UCLA

UCLA Electronic Theses and Dissertations

Title

A Hopf-Lax formulation of the eikonal equation for parallel redistancing and oblique projection

Permalink

https://escholarship.org/uc/item/04f9942g

Author

Royston, Michael Wayne

Publication Date

2017

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

UNIVERSITY OF CALIFORNIA

Los Angeles

A Hopf-Lax formulation of the eikonal equation for parallel redistancing and oblique projection

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Mathematics

by

Michael Wayne Royston

2017

© Copyright by

Michael Wayne Royston

2017

ABSTRACT OF THE DISSERTATION

A Hopf-Lax formulation of the eikonal equation for parallel redistancing
and oblique projection

by

Michael Wayne Royston
Doctor of Philosophy in Mathematics
University of California, Los Angeles, 2017
Professor Joseph M. Teran, Chair

The level set method is one of the most useful algorithms for scientific computing problems that require computations over complex geometric domains. By representing geometry in terms of implicit functions defined over regular Cartesian grids, level set methods automatically capture a wide range of dynamic shapes and topology changes. They have gained wide adoption for problems in fluid dynamics, image processing, chip manufacturing, combustion, computer animation and many more. Dynamic changes in the geometry are automatically resolved by advecting the implicit function throughout a flow domain. However, it is typically necessary for the function describing the level set to have some important properties that the advection process does not preserve. One such property is that the gradient of the implicit function should have unit norm. Given this requirement, much research has been done on replacing an implicit function with a new function that preserves the old zero isocontour while satisfying the unit norm gradient constraint. This process is called redistancing and its solution satisfies the eikonal equation.

Many methods currently exist to solve this problem. Fast marching [1, 2] and fast sweeping [3] are the most popular redistancing methods due to their efficiency and relative simplicity. However, these methods require propagation of information from the zero-isocontour outwards, and this data dependence complicates efficient implementation on today's multiprocessor hardware. Recently an interesting alternative view has been developed that utilizes the Hopf-Lax formulation of the solution to the eikonal equation [4, 5]. In this approach, the signed distance at an arbitrary point is obtained without the need of distance information from neighboring points. We extend the work of Lee et al.

[4] to redistance functions defined via interpolation over a regular grid. The grid-based definition is essential for practical application in level set methods. We demonstrate the effectiveness of our approach with GPU parallelism on a number of representative examples.

In addition, we show how our method can be modified to solve the problem of oblique projection. This problem arises in the simulation of elastoplastic materials. An elastoplastic material has a region of allowed (or feasible) stress, and during simulation the stresses can leave the region. A projection is necessary to return the stress back to the feasible region. As we will show, this projection is not always a right angle projection. The shape of the allowed stress region changes depending on the material being simulated. While some closed form solutions exist for determining the projection, in general this problem is difficult. We provide a parallel solution for finding the oblique projection needed for an arbitrary region of allowed stress that generalizes the Hopf-Lax approach to redistancing.

The dissertation of Michael Wayne Royston is approved.

Luminita Vese

Stanley Osher

Wotao Yin

Joseph M. Teran, Committee Chair

University of California, Los Angeles

2017

TABLE OF CONTENTS

1 Redistancing **1**

1.1 Introduction 1

1.2 Mathematical properties of the eikonal equation 2

1.2.1 Approximating the eikonal equation 6

1.3 Fast marching 9

1.4 Fast sweeping 10

1.5 Higher order methods 14

1.6 Parallel fast marching and fast sweeping 15

1.7 Hopf-Lax formula 19

1.8 Method 21

1.8.1 Secant method for roots of $\tilde{\phi}(\mathbf{x_i}, \hat{t}) = 0$ 22

1.8.2 Hopf-Lax formula for $\tilde{\phi}(\mathbf{x_i}, t^k)$ 23

1.8.3 Grid interpolation order 28

1.8.4 Computing in a narrow band 29

1.8.5 Computing geometric quantities 31

1.8.6 Preventing volume loss 32

1.9 Results 32

1.9.1 Scaling 35

2 Continuum Mechanics **39**

2.1 Introduction 39

2.2 Elastoplastic deformations 41

2.3 Conservation of mass 41

2.4 Conservation of momentum 42

2.5 Stress and strain 43

2.5.1 Deformation gradient 43

2.5.2 Strain 44

2.5.3 Stress 45

2.6 Plasticity 48

2.6.1 Plastic flow 49

2.7 Rates of plastic flow 49

2.7.1 Energy dissipation and stress/rate pairs 52

2.7.2 Isotropy 53

2.7.3 Plastic dissipation rate without hardening 54

2.8 Associative 55

2.8.1 Yield surface and plastic flow 56

2.8.2 Isoptropic yield surface 56

2.9 Hencky strain 57

2.9.1 Yield surface and plastic rate of change 57

2.10 Flow direction 58

2.11 Return mapping 59

2.12 Modifications to algorithm 61

2.13 Results 61

3 Material Point Method 77

3.1 Overview 77

3.1.1 Material state 78

3.1.2 Transfer weights 79

3.2 Transfer to grid 79

3.3 Grid update 81

3.3.1 Explicit integration 81

3.3.2 Implicit integration 81

3.3.3 Collision and friction 82

3.4 Transfer to particles 83

3.5 Update particle state 84

4 Appendix 85

4.1 Appendix: eigen decomposition differentials 85

4.1.1 Symmetric tensors 87

4.1.2 Scalar functions of symmetric tensors 87

References 88

LIST OF FIGURES

1.1 Here we see two situations where characteristics can intersect. In both images our surface is represented by the black line, and the characteristics by the colored (and gray) lines. In the left image a shock is formed when multiple characteristics from different starting points intersect. On the right a rarefaction fan is formed when multiple characteristics are all formed from the same point(due to the boundary being non smooth). 3

1.2 Consider a 1d grid representation. The blue dots represent ϕ_i, the red dots represent ϕ_{i-1} and the green dots represent ϕ_{i+1}. In all figures we assume that $\phi_i, \phi_{i+1}, \phi_{i-1} > 0$ The top row all represent valid representations that satisfy Equation (1.24). The bottom two figures, however, do not. The left figure represents a case where $D^-_{ij}\phi = 1$ however $-D^+_{ij}\phi = 2$. In order for this figure to be valid, the blue dot would need the same value as the red dot. The bottom right figure represents a case where $D^-_{ij}\phi = -D^+_{ij}\phi = -1$ 8

1.3 Consider a level set with zero iso-countour given by the ellipse. The black nodes have a value $\phi_{ij} = 0$ given by step 1. For our purposes we are only considering nodes outside of Γ where $\phi_0 > 0$. The blue nodes are the original nodes that need to be defined before the algorithm starts in step 2. In most cases these values are found by using a ”linear” approximation near the boundary. The orange nodes are those found by calculating all approximations in neighbor nodes in step 3. The circled orange node is the node with the smallest value ϕ_{ij} found in step 4. The green nodes are those that were calculated using the new value of the blue circled node in steps 5 and 6. These steps would then continue until every grid node of consideration was blue. . 11

1.4 Consider a level set with zero iso-countour given by the ellipse. Each color represents a different sweep direction. Each sweep will first follow the horizontal axis, then the vertical axis. 13

1.5 Consider a level set with zero iso-countour given by the ellipse. Each color represents a different sweep direction. The sweep directions are now aligned along the diagonals. Nodes connected by a colored line are independent in a given sweep and are updated in parallel before moving to the next row. . . . 17

1.6 Consider a level set with zero iso-countour given by the ellipse. A grid is broken up into 4 subdomains. The red nodes represent ”ghost nodes” that when updated will be used to transfer information into a separate subdomain. 18

1.7 The two vertical lines are the boundary of minimization. Grid node x_i = 1.8 is in the middle of the region, and also the starting guess for projected gradient descent. The sequence of points leading off to the right represent the subsequent steps of gradient descent. These points converge to the incorrect argmin $x = 2.5$. The correct solution is at $x = 0.4$. In order to converge to this point, the initial guess would have to be less than 1.25 25

1.8 The figure illustrates representative random initial guesses used in solving for $\tilde{\phi}(\mathbf{x}_i, t^k)$. In addition, we use an initial guess equal to the minimizer computed in the previous secant iteration shown in magenta. 27

1.9 The plots above are the points $(\tilde{\phi}(\mathbf{x_i}, t^k), t^k)$ found when running our algorithm with different choices of random guess and gradient descent iterations on circle initial data. The left most plot was run with 100 random guesses, and 1 gradient descent iteration. The middle plot was run with 1 random guess, and 100 gradient descent iterations. The right plot was run with 1 random guess and 5 gradient descent iterations. Note that in all cases, the correct root was found. 28

1.10 For both images Bi-Cubic or Bi-Linear interpolation were used to represent J. Both started with the initial data as a pyramid with a corner in the center of a cell. Each was then reinitialized approximately 50 times and the changing of the zero-isocounter was shown as above. For cubic on the left, instability is introduced by trying to fit the cubic interpolant to the data. While it handles smooth data better than linear, at sharp corners the data over fits and becomes unstable. Linear on the right suffers from no such instability, however because the sharp corner is lost with interpolation, it suffers from volume loss over time. 29

1.11 In this image, the red dot in the center is $\mathbf{x_i}$, the solid red line represents the ball of radius t_k and the dotted line represents the ball of radius t_{k-1}. The magenta point was the approximate argmin $\mathbf{y}_{k-1}$ of ϕ^0 over the ball of radius t_{k-1}. Since it is unlikely for the minimizer to be inside of t_{k-1} we use coarse (random) grid initial guesses in the interior. However, since it is possible that expanding t will move the minimizer to a different location we take a large number of initial guesses along the boundary of t_k 30

1.12 A coarse grid 8X smaller than the fine grid was solved for initially. Using those values the fine grid was only solved on cells where the distance to the boundary could be less than 0.1, represented as the solid areas of the left image. In the right image those coarse areas are defined from bilinear interpolation. This coarse/banding approach provided approximately a 2.5 times increase in performance. 31

1.13 Scaled circle: the initial data is $\phi^0 = exp(x) * (.125 - (.5 - x)^2 + (.5 - y)^2)$. The zero level set is a circle of radius .25 centered around (.5,.5) 33

1.14 Square: the initial data is $\phi^0 = \texttt{min}(.25 - |x - .5|, .25 - |y - .5|)$. The zero level is a square with side length .5 centered around (.5,.5) 34

1.15 Union of circles: the initial data is $\phi^0 = \texttt{max}((.25 - \|(.3, .5) - (x, y)\|_2), (.25 - \|(.7, .5) - (x, y)\|_2))$. The zero level set is a union of two circles both with radius .25, one centered at (.3,.5) and the other centered at (.7,.5) 34

1.16 Many local minima: the objective function is $\phi^0 = \texttt{sin}(4 * \pi * x) * \texttt{sin}(4 * \pi * y) - .01$. The zero level set is a group of rounded squares that almost touch. 35

1.17 Parallel speed up is plotted both with and without including the cost of updating memory on the GPU. With 64 GPUs the memory update can take up to 33% of the runtime. However without the memory update (I.e. if the data is already on the GPU) the method scales simply. 37

1.18 Practical application: vortex advection test at t = 0,1,2,3,4,5 38

2.1 There exists a map ϕ that transforms the material state (left) into the spatial or current state (right). 41

2.2 Drucker Prager with cohesion of 1.8 using an associated flow rule. Due to the large amount of cohesion allowed, the material is bouncy and tends to break into large chunks. 63

2.3 Drucker Prager with cohesion of .36 Associated. We provide one more example of wet sand. Again we note how the sand breaks apart in chunks, but is still able to flow through the hourglass. 64

2.4 Drucker Prager with cohesion of .18 using an associated flow rule. Here we start to see more of what we might expect with wet sand, however due to our return mapping being non volume preserving, all three simulations tend to see a large amount of volume growth. 65

2.5 Drucker Prager with cohesion of .018 using an associated flow rule. This small amount of cohesion simulates nearly dry sand. The sand flows as one might expect, however again there is a large amount of volume gain. 66

2.6 Drucker Prager with cohesion of .18 using a non Associated flow rule. By using a volume preserving flow rule we can see a closer expectation of what our wet sand might actually look like. The sand sticks together, but still piles nicely. 67

2.7 Drucker Prager with cohesion of .018 non Associated. Again we simulate nearly dry sand with a volume preserving flow rule. There is no volume growth, and in the wedge example we can even see the two waves of sand colliding beneath the wedge 68

2.8 With the von Mises 2d projection we don't really see what we would expect from a simulation of metals. Instead we see more of a Jello like material that jiggles and is able to be squeezed. The hourglass simulation reached a steady state by 2.5 seconds, so we show you the results after 1.5 and 2.5 seconds. . 69

2.9 By shrinking the radius of the ellipse given in the 2d projection of the von Mises yield criterion, more of the stresses get transferred into the plasticity. This allows our material to be more willing to shrink and squeeze. During the hourglass simulation for a brief time the material separates before reuniting at the end. Again the hourglass simulation is shown at 1.25 and 2.5 seconds. 70

2.10 As before by shrinking the ellipse even further, more stress gets pushed into plasticity. This causes the falling materials to shrink more than the collapsing column, as larger stresses are shown in the falling materials. The small stresses experienced in the collapsing column are allowed to remain elastic. 71

2.11 Here our yield surface is a lemniscape with ”radius” 1. The most interesting simulation ends up being the column collapse. Due to shear forces not being elastic, the entire material expands sideways as the plastic forces compensate before settling in. In the wedge drop, the material hits the ground and expands sideways as the shear force is not allowed to be elastic. 72

2.12 Again as seen in the von Mises examples, by shrinking the size of the yield surface, large stresses are moved into the plasticity and result in large volume shrinkages. 73

2.13 Here we have a material that allows more compression than expansion. As such the material tends to stick together through the forces we applied to it. Worth noting is that these are some of the bouncier examples, probably due to the larger yield surface. 74

2.14 When both compression and tension are given a large area to work with, we end up with a material that is very bouncy and does not break up easily. A very small portion of it breaks off in the hourglass example. The impact with the floor is enough to break up the material further in the wedge example. . 75

2.15 In this material the yield surface allows for more compression than tension. This seems to cause the material to actually bounce off of the surfaces it hits. In addition it breaks up into larger chunks. 76

LIST OF TABLES

1.1 Timing and error at different grid resolutions using a square as our zero level set. The average L2 error is calculated by calculating the L2 distance between the computed answer and the analytic solution 36

1.2 Timing in ms showing scaling in number of GPUs. The first two columns show the time it takes to run our problem when we include the timing cost of transferring the grid data to the GPU (approximatly 1.2ms), while the last two columns show the scaling without the cost of transferring data 36

CHAPTER 1

Redistancing

1.1 Introduction

The level set method [6] is a powerful technique used in a large variety of problems in computational fluid dynamics, minimal surfaces and image processing [7]. In general these methods are concerned with the transport evolution $\frac{\partial \phi}{\partial t} + \mathbf{v} \cdot \nabla \phi$ of a level set function $\phi : \mathbb{R}^n \rightarrow \mathbb{R}$ in velocity field $\mathbf{v}$. While the essential property of ϕ is typically the location of its zero iso-contour ($\Gamma = \{\mathbf{x} \in \mathbb{R}^n | \phi(\mathbf{x}) = 0\}$), in practice many applications additionally require that it be a signed distance function ($|\nabla \phi| = 1$). This property will not generally be preserved by the transport evolution, but it can be recovered without modifying the location of the zero isocontour. This process is commonly referred to as redistancing or reinitialization [8, 9, 10].

We describe the redistancing problem in terms of the signed distance function ϕ that is obtained from an arbitrary non-signed-distance level set function ϕ^0 (while preserving the zero isocontour of ϕ^0). Mathematically, the process obeys the eikonal equation as

$$\begin{aligned} |\nabla \phi(\mathbf{x})|_2^2 &= 1 \qquad (1.1) \\ \operatorname{sgn}(\phi(\mathbf{x})) &= \operatorname{sgn}(\phi^0(\mathbf{x})). \end{aligned}$$

There is extensive previous work related to the solution of (1.1). The most commonly used methods are the fast marching method (FMM) [1, 2] and the fast sweeping method (FSM) [3].

In this chapter we will first discuss the mathematical properties of the eikonal equation that are important to consider when deriving a discrete approximation. We will then discuss some of the methods that currently exist to solve the eikonal equation along with their parallel versions. We will then derive our method using a Hopf-Lax formulation of the time dependent eikonal equation, and show the results of running our method in parallel on a modern consumer GPU.

1.2 Mathematical properties of the eikonal equation

One way to find a solution to the eikonal equation is to use the method of characteristics. We can rewrite Equation (1.1) as,

$$\begin{aligned} F(\mathbf{x}, \phi, \mathbf{p}) &= |\mathbf{p}|_2^2 - 1 = 0 \qquad (1.2)\\ \phi(\mathbf{x}) &= 0 \;\forall\; \mathbf{x} \in \Gamma \\ \mathbf{p} &= \nabla\phi(\mathbf{x}) \end{aligned}$$

Now suppose we have a curve in $\mathbb{R}^{2n+1}$ represented by $(\mathbf{x}(s), z(s), \mathbf{p}(s))$ and furthermore assume that $z(s) = \phi(\mathbf{x}(s))$ is a solution to Equation (1.2) then following the method of characteristics from Evans [11] we get

$$\begin{aligned} \dot{\mathbf{p}}(s) &= \lambda(-F_{\mathbf{x}}(\mathbf{x}(s), z(s), \mathbf{p}(s)) - F_z(\mathbf{x}(s), z(s), \mathbf{p}(s))\mathbf{p}(s)) \\ \dot{z}(s) &= \lambda(-F_{\mathbf{p}}(\mathbf{x}(s), z(s), \mathbf{p}(s)) \cdot \mathbf{p}(s)) \\ \dot{\mathbf{x}}(s) &= \lambda(-F_{\mathbf{p}}(\mathbf{x}(s), z(s), \mathbf{p}(s))) \end{aligned}$$

Now due to the simplicity of our F we get that $F_{\mathbf{x}} = 0$, $F_z = 0$ and $F_{\mathbf{p}} = 2\mathbf{p}$. Thus

$$\begin{aligned} \dot{\mathbf{p}}(s) &= 0 \\ \dot{z}(s) &= 2\lambda|\mathbf{p}|_2^2 \\ \dot{\mathbf{x}}(s) &= 2\lambda\mathbf{p} \end{aligned}$$

Now in addition we know that $\frac{d}{ds}\nabla\phi(\mathbf{x}(s)) = 0$ from $\dot{\mathbf{p}}(s) = 0$. This means that along this curve the value of $\nabla\phi$ does not change. Therefore we know that $\mathbf{x}(s)$ follows a straight line, in addition if we choose $\lambda = -\frac{1}{2}$ we get

$$\mathbf{x}(s) = \nabla\phi(\mathbf{x}(0))s + \mathbf{x}(0) \qquad (1.3)$$

This can be seen as saying that the characteristics of ϕ move in straight lines. In addition since we know that $|\mathbf{p}|_2^2 = 1$ we get

$$z(s) = s + z(0) \qquad (1.4)$$

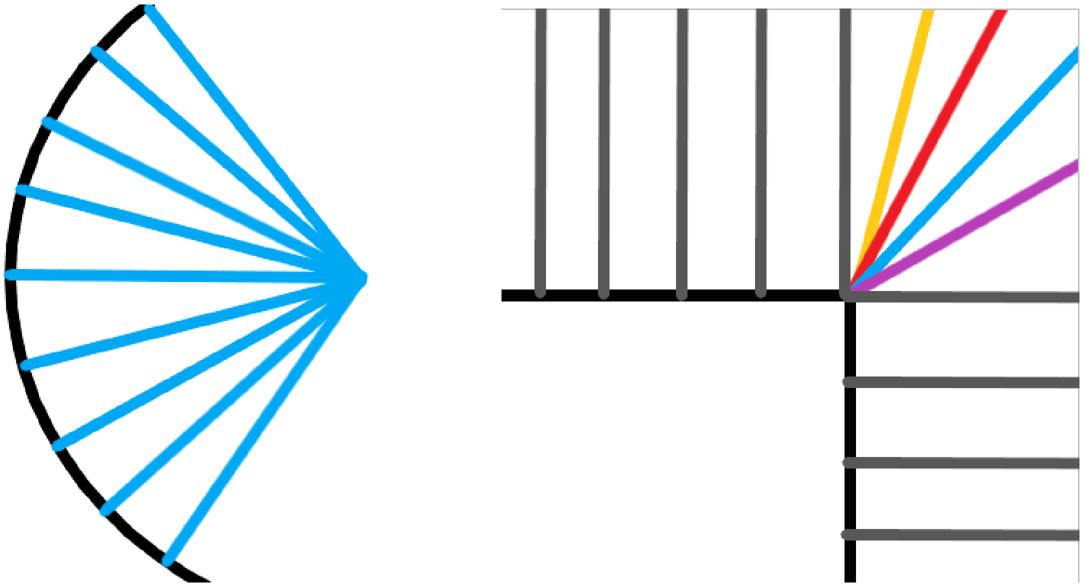

Figure 1.1: Here we see two situations where characteristics can intersect. In both images our surface is represented by the black line, and the characteristics by the colored (and gray) lines. In the left image a shock is formed when multiple characteristics from different starting points intersect. On the right a rarefaction fan is formed when multiple characteristics are all formed from the same point(due to the boundary being non smooth).

We note that these characteristics are only valid for the supposed curve in $\mathbb{R}^{2n+1}$. These curves do not have to be infinite, in fact when two characteristic curves coincide they will both terminate. This can happen in two distinct situations both shown in Figure 1.1. The first is considered a shock. This is when two different points $\mathbf{x}, \mathbf{y} \in \Gamma$ exist with s_1, s_2 such that

$$\mathbf{x}(s_1) = \mathbf{y}(s_2) \tag{1.5}$$

This will occur, for example, on the inside of a convex surface. The second situation is called a rarefaction fan. This occurs when Γ is non-smooth. At any point $\mathbf{x} \in \Gamma$ where the boundary is non-smooth $\nabla\phi$ will not exist. Instead there will be a set of sub differentials $\partial\phi$. These sub differentials will define the directions of characteristics. Essentially this means that if we have $\mathbf{x}_1(s)$ and $\mathbf{x}_2(s)$ defined from different sub differentials, with $\mathbf{x}_1(0) - \mathbf{x}_2(0) = \mathbf{0}$ we can still have $\mathbf{x}_1(s) \neq \mathbf{x}_2(t)\ \forall t \neq 0$. We note that a solution ϕ will be continuous everywhere, and will have a continuous derivative everywhere except at shocks.

With this choice of λ we can see that a solution $z(s)$ will represent a signed distance function along the curve as long as it does not hit a shock. Suppose $\mathbf{x}(0) \in \Gamma$, then $z(0) = 0$ and

$$\|\mathbf{x}(s) - \mathbf{x}(0)\| = \|s\nabla\phi(\mathbf{x}(0))\| = s = z(s) \tag{1.6}$$

where we used that $\|\nabla\phi(\mathbf{x}(0))\| = 1$. Since the only time that more than one point on the boundary can have a characteristic curve intersect a given point is at a shock, we can say that $\mathbf{x}(0)$ is the closest point on the boundary to $\mathbf{x}(s)$

Now the eikonal equation is not the only PDE that can be used to solve the redistancing problem. We note that the current formulation given in Equation (1.1) is only spatially variant. There are two similar variations on the eikonal equation that also include a temporal term. The first variation is

$$\tilde{\phi}_t(\mathbf{x}, t) + |\nabla\tilde{\phi}| = 1 \tag{1.7}$$

This PDE has the nice property that as time $t \to \infty$ the solution $\tilde{\phi}$ reaches a steady state where $\tilde{\phi}(\mathbf{x}, \infty) = \phi(\mathbf{x})$. Essentially after time T all points within T distance of the boundary Γ will have reached the correct solution [7], i.e.

$$\tilde{\phi}(\mathbf{x},T) = \phi(\mathbf{x}) \;\forall\; \mathbf{x} \;\texttt{s.t.}\; \phi(\mathbf{x}) \le T \tag{1.8}$$

The second variation we will derive as it is used further in the paper. This second variation is very similar to Equation (1.7) however we set the right hand side equal to 0. This variation is given in the work of Lee et al. [4]

$$\begin{aligned} |\nabla\tilde{\phi}| + \tilde{\phi}_t = 0 \\ \tilde{\phi}(\mathbf{x},0) = 0 \;\forall\; \mathbf{x} \in \Gamma \end{aligned} \tag{1.9}$$

Equation (1.1) and Equation (1.9) are not immediately seen as similar. However, if we follow the work of Osher [12] we can see that $\tilde{\phi}(\mathbf{x},t) = 0 \implies \phi(\mathbf{x}) = t$, i.e. the zero level set of $\tilde{\phi}(\mathbf{x},t)$ flow outward along its (the zero level set) normals. The speed at which it grows is such that after s units of time, a point on the zero level set will have moved s distance away from its previous location (assuming it has not hit a shock).

To begin, assume we have a PDE written in the following form

$$F(\mathbf{x},\phi,\nabla\phi) = 0 \tag{1.10}$$

$$\tag{1.11}$$

with Dirichlet boundary data given on Γ a closed set in $\mathbb{R}^2$ with an interior Ω and an exterior $\mathbb{R}^n/\Omega$

$$\phi(\mathbf{x}) = 0 \;\forall\; \mathbf{x} \in \Gamma$$

our goal is to find a new formulation of the PDE that satisfies the Hamilton-Jacobi equation

$$\begin{aligned} \frac{\partial}{\partial t}\tilde{\phi}(\mathbf{x},t) + H(\mathbf{x},\nabla\tilde{\phi}) = 0 \;\texttt{in}\; \mathbb{R}^n \times (0,\infty) \\ \tilde{\phi} = \phi_0 \;\texttt{on}\; \mathbb{R}^n \times (t=0) \end{aligned} \tag{1.12}$$

If we wish to reproduce the eikonal equation in this form, we can start with

$$F(\mathbf{x},\phi,\nabla\phi) = |\nabla\phi| - 1 = 0 \tag{1.13}$$

where

$$\phi(\mathbf{x}) = 0 \;\texttt{for}\; \mathbf{x} \in \Gamma \tag{1.14}$$

Now assume we have a solution to the eikonal equation (1.1) $\phi(\mathbf{x}) = t$. We wish to construct a function $\tilde{\phi}(\mathbf{x}, t), t \geq 0$ such that

$$\tilde{\phi}(\mathbf{x}, t) = 0 \implies \phi(\mathbf{x}) = t \tag{1.15}$$

Any such constructions will satisfy

$$\frac{\partial}{\partial x_i}\tilde{\phi}(\mathbf{x}, \phi(\mathbf{x})) = 0 = \tilde{\phi}_{x_i} + \tilde{\phi}_t \phi_{x_i} \tag{1.16}$$

for all the partial derivatives $\frac{\partial}{\partial x_i}$. This implies

$$\phi_{x_i} = -\frac{\tilde{\phi}_{x_i}}{\tilde{\phi}_t} \tag{1.17}$$

$$F(\mathbf{x}, t, \frac{\nabla\tilde{\phi}}{\tilde{\phi}_t}) = 0 \tag{1.18}$$

$$|\frac{\nabla\tilde{\phi}}{\tilde{\phi}_t}| - 1 = 0 \tag{1.19}$$

$$|\nabla\tilde{\phi}| \pm |\tilde{\phi}_t| = 0 \tag{1.20}$$

If we assume that $\tilde{\phi}$ is monotonic in t then the choice of plus or minus determines which direction the interface is traveling. As such if we have $\tilde{\phi}$ a solution to

$$|\nabla\tilde{\phi}| + \tilde{\phi}_t = 0 \tag{1.21}$$

$$\tilde{\phi}(\mathbf{x}, 0) = \phi_0 \tag{1.22}$$

we get $\tilde{\phi}(\mathbf{x}, t) = 0 \implies \phi(\mathbf{x}) = t$ giving us a time dependent formulation of the eikonal equation in a Hamilton-Jacobi formulation. We note that the choice of ϕ_0 is important. As we noted at the start, the zero iso-contour follows its outward normal. The outward normal is defined as the direction followed by the gradient. As such, if we have $\phi_0(\mathbf{x}) = 0$ then the direction the level set will travel at $\mathbf{x}$ is determined to be $\nabla\phi_0(\mathbf{x})$

1.2.1 Approximating the eikonal equation

If we wish to use a numerical solution of the eikonal equation (1.1), we first need to discretize it. We start by assuming we are working on a regular grid in $\mathbb{R}^2$. Assume we want a grid $\mathbf{G}$ of size $n \times n$ with width between grid nodes h and whose bottom left corner is defined as $\mathbf{x}_0$. Then a grid node $\mathbf{x}_{ij} \in \mathbf{G};\ i < n,\ j < n$ represents

$$\mathbf{x}_{ij} = \mathbf{x}_0 + (ih, jh) \tag{1.23}$$

A first order approximation to the eikonal equation on this regular grid is

$$\texttt{max}(D_{ij}^{-x}\phi, -D_{ij}^{+x}\phi, 0)^2 + \texttt{max}(D_{ij}^{-y}\phi, -D_{ij}^{+y}\phi, 0)^2 = 1 \tag{1.24}$$

where

$$\begin{aligned}\phi_{ij} &= \phi(\mathbf{x}_{ij})\\ \phi_x(\mathbf{x}_{ij}) \approx D_{ij}^{\pm x}\phi &= \frac{\phi_{i\pm1,j} - \phi_{ij}}{\pm h}\\ \phi_y(\mathbf{x}_{ij}) \approx D_{ij}^{\pm y}\phi &= \frac{\phi_{i,j\pm1} - \phi_{ij}}{\pm h}\end{aligned}$$

Where Equation (1.24) needs to be true at every point $\mathbf{x}_{ij}$. This gives us a series of coupled equations that all need to be solved simultaneously. The use of the max in Equation (1.24) can be seen to be choosing which side of the discretization D_{ij}^- or D_{ij}^+ the information is flowing from. Consider Figure 1.2, we know from the solution given by the method of characteristics that information always propagates away from the boundary, thus that information propagates from nodes with smaller values to nodes with larger values. The choice of 0 in the max corresponds to when the sign of both D_{ij}^- and D_{ij}^+ is negative, which means information should be propagating away from ϕ_{ij} in both directions which should not be possible. We note that in dimensions > 1 this simply means that at the given point this direction does not contribute to the propagation(e.g. a shock). In addition the use of the maximum guarantees there will not be large discontinuities in the solution where $\nabla\phi > 1$ in the discretization.

Since the eikonal equation flows information away from the boundary, we can simplify the Equation (1.24) to the following

$$\begin{aligned}\phi_H &= \texttt{min}(\phi_{i-1,j}, \phi_{i+1,j})\\ \phi_V &= \texttt{min}(\phi_{i,j-1}, \phi_{i,j+1})\\ 1 &= \left(\frac{\phi_{ij} - \phi_H}{h}\right)^2 + \left(\frac{\phi_{ij} - \phi_V}{h}\right)^2\\ \phi_{ij} &= \frac{\phi_H + \phi_V}{2} + \frac{1}{2}\sqrt{(\phi_H + \phi_V)^2 - 2(\phi_H^2 + \phi_V^2 - h^2)}\end{aligned} \tag{1.25}$$

This approximation requires $|\phi_H - \phi_V| < h$. This will be the case when the solution is smooth around a grid node. The solution will not be smooth if a shock has formed in the solution. In this case the solution will require one of the partial derivatives to be zero.

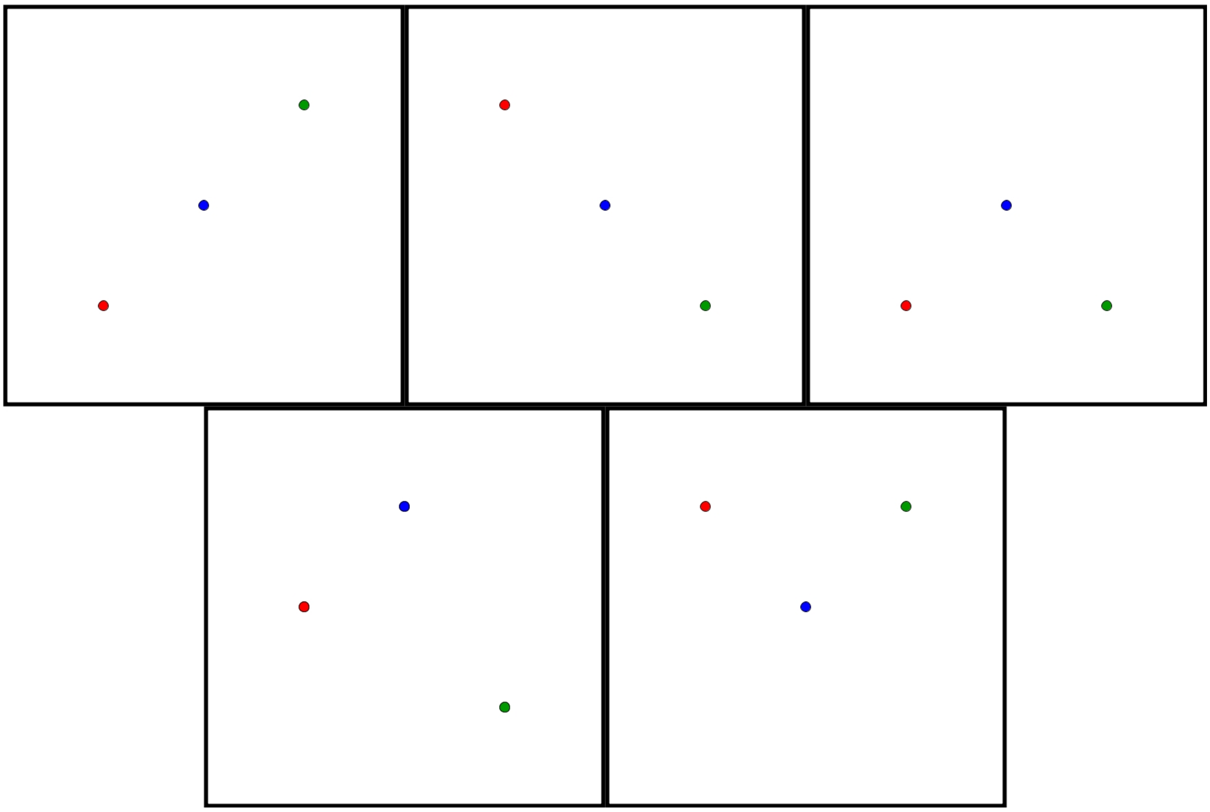

Figure 1.2: Consider a 1d grid representation. The blue dots represent ϕ_i, the red dots represent ϕ_{i-1} and the green dots represent ϕ_{i+1}. In all figures we assume that $\phi_i, \phi_{i+1}, \phi_{i-1} > 0$ The top row all represent valid representations that satisfy Equation (1.24). The bottom two figures, however, do not. The left figure represents a case where $D^-_{ij}\phi = 1$ however $-D^+_{ij}\phi = 2$. In order for this figure to be valid, the blue dot would need the same value as the red dot. The bottom right figure represents a case where $D^-_{ij}\phi = -D^+_{ij}\phi = -1$

Thus we will simply take the shortest distance from a neighboring cell and use that as our approximation.

$$\phi_{ij} = \mathtt{min}(\phi_H, \phi_V) + h$$

The difference in methods that use this discretization is simply down to the order in which grid nodes are updated. While the final solution ϕ_{ij} must satisfy Equation (1.24) everywhere the methods to produce ϕ_{ij} incrementally update the grid nodes until the final solution is reached. Two of the most common methods used to redistance are the fast marching method and the fast sweeping method.

