CD-MPM: Continuum Damage Material Point Methods for Dynamic Fracture Animation

Siggraph 2019

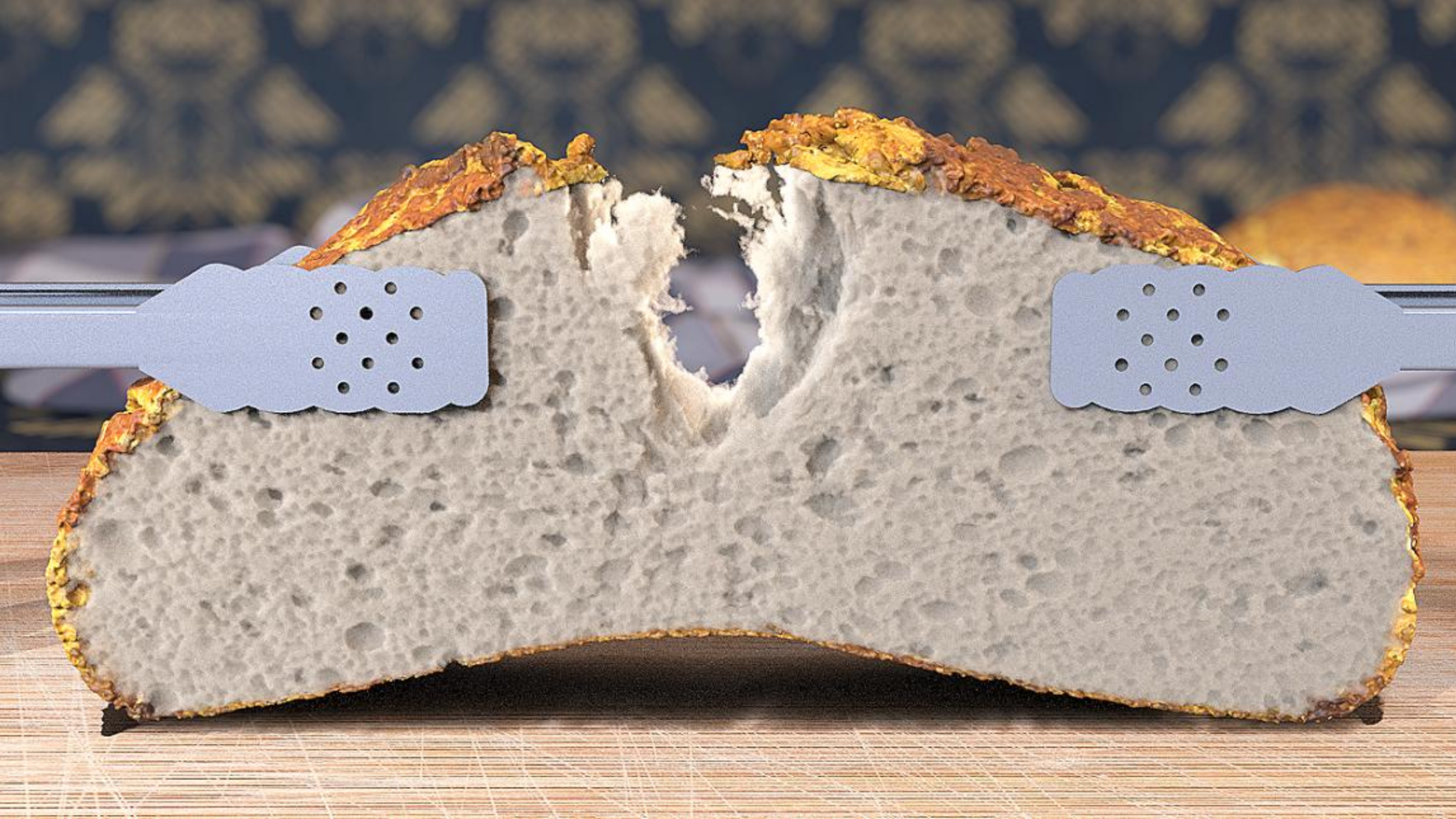
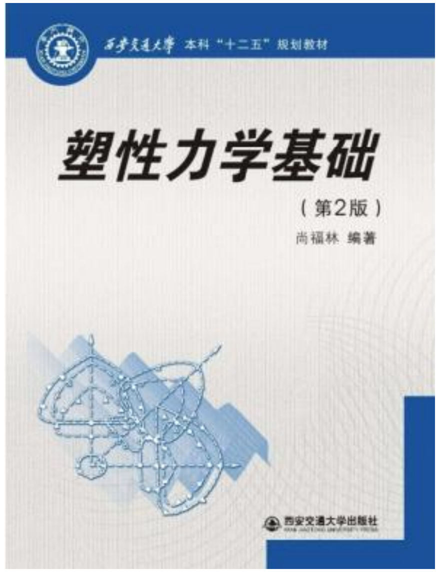


Table of Contents



Author

- 老熟人

: Joshuah Wolper, Yu Fang, Minchen Li, Jiecong Lu, Ming Gao, Chenfanfu Jiang

~~Fluid~~ Continuum Simulation in Computer Graphics

- ~~Fluid~~ Continuum
  - Macroscale
    - Euler
      - Finite Difference (FDM)
      - Finite Volume (FVM)
      - Finite Element (FEM)
      - Conventional CFD
    - Lagrange
      - Smoothed Particle Hydrodynamics (SPH)
      - Position Based Fluid (PBF)
    - Hybrid
      - Semi-Lagrange
      - Fluid Implicit Particles (FLIP)
      - Affine Particle-in-Cell (APIC)
      - **Material Point Method (MPM)**
      - CG
  - Mesoscale — Lattice Boltzmann Method (LBM)
  - Microscale — Molecular Dynamics (MD)

today's topic

Section 1.

Deformation

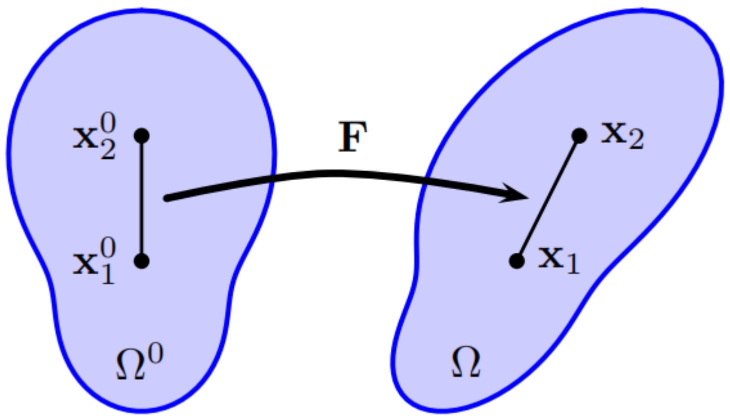
Deformation

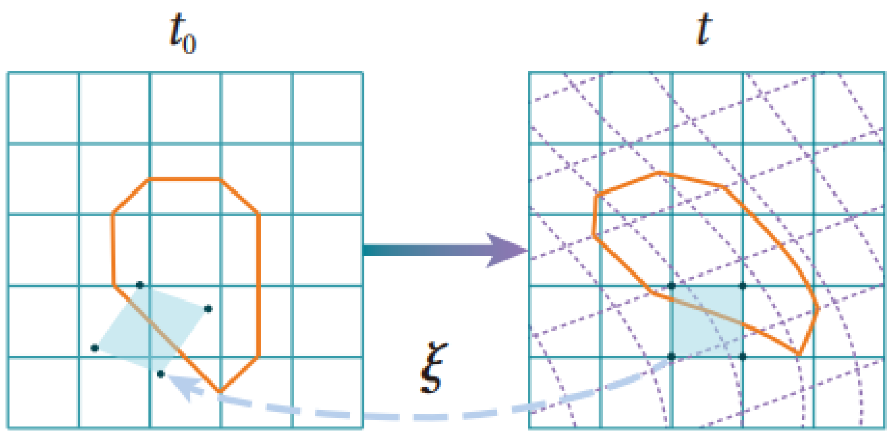
- 如何形式化定义形变?
- 假设无形变空间是$\mathbf{X}$，形变空间是$\mathbf{x}$，那么形变就是$\phi(\mathbf{X},t):\Omega^0\to\Omega^t$，即

$$\mathbf{x}=\mathbf{x}(\mathbf{X},t)=\phi(\mathbf{X},t)$$

- 例如，对于刚体形变，$\mathbf{x}=\mathbf{R}\mathbf{X}+\mathbf{b}$
- 定义形变梯度表示任意时刻形变的雅可比矩阵:

$$\mathbf{F}(\mathbf{X},t)=\frac{\partial\phi}{\partial\mathbf{X}}(\mathbf{X},t)=\frac{\partial\mathbf{x}}{\partial\mathbf{X}}(\mathbf{X},t)$$

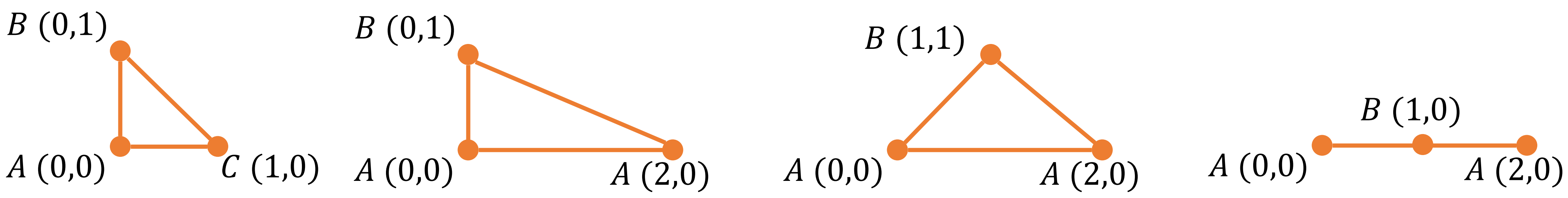




Deformation

- 形变梯度的行列式记为J，表示了体积变化
  - $J > 1$表示体积增加，$J < 1$表示体积减小
  - 对于刚体形变，$\mathbf{x} = \mathbf{R}\mathbf{X} + \mathbf{b}$，$\mathbf{F} = \mathbf{R}$，则$J = 1$。说明刚体形变不影响体积，很合理。
- 实际上，ϕ建立起了Euler视角和Lagrange视角之间的联系
  - $\mathbf{X} = \phi^{-1}(\mathbf{x}, t)$，假设$\phi$双射
  - $\mathbf{v}(\mathbf{x}, t)$表示当前点$\mathbf{x}$当下的速度，$\mathbf{v}(\mathbf{x}, t) = \mathbf{V}(\phi^{-1}(\mathbf{x}, t), t)$：$\phi^{-1}$找到了$\mathbf{x}$初始对应点$\mathbf{X}$
  - $\mathbf{V}(\mathbf{X}, \boldsymbol{t})$表示初始点$\mathbf{X}$当下的速度，$\mathbf{V}(\mathbf{X}, t) = \mathbf{v}(\phi(\mathbf{X}, t), t)$：$\phi$找到了初始点$\mathbf{X}$的当前位置
  - 对于其他物理量，有类似的性质
- 物质导数
  - $\frac{D}{Dt} f(\mathbf{x}, t) = \frac{\partial f}{\partial t}(\mathbf{x}, t) + \sum_j \frac{\partial f}{\partial x_j}(\mathbf{x}, t) v_j(\mathbf{x}, t)$
  - 分成当地导数和迁移导数两部分

Example



State 0　State 1　State 2　State 3

已知如上四个状态，如何求State 1、2、3的形变梯度?

考虑$\mathrm{d}\mathbf{x} = \mathbf{F}\mathrm{d}\mathbf{X}$，在一个三角形微元内，可以认为形变遵循坐标变换

因此构造两个状态的基坐标系$\mathbf{A}, \mathbf{B}$，则$\mathbf{B}\mathbf{A}^{-1}$就是目标形变梯度。

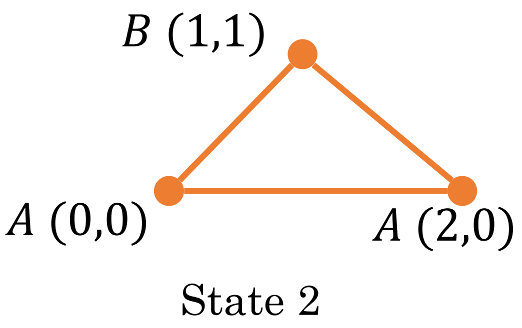
$$\mathbf{A} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\mathbf{B}_1 = \mathbf{F} = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$$
$$J = 2$$

$$\mathbf{B}_2 = \mathbf{F} = \begin{bmatrix} 2 & 1 \\ 0 & 1 \end{bmatrix}$$
$$J = 2$$

$$\mathbf{B}_3 = \mathbf{F} = \begin{bmatrix} 2 & 1 \\ 0 & 0 \end{bmatrix}$$
$$J = 0$$

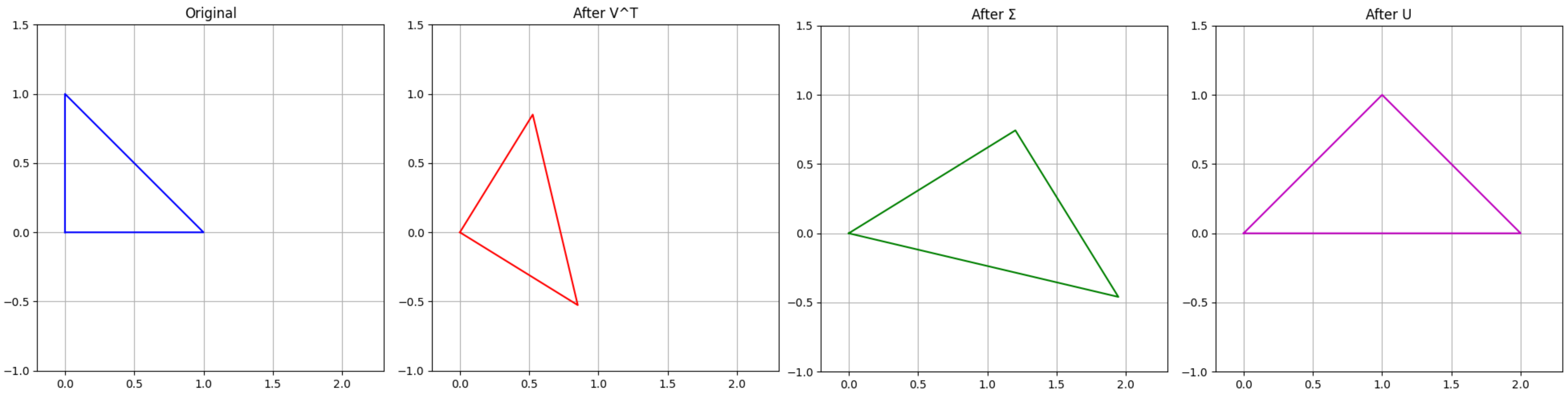
Example



$$\mathbf{B_2} = \mathbf{F} = \begin{bmatrix} 2 & 1 \\ 0 & 1 \end{bmatrix}$$
$$J = 2$$

