t}} = \mathbf{0} \tag{46}$$

6.3 Reverse G2P

Now that we have initialized our values, we can start back-propagating. We call this reverse G2P because we are calculating $\frac{\partial L}{\partial}$ for grid properties using $\frac{\partial L}{\partial}$ from particle properties. For $\mathbf{v}_i^n$:

$$\mathbf{v}_p^{n+1} = \sum_i N(\mathbf{x}_i - \mathbf{x}_p^n)\mathbf{v}_i^n \tag{18 revisited}$$

$$\mathbf{C}_p^{n+1} = \frac{3}{\Delta x^2}\sum_i N(\mathbf{x}_i - \mathbf{x}_p^n)\mathbf{v}_i^n(\mathbf{x}_i - \mathbf{x}_p^n)^T \tag{19 revisited}$$

$$\implies \frac{\partial L}{\partial \mathbf{v}_i^n} = \sum_p \frac{\partial L}{\partial \mathbf{v}_p^{n+1}}\frac{\partial \mathbf{v}_p^{n+1}}{\partial \mathbf{v}_i^n} + \sum_p \frac{\partial L}{\partial \mathbf{C}_p^{n+1}}\frac{\partial \mathbf{C}_p^{n+1}}{\partial \mathbf{v}_i^n} \tag{47}$$

$$= \sum_p \left[\frac{\partial L}{\partial \mathbf{v}_p^{n+1}} N(\mathbf{x}_i - \mathbf{x}_p^n) + \frac{3}{\Delta x^2} N(\mathbf{x}_i - \mathbf{x}_p^n)\frac{\partial L}{\partial \mathbf{C}_p^{n+1}}(\mathbf{x}_i - \mathbf{x}_p^n)\right] \tag{48}$$

6.4 Reverse Grid Operations

For $\mathbf{p}_i^n$:

$$\mathbf{v}_i^n = \frac{1}{m_i^n}\mathbf{p}_i^n \tag{17 revisited}$$

$$\implies \frac{\partial L}{\partial \mathbf{p}_i^n} = \frac{\partial L}{\partial \mathbf{v}_i^n}\frac{\partial \mathbf{v}_i^n}{\partial \mathbf{p}_i^n} \tag{49}$$

$$= \frac{\partial L}{\partial \mathbf{v}_i^n}\frac{1}{m_i^n} \tag{50}$$

For m_i^n:

$$\mathbf{v}_i^n = \frac{1}{m_i^n}\mathbf{p}_i^n \tag{17 revisited}$$

$$\implies \frac{\partial L}{\partial m_i^n} = \frac{\partial L}{\partial \mathbf{v}_i^n}\frac{\partial \mathbf{v}_i^n}{\partial m_i^n} \tag{51}$$

$$= -\frac{1}{(m_i^n)^2}\mathbf{p}_i^n \cdot \frac{\partial L}{\partial \mathbf{v}_i^n} \tag{52}$$

$$= -\frac{1}{m_i^n}\mathbf{v}_i^n \cdot \frac{\partial L}{\partial \mathbf{v}_i^n} \tag{53}$$

6.5 Reverse P2G

For $\mathbf{P}_p^n$:

$$\mathbf{p}_i^n = \sum_p N(\mathbf{x}_i - \mathbf{x}_p^n) \left[m_p \mathbf{v}_p^n (1 - \Delta t \gamma) + (-\frac{3}{\Delta x^2} \Delta t V_p^0 \mathbf{P}_p^n (\mathbf{F}_p^n + \mathbf{F}_{pc}^n)^T + m_p \mathbf{C}_p^n)(\mathbf{x}_i - \mathbf{x}_p^n) \right] \tag{16 revisited}$$

$$\implies \frac{\partial L}{\partial \mathbf{P}_p^n} = \sum_i \left(\frac{\partial \mathbf{p}_i^n}{\partial \mathbf{P}_p^n} \right)^T \frac{\partial L}{\partial \mathbf{p}_i^n} \tag{54}$$