1.3 Fast marching

First proposed by Tsitsiklis [1] using optimal control, the fast marching method was independently developed by Sethian in [2] based on upwind difference schemes. It is similar to Dijkstra's method [13] for finding the shortest path between nodes in a graph. The fast marching method uses upwind difference stencils to create a discrete data propagation consistent with the characteristics of the eikonal equation. Sorting is used to determine a non-iterative update order that minimizes the number of times that a point is utilized to create a strictly increasing (or decreasing) propagation. The operation count is $\mathrm{O}(N \log(N))$ where N is the number of grid points and the $\log(N)$ term is a consequence of the sorting.

In fast marching, each node in the grid is conisdered to be in one of three states, *far*(not yet visited),*considered*(node visited and assigned tentative value), and *accepted*(node visited and assigned final value). The method follows these steps.

1. Assign every node $\mathbf{x}_{ij}$ the value $\phi_{ij} = \infty$ and label *far*

2. For nodes $\mathbf{x}_{ij} \in \Gamma$ (or nodes close to the boundary) set $\phi_{ij} = 0$ and label *accepted* (or set ϕ_{ij} to an approximation of distance at cells crossed by Γ)

3. $\forall$ $\mathbf{x}_{ij} \in$ *accepted* use an approximation of the eikonal equation (see (1.25)) to calculate an approximate value $\hat{\phi}$ for each neighbor $\tilde{\mathbf{x}}$ of $\mathbf{x}_{ij}$. If $\hat{\phi} < \tilde{\phi}$ then set $\tilde{\phi} = \hat{\phi}$ and label $\tilde{\mathbf{x}}$ *considered*. In addition, store the value $\tilde{\phi}$ in a binary heap.

4. let $\tilde{\mathbf{x}}$ be the node in set *considered* with the smallest value $\tilde{\phi}$ found in the binary heap. Label $\tilde{\mathbf{x}}$ *accepted*

5. for each neighbor $\hat{\mathbf{x}}_{ij}$ of $\tilde{\mathbf{x}}$ that is not *accepted* calculate a new tentative value $\tilde{\phi}_{ij}$

6. if $\tilde{\phi}_{ij} < \phi_{ij}$ then set $\phi_{ij} = \tilde{\phi}_{ij}$ and label $\tilde{\mathbf{x}}_{ij}$ as *considered*

7. if $\exists\ \mathbf{x}_{ij} \in$ *considered* repeat steps 4-7

Consider Figure 1.3 for a graphical example of these steps. The power behind this method has to due with the heap sort used to keep track of the unknown nodes. We know that characteristics flow outward from the boundary, this feature also lets us know that information flows from nodes with smaller values ϕ_{ij} to larger values. While we dont know the order of the final ϕ_{ij}, we can find it by sorting all of our starting nodes, and then adding on the smallest of the unknown nodes at each step. The heap sort the *considered* nodes are stored in allows us to do this with low cost.

1.4 Fast sweeping

Fast sweeping is another option to solve the eikonal equation. Like the fast marching method, fast sweeping also uses a first order approximation of the eikonal equation. However instead of an optimal update ordering requiring a sort to find the smallest node, a Gauss-Seidel iterative approach is used with alternating sweep directions. For a regular grid in $\mathbb{R}^n$ it will take 2^n different sweep directions in order to fully redistance the grid. A sweep direction is simply the order on which the value of ϕ will be calculated on the grid nodes. The motivation for the sweeps is rather simple and follow from two properties. The first is that the characteristics of the eikonal equation are straight lines. The second is that in $\mathbb{R}^n$ each axis can be broken into a positive and a negative component. This means that we can group characteristics into similar segments based off of whether they are positive or negative for each grid axis. For example, in 2D we can break the characteristics into four groups, NE, SE, NW, SW, where the N defines characteristics that are positive in the Y axis, S are characteristics that are negative in the Y axis, and W and E are defined similarly. We now define one set of sweep directions that will

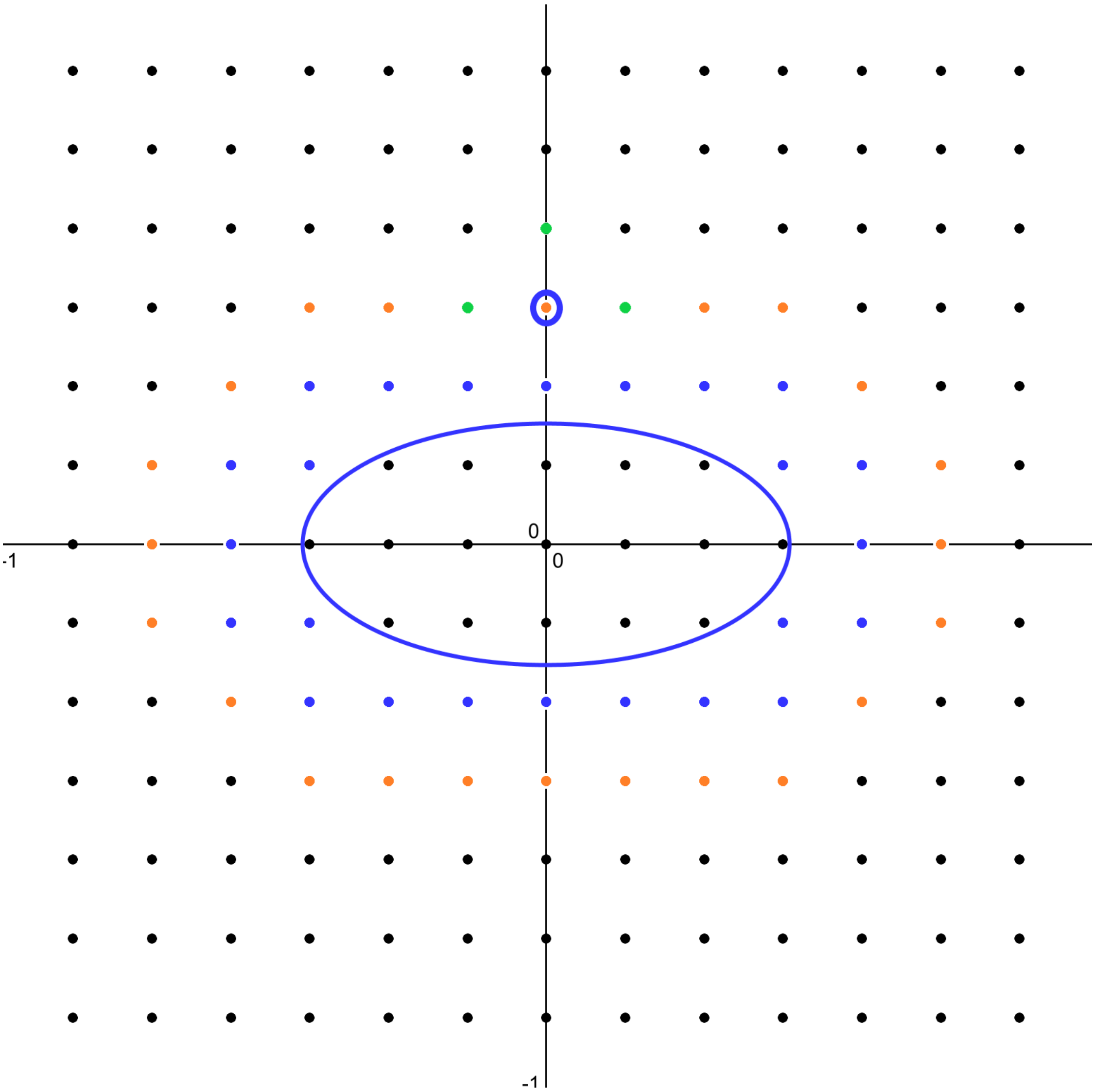

Figure 1.3: Consider a level set with zero iso-countour given by the ellipse. The black nodes have a value $\phi_{ij} = 0$ given by step 1. For our purposes we are only considering nodes outside of Γ where $\phi_0 > 0$. The blue nodes are the original nodes that need to be defined before the algorithm starts in step 2. In most cases these values are found by using a "linear" approximation near the boundary. The orange nodes are those found by calculating all approximations in neighbor nodes in step 3. The circled orange node is the node with the smallest value ϕ_{ij} found in step 4. The green nodes are those that were calculated using the new value of the blue circled node in steps 5 and 6. These steps would then continue until every grid node of consideration was blue.

capture these characteristics. Assume we have a regular grid in $\mathbb{R}^2$ of size $n \times n$, for the 4 directions given below, we iterate over i first then increment/decrement j.

$$\begin{aligned} NW &: j = 1 : n, i = 1 : n \\ SW &: j = n : 1, i = 1 : n \\ NE &: j = 1 : n, i = n : 1 \\ SE &: j = n : 1, i = n : 1 \end{aligned} \tag{1.26}$$

Because of the fact that different sweeps will carry different groups of characteristics, instead of using the ϕ_H and ϕ_V given in Equation (1.25) they are defined based on the sweep directions. Once chosen, we can define $\phi^{NE}, \phi^{SE}, \phi^{NW}, \phi^{SW}$ as follows

$$\begin{aligned} \phi_{ij}^{NE} &= \frac{\phi_{i-1,j}^{NE} + \phi_{i,j-1}^{NE}}{2} + \frac{1}{2}\sqrt{(\phi_{i-1,j}^{NE} + \phi_{i,j-1}^{NE})^2 - 2((\phi_{i-1,j}^{NE})^2 + (\phi_{i,j-1}^{NE})^2 - h^2)} \\ \phi_{ij}^{SE} &= \frac{\phi_{i-1,j}^{SE} + \phi_{i,j+1}^{SE}}{2} + \frac{1}{2}\sqrt{(\phi_{i-1,j}^{SE} + \phi_{i,j+1}^{SE})^2 - 2((\phi_{i-1,j}^{SE})^2 + (\phi_{i,j+1}^{SE})^2 - h^2)} \\ \phi_{ij}^{NW} &= \frac{\phi_{i+1,j}^{NW} + \phi_{i,j-1}^{NW}}{2} + \frac{1}{2}\sqrt{(\phi_{i+1,j}^{NW} + \phi_{i,j-1}^{NW})^2 - 2((\phi_{i+1,j}^{NW})^2 + (\phi_{i,j-1}^{NW})^2 - h^2)} \\ \phi_{ij}^{SW} &= \frac{\phi_{i+1,j}^{SW} + \phi_{i,j+1}^{SW}}{2} + \frac{1}{2}\sqrt{(\phi_{i+1,j}^{SW} + \phi_{i,j+1}^{SW})^2 - 2((\phi_{i+1,j}^{SW})^2 + (\phi_{i,j+1}^{SW})^2 - h^2)} \end{aligned} \tag{1.27}$$

Where each of $\phi^{NE}, \phi^{SE}, \phi^{NW}, \phi^{SW}$ are updated in the order defined in (1.26). Once all 4 sweeps have been run, the final value ϕ_{ij} is defined to be the minimum value over all the sweeps.

$$\phi_{ij} = \texttt{min}(\phi_{ij}^{NE}, \phi_{ij}^{SE}, \phi_{ij}^{NW}, \phi_{ij}^{SW})$$

Note that this is only one set of sweeps that fully update the grid. Any set of sweeps that can cover the 2^n directions will suffice. For a graphical example see Figure 1.4. The different directions defined in (1.27) are represented by the different colors. The NE sweep is blue, NW sweep is brown, SE sweep is purple, SWsweep is green. The colored nodes show which nodes will have their final values set from a corresponding sweep. While like fast marching, fast sweeping requires the nodes near the boundary to be updated, this step was skipped in the image for clarity. We note that on the inside of the ellipse, the color are sitting in opposite quadrants from the outside. This is simply due to the direction of the characteristics. Some interesting work by Detrixhe et al. [14]

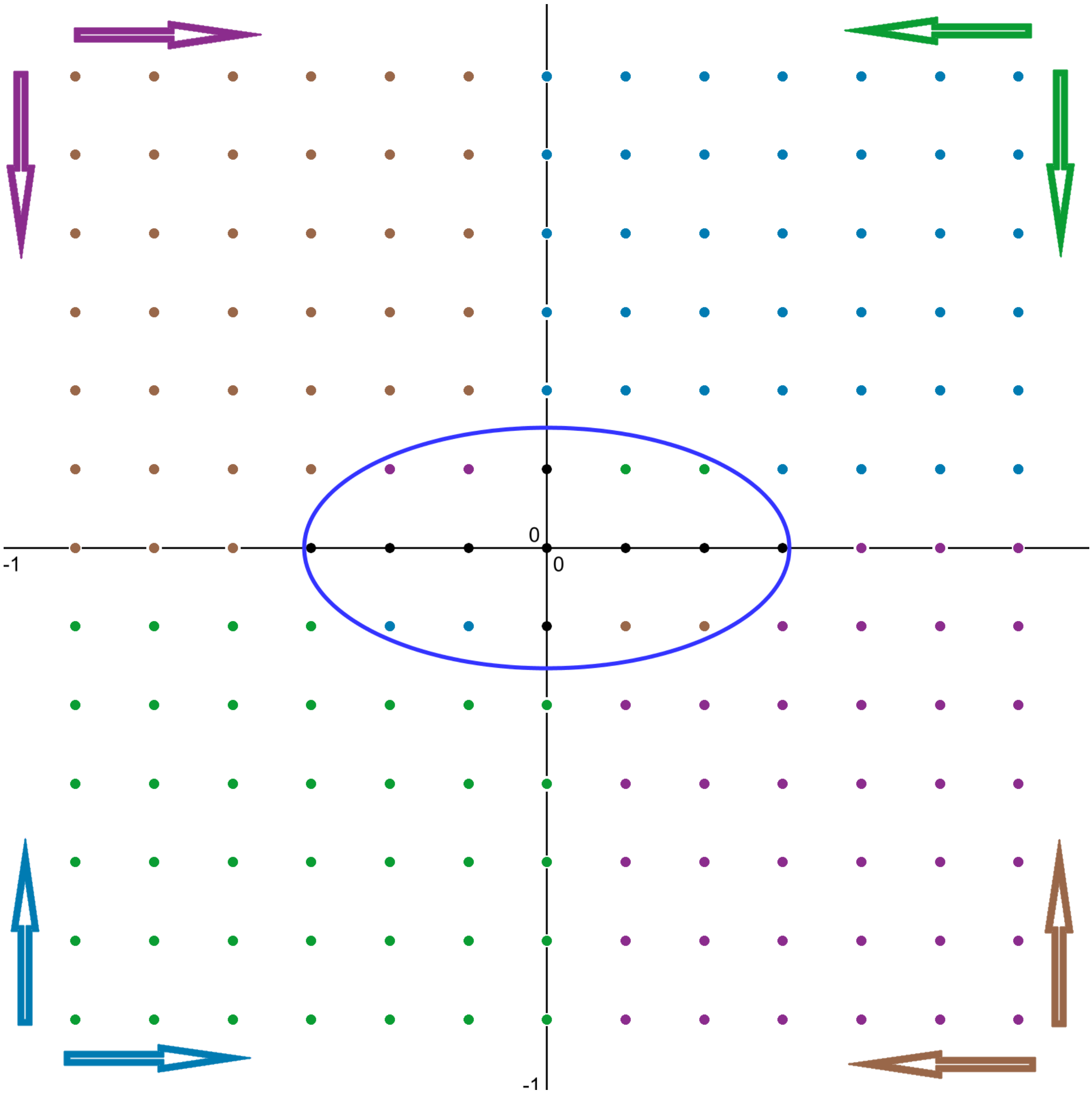

Figure 1.4: Consider a level set with zero iso-countour given by the ellipse. Each color represents a different sweep direction. Each sweep will first follow the horizontal axis, then the vertical axis.

used update directions aligned along the diagonals of the grid in order for more parallel updates. More details will be discussed in the next section.

One drawback of the fast sweeping method is the inability to adhere to a narrow banding strategy. In many cases only data close to the interface is necessary. For example in level set advection, as long as the level set function ϕ has the correct sign everywhere (positive outside of the set, negative inside, zero on the boundary), and has $\nabla\phi = 1$ ”close” to the zero-isocontour of ϕ, the advection will result in the correct answer. With fast sweeping, the only narrow banding strategy that exists would be to only update nodes that could be within the narrow band. However due to the fact that Γ is only required to be closed, the only way to update the narrow band is by checking every node in the grid, and only updating the close ones. While this will speed up the runtime by not running full updates on every grid node, it is not as nice as the fast marching method in which the algorithm can simply terminate once the computed distance reaches a certain threshold.

1.5 Higher order methods

As stated, both FMM and FSM are first order methods. This can simply be seen by the fact that we used a first order discretization of the eikonal equation. Consider only $\phi_x(\mathbf{x}_{ij})$ the first order approximation is

$$\begin{aligned}
\phi_x^+(\mathbf{x}_{ij}) &\approx D_{ij}^{+x}\phi = \frac{\phi_{i+1,j} - \phi_{ij}}{h} \\
\phi_x^-(\mathbf{x}_{ij}) &\approx D_{ij}^{-x}\phi = \frac{\phi_{i-1,j} - \phi_{ij}}{h}
\end{aligned}$$

If we instead use a second order approximation for the partial derivatives

$$\begin{aligned}
\phi_x^+(\mathbf{x}_{ij}) &\approx \frac{\phi_{i+1,j} - \phi_{ij}}{h} + \frac{h}{2}\frac{\phi_{i+2,j} - 2\phi_{i+1,j} + \phi_{ij}}{h^2} = \frac{\phi_{i+2,j} - 4\phi_{i+1,j} + 3\phi_{ij}}{2h} \\
\phi_x^-(\mathbf{x}_{ij}) &\approx \frac{\phi_{ij} - \phi_{i-1,j}}{h} + \frac{h}{2}\frac{\phi_{ij} - 2\phi_{i-1,j} + \phi_{i-2,j}}{h^2} = 3\frac{\phi_{ij} - 4\phi_{i-1,j} + \phi_{i-2,j}}{2h}
\end{aligned}$$

solving for ϕ_{ij} in

$$\max(\phi_x^+(\mathbf{x}_{ij}), \phi_x^-(\mathbf{x}_{ij}), 0)^2 + \max(\phi_y^+(\mathbf{x}_{ij}), \phi_y^-(\mathbf{x}_{ij}), 0)^2 = 1 \tag{1.28}$$

will give a second order approximation of the solution of the eikonal equation. Note this approximation breaks down near the boundary. In general we assume that we have initialized a band of nodes around Γ 1 node thick. However the second order approximation requires information 2 nodes away from a given point. In practice the two main solutions to this problem is to either initialize a band 2 nodes thick, or to use the first order approximation close to the boundary, and use the second order approximation further away from the boundary.

Another higher order method to solve the redistancing problem is to instead solve

$$\phi_t + |\nabla \phi| = 1 \tag{1.29}$$

This equation has the property that after time T, $\phi(\mathbf{x}, T)$ has the correct values for all points within T distance of the boundary. Following Osher and Fedkiw [7], we can discretize the gradient term using HJENO[6] and the time derivative using a Runga-Kutta scheme. A drawback of this method, is due to the discretization, it is possible for the interpolation of Γ to move between time steps. Further work by Sussman et al. [15] accounted for this by adding extra terms to the right hand side to combat the movement of the interface.

1.6 Parallel fast marching and fast sweeping

Notably, both the FMM and FSM approaches create data flow dependencies since information is propagated from the zero isocontour outwards and this complicates parallel implementation. Despite this, various approaches have achieved excellent performance with parallelization. The Gauss-Seidel nature of FSM makes it more amenable to parallelization than FMM. Zhao initially demonstrated this in [16] where each sweep direction was assigned to an individual thread with the final updated nodal value being the minimum nodal value from each of the threads. We note that this only allows the use of 2^n threads in n dimensions. For low dimensional problems (2d and 3d) this does not utilize the power available in many computers today. Further scaling has been achieved by splitting the individual sweeps into subdomain sweeps with a domain decomposition approach. The domain decomposition approach involves splitting the domain into m subdomains. Typically they provide ”ghost nodes” that are updated in multiple subdomains and are passed between subdomains either when updated or at different intervals. However, this strategy can require more sweep iterations than the original serial FSM and the required iterations increase with the number of domains which reduces parallel efficiency. Detrixhe et al. [14] developed a parallel FSM that scales in an arbitrary number of threads without requiring more iterations than in the serial case. Rather than performing grid-axis-aligned Gauss-Seidel sweeps, they use Cuthill-McKee ordering (grid-diagonal) to decouple the data dependency(see figure (1.5). Since the upwind difference stencil only uses grid axis neighbors, nodes along a diagonal do not participate in the same equation and can thus be updated in parallel trivially.

FMM is more difficult to implement in parallel, however even Tsitsiklis [1] developed a parallel FMM algorithm using a bucket data structure. A number of approaches use domain decomposition ideas similar to Zhao [16] and Detrixhe et al. [17] to develop parallel FMM [18, 19, 20, 21].
In domain decomposition approaches the grid is typically divided into disjoint sub grids with a layer of ghost nodes representing nodes contained in adjacent neighbors. Each sub grid is updated in parallel with the desired scheme (FMM, FSM, etc). At some point in the scheme the values stored in the ghost nodes are transfered to adjacent neighbors, and nodes that are represented by ghost nodes in neighbors are updated to new values. How often this information is propagated has a bearing on both the speed of the algorithm, as well as the number of iterations required for convergence. An algorithm that passes ghost node information rarely may run faster than another algorithm, but will likely converge slower due to the delay of information crossing the subdomain boundary. For an example of breaking a grid into subgrids see figure (1.6).

Jeong et al. [20] developed the fast iterative method (FIM), which is a parallel approach using domain decomposition but with a looser causal relationship in the node update list to localize updates for Single Instruction Multiple Data (SIMD) level parallelism. Simplifications to the update list in FMM improve parallel scaling, but tend to increase the number of worst case iterations. Dang et al. [22] extended FIM to a coarse/fine-grained approach based on domain decomposition with load balancing via master/worker model that allowed for efficient performance on heterogeneous platforms.

The domain decomposition approach has also been used in FSM, notably in De-

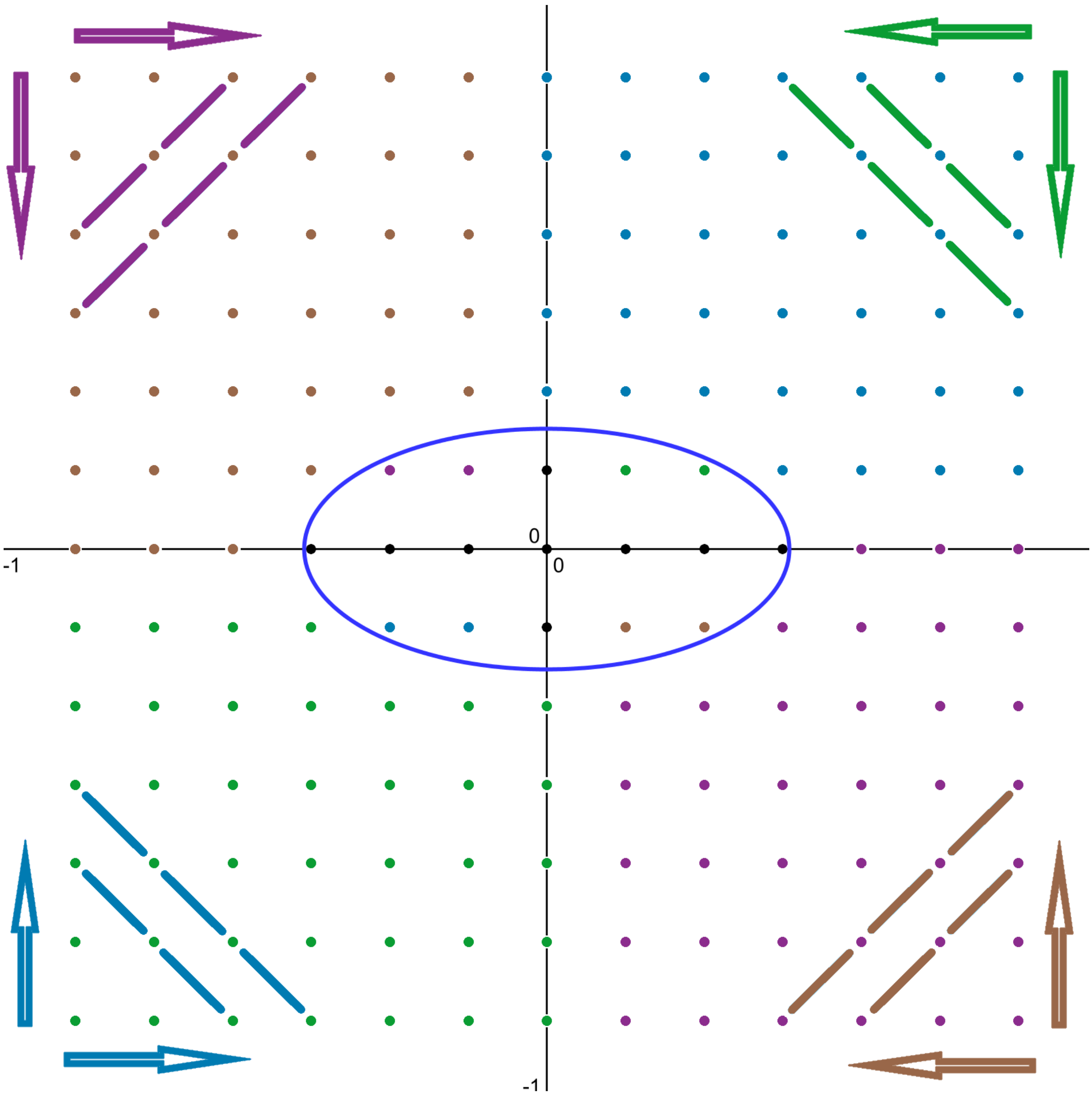

Figure 1.5: Consider a level set with zero iso-countour given by the ellipse. Each color represents a different sweep direction. The sweep directions are now aligned along the diagonals. Nodes connected by a colored line are independent in a given sweep and are updated in parallel before moving to the next row.

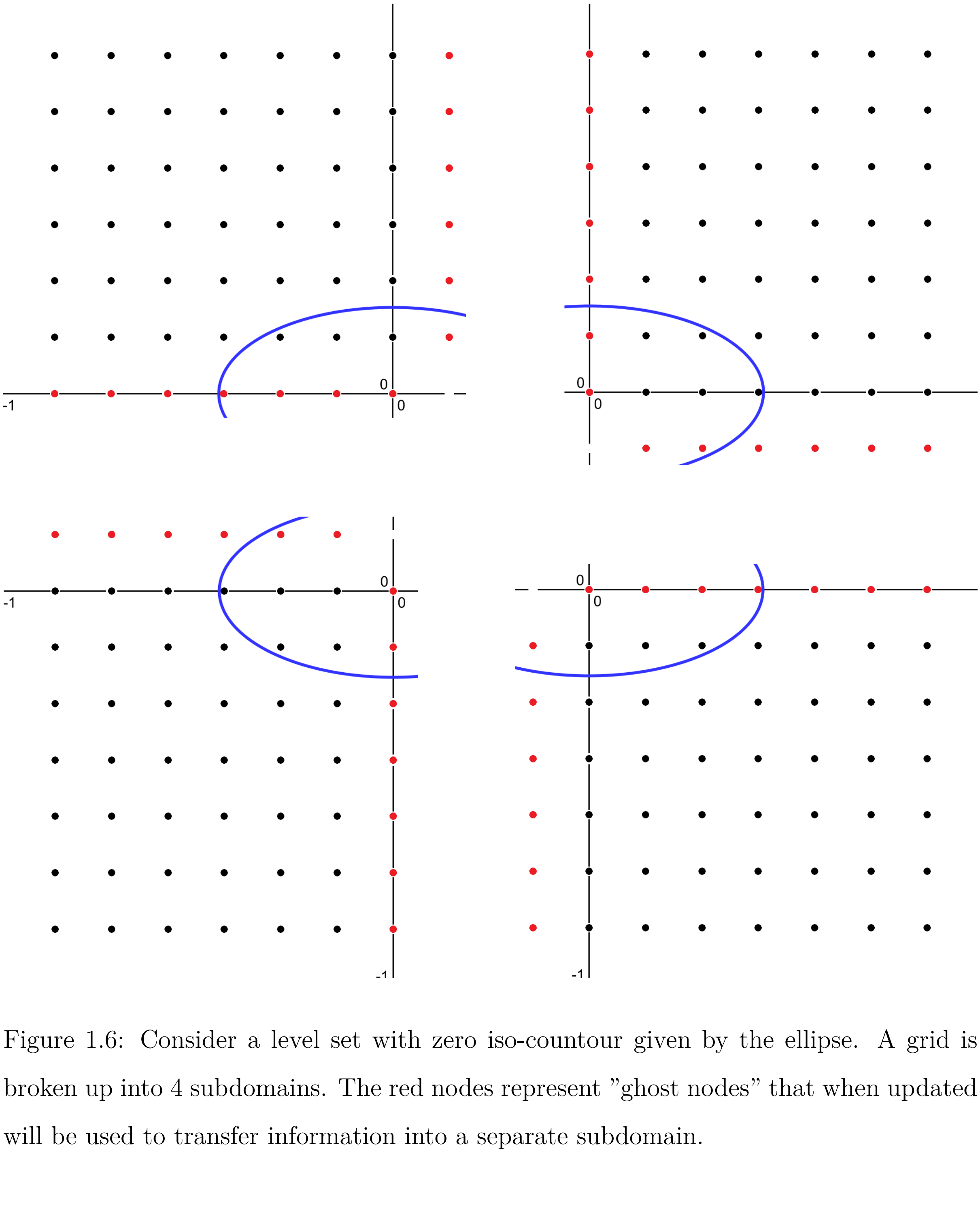

Figure 1.6: Consider a level set with zero iso-countour given by the ellipse. A grid is broken up into 4 subdomains. The red nodes represent ”ghost nodes” that when updated will be used to transfer information into a separate subdomain.

trixhe et al. [14]. They extended the previous ideas of decoupled updates to hybrid distributed/shared memory platforms in [17]. They use a domain decomposition strategy similar to Zhao [16] to divide the grid among available compute nodes and a fine grained shared memory method within each subdomain that utilizes their approach in [14] to achieve orders of magnitude performance increases.

Recently an interesting alternative to FMM and FSM has been proposed. Darbon and Osher [5] and Lee et al. [4] utilize the Hopf-Lax formulation of the solution to the Hamilton-Jacobi form of the eikonal equation. Notably, the signed distance at an arbitrary point is obtained without the need of distance information from neighboring points. This allows for the solution at any given point in any order and prevents the need for communication across cores which greatly simplifies parallel implementation. Furthermore, this inherently allows for updates done only in a narrow band near the zero-isocontour. FSM must solve over the entire domain, and while FMM can be done in a narrow band, FMM methods are generally more difficult to implement in parallel. These aspects make the Hopf-Lax approaches in [4, 5] very compelling for parallel architectures. Lee et al. demonstrated compelling results with abstractly defined functions. However, treatment of grid-based functions is essential for practical application in level set methods, and current their current method could not handle this.

1.7 Hopf-Lax formula

We previously derived a Hamilton-Jacobi formulation for the redistancing problem (Equation (1.9)). However the formulation by itself does not give us a solution to the problem. Luckily for us, Hopf and Lax have already found a solution for us[23, 24]. We will derive their solution by following the work of Evans[11]. We start by defining the Legendre-Fenchel transform of a given function H

$$H^*(x^*) = \sup\{< x, x^* > +H(x) | x \in \mathbb{R}^n\} \tag{1.30}$$

We propose the Hopf-Lax formula as a solution of Equation (1.12)

$$\tilde{\phi}(\mathbf{x_i}, t^k) = \mathtt{min}_{\mathbf{y}\in\mathbb{R}^n} \left\{ \phi_0(\mathbf{y}) + t^k H^* \left(\frac{\mathbf{x_i} - \mathbf{y}}{t^k} \right) \right\} \tag{1.31}$$

To show this suppose $\mathbf{x} \in \mathbb{R}^n, t > 0$ and $\tilde{\phi}$ defined by the Hopf-Lax formula given by Equation (1.31) is differentiable at a point $(\mathbf{x}, t) \in \mathbb{R}^n \times (0, \infty)$. Then

$$\tilde{\phi}_t(\mathbf{x}, t) + H(D\tilde{\phi}(\mathbf{x}, t)) = 0$$

Following Evans[11] we can show this in two parts 1. Fix $\mathbf{q} \in \mathbb{R}^n, h > 0$

$$\begin{aligned}\tilde{\phi}(\mathbf{x} + h\mathbf{q}, t + h) &= \min_{\mathbf{y} \in \mathbb{R}^n} \left\{ hH^* \left(\frac{\mathbf{x} + h\mathbf{q} - \mathbf{y}}{h} \right) + \tilde{\phi}(\mathbf{y}, t) \right\} \\ &\leq hH^*(\mathbf{q}) + \tilde{\phi}(\mathbf{x}, t)\end{aligned}$$

Where the equality uses a Lemma given in Evans [11]

$$\frac{\tilde{\phi}(\mathbf{x} + h\mathbf{q}, t + h) - \tilde{\phi}(\mathbf{x}, t)}{h} \leq H^*(\mathbf{q})$$

let $h \to 0^+$

$$\mathbf{q} \cdot D\tilde{\phi}(\mathbf{x}, t) + \tilde{\phi}_t(\mathbf{x}, t) \leq H^*(\mathbf{q})$$

This inequality is valid for all $\mathbf{q} \in \mathbb{R}^n$, thus

$$\tilde{\phi}_t(\mathbf{x}, t) + H(D\tilde{\phi}(\mathbf{x}, t)) = \tilde{\phi}_t(\mathbf{x}, t) + \max_{\mathbf{q} \in \mathbb{R}^n} \{\mathbf{q} \cdot D\tilde{\phi}(\mathbf{x}, t) - H^*(\mathbf{q})\} \leq 0 \tag{1.32}$$

We only need H to be convex for the equality to hold

2. Now choose $\mathbf{z}$ such that $\tilde{\phi}(\mathbf{x}, t) = tH^*(\frac{\mathbf{x}-\mathbf{z}}{t}) + \phi_0(\mathbf{z})$. fix $h > 0$ and set $s = t - h, y = \frac{s}{t}\mathbf{x} + (1 - \frac{s}{t})\mathbf{z}$. Then $\frac{\mathbf{x}-\mathbf{z}}{t} = \frac{\mathbf{y}-\mathbf{z}}{s}$, and thus

$$\begin{aligned}\tilde{\phi}(\mathbf{x}, t) - \tilde{\phi}(\mathbf{y}, s) &\geq tH^* \left(\frac{\mathbf{x} - \mathbf{z}}{t} \right) + \phi_0(\mathbf{z}) - \left[tH^* \left(\frac{\mathbf{y} - \mathbf{z}}{s} \right) + \phi_0(\mathbf{z}) \right] \\ &= (t - s)H^* \left(\frac{\mathbf{x} - \mathbf{z}}{t} \right)\end{aligned}$$

I.E.

$$\frac{\tilde{\phi}(\mathbf{x}, t) - \tilde{\phi}((1 - \frac{h}{t})\mathbf{x} + \frac{h}{t}\mathbf{z}, t - h)}{h} \geq H^* \left(\frac{\mathbf{x} - \mathbf{z}}{t} \right)$$

Let $h \to 0^+$

$$\frac{\mathbf{x} - \mathbf{z}}{t} \cdot D\tilde{\phi}(\mathbf{x}, t) + \tilde{\phi}_t(\mathbf{x}, t) \geq H^* \left(\frac{\mathbf{x} - \mathbf{z}}{t} \right)$$

As such

$$\begin{aligned}\tilde{\phi}_t(\mathbf{x}, t) + H(D\tilde{\phi}(\mathbf{x}, t)) &= \tilde{\phi}_t(\mathbf{x}, t) + \max_{\mathbf{q} \in \mathbb{R}^n} \left\{ \mathbf{q} \cdot D\tilde{\phi}(\mathbf{x}, t) - H^*(\mathbf{q}) \right\} \\ &\geq \tilde{\phi}_t(\mathbf{x}, t) + \frac{\mathbf{x} - \mathbf{z}}{t} \cdot D\tilde{\phi}(\mathbf{x}, t) - H^* \left(\frac{\mathbf{x} - \mathbf{z}}{t} \right) \\ &\geq 0\end{aligned}$$

Thus we have shown that any $\tilde{\phi}$ satisfying (1.31) is a solution for (1.12).