这个东西表示什么？考虑SVD分解…

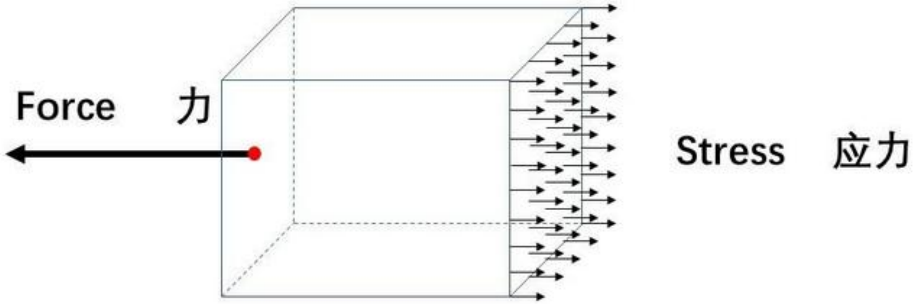
$$\mathbf{F} = \mathbf{U\Sigma V}^T$$



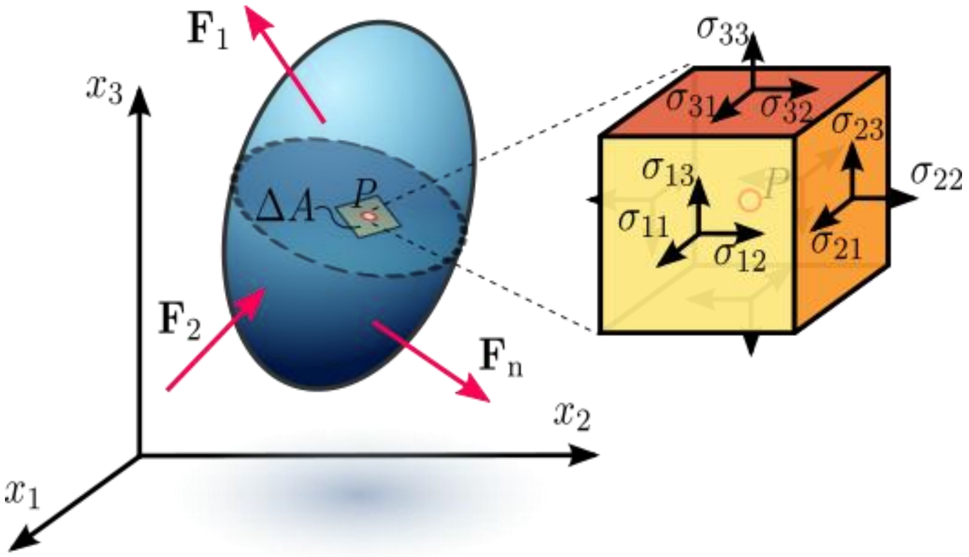
Stress

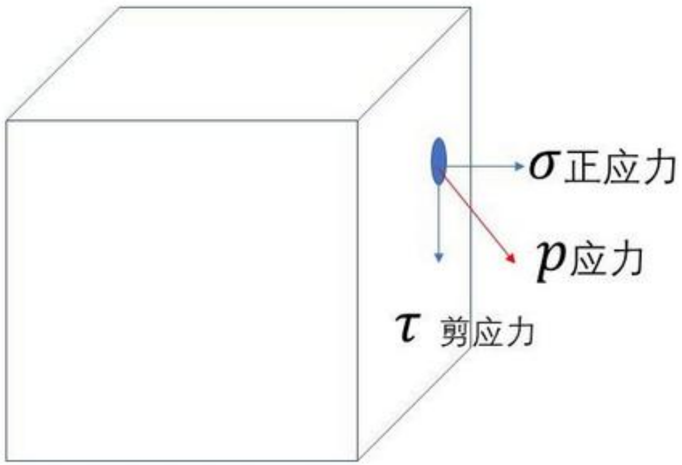
- 应力：单位面积所受的承受力

$$\sigma_{ij} = \lim_{\Delta A_i \to 0} \frac{\Delta F_j}{\Delta A_i}$$



- 应力张量可以分解成压强和剪应力两部分：
  - $p = -\frac{1}{3}\mathrm{tr}(\boldsymbol{\sigma}), \boldsymbol{\tau} = \boldsymbol{\sigma} + p\mathbf{I}$
  - 考虑流体，只有垂直于每个面的力，那么每个面都有$p = \frac{F}{S}$





Section 2.

Hyperelastic

First Piola-Kirchoff Stress

- 假设Strain Energy Density Function是Ψ，一阶Piola-Kirchoff张量定义为

$$\mathbf{P} = \frac{\partial \Psi}{\partial \mathbf{F}}$$

- 假设材质有**各向同性**，那么我们可以把Ψ写成$\mathbf{F}^{\mathrm{T}}\mathbf{F}$的函数，称为right Cauchy-Green Tensor，一般记作$\mathbf{C}$：
  - 如果带有旋转，比如$\mathbf{F} = \mathbf{RQ}$，那么$\mathbf{F}^{\mathrm{T}}\mathbf{F} = \mathbf{Q}^{\mathrm{T}}\mathbf{R}^{\mathrm{T}}\mathbf{RQ} = \mathbf{Q}^{\mathrm{T}}\mathbf{Q}$，可以约去旋转部分
  - 同时可以发现$\mathbf{C}$是一个对称矩阵：$\mathbf{C}_{ij} = \mathbf{F}_{ik}\mathbf{F}_{jk} = \mathbf{C}_{ji}$
- 考虑$\mathbf{C}$的特征值：

$$\left(\begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix} - \lambda \mathbf{I}\right)\mathbf{v} = 0 \Rightarrow \begin{bmatrix} c_{11} - \lambda & c_{12} & c_{13} \\ c_{21} & c_{22} - \lambda & c_{23} \\ c_{31} & c_{32} & c_{33} - \lambda \end{bmatrix}\mathbf{v} = 0$$

- 可以转换成如下特征多项式

$$\lambda^3 - I_C\lambda^2 - II_C\lambda - III_C = 0$$

First Piola-Kirchoff Stress (Cont.)

- 如果我们实际求一下，就会发现

$$I_{\mathbf{C}} = \mathrm{tr}(\mathbf{C})$$
$$II_{\mathbf{C}} = \frac{1}{2}\left(I_C^2 - \mathrm{tr}(\mathbf{CC})\right)$$
$$III_{\mathbf{C}} = \det(\mathbf{C}) = J^2$$

- 这三个量称为各向同性不变量。在一部分文献中，也直接用下面的关系来表示二阶

$$II_{\mathbf{C}} = \mathrm{tr}(\mathbf{CC})$$

- ……它的几何含义?
- 在图形学中，用$\mathbf{F} = \mathbf{U\Sigma V^T}$进行Polar SVD分解，$\mathbf{U}, \mathbf{V}$表示旋转，$\mathbf{\Sigma}$表示缩放。而缩放与**奇异值**强相关，因此$\mathbf{F}$的**奇异值**$\sigma_1, \sigma_2, \sigma_3$就表示了形变特性。则完全可以用$\mathbf{F}$的**奇异值**来表示这三个参量：

$$I_C = \sigma_i^2, II_C = \sigma_i^2\sigma_j^2\delta_{ij}, III_C = \sigma_i^4$$

回忆：矩阵奇异值等于A^TA或AA^T非负特征值的平方根

Neo-Hookean

- 下面我们可以写出Energy Density Function的具体形式：

$$\Psi(\mathbf{F}) = \frac{\mu}{2}(I_C - d) - \mu \ln J + \frac{\lambda}{2}\ln^2 J$$

- d表示维度：二维取2，三维取3；假设杨氏模量为E，泊松比为ν，有

$$\mu = \frac{E}{2(1+\nu)}, \lambda = \frac{E\nu}{(1+\nu)(1-2\nu)}$$

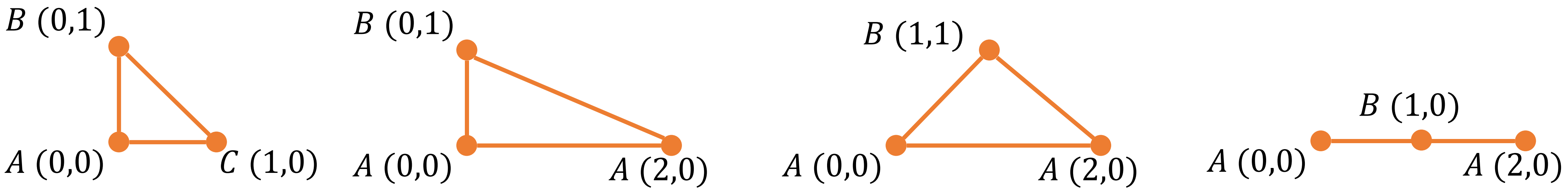
- 回想一下要计算的$\mathbf{P} = \frac{\partial \Psi}{\partial \mathbf{F}}$，对上式求导

$$\mathbf{P} = \mu(\mathbf{F} - \mathbf{F}^{-T}) + \lambda \ln J\, \mathbf{F}^{-T}$$

- 这基于$\frac{\mathrm{d}|\mathbf{A}|}{\mathrm{d}\mathbf{A}} = |\mathbf{A}|\mathbf{A}^{-T}$，其中$|\mathbf{A}|$表示$\det \mathbf{A}$。
- 带进去刚体运动$\mathbf{F} = \mathbf{R}$，则$I_\mathbf{C} = 3$，$J = 1$，最后能量为0，这也是能量最小值。如果满足$\det \mathbf{F} > 0$，则有$\Psi(\mathbf{F}) > 0$。

Example

$$\Psi(\mathbf{F}) = \frac{\mu}{2}(I_C - d) - \mu \ln J + \frac{\lambda}{2}\ln^2 J$$



State 0 | State 1 | State 2 | State 3

State 0

$$\mathbf{A} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

State 1

$$\mathbf{B_1} = \mathbf{F} = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$$

$$J = 2$$

$$\Psi(\mathbf{F}) = \frac{\mu}{2} - \mu \ln 2 + \frac{\lambda}{2}\ln^2 2$$

$$\mathbf{F}^{-T} = \begin{bmatrix} 0.5 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\mathbf{P} = \mu \begin{bmatrix} 1.5 & 0 \\ 0 & 0 \end{bmatrix} + \lambda \ln 2 \begin{bmatrix} 0.5 & 0 \\ 0 & 1 \end{bmatrix}$$

State 2

$$\mathbf{B_2} = \mathbf{F} = \begin{bmatrix} 2 & 1 \\ 0 & 1 \end{bmatrix}$$

$$J = 2$$

$$\Psi(\mathbf{F}) = \frac{\mu}{2} - \mu \ln 2 + \frac{\lambda}{2}\ln^2 2$$

$$\mathbf{F}^{-T} = \begin{bmatrix} 0.5 & 0 \\ -0.5 & 1 \end{bmatrix}$$

$$\mathbf{P} = \mu \begin{bmatrix} 1.5 & 1 \\ 0.5 & 0 \end{bmatrix} + \lambda \ln 2 \begin{bmatrix} 0.5 & 0 \\ -0.5 & 1 \end{bmatrix}$$

State 3

$$\mathbf{B_3} = \mathbf{F} = \begin{bmatrix} 2 & 1 \\ 0 & 0 \end{bmatrix}$$

$$J = 0$$

$$\Psi(\mathbf{F}) = +\infty!$$

Fixed Corotated Constitutive Model

- 解决了旋转后体积可能变大的问题
- 假设$\mathbf{F} = \mathbf{U\Sigma V}^T$，特征值为$\sigma_i$，则有如下模型

$$\Psi(\mathbf{F}) = \sum_i \mu(\sigma_i - 1)^2 + \frac{\lambda}{2}(J - 1)^2$$

- 考虑极分解$\mathbf{F} = \mathbf{RS}$，$\mathbf{R}$是正交矩阵，$\mathbf{S}$是半正定矩阵：

$$\mathbf{F} = \mathbf{U\Sigma V}^T = \mathbf{UV}^T\mathbf{V\Sigma V}^T = \mathbf{RS}$$

- 那么$\mathbf{R}^\mathbf{T}\mathbf{F} = \mathbf{V\Sigma V}^\mathbf{T}$，所以$\mathrm{tr}(\mathbf{\Sigma}) = \mathrm{tr}(\mathbf{R}^\mathbf{T}\mathbf{F})$。根据这一结论可以得出，一阶PK张量

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

- 在实践中，如果使用隐式积分，还需要用到一阶PK张量的导数，这一推导结论在附录1中给出。
- 附录2给出了一种求解弹性张量的通用方法。
- 特别的，Cauchy Stress $\boldsymbol{\sigma} = \frac{1}{J}\mathbf{PF}^T$

Section 3.

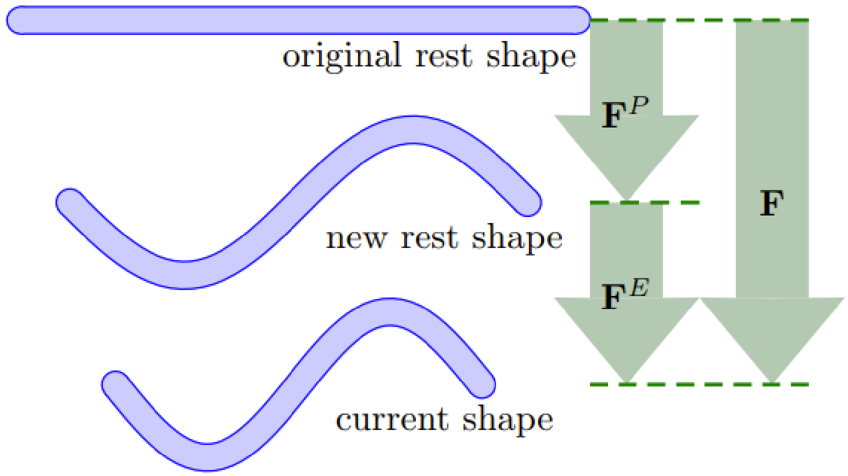
Plasticity

Plasticity..?

- 简单的思路：把形变梯度分解成弹性和塑性部分

$$\mathbf{F} = \mathbf{F}_E\mathbf{F}_P$$

- $\mathbf{F}_P$表示无记忆的部分：形变后不可恢复
- 考虑右面的金属杆，把它窝弯了之后……
- 此时，弹性响应只跟$\mathbf{F}_E$有关



- 一种直观的想法：每次更新F的时候判断超没超限度，超了就重新更新到记忆性的塑性里
- 对弹性项的奇异值做一个限制，超过就认为是塑性了……
- 形式化描述

Simplied Plasiticity

- 假设我们定义弹性范围是$[1-\theta_c, 1+\theta_s]$，从n时间步转换到$n+1$时间步，所有的形变都是弹性形变

$$\mathbf{F}^{n+1} = \tilde{\mathbf{F}}_E^{n+1}\mathbf{F}_P^n$$

- 对$\tilde{\mathbf{F}}_E^{n+1}$进行分解，假设奇异值是$\tilde{\sigma}_{Ei}^{n+1}$，对每个奇异值进行clamp操作，得到

$$\mathbf{F}_E^{n+1} = \mathbf{U}_E^{n+1}\mathrm{diag}\left(\mathrm{clamp}\left(\tilde{\sigma}_{Ei}^{n+1}, 1-\theta_c, 1+\theta_s\right)\right)\mathbf{V}_E^{n+1^T}$$

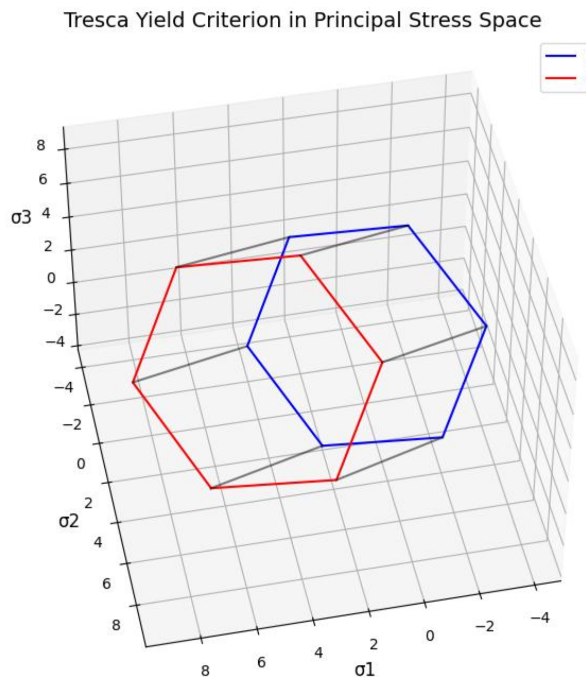
- 再把对应的塑性项更新回来

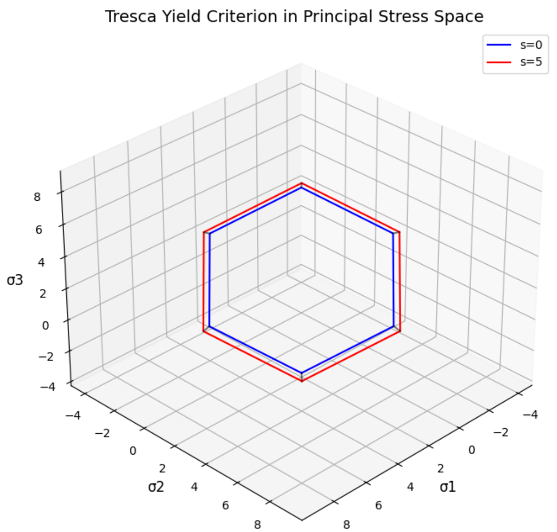
$$\mathbf{F}_P^{n+1} = (\mathbf{F}_E^{n+1})^{-1}\mathbf{F}^{n+1}$$