For $\mathbf{F}_p^n$:

$$\mathbf{F}_p^{n+1} = (\mathbf{I} + \Delta t \mathbf{C}_p^{n+1})(\mathbf{F}_p^n + \mathbf{F}_{pc}^n) \tag{20 revisited}$$

$$\mathbf{P}_p^n = \mathbf{P}_p^n (\mathbf{F}_p^n + \mathbf{F}_{pc}^n) \tag{14 revisited}$$

$$\mathbf{p}_i^n = \sum_p N(\mathbf{x}_i - \mathbf{x}_p^n) \left[m_p \mathbf{v}_p^n (1 - \Delta t \gamma) + (-\frac{3}{\Delta x^2} \Delta t V_p^0 \mathbf{P}_p^n (\mathbf{F}_p^n + \mathbf{F}_{pc}^n)^T + m_p \mathbf{C}_p^n)(\mathbf{x}_i - \mathbf{x}_p^n) \right] \tag{16 revisited}$$

$$\implies \frac{\partial L}{\partial \mathbf{F}_p^n} = \frac{\partial L}{\partial \mathbf{F}_{pc}^n} \tag{55}$$

$$= \frac{\partial \mathbf{F}_p^{n+1}}{\partial \mathbf{F}_p^n} : \frac{\partial L}{\partial \mathbf{F}_p^{n+1}} + \frac{\partial \mathbf{P}_p^n}{\partial \mathbf{F}_p^n} : \frac{\partial L}{\partial \mathbf{P}_p^n} + \sum_i \frac{\partial \mathbf{p}_i^n}{\partial \mathbf{F}_p^n} : \frac{\partial L}{\partial \mathbf{p}_i^n} \tag{56}$$

$$= (\mathbf{I} + \Delta t \mathbf{C}_p^{n+1})^T \frac{\partial L}{\partial \mathbf{F}_p^{n+1}} + \frac{\partial^2 \Psi}{\partial \mathbf{F}_p^n \partial \mathbf{F}_p^n} : \frac{\partial L}{\partial \mathbf{P}_p^n} \tag{57}$$

$$- \sum_i \frac{3}{\Delta x^2} N(\mathbf{x}_i - \mathbf{x}_p^n) \Delta t V_p^0 (\mathbf{x}_i - \mathbf{x}_p^n) \frac{\partial L}{\partial \mathbf{p}_i^n}^T \mathbf{P}_p^n \tag{58}$$

For $\mathbf{C}_p^n$:

$$\mathbf{F}_p^{n+1} = (\mathbf{I} + \Delta t \mathbf{C}_p^{n+1})(\mathbf{F}_p^n + \mathbf{F}_{pc}^n) \tag{20 revisited}$$

$$\mathbf{p}_i^n = \sum_p N(\mathbf{x}_i - \mathbf{x}_p^n) \left[m_p \mathbf{v}_p^n (1 - \Delta t \gamma) + (-\frac{3}{\Delta x^2} \Delta t V_p^0 \mathbf{P}_p^n (\mathbf{F}_p^n + \mathbf{F}_{pc}^n)^T + m_p \mathbf{C}_p^n)(\mathbf{x}_i - \mathbf{x}_p^n) \right] \tag{16 revisited}$$

$$\implies \frac{\partial L}{\partial \mathbf{C}_p^n} = \sum_i \frac{\partial \mathbf{p}_i^n}{\partial \mathbf{C}_p^n} : \frac{\partial L}{\partial \mathbf{p}_i^n} \tag{59}$$

$$= \sum_i N(\mathbf{x}_i - \mathbf{x}_p^n) m_p \frac{\partial L}{\partial \mathbf{p}_i^n} (\mathbf{x}_i - \mathbf{x}_p^n)^T \tag{60}$$

For $\mathbf{x}_p^n$:

$$m_i^n = \sum_i N(\mathbf{x}_i - \mathbf{x}_p^n) m_p \tag{15 revisited}$$

$$\mathbf{p}_i^n = \sum_p N(\mathbf{x}_i - \mathbf{x}_p^n) \left[m_p \mathbf{v}_p^n (1 - \Delta t \gamma) + (-\frac{3}{\Delta x^2} \Delta t V_p^0 \mathbf{P}_p^n (\mathbf{F}_p^n + \mathbf{F}_{pc}^n)^T + m_p \mathbf{C}_p^n)(\mathbf{x}_i - \mathbf{x}_p^n) \right] \tag{16 revisited}$$

$$\mathbf{v}_p^{n+1} = \sum_i N(\mathbf{x}_i - \mathbf{x}_p^n) \mathbf{v}_i^n \tag{18 revisited}$$