1.8 Method

As in Lee et al. [4] we use the Hamilton-Jacobi formulation of the eikonal equation(1.1) given previously.

$$\begin{aligned} &\frac{\partial}{\partial t}\tilde{\phi}(\mathbf{x},t) + \|\nabla\tilde{\phi}(\mathbf{x},t)\|_2 = 0 \\ &\tilde{\phi}(\mathbf{x},0) = \phi^0(\mathbf{x}) \end{aligned} \tag{1.33}$$

for $\mathbf{x} \in \mathbb{R}^n, t > 0$. We assume that ϕ^0 is constructed such that

$$\begin{cases} \phi^0(\mathbf{x}) < 0 & \mathbf{x} \in \Omega\backslash\Gamma \\ \phi^0(\mathbf{x}) > 0 & \mathbf{x} \in (\mathbb{R}^n\backslash\Omega) \\ \phi^0(\mathbf{x}) = 0 & \mathbf{x} \in \Gamma \end{cases}$$

for some set $\Omega \subset \mathbb{R}^n$. Similarly to Lee et al. [4] we assume that the set Ω is closed and non-empty. While the theory does not require that Ω is bounded, in practice because we will be dealing with closed bounded subsets of $\mathbb{R}^n$ we will assume that Ω is also bounded. Isocontours of the time dependent solution $\tilde{\phi}$ progress from the boundary Γ in its normal direction at a rate of 1. This property is the reason the construction of ϕ^0 is important. When constructed properly, $\nabla\phi^0$ will point in the normal direction away from the set Ω. To know the distance to the boundary Γ, we simply need to know at which time $\hat{t}$ the zero-isocontour of $\tilde{\phi}$ has progressed to the point $\mathbf{x}$. In other words, the signed distance $(\phi(\mathbf{x}))$ from the point $\mathbf{x}$ to the boundary Γ is given by the time $\hat{t}$ with $\tilde{\phi}(\mathbf{x},\hat{t}) = 0$: $\phi(\mathbf{x}) = \hat{t}$. Note that we only consider the case here of positive ϕ since the case of negative ϕ is trivially analogous.

As in Lee et al. [4], we treat the problem as root finding and use the secant method. However, unlike Lee et al. we are specifically interested in redistancing grid based functions. Thus we assume that the initial function is defined in terms of its interpolated values from grid nodes as $\phi^0(\mathbf{x}) = \sum_{\mathbf{i}} \phi^0_{\mathbf{i}} N_{\mathbf{i}}(\mathbf{x})$ where the function $N_{\mathbf{i}}$ is the bilinear interpolation kernel associated with grid node $\mathbf{x}_{\mathbf{i}}$ and $\phi^0_{\mathbf{i}} = \phi^0(\mathbf{x}_{\mathbf{i}})$. Also, when we solve for the redistanced values, we do so only at grid nodes (i.e. we solve for $\phi_{\mathbf{i}} = \phi(\mathbf{x}_{\mathbf{i}}) = \hat{t}$). Thus the secant method only requires the evaluation of the function $\tilde{\phi}(\mathbf{x}_{\mathbf{i}}, t^k)$ for iterative approximation $t^k \to \hat{t}$. We next discuss the practical implementation of the secant method and evaluation of $\tilde{\phi}(\mathbf{x}_{\mathbf{i}}, t^k)$ for grid based data.

1.8.1 Secant method for roots of $\tilde{\phi}(\mathbf{x_i}, \hat{t}) = 0$

In order to use the secant method to solve for the root in this context, we use the iterative update

$$t^{k+1} = t^k - \tilde{\phi}(\mathbf{x_i}, t^k)\frac{t^k - t^{k-1}}{\tilde{\phi}(\mathbf{x_i}, t^k) - \tilde{\phi}(\mathbf{x_i}, t^{k-1})}. \tag{1.34}$$

The initial guess t^0 can either be set from neighboring nodes that have been recently updated, or generally from a priori estimates of the distance (see Section 1.8.4). However, when no such information is possible or when it would negatively effect parallel performance we use $t^0 = 0$. We set $t^1 = t^0 + \epsilon$ where ϵ is proportionate to the grid cell size.

The main concern with using the secant method in this context is that while $\tilde{\phi}(\mathbf{x_i}, t)$ is monotonically decreasing in t, it is not strictly monotone. This means that there can be times t where $\frac{d}{dt}\tilde{\phi}(\mathbf{x_i}, t) = 0$. For example, if the minimum of $\phi^0(\mathbf{x_i})$ over the ball centered at $\mathbf{x_i}$ of radius t is in the interior of the ball (at a point of distance s from $\mathbf{x_i}$), then $\frac{d}{dt}\tilde{\phi}(\mathbf{x_i}, r) = 0$ for $s \leq r \leq t$ (see Section 1.8.2). The secant method is not well defined if we have iterates with equal function values. To compensate for this, if secant would divide by zero, and we have not already converged, we simply increase or decrease $t^{k+1} = t^k \pm \Delta t_{\max}$ in the correct direction. The correct direction is trivial to find, because if $\tilde{\phi}(\mathbf{x_i}, t^k) > 0$ then we need to increase t^k. Otherwise we need to decrease t^k. In practice, we use $\Delta t_{\max} = 5\Delta x$ where Δx is the grid size.

Another issue is that errors in the approximation of $\tilde{\phi}(\mathbf{x_i}, t^k)$ can lead to more secant iterations. This can be reduced by solving for $\tilde{\phi}(\mathbf{x_i}, t^k)$ to a higher tolerance. However, requiring more iterations to approximate $\tilde{\phi}(\mathbf{x_i}, t^k)$ more accurately can be more costly than just using more secant iterations with a less accurate (but more efficient) approximation to $\tilde{\phi}(\mathbf{x_i}, t^k)$. We discuss the process and cost of solving for $\tilde{\phi}(\mathbf{x_i}, t^k)$ in Section 1.8.2. Our modified secant algorithm is summarized in Algorithm 1.8.1.

Algorithm 1 Modified Secant Method

```
while $|\tilde{\phi}(\mathbf{x_i}, t^{k+1})| > \epsilon$ do
    $\Delta t = -\tilde{\phi}(\mathbf{x_i}, t^k)\frac{t^k - t^{k-1}}{\tilde{\phi}(\mathbf{x_i}, t^k) - \tilde{\phi}(\mathbf{x_i}, t^{k-1})}$
    if $|\Delta t| > tol$ then
        if $\tilde{\phi}(\mathbf{x_i}, t^k) > 0$ then
            $\Delta t = \Delta t_{\max}$
        else
            $\Delta t = -\Delta t_{\max}$
        end if
    end if
    $t^{k+1} = t^k + \Delta t$
end while
```

1.8.2 Hopf-Lax formula for $\tilde{\phi}(\mathbf{x_i}, t^k)$

The only piece of the puzzle we are currently missing to use Hopf-Lax formulation (1.31) is finding the Legendre-Fenchel transform of H. For our work $H = \|\cdot\|_2$ and H^* can be derived as

$$
\begin{aligned}
\|\mathbf{x}^*\|^* &= \sup_{x \in \mathrm{R}^n} \mathbf{x}^{*T}\mathbf{x} - \|x\| \\
&= \sup_{\mathbf{x} \in \mathrm{R}^n} \mathbf{x}^* T\mathbf{x} - \sup_{\mathbf{y} \in \mathrm{R}^n, \|\mathbf{y}\|_* \leq 1} \mathbf{y}^T\mathbf{x} \\
&= \inf_{\mathbf{y} \in \mathrm{R}^n, \|\mathbf{y}\|_* \leq 1} \sup_{\mathbf{x} \in \mathrm{R}^N} \mathbf{x}^T(\mathbf{x}^* - \mathbf{y}) \\
&= \inf_{\mathbf{y} \in \mathrm{R}^n, \|\mathbf{y}\|_* \leq 1} \begin{cases} 0 & \mathbf{y} = \mathbf{x}^* \\ \infty & else \end{cases} \\
&= \begin{cases} 0 & \|\mathbf{x}^*\|_* \leq 1 \\ \infty & else \end{cases}
\end{aligned}
$$

where $\|\cdot\|_*$ is the dual norm of $\|\cdot\|$. For a norm $\|\cdot\|_p$ its dual norm is $\|\cdot\|_q$ where $\frac{1}{p}+\frac{1}{q}=0$. Note that this means that the dual norm of the 2 norm, is itself the 2 norm

$$H^*(\mathbf{x}) = \begin{cases} 0 & \|\mathbf{x}\|_2 \leq 1 \\ \infty & otherwise \end{cases}$$

or equivalently

$$\tilde{\phi}(\mathbf{x_i}, t^k) = \min_{\mathbf{y} \in B(\mathbf{x_i}, t^k)} \phi^0(\mathbf{y}) \tag{1.35}$$

where $B(\mathbf{x_i}, t^k)$ is the ball of radius t^k around grid node $\mathbf{x_i}$. Thus the problem of evaluating $\tilde{\phi}(\mathbf{x_i}, t^k)$ amounts to finding the minimum of the initial ϕ^0 over a ball. While Lee et al. [4] use Split Bregman iteration to solve this, we instead simply use projected gradient descent. The only constrain on the system is restricting the minimum to a ball. Since projecting to a ball is simple, the extra work required in Split Bregman only served to slow down the calculations. We found that we could use a few hundred projected gradient iterations and receive accurate answers much faster than in a Split Bregman iteration largely due to the simplicity of the projection. Using $\mathbf{y}_k^0$ as an initial guess for the argmin of ϕ^0 over the ball $B(\mathbf{x_i}, t^k)$, we iteratively update the approximation from

$$\tilde{\mathbf{y}}_k^{j+1} = \mathbf{y}_k^j - \gamma \nabla \phi^0(\mathbf{y}_k^j) \tag{1.36}$$

$$\mathbf{y}_k^{j+1} = \mathtt{PROJ}_{B(\mathbf{x_i}, t^k)}(\tilde{\mathbf{y}}_k^{j+1}) \tag{1.37}$$

where

$$\mathtt{PROJ}_{B(\mathbf{x_i}, t^k)}(\mathbf{y}) = \begin{cases} \mathbf{y} & \|\mathbf{x_i} - \mathbf{y}\|_2 \leq t^k \\ \mathbf{x_i} - t^k \frac{\mathbf{x_i} - \mathbf{y}}{\|\mathbf{x_i} - \mathbf{y}\|_2} & otherwise \end{cases}$$

In practice, we set the step size γ equal to the grid spacing Δx. We found this to be necessary due to the non-smooth nature of our grid approximation. The gradient is not defined over the grid cell boundaries when using a bi-linear interpolation. When using a bi-linear approximation a large amount of the time the minimum over a ball will occur either on a cell boundary or a cell node. To resolve these accurately we need a small step size. Note that the gradients $\nabla \phi^0(\mathbf{y}_k^j)$ are computed using the bilinear interpolation kernels $N_\mathbf{l}(\mathbf{x})$ as $\nabla \phi^0(\mathbf{y}_k^j) = \sum_\mathbf{l} \phi_\mathbf{l}^0 \nabla N_\mathbf{l}(\mathbf{y}_k^j)$. We emphasize that for efficiency the sum over $\mathbf{l}$ can be taken over only the four grid nodes surrounding the cell that the argument

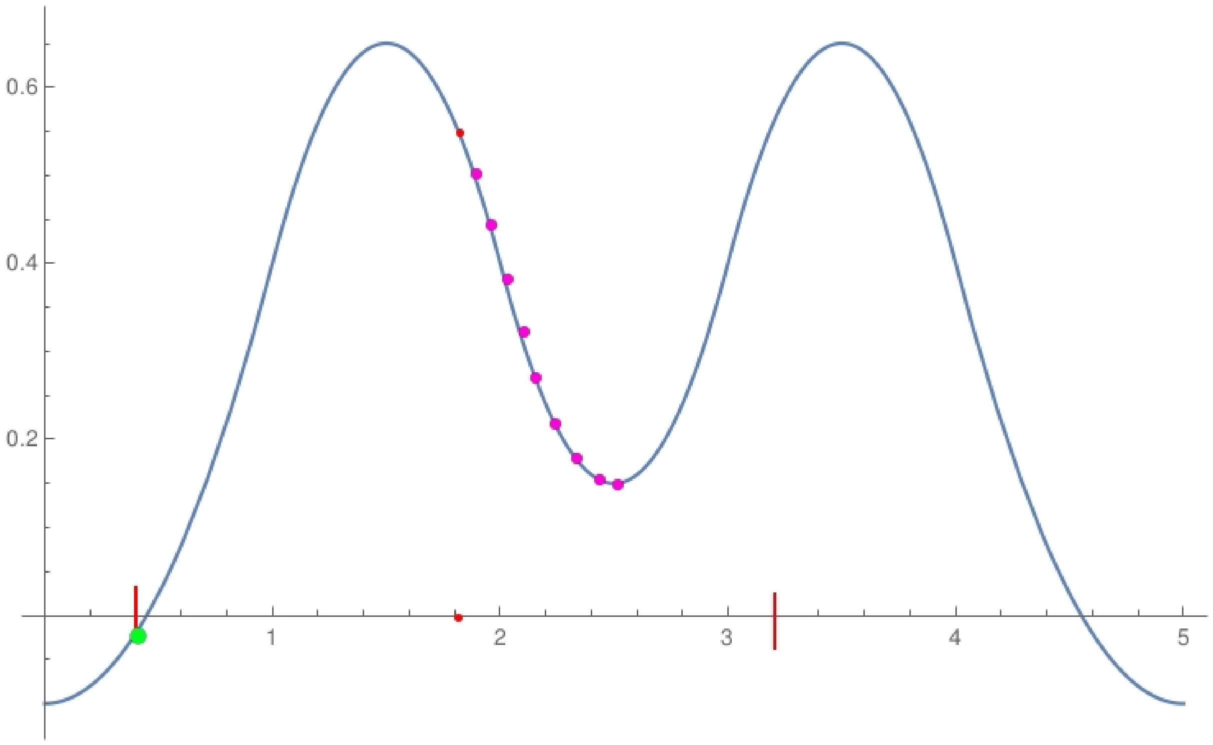

Figure 1.7: The two vertical lines are the boundary of minimization. Grid node $x_i = 1.8$ is in the middle of the region, and also the starting guess for projected gradient descent. The sequence of points leading off to the right represent the subsequent steps of gradient descent. These points converge to the incorrect argmin $x = 2.5$. The correct solution is at $x = 0.4$. In order to converge to this point, the initial guess would have to be less than 1.25

$\mathbf{y}_k^j$ is in. We further note that the index for the cell containing the argument $\mathbf{y}_k^j$ can be found in constant time using floor($\frac{y_{\alpha k}^j}{\Delta x}$) where $y_{\alpha k}^j$ are the components of $\mathbf{y}_k^j$. In general, ϕ^0 is a non-convex function defined from the grid interpolation and projected gradient descent will only converge to a local minimum. We illustrate this in Figure 1.7. Failure to converge to a global minimum can lead to large errors in the approximation of $\tilde{\phi}(\mathbf{x_i}, t^k)$. While it is impractical to ensure we achieved a global minimizer, it is possible to find multiple local minimizers increasing the probability we find a global minimizer. We solve (1.35) multiple times with different initial guesses y_k^0 and then take the minimum over these solutions to come up with a final answer that is likely to be close to a global minimzer. We found in practice, on the order of one guess per grid cell in the ball $B(\mathbf{x_i}, t^k)$ is sufficient to find a global minimizer. For problems without many local extrema the number of initial guesses can be reduced. In general when finding $\tilde{\phi}(\mathbf{x_i}, t^k)$ we use as initial guesses $\texttt{PROJ}_{B(\mathbf{x_i},t^k)}(\mathbf{y}_{k-1})$ where

$$\mathbf{y}_{k-1} = \underset{\mathbf{y} \in B(\mathbf{x_i}, t^{k-1})}{\text{argmin}} \phi^0(\mathbf{y})$$

is the argmin of ϕ^0 used in the previous secant iteration as well as a small number of random points in $B(\mathbf{x_i}, t^k)$. We use this strategy because $\texttt{PROJ}_{B(\mathbf{x_i},t^k)}(\mathbf{y}_{k-1})$ tends to be a good guess for the global minimum. In general, it is very likely that at the next step, the minimum will either be the same point, or along the boundary. Therefore, we prioritize random initial guesses near the boundary of the ball. In fact, for $t^{k-1} < t^k$ we know that the argmin will be at a distance s from $\mathbf{x_i}$ with $t^{k-1} \leq s \leq t^k$ so in theory we should only sample in the annulus. However, in practice we do not have full confidence in the argmin attained at iteration $k-1$ since our procedure is iterative. Allowing for initial guesses at some locations closer to $\mathbf{x_i}$ than t^{k-1} admits the possibility of finding a more accurate argmin. Thus, we found that skewing the sampling density to be higher towards the boundary of the ball struck a good balance between efficiency and accuracy. We illustrate this process in Figure 1.11.

Failure to find the global minimum over the ball can cause unpredictable behavior in the secant iteration for $\hat{t}$. This includes false positives where a t^k is incorrectly interpreted as a root. However, continuing to run secant to a fixed large number of iterations usually corrects for this. In general, there is a tradeoff between the number of initial guesses

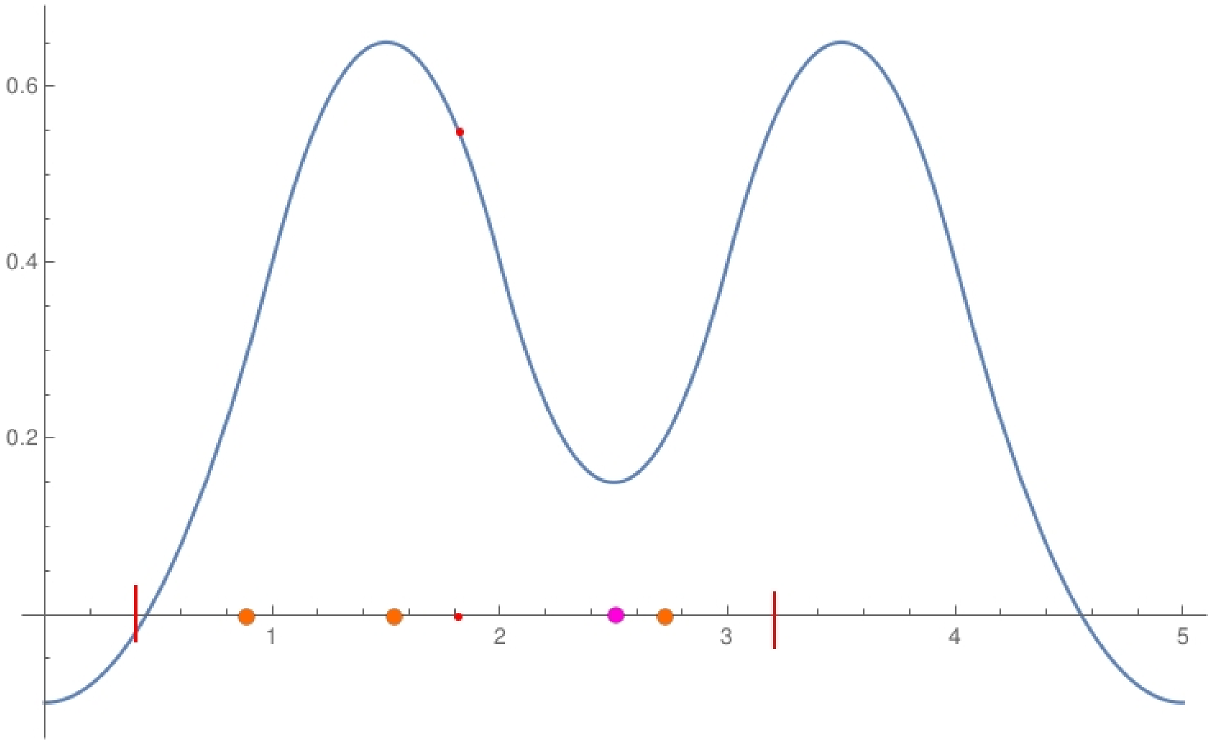

Figure 1.8: The figure illustrates representative random initial guesses used in solving for $\tilde{\phi}(\mathbf{x}_i, t^k)$. In addition, we use an initial guess equal to the minimizer computed in the previous secant iteration shown in magenta.

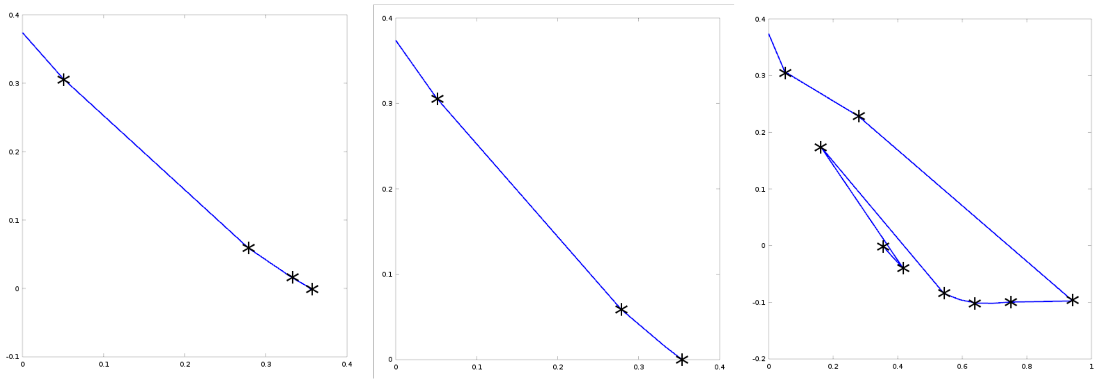

Figure 1.9: The plots above are the points $(\tilde{\phi}(\mathbf{x}_{\mathbf{i}}, t^k), t^k)$ found when running our algorithm with different choices of random guess and gradient descent iterations on circle initial data. The left most plot was run with 100 random guesses, and 1 gradient descent iteration. The middle plot was run with 1 random guess, and 100 gradient descent iterations. The right plot was run with 1 random guess and 5 gradient descent iterations. Note that in all cases, the correct root was found.

and iterations of projected gradient and the number of secant iterations. We illustrate this in Figure 1.9 which shows the path to convergence for a few choices of iteration parameters. When $\tilde{\phi}(\mathbf{x}_{\mathbf{i}}, t^k)$ is solved with high accuracy, the secant iteration converges with minimal overshoot in 7 iterations. When $\tilde{\phi}(\mathbf{x}_{\mathbf{i}}, t^k)$ is not solved to high accuracy, secant overshoots by a large margin, and takes 16 iterations to converge, but notably still converges. However because each iteration is cheaper, the total runtime is lower to reach the same convergence for $\hat{t}$. In practice we found that a few hundred projected gradient iterations combined with our initial guess sampling strategy struck a good balance between accuracy and efficiency.

1.8.3 Grid interpolation order

We experimented with higher-order B-spline interpolation for the kernels in $\phi^0(\mathbf{x}) = \sum_{\mathbf{i}} \phi^0_{\mathbf{i}} N_{\mathbf{i}}(\mathbf{x})$. We tried two different ways to implement cubic B-splines. First we simply tried to use the grid node values as the weights ϕ^0_i. This method failed by causing the interface to move too much in non-smooth regions, and by not capturing small features. When the width of the interface was on the order of one grid cell the interpolation would fail to capture any of the interface (the interpolation would be > 0

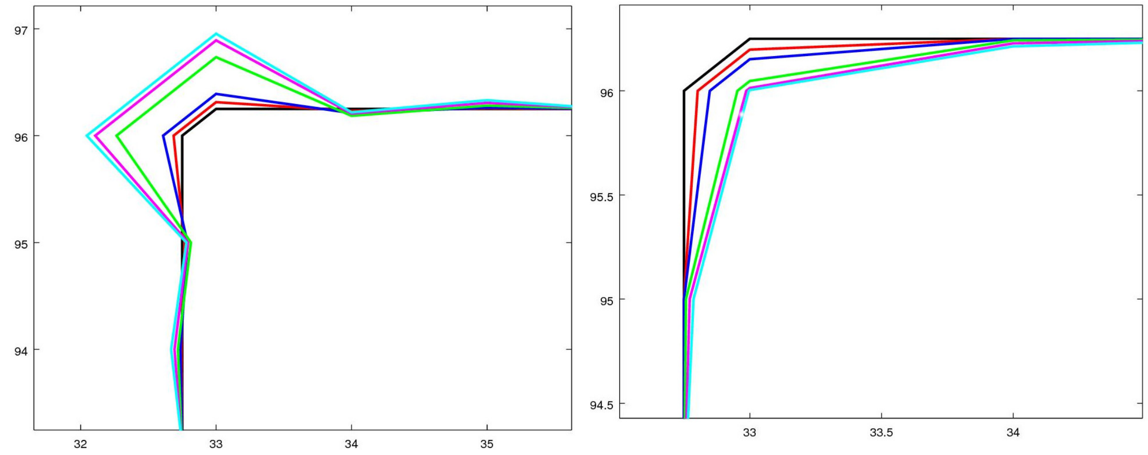

Figure 1.10: For both images Bi-Cubic or Bi-Linear interpolation were used to represent J. Both started with the initial data as a pyramid with a corner in the center of a cell. Each was then reinitialized approximately 50 times and the changing of the zero-iso-counter was shown as above. For cubic on the left, instability is introduced by trying to fit the cubic interpolant to the data. While it handles smooth data better than linear, at sharp corners the data over fits and becomes unstable. Linear on the right suffers from no such instability, however because the sharp corner is lost with interpolation, it suffers from volume loss over time.

in the region). To fix this we tried to find weights $\tilde{\phi}_i^0$ that caused the interpolation to fit the grid nodes exactly. While these weights were able to be found as a precomputation step, we found that when Γ was non-smooth the interpolation tended to overfit the boundary. This causes unwanted growth in the zero-isocontour. Linear interpolation is not perfect either. Linear interpolation suffers from some volume loss at corners and curves. The zero-isocontour however will never cross grid nodes with linear interpolation, so under refinement the volume loss is mitigated. Figure 1.10 shows the problem with both methods

1.8.4 Computing in a narrow band

In many applications, only data close to the interface is needed. Since each grid node can be solved for independently, and in any order, the Hopf-Lax approach naturally lends itself to narrow banding strategies for these applications. We first redistance the function on the coarse grid, then we interpolate values from the coarse grid to the fine grid. We

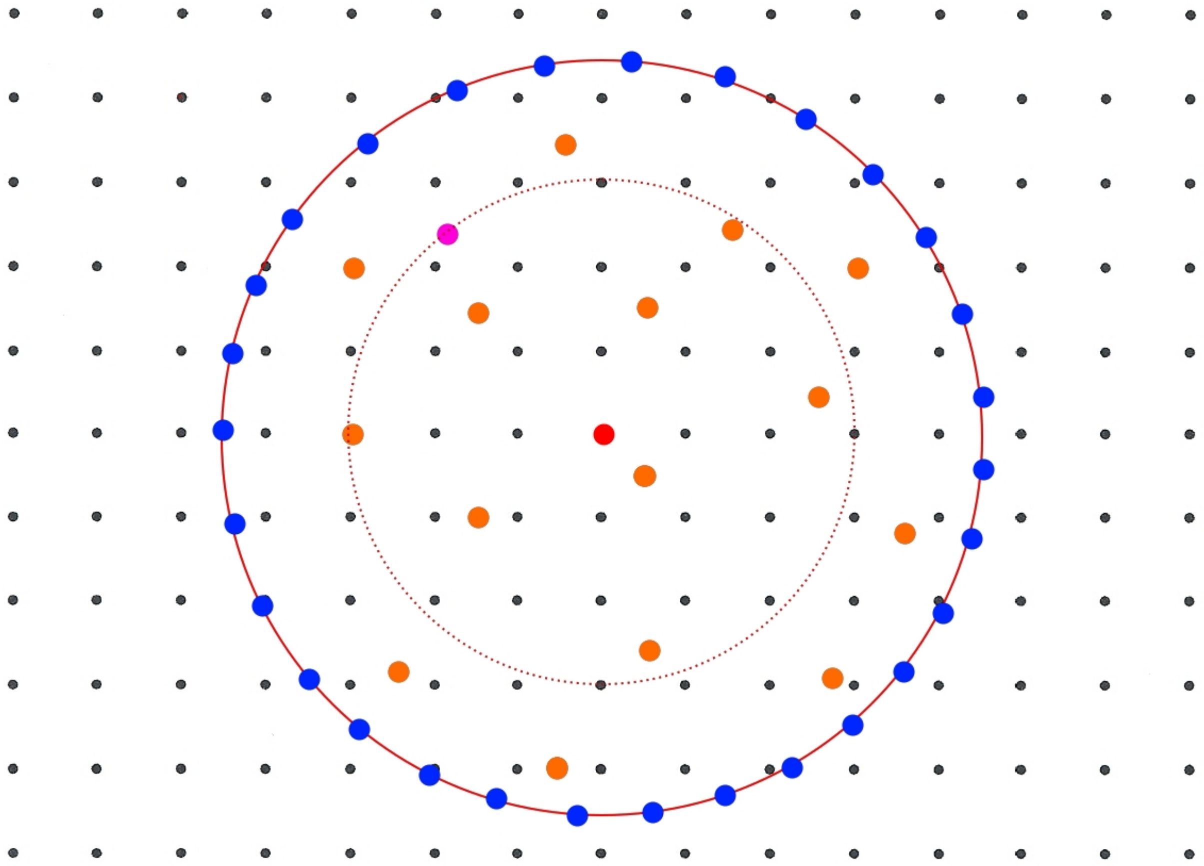

Figure 1.11: In this image, the red dot in the center is $\mathbf{x_i}$, the solid red line represents the ball of radius t_k and the dotted line represents the ball of radius t_{k-1}. The magenta point was the approximate argmin $\mathbf{y}_{k-1}$ of ϕ^0 over the ball of radius t_{k-1}. Since it is unlikely for the minimizer to be inside of t_{k-1} we use coarse (random) grid initial guesses in the interior. However, since it is possible that expanding t will move the minimizer to a different location we take a large number of initial guesses along the boundary of t_k

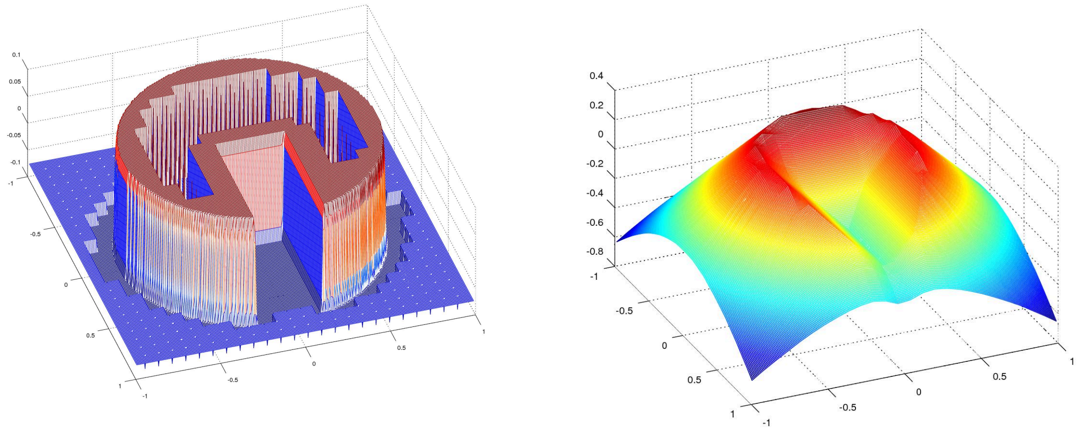

Figure 1.12: A coarse grid 8X smaller than the fine grid was solved for initially. Using those values the fine grid was only solved on cells where the distance to the boundary could be less than 0.1, represented as the solid areas of the left image. In the right image those coarse areas are defined from bilinear interpolation. This coarse/banding approach provided approximately a 2.5 times increase in performance.

then only recompute values on the fine grid that are smaller than a threshold value and we use the value interpolated from the coarse nodes as an initial guess t_0 for the computation on the fine grid. As an example see Figure 1.12. We note that we lose no accuracy by computing the distance at coarse grid nodes because we are still using the fine grid data as our initial conditions. One benefit of this method is our ability to reconstruct distances via interpolation in the coarse regions. This can be useful if the application only needs accurate data near the boundary but still expects data everywhere.

1.8.5 Computing geometric quantities

The Hopf-Lax formulation naturally allows us to compute normals ($\mathbf{n} = \nabla\phi$) and curvatures ($\nabla \cdot \mathbf{n}$) at the grid nodes. As pointed out in Lee et al. [4], as the argmin $\mathbf{y}^k$ from Equation (1.36) is iterated to convergence, it approaches the closest to point to the grid node $\mathbf{x_i}$ on the zero isocontour of ϕ^0. Therefore, recalling that when t^k has converged (within a tolerance) to the root $\hat{t}$ of $\tilde{\phi}(\mathbf{x}, \hat{t}) = 0$, t^k is approximately the distance to the zero isocontour, we can compute the unit normal at the grid node from

$$\mathbf{n}(\mathbf{x_i}) = \frac{\mathbf{x_i} - \mathbf{y}^k}{t^k}.$$

Notably, this approximation is very close to the exact signed distance function with zero isocontour determined by the bilinearly interpolated ϕ^0. It is as accurate as the argmin $\mathbf{y}^k$ so it essentially only depends on the accuracy of the secant method. We get this very accurate geometric information for free. Moreover, the curvature $(\nabla \cdot \mathbf{n})$ can be computed accurately by taking a numerical divergence of $\mathbf{n}(\mathbf{x_i})$ that would have accuracy equal to the truncation error in the stencil (since no grid-based errors are accumulated in the computation of $\mathbf{n}(\mathbf{x_i})$).

1.8.6 Preventing volume loss

For problems where the input data is close to a signed distance function, such as when redistancing the data given by a step in the level set method, a modification can be made to mitigate the volume loss suffered in the bilinear interpolation. Taking inspiration from fast sweeping, we simply do not update nodes next to the boundary. Because boundary nodes are not updated, the zero-isocontour is not changed. Everywhere else we use the signed distance function. This gives us a nice function, without changing the zero-isocontour. The results of this modification with a level set advection scheme can be shown in Figure 1.18.