- 有没有更加物理准确的描述?
- 实际上我们要做的就是如何计算$\mathbf{F}_E^{n+1}$来完成投影操作，我们介绍常用的Drucker-Prager方法。

屈服准则

- 屈服准则：什么条件下进入塑性状态
- 最简单的准则：Tresca准则
- 假设考虑Kirhoff应力$\boldsymbol{\tau}$，它的核心形变由三个特征值$\sigma_1, \sigma_2, \sigma_3$决定，Tresca准则说明：最大的应力与最小的应力之间距离大于某个值的时候进入塑性
- 它是什么形状？六棱柱
- 它的截面是什么特征？一个正六边形
- 从什么地方截？$x = y = z$方向
- 我们把$p = -\frac{1}{d}\mathrm{tr}(\boldsymbol{\tau})$称为**静水压力**，表示了材料的等向性压缩。如果只有纯静水压力影响，材料只有体积变化而无剪切。Tresca准则下屈服与静水压力无关。





屈服准则

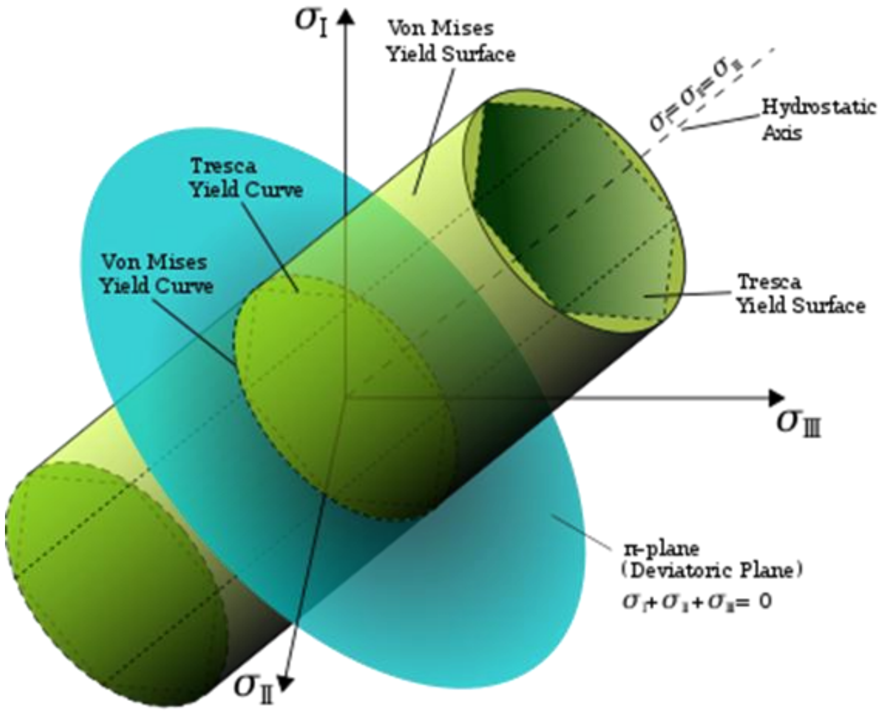
- Von-Mises准则
- 上面的屈服准则用的大于等于太难处理了，为什么不直接简化成π平面上一个圆?
- 为了让它仍然保证静水压力无关，使用如下等效应力

$$q = \sqrt{\frac{1}{2}[(\sigma_1 - \sigma_2)^2 + (\sigma_2 - \sigma_3)^2 + (\sigma_3 - \sigma_1)^2]}$$

- 为了方便计算，引入偏应力张量$\mathbf{s} = \boldsymbol{\tau} + p\mathbf{I}$，则偏应力张量反应了无体积变化的部分。则有如下关系：

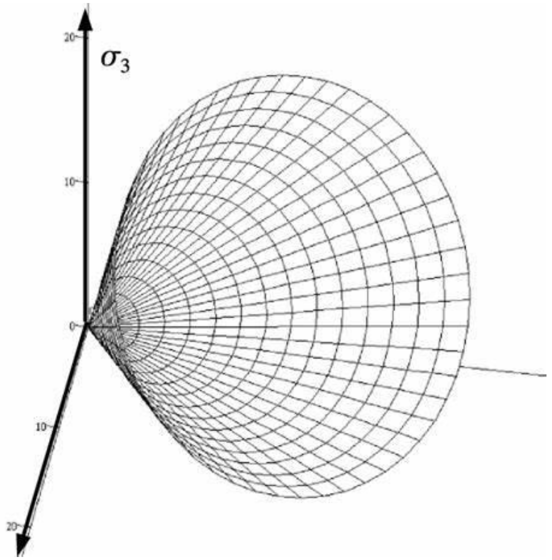
$$q = \sqrt{\frac{3}{2}}\|\mathbf{s}\|_F$$

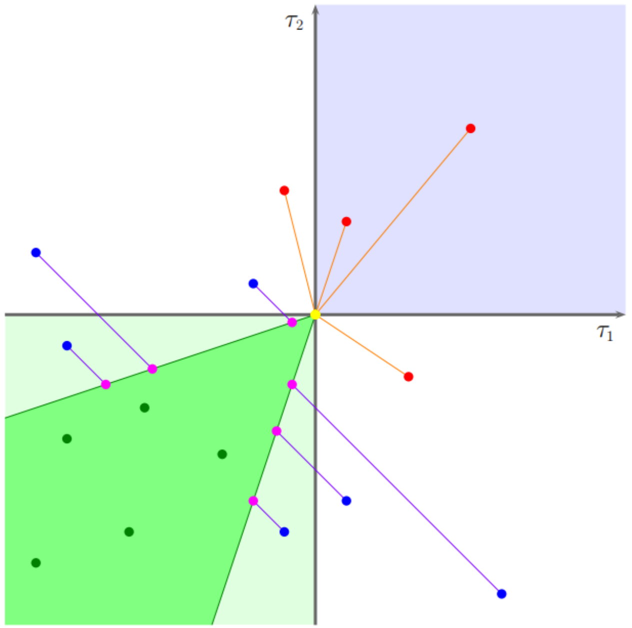
- 这样，Von-Mises准则确立一个圆柱体



屈服准则

- 但是真的跟静压力无关吗？
- 一个直观的观察：对于摩擦形材料（比如沙子和岩石），我们使劲按压的时候它发生滑动比较难
- 所以静水压力越大，就越难发生塑性流动
- 怎么办？给屈服条件加一个静水压力的补正
- Drucker-Prager模型
- 屈服准则是一个圆锥面：
  - 原点，表示体积不变化，没有抗拉能力
  - 侧壁，表示有明显的屈服极限
- 如果用来模拟沙子或者雪：
  - 红色部分表示产生拉伸，直接对应到原点
  - 蓝色部分表示压缩超过限度，投影到圆锥侧壁上



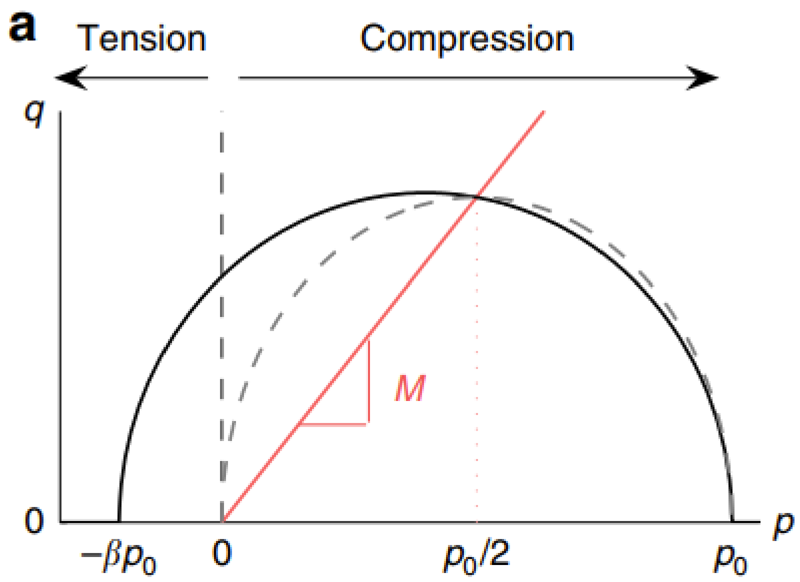


屈服准则

- 还有没有其他应该考虑的因素?
  - 过压时抗剪能力反而变差
  - 如果不是仿真沙子，应该有拉伸
- 修改的Cam-Clay模型

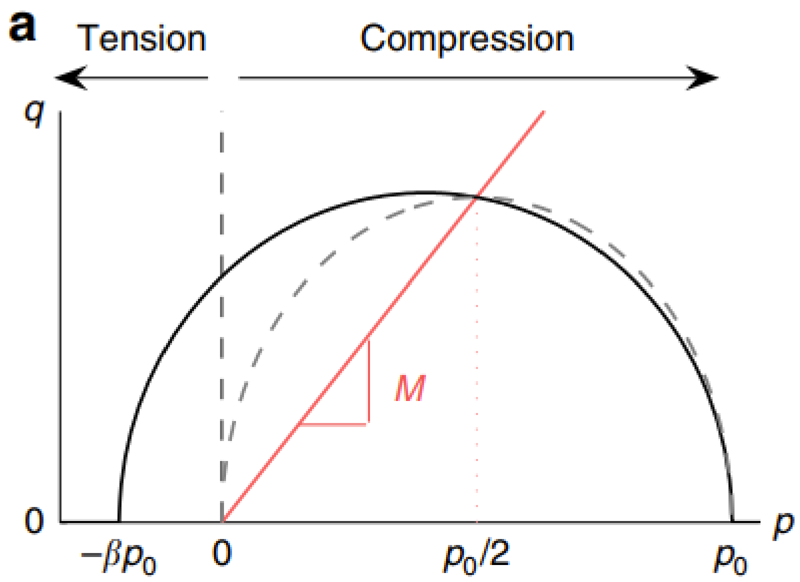
$$y(p,q) = q^2(1+2\beta) + M^2(p+\beta p_0)(p-p_0)$$
$$p_0 = K\sinh(\xi\,\max(-a,0))$$

- 屈服准则是个椭球
  - p_0表示硬化压力
  - M表示关键状态线斜率，反应摩擦大小
  - β表示拉伸和压缩的状态
- 那么如何把屈服准则和前面介绍的东西联系在一起?



屈服准则

- 直观感受:
  - β越大，抗拉伸能力越强
  - α越大，越容易达到破碎临界点



屈服准则和形变梯度

- ~~数学公式又回来了~~
- 定义左Cauchy Green张量$\mathbf{b}=\mathbf{F}\mathbf{F}^T$，那么考虑$\mathbf{F}=\mathbf{F}^E\mathbf{F}^P$，有

$$\mathbf{b}^E=\mathbf{F}^E\mathbf{F}^{E^T}=\mathbf{F}(\mathbf{F}^P)^{-1}(\mathbf{F}^P)^{-T}\mathbf{F}^T=\mathbf{F}\mathbf{C}^{P^{-1}}\mathbf{F}^T$$

- 所以弹性势能的左Cauchy Green张量可以这样演化：

$$\frac{D\mathbf{b}^E}{Dt}=\frac{D\mathbf{F}}{Dt}\mathbf{C}^{P^{-1}}\mathbf{F}^T+\mathbf{F}\mathbf{C}^{P^{-1}}\frac{D\mathbf{F}^T}{Dt}+\mathbf{F}\frac{D\mathbf{C}^{P^{-1}}}{Dt}\mathbf{F}^T$$

- 对它做分裂，变成两步求解过程。第一步：

$$\frac{D\mathbf{b}^{E0}}{Dt}=\frac{D\mathbf{F}}{Dt}\mathbf{C}^{P^{-1}}\mathbf{F}^T+\mathbf{F}\mathbf{C}^{P^{-1}}\frac{D\mathbf{F}^T}{Dt}$$

- 这个时候相当于我们假设变形没有塑性，那么$\mathbf{C}^{P^{-1}}=\mathbf{I}$，完全相当于演化弹性。
- 第二步：

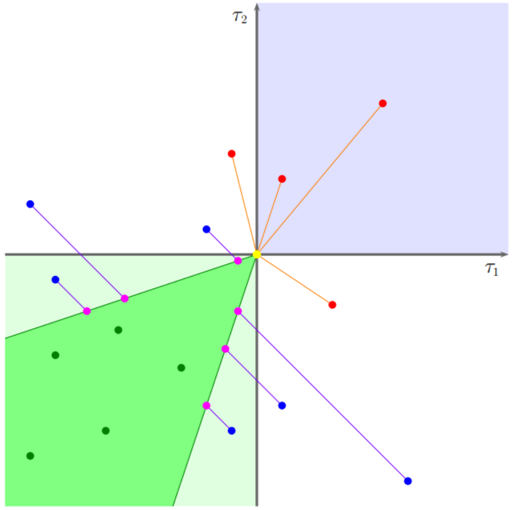
$$\frac{D\mathbf{b}^E}{Dt}=\mathbf{F}\frac{D\mathbf{C}^{P^{-1}}}{Dt}\mathbf{F}^T$$

屈服准则和形变梯度

- 这玩意一般称为Lie导数。它肯定是不能直接算的，所以我们直接假设：

$$\frac{D\mathbf{b}^E}{Dt} = -2\gamma \mathrm{dev}\left(\frac{\partial y}{\partial \boldsymbol{\tau}}\right)\mathbf{b}^E$$

- 这里的y就是屈服平面。看上去摸不着头脑，一层一层看：
  - $\left(\frac{\partial y}{\partial \boldsymbol{\tau}}\right)$可以理解成屈服面上的梯度函数，指向下降最快的方向
  - $\mathrm{dev}\left(\frac{\partial y}{\partial \boldsymbol{\tau}}\right)$把这个梯度函数去掉了体积部分，只留下形变部分
  - 所以它就相当于沿着形变方向走一段距离来尽可能靠近屈服面！
- 回顾：蓝色粒子的表现



$$\mathrm{dev}(\mathrm{A}) = \mathbf{A} - \frac{1}{d}\mathrm{tr}(\mathbf{A})\mathbf{I}$$

它相当于去掉体积变化部分
只考虑纯粹的形变

屈服准则和形变梯度

- 如何求解这个微分方程$\frac{D\mathbf{b}^E}{Dt} = -2\gamma\mathrm{dev}\left(\frac{\partial y}{\partial \boldsymbol{\tau}}\right)\mathbf{b}^E$?
- 直接建立隐式格式， $\mathbf{b}^{E0}$是假设全部弹性计算的结果

$$\mathbf{b}^{E,n+1} - \mathbf{b}^{E0} = -2\delta\gamma\mathrm{dev}\left(\frac{\partial y}{\partial \boldsymbol{\tau}}\right)\mathbf{b}^{E,n+1}$$

- 假设$\mathbf{s} = \mathrm{dev}(\boldsymbol{\tau})$，表示变形分量。那么，向屈服面投影应该不改变偏应力方向，即

$$\frac{\mathbf{s}^{n+1}}{\|\mathbf{s}^{n+1}\|} = \frac{\mathbf{s}^{n0}}{\|\mathbf{s}^{n0}\|}$$

- 而根据一系列推导，有

$$y = \frac{6-d}{2}\|\mathbf{s}^{n+1}\|^2(1+2\beta) + M^2(p^{n0} + \beta p_0)(p^{n0} - p_0) = 0$$

- 这个方程根据p^{n0}而确定解的类型，$p^{n0} = -\frac{1}{d}\mathrm{tr}(\tau) = -J^{E0}\Psi'(J^{E0})$

屈服准则和形变梯度

- 分成三类:
- Case 3，这个时候方程有解，直接解

$$\mathbf{s}^{n+1} = \frac{\mathbf{s}^{n0}}{\|\mathbf{s}^{n0}\|} M \sqrt{\frac{2(p^{n0} + \beta p_0)(p^{n0} - p_0)}{(d-6)(1+2\beta)}}$$

$$\mathbf{b}^{E,n+1} = \mathrm{dev}(\mathbf{b}^{E,n+1}) + \frac{1}{d}\mathrm{tr}(\mathbf{b}^{E,n+1})\mathbf{I} = \frac{\mathbf{s}^{n+1}}{\mu J^{2a}} + \mathrm{tr}(\mathbf{F}^{n0}\mathbf{F}^{n0^T})$$

