Material Point Method for Water and Sand Simulation

Zhuo Liu
zhuol@mit.edu

1 Introduction

Material point method (MPM) [1] mainly rises as the generalization of Particle-in-Cell (PIC) that is widely used in physics, especially plasma physics, research. MPM methods combine Lagrangian material particles (points) with Eulerian Cartesian grids, which treats the material as a whole and implicitly handles self-collision and fracture with the internal force calculating on the Eulerian grids. It has been shown to be a very effective method for simulating various materials including solid and complex fluids in computer graphics. It can even handle very challenging simulations like coupling between different materials.

2 Background work

In this project, I implement the Affine Particle-in-Cell (APIC) [3] method for stable and angular momentum conserving particle/grid transfers. APIC was introducted by Chenfanfu Jiang in 2015. The key observation is that normally a single particle receives data from multiple grid points, but it is typically forced to reduce those influences to a single value, leading to loss of information. Specifically, the regular PIC severely damps the rotational motion because the transfer from the grid back to particles does not conserve angular momentum. APIC solves this problem by idealizing the velocity as locally affine on each particle and defining affine transfer between grids and particles.

3 Algorithm

3.1 Notation

The general rule for the notation is describing as the following: subscripts p and q are used to refer to particles. Subscripts i and j are used to refer to regular grid indices. Subscriptsα refers to different axis direction. The superscripts n is to indicate different time steps. The quantities that are used are summarized in Table 1.

Table 1. Table of notation. Some superscript n is omitted if the meaning is already clear.

category	symbol	meaning
particle	m_p	particle mass
particle	$\mathbf{x_p^n}$	particle position
particle	$\mathbf{v_p^n}$	particle velocity
particle	$\mathbf{B_p^n}$	affine momentum
particle	$\mathbf{C_p^n}$	particle velocity derivative
particle	$\mathbf{D_p^n}$	affine inertial tensor
particle	$(\nabla \mathbf{v})_p$	grid-based velocity gradient
particle:sand	$\mathbf{F_p^E}$	sand elastic deformation gradient
particle:sand	$\mathbf{F_p^P}$	sand plastic deformation gradient
particle:sand	α_p	sand yield surface size
particle:sand	q_p	sand hardening parameter
particle:water	J^w	water deformation gradient
grid	m_i^n	grid node mass
grid	$\mathbf{x_i^n}$	grid node location
grid	$\overline{\mathbf{x_i}}^{\mathbf{n+1}}$	gird posistion moved by $\overline{\mathbf{v_i}}^{\mathbf{n+1}}$
grid	$\mathbf{v_i^n}$	grid velocity
grid	$\mathbf{v_i^*}$	updated grid velocity with force
grid	$\overline{\mathbf{v_i}}^{\mathbf{n+1}}$	updated grid velocity after collision
grid	$\tilde{\mathbf{v_i}}^{\mathbf{n+1}}$	final updated grid velocity
grid	$\mathbf{f_i}$	force on the grid
global	w_{ip}	interpolation weight
global	∇w_{ip}	interpolation weight gradient

3.2 weights

When doing the transformation between particles and grids, we need to do interpolation, which requires computing a weight w_{ip} for each particle p on each grid node i. The weights are evaluated based on a kernel function, which is chosen with trade-offs with respect to smoothness and computational efficiency. I am using the kernel

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

$$N(\Delta \mathbf{x_{pi}}) = \hat{N}(\Delta \mathbf{x_{pi_x}}/h)\hat{N}(\Delta \mathbf{x_{pi_y}}/h)\hat{N}(\Delta \mathbf{x_{pi_z}}/h) \tag{2}$$

where h is grid size. For particle p, its weight on each grid i is then

$$w_{ip}^n = N(\mathbf{x_p^n} - \mathbf{x_i^n}) \tag{3}$$

3.3 Particle to grid

The first step is to transfer the quantities of particles to fixed Eulerian Cartesian grids. To transfer grid mass

$$m_i^n = \sum_p w_{ip}^n m_p \tag{4}$$

The velocity tranformation is described by APIC

$$\mathbf{v}_i^n = \frac{1}{m_i^n} \sum_p w_{ip}^n m_p \left(\mathbf{v}_p^n + \mathbf{B}_p^n \left(\mathbf{D}_p^n \right)^{-1} \left(\mathbf{x}_i^n - \mathbf{x}_p^n \right) \right) \tag{5}$$