1.9 Results

All of the following results were run on an Intel 6700k processor with an Nvidia GTX 1080 GPU. The domain for each problem was set to be $[0,1]X[0,1]$ and was run on a 512X512 grid. To ensure efficient performance on the GPU, both projected gradient descent and the secant method were run for a fixed number of iterations rather than to a specific tolerance. All timings listed in this section are averages over 5 seperate runs, with the exception of the Vortex problem which is already an average. This is due to the fact that we noticed in practice variations of up to 10% in the individual runtimes of a given problem

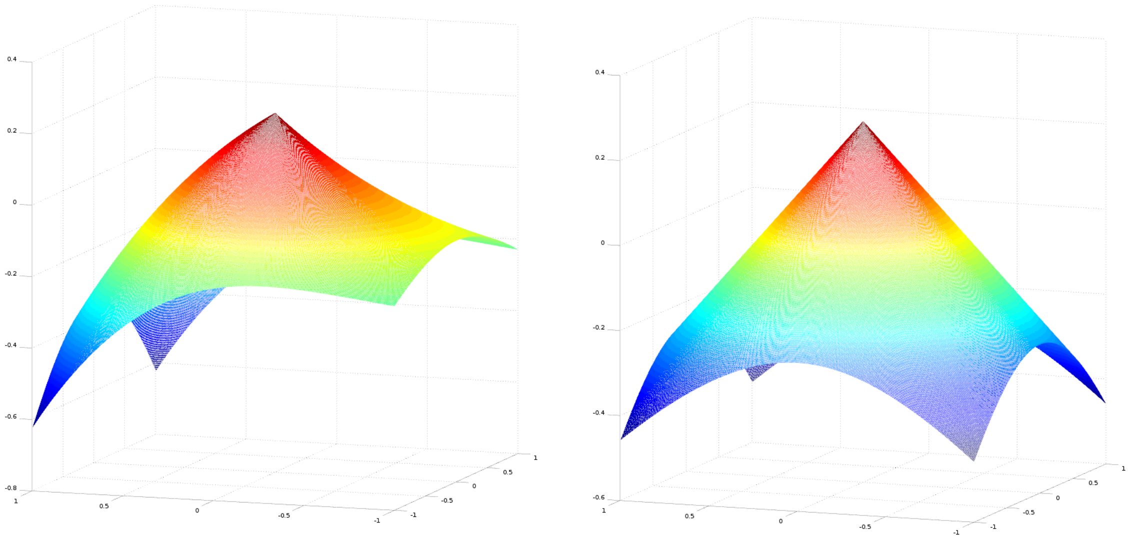

Figure 1.13: Scaled circle: the initial data is $\phi^0 = exp(x) * (.125 - (.5 - x)^2 + (.5 - y)^2)$. The zero level set is a circle of radius .25 centered around (.5,.5)

Problem	*num_secant*	num_rand	num_proj	Timing(ms)
Circle	10	5	100	47.567
Two Points	10	5	100	45.199
Vortex(Per Frame)	10	5	200	73.248
Square	10	4	200	67.582
Sine	10	5	200	71.429

Figure 1.13 shows initial data ϕ^0 with a zero-isocontour given by a circle with radius .25. Figure 1.14 shows a more complicated test. The zero-isocontour is a square bounded between $[.25, .75]$ in x and y. The corners present rarefaction fans, and the inside needs to handle sharp corners along the diagonals. Because of these difficulties (especially the sharp gradient in our original interpolated ϕ^0), more work is needed in resolving the projected gradient descent step to ensure quick convergence of secant method. The zero-isocontour shown in Figure 1.15 is the union of two overlapping circles. Like in Figure 1.13 the gradient is fairly smooth and thus requires less computation to successfully converge in gradient descent. In Figure 1.16 we demonstrate our algorithm with a large number of local minima. This problem requires more projected gradient iterations than the simpler examples.

Figure 1.18 shows our method being used in a level set advection scheme using a

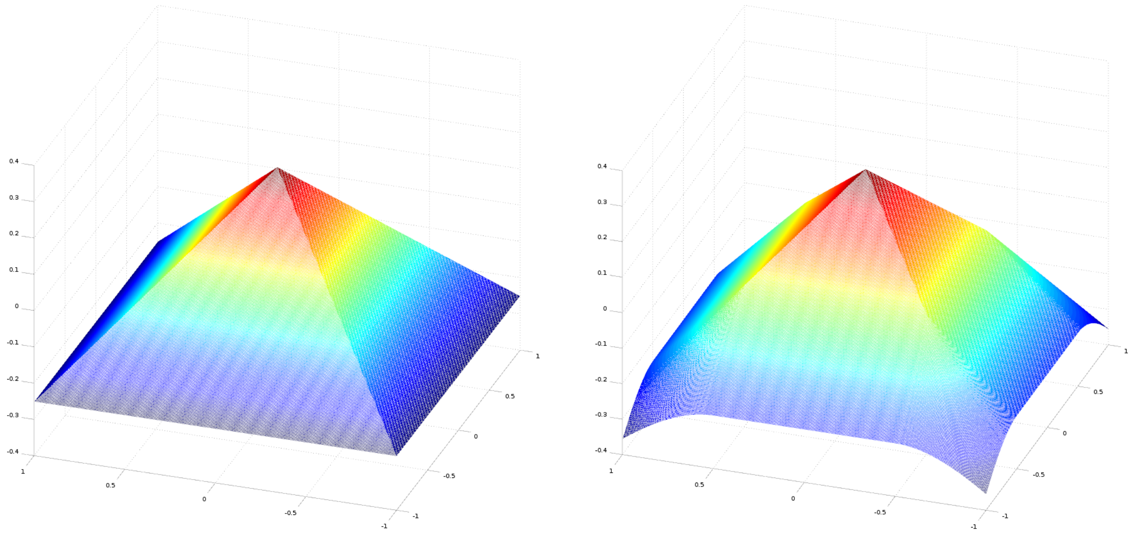

Figure 1.14: Square: the initial data is $\phi^0 = \mathtt{min}(.25 - |x - .5|, .25 - |y - .5|)$. The zero level is a square with side length .5 centered around (.5,.5)

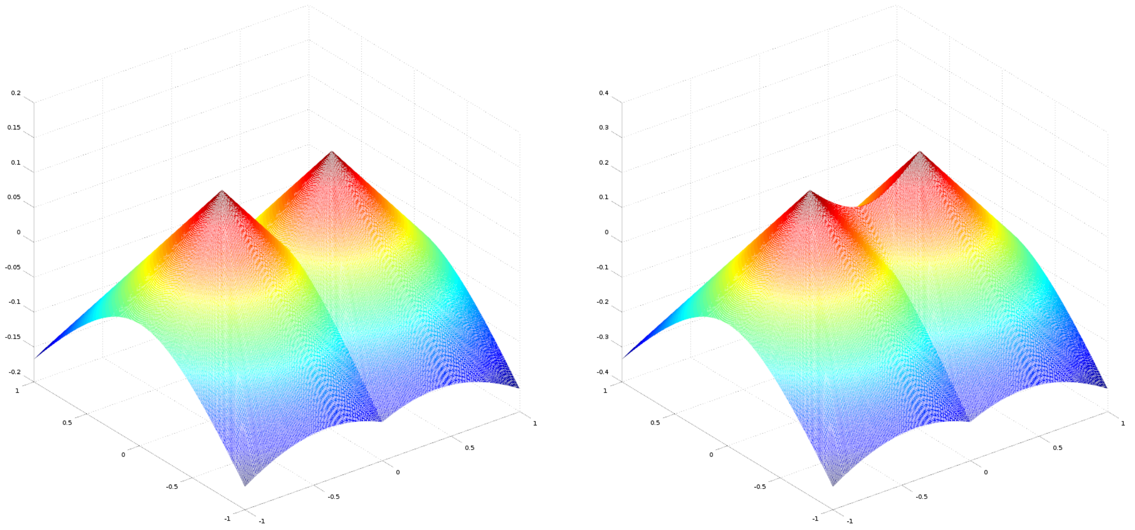

Figure 1.15: Union of circles: the initial data is $\phi^0 = \mathtt{max}((.25 - \|(.3, .5) - (x, y)\|_2), (.25 - \|(.7, .5) - (x, y)\|_2))$. The zero level set is a union of two circles both with radius .25, one centered at (.3,.5) and the other centered at (.7,.5)

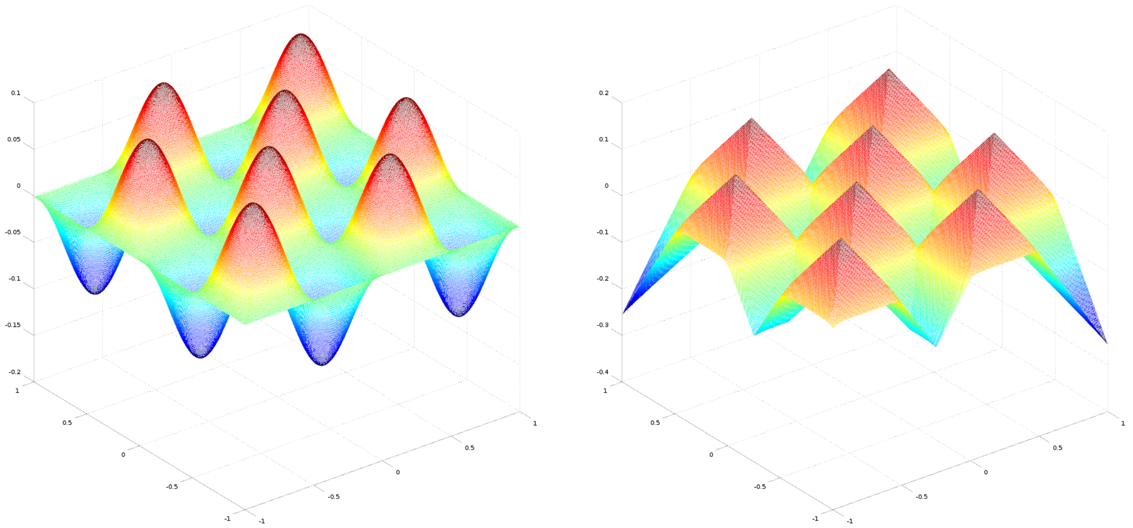

Figure 1.16: Many local minima: the objective function is $\phi^0 = \texttt{sin}(4 * \pi * x) * \texttt{sin}(4 * \pi * y) - .01$. The zero level set is a group of rounded squares that almost touch.

simple vortex test. Like previous problems, it was run on a 512X512 grid. For this problem the average time per frame for redistancing was 73.248ms.

1.9.1 Scaling

The results in table 1.1 were run with the square given in Figure 1.14 with the same parameters. The poor scaling at the low resolutions is due to not using all of the threads possible on the GPU.

For table 1.2 we ran our algorithm on the initial data given by 1.13 on a 1024X1024 grid. The problem was broken up into sub-domains and each domain was run separately on the GPU. Because our algorithm does not require the GPUs to communicate we can simulate multiple GPUs by running each sub-domain sequentially. We take the average over the total time as the time per GPU. We note that each GPU runtime was typically found to be within 5% of the average runtime. The scaling breaks down at high number of GPUs when we include the time it takes to transfer the data to the GPU. The transfer time takes approximatly 1.2 ms. If we ignore the time it takes to transfer data to the GPU we get a result that is close to being perfectly parallel. These results are shown in Figure 1.17

Size	Timing(ms)	average L2 error
32X32	3.303	$6.67 * 10^{-05}$
64X64	3.392	$1.5917 * 10^{-05}$
128X128	4.477	$3.8723 * 10^{-06}$
256X256	17.8400	$9.7391 * 10^{-07}$
512X512	67.533	$2.4624 * 10^{-07}$
1024X1024	274.216	$6.4343 * 10^{-08}$
2048X2048	1185.87	$2.3843 * 10^{-08}$

Table 1.1: Timing and error at different grid resolutions using a square as our zero level set. The average L2 error is calculated by calculating the L2 distance between the computed answer and the analytic solution

# GPUs	Total (w/)	per GPU (w/)	Total (w/o)	per GPU (w/o)
1	125.10	125.10	124.05	124.05
2	126.25	63.13	124.76	62.38
4	130.17	32.54	124.92	31.23
8	139.26	17.41	131.10	16.39
16	149.26	9.33	133.24	8.33
32	167.97	5.25	133.07	4.1585
64	206.07	3.22	134.85	2.11

Table 1.2: Timing in ms showing scaling in number of GPUs. The first two columns show the time it takes to run our problem when we include the timing cost of transferring the grid data to the GPU (approximatly 1.2ms), while the last two columns show the scaling without the cost of transferring data

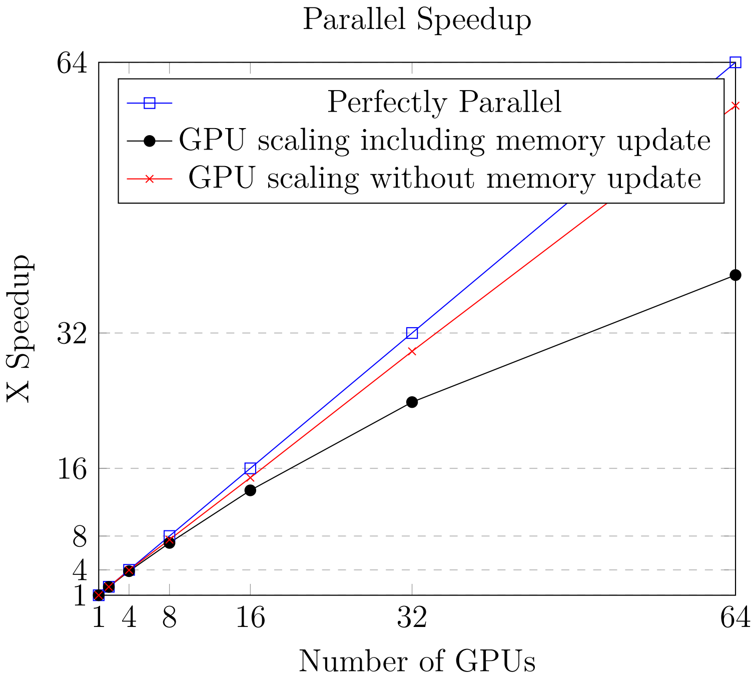

Figure 1.17: Parallel speed up is plotted both with and without including the cost of updating memory on the GPU. With 64 GPUs the memory update can take up to 33% of the runtime. However without the memory update (I.e. if the data is already on the GPU) the method scales simply.

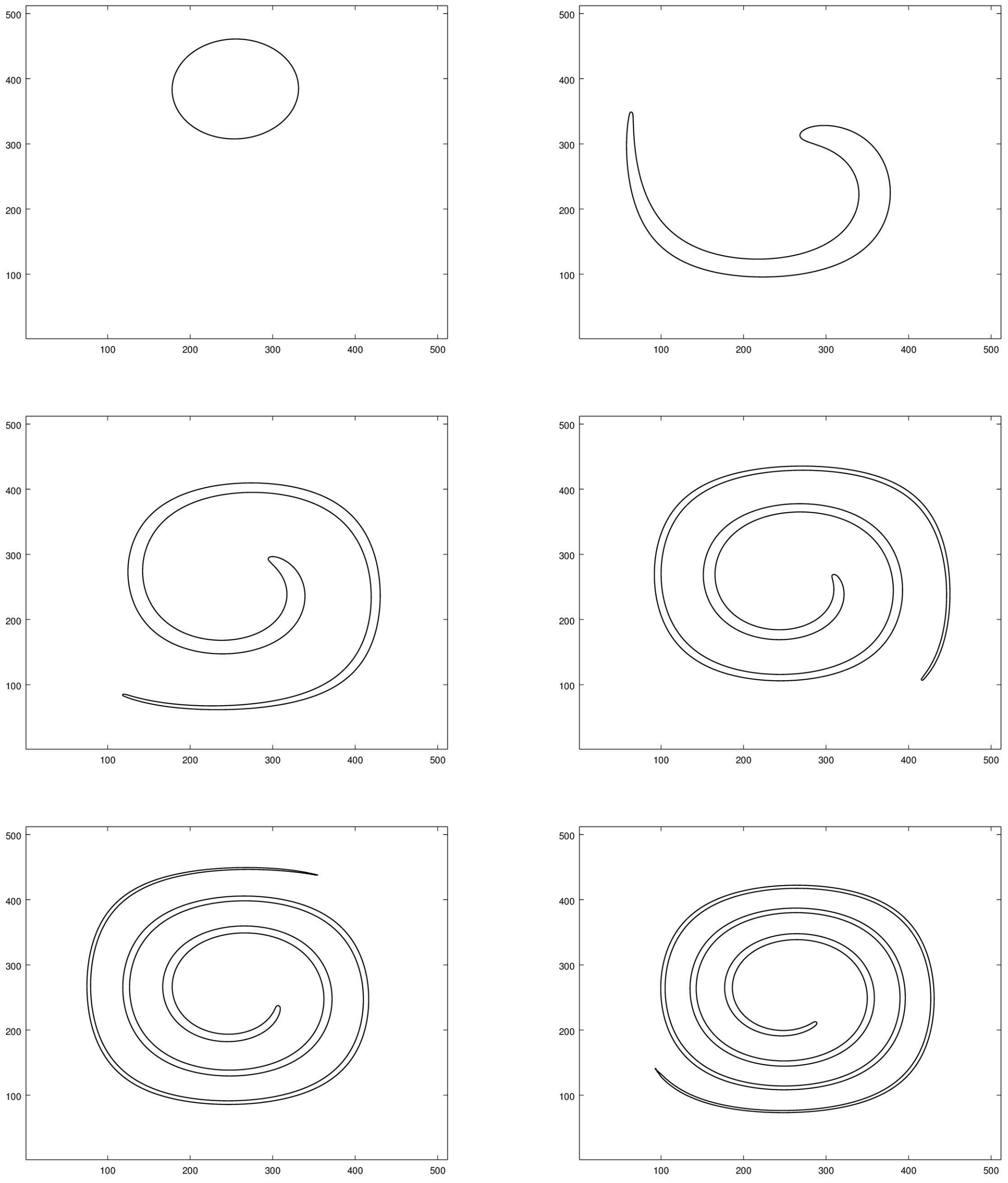

Figure 1.18: Practical application: vortex advection test at t = 0,1,2,3,4,5

CHAPTER 2

Continuum Mechanics

2.1 Introduction

Many materials can be modeled using the theory of elastoplasticity. From granular materials like sand and concrete [25], to solid materials like bone[26], sheet metal[27], or polymeric foams[28]. The idea of elastoplasticity is simple, when a small amount of force is applied to a material it will deform, and when the force is removed the material will return to its original state. Consider a slinky, when pulled on slightly it will expand, and when released will return to its compressed shape. However when a large amount of force is applied, there will be permanent deformation. Take for example a long straight metal rod, when twisted it will end up like a slinky, and when the force is removed the deformations will remain. Any force that when removed will allow a material to return to its original rest state is considered an elastic force. In this work we will model our elastoplastic materials with the large deformation theory. The idea being that the deformations are on the order of the size of the material. Once we have a deformation we can break it up multiplicatively into a plastic and elastic portion[29].

The elastic portion can be modeled in many ways, but we will assume we have a hyperelastic material. This simply means that a strain energy density function W exists that relates the stress ω in the material to the derivative of W with respect to the strain. Our work will be primarily concerned with the treatment of the plastic portion. The plastic deformations are modeled by the choice of the yield criterion that defines a yield function and a flow rule.

The yield function captures the limit of stress at which further forces on the object will lead to permanent deformations. While the yield functions are not easily derived from physical properties, they can be hypothesized based off of stress and strain data from real materials. Previous work has given different yield functions that represent the behavior in different classes of materials. The von Mises yield criterion can effectively model metals and other ductile materials[30]. The Mohr-Coulomb yield criterion models granular materials such as sand and concrete[31]. The Drucker-Prager yield criterion also models granular materials, but is smoother so is better suited for most numerical simulations[32].

The flow rule, or plastic flow rule, is used to determine what portion of the deformation will be considered plastic when the deformation is determined to be on the boundary of the yield surface. There are two ways to define the plastic flow, the first is with an associative flow rule. An associative flow rule simply states that the part of the deformation aligned with the gradient of the yield function will be considered plastic, and the rest elastic. An associative flow rule will enforce the second law of thermodynamics on the system. A non associative flow rule is any rule that does not follow the previous property. While care must be taken to avoid adding or losing energy with a non associative flow rule, they can be used to avoid non-physical results in the simulation such as volume loss or gain.

This flow rule can be viewed as a projection back to the zero-isocontour of the yield surface. However depending on our coordinates, this projection will not nescassarily be a right angle projection. Instead we can use an oblique projection where the angle between the projection and the set can vary from a right angle. While working with the Hopf-Lax formulation of the previous problem we discovered that with a modification of the Hamilton-Jacobi equation we could use this method to obliquely project to sets.

In the following chapter we will derive the properties surrounding large deformation theory. We follow the work of Gonzalez and Stuart [33] and Bonet and Wood [34], and for further details we refer the reader to those sources. We will derive the associative flow rule and show how it can be viewed as an oblique projection back onto the yield surface. Finally we will show results with both normal and non-physical yield surfaces using our method as the flow mapping.

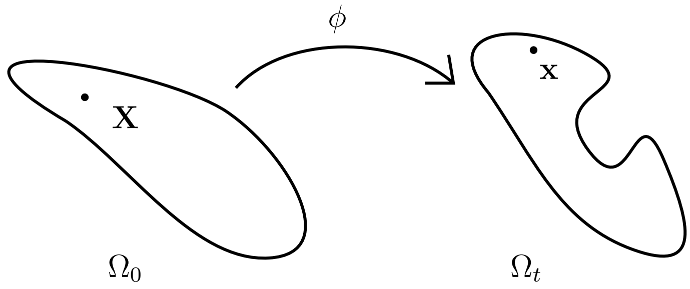

Figure 2.1: There exists a map ϕ that transforms the material state (left) into the spatial or current state (right).

2.2 Elastoplastic deformations

We consider a material in its rest configuration Ω_0, this will also be called the material configuration. A point $\mathbf{X} \in \Omega_0$ is considered to be a point in material space. We will call $\mathbf{X}$ both a material point and a particle. Let Ω_t be the evolution of Ω_0 after forces have been applied up to time t. We will call ϕ the deformation map that tells us where a point $\mathbf{X} \in \Omega_0$ has moved to at time t. This will be given by $\mathbf{x} = \phi(\mathbf{X}, t)$ and $\mathbf{x} \in \Omega_t$. We will call Ω_t the spatial or current configuration. You can see this relation between Ω_0 and Ω_t in Figure 2.2

The deformation gradient $\mathbf{F}(\mathbf{X}, t) = \frac{\partial \phi}{\partial \mathbf{X}}$ is used to describe the local deformation of a local neighborhood around $\mathbf{X}$. Essentially, given an infinitesimal vector $d\mathbf{X}$, we can calculate the vector it is transformed into by ϕ as $d\mathbf{x} = \mathbf{F}(\mathbf{X}, t)d\mathbf{X}$. In addition, we can calculate the change in volume of an infinitesimal region around $\mathbf{X}$ by using $J = \det \mathbf{F}$. $J = 1$ is the case when there is no volume change. When $J > 1$ then the neighborhood has increased in volume, similarly when $J < 1$ the neighborhood has decreased in volume.

In a material with no plastic deformation, if we can define the elastic potential, or stored strain engergy function ψ only using $\mathbf{F}$ then the material is considered to be hyperelastic.

2.3 Conservation of mass

For a general elastoplastic material we can describe the state of the material using its density $\rho(\mathbf{x}, t)$ and its velocity $\mathbf{v}(\mathbf{x}, t)$. Take an arbitrary volume V bounded by a surface

S that is fixed in space. The mass inside of the volume is given by

$$\int_V \rho dV$$

for conservation of mass to hold, we will assume that the decrease of mass in the fixed volume V

$$-\frac{d}{dt}\int_V \rho dV = -\int_V \frac{\partial \rho}{\partial t} dV \tag{2.1}$$

is equal to the rate of mass flux out of V

$$\int_S \rho \mathbf{v} \cdot d\mathbf{S} = \int_V \nabla \cdot (\rho \mathbf{v}) dV \tag{2.2}$$

Since Equations (2.1) and (2.2) must be equal for any volume V. As such, conservation of mass can be written as

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = 0 \tag{2.3}$$

Equation (2.3) works with conservation of mass at a specific location in space. However sometimes we want to work in material space. For any quantity $f = f(\mathbf{x}, t)$ we can take two different time derivatives. The first is $\frac{\partial f}{\partial t}$ which is the rate of change in f at a fixed point in space. We can also ask for the rate of change of a particular point in the material as it moves through space (given by $\mathbf{x}(t)$)

$$\begin{aligned} \frac{Df}{Dt} &= \frac{d}{dt} f(x_1(t), x_2(t), x_3(t), t) & (2.4)\\ &= \frac{\partial f}{\partial t} + \frac{dx_1}{dt}\frac{\partial f}{\partial x_1} + \frac{dx_2}{dt}\frac{\partial f}{\partial x_2} + \frac{dx_3}{dt}\frac{\partial f}{\partial x_3} & (2.5)\\ &= \frac{\partial f}{\partial t} + \mathbf{v} \cdot \nabla f & (2.6) \end{aligned}$$

As such we can rewrite the conservation of mass laws as

$$\begin{aligned} \frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) &= \frac{\rho \partial}{\partial t} + \rho \cdot \nabla \mathbf{v} + \mathbf{v} \cdot \nabla \rho & (2.7)\\ &= \frac{D\rho}{Dt} + \rho \nabla \cdot \mathbf{v} = 0 & (2.8) \end{aligned}$$

2.4 Conservation of momentum

Take a volume V bounded by a surface **S** that moves with the flow. Then its momentum is

$$\int_V DV \rho \mathbf{v} \tag{2.9}$$

so the rate of change of momentum is

$$\frac{d}{dt}\int_V dV\rho\mathbf{v} = \int_V dV\rho\frac{D\mathbf{v}}{Dt} \tag{2.10}$$

because the surface moves with the flow the mass contained by the volume V is constant. By Newton's second law, the change in momentum must equal the net forces on the volume. The forces we will be dealing with are $\mathbf{g}$ the force due to gravity and σ the force due to internal stress. This internal stress will be defined later. Given these two forces, the total force will be

$$\int_V dV\rho\mathbf{g} + \int_S \sigma\cdot d\mathbf{S} = \int_V dV(\rho\mathbf{g} + \nabla\cdot\sigma) \tag{2.11}$$

Again since this must be true for all V we get

$$\rho\frac{D\mathbf{v}}{Dt} = \rho\mathbf{g} + \nabla\cdot\sigma \tag{2.12}$$

2.5 Stress and strain

Stress and strain are two quantities that are useful to describe the state of the material. In this section we will derive the stresses and strains we used, as well as others in order to give a more complete picture. We begin by defining the deformation gradient as it will be needed to study the strain.

2.5.1 Deformation gradient

Suppose we have the material rest state of Ω_0 at time $t = 0$. We wish to define how the material has deformed moving from time $t = 0$ to time $t = s$. There exists ϕ such that

$$\forall\ \mathbf{x}_s \in \Omega_s\ \exists\ \mathbf{X}\ \texttt{s.t.}\ \ \mathbf{x}_s = \phi(\mathbf{X}, s)$$

I.e. ϕ is a mapping between Ω_0 and Ω_s for any time $t = s$. Suppose we have a point $\mathbf{X}_P \in \Omega_0$ and a point $\mathbf{X}_Q$ in a neighborhood of $\mathbf{X}$. These points translate to

$$\mathbf{x}_p = \phi(\mathbf{X}_P, t)\mathbf{x}_q = \phi(\mathbf{X}_Q, t)$$

If we define the vector $D\mathbf{X}$ and $D\mathbf{x}$ as

$$d\mathbf{X} = \mathbf{X}_Q - \mathbf{X}_P$$

$$d\mathbf{x} = \mathbf{x}_q - \mathbf{x}_p$$

these are related by

$$d\mathbf{x} = \mathbf{x}_q - \mathbf{x}_p = \phi(\mathbf{X}_P + d\mathbf{X}, t) - \phi(\mathbf{X}_P) = \frac{\partial \phi}{\partial \mathbf{X}} d\mathbf{X}$$

We then define the deformation gradient tensor $\mathbf{F}$ as

$$\mathbf{F} = \frac{\partial \phi}{\partial \mathbf{X}}$$

The deformation gradient is useful because it captures rotation, stretching, and shear forces.

2.5.2 Strain

Strain, or deformation, is used as a way to capture the stretching of local neighborhoods from Ω_0 to Ω_t. There are many different ways to define strain, but for our work in elastoplasticy the most useful are the elastic left Cauchy-Green deformation tensor and the plastic right Cauchy-Green deformation tensor. In order to motivate their definitions we first ignore plasticity.

The left Cauchy-Green deformation tensor is defined as

$$\mathbf{b} = \mathbf{F}\mathbf{F}^T \tag{2.13}$$

and the right Cauchy-Green deformation tensor is defined as

$$\mathbf{C} = \mathbf{F}^T\mathbf{F} \tag{2.14}$$

Both of these tensors are symmetric and positive definite. In order to see the motivation consider the polar decomposition of the deformation gradient

$$\mathbf{F} = \mathbf{R}\mathbf{U} = \mathbf{V}\mathbf{R} \tag{2.15}$$

This decomposition breaks down the stretch and rotation from $\mathbf{F}$ into two separate parts. However, both $\mathbf{b}$ and $\mathbf{C}$ only consider stretch. We refer the reader to Lubarda[35, 36] for work considering rotation in the deformation gradient and strains.

$$\mathbf{b} = \mathbf{F}\mathbf{F}^T = \mathbf{V}^2 \tag{2.16}$$

$$\mathbf{C} = \mathbf{F}^T\mathbf{F} = \mathbf{U}^2 \tag{2.17}$$

The difference between the two Cauchy-Green tensors is that $\mathbf{b}$ operates on the spatial vectors, and $\mathbf{C}$ operates on material vectors. Similar to above we define the elastic left Cauchy-Green tensor and the plastic right Cauchy-Green tensor as

$$\mathbf{b}_E = \mathbf{F}_E\mathbf{F}_E^T \tag{2.18}$$

$$\mathbf{C}_P = \mathbf{F}_P^T\mathbf{F}_P \tag{2.19}$$

as before, $\mathbf{b}_E$ is a map over spatial coordinates, and $\mathbf{C}_P$ is a map over material coordinates. One nice property of these tensors is that they are invariant under the rotation. As such, any stresses defined off of them will also be invariant to rotation.
The only other strain we will examine is the Hencky strain

$$\epsilon = \frac{1}{2}\log \mathbf{b} \tag{2.20}$$

and the elastic Hencky strain is given by

$$\epsilon_E = \frac{1}{2}\log \mathbf{b}_E \tag{2.21}$$

The Hencky strain will be useful later to simplify some expressions.

2.5.3 Stress

The deformation gradient tells us how a material has changed, but does not tell us what forces the material is experiencing. In order to understand these forces we will derive the Cauchy Stress σ.

We start by assuming that the stresses in a material Ω can be nonuniform. This assumption is not hard to make, for example if you apply a force to a small section of a wood block sitting on the ground, you would expect the stresses near the point of force to be higher than those far away. Take a point $\mathbf{p}$ in the material Ω if we cut the material by a plane $\mathbf{P}$ with normal $\mathbf{n}$ passing through $\mathbf{p}$ we can calculate the force acting on the material through that plane $F_{\mathbf{p}}$. By shrinking the plane($\delta\mathbf{P}$ being the area of the plane we are working with) we can calculate the traction force $\mathbf{t}$ corresponding to a given direction $\mathbf{n}$

$$\mathbf{t}(\mathbf{n}) = \lim_{\delta\mathbf{P}\to 0} \frac{F_{\mathbf{p}}}{\delta\mathbf{P}} \tag{2.22}$$

We note that $\mathbf{t}(\mathbf{n})$ is a vector containing the direction and amount of force being applied from a given normal. The direction may vary with changing $\mathbf{n}$ but the following property must hold

$$\mathbf{t}(-\mathbf{n}) = -\mathbf{t}(\mathbf{n})$$

In fact, Cauchy's Law states that there exists a stress tensor σ which maps $\mathbf{n}$ to $\mathbf{t}$ by

$$\mathbf{t} = \sigma\mathbf{n}$$

To prove this, take an infinitesimally small tetrahedra with one vertex on the origin, and the other three vertices along the major axis defined by the unit vectors $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3$. Let da be the area of the face defined by the non origin points with normal $\mathbf{n}$, and da_i be the area of faces with normals $\mathbf{e}_i$. dv is the volume of the tetrahedra, and $\mathbf{f}$ is the force per unit volume acting on our point $\mathbf{p}$. Using Newton's second law we have

$$\mathbf{t}(\mathbf{n})da + \sum_{i=1}^{3} \mathbf{t}(-\mathbf{e}_i)da_i + \mathbf{f}dv = 0 \tag{2.23}$$

now since $da_i = (\mathbf{n} \cdot \mathbf{e}_i)da$ we get

$$\begin{aligned}
\mathbf{t}(\mathbf{n}) &= -\sum_{i=1}^{3} \mathbf{t}(-\mathbf{e}_i)\frac{da_i}{da} - \mathbf{f}\frac{dv}{da} && (2.24)\\
&= \sum_{i=1}^{3} \mathbf{t}(\mathbf{e}_i)(\mathbf{n} \cdot \mathbf{e}_i) && (2.25)\\
&= \sum_{ij=1}^{3} \sigma_{ij}(\mathbf{e}_j \cdot \mathbf{n})\mathbf{e}_i && (2.26)
\end{aligned}$$

where the force term $\mathbf{f}$ disappears because $\frac{dv}{da} \to 0$ as we shrink the tetrahedra. The rest is simply the result of tensor calculation

$$\begin{aligned}
\mathbf{t}(\mathbf{n}) &= \sum_{ij=1}^{3} \sigma_{ij}(\mathbf{e}_j \cdot \mathbf{n})\mathbf{e}_i && (2.27)\\
&= \sum_{ij=1}^{3} \sigma_{ij}(\mathbf{e}_i \otimes \mathbf{e}_j)\mathbf{n} && (2.28)\\
&= \left[\sum_{ij=1}^{3} \sigma_{ij}(\mathbf{e}_i \otimes \mathbf{e}_j)\right]\mathbf{n} && (2.29)\\
&= \sigma\mathbf{n} && (2.30)\\
& && (2.31)
\end{aligned}$$

where

$$\sigma = \sum_{ij=1}^{3} \sigma_{ij}(\mathbf{e}_i \otimes \mathbf{e}_j) \tag{2.32}$$

The Cauchy stress is sometimes called the true stress. This is because it is a measure of the force per unit area of the current, deformed configuration. It can be useful to consider other measures of stress for simulation purposes.

The Kirchhoff stress tensor τ is simply the Cauchy stress scaled by the volume change compared to the original configuration

$$\tau = J\sigma \tag{2.33}$$

where J is the Jacobian of $\mathbf{F}$ the deformation gradient.

The first Piola-Kirchhoff stress tensor $\mathbf{P}$ relates the forces acting in the deformed configuration to the surface element in the original reference configuration. Consider an element of a surface in the reference configuration defined by $\mathbf{N}dS$, where dS is the area of the surface, and $\mathbf{N}$ is the unit normal. After deformation this element will be deformed to $\mathbf{n}ds$ where ds is the area of the deformed element, and $\mathbf{n}$ is the new normal. Suppose we have a force $d\mathbf{f}$ acting on the element in the current(deformed) configuration. Then using the cauchy stress we have

$$d\mathbf{f} = \sigma\mathbf{n}ds \tag{2.34}$$

we define the first Piola-Kirchhoff stress tensor $\mathbf{P}$ by

$$d\mathbf{f} = \mathbf{P}\mathbf{N}dS \tag{2.35}$$

these two stresses are related by

$$\mathbf{P} = J\sigma\mathbf{F}^{-T} \tag{2.36}$$

Whereas the first Piola-Kirchhoff stress relates forces in the deformed configuration to areas in the reference configuration, the second Piola-Kirchhoff stress($\mathbf{S}$) relates forces in the reference configuration to areas in the reference configuration. It can be calculated by taking the force vector in the current deformed configuration $d\mathbf{f}$ and finding the corresponding vector in the undeformed configuration $d\bar{\mathbf{f}} = \mathbf{F}^{-1}d\mathbf{f}$ it can be written as

$$\mathbf{S} = J\mathbf{F}^{-1}\sigma\mathbf{F}^{-T} \tag{2.37}$$

Worth noting is that both the Cauchy stress and the second Piola-Kirchhoff stresses are symmetric. However the first Piola-Kirchhoff stress is not symmetric due to having to relate the reference configuration and the current configuration.