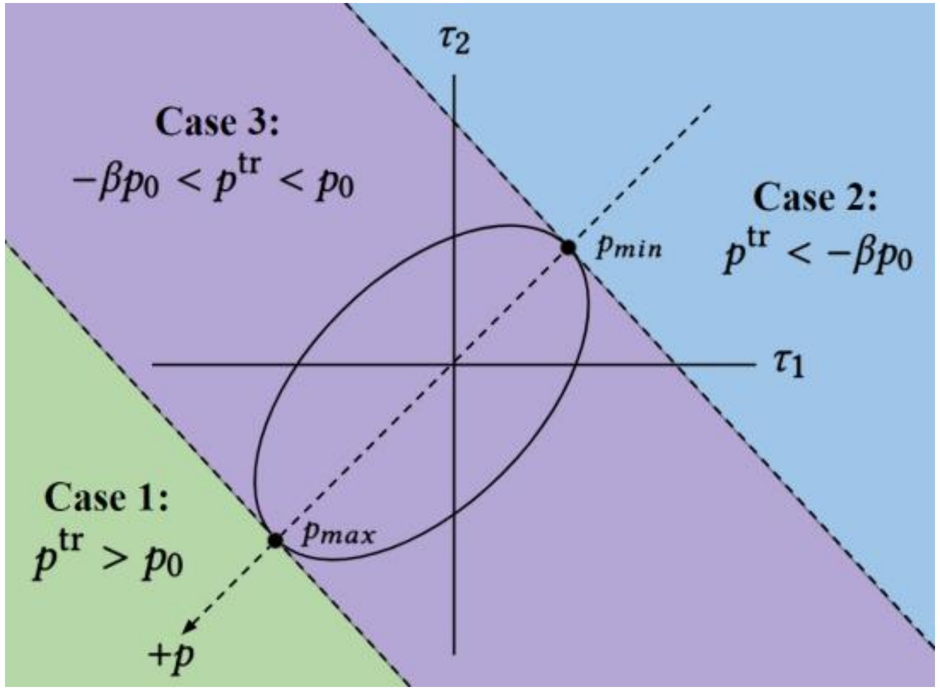
- Case 1，

$$p^{n+1} = p_{\max} = p_0, q = 0$$

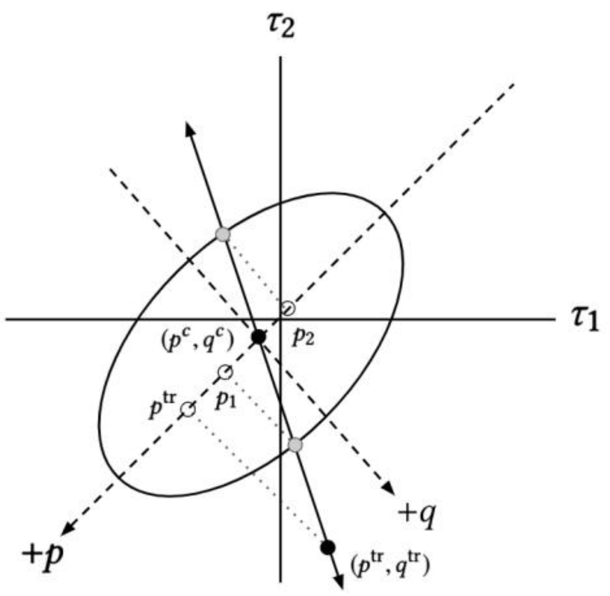
- Case 2，

$$p^{n+1} = p_{\min} = -\beta p_0, q = 0$$

- 然后反推特征值



硬化



- 硬化：随着塑性变化的发生，屈服面发生改变
- 为此，在屈服面中考虑硬化参数α：

$$p_0 = K\sinh(\xi \max(-a, 0))$$

- 然后每次迭代后更新α

$$\alpha^{n+1} = \log\left(\frac{J^{E,n0}}{J^{E,n+1}}\right) + \alpha^n$$

- 问题是在State3中，用dev更新不改变体积，导致α无法正确更新。所以重新计算与屈服面的交点作为正确的值做更新：
  - 全弹性时候计算出来点是(p^{tr}, q^{tr})
  - 椭圆的中心是(p^c, q^c)
  - 两点连线与椭圆方程形成一个二次型，交点$(p^\times, q^\times)$
  - 使用$p^\times$计算新的$J^\times$，然后用上述公式更新α^{n+1}

NACC Step

Algorithm 2 NACC Plasticity

Run MPM step until we have updated elastic deformation gradients, ${F_p}^{E,n+1}$
procedure ProjectStrainNACC
  $U,\ \Sigma,\ V^T = \text{SVD}({F_p}^{E,n+1})$
  $p_0 = \kappa\ (0.00001 + \xi\ \sinh(\max(\text{-}\alpha,0)))$ //buffer prevents YS from collapsing
  $J^{E,tr} = \Sigma_{0,0} * \Sigma_{1,1} * \Sigma_{2,2}$
  $\hat{\boldsymbol{s}}^{tr} = \mu {J^{E,tr}}^{\frac{-2}{d}}\ \text{dev}(\Sigma^2)$
  $\Psi^{\kappa\prime} = \frac{\kappa}{2}(J^{E,tr} - \frac{1}{J^{E,tr}})$
  $p^{tr} = -\Psi^{\kappa\prime} J^{E,tr}$
  if $p^{tr} > p_0$ **then** //Case 1: Project to max tip of YS
    $J^{E,n+1} = \sqrt{\frac{-2p_0}{\kappa} + 1}$
    $\Sigma^{n+1}{}_{i,i} = {J^{E,n+1}}^{\frac{1}{d}}$ for $i = 0,\ 1,\ 2$
    $\alpha \mathrel{+}= \log(\frac{J^{E,tr}}{J^{E,n+1}})$
    return $U * \Sigma^{n+1} * V^T$
  if $p^{tr} < -\beta p_0$ **then** //Case 2: Project to min tip of YS
    $J^{E,n+1} = \sqrt{\frac{2\beta p_0}{\kappa} + 1}$
    $\Sigma^{n+1}{}_{i,i} = {J^{E,n+1}}^{\frac{1}{d}}$ for $i = 0,\ 1,\ 2$
    $\alpha \mathrel{+}= \log(\frac{J^{E,tr}}{J^{E,n+1}})$
    return $U * \Sigma^{n+1} * V^T$
  $y = (1 + 2\beta)(\frac{6-d}{2})||\hat{\boldsymbol{s}}^{tr}|| + M^2(p^{tr} + \beta p_0)(p^{tr} - p_0)$
  if $y < 0.0001$ **then** //Inside YS
    return $U * \Sigma * V^T$
  if $p_0 > 0.0001$ and $p^{tr} < p_0 - 0.0001$ and $p^{tr} > -\beta p_0 + 0.0001$ **then**
    $p^c = (1-\beta)\frac{p_0}{2}$
    $q^{tr} = \sqrt{\frac{6-d}{2}}||\hat{\boldsymbol{s}}^{tr}||$
    $direction[0] = p^c - p^{tr}$
    $direction[1] = 0 - q^{tr}$
    $direction = \frac{direction}{||direction||}$
    $C = M^2(p^c + \beta p_0)(p^c - p_0)$
    $B = M^2 direction[0](2p^c - p_0 + \beta p_0)$
    $A = M^2 direction[0]^2 + (1 + 2\beta) direction[1]^2$
    $l_1 = \frac{-B + \sqrt{B^2 - 4AC}}{2A}$
    $l_2 = \frac{-B - \sqrt{B^2 - 4AC}}{2A}$
    $p_1 = p^c + l_1 direction[0]$
    $p_2 = p^c + l_2 direction[0]$
    $p^{\times} = (p^{tr} - p^c)(p_1 - p^c)\ ?\ p_1 : p_2$
    $J^{E,\times} = \sqrt{\frac{-2p^{\times}}{\kappa} + 1}$
    if $J^{E,\times} > 0.0001$ **then**
      $\alpha \mathrel{+}= \log(\frac{J^{E,tr}}{J^{E,\times}})$
  //Case 3: Yield Surface Projection
  $\hat{\boldsymbol{b}}^{E,n+1} = \sqrt{\frac{-M^2(p^{tr}+\beta p_0)(p^{tr}-p_0)}{(1+2\beta)(\frac{6-d}{2})}}(\frac{{J^{E,tr}}^{\frac{2}{d}}}{\mu})\frac{\hat{\boldsymbol{s}}^{tr}}{||\hat{\boldsymbol{s}}^{tr}||} + \frac{1}{d}\text{trace}(\Sigma^2)$
  $\Sigma^{n+1}{}_{i,i} = \sqrt{\hat{\boldsymbol{b}}_i^{E,n+1}}$ for $i = 0,\ 1,\ 2$
  return $U * \Sigma^{n+1} * V^T$

Section 4.

MLS-MPM

Review: Semi-Lagrange Solver

- 动机：将NS方程拆开分步计算

Any Solver, e.g. RK4

$$\frac{\mathrm{d}q}{\mathrm{d}t} = f(q) + g(q) \xrightarrow[\text{Split}]{\text{Forward Eular}} \begin{cases} \tilde{q}^{(n)} = q^{(n)} + \Delta t f(q^{(n)}) \\ q^{(n+1)} = \tilde{q}^{(n)} + \Delta t\, g(\tilde{q}^{(n)}) \end{cases} \xrightarrow{\text{Generalize}} \begin{cases} \tilde{q}^{(n)} = \boxed{F(\Delta t, q^{(n)})} \\ q^{(n+1)} = \boxed{G(\Delta t, \tilde{q}^{(n)})} \end{cases}$$

$$\frac{D\boldsymbol{u}}{Dt} = -\frac{1}{\rho}\nabla p + \nu\nabla^2\boldsymbol{u} + \boldsymbol{g} - \delta_{\partial\Omega}\sigma\kappa\hat{\boldsymbol{n}} \xrightarrow[\text{Split}]{} \begin{cases} \frac{D\boldsymbol{u}}{Dt} = 0 & \text{Advection} \\ \frac{\partial \boldsymbol{u}}{\partial t} = \boldsymbol{g} & \text{Forward Euler} \\ \boxed{\frac{\partial \boldsymbol{u}}{\partial t} + \frac{1}{\rho}\nabla p = 0 \quad \text{Projection}} \end{cases}$$

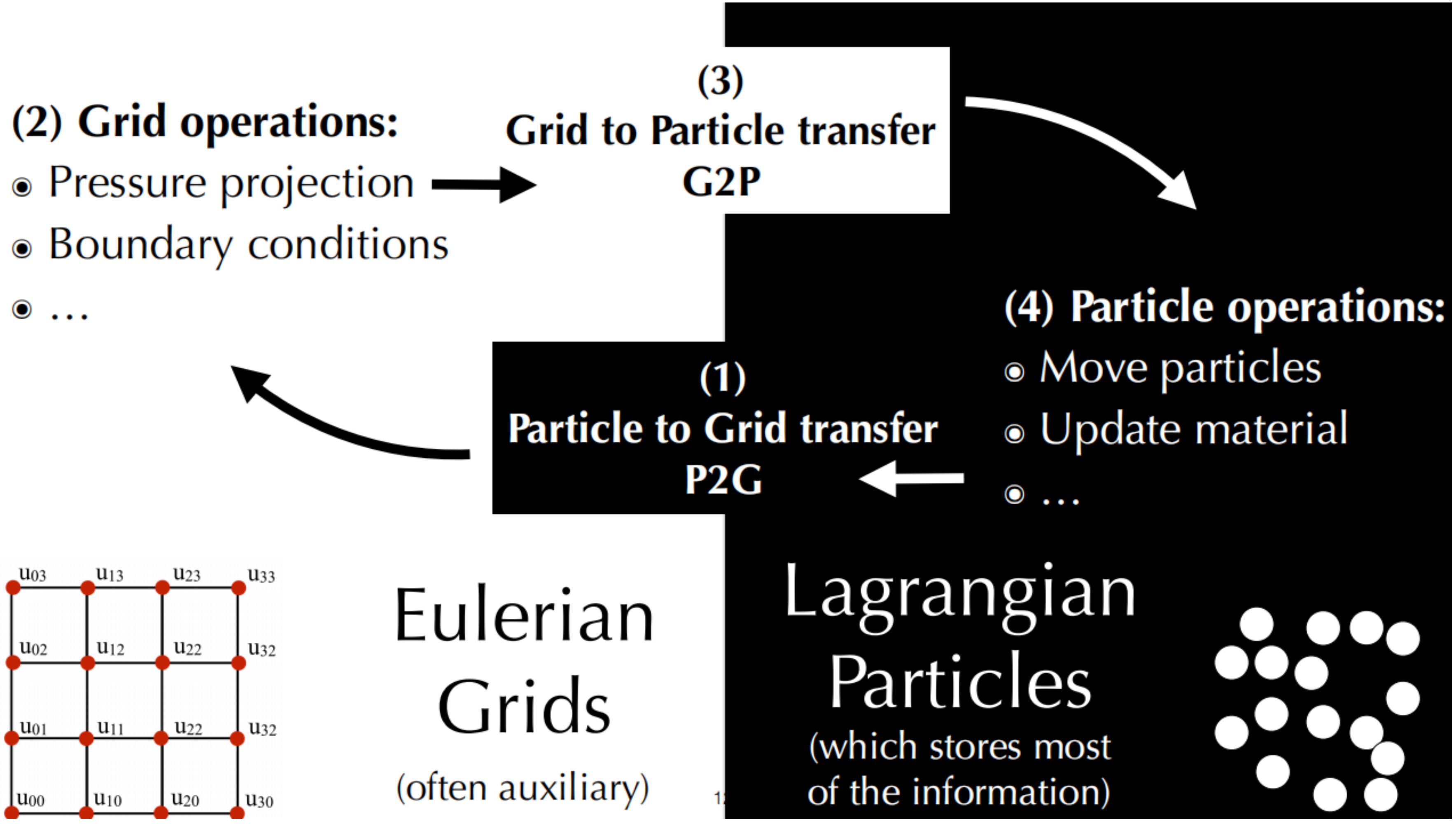
MPM是弱可压
不需要压力投影!

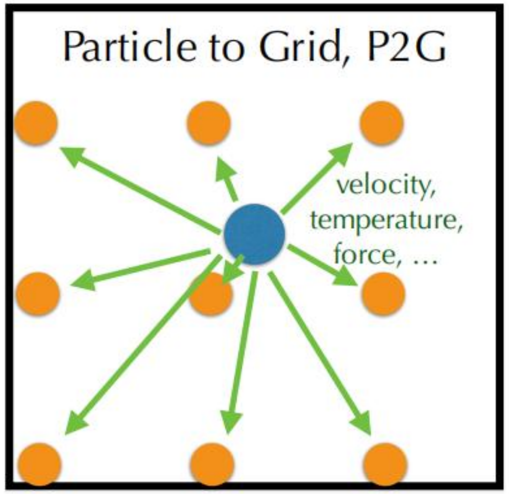
算法概要：

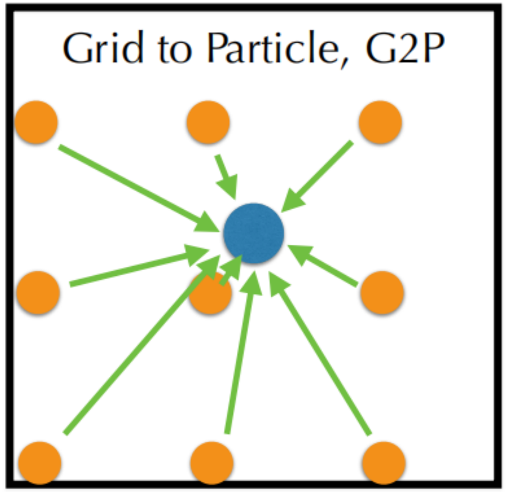
1. initialize $\boldsymbol{u}$, such that $\nabla \cdot \boldsymbol{u} = 0$
2. for time step $n = 0,1,2,\ldots$
   - 2.1 $t_{n+1} = t_n + \Delta t$
   - 2.2 $\boldsymbol{u}^A = \mathrm{advect}\left(\boldsymbol{u}^{(n)}, \Delta t\right)$
   - 2.3 $\boldsymbol{u}^B = \boldsymbol{u}^A + \Delta t \boldsymbol{g}$
   - 2.4 $\boldsymbol{u}^{(n+1)} = \mathrm{project}(\Delta t, \boldsymbol{u}^B)$

From Semi-Langrage to PIC

- 拉格朗日视角适合求解粒子状态传递，欧拉视角来做力计算，为什么不结合?