$\mathbf{D_p^n}$ is like an inertia tensor

$$\mathbf{D}_p^n = \sum_i w_{ip}^n \left(\mathbf{x}_i^n - \mathbf{x}_p^n \right) \left(\mathbf{x}_i^n - \mathbf{x}_p^n \right)^T = \frac{h^2}{3}\mathbf{I} \tag{6}$$

The particle velocity derivative is $\mathbf{C}_\mathbf{p}^\mathbf{n} = \mathbf{B}_p^n \left(\mathbf{D}_p^n\right)^{-1}$. $\mathbf{B}_p^n$ is the affine momentum, which can be initialized with zeros.

3.4 Grid update

I use an explicit way to update grids. That is

$$\mathbf{v}_i^\star = \mathbf{v}_i^n + \frac{\Delta t}{m_i^n}\mathbf{f}_i\left(\left\langle \mathbf{F}_p^{E,n}\right\rangle\right) \tag{7}$$

The force is evaluated by the elastic deformation gradient $\mathbf{F}_p^E$ (at the time step n, which is omitted in this subsection). And the potential energy Ψ can be calculated with a sum on particles' deformation gradient

$$\Psi = \sum_p V_p^0 \psi\left(\mathbf{F}_p^E(\langle \mathrm{x}_i \rangle)\right) \tag{8}$$

$$\mathbf{f}_i\left(\left\langle \mathbf{F}_p^E\right\rangle\right) = -\frac{\partial \Psi}{\partial \mathbf{x}_i} = -\sum_p V_p^0 \left(\frac{\partial \psi}{\partial \mathbf{F}}\left(\mathbf{F}_p^E\right)\right)\left(\mathbf{F}_p^{E,n}\right)^T \nabla w_{ip}^n \tag{9}$$

where V_p^0 is the initial volume of each particle.

3.4.1 Constitutive model.

To compute $\frac{\partial \psi}{\partial \mathrm{F}}\left(\mathbf{F}_p^E\right)$, we need constitutive model.

For water simulation, I use J^w instead of $\mathbf{F}_p^E$ to represent the elastic deformation gradient. J^w is a scalar and the constitutive model is

$$\psi^w\left(J^w\right) = -k\left(\frac{\left(J^w\right)^{(1-\gamma)}}{1-\gamma} - J^w\right) \tag{10}$$

where k and γ are water parameters.

Sand is more complicated. The deformation matrix is separated into elastic part $\mathbf{F}_p^{E,n}$ and plastic part $\mathbf{F}_p^{P.n}$. The model is most conveniently written in terms of singular value decomposition of elastic deformation $\mathbf{F}_p^E = \mathbf{U}\Sigma\mathbf{V}^\mathbf{T}$ and the force only depends on the elastic part

$$\psi(\mathbf{F}_\mathbf{p}^\mathbf{E}) = \mu\,\mathrm{tr}\left((\ln\Sigma)^2\right) + \frac{1}{2}\lambda(\mathrm{tr}(\ln\Sigma))^2 \tag{11}$$

$$\frac{\partial \psi}{\partial \mathbf{F}_\mathbf{p}^\mathbf{E}} = \mathbf{U}\left(2\mu\Sigma^{-1}\ln\Sigma + \lambda\,\mathrm{tr}(\ln\Sigma)\Sigma^{-1}\right)\mathbf{V}^T \tag{12}$$

where μ and λ are sand parameters.

3.4.2 Collision.

To prevent the particles going outside of the simulation, we need to add borders and apply collisions between borders and particles in the simulations. I am using the "separating" type of collision in the project.