2.6 Plasticity

In elastoplasticity we assume that when too large a force is applied to a material it will undergo both plastic and elastic deformation. This means that we must have a maximal attainable stress allowed in a material before plastic deformation occurs. We will call this the yield condition, i.e. the point in which the material starts to yield under load. This is the point at which the material is no longer able to ”resist” the deformation elastically and any further deformation becomes permanent.

We can think of the yield condition as defining a region of stress space as allowable. We start with a yield surface, which is a region of stress space where the elastic stress is allowed to be. We assume that we can construct a yield function F as a scaler valued function of stress, such that $F \leq 0$ implies the stress is allowable, and $F > 0$ implies the stress is non physical and not allowed.

Suppose we have three quantities F the yield function, $\dot{F}$ its rate of change and τ the stress it is defined over. Note while we use τ here, the yield function can be defined over any measure of stress. Then we have three cases

- *Case I* $F < 0 \implies \tau$ is inside the yield surface and the deformation is purely elastic
- *Case II* $F = 0\ ,\ \dot{F} < 0 \implies \tau$ is on the yield surface, but is moving into the allowed domain, thus the deformation is still only elastic
- *Case III* $F = 0\ ,\ \dot{F} = 0 \implies \tau$ is on the yield surface and staying on it, therefore some of the deformation must be plastic.

Note that $F > 0$ and $F = 0\ ,\ \dot{F} > 0$ are non physical and are thus not allowed.

Case III is the difficult case to handle. While the first two cases simply state that the stress is elastic, the last case requires some modification of the stress in order to maintain feasibility. Once a stress τ is on the yield surface, further forces applied to the surface will not cause τ to move elastically. Instead some of that force must be transfered into the plastic deformation. In order to decide how that force is transfered, we need a plastic flow rule.

2.6.1 Plastic flow

Once we have a yield surface we simply need to know how the elastic stress ”flows” along the yield surface. While previously we said that F was a function of stress, we note that stress is a function of strain. For the following derivations it will be easier to first start with strain before moving back to stress. As such we will consider F to be a function of the elastic left Cauchy-Green strain $\mathbf{b}_E$. We will use a plastic flow rule to determine how $\dot{\mathbf{b}_E}$ can be determined from $\dot{F}$.

A plastic flow rule needs to satisfy the following conditions.

1. Yield condition
2. Principle of maximum plastic dissipation
3. Second law of thermodynamics

For the yield condition we simply state that in order to activate plastic flow, the stress/strain must be on the yield surface. The other two conditions will be used to determine how to arrive at an appropriate solution.

2.7 Rates of plastic flow

We first start by deriving $\dot{\mathbf{b}_E}$ from $\dot{F}$. In order to do so we need to derive the rates of change of the elastic strain and the plastic strain.

$$\begin{aligned} \dot{\mathbf{F}} &= \dot{\mathbf{F}}_E \mathbf{F}_P + \mathbf{F}_E \dot{\mathbf{F}}_P \qquad (2.38) \\ \dot{\mathbf{F}}_E &= \dot{\mathbf{F}} \mathbf{F}_P^{-1} - \mathbf{F}_E \dot{\mathbf{F}}_P \mathbf{F}_P^{-1} \\ \dot{\mathbf{F}}_P &= \mathbf{F}_E^{-1} \dot{\mathbf{F}} - \mathbf{F}_E^{-1} \dot{\mathbf{F}}_E \mathbf{F}_P . \end{aligned}$$

using $\mathbf{b}_E = \mathbf{F}_E\mathbf{F}_E^T$ we can see

$$\dot{\mathbf{b}}_E = \dot{\mathbf{F}}_E\mathbf{F}_E^T + \mathbf{F}_E\dot{\mathbf{F}}_E^T$$

$$\dot{\mathbf{b}}_E = \dot{\mathbf{F}}\mathbf{F}_P^{-1}\mathbf{F}_E^T - \mathbf{F}_E\dot{\mathbf{F}}_P\mathbf{F}_P^{-1}\mathbf{F}_E^T + \mathbf{F}_E\mathbf{F}_P^{-T}\dot{\mathbf{F}}^T - \mathbf{F}_E\mathbf{F}_P^{-T}\dot{\mathbf{F}}_P^T\mathbf{F}_E^T \tag{2.39}$$

$$\dot{\mathbf{b}}_E = \dot{\mathbf{F}}\mathbf{F}_P^{-1}\mathbf{F}_E^{-1}\mathbf{F}_E\mathbf{F}_E^T + \mathbf{F}_E\mathbf{F}_E^T\mathbf{F}_E^{-T}\mathbf{F}_P^{-T}\dot{\mathbf{F}}^T - \mathbf{F}_E\dot{\mathbf{F}}_P\mathbf{F}_P^{-1}\mathbf{F}_E^T - \mathbf{F}_E\mathbf{F}_P^{-T}\dot{\mathbf{F}}_P^T\mathbf{F}_E^T \tag{2.40}$$

$$\dot{\mathbf{b}}_E = \dot{\mathbf{F}}\mathbf{F}^{-1}\mathbf{b}_E + \mathbf{b}_E\mathbf{F}^{-T}\dot{\mathbf{F}}^T - \mathbf{F}_E\mathbf{F}_P\mathbf{F}_P^{-1}\mathbf{F}_P^{-T}\mathbf{F}_P^T\dot{\mathbf{F}}_P\mathbf{F}_P^{-1}\mathbf{F}_P^{-T}\mathbf{F}_P^T\mathbf{F}_E^T - \mathbf{F}_E\mathbf{F}_P^{-T}\dot{\mathbf{F}}_P^T\mathbf{F}_P^T\mathbf{F}_E^T \tag{2.41}$$

$$\dot{\mathbf{b}}_E = \frac{\partial \mathbf{v}}{\partial \mathbf{x}}\mathbf{b}_E + \mathbf{b}_E\frac{\partial \mathbf{v}}{\partial \mathbf{x}}^T + \mathbf{F}\mathbf{C}_p^{-1}\mathbf{F}_P^T\dot{\mathbf{F}}_P\mathbf{C}_p^{-1}\mathbf{F} - \mathbf{F}\mathbf{C}_p^{-1}\dot{\mathbf{F}}_P^T\mathbf{F}_P\mathbf{C}_p^{-1}FF^T \tag{2.42}$$

$$\dot{\mathbf{b}}_E = \frac{\partial \mathbf{v}}{\partial \mathbf{x}}\mathbf{b}_E + \mathbf{b}_E\frac{\partial \mathbf{v}}{\partial \mathbf{x}}^T + \mathcal{L}_{\mathbf{v}}\mathbf{b}_E \tag{2.43}$$

where

$$\mathcal{L}_{\mathbf{v}}\mathbf{b}_E = \mathbf{F}\dot{\mathbf{C}}_p^{-1}\mathbf{F}^T = -\mathbf{F}\mathbf{C}_p^{-1}\dot{\mathbf{C}}_p\mathbf{C}_p^{-1}\mathbf{F}^T = -\mathbf{F}\mathbf{C}_p^{-1}\dot{\mathbf{F}}_P^T\mathbf{F}_P\mathbf{C}_p^{-1}\mathbf{F}^T - \mathbf{F}\mathbf{C}_p^{-1}\mathbf{F}_P^T\dot{\mathbf{F}}_P\mathbf{C}_p^{-1}\mathbf{F}^T \tag{2.44}$$

and

$$\dot{\mathbf{F}} = \frac{\partial}{\partial t}\left(\frac{\partial \mathbf{x}}{\partial \mathbf{X}}\right) = \frac{\partial}{\partial \mathbf{X}}\left(\frac{\partial \mathbf{x}}{\partial t}\right) = \frac{\partial \mathbf{v}}{\partial \mathbf{X}} = \frac{\partial \mathbf{v}}{\partial \mathbf{x}}\cdot\frac{\partial \mathbf{x}}{\partial \mathbf{X}} = \frac{\partial \mathbf{v}}{\partial \mathbf{x}}\cdot\mathbf{F} \tag{2.45}$$

and $\mathbf{C}_p = \mathbf{F}_P^T\mathbf{F}_P$ is the plastic left Cauchy-Green strain. It is convenient to use the notation

$$\dot{\mathbf{b}}_E = \dot{\mathbf{b}}_E|_{\dot{\mathbf{F}}_P=\mathbf{0}} + \mathcal{L}_{\mathbf{v}}\mathbf{b}_E,\ \dot{\mathbf{b}}_E|_{\dot{\mathbf{F}}_P=\mathbf{0}} = \frac{\partial \mathbf{v}}{\partial \mathbf{x}}\mathbf{b}_E + \mathbf{b}_E\frac{\partial \mathbf{v}}{\partial \mathbf{x}}^T \tag{2.46}$$

where $\dot{\mathbf{b}}_E|_{\dot{\mathbf{F}}_P=\mathbf{0}}$ is the strain in the absence of plasticity. We first note that $\dot{\mathbf{b}}_E|_{\dot{\mathbf{F}}_P=\mathbf{0}}$ uses terms that we assume are already known. However $\mathcal{L}_{\mathbf{v}}\mathbf{b}_E = \mathbf{F}\dot{\mathbf{C}}_p^{-1}\mathbf{F}^T$ is based off of the rate of change of the plastic strain, which we do not know. As such we can view $\mathcal{L}_{\mathbf{v}}\mathbf{b}_E$ as the free variable in our plastic solve and write it as $\mathcal{L}_{\mathbf{v}}\mathbf{b}_E = -\gamma\mathbf{L}$ where $\mathbf{L}$ is an arbitrary direction, and γ is chosen such that $\dot{F} \leq 0$. In the next sections we will show how to choose $\mathbf{L}$ in order to satisfy the last two constraints.

In most cases the yield function F is a function of stress τ. τ is a function of the left Cauchy-Green strain $\mathbf{b}_E$, which is in itself a function of time. This means we can write $F(\tau(\mathbf{b}_E(t)))$, which means if we want to write its derivative with respect to time we get, by the chain rule,

$$\begin{aligned}\dot{\mathbf{F}}(\tau(\mathbf{b}_E(t))) &= \frac{\partial F}{\partial \tau}(\tau(\mathbf{b}_E(t))) : \frac{\partial \tau}{\partial \mathbf{b}_E}(\mathbf{b}_E(t)) : \frac{D\mathbf{b}_E}{Dt}(t) \\ &= \frac{\partial F}{\partial \tau}(\tau(\mathbf{b}_E(t))) : \frac{\partial \tau}{\partial \mathbf{b}_E}(\mathbf{b}_E(t)) : (\frac{\partial \mathbf{v}}{\partial \mathbf{x}}\mathbf{b}_E + \mathbf{b}_E\frac{\partial \mathbf{v}}{\partial \mathbf{x}}^T + \mathcal{L}_{\mathbf{v}}\mathbf{b}_E)\end{aligned} \tag{2.47}$$

Where (:) is the generalized dot product for matrices, i.e.

$$\mathbf{A} : \mathbf{B} = \sum_i \sum_j A_{ij} B_{ij}$$

We recall that $\mathcal{L}_{\mathbf{v}} \mathbf{b}_E = 0$ in the absence of plasticity. As such during period of only elastic deformation the rate of change can be defined as β

$$\beta = \frac{\partial F}{\partial \tau}(\tau(\mathbf{b}_E(t))) : \frac{\partial \tau}{\partial \mathbf{b}_E}(\mathbf{b}_E(t)) : (\frac{\partial \mathbf{v}}{\partial \mathbf{x}} \mathbf{b}_E + \mathbf{b}_E \frac{\partial \mathbf{v}}{\partial \mathbf{x}}^T) \tag{2.48}$$

If we combine this with the choice of $\mathcal{L}_{\mathbf{v}} \mathbf{b}_E = -\gamma \mathbf{L}$ then Equation (2.47) can be rewritten as

$$\dot{\mathbf{F}} = \beta - \gamma \frac{\partial F}{\partial \tau}(\tau(\mathbf{b}_E(t))) : \frac{\partial \tau}{\partial \mathbf{b}_E}(\mathbf{b}_E(t)) : \mathbf{L} \tag{2.49}$$

Now we can begin to define $\mathcal{L}_{\mathbf{v}} \mathbf{b}_E$ based off of the three cases listed above.

In *Case I* we have $F(\tau(\mathbf{b}_E(t))) < 0$ meaning there is no plastic deformation happening so $\mathcal{L}_{\mathbf{v}} \mathbf{b}_E = 0$

In *Case II* we have $F(\tau(\mathbf{b}_E(t))) = 0$ and $\dot{\mathbf{F}} < 0$ this means that the rate of change due to elasticity is enough to keep us in the allowed stress region, i.e. $\beta < 0$. This means that even with $\mathcal{L}_{\mathbf{v}} \mathbf{b}_E = 0$ the stress is still feasible, so again there is no plastic deformation.

In *Case III* we have $F(\tau(\mathbf{b}_E(t))) = 0$ and $\dot{\mathbf{F}} = 0$. There are two ways that this can happen. Either the elastic evolution of our material will keep the stress on the yield surface, i.e. $\beta = 0$, or it will try to bring the stress outside the yield surface ($\beta > 0$). The first situation with $\beta = 0$ requires no plastic deformation so again we can say $\mathcal{L}_{\mathbf{v}} \mathbf{b}_E = 0$. The second situation however requires some plastic flow in order to balance out β. While we have not yet decided on how to choose $\mathbf{L}$, once chosen γ is easy to define

$$\gamma = \frac{\beta}{\frac{\partial F}{\partial \tau}(\tau(\mathbf{b}_E(t))) : \frac{\partial \tau}{\partial \mathbf{b}_E}(\mathbf{b}_E(t)) : \mathbf{L}} \tag{2.50}$$

We can condense the previous cases into the following definition

$$\mathcal{L}_{\mathbf{v}} \mathbf{b}_E = \begin{cases} \mathbf{0} & \text{if } F(\tau(\mathbf{b}_E(t))) < 0 \text{ or } [F(\tau(\mathbf{b}_E(t))) = 0] \text{ and } \beta \leq 0] \\ -\gamma \mathbf{L} & \text{if } F(\tau(\mathbf{b}_E(t))) = 0 \text{ and } \beta > 0 \end{cases} \tag{2.51}$$

We haven't yet chosen the direction of flow $\mathbf{L}$. We note that there are many different choices of $\mathbf{L}$ that will allow us to satisfy the feasibility constraint. as long as $\frac{\partial F}{\partial \tau}(\tau(\mathbf{b}_E(t))) : \frac{\partial \tau}{\partial \mathbf{b}_E}(\mathbf{b}_E(t)) : \mathbf{L} \neq 0$. However we can use the second law of thermodynamics to give us a better idea of the best $\mathbf{L}$ to choose to more accurately represent physical materials.

2.7.1 Energy dissipation and stress/rate pairs

By the second law of thermodynamics, we know that the plastic flow should not decrease the entropy of the system. In other words, it must not instantaneously increase the rate of change of the total energy [33]. In purely elastic motion the total energy does not change. Physically we expect that plastic deformation should cause dissipation of energy and the total energy of the system should decrease[37, 38]. Using these properties we will derive a choice of **L**.

The energy at time t of the material in B^0 is

$$E(t; B^0) = E_K(t; B^0) + E_P(t; B^0) \tag{2.52}$$

$$E(t; B^0) = \int_{B^0} \frac{R(\mathbf{X}, 0)}{2} \left|\mathbf{V}(\mathbf{X}, t)\right|_2^2 d\mathbf{X} + \int_{B^0} \psi(\mathbf{F}_E(\mathbf{X}, t), \mathbf{F}_P(\mathbf{X}, t)) d\mathbf{X}. \tag{2.53}$$

Where $R(\mathbf{X}, t)$ is the Lagrangian mass density at time t, $\mathbf{V}(\mathbf{X}, t)$ is the velocity at a material point $\mathbf{X}$ and ψ is an energy density function to be defined later. The rate of change of the energy is

$$\begin{aligned} E'(t; B^0) = &\int_{B^0} R(\mathbf{X}, 0)\mathbf{V}(\mathbf{X}, t)\mathbf{A}(\mathbf{X}, t) d\mathbf{X} + \int_{B^0} \frac{\partial \psi}{\partial \mathbf{F}_E}(\mathbf{F}_E(\mathbf{X}, t), \mathbf{F}_P(\mathbf{X}, t)) : \dot{\mathbf{F}}_E(\mathbf{X}, t) d\mathbf{X} \\ &+ \int_{B^0} \frac{\partial \psi}{\partial \mathbf{F}_P}(\mathbf{F}_E(\mathbf{X}, t), \mathbf{F}_P(\mathbf{X}, t)) : \dot{\mathbf{F}}_P(\mathbf{X}, t) d\mathbf{X}. \end{aligned} \tag{2.54}$$

where $\mathbf{A}$ is the acceleration at a material point $\mathbf{X}$ and

$$\begin{aligned} \int_{B^0} \frac{\partial \psi}{\partial \mathbf{F}_E}(\mathbf{F}_E, \mathbf{F}_P) : \dot{\mathbf{F}}_P\ ] d\mathbf{X} = \\ \int_{B^0} \frac{\partial \psi}{\partial \mathbf{F}_E}(\mathbf{F}_E, \mathbf{F}_P) : \left(\dot{\mathbf{F}}\mathbf{F}_P^{-1} - \mathbf{F}_E\dot{\mathbf{F}}_P\mathbf{F}_P^{-1}\right) d\mathbf{X} = \\ \int_{B^0} \mathbf{P} : \left(\dot{\mathbf{F}} - \mathbf{F}_E\dot{\mathbf{F}}_P\right) d\mathbf{X} = \\ -\int_{B^0} \mathbf{V} \cdot \left(\nabla^{\mathbf{X}} \cdot \mathbf{P}\right) d\mathbf{X} + \int_{\partial B^0} \mathbf{V} \cdot (\mathbf{P}\mathbf{N})\, ds(\mathbf{X}) - \int_{B^0} \left(\mathbf{F}_E^T\mathbf{P}\right) : \dot{\mathbf{F}}_P d\mathbf{X} = \end{aligned}$$

with $\mathbf{P} = \frac{\partial \psi}{\partial \mathbf{F}_E}(\mathbf{F}_E, \mathbf{F}_P)\mathbf{F}_P^{-T}$ the first Piola Kirchoff Stress. Using $R(\mathbf{X}, 0)\mathbf{A}(\mathbf{X}, t) = (\nabla^{\mathbf{X}} \cdot \mathbf{P})(\mathbf{X}, t)$, which is conservation of linear momentum in Lagrangian view, with Equation (2.54) gives

$$\begin{aligned} E'(t; B^0) = \int_{\partial B^0} \mathbf{V} \cdot (\mathbf{P}\mathbf{N})\, ds(\mathbf{X}) - \int_{B^0} \left(\mathbf{F}_E^T\mathbf{P}\right) : \dot{\mathbf{F}}_P d\mathbf{X} + \\ \int_{B^0} \frac{\partial \psi}{\partial \mathbf{F}_P}(\mathbf{F}_E, \mathbf{F}_P) : \dot{\mathbf{F}}_P d\mathbf{X}. \end{aligned} \tag{2.55}$$

Note that $\left(\mathbf{F}_E^T\mathbf{P}\right) : \dot{\mathbf{F}}_P = \left(\mathbf{F}_E^T\frac{\partial\psi}{\partial\mathbf{F}_E}(\mathbf{F}_E, \mathbf{F}_P)\right) : \left(\dot{\mathbf{F}}_P\mathbf{F}_P^{-1}\right)$. The term

$$\mathbf{L}^P = \dot{\mathbf{F}}_P\mathbf{F}_P^{-1} \tag{2.56}$$

is called the plastic velocity gradient. Using this we can write the change in energy as

$$E'(t; B^0) = \int_{\partial B^0} \mathbf{V}\cdot(\mathbf{P}\mathbf{N})\, ds(\mathbf{X}) - \int_{B^0} \mathbf{M}^E : \mathbf{L}^P d\mathbf{X} + \tag{2.57}$$
$$\int_{B^0} \frac{\partial\psi}{\partial\mathbf{F}_P}(\mathbf{F}_E, \mathbf{F}_P) : \dot{\mathbf{F}}_P d\mathbf{X}.$$

where we define the Mandel stress $\mathbf{M}^E$ as

$$\mathbf{M}^E = \mathbf{F}_E^T\frac{\partial\psi}{\partial\mathbf{F}_E}(\mathbf{F}_E, \mathbf{F}_P). \tag{2.58}$$

The term $\int_{\partial B^0} \mathbf{V}\cdot(\mathbf{P}\mathbf{N})\, ds(\mathbf{X})$ is the rate of work done on B^0 at time t via contact with material external to the region.

2.7.2 Isotropy

Assume we have an energy density $\psi(\mathbf{F}_E, \mathbf{F}_P)$ that is isotropic. Due to isotropy ψ can be writen in the form of $\psi(\mathbf{F}_E, \mathbf{F}_P) = \hat{\psi}(I(\mathbf{F}_E), II(\mathbf{F}_E), III(\mathbf{F}_E))$

$$I(\mathbf{F}_E) = \mathrm{tr}(\mathbf{b}_E)$$
$$II(\mathbf{F}_E) = \mathrm{tr}(\mathbf{b}_E)^2 - \mathrm{tr}(\mathbf{b}_E^T\mathbf{b}_E)$$
$$III(\mathbf{F}_E) = \det(\mathbf{b}_E)$$

Where the invariants are defined to be the invariants of $\mathbf{b}_E$ due to the assumed isotropy of ψ.

$$\frac{\partial I}{\partial\mathbf{F}_E} = 2\mathbf{F}_E$$
$$\frac{\partial II}{\partial\mathbf{F}_E} = 2(I(\mathbf{F}_E)\mathbf{F}_E - \mathbf{b}_E\mathbf{F}_E)$$
$$\frac{\partial III}{\partial\mathbf{F}_E} = 2III(\mathbf{F}_E)\mathbf{F}_E^{-1}$$
$$\frac{\partial\psi}{\partial\mathbf{F}_E}(\mathbf{F}_E, \mathbf{F}_P) = \frac{\partial\hat{\psi}}{\partial I}\frac{\partial I}{\partial\mathbf{F}_E} + \frac{\partial\hat{\psi}}{\partial II}\frac{\partial II}{\partial\mathbf{F}_E} + \frac{\partial\hat{\psi}}{\partial III}\frac{\partial III}{\partial\mathbf{F}_E}$$
$$\frac{\partial\psi}{\partial\mathbf{F}_E}(\mathbf{F}_E, \mathbf{F}_P) = 2\left(\frac{\partial\hat{\psi}}{\partial I} + I\frac{\partial\hat{\psi}}{\partial II}\right)\mathbf{F}_E - 2\frac{\partial\hat{\psi}}{\partial II}\mathbf{b}_E\mathbf{F}_E + 2III\frac{\partial\hat{\psi}}{\partial III}\mathbf{F}_E^{-1}$$
$$\frac{\partial\psi}{\partial\mathbf{F}_E}(\mathbf{F}_E, \mathbf{F}_P) = \alpha(\mathbf{F}_E)\mathbf{F}_E + \beta(\mathbf{F}_E)\mathbf{b}_E\mathbf{F}_E + \gamma(\mathbf{F}_E)\mathbf{F}_E^{-T}$$
$$\boldsymbol{\tau} = \mathbf{P}\mathbf{F}^T = \frac{\partial\psi}{\partial\mathbf{F}_E}(\mathbf{F}_E, \mathbf{F}_P)\mathbf{F}_P^{-T}\mathbf{F}^T = \alpha(\mathbf{F}_E)\mathbf{b}_E + \beta(\mathbf{F}_E){\mathbf{b}_E}^2 + \gamma(\mathbf{F}_E)\mathbf{I}.$$

where α, β, γ are all scaler functions that depend on $\mathbf{F}_E$.

Note that $\boldsymbol{\tau}$ and $\mathbf{b}_E$ as well as $\boldsymbol{\tau}$ and ${\mathbf{b}_E}^{-1}$ commute in this case

$$\mathbf{b}_E\boldsymbol{\tau} = \mathbf{b}_E\boldsymbol{\tau},\ {\mathbf{b}_E}^{-1}\boldsymbol{\tau} = \boldsymbol{\tau}{\mathbf{b}_E}^{-1}. \tag{2.59}$$

$\boldsymbol{\tau}$ is the Kirchhoff stress and it is convenient to work with for some models. E.g. we can rewrite the plastic dissipation in terms of $\boldsymbol{\tau}$ since

$$\mathbf{M}^E : \mathbf{L}^P = \boldsymbol{\tau} : \left(\mathbf{F}_E\dot{\mathbf{F}}_P\mathbf{F}^{-1}\right). \tag{2.60}$$

Using the definitions in Equations (2.39) and (2.44) and

$$\mathcal{L}_{\mathbf{v}}\mathbf{b}_E\mathbf{b}_E^{-1} = -\mathbf{F}_E\mathbf{F}_P^{-T}\dot{\mathbf{F}}_P^T\mathbf{F}_E^{-1} - \mathbf{F}_E\dot{\mathbf{F}}_P\mathbf{F}^{-1} \tag{2.61}$$

we can conclude that in the case of isotropic energy density

$$\begin{aligned}
\boldsymbol{\tau} : \left(\mathcal{L}_{\mathbf{v}}\mathbf{b}_E\mathbf{b}_E^{-1}\right) &= -\boldsymbol{\tau} : \left(\mathbf{F}_E\mathbf{F}_P^{-T}\dot{\mathbf{F}}_P^T\mathbf{F}_E^{-1}\right) - \boldsymbol{\tau} : \left(\mathbf{F}_E\dot{\mathbf{F}}_P\mathbf{F}^{-1}\right) \\
&= -\mathrm{tr}\left(\boldsymbol{\tau}\mathbf{F}_E\mathbf{F}_P^{-T}\dot{\mathbf{F}}_P^T\mathbf{F}_E^{-1}\right) - \boldsymbol{\tau} : \left(\mathbf{F}_E\dot{\mathbf{F}}_P\mathbf{F}^{-1}\right) \\
&= -\mathrm{tr}\left(\boldsymbol{\tau}\mathbf{F}_E\mathbf{F}_E^T\mathbf{F}_E^{-T}\mathbf{F}_P^{-T}\dot{\mathbf{F}}_P^T\mathbf{F}_E^{-1}\right) - \boldsymbol{\tau} : \left(\mathbf{F}_E\dot{\mathbf{F}}_P\mathbf{F}^{-1}\right) \\
&= -\mathrm{tr}\left(\boldsymbol{\tau}\mathbf{b}_E\mathbf{F}^{-T}\dot{\mathbf{F}}_P^T\mathbf{F}_E^{-1}\right) - \boldsymbol{\tau} : \left(\mathbf{F}_E\dot{\mathbf{F}}_P\mathbf{F}^{-1}\right) \\
&= -\mathrm{tr}\left(\mathbf{b}_E\boldsymbol{\tau}\mathbf{F}^{-T}\dot{\mathbf{F}}_P^T\mathbf{F}_E^{-1}\right) - \boldsymbol{\tau} : \left(\mathbf{F}_E\dot{\mathbf{F}}_P\mathbf{F}^{-1}\right) \\
&= -\mathrm{tr}\left(\boldsymbol{\tau}\mathbf{F}^{-T}\dot{\mathbf{F}}_P^T\mathbf{F}_E^T\right) - \boldsymbol{\tau} : \left(\mathbf{F}_E\dot{\mathbf{F}}_P\mathbf{F}^{-1}\right) \\
&= -2\boldsymbol{\tau} : \left(\mathbf{F}_E\dot{\mathbf{F}}_P\mathbf{F}^{-1}\right)
\end{aligned} \tag{2.62}$$

Where much of the cancelling can be attributed to $\mathbf{b}_E$ and $\boldsymbol{\tau}$ being symmetric and

$$\mathrm{tr}(\mathbf{AB}) = \mathrm{tr}(\mathbf{BA})$$

2.7.3 Plastic dissipation rate without hardening

In summary, if we assume there is no energy added into the system via contact with material external to the region, and that ψ is defined only in terms of the elastic strain, the rate of energy release due to plasticity (with no hardening) can be written as

$$\int_{B^0} \dot{w}_p(\mathbf{X}, t)d\mathbf{X} \tag{2.63}$$

where

$$\dot{w}_p = \left(\mathbf{F}_E^T \mathbf{P}\right) : \dot{\mathbf{F}}_P = \mathbf{M}^E : \mathbf{L}^p = -\frac{1}{2}\boldsymbol{\tau} : \left(\mathcal{L}_{\mathbf{v}} \mathbf{b}_E \mathbf{b}_E^{-1}\right) \tag{2.64}$$

where the last equality only holds for isotropic energy density.

2.8 Associative

Assume we have no hardening, e.g.

$$\tilde{\psi}(\mathbf{F}_E) = \hat{\psi}(\frac{1}{2}\left({\mathbf{F}_E}^T \mathbf{F}_E - \mathbf{I}\right))$$

Thus

$$\mathbf{P} = \mathbf{F}_E \frac{\partial \hat{\psi}}{\partial \mathbf{E}^E}(\frac{1}{2}\left({\mathbf{F}_E}^T \mathbf{F}_E - \mathbf{I}\right))\mathbf{F}_P^{-T}$$

and the Mandel stress $\mathbf{M}^E$ satisfies

$$\mathbf{M}^E = \mathbf{F}_E^T \frac{\partial \psi}{\partial \mathbf{F}_E} = \mathbf{C}_E \frac{\partial \hat{\psi}}{\partial \mathbf{E}^E}. \tag{2.65}$$

If we choose $\mathbf{L}^P$ such that

$$\mathbf{M}^E : \mathbf{L}^P \geq \mathbf{M}^* : \mathbf{L}^P \tag{2.66}$$

for all admissible states of stress $\mathbf{M}^*$, then

1. If $\mathbf{M}^* = \mathbf{0}$ is an admissible state of stress, then

   $$E'(t; B^0) \leq \int_{\partial B^0} \mathbf{V} \cdot (\mathbf{P}\mathbf{N}) \, ds(\mathbf{X}) \tag{2.67}$$

   which says that the plasticity dissipates energy.

2. If the region of admissible $\mathbf{M}^*$ is (a) convex and (b) defined via $f(\mathbf{M}^*) \leq 0$ then $\mathbf{L}^P \in \partial f(\mathbf{M}^E)$ satisfies Equation (2.66).

Similarly, if we choose $-\frac{1}{2}\mathcal{L}_{\mathbf{v}} \mathbf{b}_E \mathbf{b}_E^{-1} \in \partial f$ we get an associative plastic flow when we write the yield surface in terms of $\boldsymbol{\tau}$: $f(\boldsymbol{\tau})$.

2.8.1 Yield surface and plastic flow

We will have plastic flow $\dot{\mathbf{F}}_P \neq \mathbf{0}$ when our stress is on the boundary of the feasible region, and without plasticity we would leave the region. In the case of isoptropy and a yield surface defined in terms of the Kirchhoff stress, then

$$\mathcal{L}_{\mathbf{v}}\mathbf{b}_E = -2\lambda\frac{\partial f}{\partial \boldsymbol{\tau}}(\boldsymbol{\tau})\mathbf{b}_E \tag{2.68}$$

where

- If $f(\boldsymbol{\tau}) < 0$ or $f(\boldsymbol{\tau}) = 0$ and $\alpha \leq 0$, then $\lambda = 0$.
- Otherwise if, $f(\boldsymbol{\tau}) = 0$ and $\alpha > 0$, then $\lambda = \frac{\alpha}{\beta}$

where

$$\alpha = \frac{\partial f}{\partial \boldsymbol{\tau}} : \frac{\partial \boldsymbol{\tau}}{\partial \mathbf{b}_E} : \dot{\mathbf{b}}_E|_{\dot{\mathbf{F}}_P=\mathbf{0}}, \ \beta = 2\frac{\partial f}{\partial \boldsymbol{\tau}} : \frac{\partial \boldsymbol{\tau}}{\partial \mathbf{b}_E} : \left(\frac{\partial f}{\partial \boldsymbol{\tau}}(\boldsymbol{\tau})\mathbf{b}_E\right). \tag{2.69}$$

2.8.2 Isoptropic yield surface

Suppose the yield surface function $f : \mathcal{V}^2_{\text{sym}} \to \mathbb{R}$ is isotropic: $f(\mathbf{V}\boldsymbol{\tau}\mathbf{V}^T) = f(\boldsymbol{\tau})$ for all rotations $\mathbf{V}$. Then as discussed in Appendix (§4.1.2), we can write $f(\boldsymbol{\tau}) = \hat{f}(\tau_1, \tau_2, \tau_3)$ where $\boldsymbol{\tau} = \sum_i \tau_i \mathbf{u}_i \otimes \mathbf{u}_i$ and $\frac{\partial f}{\partial \boldsymbol{\tau}}(\boldsymbol{\tau}) = \sum_i \frac{\partial \hat{f}}{\partial \tau_i}\mathbf{u}_i \otimes \mathbf{u}_i$. Therefore since $\boldsymbol{\tau}$ and $\mathbf{b}_E$ have the same eigenvectors

$$\frac{\partial f}{\partial \boldsymbol{\tau}}(\boldsymbol{\tau})\mathbf{b}_E = \sum_i \frac{\partial \hat{f}}{\partial \tau_i}\lambda^2_{Ei}\mathbf{u}_i \otimes \mathbf{u}_i \tag{2.70}$$

Furthermore using the properties of isotropic energy density,

$$\beta = 2\sum_{i,j}\frac{\partial \hat{f}}{\partial \tau_i}\tilde{C}_{ij}(\mathbf{b}_E)\frac{\partial \hat{f}}{\partial \tau_j}\lambda^2_{Ej} \tag{2.71}$$

where

$$\frac{\partial \boldsymbol{\tau}}{\partial \mathbf{b}_E}(\mathbf{b}_E) : \left(\sum_j \sigma_j \mathbf{u}_j \otimes \mathbf{u}_j\right) = \sum_{i,j}\tilde{C}_{ij}(\mathbf{b}_E)\sigma_j \mathbf{u}_i \otimes \mathbf{u}_i$$

for arbitrary $\sum_j \sigma_j \mathbf{u}_j \otimes \mathbf{u}_j$.