From Semi-Langrage to PIC

- P2G：分成三个方向分别影响

$$N_{\mathbf{i}}(\mathbf{x}_p) = N\left(\frac{x_p - x_i}{h}\right) N\left(\frac{y_p - y_i}{h}\right) N\left(\frac{z_p - z_i}{h}\right)$$

Linear

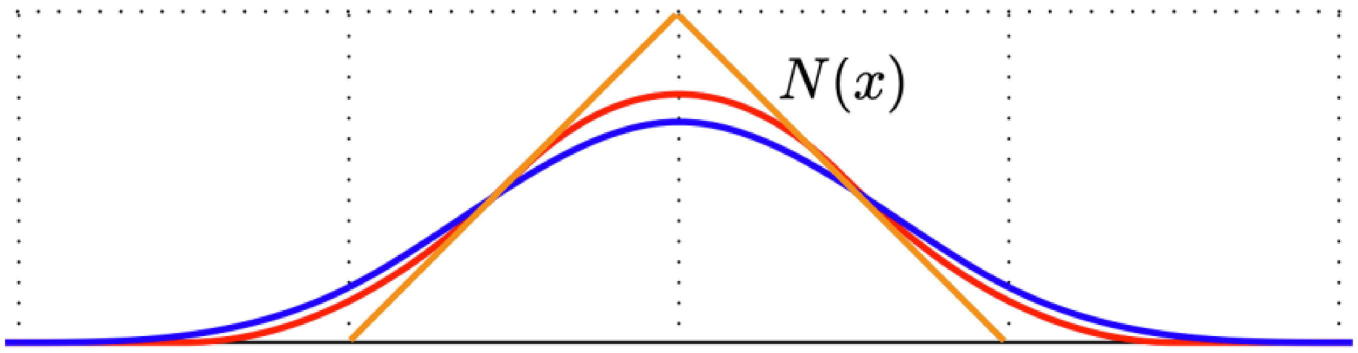
$$N(x) = \begin{cases} 1 - |x| & 0 \leqslant |x| < 1 \\ 0 & 1 \leqslant |x| \end{cases}$$

Quadratic

$$N(x) = \begin{cases} \frac{3}{4} - |x|^2 & 0 \leqslant |x| < \frac{1}{2} \\ \frac{1}{2}\left(\frac{3}{2} - |x|\right)^2 & \frac{1}{2} \leqslant |x| < \frac{3}{2} \\ 0 & \frac{3}{2} \leqslant |x| \end{cases}$$

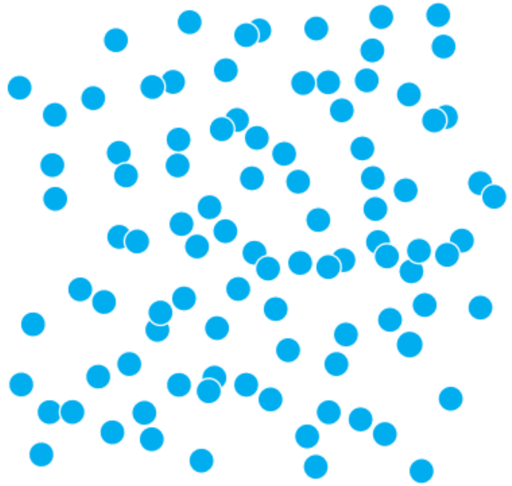
Cubic

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



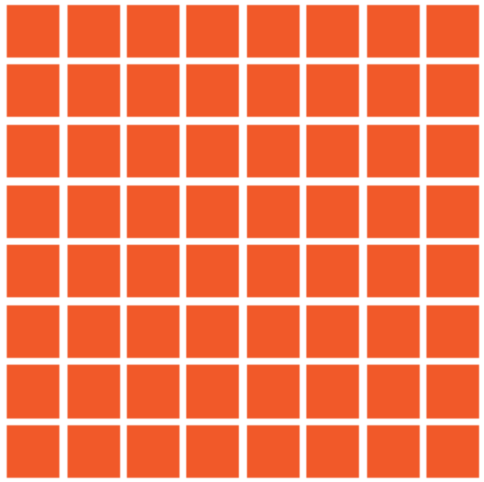
Frame

The Material Point Method as of 2018



Particle to Grid (P2G)

Grid to Particle (G2P)



Transfer (Particle-in-Cell, PIC)
Affine PIC, APIC [Jiang et al. 2016]
Polynomial PIC, PolyPIC [Fu et al. 2017]
High-performance GIMP [Gao et al. 2017]
Moving Least Squares [Hu et al. 2018]
Compatible PIC [Hu et al. 2018]
...

Particles (Constitutive models)
Snow [Stomakhin et al. 2013],
Foam [Ram et al. 2015, Yue et al. 2015]
Sand [Klar et al. 2015, Pradhana et al 2017]

Grid
SPGrid [Setaluri et al. 2014],
OpenVDB [Museth 2013]
Multiple Grids [Pradhana et al. 2017]

34

Transfer

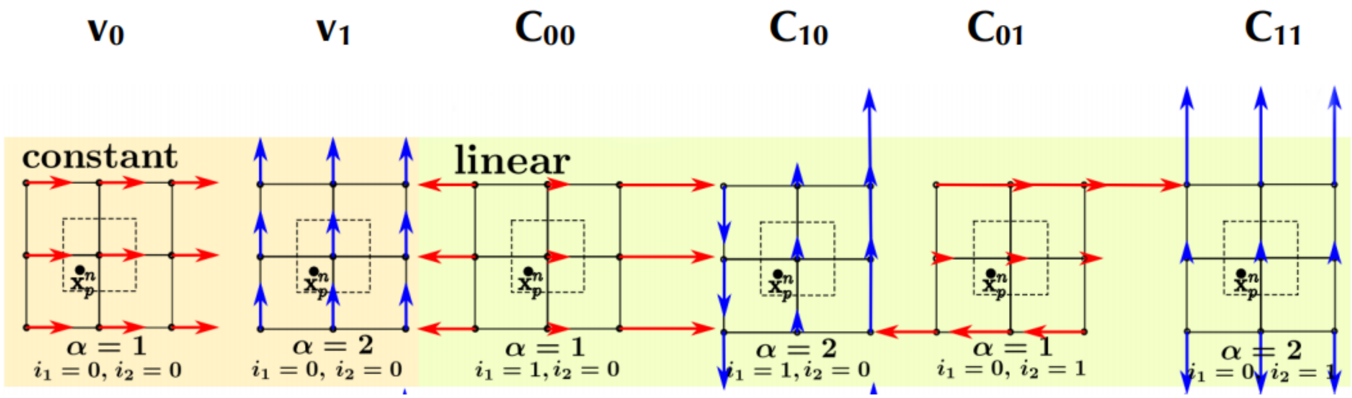
- 简单的传递方式:
- P2G传递过程

$$m_{\mathbf{i}} = \sum_{\mathbf{p}} m_{\mathbf{p}} N_{\mathbf{i}}(\mathbf{x}_{\mathbf{p}}), (m\mathbf{v})_{\mathbf{i}} = \sum_{\mathbf{p}} m_{\mathbf{p}} \mathbf{v}_{\mathbf{p}} N_{\mathbf{i}}(\mathbf{x}_{\mathbf{p}}), \mathbf{v}_{\mathbf{i}} = \frac{(m\mathbf{v})_{\mathbf{i}}}{m_{\mathbf{i}}}$$

- 反过来，G2P传递过程

$$v_{\mathbf{p}} = \sum_{\mathbf{i}} m_{\mathbf{i}} N_{\mathbf{p}}(\mathbf{x}_{\mathbf{i}})$$

- 问题：会丢失自由度，假如3x3传递，这样的网格有18个自由度，而生成的粒子只有一个=>APIC



Transfer

- APIC Transfer：额外引入一个仿射变换矩阵做动量传递

$$m_{\mathbf{i}}\mathbf{v}_{\mathbf{i}} = \sum_p w_{\mathbf{i}p} m_p \left(\mathbf{v}_p + \mathbf{B}_p \left(\mathbf{D}_p\right)^{-1} \left(\mathbf{x}_i - \mathbf{x}_p\right)\right)$$

- 其中，

$$\mathbf{D}_p = \sum_{\mathbf{i}} w_{\mathbf{i}p} \left(\mathbf{x}_i - \mathbf{x}_p\right)\left(\mathbf{x}_{\mathbf{i}} - \mathbf{x}_p\right)^T$$

- 对于二次插值，$\mathbf{D}_p = \frac{1}{4}\Delta x^2 \mathbf{I}$；对于三次插值，$\mathbf{D}_p = \frac{1}{3}\Delta x^2 \mathbf{I}$。
- 而$\mathbf{B}_p$是粒子一个特征，可以在G2P的时候顺便更新：

$$\mathbf{B}_p = \sum_{\mathbf{i}} w_{\mathbf{i}p} \mathbf{v}_{\mathbf{i}} \left(\mathbf{x}_{\mathbf{i}} - \mathbf{x}_p\right)^T$$

Transfer

- taichi Code
- easy to impl

```python
for p in x:
    base = (x[p] * inv_dx - 0.5).cast(int)
    fx = x[p] * inv_dx - base.cast(float)
    # Quadratic B-spline
    w = [0.5 * (1.5 - fx) ** 2, 0.75 - (fx - 1) ** 2, 0.5 * (fx - 0.5) ** 2]
    affine = C[p]
    for i in ti.static(range(3)):
        for j in ti.static(range(3)):
            offset = ti.Vector([i, j])
            dpos = (offset.cast(float) - fx) * dx
            weight = w[i][0] * w[j][1]
            grid_v[base + offset] += weight * (v[p] + affine @ dpos)
            grid_m[base + offset] += weight
```

```python
for p in x:
    base = (x[p] * inv_dx - 0.5).cast(int)
    fx = x[p] * inv_dx - base.cast(float)
    # Quadratic B-spline
    w = [
        0.5 * (1.5 - fx) ** 2, 0.75 - (fx - 1.0) ** 2, 0.5 * (fx - 0.5) ** 2
    ]
    new_v = ti.Vector.zero(ti.f32, 2)
    new_C = ti.Matrix.zero(ti.f32, 2, 2)
    for i in ti.static(range(3)):
        for j in ti.static(range(3)):
            dpos = ti.Vector([i, j]).cast(float) - fx
            g_v = grid_v[base + ti.Vector([i, j])]
            weight = w[i][0] * w[j][1]
            new_v += weight * g_v
            new_C += 4 * weight * g_v.outer_product(dpos) * inv_dx

    x[p] = clamp_pos(x[p] + new_v * dt)
    v[p] = new_v
    C[p] = new_C
```

Force Update

- Appendix D&E&F
- 直接写结论：

$$\mathbf{F}_p^{n+1} = \left(\mathbf{I} + \Delta t \mathbf{B}_p^n \left(\mathbf{D}_p\right)^{-1}\right) \mathbf{F}_p^n$$

- 使用塑性模型对$\mathbf{F}_p^{n+1}$进行投影，得到$\bar{\mathbf{F}}_p^{n+1}$
- 计算力

$$\mathbf{f} = -\frac{4}{\Delta x^2} V_p^0 \mathbf{P}\left(\mathbf{F}_p\right) \mathbf{F}_p^T w_{\mathbf{i}p} (x_\mathbf{i} - x_p)$$

- 计算动量

$$(m\mathbf{v})_\mathbf{i} = \sum_p w_{\mathbf{i}p} (m_p \mathbf{v}_p + m_p \mathbf{B}_p^n \left(\mathbf{D}_p\right)^{-1} \left(\mathbf{x}_\mathbf{i} - \mathbf{x}_p\right) + \mathbf{f}\Delta t)$$

- 计算速度：$\mathbf{v}_\mathbf{i} = \frac{(m\mathbf{v})_\mathbf{i}}{m_\mathbf{i}}$

Frame – Explicit MPM

1. Particle to grid (P2G)
   - $\mathbf{F}_p^{n+1} = (\mathbf{I} + \Delta t \mathbf{C}_p^n)\mathbf{F}_p^n, \ldots$ **(Deformation update)**
   - $(m\mathbf{v})_i^{n+1} = \sum_p w_{ip}\{m_p \mathbf{v}_p^n + [m_p \mathbf{C}_p^n - \frac{4\Delta t}{\Delta x^2}\sum_p V_p^0 \mathbf{P}(\mathbf{F}_p^{n+1})(\mathbf{F}_p^{n+1})^T](\mathbf{x}_i - \mathbf{x}_p^n)\}$ **(Grid momentum)**
   - $m_i^{n+1} = \sum_p m_p w_{ip}$ (Grid mass)
2. Grid operations
   - $\hat{\mathbf{v}}_i^{n+1} = (m\mathbf{v})_i^{n+1} / m_i^{n+1}$ (Grid velocity)
   - $\mathbf{v}_i^{n+1} = \mathrm{BC}(\hat{\mathbf{v}}_i^{n+1})$ (Grid boundary condition. BC is the boundary condition operator.)
3. Grid to particle (G2P)
   - $\mathbf{v}_p^{n+1} = \sum_i w_{ip} \mathbf{v}_i^{n+1}$ (Particle velocity)
   - $\mathbf{C}_p^{n+1} = \frac{4}{\Delta x^2}\sum_i w_{ip} \mathbf{v}_i^{n+1}(\mathbf{x}_i - \mathbf{x}_p^n)^T$ (Particle velocity gradient)
   - $\mathbf{x}_p^{n+1} = \mathbf{x}_p^n + \Delta t \mathbf{v}_p^{n+1}$ (Particle position)

Force Update - impl

- MPM99 – P2G　简化塑性模型

```
for p in x:  # Particle state update and scatter to grid (P2G)
    base = (x[p] * inv_dx - 0.5).cast(int)
    fx = x[p] * inv_dx - base.cast(float)
    # Quadratic kernels  [http://mpm.graphics   Eqn. 123, with x=fx, fx-1,fx-2]
    w = [0.5 * (1.5 - fx) ** 2, 0.75 - (fx - 1) ** 2, 0.5 * (fx - 0.5) ** 2]
    # F[p]: deformation gradient update
    F[p] = (ti.Matrix.identity(float, 2) + dt * C[p]) @ F[p]
    # h: Hardening coefficient: snow gets harder when compressed
    h = ti.exp(10 * (1.0 - Jp[p]))
    if material[p] == 1:  # jelly, make it softer
        h = 0.3
    mu, la = mu_0 * h, lambda_0 * h
    if material[p] == 0:  # liquid
        mu = 0.0
    U, sig, V = ti.svd(F[p])
    # Avoid zero eigenvalues because of numerical errors
    for d in ti.static(range(2)):
        sig[d, d] = ti.max(sig[d, d], 1e-6)
    J = 1.0
    for d in ti.static(range(2)):
        new_sig = sig[d, d]
        if material[p] == 2:  # Snow
            new_sig = ti.min(ti.max(sig[d, d], 1 - 2.5e-2), 1 + 4.5e-3)  #
            Plasticity
        Jp[p] *= sig[d, d] / new_sig
        sig[d, d] = new_sig
        J *= new_sig
```

```
if material[p] == 0:
    # Reset deformation gradient to avoid numerical instability
    F[p] = ti.Matrix.identity(float, 2) * ti.sqrt(J)
elif material[p] == 2:
    # Reconstruct elastic deformation gradient after plasticity
    F[p] = U @ sig @ V.transpose()
stress = 2 * mu * (F[p] - U @ V.transpose()) @ F[p].transpose() + ti.Matrix.
identity(float, 2) * la * J * (J - 1)
stress = (-dt * p_vol * 4 * inv_dx * inv_dx) * stress
affine = stress + p_mass * C[p]
# Loop over 3x3 grid node neighborhood
for i, j in ti.static(ti.ndrange(3, 3)):
    offset = ti.Vector([i, j])
    dpos = (offset.cast(float) - fx) * dx
    weight = w[i][0] * w[j][1]
    grid_v[base + offset] += weight * (p_mass * v[p] + affine @ dpos)
    grid_m[base + offset] += weight * p_mass
```

Force Update - impl

- Boundary & external force

```
for i, j in grid_m:
    if grid_m[i, j] > 0:  # No need for epsilon here
        grid_v[i, j] = (1 / grid_m[i, j]) * grid_v[i, j]
        grid_v[i, j][1] -= dt * 50  # gravity
        if i < 3 and grid_v[i, j][0] < 0:
            grid_v[i, j][0] = 0  # Boundary conditions
        if i > n_grid - 3 and grid_v[i, j][0] > 0:
            grid_v[i, j][0] = 0
        if j < 3 and grid_v[i, j][1] < 0:
            grid_v[i, j][1] = 0
        if j > n_grid - 3 and grid_v[i, j][1] > 0:
            grid_v[i, j][1] = 0
```

- G2P

```
for p in x:  # grid to particle (G2P)
    base = (x[p] * inv_dx - 0.5).cast(int)
    fx = x[p] * inv_dx - base.cast(float)
    w = [0.5 * (1.5 - fx) ** 2, 0.75 - (fx - 1.0) ** 2, 0.5 * (fx - 0.5) ** 2]
    new_v = ti.Vector.zero(float, 2)
    new_C = ti.Matrix.zero(float, 2, 2)
    for i, j in ti.static(ti.ndrange(3, 3)):
        # loop over 3x3 grid node neighborhood
        dpos = ti.Vector([i, j]).cast(float) - fx
        g_v = grid_v[base + ti.Vector([i, j])]
        weight = w[i][0] * w[j][1]
        new_v += weight * g_v
        new_C += 4 * inv_dx * weight * g_v.outer_product(dpos)
    v[p], C[p] = new_v, new_C
    x[p] += dt * v[p]  # advection
```

- 演示

Section 5.