Writing $d(\mathbf{x}_\mathbf{i}^\mathbf{n})$ as the distance between the grid node and the border and $d(\mathbf{x}_\mathbf{i}^\mathbf{n} + \mathbf{v}^* * \Delta t)$ as the distance between the grid node and the border after one time step, if $d(\mathbf{x}_\mathbf{i}^\mathbf{n}) < 0$ 0 and $d(\mathbf{x}_\mathbf{i}^\mathbf{n} + \mathbf{v}^* * \Delta t) < d(\mathbf{x}_\mathbf{i}^\mathbf{n})$, it means that the particle is still going into the border and we need velocity update

$$\overline{\mathbf{v}_\mathbf{i}}^{\mathbf{n+1}} = \mathbf{v}_\mathbf{i}^* + \left(d(\mathbf{x}_\mathbf{i}^\mathbf{n}) - d(\mathbf{x}_\mathbf{i}^\mathbf{n} + \mathbf{v}^* * \Delta t)\right)/\Delta t \tag{13}$$

If $d(\mathbf{x}_\mathbf{i}^\mathbf{n}) \geq 0$ but $d(\mathbf{x}_\mathbf{i}^\mathbf{n} + \mathbf{v}^* * \Delta t) < 0$, it also means that the particle is going into the border, and we need velocity update according to distance that goes into the border

$$\overline{\mathbf{v}_\mathbf{i}}^{\mathbf{n+1}} = \mathbf{v}_\mathbf{i}^* + \left(0 - d(\mathbf{x}_\mathbf{i}^\mathbf{n} + \mathbf{v}^* * \Delta t)\right)/\Delta t \tag{14}$$

3.4.3 Friction.

Friction between the particle and border is applied after the collision update. First we need to find the tangential part of the grid velocity $\mathbf{v_{it}} = \overline{\mathbf{v_i}}^{\mathbf{n+1}} - \mathbf{n}\cdot\overline{\mathbf{v_i}}^{\mathbf{n+1}}$ and tangential direction $\mathbf{t} = \frac{\mathbf{v_{it}}}{||\mathbf{v_{it}}||}$. During the collision, the momentum impulse is $\mathbf{j} = m_i^n(\overline{\mathbf{v_i}}^{\mathbf{n+1}} - \mathbf{v}_\mathbf{i}^*)$. The maximum momentum that the grid can lose due to friction is $\mu_b||\mathbf{j}||$. Therefore, after friction,

$$\tilde{\mathbf{v_i}}^{\mathbf{n+1}} = \overline{\mathbf{v_i}}^{\mathbf{n+1}} - \frac{\mu_b}{m_i^n}||\mathbf{j}||\mathbf{t}\text{ if }||\mathbf{v_{it}}|| \leq \frac{\mu_b}{m_i^n}||\mathbf{j}|| \tag{15}$$

$$\tilde{\mathbf{v_i}}^{\mathbf{n+1}} = \mathbf{n}\cdot\overline{\mathbf{v_i}}^{\mathbf{n+1}}\text{ if }||\mathbf{v_{it}}|| > \frac{\mu_b}{m_i^n}||\mathbf{j}|| \tag{16}$$

3.5 Grid to particle

Transfer grid node velocity back to particle

$$\mathbf{v}_p^{n+1} = \sum_i w_{ip}^n \tilde{\mathbf{v}}_i^{n+1} \tag{17}$$

Affine momentum update is

$$\mathbf{B}_p^{n+1} = \sum_i w_{ip}^n \tilde{\mathbf{v}}_i^{n+1}\left(\mathbf{x}_i^n - \mathbf{x}_p^n\right)^T \tag{18}$$

3.6 Particle update

Position update

$$\mathrm{x}_p^{n+1} = \sum_i w_{ip}^n \overline{\mathbf{x}}_i^{n+1} \tag{19}$$

$$\text{where } \overline{\mathbf{x}}_i^{n+1} = \mathbf{x}_i^n + \overline{\mathbf{v_i}}^{\mathbf{n+1}} * \Delta t \tag{20}$$

Deformation update for water is simple

$$J_p^{w,n+1} = \left(\mathrm{I} + \Delta t\,\mathrm{tr}\left(\nabla \mathrm{v}_p^{w,n+1}\right)\right)J^{w,n} \tag{21}$$

$$\text{where } (\nabla\mathbf{v})_p = \sum_i \overline{\mathbf{v}}_i^{n+1}\left(\nabla w_{ip}^n\right)^T \tag{22}$$

3.6.1 Plasticity of Sand.

Deformation update for sand is complicated. That is because we have to update the plasticity and also take the hardening effect into account. The first step is to assume plasticity part is not changing

$$\hat{\mathbf{F}}_p^{E,n+1} = \mathbf{F}_p^{E,n} + \Delta t(\nabla\mathbf{v})_p\mathbf{F}_p^{E,n} \tag{23}$$