$$\mathbf{C}_p^{n+1} = \frac{3}{\Delta x^2} \sum_i N(\mathbf{x}_i - \mathbf{x}_p^n) \mathbf{v}_i^n (\mathbf{x}_i - \mathbf{x}_p^n)^T \tag{19 revisited}$$

$$\mathbf{x}_p^{n+1} = \mathbf{x}_p^n + \Delta t \mathbf{v}_p^{n+1} \tag{21 revisited}$$

$$\mathbf{G}_p^n := \left(-\frac{3}{\Delta x^2} \Delta t V_p^0 \mathbf{P}_p^n (\mathbf{F}_p^n + \mathbf{F}_{pc}^n)^T + m_p \mathbf{C}_p^n \right) \tag{61}$$

$$\implies \frac{\partial L}{\partial \mathbf{x}_p^n} = \sum_i \frac{\partial L}{\partial m_i^n} \frac{\partial m_i^n}{\partial \mathbf{x}_p^n} + \sum_i \left(\frac{\partial \mathbf{p}_i^n}{\partial \mathbf{x}_p^n} \right)^T \frac{\partial L}{\partial \mathbf{p}_i^n} \tag{62}$$

$$+ \left(\frac{\partial \mathbf{v}_p^{n+1}}{\partial \mathbf{x}_p^n} \right)^T \frac{\partial L}{\partial \mathbf{v}_p^{n+1}} + \frac{\partial \mathbf{C}_p^{n+1}}{\partial \mathbf{x}_p^n} : \frac{\partial L}{\partial \mathbf{C}_p^{n+1}} + \left(\frac{\partial \mathbf{x}_p^{n+1}}{\partial \mathbf{x}_p^n} \right)^T \frac{\partial L}{\partial \mathbf{x}_p^{n+1}} \tag{63}$$

$$= m_p \sum_i \frac{\partial L}{\partial m_i} \frac{\partial N(\mathbf{x}_i - \mathbf{x}_p)}{\partial \mathbf{x}_p} \tag{64}$$

$$+ \sum_i \left[\frac{\partial N(\mathbf{x}_i - \mathbf{x}_p^n)}{\partial \mathbf{x}_p^n} \left(m_p \mathbf{v}_p^n + \mathbf{G}_p^n (\mathbf{x}_i - \mathbf{x}_p^n) \right)^T - N(\mathbf{x}_i - \mathbf{x}_p^n) (\mathbf{G}_p^n)^T \right] \frac{\partial L}{\partial \mathbf{p}_i^n} \tag{65}$$

$$+ \sum_i \frac{\partial N(\mathbf{x}_i - \mathbf{x}_p^n)}{\partial \mathbf{x}_p^n} (\mathbf{v}_i^n)^T \frac{\partial L}{\partial \mathbf{v}_p^{n+1}} \tag{66}$$

$$+ \sum_i \frac{3}{\Delta x^2} \left\{ \frac{\partial L}{\partial \mathbf{C}_p^{n+1}} : \left(\mathbf{v}_i^n (\mathbf{x}_i - \mathbf{x}_p^n)^T \right) \frac{\partial N(\mathbf{x}_i - \mathbf{x}_p^n)}{\partial \mathbf{x}_p^n} \right. \tag{67}$$

$$\left. - N(\mathbf{x}_i - \mathbf{x}_p^n) (\frac{\partial L}{\partial \mathbf{C}_p^{n+1}})^T \mathbf{v}_i^n \right\} \tag{68}$$

$$+ \frac{\partial L}{\partial \mathbf{x}_p^{n+1}} \tag{69}$$

For $\mathbf{v}_p^n$:

$$\mathbf{p}_i^n = \sum_p N(\mathbf{x}_i - \mathbf{x}_p^n) \left[m_p \mathbf{v}_p^n (1 - \Delta t \gamma) + (-\frac{3}{\Delta x^2} \Delta t V_p^0 \mathbf{P}_p^n (\mathbf{F}_p^n + \mathbf{F}_{pc}^n)^T + m_p \mathbf{C}_p^n)(\mathbf{x}_i - \mathbf{x}_p^n) \right] \tag{16 revisited}$$

$$\mathbf{x}_p^{n+1} = \mathbf{x}_p^n + \Delta t \mathbf{v}_p^{n+1} \tag{21 revisited}$$

$$\implies \frac{\partial L}{\partial \mathbf{v}_p^n} = \frac{\partial L}{\partial \mathbf{x}_p^n} \frac{\partial \mathbf{x}_p^n}{\partial \mathbf{v}_p^n} + \sum_i \frac{\partial L}{\partial \mathbf{p}_i^n} \frac{\partial \mathbf{p}_i^n}{\partial \mathbf{v}_p^n} \tag{70}$$

$$= \Delta t \frac{\partial L}{\partial \mathbf{x}_p^n} + \sum_i N(\mathbf{x}_i - \mathbf{x}_p^n) m_p (1 - \Delta t \gamma) \frac{\partial L}{\partial \mathbf{p}_i^n} \tag{71}$$