2.9 Hencky strain

The elastic Hencky strain $\boldsymbol{\epsilon}_E$ is defined as

$$\boldsymbol{\epsilon}_E = \frac{1}{2}\log\left(\mathbf{b}_E\right). \tag{2.72}$$

The rate of change of the elastic Hencky strain is given by

$$\dot{\boldsymbol{\epsilon}}_E = \left([\mathbf{B}](\mathbf{b}_E) \circ [\dot{\mathbf{b}}_E]\right)_{kl} \mathbf{u}_k \otimes \mathbf{u}_l \tag{2.73}$$

where $\mathbf{b}_E = \sum_i \lambda_i^{E^2} \mathbf{u}_i \otimes \mathbf{u}_i$, $\boldsymbol{\epsilon}_E = \sum_i \log\left(\lambda_i^E\right) \mathbf{u}_i \otimes \mathbf{u}_i$, $[\dot{\mathbf{b}}_E]_{ij} = \mathbf{u}_i \cdot \left(\dot{\mathbf{b}}_E \mathbf{u}_j\right)$ are the components of $\dot{\mathbf{b}}_E$ in the eigen basis and

$$[\mathbf{B}](\mathbf{b}_E) = \begin{pmatrix} \frac{1}{2\lambda_{E1}^2} & \frac{\log(\lambda_{E1})-\log(\lambda_{E2})}{\lambda_{E1}^2-\lambda_{E2}^2} & \frac{\log(\lambda_{E1})-\log(\lambda_{E3})}{\lambda_{E1}^2-\lambda_{E3}^2} \\ \frac{\log(\lambda_{E2})-\log(\lambda_{E1})}{\lambda_{E2}^2-\lambda_{E1}^2} & \frac{1}{2\lambda_{E2}^2} & \frac{\log(\lambda_{E2})-\log(\lambda_{E3})}{\lambda_{E2}^2-\lambda_{E3}^2} \\ \frac{\log(\lambda_{E3})-\log(\lambda_{E1})}{\lambda_{E3}^2-\lambda_{E1}^2} & \frac{\log(\lambda_{E3})-\log(\lambda_{E2})}{\lambda_{E3}^2-\lambda_{E2}^2} & \frac{1}{2\lambda_{E3}^2} \end{pmatrix}. \tag{2.74}$$

See Appendix (§4.1) and Equation (4.3) for the derivation. If we define the elastic potential as a function of the Hencky strain as

$$\psi(\mathbf{F}_E, \mathbf{F}_P) = \mu\boldsymbol{\epsilon}_E : \boldsymbol{\epsilon}_E + \frac{\lambda}{2}\mathrm{tr}\left(\boldsymbol{\epsilon}_E\right)^2 \tag{2.75}$$

then

$$\boldsymbol{\tau} = \boldsymbol{C}\boldsymbol{\epsilon}_E = 2\mu\boldsymbol{\epsilon}_E + \lambda\mathrm{tr}\left(\boldsymbol{\epsilon}_E\right)\mathbf{I}. \tag{2.76}$$

This can be written in terms of the eigen basis of $\mathbf{b}_E$ as

$$\boldsymbol{\tau} = \boldsymbol{C}\boldsymbol{\epsilon}_E = \sum_{i,j} \hat{C}_{ij} \log\left(\lambda_j^E\right) \mathbf{u}_i \otimes \mathbf{u}_i \tag{2.77}$$

with

$$[\hat{\mathbf{C}}] = \begin{pmatrix} 2\mu+\lambda & \lambda & \lambda \\ \lambda & 2\mu+\lambda & \lambda \\ \lambda & \lambda & 2\mu+\lambda \end{pmatrix}.$$

2.9.1 Yield surface and plastic rate of change

With this energy density, the rate of change of the elastic Hencky strain has the favorable property that its direction is simply related to the yield surface when it is written in terms of $\boldsymbol{\epsilon}_E$. Specifically, α and β in Equation (2.69) can be written as

$$\alpha = \frac{\partial f}{\partial \boldsymbol{\tau}} : \frac{\partial \boldsymbol{\tau}}{\partial \boldsymbol{\epsilon}_E} : \dot{\boldsymbol{\epsilon}}_E|_{\dot{\mathbf{F}}_P=\mathbf{0}}, \; \beta = 2\frac{\partial f}{\partial \boldsymbol{\tau}} : \frac{\partial \boldsymbol{\tau}}{\partial \boldsymbol{\epsilon}_E} : \left(\left([\mathbf{B}](\mathbf{b}_E) \circ [\frac{\partial f}{\partial \boldsymbol{\tau}}(\boldsymbol{\tau})\mathbf{b}_E] \right)_{kl} \mathbf{u}_k \otimes \mathbf{u}_l \right) \tag{2.78}$$

and since $\frac{\partial f}{\partial \boldsymbol{\tau}}(\boldsymbol{\tau}) = \sum_i \frac{\partial \hat{f}}{\partial \tau_i}\mathbf{u}_i \otimes \mathbf{u}_i$ and $\mathbf{b}_E = \sum_i \lambda_{Ei}^2 \mathbf{u}_i \otimes \mathbf{u}_i$ and

$$[\frac{\partial f}{\partial \boldsymbol{\tau}}(\boldsymbol{\tau})\mathbf{b}_E]_{ij} = \mathbf{u}_i \cdot \left(\frac{\partial f}{\partial \boldsymbol{\tau}}(\boldsymbol{\tau})\mathbf{b}_E \mathbf{u}_j \right) = \begin{cases} \frac{\partial \hat{f}}{\partial \tau_i}\lambda_{Ei}^2, & i = j \\ 0, & \text{otherwise} \end{cases}$$

and $[\mathbf{B}](\mathbf{b}_E)$ from Equation (2.74), as well as $\frac{\partial \boldsymbol{\tau}}{\partial \boldsymbol{\epsilon}_E} = \boldsymbol{C}$ from Equation (2.77)

$$\beta = \sum_{i,j} \frac{\partial \hat{f}}{\partial \tau_i} \hat{C}_{ij} \frac{\partial \hat{f}}{\partial \tau_j} \tag{2.79}$$

2.10 Flow direction

In order to stay inside of our yield surface during plastic deformation we know that for an associative model

$$\mathcal{L}_{\mathbf{v}}\mathbf{b}_E = -2\lambda \frac{\partial f}{\partial \boldsymbol{\tau}}(\boldsymbol{\tau})\mathbf{b}_E \tag{2.80}$$

This allows us to say that

$$\dot{\boldsymbol{\tau}} = \frac{\partial \boldsymbol{\tau}}{\partial \mathbf{b}_E}\dot{\mathbf{b}_E} \tag{2.81}$$

breaking up $\dot{\mathbf{b}_E}$ into elastic and plastic parts, we get that in order for $\dot{f} = 0$ we get

$$\dot{\boldsymbol{\tau}}_p = \frac{\partial \boldsymbol{\tau}}{\partial \mathbf{b}_E} : -2\lambda \frac{\partial f}{\partial \boldsymbol{\tau}}\mathbf{b}_E \tag{2.82}$$

however if we are instead working with the Hencky strain $\boldsymbol{\epsilon}_E$ using (2.73) and (2.68) we can see that

$$\dot{\boldsymbol{\tau}}_p = \lambda \frac{\partial \boldsymbol{\tau}}{\partial \boldsymbol{\epsilon}_E} : \left(\left([\mathbf{B}](\mathbf{b}_E) \circ [\frac{\partial f}{\partial \boldsymbol{\tau}}(\boldsymbol{\tau})\mathbf{b}_E] \right)_{kl} \mathbf{u}_k \otimes \mathbf{u}_l \right) \tag{2.83}$$

and noting that $\dot{\boldsymbol{\tau}}_p$ only changes in its eigen values we get

$$\frac{\partial \hat{\tau}_p}{\partial t} = \lambda \hat{C} \frac{\partial \hat{f}}{\partial \tau} \tag{2.84}$$

where λ serves as the magnitude of the plasticity and $\hat{C}\frac{\partial \hat{f}}{\partial \tau}$ is the direction

2.11 Return mapping

When we move from the continuous setting to the discrete setting, the modification of the stress due to plasticity needs to be modified. Since plasticity only turns on at the boundary it is easier to return an invalid stress state to the boundary then enforce the stress to never leave the boundary. As such during a discrete time step, if our stresses leave the yield surface we will project the stress back. We note that this projection is not the closest point projection. If it were, the direction in the continuous case would just be $\frac{\partial \hat{f}}{\partial \tau}$. Instead we use an oblique projection such that

$$\mathbf{X} - P(\mathbf{X}) = \gamma A \nabla f(P(\mathbf{X})) \tag{2.85}$$

or the projection direction is proportional to the normal of the surface multiplies by a matrix. The following Hamilton Jacobi equation

$$\frac{\partial}{\partial t}\phi + \|\nabla \phi\|_{\mathbf{C}} = 0 \tag{2.86}$$

$$\phi(\mathbf{X}, 0) = f(\mathbf{X}) \tag{2.87}$$

where

$$\|\mathbf{X}\|_{\mathbf{C}} = \sqrt{\mathbf{X}^T \mathbf{C} \mathbf{X}}$$

has the property that when a minimizer y_0 of the Hopf-Lax equation is found

$$X - y_0 = \mathbf{C} \nabla f(y_0) \tag{2.88}$$

As such, the return mapping from invalid stress states simply requires solving the Hopf-Lax formulation for the modified Hamilton Jacobi equation and outputting y_0. We note that $\|\mathbf{X}\|_{\mathbf{C}}$ is a norm given $\mathbf{C}$ is Symmetric Positive Definite. The Legendre transform of a norm is given as follows. Assume we have a norm $\|\cdot\|$ and its dual $\|x\|_* = \max_{y:\|y\|\leq 1} y^T x$. The Legendre transform of a function f is defined as

$$f^*(\mathbf{x}^*) = \sup_{\mathbf{x} \in \mathrm{R}^n} \mathbf{x}^{*T}\mathbf{x} - f(\mathbf{x})$$

we note that the dual of the dual norm is the original norm $\|\cdot\| = \|\cdot\|_{**}$ this implies

$\|\mathbf{x}\| := \sup_{\mathbf{y}\in\mathrm{R}^n, \|\mathbf{y}\|_* \leq 1} \mathbf{x}^T\mathbf{y}$

$$\begin{aligned}
\|\mathbf{x}^*\|^* &= \sup_{x\in\mathrm{R}^n} \mathbf{x}^{*T}\mathbf{x} - \|x\| \\
&= \sup_{\mathbf{x}\in\mathrm{R}^n} \mathbf{x}^* T\mathbf{x} - \sup_{\mathbf{y}\in\mathrm{R}^n, \|\mathbf{y}\|_*\leq 1} \mathbf{y}^T\mathbf{x} \\
&= \inf_{\mathbf{y}\in\mathrm{R}^n, \|\mathbf{y}\|_*\leq 1} \sup_{\mathbf{x}\in\mathrm{R}^N} \mathbf{x}^T(\mathbf{x}^* - \mathbf{y}) \\
&= \inf_{\mathbf{y}\in\mathrm{R}^n, \|\mathbf{y}\|_*\leq 1} \begin{cases} 0 & \mathbf{y} = \mathbf{x}^* \\ \infty & else \end{cases} \\
&= \begin{cases} 0 & \|\mathbf{x}^*\|_* \leq 1 \\ \infty & else \end{cases}
\end{aligned}$$

Now we simply need to calculate the dual norm of $\|\cdot\|_{\mathbf{C}}$ and we are ready to move forward

$$\|\mathbf{x}\|_{\mathbf{C}*} = \max_{\mathbf{y}:\|\mathbf{y}\|_{\mathbf{C}}\leq 1} \mathbf{y}^T\mathbf{x}$$

This max occurs when

$$\frac{\mathbf{C}\mathbf{y}}{\sqrt{\mathbf{y}^T\mathbf{C}\mathbf{y}}} = \mathbf{x}$$

We note that this max occurs on the boundary of the set, so $\|\mathbf{y}\|_{\mathbf{C}} = 1$

$$\mathbf{y} = \mathbf{C}^{-1}\mathbf{x}$$

$$\|\mathbf{x}\|_{\mathbf{C}*} = \mathbf{x}^T\mathbf{C}^{-1}\mathbf{x}$$

Our Hop-lax formulation then becomes

$$\tilde{\phi}(\mathbf{x_i}, t^k) = \mathtt{min}_{\mathbf{y}\in\mathbb{R}^n} \left\{ \phi^0(\mathbf{y}) + t^k H^* \left(\frac{\mathbf{x_i} - \mathbf{y}}{t^k} \right) \right\}$$

where H^* is the Legendre-Fenchel transform of $H = \|\cdot\|_{\mathbf{C}}$

$$H^*(\mathbf{x}) = \begin{cases} 0 & \|\mathbf{x}\|_{\mathbf{C}*} \leq 1 \\ \infty & otherwise \end{cases}$$

or equivalently we minimize $\phi^0(y)$ over the ellipse defined by $\sqrt{(\mathbf{x}-\mathbf{y})^T\mathbf{C}^{-1}(\mathbf{x}-\mathbf{y})} < t^k$

2.12 Modifications to algorithm

When implementing the return mapping, the only part of our previous algorithm we need to modify is equations (1.36)

$$\tilde{\mathbf{y}}_k^{j+1} = \mathbf{y}_k^j - \gamma \nabla \phi^0(\mathbf{y}_k^j) \tag{2.89}$$

$$\mathbf{y}_k^{j+1} = \texttt{PROJ}_{E_{\mathbf{C}}(\mathbf{x_i},t^k)}(\tilde{\mathbf{y}}_k^{j+1}) \tag{2.90}$$

where $\texttt{PROJ}_{E_{\mathbf{C}}(\mathbf{x_i},t^k)}(\tilde{\mathbf{y}}_k^{j+1})$ projects to the ellipse defined by $\sqrt{(\mathbf{x}-\mathbf{y})^T\mathbf{C}^{-1}(\mathbf{x}-\mathbf{y})} < t^k$. In order to find this projection, we simply find $\hat{y}_k^{j+1} = \mathbb{R}(\tilde{\mathbf{y}}_k^{j+1} - \mathbf{x}_i)$ where $\mathbb{R}$ is the rotation matrix defined by the Eigenvectors of $\mathbf{C}$. This allows us to project back assuming that the ellipse is orientated along the x, y, and z axis. For 2D the projection is found by assuming that our point is in the first quadrant ($\hat{y}_0 > 0$ and $\hat{y}_1 > 0$) and our ellipse is orientated along the x and y axis. then we can use Newton to find the roots of the following function in t to find the closest projection point.

$$F(t) = (\frac{e_0 y_0}{t + e_0^2})^2 + (\frac{e_1 y_1}{t + e_1^2})^2 - 1 = 0$$

In addition, our initial data will be an analytic function whose zero isocontour is the Yield Surface we desire.

2.13 Results

To show the possibilities of our method, we ran our stress projection with an MPM solver (see next chapter for an explanation on MPM) to see how different yield surfaces would result in different material behaviors. For each set of figures below we provide the yield surface used, as well as three different simulations that were run. The yield surfaces are provided using the Hencky strain as their base. As such $(0, 0)$ is a point with no stress. The first quadrant are points that are in tension, the third quadrant are points in compression, and the second and fourth quadrant are points under shear force. For the following section we will call the line $y = x$ the hydrostatic axis. All of our yield surfaces will be symmetric across the hydrostatic axis. Every simulation was run using the Saint Venant–Kirchhoff model for the hyperelasticity portion, using a Young's modulus of 5 and a Poissons ratio of 0.4. We have 4 groups of simulations. The first uses the Drucker

Prager yield surface. Drucker Prager is most often used to model sand and other granular materials. For our simulations we applied varying amounts of ”cohesion” to the model, which for our purposes was how much we shifted the cone along the axis $y = x$ into the first quadrant. Higher values of cohesion meant that the material would stick together through larger forces. The other three yield surfaces are all non physical. The first two are the 2D projection of their 3D yield surfaces. The von Mises yield criterion in 3d is used to simulate some metals, and its 2D projection is an ellipse aligned along the hydrostatic axis. The Tresca yield criterion is based around the idea of having a maximum for the amount of shear allowed in a material. Again we project it onto a 2D space and we find a 6 sided prism. The last yield surface we provide is a lemniscape aligned along the hydrostatic axis. This is a material that seems to avoid shear, however as it is non convex it is very non physical.

Most of these simulations use the associated flow rule we derived previously. However one problem with an associated flow rule is that it is not nescessarily volume preserving[39]. As such, some of the Drucker Prager cone examples were run with a non associated flow rule that was volume preserving. This non associated flow rule also used our method, just with a different matrix $\mathbf{C}$ that was found to return the projection along the line $y = -x$.

We ran three different simulations for each yield surface. The first simulation in the top right corner of each figure is the result of 2.5 seconds of a verticle column of the material collapsing under its own weight. The middle row is the results of a block of the material falling through an open hour glass after 2.5 and 5 seconds, occasionally when the material did not need the full simulation time we use 1.25 and 2.5 seconds for the images. The last row is a block of the material falling onto a sharp wedge, cutting it in half, after 1 and 2 seconds.

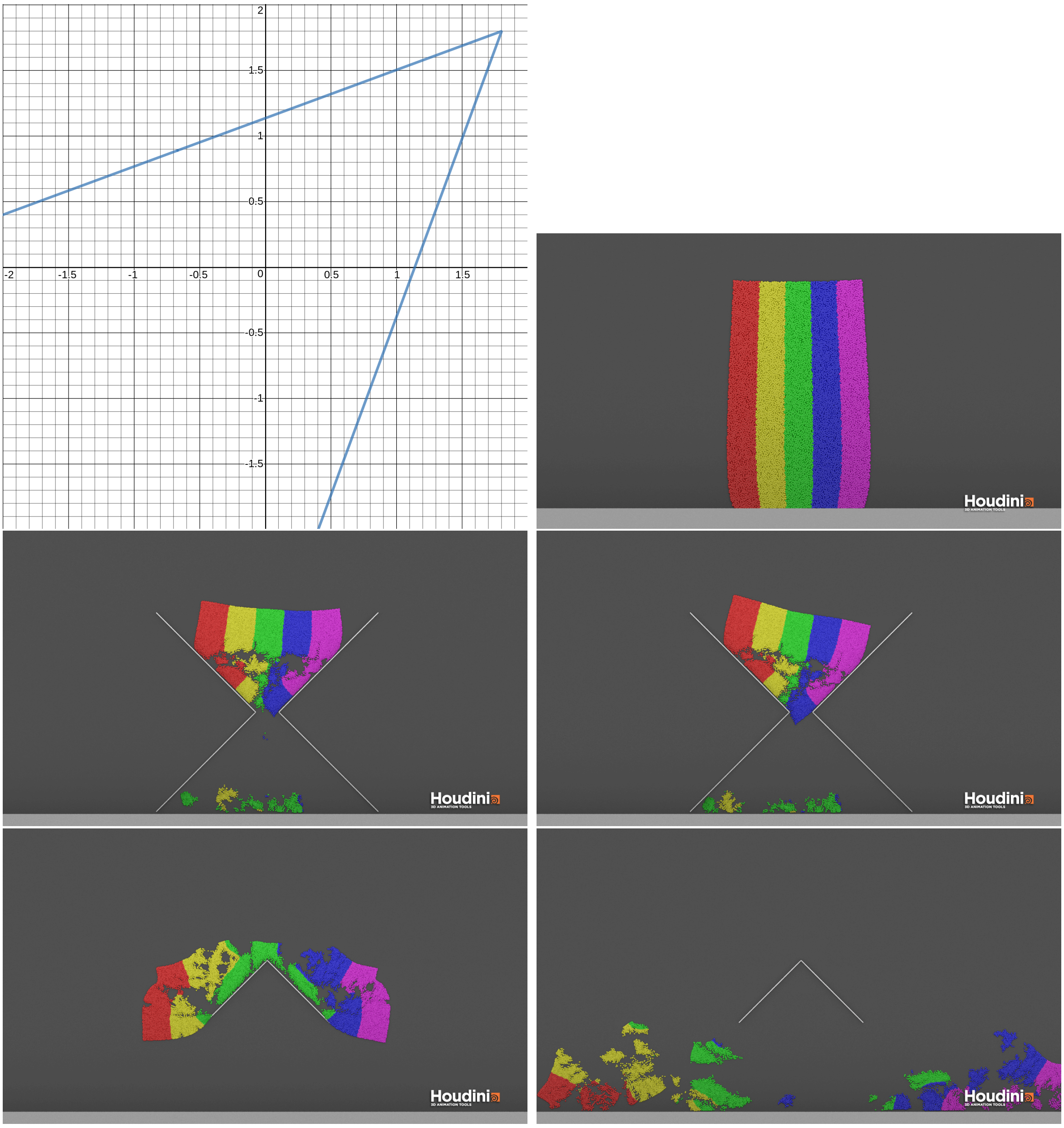

Figure 2.2: Drucker Prager with cohesion of 1.8 using an associated flow rule. Due to the large amount of cohesion allowed, the material is bouncy and tends to break into large chunks.

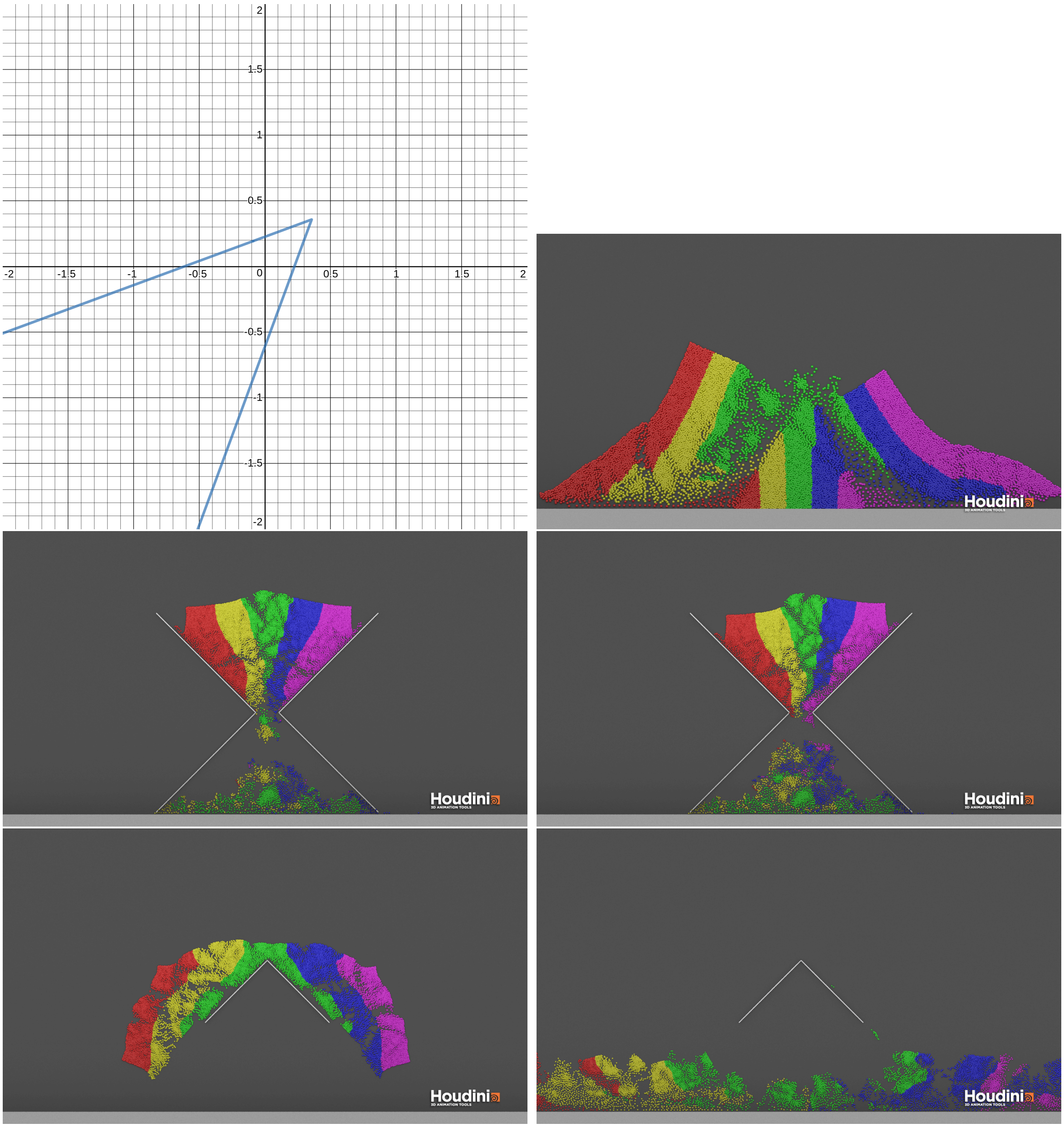

Figure 2.3: Drucker Prager with cohesion of .36 Associated. We provide one more example of wet sand. Again we note how the sand breaks apart in chunks, but is still able to flow through the hourglass.

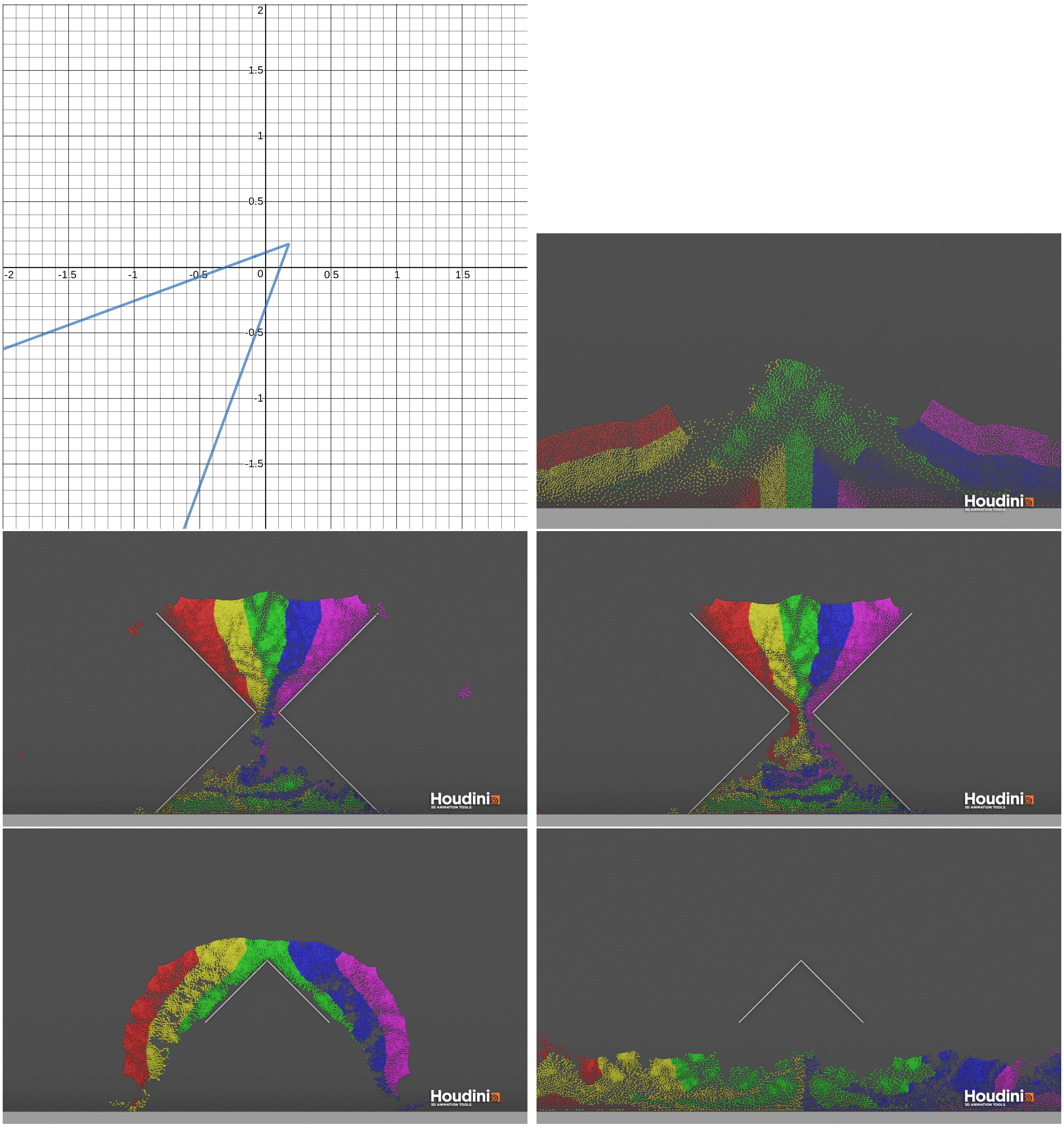

Figure 2.4: Drucker Prager with cohesion of .18 using an associated flow rule. Here we start to see more of what we might expect with wet sand, however due to our return mapping being non volume preserving, all three simulations tend to see a large amount of volume growth.

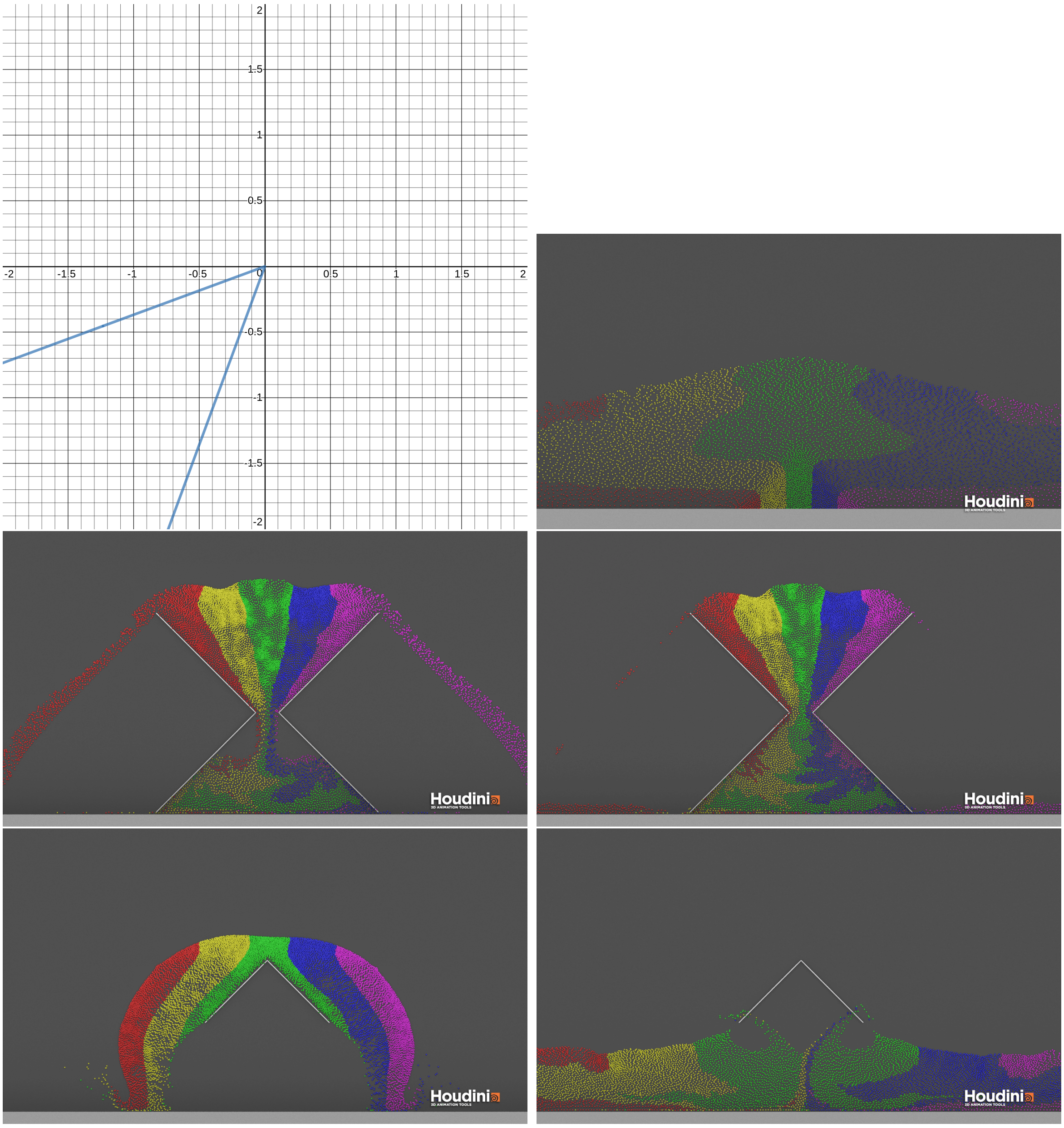

Figure 2.5: Drucker Prager with cohesion of .018 using an associated flow rule. This small amount of cohesion simulates nearly dry sand. The sand flows as one might expect, however again there is a large amount of volume gain.

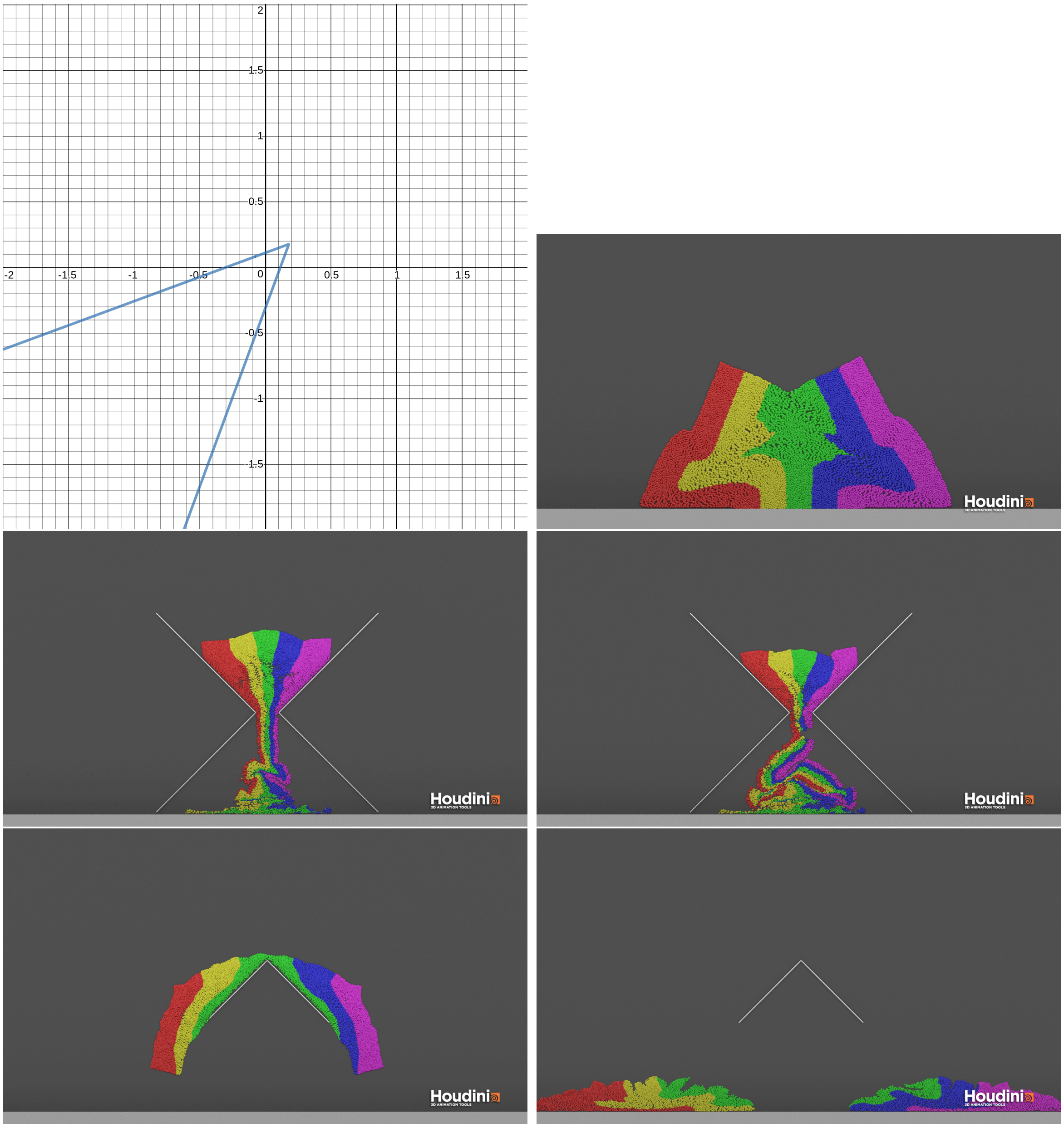

Figure 2.6: Drucker Prager with cohesion of .18 using a non Associated flow rule. By using a volume preserving flow rule we can see a closer expectation of what our wet sand might actually look like. The sand sticks together, but still piles nicely.