Let $\hat{\mathbf{F}}_p^{E,n+1} = \mathbf{U_p}\Sigma_\mathbf{p}\mathbf{V}_\mathbf{p}^\mathbf{T}$, $\epsilon_p = \ln\Sigma_p$ and $\hat{\epsilon}_p = \epsilon_p - \frac{\mathrm{tr}(\epsilon_p)}{d}\mathbf{I}$. Plasticity deformation is described as

$$\delta\gamma_p = \left\|\hat{\epsilon}_p\right\|_F + \frac{d\lambda + 2\mu}{2\mu}\,\mathrm{tr}\left(\epsilon_p\right)\alpha_p^n \tag{24}$$

where d is the spatial dimension (2 or 3), α_p is the yield surface size. There are three possible cases for plasticity update. If $\delta\gamma_p \leq 0$, then there is no plasticity and $\mathbf{F}_p^{E,n+1} = \hat{\mathbf{F}}_p^{E,n+1}$, $\delta q_p = 0$. If $tr(\epsilon_\mathrm{p}) > 0$, $\mathbf{F}_p^{E,n+1} = \mathbf{U_p}\mathbf{V}_\mathbf{p}^\mathbf{T}$, $\delta q_p = \left\|\epsilon_p^{E,n+1}\right\|_F$. Otherwise, $\mathbf{F}_p^{E,n+1} = \mathbf{U_p}e^{\mathbf{H_p}}\mathbf{V}_\mathbf{p}^\mathbf{T}$, $\delta q_p = \delta\gamma_p$, where $\mathbf{H}_p =$

$\epsilon_p - \delta\gamma_p \frac{\hat{\epsilon}_p}{\|\hat{\epsilon}_p\|_F}$ and δq_p is for hardening update. Based on the updated elastic deformation, the plastic deformation is obtained by assuming the full deformation gradient

$$\mathbf{F}_p^{P,n+1} = \left(\mathbf{F}_p^{E,n+1}\right)^{-1} \hat{\mathbf{F}}_p^{E,n+1}\mathbf{F}_p^{P,n} \tag{25}$$

3.6.2 Hardening of Sand. Hardening basically means that the plasticity can change when the material reaches different deformation level. The hardening update is defined by

$$q_p^{n+1} = q_p^n + \delta q_p \tag{26}$$

$$\phi_{F_p} = h_0 + \left(h_1 q_p^{n+1} - h_3\right) e^{-h_2 q_p^{n+1}} \tag{27}$$

$$\alpha_p^{n+1} = \sqrt{\frac{2}{3}}\frac{2\sin\phi_{F_p}}{3-\sin\phi_{F_p}} \tag{28}$$

where hardening is characterized by q_p and h_0, h_1, h_2, h_3 are hardening parameters. It determines the yield surface size α_p, which is used to calculate plasticity in (24). In practice, q_p^0 is initialized as zero.

3.6.3 Summary for sand particle update. First calculate (23) with the old yield surface size. Then update the elastic deformation gradient $\mathbf{F_p^{E,n+1}}$ according to the three different cases and update the plastic deformation gradient $\mathbf{F_p^{p,n+1}}$ according to (25). Finally, apply hardening and calculate the new yield surface size α_p^{n+1} for next update's use.

4 Implementation

4.1 Initialization

The Eulerian Cartesian grids are 101 x 101, which defines the simulation domain (2D). 4 lines (in 3D, 6 planes) are placed 2 grids away from the furthest grids in each direction, forming a rectangle (in 3D, a cube) border (as shown in Figure 1, 2 as orange lines). The reason why sparing 2 grids from the furthest grids is that the weight kernel function is zero when $|x| \geq 2$. (However, there is still a potential that if the particle's speed is super fast and the time step is too large, the particle can still go out of the border and causes segmentation error.)

For water simulation, I used the same parameters given in ([5]) ($k = 50, \gamma = 3$). Volume is $V_p^0 = 1.0$ and particle mass is $m_p = 0.001$. The gradient is pointing at the negative y direction with amplitude 10. 30 water particles with initial speed 10 pointing at the negative y direction are generated at the top of the simulation box every 3 steps, with 3000 particles in total.

For sand simulation, I used the same parameter given in [4], ($h_0 = 0.61, h_1 = 0.16, h_2 = 0.2, h_3 = 0.17, \lambda = 2.04 \times 10^6, \mu = 1.36 \times 10^5$). A block of sand with 400 particles are generated before the simulation starts. $m_p = 32.0$ and $V_p^0 = 2.0$.