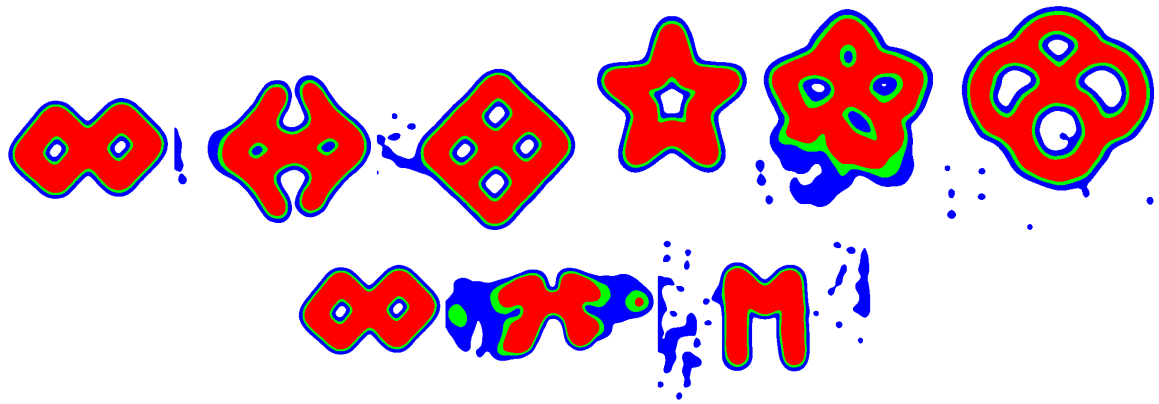

Figure 4: Control examples using the mass loss function (11). These have been rendered using the mass rendering technique. (Top) genus 2 to genus 4, (middle) genus 1 to genus 4, (bottom) genus 2 to genus 0.

7 Results

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

7.1 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.

7.2 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.

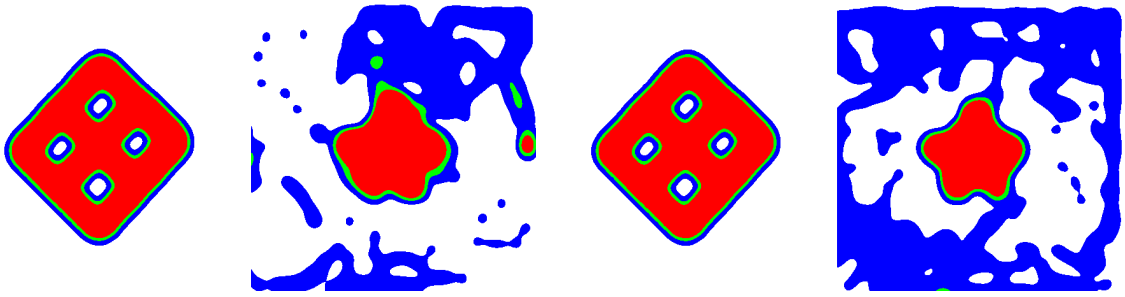

Figure 5: Initial and final states of a genus 4 object turning into a star. (Top) Non-converged optimization. (Bottom) Converged optimization. Notice how the converged optimization produces a sharper star, although it has more mass ejected. However, the mass missing the target shape is mostly of low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

7.3 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.

