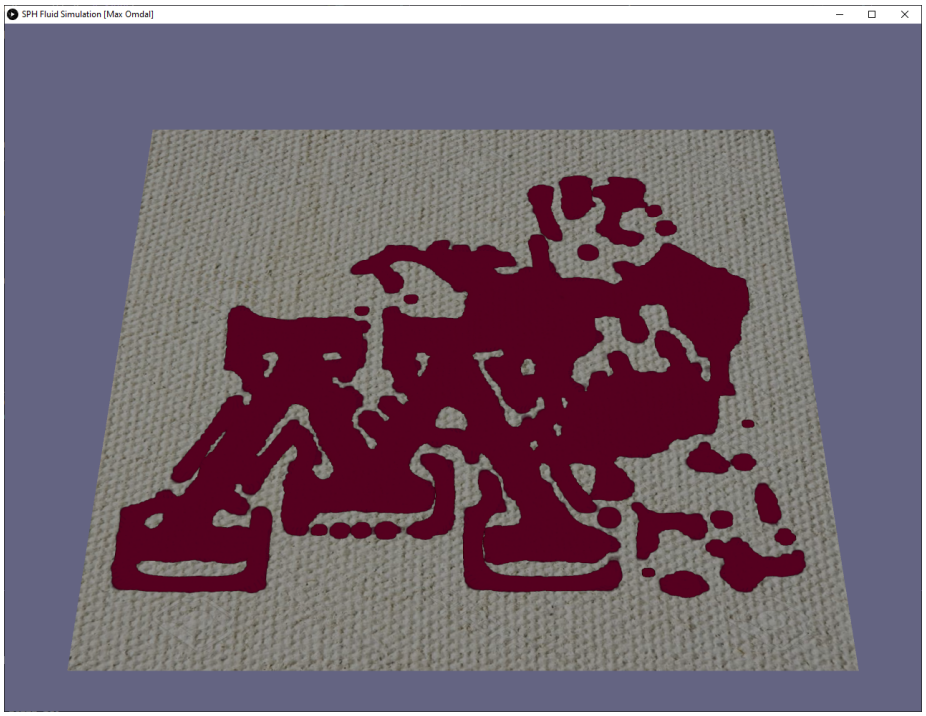


Figure 1: A painting produced using an SPH fluid simulation and applying particle's 2D positions to a canvas

Virtual Ink Painting using SPH simulations

Maxwell Omdal

July 2020

1 Introduction

Real-time fluid simulation is desirable for it's applications in video games and interactive visualizations/simulations. Here we apply a particle-based fluid simulation to create a virtual painting. Fluid simulations can also be useful in Movie virtual effects, and studies of fluid simulation. Smoothed-Particle Hydrodynamics (SPH) is a common and proven technique for animation in computer

graphics. Ihmsen et al reviewed SPH Fluids and how they can be applied generally to scenes requiring rigidbody-fluid interaction, foaming, wave effects and more [3]. The particle method, however can be limited by the computational complexity of updating each particle, specifically collision and neighbor detection. A spatial data structure can be used to reduce this complexity. In this paper, I outline an implementations of an SPH Fluid simulation technique first presented in [2].

2 Guided Work

I implemented work by Clavet et al for simulating viscoelastic non-compressible fluids. This Lagrangian method relies on a simulation step and a displacement correction step. This saves the simulation from needing a very small time step because there is no explicit integration step that could lead to instability at larger time steps. This dramatically reduces the computation requirements. Note a smaller timestep will still produce smoother results.

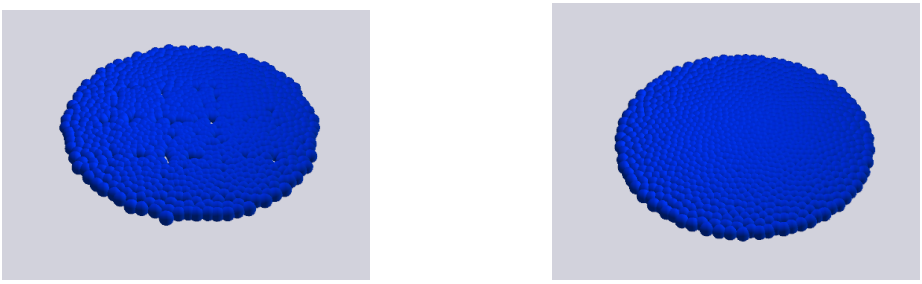
The displacement update is calculated as a function of each particle's distance to it's neighbors and it's gravity. This update is called the Double Density relaxation step. As noted, we are simulating non-compressible fluids, so there is no true change in density. A faux density parameter is used to define the forces that create cohesion and repulsion within a fluid. A set of parameters are provided that can be tuned to achieve the results that are desired. These parameters define the stiffness and density of the fluid. Other attributes such as the effects of gravity can be tweaked to further adjust results.