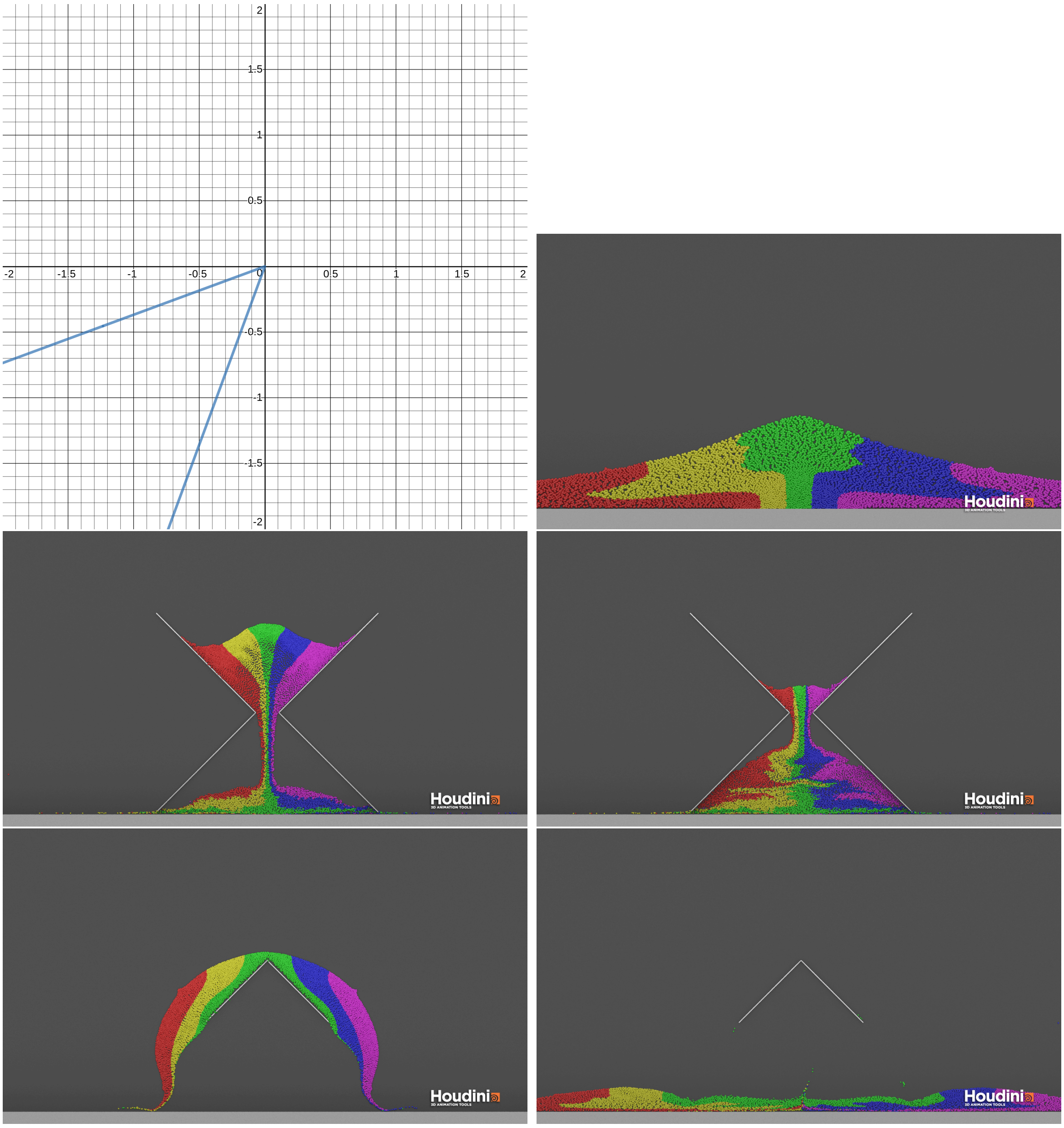

Figure 2.7: Drucker Prager with cohesion of .018 non Associated. Again we simulate nearly dry sand with a volume preserving flow rule. There is no volume growth, and in the wedge example we can even see the two waves of sand colliding beneath the wedge

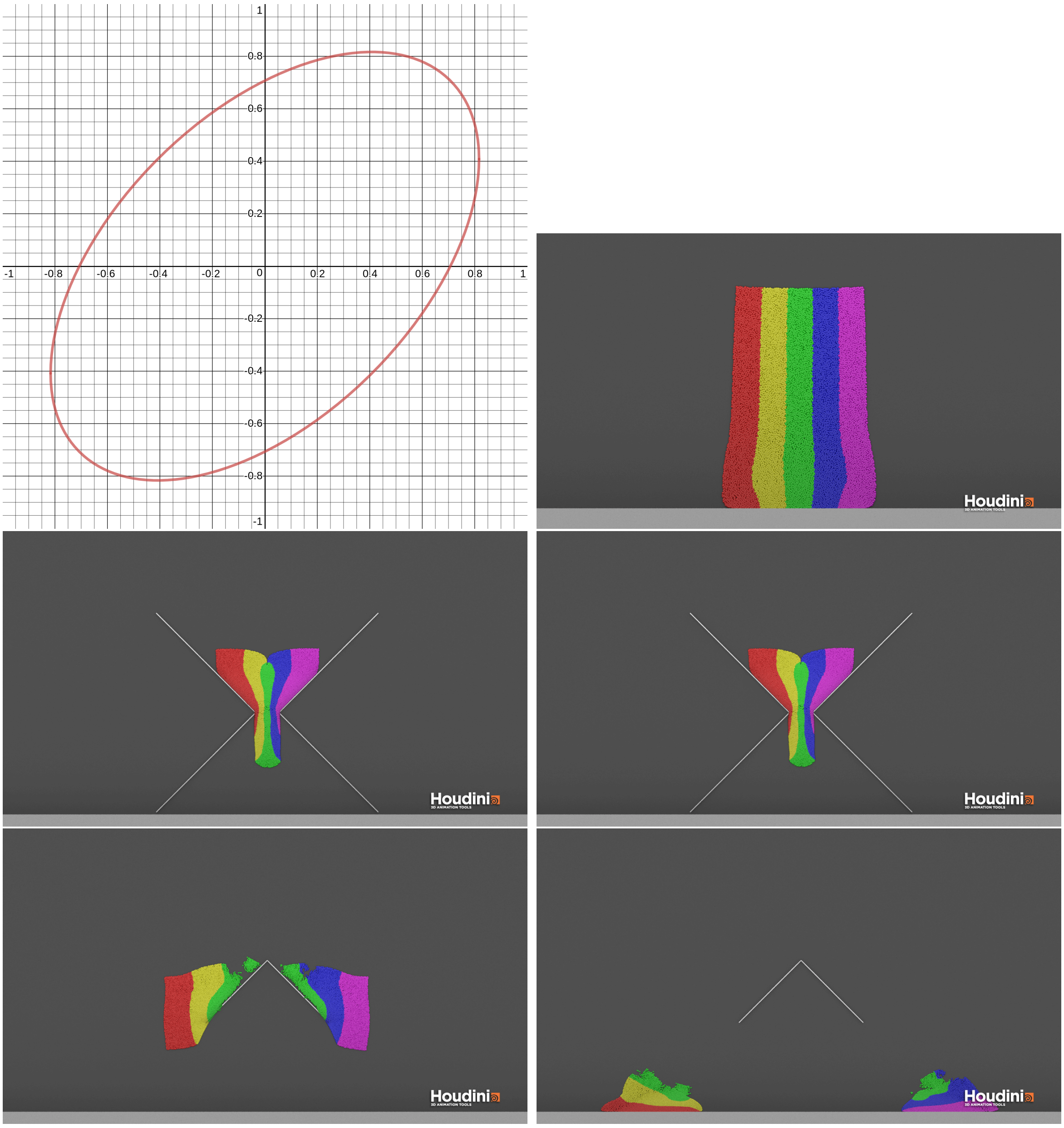

Figure 2.8: With the von Mises 2d projection we don't really see what we would expect from a simulation of metals. Instead we see more of a Jello like material that jiggles and is able to be squeezed. The hourglass simulation reached a steady state by 2.5 seconds, so we show you the results after 1.5 and 2.5 seconds.

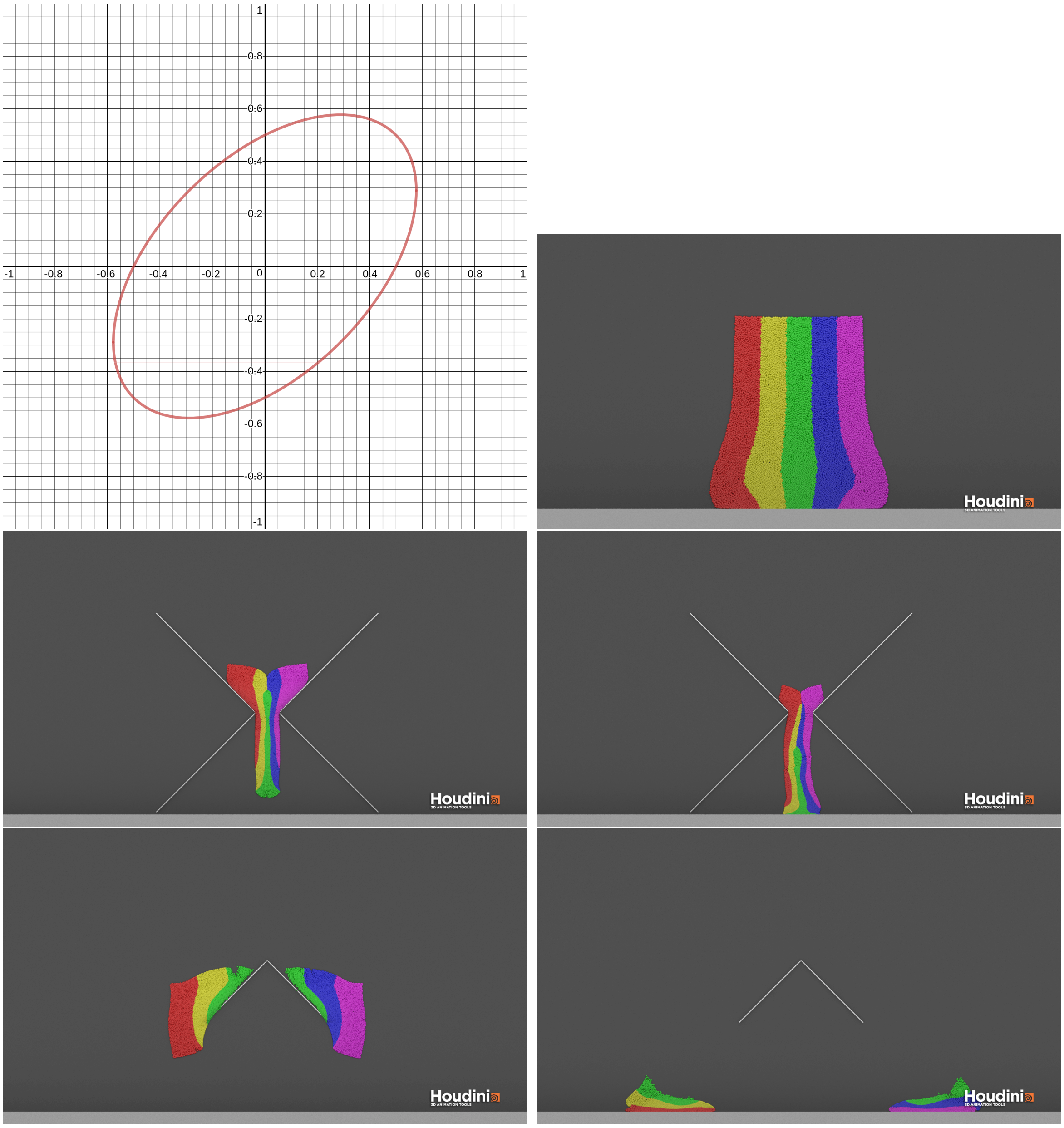

Figure 2.9: By shrinking the radius of the ellipse given in the 2d projection of the von Mises yield criterion, more of the stresses get transferred into the plasticity. This allows our material to be more willing to shrink and squeeze. During the hourglass simulation for a brief time the material separates before reuniting at the end. Again the hourglass simulation is shown at 1.25 and 2.5 seconds.

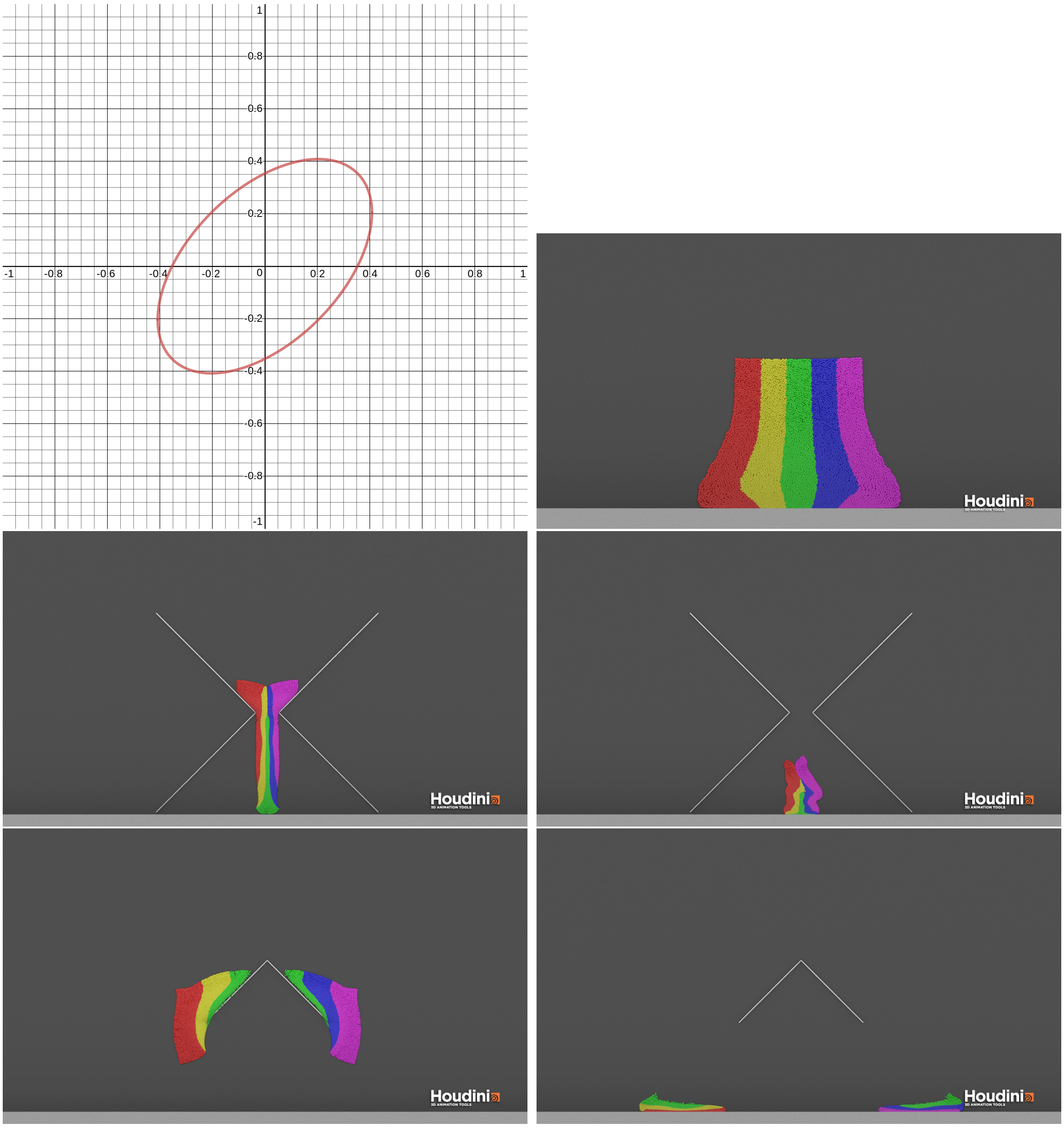

Figure 2.10: As before by shrinking the ellipse even further, more stress gets pushed into plasticity. This causes the falling materials to shrink more than the collapsing column, as larger stresses are shown in the falling materials. The small stresses experienced in the collapsing column are allowed to remain elastic.

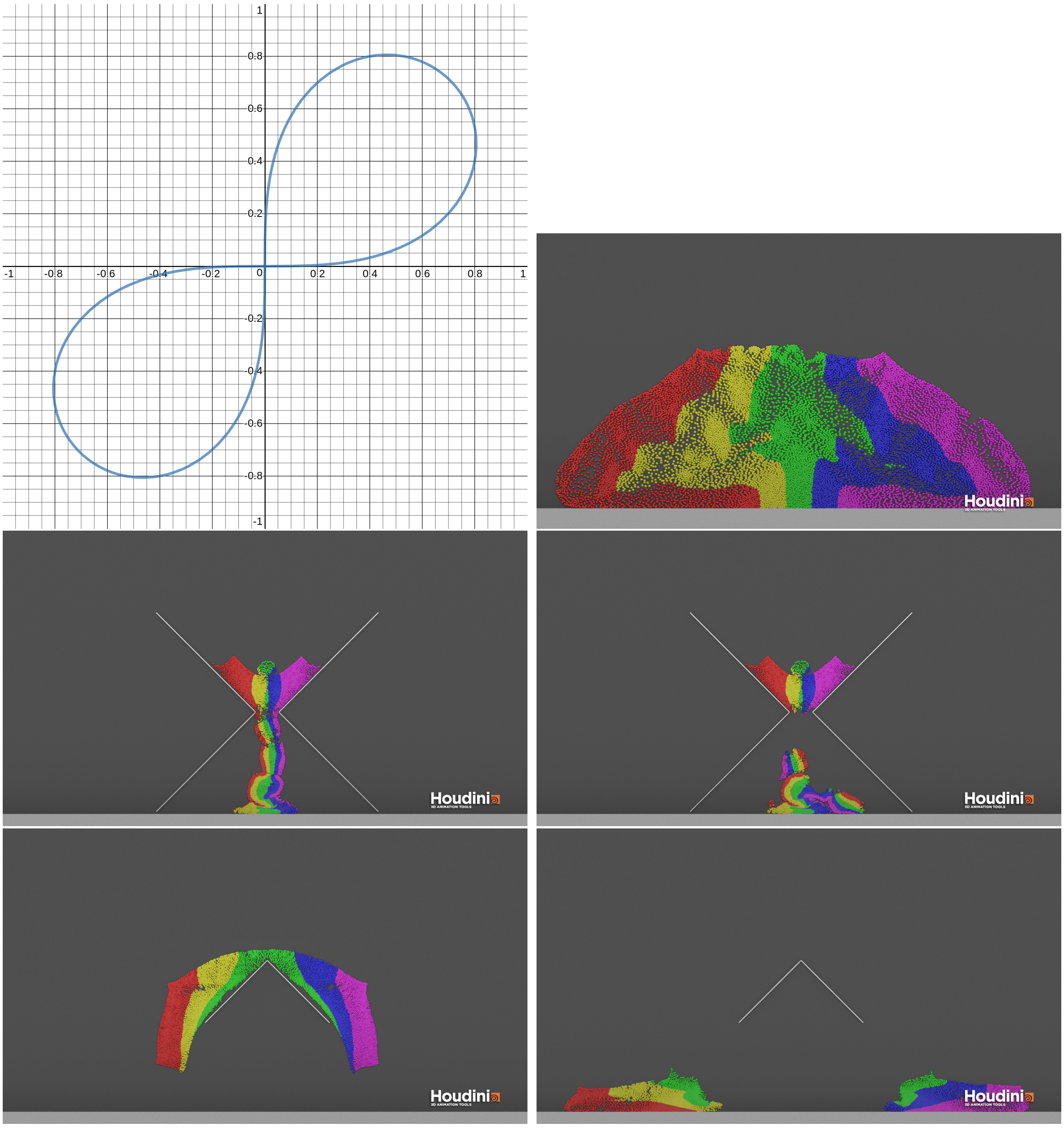

Figure 2.11: Here our yield surface is a lemniscape with ”radius” 1. The most interesting simulation ends up being the column collapse. Due to shear forces not being elastic, the entire material expands sideways as the plastic forces compensate before settling in. In the wedge drop, the material hits the ground and expands sideways as the shear force is not allowed to be elastic.

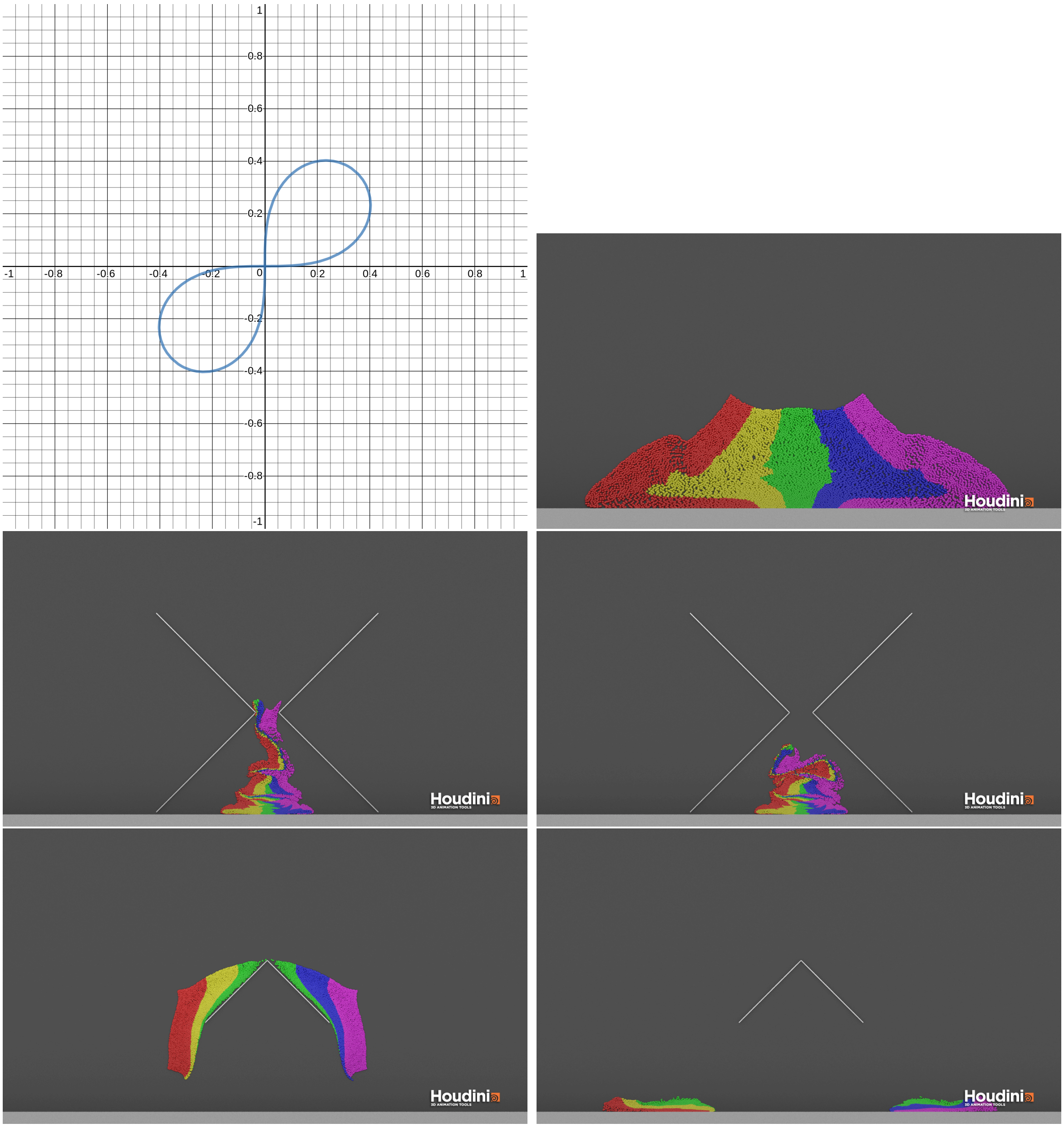

Figure 2.12: Again as seen in the von Mises examples, by shrinking the size of the yield surface, large stresses are moved into the plasticity and result in large volume shrinkages.

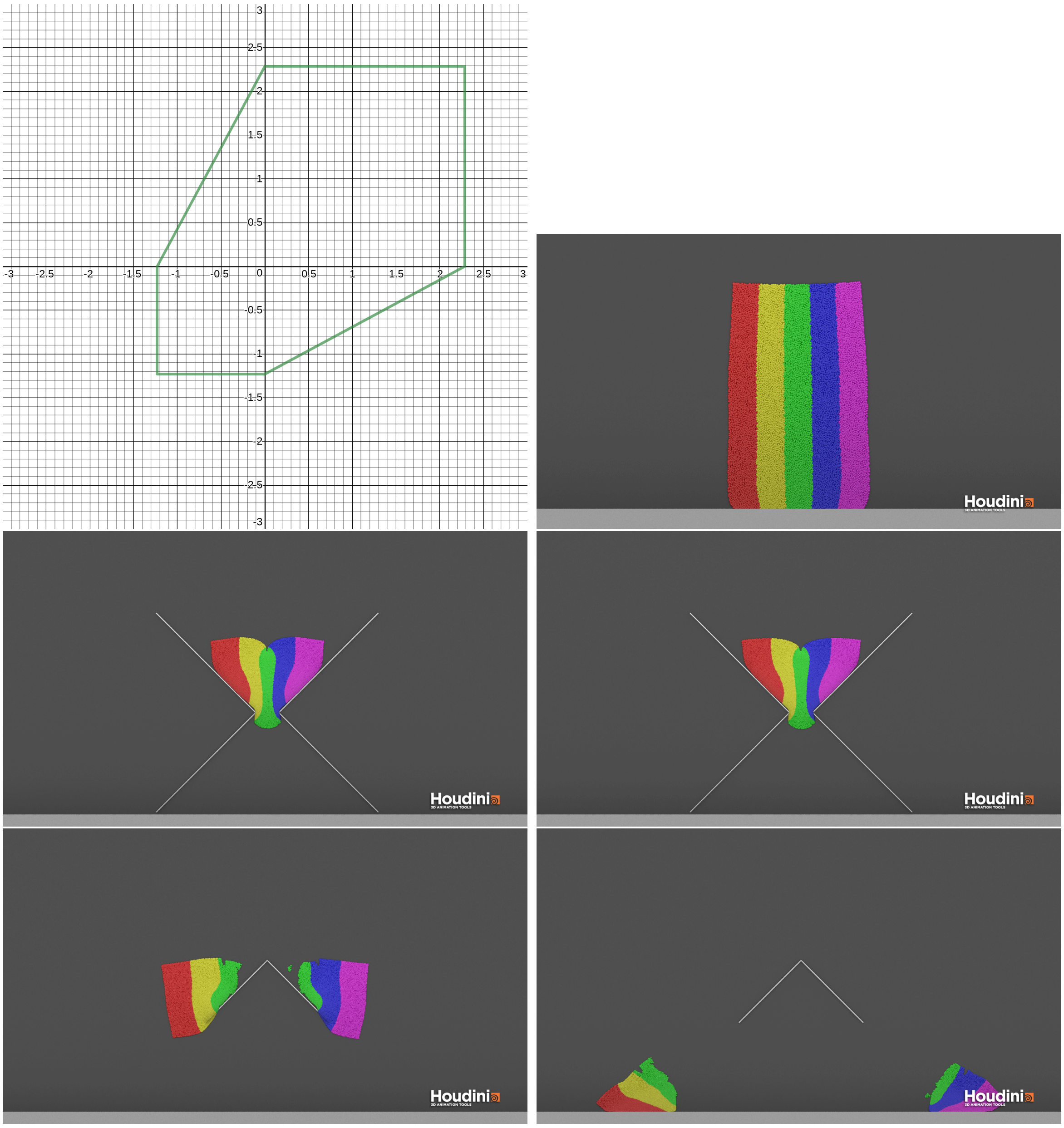

Figure 2.13: Here we have a material that allows more compression than expansion. As such the material tends to stick together through the forces we applied to it. Worth noting is that these are some of the bouncier examples, probably due to the larger yield surface.

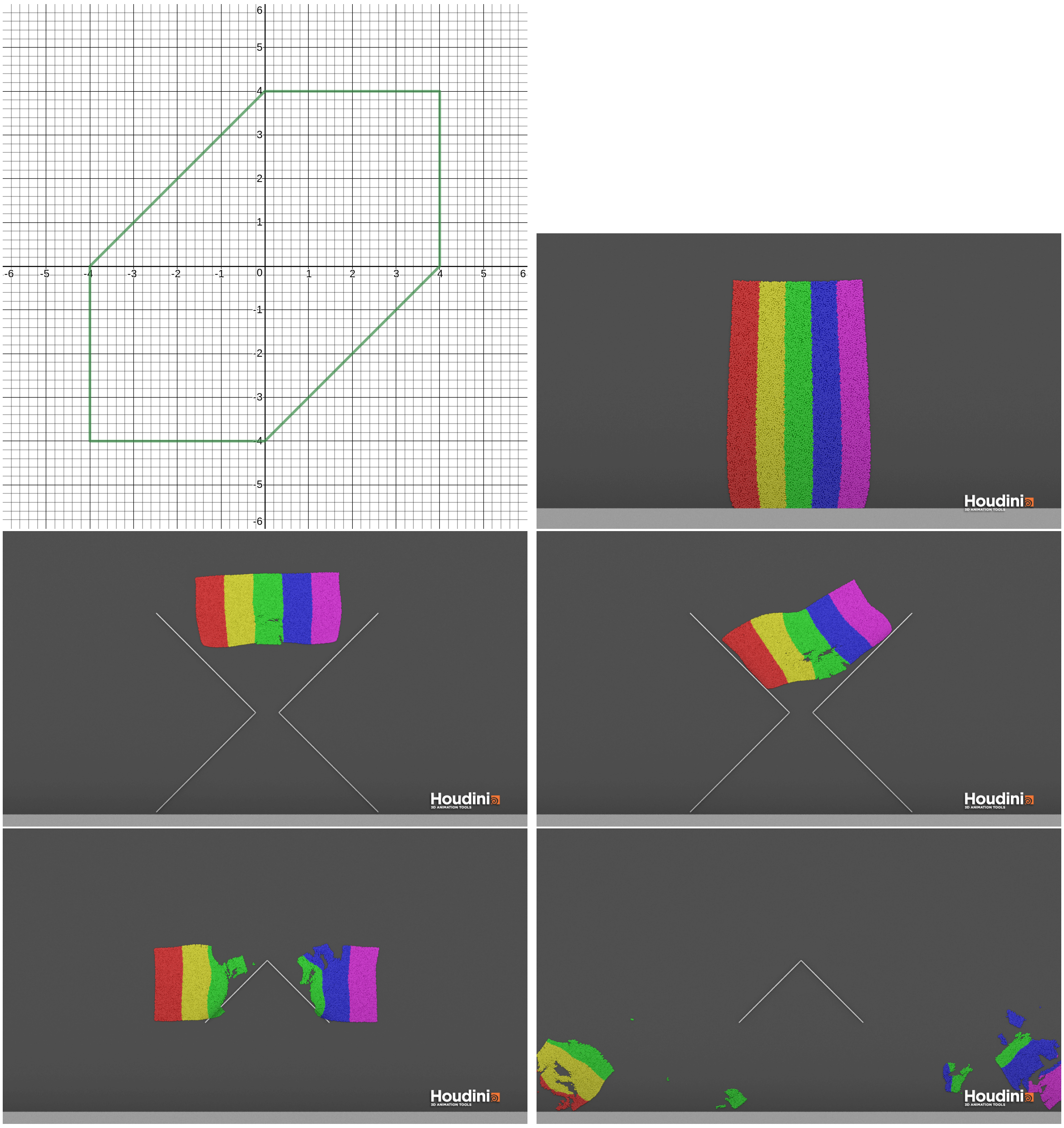

Figure 2.14: When both compression and tension are given a large area to work with, we end up with a material that is very bouncy and does not break up easily. A very small portion of it breaks off in the hourglass example. The impact with the floor is enough to break up the material further in the wedge example.

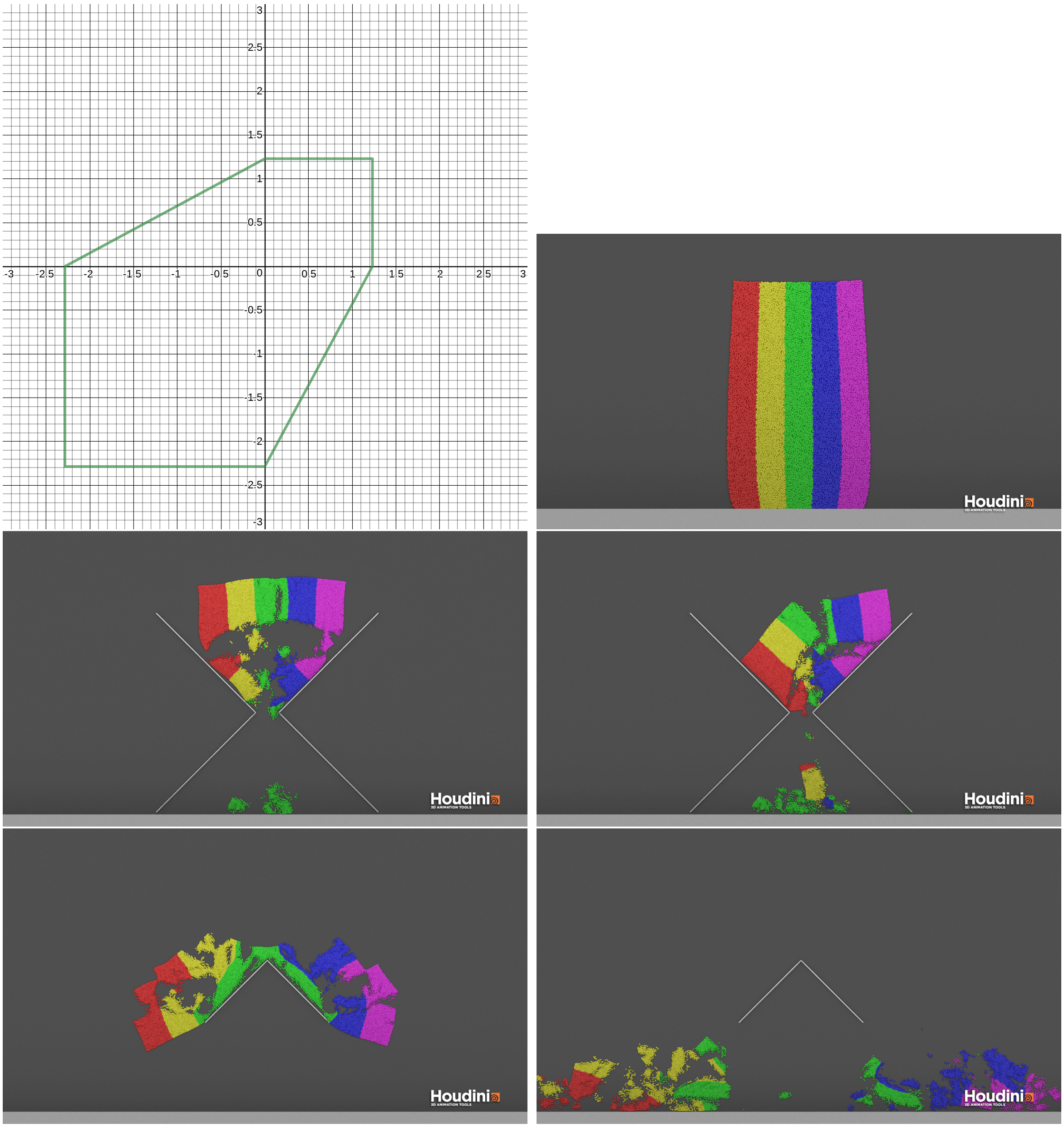

Figure 2.15: In this material the yield surface allows for more compression than tension. This seems to cause the material to actually bounce off of the surfaces it hits. In addition it breaks up into larger chunks.