Fracture

Phase Field

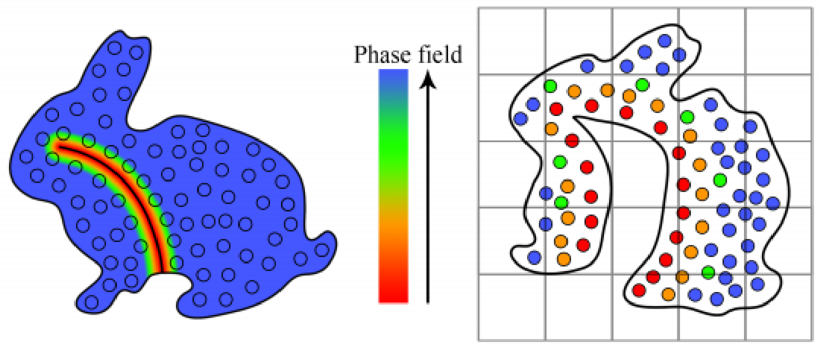
- 从能量说起：假设裂缝区域是Γ，那么总体能量是

$$E(\mathbf{F},\Gamma)=\int_{\Omega^0}\Psi(\mathbf{F})\mathrm{d}\mathbf{X}+\int_{\Gamma}G\mathrm{d}\mathbf{X}$$

- 这里的G表示单位区域断裂时释放的能量。理论上，这个方程的最小化可以用来预测断裂的演化趋势，但后面那个很难直接求解。
- 基本思路：将右面的积分转换到整个区域上，引入一个$c(\mathbf{X},t)$表示断裂程度：0表示完全断裂，1表示完全弹性，那么

$$\int_{\Gamma}G\mathrm{d}\mathbf{X}=\int_{\Omega^0}\left(\frac{(c-1)^2}{4l_0}+l_0|\nabla c|^2\right)G\mathrm{d}\mathbf{X}$$

  - c=0，能量变大；c=1，能量变小；
  - ∇c惩罚变化趋势，尽可能保证变化平缓
  - 在MPM中，取$l_0=0.5\Delta x$



Phase Field (Cont.)

- 既然有了这个定义，那么弹性能量也需要加权重来促进材料沿裂纹分离。假如超弹性可以分解成拉伸项Ψ^+和压缩项Ψ^-：

$$\Psi(\mathbf{F}) = g(c)\Psi^+(\mathbf{F}) + \Psi^-(\mathbf{F})$$

- 其中，$g(c) = (1-\varepsilon)c^2 + \varepsilon$， ε用来防止浮点误差。最终，综合上面的定义，得到带有断裂特性的材料本构：

$$\Psi(\mathbf{F}, c) = g(c)\Psi^+(\mathbf{F}) + \Psi^-(\mathbf{F}) + \left(\frac{(c-1)^2}{4l_0} + l_0|\nabla c|^2\right)G$$

- 也就是比起传统MPM，只需要用这个来维护$\mathbf{P}$的计算即可。现在还有一个问题：如何定义这里的超弹性模型？分别用剪切和体积项定义，详见Appendix C.

$$\hat{\Psi} = \Psi^\mu(\mathbf{F}) + \Psi^\kappa(J), \Psi^\mu(\mathbf{F}) = \frac{\mu}{2}(\mathrm{tr}(\mathbf{F}^T\mathbf{F}) - d), \Psi^\kappa(J) = \frac{\kappa}{2}\left(\frac{J^2-1}{2} - \log J\right)$$

$$\Psi^+ = \begin{cases} \Psi^\mu(J^a\mathbf{F}) + \Psi^\kappa(\mathbf{F}), J \geq 1 \\ \Psi^\mu(J^a\mathbf{F}), J < 1 \end{cases}, \Psi^- = \begin{cases} 0, J \geq 1 \\ \Psi^\kappa(\mathbf{F}), J < 1 \end{cases}, a = -\frac{1}{d}$$

Phase Evolution

- 如何更新下一个时间步的断裂状态c^{n+1}?
- 一种简单的作法：对于每个质点分析，如果应力超过某个阈值则认为断裂

$$c^{n+1} = \min\left(c^n, 1 - H_s\left(1 - \frac{\sigma_m}{\sigma_f}\right)\right)$$

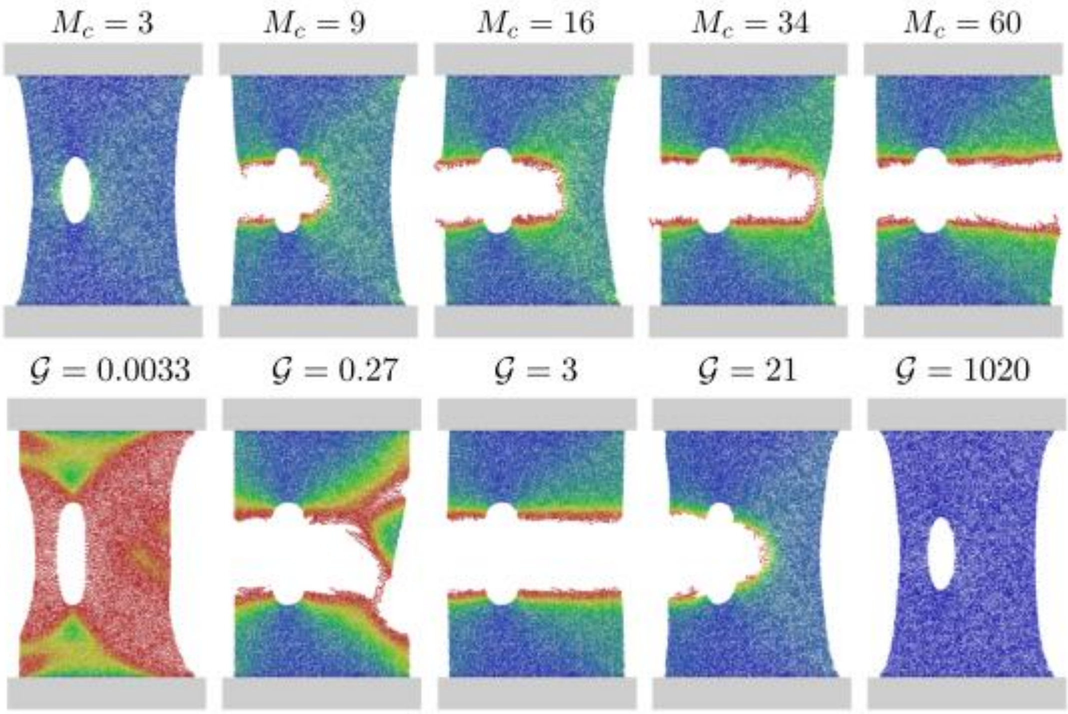
- σ_f是临界点，H_s是一个常数。应力越大就越容易断裂
- 问题：只有局部特征，没有全局特征
- 解决方案：建立有关断裂的变分，使用Euler-Lagarangian方程得到

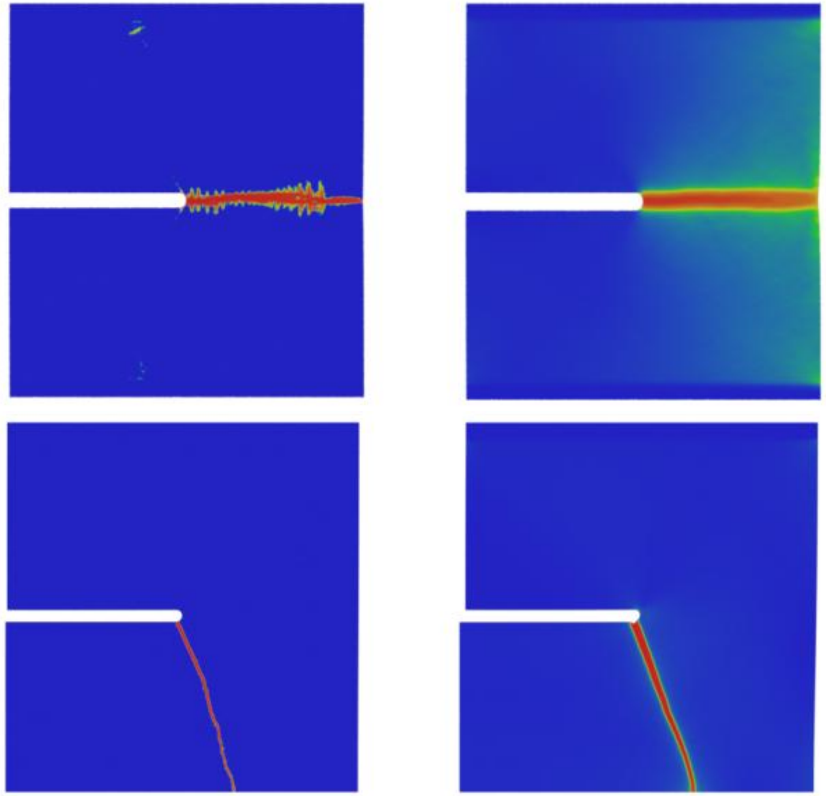
$$\left(\frac{4l_0M_c(1-r)\Psi^+}{G} + M_c + \frac{1}{\Delta t}\right)c^{n+1} - (4l_0^2M_c)\nabla^2c^{n+1} = M_c + \frac{c^n}{t}$$

- 其中$M_c = \frac{G}{2R_cl_0}$。这样就可以迭代求解。

Phase Evolution(Cont.)

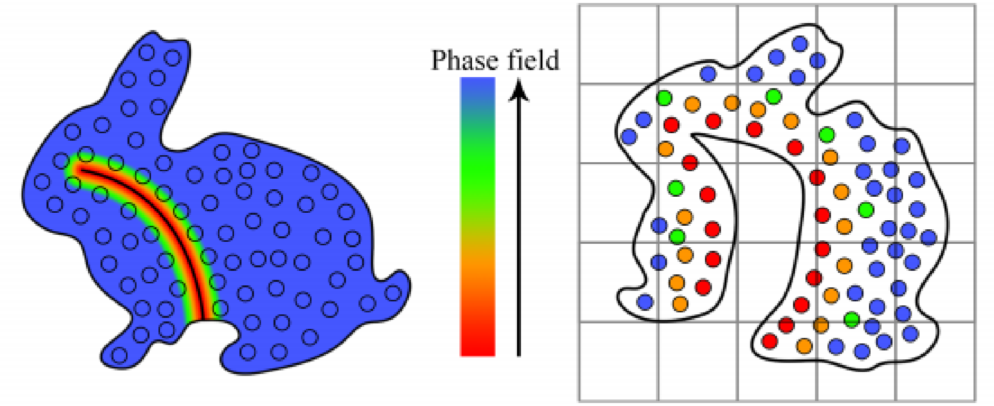
- 参数分析：M_c越大，越容易断裂；G影响能量扩散特性
- 对比分析：更平滑、在低分辨率网格下也可以做的比较好





Phase Field Integration

- 怎么演化？用粒子来携带c属性进行正常的MPM操作，类似的PhaseP2G和PhaseG2P



- 使用一阶精度来gather和scatter：
  - P2G：$c_{\mathbf{i}}^{n} = w_{\mathbf{i}p} c_p^n$
  - G2P：$c_p^{n+1} = \max\left(0, \min\left(c_p^n, c_p^n + \sum_{\mathbf{i}}\left(c_{\mathbf{i}}^{n+1} - c_{\mathbf{i}}^{n}\right) w_{ip}\right)\right)$ 【FLIP transfer】
- 最后的问题：在MPM框架下如何计算c_p^{n+1}?

Phase Field Integration(Cont.)

- 不加证明的给出结论（啃不动了）：

$$(\mathbf{M}+\mathbf{H})\mathbf{c}=\mathbf{r}$$

- 建立一个线性系统，将i,j展平为一个统一的下标

$$M_{ij}=\delta_{ij}\sum_p V_p^n\left(\frac{4l_0M_c(1-r)\Psi^+}{G}+M_c+\frac{1}{\Delta t}\right)w_{ip}$$

$$H_{ij}=\sum_p V_p^n 4l_0M_c\left(M_p^{-1}w_{ip}^n\left(x_{\alpha i}-x_{\alpha p}^n\right)\right)^T\left(M_p^{-1}w_{jp}^n\left(x_{\alpha j}-x_{\alpha p}^n\right)\right)$$