7.4 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

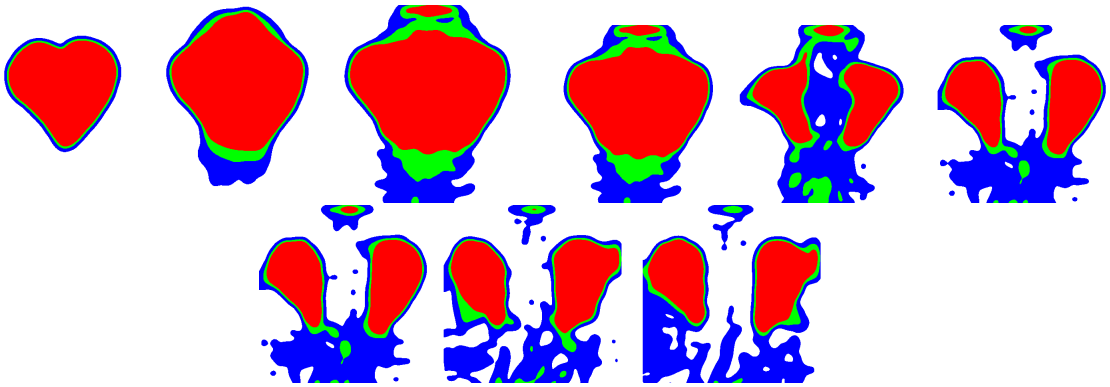

Figure 6: Initial and final states of a heart breaking apart in an extended animation. Each row represents one optimization, and each new row uses the final point cloud from the previous row as its initial point cloud. All rows are optimizing toward to the same target shape. The first row shows that 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.

amount of time to morph into a target shape, but it can also just 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.

7.5 Color field results

We can also try producing color field animations. Examples are shown in figures 7 and 8.

Figure 7: Grid node color rendering. This is a color field optimization where the color distribution in the circles wants to become half-half.

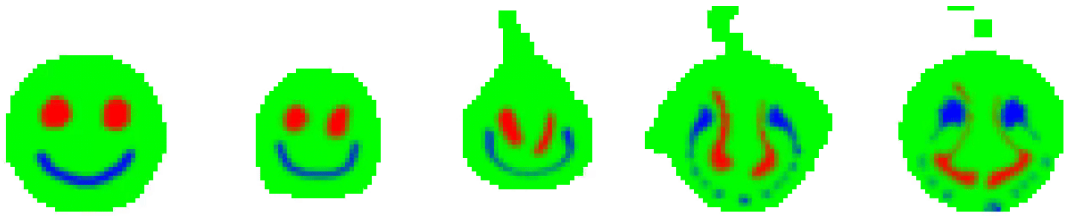

Figure 8: Grid node color rendering. This is a color field optimization where the red and blue nodes want to switch.

8 Enhancing the Loss Function

8.1 Higher target grid resolution

In this section, we run an experiment on how higher target grid resolutions affect. We simply perform gradient descent using the $\frac{\partial L}{\partial \mathbf{x}_p^{N_t}}$ gradients. Our goal is not to produce a physics-based animation, or an animation at all for that matter, but to examine what effects grid resolution can have on our optimization. The goal of our method is to move the material points in a physics-based way, and any way we choose to do that, we will need to use the $\frac{\partial L}{\partial \mathbf{x}_p^{N_t}}$ gradients.

Using the initial and target shapes in Figure 9, we apply this optimization experiment. We see in Figure 10 that the higher resolution target grids allow for a more precise, even distribution along the outline of the target shape. However, not all the points are able to be shifted to the target shape. This is due to our use of cubic B-spline kernels for our grid node shape functions. Higher resolution grids occupying the same space have the side effect that each grid node has a small range of influence. When points are outside of this range of influence, it will result in null gradients. This is why we see in the lower resolution target grid examples, almost all points reach the target shape outline. This shows that the lower resolution grid can give more gradient information regarding the high-level image of the target shape.

We can combine the accuracy of higher resolution grids with the overall shape information given by the lower resolution grids to improve our loss function. What we do is use multiple overlapping target grids of varying resolutions for our mass and color loss functions. An example of this can be seen in figure

8.2 Penalty Grids

If mass ejection is undesired, we can add an extra penalty to mass on nodes outside of the target shape. We can add this as a term in the loss function with a variable α_i, which is a number greater than 1 if $m_{iu}^n = 0$. This will put an extra penalty on nodes where the corresponding target node has no mass.

Figure 9: Left: initial image. Right: target image.