The gravity is set to be 10.0 pointing at the negative y direction.

4.2 Rendering

For visualization, I use OpenGL 3.3 APIs (instead of GLOO). I directly render each particle as a blue sphere for simplicity. However, the difficult part is that MPM requires a huge amount of particles to make the visualization looks real, which is too expensive for a personal computer even for 2D simulation. To make to visualization feasible, I use instancing. Instancing is a technique where we draw many (equal mesh data) objects at once with a single render call, saving us all the CPU -> GPU communications each time we need to render an object. I achieve this by only creating one sphere mesh but adding a instance matrix in the vertex shader to assign different location for each particle. I set up a perspective camera to view the scene.

4.3 Other details

Some other optimizations have been realized to speed up the simulation a little bit. One observation is that only a part of grids is active (the grid nodes with mass larger than zero), so there is no need to summing over every grid node for each particle. In fact, only 16 nodes (in 2D, 64 in 3D) are active for each particle in one overall update step. I can store the active grid node indices for each particle and there is no need to search for active node grids in the whole space. Based on this idea, because the position of each particle is not changed until the final update in one update step, I can also store the w_{ip} and dw_{ip} for each particle right at the beginning of each step. There is no need to calculate it several times for each particle.

5 Results

I demonstrate the the model by showing two simulations – one for water and the other for sand. In Figure 1 water is falling from the top with some initial speed. The water then hits on the ground under the influence of gravity and collides with the bottom, left and right boundary. some turbulence features are also captured by the model. In Figure 2, a block of sand falls down under the influence of gravity. With the plasticity and hardening model, the sand forms a smooth granular flow and piles up at the bottom.

6 Summary and future work

Although my work shows relatively decent results for physics-based simulation of complicated material, it does not come with very nice visualization. For example, the water does not look real. One reason is that I did not apply texture, light and other effects on it. Furthermore, the number of particles is far from enough. I only have 3000 particles in my water simulation and I drew each particle relatively big to compensate for the distance between particles, which is unreal. It still took me an hour to render a 30 seconds videos (1500 frames) on my personal computer (i7-9700k) even under this circumstance. This process is even more unbearably slow

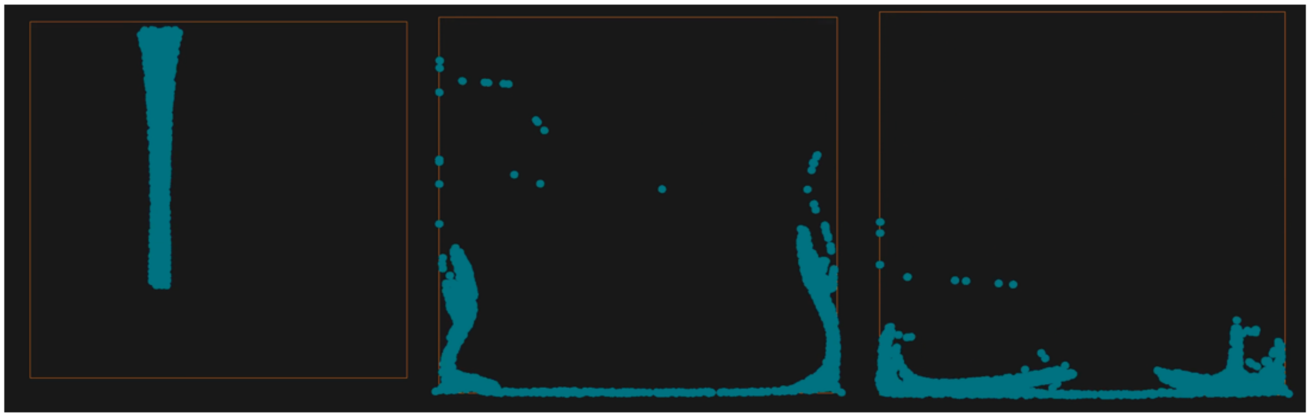

Figure 1. Water.