$$r_i=\sum_p V_p^n\left(M_c+\frac{c_p^n}{\Delta t}\right)w_{ip}^n$$

- 用Jacobi作为Preconditioner的共轭梯度求解即可

PFF-MPM Step

Algorithm 1 PFF-MPM Step

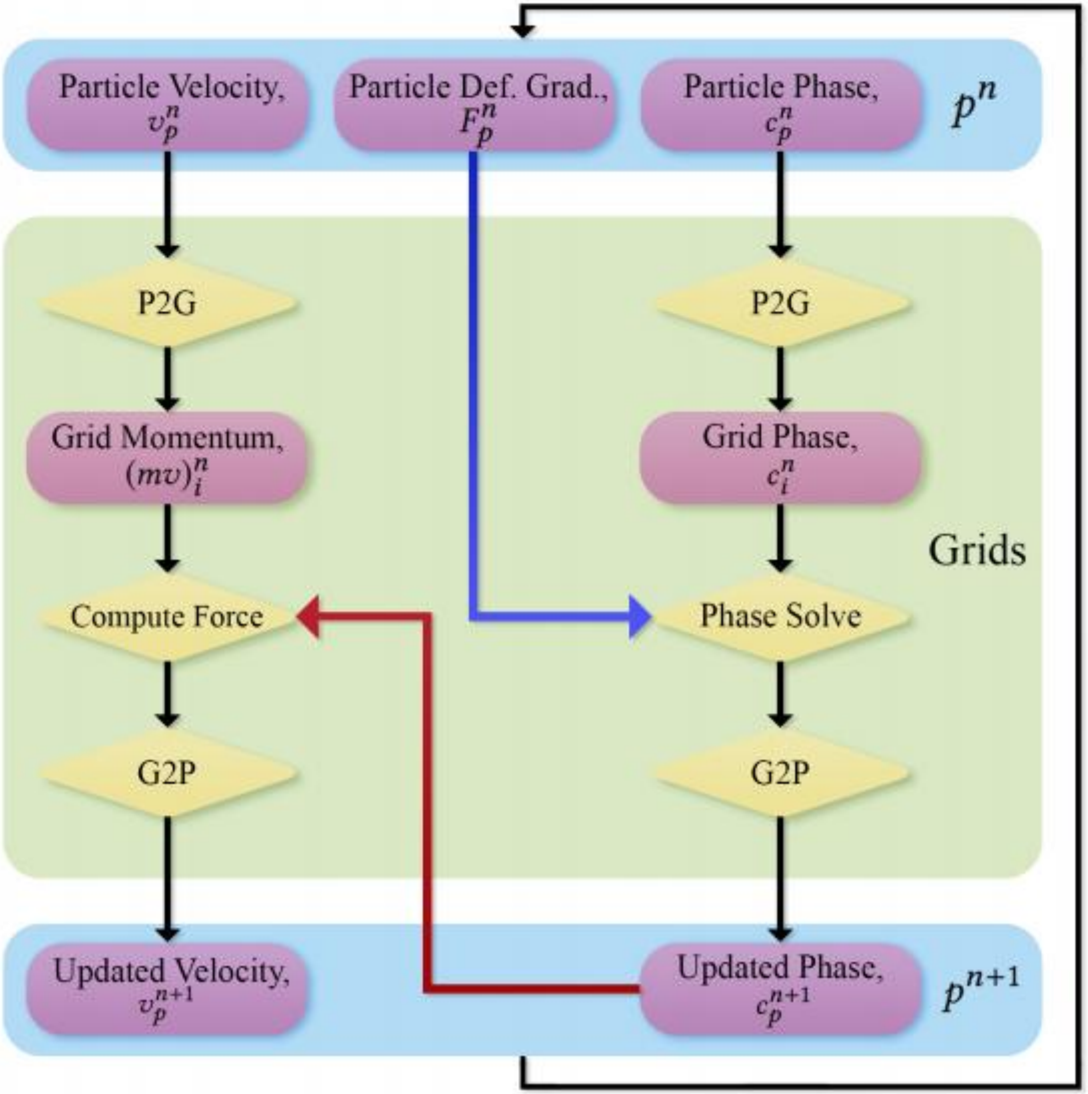
procedure PhaseP2G
    Compute interpolation weights, w_{ip}^n //We use quadratic B-spline
    for each grid node, i **do**
        $c_i^n = \frac{\sum_p w_{ip}^n c_p^n}{\sum_p w_{ip}^n}$
procedure PhaseSolve
    //Goal is to contruct and solve this system for $\mathbf{c}$: $(\mathcal{M} + \boldsymbol{H})\mathbf{c} = \boldsymbol{r}$
    $\boldsymbol{r} = [r_i] = \sum_p V_p^n (M_c + \frac{c_p^n}{\Delta t}) w_{ip}^n$ //Build rhs
    $\boldsymbol{H} = [\boldsymbol{H}_{ij}] = \sum_p V_p^n (4 l_0^2 M_c)(\nabla \Theta_i(\boldsymbol{x}_p^n))^T (\nabla \Theta_j(\boldsymbol{x}_p^n))$ //Build MPM discrete Laplace operator
    $\mathcal{M} = [\mathcal{M}_{ii}] = \sum_p V_p^n \left(\frac{4 l_0 M_c (1-r) \Psi_p^{\mathcal{H}}}{\mathcal{G}} + M_c + \frac{1}{\Delta t} \right) w_{ip}^n$ //Build diagonal lumped mass matrix
    Solve the system with PCG (Jacobi preconditioner takes around 4 iters)
procedure PhaseG2P
    for each particle, p **do**
        $c_p^{n+1} = \max(0, \min(c_p^n, c_p^n + \sum_i (c_i^{n+1} - c_i^n) w_{ip}^n))$ //Prevent material healing and keep $c \in [0, 1]$
Run traditional MPM step as usual until computeForce
procedure computeForce //Key difference is this incorporates c_p^{n+1}
    if symplectic **then**
        $\boldsymbol{f}_i^n = -\sum_p V_p^0 w_{ip}^n M_p^{-1} \frac{\partial \Psi}{\partial \boldsymbol{F}} (\boldsymbol{F}_p^n, c_p^{n+1}) \boldsymbol{F}_p^{nT} (\boldsymbol{x}_i^n - \boldsymbol{x}_p^n)$
    else if implicit **then**
        $\boldsymbol{f}_i^{n+1} = -\sum_p V_p^0 w_{ip}^n M_p^{-1} \frac{\partial \Psi}{\partial \boldsymbol{F}} (\boldsymbol{F}_p^{n+1}, c_p^{n+1}) \boldsymbol{F}_p^{nT} (\boldsymbol{x}_i^n - \boldsymbol{x}_p^n)$
Finish MPM step like usual (with or without plasticity return mapping)

Section 6.

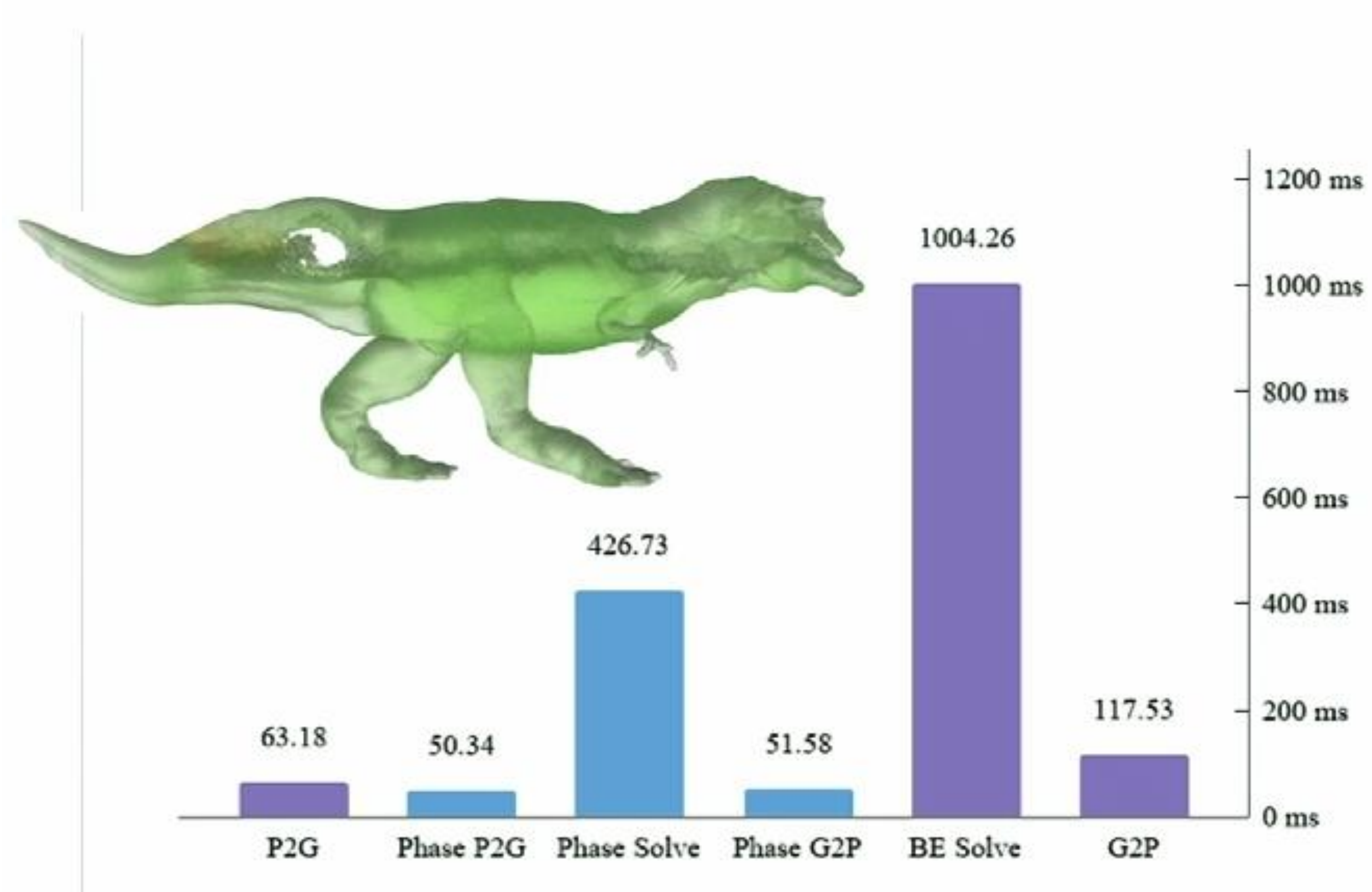
Result

Algorithm in a Nutshell

- 将上述过程嵌入到MPM总过程里有点麻烦，有没有什么好办法?
- 交错求解形变和断裂，一直迭代到收敛即可，一般1-2次就够。



Efficiency



剩下的见视频，来不及截了

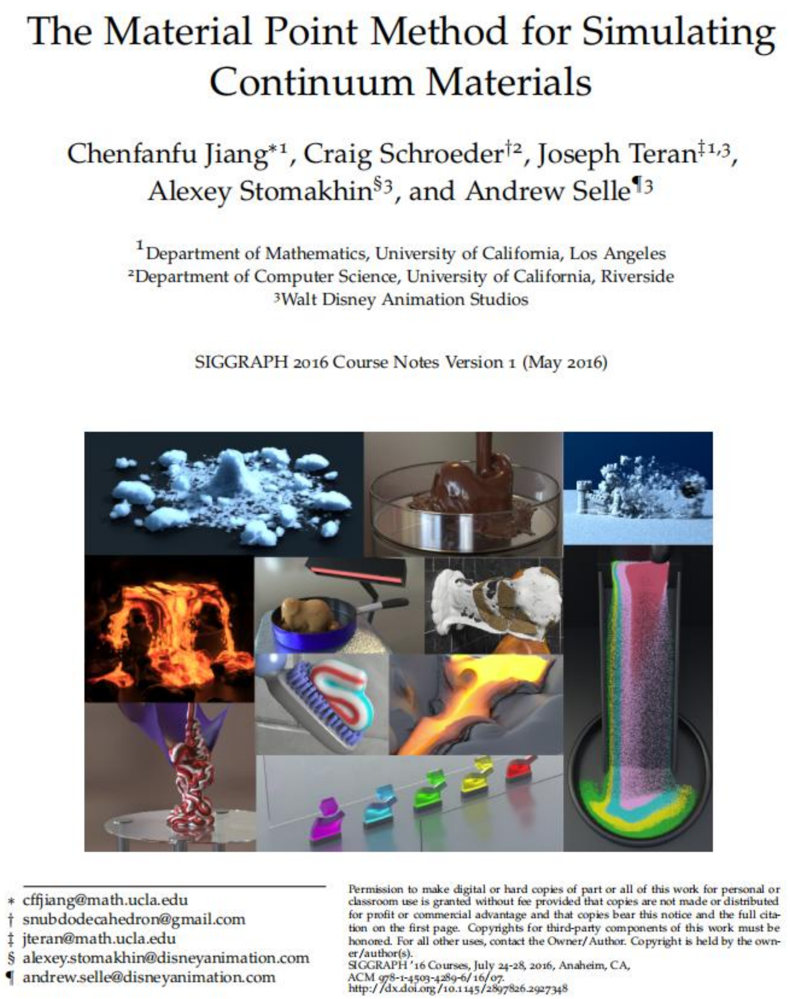
Future Works

- 各项异性材质
- 三角网格拉格朗日力
- 流固耦合、IPC结合

Further Materials

- MPM Course Note

- Strongly Recommend



The Material Point Method for Simulating Continuum Materials

Chenfanfu Jiang*[1], Craig Schroeder†[2], Joseph Teran‡[1,3], Alexey Stomakhin§[3], and Andrew Selle¶[3]

[1]Department of Mathematics, University of California, Los Angeles
[2]Department of Computer Science, University of California, Riverside
[3]Walt Disney Animation Studios

SIGGRAPH 2016 Course Notes Version 1 (May 2016)

* cffjiang@math.ucla.edu
† snubdodecahedron@gmail.com
‡ jteran@math.ucla.edu
§ alexey.stomakhin@disneyanimation.com
¶ andrew.selle@disneyanimation.com

SIGGRAPH '16 Courses, July 24-28, 2016, Anaheim, CA,
ACM 978-1-4503-4289-6/16/07.
http://dx.doi.org/10.1145/2897826.2927348

Appendix A. Deduction of $\delta\mathbf{P}$

- 首先要理解一件事：对于矩阵运算，通常无法直接定义导数，可以用变分来代替
- 因此有下面的基础公式：

$$\frac{\partial A}{\partial \mathbf{F}} : \delta\mathbf{F} = \delta A$$

- 这里的:表示双线性收缩，相当于对应分量乘法相加。它实际上相当于变分语境下的 $\mathrm{d}F(A) = F'(A)\mathrm{d}A$.同样，如果是一阶PK张量$\mathbf{P}$对矩阵$\mathbf{F}$求导，结果应该是一个高阶张量；但我们仍然通过对应的双线性收缩，将其变换为

$$\frac{\partial^2 \Psi}{\partial \mathbf{F}\partial \mathbf{F}} : \delta\mathbf{F} = \delta\left(\frac{\partial \Psi}{\partial \mathbf{F}}\right) = \delta(2\mu(\mathbf{F} - \mathbf{R}) + \lambda(J-1)J\mathbf{F}^{-T})$$

- 大多数导数的性质都可以推广到变分上。
- 下面我们来介绍几个重要变分公式。

Appendix A. Deduction of $\delta\mathbf{P}$ (Cont.)

- 几个常用变分结论：

（1） $\delta\mathbf{F}^{-1} = -\mathbf{F}^{-1}\delta\mathbf{F}\mathbf{F}^{-1}$

（2） $\delta\mathbf{F}^{T} = (\delta\mathbf{F})^{T}$

（3） $\delta\mathbf{F}^{-T} = -\mathbf{F}^{-T}(\delta\mathbf{F})^{T}\mathbf{F}^{-T}$

（4） $\delta J = (J\mathbf{F}^{-T}):\delta\mathbf{F}$

- 所以进一步推导：

$$
\begin{aligned}
&\delta(2\mu(\mathbf{F}-\mathbf{R}) + \lambda(J-1)J\mathbf{F}^{-T}) \\
&= 2\mu\delta\mathbf{F} - 2\mu\delta\mathbf{R} + \lambda\delta(J-1)J\mathbf{F}^{-T} + \lambda(J-1)\delta J\mathbf{F}^{-T} + \lambda(J-1)J\delta\mathbf{F}^{-T} \\
&= 2\mu\delta\mathbf{F} - 2\mu\delta\mathbf{R} + \lambda(2J-1)(J\mathbf{F}^{-T}:\delta\mathbf{F}) - \lambda(J-1)J\mathbf{F}^{-T}(\delta\mathbf{F})^{T}\mathbf{F}^{-T}
\end{aligned}
$$

- 现在问题很明确？怎么求$\delta\mathbf{R}$？考虑$\mathbf{R}$本身是QR分解得来的……

Appendix A. Deduction of $\delta\mathbf{P}$ (Cont.)

$$\delta\mathbf{F} = \delta\mathbf{R}\mathbf{S} + \mathbf{R}\delta\mathbf{S}$$

- $\mathbf{R}$是一个正交矩阵，$\mathbf{S}$是一个正定矩阵。所以两边同时乘一个$\mathbf{R}^T$：

$$\mathbf{R}^T\mathbf{F} = \mathbf{R}^T\delta\mathbf{R}\mathbf{S} + \delta\mathbf{S}$$

- 问题转换为怎么转化$\delta\mathbf{S}$。考虑$\mathbf{F} = \mathbf{R}\mathbf{S}$，则$\mathbf{F}^T\mathbf{R} = \mathbf{S}^T\mathbf{R}^T\mathbf{R} = \mathbf{S}^T$。考虑$\delta\mathbf{S}$是对称矩阵，则$\delta\mathbf{S} = \delta\mathbf{S}^T = \delta(\mathbf{F}^T\mathbf{R}) = (\delta\mathbf{F}^T)\mathbf{R} + \mathbf{F}^T\delta\mathbf{R} = \delta\mathbf{F}^T\mathbf{R} + \mathbf{F}^T\delta\mathbf{R}$。这里其实类似一个分部积分，我们继续往里变形：

$$\mathbf{R}^T\mathbf{F} - \delta\mathbf{F}^T\mathbf{R} = \mathbf{R}^T\delta\mathbf{R}\mathbf{S} + \mathbf{F}^T\delta\mathbf{R}$$