CHAPTER 3

Material Point Method

The Material Point Method(MPM) is a modern method of simulating continuum materials like solids, liquids, and gases. It was derived by Sulsky et al.[40] as an extension of the Fluid-Implicit-Particle(FLIP) [41]. Being a hybrid method, it shares many common features with the Particle-in-Cell(PIC) method [42, 43]

In MPM the material Ω is simulated using both Lagrangian particles and an Eularian grid. The Lagrangian particles are used to store variables such as mass, position, velocity, etc., while the Eularian grid is used to efficiently calculate gradients used in force calculations and deformation gradients. Both the Lagrangian and Eularian views have advantages when it comes to calculating states in simulations. The Lagrangian view can efficiently track the motion of the material through advection, while the Eularian view can easily capture the interactions between particles as well as self collision. Because the mesh in the Eularian view is independent of the material Ω MPM can easily handle large deformations that would cause pure grid based methods to struggle. The Lagrangian view allows easy calculation of gradients and interaction between separate materials that a pure particle based method would struggle with.

In this chapter we will present a derivation of the Material Point Method specifically for elasoplastic simulations following the work of Klar et al.[44].

3.1 Overview

Every MPM time step begins with all information being carried on the particles. It then follows a number of steps, and finishes with all information again on the particles. As such, the grid being used does not need to remain between time steps. In addition one could form a different grid at each time step, however in practice the same grid will be reused for efficiency.

The following steps will be carried out at every time step

1. Transfer to grid: Transfer mass and momentum from the particles to the grid. Use the mass and momentum on the grid to calculate velocity.
2. Apply forces: Compute elastic forces using a deformation gradient that has been projected in step 7, and apply the forces to the grid velocities.
3. Grid collisions: Project grid velocity to check for collisions against any rigid bodies or obstacles in the simulation.
4. Friction: Compute and apply friction based on any collisions in previous step. Retain velocities from before and after this step.
5. Transfer to particles: Transfer velocities from grid to particles.
6. Update particles: Update remaining particle states, such as position and deformation gradient
7. Plasticy and hardening: Project the deformation gradient for plasticity, and update the elastic and plastic parts.

3.1.1 Material state

In MPM the state of the simulation is stored on the particles. We store mass m_p, position $\mathbf{x}_p^n$, velocity $\mathbf{v}_p^n$, and affine momentum $\mathbf{B}_p^n$. This affine momenum matrix $\mathbf{B}_p^n$ is stored for use in the APIC transfer[45]. We use APIC due to its increased stability over FLIP.

In addition we store the elastic component of the deformation gradient, $\mathbf{F}_E^{n,p}$. The plastic component of the deformation gradient $\mathbf{F}_P^{n,p}$ is not needed in our simulations, this is due to our assumptions on energy and strain being derived only from the elastic deformation gradient. Note that we keep track of whether $\mathbf{F}_E$ is on the particle by using a super script in order to keep with notation from previous chapters.

3.1.2 Transfer weights

In order to facilitate the frequent transfer between particles and grid we need to associate between each particle p and each grid node i a weightw_{ip}^n which determines how strongly the particle and nodes interact. For particles and nodes close together we expect the weight to be large. As particles and nodes get further apart we expect the weights to decrease. In addition for simplicity of computation we want each particle to have a finite range of influence, i.e. if a particle and node are further apart then a distance d we should expect $w_{ip}^n = 0$. To satisfy this we compute our weights based off of a kernel $w_{ip}^n = N(\mathbf{x}_p^n - \mathbf{x}_i^n)$, where $\mathbf{x}_p^n$ and $\mathbf{x}_i^n$ are the locations of the particle and grid node respectivly. In addition when computing forces we will need the spatial derivatives of the weights $\nabla w_{ip}^n = \nabla N(\mathbf{x}_p^n - \mathbf{x}_i^n)$

The choice of the kernel N leads to trade off with respect to smoothness, computational efficiency, and the area of influence. Multilinear kernels have typically been used in FLIP solvers due to their simplicity. However Steffen et al. [46] show that due to the discontinuity of the gradient, as well as the gradient being far from zero when N is close to zero, leading to large forces being applied to grid nodes with tiny weights, multilinear kernels are not the best choice. Instead we use cubic b-splines, when the grid spacing is h, the cubic b spline is given as

$$\hat{N}(x) = \begin{cases} \frac{1}{2}|x|^3 - |x|^2 + \frac{2}{3} & 0 \leq |x| < 1 \\ \frac{1}{6}(2 - |x|)^3 & 1 \leq |x| < 2 \\ 0 & 2 \leq |x| \end{cases} \tag{3.1}$$

$$N(\mathbf{x}) = \hat{N}(\frac{\mathbf{x}_x}{h})\hat{N}(\frac{\mathbf{x}_y}{h})\hat{N}(\frac{\mathbf{x}_z}{h}) \tag{3.2}$$

3.2 Transfer to grid

The first step of every time step is to transfer mass from the particles to the grid. This is done by simplying suming each weighted particles mass.

$$m_i^n = \sum_p w_{ip}^n m_p \tag{3.3}$$

Again particles far away from a grid node do not contribute to its mass, as such we can assume that they do not contribute to any of the other transfers given below.

Next we need to transfer velocity. This is done using the APIC transfers given in Jiang et al. [45]. We use APIC instead of PIC or FLIP due to its ability to conserve linear and angular momentum [45, 47]. The PIC transfers suffer from excessive dissipation [41, 48], and FLIP transfers exhibit ringing instabilities caused by particle velocity modes hidden from the grid [48, 49]. The PIC and FLIP transfers are done by

$$(m\mathbf{v})_i^n = \sum_p w_{ip}^n m_p \mathbf{v}_p^n \tag{3.4}$$

This follows from the assumption that the volume represented by a particle p moves with uniform $\mathbf{v}_p$ velocity. APIC discards this assumption and instead surrounds each particle with a local velocity field given by the affine function

$$\mathbf{v}_p^n(\mathbf{x}) = \mathbf{v}_p^n + \mathbf{C}_p^n(\mathbf{x} - \mathbf{x}_p^n) \tag{3.5}$$

where $\mathbf{C}_p^n$, the velocity spatial derivative, is expressed using the affine momentum matrix $\mathbf{B}_p^n$ and inertia tensor $\mathbf{D}_p^n$ as

$$\mathbf{C}_p^n = \mathbf{B}_p^n(\mathbf{D}_p^n)^{-1} \tag{3.6}$$

where

$$\mathbf{D}_p^n = \sum_i w_{ip}^n(\mathbf{x}_i^n - \mathbf{x}_p^n)(\mathbf{x}_i^n - \mathbf{x}_p^n)^T = \frac{h^2}{3}\mathbf{I} \text{ for N cubic} \tag{3.7}$$

Although typically the inertia tensor $\mathbf{D}_p^n$ is a high order polynomial based off of the grid nodes and particles, for the cubic B splines it simplifies to the constant tensor given. As such we can calculate the transfered particle momentum as

$$(m\mathbf{v})_i^n = \sum_p w_{ip}^n m_p(\mathbf{v}_p^n(\mathbf{x}_i) + \mathbf{B}_p^n(\mathbf{D}_p^n)^{-1}(\mathbf{x}_i^n - \mathbf{x}_p^n)) \tag{3.8}$$

Then the grid velocities are simply calculated by dividing momentum by mass

$$\mathbf{v}_i^n = \frac{(m\mathbf{v})_i^n}{m_i^n} \tag{3.9}$$

3.3 Grid update

We next update velocities on the grid. In order to do so, we need to calculate the forces being applied on the material, and use these forces to update the grid velocities. The velocity update can either be explicit or implicit. The explicit velocity updates works well with problems that have a relatively low stiffness. When the stiffness of the system becomes too large, it becomes advisable to move to an implicit solver. In either case we will need to start with the force $\mathbf{f}_i(<\mathbf{F}_E^{n,p}>)$ where we use <> to denote that the elastic deformation gradient at every particle $\mathbf{F}_E^{n,p}$ is used in the calculation.

$$\mathbf{f}_i(<\mathbf{F}_E^p>) = -\sum_p V_p^0 \frac{\partial \psi}{\partial \mathbf{F}_E}(<\mathbf{F}_E^p>)\mathbf{F}_E^{p\,T}\nabla w_{ip}^n \tag{3.10}$$

where V_p^0

3.3.1 Explicit integration

For explicit integration we assume that the forces can be calculated based off of the current state giving

$$\mathbf{v}_i^* = \mathbf{v}_i^n + \frac{\delta t}{m_i^n}\mathbf{f}_i(<\mathbf{F}_E^{n,p}>) \tag{3.11}$$

In the explicit integration we can compute the force on each node separately (including external forces) and apply them per grid node

$$\mathbf{f}_i^n = -\sum_p V_p^0 \frac{\partial \psi}{\partial \mathbf{F}_E}(\mathbf{F}_E^{n,p})\mathbf{F}_E^{n,pT}\nabla w_{ip}^n + f^{ext}(\mathbf{x}_i) \tag{3.12}$$

$$\mathbf{v}_i^* = \mathbf{v}_i^n + \frac{\delta t}{m_i^n}\mathbf{f}_i^n \tag{3.13}$$

Once forces have been applied we will have to handle collisions $\mathbf{v}_i^* \rightarrow \bar{\mathbf{v}}_i^{n+1}$ and then apply friction $\bar{\mathbf{v}}_i^{n+1} \rightarrow \tilde{v}_i^{n+1}$ The process for collision is detailed in section 3.3.3.

3.3.2 Implicit integration

For implicit integration we need to know the updated velocities in order to compute the forces acting on the grid nodes. In addition, because the collision handling impacts the velocities it needs to be incorporated into the implicit solver. To enforce collision response we use a set of constraints G_k to be defined in section 3.3.3.

$$\bar{\mathbf{v}}_i^{n+1} = \mathbf{v}_i^n + \frac{\delta t}{m_i^n}\mathbf{f}_i(< \mathbf{F}_E^{n+1,p} >) + \sum_k \nabla G_{ki}\lambda_k \tag{3.14}$$

$$G_k \geq 0 \tag{3.15}$$

$$\lambda_k \geq 0 \tag{3.16}$$

$$G_k\lambda_k = 0 \tag{3.17}$$

Where a (collision object/node) pair is considered collision free if $G_k(< \bar{\mathbf{x}}_i >) \geq 0$ where $\bar{\mathbf{x}}_i = \mathbf{x}_i^n + \delta t\bar{\mathbf{v}}_i$. Note that friction is still handled explicitly in this formulation.

The system is nonlinear and is solved using Newton's method. The collision constraints are solved for by projection, and each $\mathbf{F}_E^{n+1,p}$ is updated with the projection given in the previous chapter. Due to plasticity the system is asymmetric, as such GMRES is used to solve it. In the absence of plasticity, the Conjugate Gradient method can be used in each newton step.

3.3.3 Collision and friction

We separate the collision and friction into two steps. While this is not necessary, in the implicit solve we only factor in the collision response. Thus separating the two makes the derivation easier for the implicit solve and has no effect on the explicit solve.

Each collision object is represented by a signed distance function $\phi(\mathbf{x})$ with the convention it is positive outside and negative inside. If we could handle the collision detection on particles, we would simply enforce $\phi(\mathbf{x}_p^{n+1} \geq 0$. In practice however doing so would cause the deformation gradient $\mathbf{F}_E^{n+1,p}$ to become out of sync due to direct updates on $\mathbf{x}_p^{n+1}$. To solve this we instead use the grid velocity to process collisions.

While one could process collisions by adjusting $\bar{\mathbf{v}}_i^{n+1}$ to enforce $\phi(\mathbf{x}_p^{n+1}) \geq 0$, this would require very complicated collision process in both the explicit and implicit cases due to $\bar{\mathbf{v}}_i^{n+1}$ affecting multiple nodes. Instead we process collisions using the nodes themselves as in Gast et al. [50]. We use the deformed grid to apply constraints instead of the particles. The deformed grid nodes $\bar{\mathbf{x}}_i$ are defined as

$$\bar{\mathbf{x}}_i = \mathbf{x}_i^n + \delta t\bar{\mathbf{v}}_i \tag{3.18}$$

Now we can not simply say that $\phi(\bar{\mathbf{x}}_i^{n+1}) \geq 0$ because we expect there to be grid nodes inside of an obstacle. Instead we will have a set of constraints $G_k(< \bar{\mathbf{x}}_i^{n+1} >) geq 0$ or $G_k(< \bar{\mathbf{x}}_i^{n+1} >) = 0$. These constraints can be applied indepently on each grid node when collision is detected. There are three types of collision constraints that our MPM method handles:

1. Sticky: The sticky constraint requires that $\bar{\mathbf{x}}_p^{n+1}$ remain stationary with respect to the collision object
2. Separating: The separating constraint requires that a node $\bar{\mathbf{x}}_p^{n+1}$ does not penetrate deeper into the object in the next time step, it is however free to move in any direction that satisfies this constraint
3. Slip: The slip constraint simply requires that if a node is colliding with the object, it is allowed to move along the surface of the object but it must stay at the same depth. A node not colliding has no collision response.

3.4 Transfer to particles

Once we have the post friction velocities $\tilde{\mathbf{v}}_i^{n+1}$, we can compute the new velocities $\mathbf{v}_p^{n+1}$ and affine momentum $\mathbf{B}_p^{n+1}$. If we were using PIC or FLIP the new velocities would be

$$\mathbf{v}_p^{PIC} = \sum_i \tilde{\mathbf{v}}_i^{n+1} w_{ip}^n \tag{3.19}$$

$$\mathbf{v}_p^{FLIP} = \sum_i (\tilde{\mathbf{v}}_i^{n+1} - \mathbf{v}_i^n) w_{ip}^n \tag{3.20}$$

With FLIP the particle velocities are set as a linear combination of the two with ratio α as α-FLIP

$$\mathbf{v}_p^{n+1} = \alpha(\mathbf{v}_p^n + \mathbf{v}_p^{FLIP}) + (1-\alpha)\mathbf{v}_p^{PIC} \tag{3.21}$$

With PIC and APIC the particle velocities are simply

$$\mathbf{v}_p^{n+1} = \mathbf{v}_p^{PIC} \tag{3.22}$$

In addition APIC needs to update the affine momentum as well

$$\mathbf{B}_p^{n+1} = \sum_i w_{ip}^n \tilde{\mathbf{v}}_i^{n+1} (\mathbf{x}_i^n - \mathbf{x}_p^n)^T \tag{3.23}$$

3.5 Update particle state

Next the particles position and deformation gradient need to be updated. The position is simply found by interpolating back from the moving grid nodes

$$\mathbf{x}_p^{n+1} = \sum_i w_{ip}^n \bar{\mathbf{x}}_i^{n+1} \tag{3.24}$$

Note that this uses the weights from the start of the time step. Since the particles move with the flow, the deformation gradient at time $n+1$ is simply found by

$$\hat{\mathbf{F}_E}^{n+1,p} = \mathbf{F}_E^{n+1,p} + \delta t (\nabla \tilde{\mathbf{v}})_p \mathbf{F}_E^{n,p} \tag{3.25}$$

where

$$(\nabla \tilde{\mathbf{v}})_p = \sum_i \tilde{\mathbf{v}}_i^{n+1} (\nabla w_{ip}^n)^T \tag{3.26}$$

The last step is to ensure that the deformation gradient $\hat{\mathbf{F}_E}^{n+1,p}$ stays within the yield surface. For this we simply use the projection given in the previous chapter to arrive at

$$\mathbf{F}_E^{n+1,p} = \mathbf{Z}(\hat{\mathbf{F}_E}^{n+1,p}) \tag{3.27}$$

where $\mathbf{Z}$ denotes the projection.

CHAPTER 4

Appendix

4.1 Appendix: eigen decomposition differentials

Consider the space of symmetric 3×3 matrices $\mathbb{R}^{3\times3}_{\text{sym}}$, thus $\mathbf{S} \in \mathbb{R}^{3\times3}_{\text{sym}}$ have eigen decompositions, $\mathbf{S} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^T$ for some orthogonal $\mathbf{V}$ and diagonal $\boldsymbol{\Lambda}$. We can define a class of functions $\mathbf{g} : \mathbb{R}^{3\times3}_{\text{sym}} \to \mathbb{R}^{3\times3}_{\text{sym}}$ that are inherited from scalar functions $g : \mathbb{R} \to \mathbb{R}$ as

$$\mathbf{g}(\mathbf{S}) = \mathbf{V}g(\boldsymbol{\Lambda})\mathbf{V}^T$$

where we use the notation

$$g(\boldsymbol{\Lambda}) = \begin{pmatrix} g(\lambda_1) & & \\ & g(\lambda_2) & \\ & & g(\lambda_3) \end{pmatrix} \quad \text{and} \quad \boldsymbol{\Lambda} = \begin{pmatrix} \lambda_1 & & \\ & \lambda_2 & \\ & & \lambda_3 \end{pmatrix}.$$

We can derive the differentials of scalar inherited $\mathbf{g}$ using the expressions for the differentials of the eigen decomposition of $\mathbf{S}$. The eigen decomposition of the symmetric matrix $\mathbf{S}$ can be thought of as a function over $\mathbb{R}^{3\times3}_{\text{sym}}$: $\mathbf{V} : \mathbb{R}^{3\times3}_{\text{sym}} \to \mathbb{R}^{3\times3}_{\text{orth}}$ and $\boldsymbol{\Lambda} : \mathbb{R}^{3\times3}_{\text{sym}} \to \mathbb{R}^{3\times3}_{\text{diag}}$, or $\mathbf{V}(\mathbf{S})$ and $\boldsymbol{\Lambda}(\mathbf{S})$ to emphasize the dependent variable. By definition, we have the relation

$$\delta\mathbf{S} = \delta\mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^T + \mathbf{V}\delta\boldsymbol{\Lambda}\mathbf{V}^T + \mathbf{V}\boldsymbol{\Lambda}\delta\mathbf{V}^T$$

and since $\mathbf{V}^T\mathbf{V} = \mathbf{I}$,

$$\delta\mathbf{V}^T\mathbf{V} + \mathbf{V}^T\delta\mathbf{V} = \mathbf{0}.$$

Using $\mathbf{W} = \delta\mathbf{V}^T\mathbf{V}$, we see that $\mathbf{W}$ is skew symmetric and that

$$\mathbf{V}^T\delta\mathbf{S}\mathbf{V} = \mathbf{W}^T\boldsymbol{\Lambda} + \delta\boldsymbol{\Lambda} + \boldsymbol{\Lambda}\mathbf{W}.$$

Since $\mathbf{W}$ is skew symmetric, it can be written as

$$\mathbf{W} = \begin{pmatrix} 0 & \omega_3 & -\omega_2 \\ -\omega_3 & 0 & \omega_1 \\ \omega_2 & -\omega_1 & 0 \end{pmatrix}$$

and thus

$$\mathbf{V}^T \delta \mathbf{S} \mathbf{V} = \begin{pmatrix} \delta\lambda_1 & -\omega_3(\lambda_2 - \lambda_1) & \omega_2(\lambda_3 - \lambda_1) \\ -\omega_3(\lambda_2 - \lambda_1) & \delta\lambda_2 & -\omega_1(\lambda_3 - \lambda_1) \\ \omega_2(\lambda_3 - \lambda_1) & -\omega_1(\lambda_3 - \lambda_2) & \delta\lambda_3 \end{pmatrix}. \tag{4.1}$$

Thus denoting $\mathbf{A} = \mathbf{V}^T \delta\mathbf{S}\mathbf{V}$, we have the expressions

$$\omega_1 = -\frac{a_{32}}{\lambda_3 - \lambda_2},\ \omega_2 = \frac{a_{31}}{\lambda_3 - \lambda_1},\ \omega_3 = -\frac{a_{21}}{\lambda_2 - \lambda_1},\ \text{and } \delta\lambda_i = a_{ii},\ i = 1, 2, 3$$

Similar to the eigen decomposition

$$\mathbf{V}^T \delta\mathbf{g}\mathbf{V} = \mathbf{W}^T g(\mathbf{\Lambda}) + \delta g(\mathbf{\Lambda}) + g(\mathbf{\Lambda})\mathbf{W}$$

where

$$\delta g(\mathbf{\Lambda}) = \begin{pmatrix} g'(\lambda_1)\delta\lambda_1 & & \\ & g'(\lambda_2)\delta\lambda_2 & \\ & & g'(\lambda_3)\delta\lambda_3 \end{pmatrix}.$$

Thus,

$$\mathbf{V}^T \delta\mathbf{g}\mathbf{V} = \begin{pmatrix} g'(\lambda_1)a_{11} & \frac{g(\lambda_2)-g(\lambda_1)}{\lambda_2-\lambda_1}a_{21} & \frac{g(\lambda_3)-g(\lambda_1)}{\lambda_3-\lambda_1}a_{31} \\ \frac{g(\lambda_2)-g(\lambda_1)}{\lambda_2-\lambda_1}a_{21} & g'(\lambda_2)a_{22} & \frac{g(\lambda_3)-g(\lambda_2)}{\lambda_3-\lambda_2}a_{32} \\ \frac{g(\lambda_3)-g(\lambda_1)}{\lambda_3-\lambda_1}a_{31} & \frac{g(\lambda_3)-g(\lambda_2)}{\lambda_3-\lambda_2}a_{32} & g'(\lambda_3)a_{33} \end{pmatrix}$$

and

$$\delta\mathbf{g} = \mathbf{V} \begin{pmatrix} g'(\lambda_1)a_{11} & \frac{g(\lambda_2)-g(\lambda_1)}{\lambda_2-\lambda_1}a_{21} & \frac{g(\lambda_3)-g(\lambda_1)}{\lambda_3-\lambda_1}a_{31} \\ \frac{g(\lambda_2)-g(\lambda_1)}{\lambda_2-\lambda_1}a_{21} & g'(\lambda_2)a_{22} & \frac{g(\lambda_3)-g(\lambda_2)}{\lambda_3-\lambda_2}a_{32} \\ \frac{g(\lambda_3)-g(\lambda_1)}{\lambda_3-\lambda_1}a_{31} & \frac{g(\lambda_3)-g(\lambda_2)}{\lambda_3-\lambda_2}a_{32} & g'(\lambda_3)a_{33} \end{pmatrix} \mathbf{V}^T.$$

We can rewrite this in terms of the matrix

$$\mathbf{B} = \begin{pmatrix} g'(\lambda_1) & \frac{g(\lambda_2)-g(\lambda_1)}{\lambda_2-\lambda_1} & \frac{g(\lambda_3)-g(\lambda_1)}{\lambda_3-\lambda_1} \\ \frac{g(\lambda_2)-g(\lambda_1)}{\lambda_2-\lambda_1} & g'(\lambda_2) & \frac{g(\lambda_3)-g(\lambda_2)}{\lambda_3-\lambda_2} \\ \frac{g(\lambda_3)-g(\lambda_1)}{\lambda_3-\lambda_1} & \frac{g(\lambda_3)-g(\lambda_2)}{\lambda_3-\lambda_2} & g'(\lambda_3) \end{pmatrix}$$

using the Hadamard product (or entry-wise product) where the i, j entry of $\mathbf{A} \circ \mathbf{B}$ is $A_{ij}B_{ij}$ (with no summation on the repeated indices). That is,

$$\delta\mathbf{g} = \mathbf{V}\left(\mathbf{B} \circ \left(\mathbf{V}^T \delta\mathbf{S}\mathbf{V}\right)\right)\mathbf{V}^T \tag{4.2}$$

4.1.1 Symmetric tensors

This result generalizes to functions over symmetric tensors. If $\mathbf{g} : \mathcal{V}^2_{\text{sym}} \to \mathcal{V}^2_{\text{sym}}$, then

$$\delta \mathbf{g} = ([\mathbf{B}](\mathbf{S}) \circ [\delta \mathbf{S}])_{kl}\, \mathbf{u}_k \otimes \mathbf{u}_l \tag{4.3}$$

where $\mathbf{S} = \sum_i \lambda_i \mathbf{u}_i \times \mathbf{u}_i$ is the eigenvalue decomposition of $\mathbf{S}$. $[\delta \mathbf{S}], [\mathbf{B}](\mathbf{S}) \in \mathbb{R}^{3\times 3}$ and $[\mathbf{B}](\mathbf{S}) \circ [\delta \mathbf{S}] \in \mathbb{R}^{3\times 3}$ is their Hadamard product. The entries in the matrix $[\delta \mathbf{S}]$ are $[\delta \mathbf{S}]_{ij} = \mathbf{u}_i \cdot (\delta \mathbf{S} \mathbf{u}_j)$, i.e. it is the expression of $\delta \mathbf{S}$ in the eigenbasis of $\mathbf{S}$. We would assume the convention $\lambda_1 \geq \lambda_2 \geq \lambda_3$ to make the mapping $[\mathbf{B}] :\to \mathcal{V}^2_{\text{sym}}$ well defined from

$$[\mathbf{B}](\mathbf{S}) = \begin{pmatrix} g'(\lambda_1) & \frac{g(\lambda_2)-g(\lambda_1)}{\lambda_2-\lambda_1} & \frac{g(\lambda_3)-g(\lambda_1)}{\lambda_3-\lambda_1} \\ \frac{g(\lambda_2)-g(\lambda_1)}{\lambda_2-\lambda_1} & g'(\lambda_2) & \frac{g(\lambda_3)-g(\lambda_2)}{\lambda_3-\lambda_2} \\ \frac{g(\lambda_3)-g(\lambda_1)}{\lambda_3-\lambda_1} & \frac{g(\lambda_3)-g(\lambda_2)}{\lambda_3-\lambda_2} & g'(\lambda_3) \end{pmatrix}.$$

4.1.2 Scalar functions of symmetric tensors

Let $f : \mathcal{V}^2_{\text{sym}} \to \mathbb{R}$ with $f(\mathbf{S}) = \hat{f}(\lambda_1, \lambda_2, \lambda_3) = \tilde{f}(I(\mathbf{S}), II(\mathbf{S}), III(\mathbf{S}))$ where

$$I(\mathbf{S}) = \lambda_1 + \lambda_2 + \lambda_3,\ II(\mathbf{S}) = \lambda_1\lambda_2 + \lambda_1\lambda_3 + \lambda_2\lambda_3,\ III(\mathbf{S}) = \lambda_1\lambda_2\lambda_3. \tag{4.4}$$

Using Equation (4.1), we can conclude

$$\delta f = \frac{\partial f}{\partial \mathbf{S}}(\mathbf{S}) = \sum_i \frac{\partial \hat{f}}{\partial \lambda_i}(\lambda_1, \lambda_2, \lambda_3)\delta\lambda_i = \sum_i \frac{\partial \hat{f}}{\partial \lambda_i}(\lambda_1, \lambda_2, \lambda_3)\mathbf{u}_i \cdot (\delta \mathbf{S} \mathbf{u}_i)$$

Thus, the derivative is given by

$$\frac{\partial f}{\partial \mathbf{S}}(\mathbf{S}) = \sum_i \frac{\partial \hat{f}}{\partial \lambda_i}(\lambda_1, \lambda_2, \lambda_3)\mathbf{u}_i \otimes \mathbf{u}_i. \tag{4.5}$$

REFERENCES

[1] J. N. Tsitsiklis, Efficient algorithms for globally optimal trajectories, IEEE Trans Auto Cont 40 (9) (1995) 1528–1538.

[2] J. A. Sethian, A fast marching level set method for monotonically advancing fronts, Proc Nat Acad Sci 93 (4) (1996) 1591–1595.

[3] H. Zhao, A fast sweeping method for eikonal equations, Math Comp 74 (250) (2005) 603–627.

[4] B. Lee, J. Darbon, S. Osher, M. Kang, Revisiting the redistancing problem using the hopf–lax formula, J Comp Phys 330 (2017) 268–281.

[5] J. Darbon, S. Osher, Algorithms for overcoming the curse of dimensionality for certain hamilton–jacobi equations arising in control theory and elsewhere, Res Math Sci 3 (1) (2016) 19.

[6] S. Osher, J. A. Sethian, Fronts propagating with curvature-dependent speed: algorithms based on hamilton-jacobi formulations, J Comp Phys 79 (1) (1988) 12–49.

[7] S. Osher, R. P. Fedkiw, Level set methods and dynamic implicit surfaces, Applied mathematical science, Springer, New York, N.Y., 2003.
URL http://opac.inria.fr/record=b1099358

[8] M. Sussman, P. Smereka, S. Osher, A level set approach for computing solutions to incompressible two-phase flow, J Comp Phys 114 (1) (1994) 146–159.

[9] G. Russo, P. Smereka, A remark on computing distance functions, J Comp Phys 163 (1) (2000) 51–67.

[10] M. G. Crandall, P.-L. Lions, Two approximations of solutions of hamilton-jacobi equations, Math Comp 43 (167) (1984) 1–19.

[11] L. Evans, Partial Differential Equations, Graduate studies in mathematics, American Mathematical Society, 2010.
URL https://books.google.com/books?id=Xnu0o_EJrCQC

[12] S. Osher, A level set formulation for the solution of the dirichlet problem for hamilton–jacobi equations, SIAM Journal on Mathematical Analysis 24 (5) (1993) 1145–1152.

[13] E. W. Dijkstra, A note on two problems in connexion with graphs, Numer Math 1 (1) (1959) 269–271.

[14] M. Detrixhe, F. Gibou, C. Min, A parallel fast sweeping method for the eikonal equation, J Comp Phys 237 (2013) 46–55.

[15] M. Sussman, E. Fatemi, An efficient, interface-preserving level set redistancing algorithm and its application to interfacial incompressible fluid flow, SIAM J Sci Comp 20 (4) (1999) 1165–1191.

[16] H. Zhao, Parallel implementations of the fast sweeping method, J Comp Math (2007) 421–429.

[17] M. Detrixhe, F. Gibou, Hybrid massively parallel fast sweeping method for static hamilton–jacobi equations, J Comp Phys 322 (2016) 199–223.

[18] M. Herrmann, A domain decomposition parallelization of the fast marching method, Tech. rep., DTIC Document (2003).

[19] J. Yang, F. Stern, A highly scalable massively parallel fast marching method for the eikonal equation, J Comp Phys 332 (2017) 333–362.

[20] W.-k. Jeong, R. Whitaker, et al., A fast eikonal equation solver for parallel systems, in: SIAM Conf Comp Sci Eng, Citeseer, 2007.

[21] M. Breuß, E. Cristiani, P. Gwosdek, O. Vogel, An adaptive domain-decomposition technique for parallelization of the fast marching method, App Math Comp 218 (1) (2011) 32–44.

[22] F. Dang, N. Emad, Fast iterative method in solving eikonal equations: a multi-level parallel approach, Proc Comp Sci 29 (2014) 1859–1869.

[23] E. Hopf, Generalized solutions of non-linear equations of first order, Journal of Mathematics and Mechanics 14 (6) (1965) 951–973.

[24] P. D. Lax, Hyperbolic systems of conservation laws ii, Communications on pure and applied mathematics 10 (4) (1957) 537–566.

[25] G. Daviet, F. Bertails-Descoubes, A semi-implicit material point method for the continuum simulation of granular materials, ACM Trans Graph 35 (4) (2016) 102:1–102:13.

[26] K. W. Luczynski, A. Steiger-Thirsfeld, J. Bernardi, J. Eberhardsteiner, C. Hellmich, Extracellular bone matrix exhibits hardening elastoplasticity and more than double cortical strength: evidence from homogeneous compression of non-tapered single micron-sized pillars welded to a rigid substrate, Journal of the mechanical behavior of biomedical materials 52 (2015) 51–62.

[27] I. N. Vladimirov, M. P. Pietryga, S. Reese, Anisotropic finite elastoplasticity with nonlinear kinematic and isotropic hardening and application to sheet metal forming, International Journal of Plasticity 26 (5) (2010) 659–687.

[28] J. Zhang, Z. Lin, A. Wong, N. Kikuchi, V. Li, A. Yee, G. Nusholtz, Constitutive modeling and material characterization of polymeric foams, Journal of engineering materials and technology 119 (3) (1997) 284–291.

[29] E. H. Lee, Elastic-plastic deformation at finite strains, ASME, 1969.

[30] R. v. Mises, Mechanik der festen körper im plastisch-deformablen zustand, Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse 1913 (4) (1913) 582–592.

[31] M.-h. Yu, Advances in strength theories for materials under complex stress state in the 20th century, Applied Mechanics Reviews 55 (3) (2002) 169–218.

[32] D. C. Drucker, W. Prager, Soil mechanics and plastic analysis or limit design, Quarterly of applied mathematics 10 (2) (1952) 157–165.

[33] O. Gonzalez, A. Stuart, A first course in continuum mechanics, Cambridge University Press, 2008.

[34] J. Bonet, R. Wood, Nonlinear continuum mechanics for finite element analysis, Cambridge University Press, 2008.

[35] V. Lubarda, Constitutive analysis of large elasto-plastic deformation based on the multiplicative decomposition of deformation gradient, International Journal of Solids and Structures 27 (7) (1991) 885–895.

[36] V. A. Lubarda, Elastoplasticity theory, CRC press, 2001.

[37] R. Hill, A variational principle of maximum plastic work in classical plasticity, The Quarterly Journal of Mechanics and Applied Mathematics 1 (1) (1948) 18–28.

[38] R. Hill, The mathematical theory of plasticity, Vol. 11, Oxford university press, 1998.

[39] C. Mast, Modeling landslide-induced flow interactions with structures using the material point method, Ph.D. thesis (2013).

[40] D. Sulsky, S. Zhou, H. Schreyer, Application of a particle-in-cell method to solid mechanics, Comp Phys Comm 87 (1) (1995) 236–252.

[41] J. Brackbill, H. Ruppel, FLIP: A method for adaptively zoned, particle-in-cell calculations of fluid flows in two dimensions, J Comp Phys 65 (1986) 314–343.

[42] F. Harlow, The particle-in-cell method for numerical solution of problems in fluid dynamics, Meth Comp Phys 3 (1964) 319–343.

[43] F. Harlow, E. Welch, Numerical calculation of time dependent viscous flow of fluid with a free surface, Phys Fluid 8 (12) (1965) 2182–2189.

[44] G. Klár, T. Gast, A. Pradhana, C. Fu, C. Schroeder, C. Jiang, J. Teran, Drucker-prager elastoplasticity for sand animation, ACM Trans Graph 35 (4) (2016) 103:1–103:12.

[45] C. Jiang, C. Schroeder, A. Selle, J. Teran, A. Stomakhin, The affine particle-in-cell method, ACM Trans Graph 34 (4) (2015) 51:1–51:10.

[46] M. Steffen, R. M. Kirby, M. Berzins, Analysis and reduction of quadrature errors in the material point method (MPM), Int J Numer Meth Eng 76 (6) (2008) 922–948.

[47] C. Jiang, C. Schroeder, J. Teran, An Angular Momentum Conserving Affine-Particle-In-Cell Method, ArXiv e-printsarXiv:1603.06188.

[48] J. Brackbill, The ringing instability in particle-in-cell calculations of low-speed flow, J Comp Phys 75 (2) (1988) 469–492.

[49] C. E. Gritton, Ringing instabilities in particle methods, Ph.D. thesis, The University of Utah (2014).

[50] T. Gast, C. Schroeder, A. Stomakhin, C. Jiang, J. Teran, Optimization integrator for large time steps, IEEE Trans Vis Comp Graph 21 (10) (2015) 1103–1115.