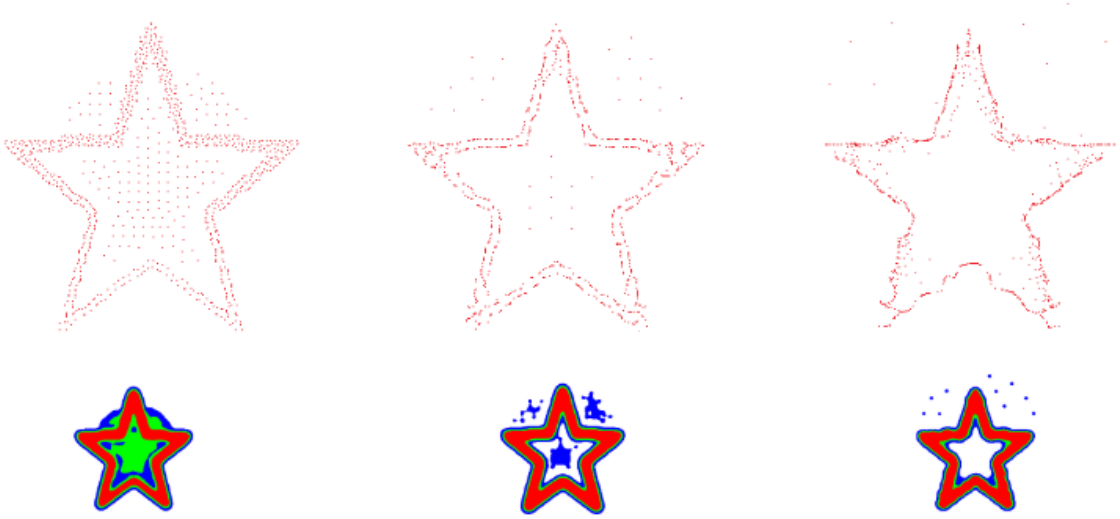

Figure 10: Top: material point rendering. Bottom: density field rendering. Left: position optimization on mass loss function with a 128x128 target grid. Middle: position optimization on mass loss function with a 64x64 target grid. Right: position optimization on mass loss function with a 32x32 target grid.

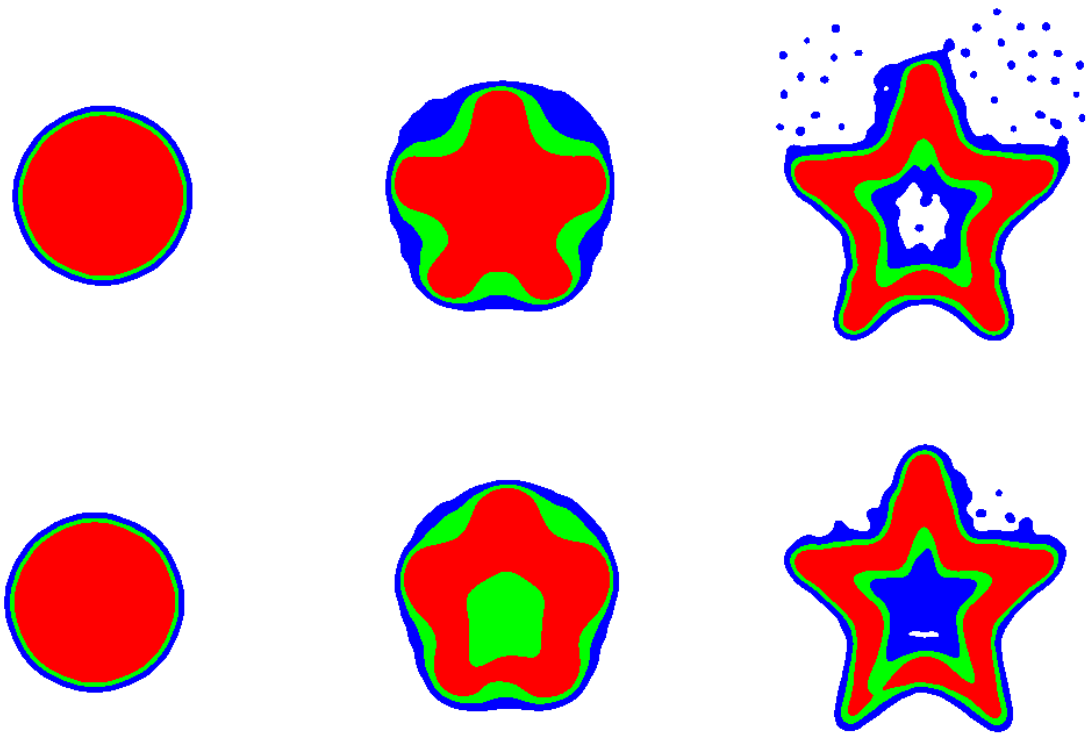

Figure 11: Circle to star morph. Top: using a combination of a 32x32 and 128x128 target grid. Bottom: Using just a 128x128 target grid.