- 还是没办法求解，再做一步变形：$\mathbf{F}^T = \mathbf{S}^T\mathbf{R}^T = \mathbf{S}\mathbf{R}^T$。带进去之后发现上式变成了这样一个形式：

$$\mathbf{R}^T\mathbf{F} - \delta\mathbf{F}^T\mathbf{R} = (\mathbf{R}^T\delta\mathbf{R})\mathbf{S} + \mathbf{S}(\mathbf{R}^T\delta\mathbf{R})$$

- 这种方程是关于$\mathbf{R}^T\delta\mathbf{R}$李雅普诺夫方程，确实没有直接的解法。但别忘了我们是在一个三维张量下讨论，直接形成一个3x3的线性系统求解即可。最后，$\delta\mathbf{R} = \mathbf{R}(\mathbf{R}^T\delta\mathbf{R})$。

Appendix B. A Pratical Differential Strategy for Elasticity

- 基于奇异值对求解过程优化。假设$\mathbf{F} = \mathbf{U\Sigma V}^T$，我们又知道$\Psi(\mathbf{F}) = \widetilde{\Psi}(\Sigma)$，可得到

$$\mathbf{P} = \mathbf{U}\mathrm{diag}\left(\frac{\partial \widehat{\Psi}}{\partial \sigma_i}\right)\mathbf{V}^T$$

- 从直观上理解， $\mathbf{U}, \mathbf{V}$ 都是旋转矩阵，不影响一阶PK张量大小，所以可以等价求解 Σ 。
- 使用该方法对$\Psi(\mathbf{F}) = \mu(\sigma_i - 1)^2 + \frac{\lambda}{2}(J-1)^2$求解：因为$J = \sigma_1\sigma_2\sigma_3$，得到

$$\mathbf{P} = \mathbf{U}\mathrm{diag}\left(2\mu\sigma_i - 2\mu + \frac{\lambda(J-1)J}{\sigma_i}\right)\mathbf{V}^T$$

- 拆成三项分开计算，即得原来所求的式子：

$$\mathbf{U}\mathrm{diag}(2\mu\sigma_i)\mathbf{V}^T = 2\mu\mathbf{U}\mathrm{diag}(\sigma_i)\mathbf{V}^T = 2\mu\mathbf{F}$$
$$\mathbf{U}\mathrm{diag}(2\mu)\mathbf{V}^T = 2\mu\mathbf{U}\mathbf{V}^T = 2\mu\mathbf{R}$$
$$\mathbf{U}\mathrm{diag}\left(\frac{\lambda J(J-1)}{\sigma_i}\right)\mathbf{V}^T = \lambda J(J-1)\mathbf{U}\mathrm{diag}\left(\frac{1}{\sigma_i}\right)\mathbf{V}^T = \lambda J(J-1)\mathbf{U\Sigma}^{-1}\mathbf{V}^T = \lambda J(J-1)\mathbf{F}^{-T}$$

Appendix B. A Pratical Differential Strategy for Elasticity

- 接下来讨论如何计算$\delta\mathbf{P}$。我们希望用类似的逻辑，但因为$\mathbf{P}$本身没有分解特性，所以补充两个正交矩阵：

$$\mathbf{P}(\mathbf{F}) = \mathbf{P}(\mathbf{R}\mathbf{R}^T\mathbf{F}\mathbf{Q}\mathbf{Q}^T) = \mathbf{R}\mathbf{P}(\mathbf{R}^T\mathbf{F}\mathbf{Q})\mathbf{Q}^T$$

- 取$\mathbf{R} = \mathbf{U}$，$\mathbf{Q} = \mathbf{V}$，那么$\mathbf{R}^\mathbf{T}\mathbf{F}\mathbf{Q} = \mathbf{U}^\mathbf{T}\mathbf{F}\mathbf{V} = \mathbf{U}^\mathbf{T}\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\mathbf{T}\mathbf{V} = \boldsymbol{\Sigma}$，从而

$$\delta\mathbf{P}(\mathbf{F}) = \mathbf{U}\left[\frac{\partial\mathbf{P}}{\partial\mathbf{F}}(\boldsymbol{\Sigma}) : (\mathbf{U}^T\delta\mathbf{F}\mathbf{V})\right]\mathbf{V}^T$$

- 为了更方便，我们用下标表示，得到

$$\delta\mathbf{P}_{ij} = U_{ik}\left(\frac{\partial\mathbf{P}}{\partial\mathbf{F}}(\boldsymbol{\Sigma})\right)_{klmn} U_{rm}(\delta\mathbf{F})_{rs}V_{sn}V_{jl}$$

根据变分的基本定义，

$$\delta\mathbf{P}_{ij} = \left(\frac{\partial\mathbf{P}}{\partial\mathbf{F}}(\mathbf{F})\right)_{klmn} (\delta\mathbf{F})_{rs}$$

Appendix B. A Pratical Differential Strategy for Elasticity

- 联立上两式，立即得到

$$\left(\frac{\partial \mathbf{P}}{\partial \mathbf{F}}(\mathbf{F})\right)_{klmn} = \left(\frac{\partial \mathbf{P}}{\partial \mathbf{F}}(\mathbf{\Sigma})\right)_{klmn} U_{ik}U_{rm}V_{sn}V_{jl}$$

- 下面问题就转化成这个奇异值怎么求。考虑罗德里格斯旋转公式：

$$\mathbf{R} = \mathbf{I} + \sin\theta\,\mathbf{K} + (1-\cos\theta)\mathbf{K}^2$$

- 说明任何一个旋转都可以用三个参数来参数化。因此， $\mathbf{U}, \mathbf{V}$ 实际上只有三个自由度。因此，我们可以用九个参数对**F**做重参数化：$(s_1, s_2, s_3, u_1, u_2, u_3, v_1, v_2, v_3)$分别代表奇异值和 **UV** 的参数。所以我们可以把目标的3x3x3x3矩阵转换成一个9x9的来处理。
- 根据链式法则：

$$\frac{\partial \mathbf{P}}{\partial \mathbf{F}}(\mathbf{\Sigma}) = \frac{\partial \mathbf{P}}{\partial \mathbf{S}}(\mathbf{\Sigma})\frac{\partial \mathbf{S}}{\partial \mathbf{F}}(\mathbf{\Sigma})$$

- 并且，上面的过程可以用符号计算软件来求解。以MMA代码为例：

Appendix B. A Pratical Differential Strategy for Elasticity

```
id=IdentityMatrix[3];
var={s1,s2,s3,u1,u2,u3,v1,v2,v3};
Sigma=DiagonalMatrix[{s1,s2,s3}];
cp[k1_,k2_,k3_]={{0,-k3,k2},{k3,0,-k1},{-k2,k1,0}};
vV={v1,v2,v3};
vU={u1,u2,u3};
nv=Sqrt[Dot[vV,vV]];
nu=Sqrt[Dot[vU,vU]];
UU=cp[u1,u2,u3]/nu;
VV=cp[v1,v2,v3]/nv;
U=id+Sin[nu]*UU+(1-Cos[nu])*UU.UU;
V=id+Sin[nv]*VV+(1-Cos[nv])*VV.VV;
F=U.Sigma.Transpose[V];
```

通过罗德里格斯旋转公式
计算参数化矩阵

```
dFdS=D[Flatten[F],{var}];
dFdS0=dFdS/.{u1->e,u2->e,u3->e,v1->e,v2->e,v3->e};
dFdS1=Limit[dFdS0,e->0,Direction->-1];
dSdF0=Inverse[dFdS1];
Phat=DiagonalMatrix[{t1[s1,s2,s3],t2[s1,s2,s3],t3[s1,s2,s3]}];
P=U.Phat.Transpose[V];
dPdS=D[Flatten[P],{var}];
dPdS0=dPdS/.{u1->e,u2->e,u3->e,v1->e,v2->e,v3->e};
dPdS1=Limit[dPdS0,e->0,Direction->-1];
dPdF=Simplify[dPdS1.dSdF0];
```

符号化计算dF/dS

将其中旋转参数都换成ϵ，这样旋转几乎为0

进一步让$\epsilon \to +0$，求极限

将上述结果变成dS/dF

求解$\mathbf{P}(\mathbf{\Sigma})$

求解$\mathbf{P}(\mathbf{F})$

求解dP/dS

求解dP/dF

Appendix B. A Pratical Differential Strategy for Elasticity

- 对2D的情况，只需要参数化成两个旋转度数θ_1, θ_2即可

```
id=IdentityMatrix[2];
var={s1,s2,u1,v1};
S=DiagonalMatrix[{s1,s2}];
U={{Cos[u1],-Sin[u1]},{Sin[u1],Cos[u1]}};
V={{Cos[v1],-Sin[v1]},{Sin[v1],Cos[v1]}};
F=U.S.Transpose[V];
dFdS=D[Flatten[F],{var}];
dFdS0=dFdS/.{u1->e,v1->e};
dFdS1=Limit[dFdS0,e->0,Direction->-1];
dSdF0=Inverse[dFdS1];
Phat=DiagonalMatrix[{t1[s1,s2],t2[s1,s2]}];
P=U.Phat.Transpose[V];
dPdS=D[Flatten[P],{var}];
dPdS0=dPdS/.{u1->e,v1->e};
dPdS1=Limit[dPdS0,e->0,Direction->-1];
dPdF=Simplify[dPdS1.dSdF0];
```

SymPY计算结果

$$\begin{bmatrix} \frac{\partial}{\partial s_1}(t_1(s_1, s_2)) & 0 & 0 & \frac{\partial}{\partial s_2}(t_1(s_1, s_2)) \\ 0 & \frac{s_1 \cdot t_1(s_1, s_2) - s_2 \cdot t_2(s_1, s_2)}{s_1^2 - s_2^2} & \frac{-s_1 \cdot t_2(s_1, s_2) + s_2 \cdot t_1(s_1, s_2)}{s_1^2 - s_2^2} & 0 \\ 0 & \frac{-s_1 \cdot t_2(s_1, s_2) + s_2 \cdot t_1(s_1, s_2)}{s_1^2 - s_2^2} & \frac{s_1 \cdot t_1(s_1, s_2) - s_2 \cdot t_2(s_1, s_2)}{s_1^2 - s_2^2} & 0 \\ \frac{\partial}{\partial s_1}(t_2(s_1, s_2)) & 0 & 0 & \frac{\partial}{\partial s_2}(t_2(s_1, s_2)) \end{bmatrix}$$

Appendix C. Modified Neo-Hookean

- 先列出这一模型的公式：

$$\widehat{\Psi} = \Psi^{\mu}(\mathbf{F}) + \Psi^{\kappa}(J)$$
$$\Psi^{\mu}(\mathbf{F}) = \frac{\mu}{2}(\mathrm{tr}(\mathbf{F}^{T}\mathbf{F}) - d)$$
$$\Psi^{\kappa}(J) = \frac{\kappa}{2}\left(\frac{J^2 - 1}{2} - \log J\right)$$

- 这里$\kappa = \frac{2}{3}\mu + \lambda$表示体积模量。对弹性梯度做如下分解：

$$\mathbf{F} = \mathbf{F}^{dev}\mathbf{F}^{vol} = (J^{a}\mathbf{F}) \cdot (J^{-a}\mathbf{I})$$

- 其中，$a = -1/d$，dev表示形状的拉伸和扭曲，不涉及体积变化；vol表示体积变化。
- 为什么要这么分解？因为无形变的时候我们希望

$$|J^{a}\mathbf{F}| = 1 \Rightarrow J^{ad}|\mathbf{F}| = 1 \Rightarrow J^{ad+1} = 0 \Rightarrow a = -1/d$$

- 也就是此时构造的新参量$J^{a}\mathbf{F}$是不涉及形变的。这样，我们可以将原式改写成

$$\widehat{\Psi} = \Psi^{\mathrm{dev}}(J^{a}\mathbf{F}) + \Psi^{\mathrm{vol}}(J)$$

Appendix C. Modified Neo-Hookean

- 下面开始对Kirhoff张量进行推导。首先计算$\mathbf{P}_{dev}$

$$\mathbf{P}_{dev} = \frac{\partial \Psi^{dev}}{\partial \mathbf{F}}\bigg|_{J^a\mathbf{F}} : \frac{\partial\, J^a \mathbf{F}}{\partial \mathbf{F}}$$

- 第一项可以直接求导，结果是$\mu\mathbf{F}$。第二项比较抽象，我们知道它会是个四阶的张量：

$$\frac{\partial\, J^a \mathbf{F}}{\partial \mathbf{F}} = \mathbf{F} \otimes \frac{\partial\, J^a}{\partial \mathbf{F}} + J^a \mathbf{I}^{4\text{th}} = J^a\left(a\mathbf{F}\otimes\mathbf{F}^{-T} + \mathbf{I}^{4th}\right)$$

- 综合上两项，得到

$$\mathbf{P}^{dev} = \mu J^a \mathbf{F}\colon J^a\left(a\mathbf{F}\otimes\mathbf{F}^{-T} + \mathbf{I}^{4th}\right) = \mu J^{2a}\left(a\mathbf{F}\colon\mathbf{F}\otimes\mathbf{F}^{-T} + \mathbf{F}\right)$$

- 从而，Kirhoff张量$\tau_{ij}^{dev} = P_{ik}^{dev} F_{jk} = \mu J^{2a}\left(aF_{kl}F_{kl}\delta_{ij} + F_{ik}F_{jk}\right)$。考虑$F_{ik}F_{jk} = b_{ij}$，即左Cauchy张量，再带入$a = -1/d$，得到

$$\boldsymbol{\tau}_{dev} = \mu J^{-\frac{2}{d}}\left(\mathbf{b} - \frac{1}{d}tr(\mathbf{b})\boldsymbol{I}\right) = \mu J^{-\frac{2}{d}}\mathrm{dev}(\mathbf{b})$$

- 这里dev是一个新定义的算符，定义如上。

Appendix C. Modified Neo-Hookean

- 对Volume项推导则比较直接：

$$\mathbf{P}_{vol} = \frac{\Psi^{\text{vol}}(J)}{\partial \mathbf{F}} = J\Psi^{\text{vol}\prime}(J)\mathbf{F}^{-T}$$

$$\boldsymbol{\tau}_{vol} = J\Psi^{\text{vol}\prime}(J)\mathbf{I}$$

- 把两项综合起来，得到

$$\boldsymbol{\tau} = \mu J^{-\frac{2}{d}}\text{dev}(\mathbf{b}) + J\Psi^{\text{vol}\prime}(J)\mathbf{I}$$

- 且容易发现$\text{dev}(\boldsymbol{\tau}) = \boldsymbol{\tau}_{dev}$。
- 最后，我们不加证明的写出$\delta\mathbf{P}^{dev}$和$\delta\mathbf{P}^{vol}$的形式：

$$\begin{aligned}
\delta \boldsymbol{P}^{\text{dev}} =& 2a\mu J^{2a}\boldsymbol{F}^{-T} : \delta F(a\boldsymbol{F} : \boldsymbol{F}\boldsymbol{F}^{-T} + \boldsymbol{F}) \\
& + \mu J^{2a}\left(2a\boldsymbol{F} : \delta\boldsymbol{F}\boldsymbol{F}^{-T} - a\boldsymbol{F} : \boldsymbol{F}(\boldsymbol{F}^{-1}\delta\boldsymbol{F}\boldsymbol{F}^{-1})^{T} + \delta\boldsymbol{F}\right) \\
\delta \boldsymbol{P}^{\text{vol}} &= \delta\left(J\Psi^{\text{vol}\prime}(J)\boldsymbol{F}^{-T}\right) \\
&= \delta J\Psi^{\text{vol}\prime}(J)\boldsymbol{F}^{-T} + J\delta(\Psi^{\text{vol}\prime}(J))\boldsymbol{F}^{-T} + J\Psi^{\text{vol}\prime}(J)\delta(\boldsymbol{F}^{-T}) \\
&= \delta J\Psi^{\text{vol}\prime}(J)\boldsymbol{F}^{-T} + J\delta J\Psi^{\text{vol}\prime\prime}(J)\boldsymbol{F}^{-T} + J\Psi^{\text{vol}\prime}(J)\delta(\boldsymbol{F}^{-T})
\end{aligned}$$