3 Implementation

This fluid simulation implements a viscous non-compressible fluid. Take note that in order to simplify computation and create an intuitive and clear-cut implementation, the non-compressible requirement, which would be true for a water simulation is relaxed.

3.1 Octrees

One of the complications of a particle-based fluid simulation is computing the neighbors of each particle. This is an exponential operation without a spatial data structure. To reduce this, I implemented an octree with an adjustable maximum depth. One of the disadvantages of an octree for a particle based simulation like this is the possibility of a particle to be in multiple octants at once. This is less of a problem with a hashed-grid approach like what is implemented in Clavet et al. I chose to implement an octree because I had already written a generic octree. When a particle exists in multiple octants, the worst-case complexity is actually greater than using no spatial data structure. The worst-case is when all particles contain the center-point of the octree in their radius. In this scenario, the tree holds 9 references to each object at



(a) With Array

(b) With Hash Set

Figure 2: Two frames from a fluid hitting a planar surface. 2a shows the overcompensation for particles along the edges of an octant. 2b shows the correction by removing duplicates using a hash set.

it's max depth. When particles are at this position, a max depth is required to prevent infinite recursion as we subdivide octants. This creates a greater problem by counting particles multiple times when finding neighbors. This can lead to particles disproportionately affecting others. We fix this by using a data structure which prevents duplicates, shown in figure 6. inserting into a hash set can be much slower than an array list

3.2 Ink Transfer

Particles are placed on a plane, in two layers. This creates an initial 3 dimensional volume. The XZ position of each particle is determined by a preloaded image that is thresholded. We then place a particle for each particle that is above the threshold value. As the particles disperse, the ink is "seeped" into the canvas on the plane. This is simulated by finding the position of each particle on the plane and stamping a watercolor texture on a canvas. The user can then interact with the painting creation by left-clicking to select a portion of particles and dragging them across the screen. This might not be a valuable tool for creation, but I hope it acts as inspiration for research into painterly experiences.

4 Results

5 Conclusions and Future Work

A particle-based approach is convenient because it can be extended to account for many details of fluid simulation. A marching cubes algorithm can approximate a smooth continuous surface from a collection of particles [2, 3]. Papers like WetBrush by Chen et al pave the way for advanced physics-based painting tools [1]. To allow for much higher resolution without overloading the computer,

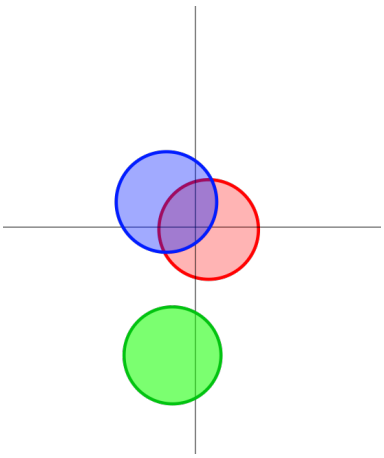
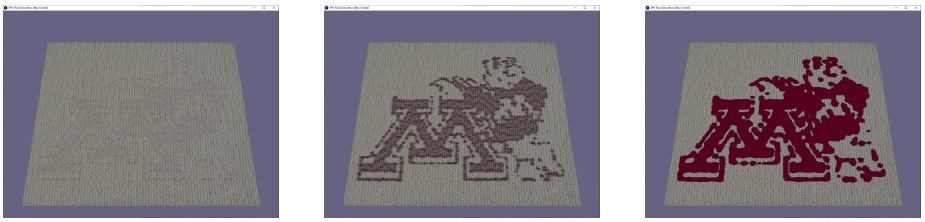


Figure 3: Example of circles overlapping several quadrants. The blue and red circles must be added to all 4 quadrants, while the green circle must be added to the bottom two quadrants. If we wanted to find the circles that share a quadrant with the blue circle, we would have to check all four quadrants, and compare to the red circle 4 times and the green circle 2 times. This is extended to octrees as well.