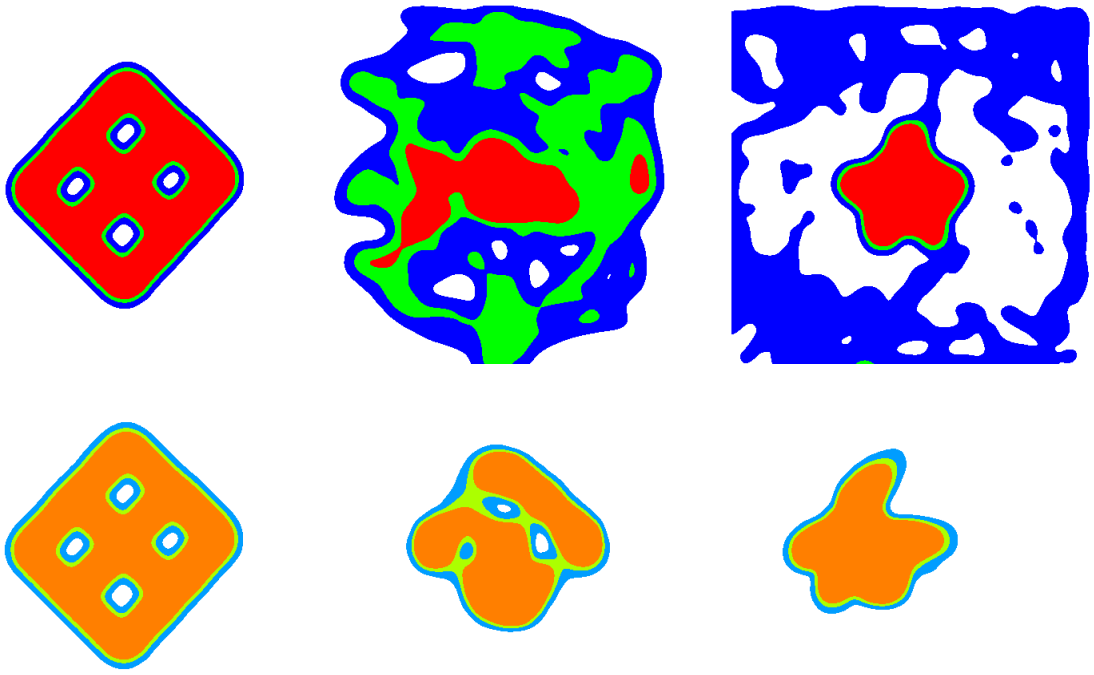

Figure 12: A genus 4 object morphing into star. (Top) Converged optimization without using a penalty grid. (Bottom) Converged optimization using a penalty grid. Note that the object with the penalty grid wasn't able to create the star shape as accurately. This is unexpected result actually makes sense, since taking away mass ejections can inhibit the objects ability to morph.

$$L(\mathcal{C}) = \sum_{i=0}^{\mathbf{M}-1} \frac{1}{2}\alpha_i \|m_i^n - m_{iu}^n\|^2 \tag{72}$$

Sometimes mass ejection is required for the object to match the target shape, even when the optimization is run to convergence. This effect can be seen in Fig. 12.

9 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.

9.1 Improving The Optimization Method

We are currently using a simple gradient descent method to minimize our loss functions. However, we recognize that there are more sophisticated methods out there that are worth experimenting with, such as adaptive gradient descent methods and momentum based methods.

9.2 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.

9.3 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.

9.4 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.

9.5 Acknowledgements

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

References

J. Barbič and J. Popović. Real-time control of physically based simulations using gentle forces. *ACM Trans. Graph.*, 27(5):163:1–163:10, Dec. 2008. ISSN 0730-0301. doi: 10.1145/1409060.1409116. URL http://doi.acm.org/10.1145/1409060.1409116.