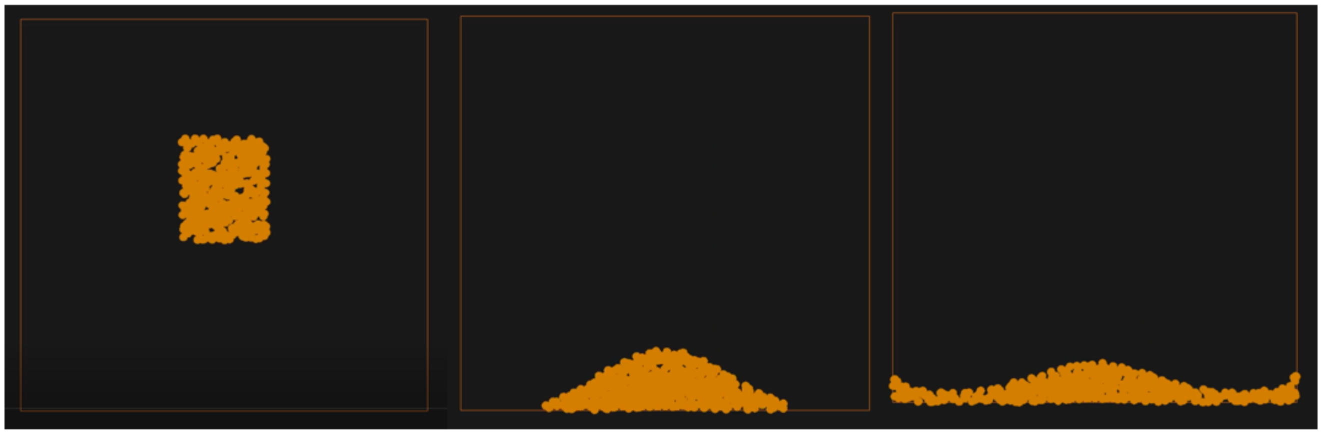

Figure 2. Sand.

in 3D, which is the reason why I demonstrate my results in 2D even the code is capable of doing 3D. The reason behind this is that the algorithm is slow. In 2013, MPM was used to render a snow scene in the movie "Forzen" by Disney engineer Alexey Stomakhin, and it took more than a week on a cluster. Recently years, many improvements have been made and the efficiency has been significantly. For example, Hu. (2018) [2] proposed Moving Least Square Material Point Method (MLS-MPM), which allows all MPM simulations to run two times faster than before. I can try to implement this method in the future.

Another aspect it that currently I am just doing water and sand simulation separately. The simulation of gravity driven landslides and debris flows with porous sand and water interactions can be achieved by considering the "unilateral hyper-elasticity" of the sand and incorporating the momentum exchange between sand and water [5]. I can improve this work by implementing this multi-species model.

References

[1] Jiang Chenfanfu, Schroeder Craig, Teran Joseph, Stomakhin Alexey, and Selle Andrew. 2015. The Material Point Method for Simulating Continuum Materials. https://www.math.ucla.edu/~cffjiang/research/mpmcourse/mpmcourse.pdf. [Online; accessed 30-Oct-2021].

[2] Yuanming Hu, Yu Fang, Ziheng Ge, Ziyin Qu, Yixin Zhu, Andre Pradhana, and Chenfanfu Jiang. 2018. A moving least squares material point method with displacement discontinuity and two-way rigid body coupling. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–14.

[3] Chenfanfu Jiang, Craig Schroeder, Andrew Selle, Joseph Teran, and Alexey Stomakhin. 2015. The Affine Particle-in-Cell Method. *ACM Trans. Graph.* 34, 4, Article 51 (July 2015), 10 pages. https://doi.org/10.1145/2766996

[4] Gergely Klár, Theodore Gast, Andre Pradhana, Chuyuan Fu, Craig Schroeder, Chenfanfu Jiang, and Joseph Teran. 2016. Drucker-Prager Elastoplasticity for Sand Animation. *ACM Trans. Graph.* 35, 4, Article 103 (jul 2016), 12 pages. https://doi.org/10.1145/2897824.2925906

[5] Andre Pradhana Tampubolon, Theodore Gast, Gergely Klár, Chuyuan Fu, Joseph Teran, Chenfanfu Jiang, and Ken Museth. 2017. Multi-Species Simulation of Porous Sand and Water Mixtures. *ACM Trans. Graph.* 36, 4, Article 105 (July 2017), 11 pages. https://doi.org/10.1145/3072959.3073651