(a) t = 0.5 seconds (b) t = 4 seconds (c) t = 8 seconds

Figure 4: Shows the evolution of the painting. As the transparent watercolor mark is layered, it becomes darker. The particles individual movement due to cohesion and adhesion create an affect as they bunch together and creates a "blotch" effect, like you might see if you thickly painted with ink.

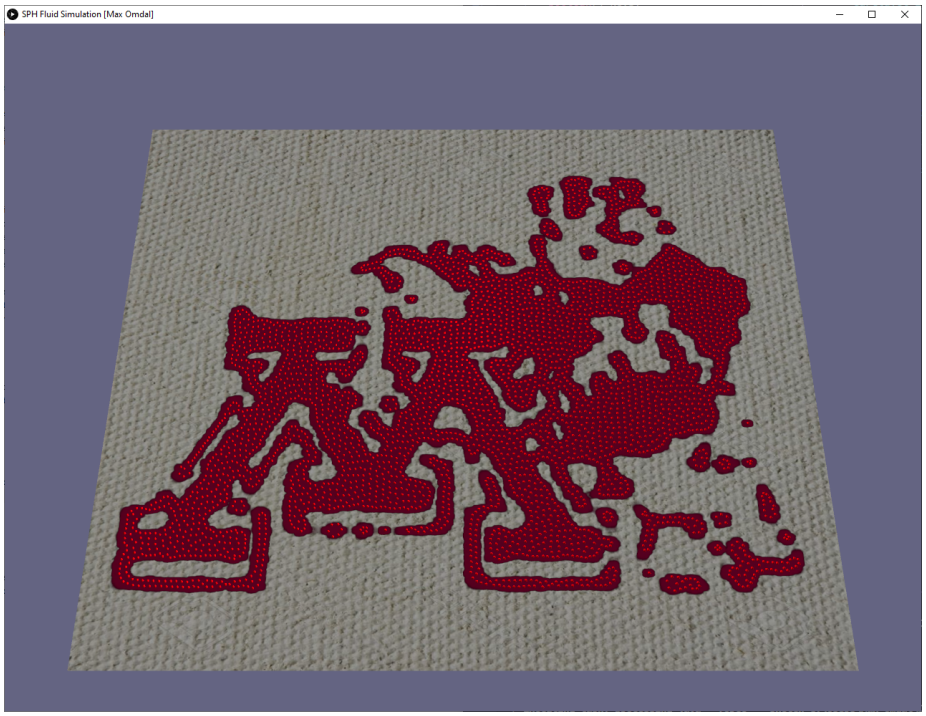


Figure 5: What the painting looks like with the particles visualized

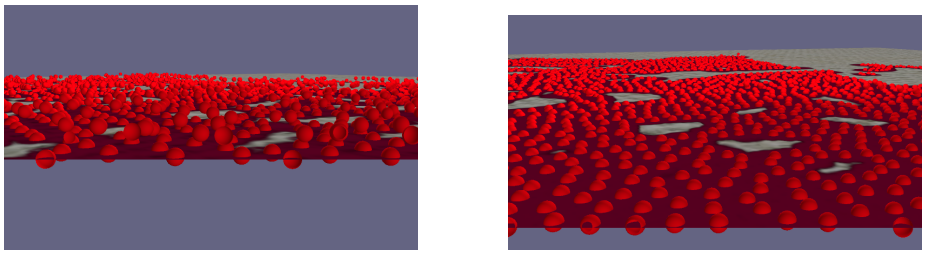
they rely on a mixture of particle-based an density gradient mapping to only require particle-based simulation near the brush where the user is working. They also deploy a GPU-based rendering strategy. Both of these practices could find practical applications in this painterly effects simulation. Currently, the program can handle around 2000 to 4000 particles, which is particularly reliant on the particles being mostly distributed on a flat plane, interacting with few neighbors. Since for each update, each particle is not affected by concurrent changes in neighbor particles, using a GPU-based approach would be relatively straightforward.

References

[1]

[2] S. Clavet, P. Beaudoin, and P. Poulin. Particle-based viscoelastic fluid simulation. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation - SCA '05*, page 219. ACM Press, 2005.

[3] M. Ihmsen, J. Orthmann, B. Solenthaler, A. Kolb, and M. Teschner. Sph fluids in computer graphics. page 22.



(a) Initial particles

(b) Settled Particles

Figure 6: Figure 6a shows the initial locations of particles. Figure 6b shows how these particles spread out as they move across the canvas, eventually flattening everywhere