S. Coros, S. Martin, B. Thomaszewski, C. Schumacher, R. Sumner, and M. Gross. Deformable objects alive! *ACM Trans. Graph.*, 31(4):69:1–69:9, July 2012. ISSN 0730-0301. doi: 10.1145/2185520.2185565. URL http://doi.acm.org/10.1145/2185520.2185565.

Y. Hu, Y. Fang, Z. Ge, Z. Qu, Y. Zhu, A. Pradhana, and C. Jiang. A moving least squares material point method with displacement discontinuity and two-way rigid body coupling. *ACM Trans. Graph.*, 37(4):150:1–150:14, July 2018a. ISSN 0730-0301. doi: 10.1145/3197517.3201293. URL http://doi.acm.org/10.1145/3197517.3201293.

Y. Hu, J. Liu, A. Spielberg, J. B. Tenenbaum, W. T. Freeman, J. Wu, D. Rus, and W. Matusik. Chainqueen: A real-time differentiable physical simulator for soft robotics. *CoRR*, abs/1810.01054, 2018b. URL http://arxiv.org/abs/1810.01054.

C. Jiang, C. Schroeder, A. Selle, J. Teran, and A. Stomakhin. The affine particle-in-cell method. *ACM Trans. Graph.*, 34(4), July 2015. ISSN 0730-0301. doi: 10.1145/2766996. URL https://doi.org/10.1145/2766996.

C. Jiang, C. Schroeder, J. Teran, A. Stomakhin, and A. Selle. The material point method for simulating continuum materials. In *ACM SIGGRAPH 2016 Courses*, SIGGRAPH '16, pages 24:1–24:52, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4289-6. doi: 10.1145/2897826.2927348. URL http://doi.acm.org/10.1145/2897826.2927348.

A. McNamara, A. Treuille, Z. Popović, and J. Stam. Fluid control using the adjoint method. *ACM Trans. Graph.*, 23(3):449–456, Aug. 2004. ISSN 0730-0301. doi: 10.1145/1015706.1015744. URL http://doi.acm.org/10.1145/1015706.1015744.

E. Sifakis and J. Barbic. Fem simulation of 3d deformable solids: A practitioner's guide to theory, discretization and model reduction. In *ACM SIGGRAPH 2012 Courses*, SIG-

GRAPH '12, pages 20:1–20:50, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1678-1. doi: 10.1145/2343483.2343501. URL http://doi.acm.org/10.1145/2343483.2343501.

A. Stomakhin, R. Howes, C. Schroeder, and J. M. Teran. Energetically consistent invertible elasticity. In *Proceedings of the 11th ACM SIGGRAPH / Eurographics Conference on Computer Animation*, EUROSCA'12, pages 25–32, Aire-la-Ville, Switzerland, Switzerland, 2012. Eurographics Association. ISBN 978-3-905674-37-8. doi: 10.2312/SCA/SCA12/025-032. URL http://dx.doi.org/10.2312/SCA/SCA12/025-032.

A. Stomakhin, C. Schroeder, L. Chai, J. Teran, and A. Selle. A material point method for snow simulation. *ACM Trans. Graph.*, 32(4):102:1–102:10, July 2013. ISSN 0730-0301. doi: 10.1145/2461912.2461948. URL http://doi.acm.org/10.1145/2461912.2461948.

D. Sulsky, S.-J. Zhou, and H. L. Schreyer. Application of a particle-in-cell method to solid mechanics. *Computer Physics Communications*, 87(1):236 – 252, 1995. ISSN 0010-4655. doi: https://doi.org/10.1016/0010-4655(94)00170-7. URL http://www.sciencedirect.com/science/article/pii/0010465594001707. Particle Simulation Methods.

S. Wang, M. Ding, T. F. Gast, L. Zhu, S. Gagniere, C. Jiang, and J. M. Teran. Simulation and visualization of ductile fracture with the material point method. *Proc. ACM Comput. Graph. Interact. Tech.*, 2(2):18:1–18:20, July 2019. ISSN 2577-6193. doi: 10.1145/3340259. URL http://doi.acm.org/10.1145/3340259.

J. Wolper, Y. Fang, M. Li, J. Lu, M. Gao, and C. Jiang. Cd-mpm: Continuum damage material point methods for dynamic fracture animation. *ACM Trans. Graph.*, 38 (4):119:1–119:15, July 2019. ISSN 0730-0301. doi: 10.1145/3306346.3322949. URL http://doi.acm.org/10.1145/3306346.3322949.