Animation of Liquids

CSE530 Final Project Technical Report

Yu-Chuan Kuo

Abstract

In this project, I implement a physics-based system to create animation of liquids. The fluid flow is computed by solving Navier Stokes Equations using both the Lagragian method and an implicit solver. A scalar field representing liquid densities is solved over time steps, and then an adaptive marching cube algorithm is used to extract the isosurface of the liquid of interest. For the two-dimensional scheme, the system is designed to be interactive, due to low computation cost. As for a 3D case, the solver cannot act in real-time, so I try to improve the quality of the results. I implement mesh consolidation to compute vertex normal, smoothing the visual appearance of the coarse mesh of liquid surface. Finally, the scene is rendered with a ray tracer and rendered image sequences are collected to make video clips.

1. Introduction

Animators have always been interested in generating animation of liquids. However, it is difficult for users to create visually pleasing fluid motion manually. Hence physics-based approaches have been widely used in computer graphics for the past few years.

The goal of this project is to develop a physics-based system that generates realistic animation of liquid, which is a specialized case of fluid animation. For liquids, human eyes are more sensitive to the surface (interface between the air and the liquid) rather than to the density, temperature, or other scalar fields of the flow. Therefore, different from smoke, rendering of liquid requires formation of the liquid surface, which should be modeled explicitly (with polygonal meshes) or implicitly (with level sets).

Another difference between animation of liquid and of smoke is that the former usually involves boundaries, while the latter is sometimes simulated in an unbounded environment. Programmers have to take care of the boundary conditions in space dimension, and consider interaction between objects and liquid carefully. Moreover, liquid is directly influenced by gravity, whereas smoke particles are so light that gravity might often be ignored.

Unlike traditional free surface approaches, I develop this system using marching cube algorithm to form the liquid surface. Marching cube is preferred since it directly constructs the surface using triangular meshes, which can be easily rendered with conventional computer graphics (either scan line or ray tracing) techniques. Furthermore, users can smooth the mesh with subdivision or deform the surface intuitively by hand. Implicit surfaces might be harder to control and cannot be rendered using scan line methods without being converted to polygonal meshes first.

In Section 2 I will describe some previous research work devoted to animation of liquids. In Section 3 I will introduce Navier Stokes equations, the foundation of my whole project. Section 4 will explain how to solve them numerically with stablity. I will further state in detail how I form the surface of the liquid in Section 5 and discuss the result and statistics in Section 6 and 7.

2. Related Work

About ten years ago, creating fluid animation required laboriously manual inputs by animators. Most physics-based methods either were unstable or only handled 2D cases. Foster and Metaxas [1] successfully used a finite differencing of the Navier Stokes equation and an explicit solver to model the motion of gases in three dimensions. Such an explicit solver suffers from instability when time steps are not small enough.

An implicit solver for the Navier Stokes equations was used by Stam in his paper [2]. He applied a semi-Lagrangian technique for velocity advection as well as a projection method to enforce mass conservation. His approach was stable even with a very large time step, which improves the performance of a fluid simulator considerably. Nevertheless, this scheme results in too much numerical dissipation. Fedkiw et al. later alleviated this problem in Visual Simulation of Smoke [3] by applying "vorticity confinement", which has been noted in the research of fluid mechanics.

About twenty years ago, some methods have been proposed to model ocean waves; Fournier et al. [4] used parametric functions, while Peachy [5] chose sinusoidal phase functions. Kass et al. [6] simulated surface waves in water of varying depth, but did not address rotational and pressure based effects, which are important to animate fluid's characteristic behavior. Chen et al. [7] computed fluid motion in two dimensions, which allowed interaction between moving objects and the flow field, but the objects must be two-dimensional.

Foster et al. [8] later modeled the liquid surface using combination of particles and implicit surfaces. This dynamical level set method managed to avoid both volume loss (when using implicit surfaces alone) and visual artifacts due to insufficient particles. Enright et al. [9] further achieved photorealism using a "thickened" front tracking technique. Ginzburg et al. [10] implemented free-surface in liquid animation using Lattice Boltzmann Model (LBM) in both two and three dimensions, but his method is not robust in complex geometries resulting from intrinsic compressibility. Matthias Muller et. al. [11] recently presents a method to deal with interaction of fluids with deformable solids; however, they focus on the interaction between particles and triangles but do not really construct the liquid isosurface.

3. Navier Stokes Equation

3.1 Representation of the Scene

Before solving the equations for fluid motion, we need to represent the fluid corresponding to the equations. First of all, we have to partition the space into a finite number of grid cells, which constitute the 2D or 3D scene, as shown in Figure 1. The fixed boundary and objects in a scene can be approximated as a subset of these cells if aligned appropriately to the coordinate system we use.

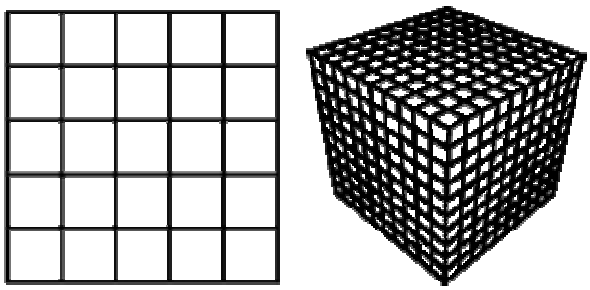


Figure 1.. Partitioning of 2D and 3D space into a finite number of grid cells.

Within each single cell, a density variable and a velocity variable are defined. The density variable stores a scalar value, representing the density of the fluid at this grid, while the velocity variable stores a vector, representing the velocity at the center point of this grid. In fact, we need two sets of them in each cell, one for the current status and the other for the status at

the previous step.

Note that this density value here is for the purpose of visualization of the fluids, and is calculated after the velocity field is solved. This is not directly related to the global density defined in Navier-Stokes Equations. Hence we may replace this "density field" with other scalar properties, for example, temperature.

3.2 The Navier Stokes Equation

The famous Navier Stokes Equations have been extensively studied in the in the field of CFD. Scientists use the equations to model and describe the flow of fluids such as smoke, fire, and liquid.

Given the velocity and pressure for some initial time t = 0 , evolution in the properties of fluids can be described by the Navier-Stokes equations:

$$\nabla \cdot u = 0 \quad \textbf{(1)}$$

$$\frac{\partial u}{\partial t} = -(u \cdot \nabla) \cdot u - \frac{1}{\rho} \nabla \cdot p + \nu \cdot \nabla^2 \cdot u + f \quad \textbf{(2)}$$

where u is the vector of velocity, p is the vector of pressure, ν is the viscosity, ρ is the global density, f is the vector of external force, and ∇ is the vector of partial derivatives in space; that is, $(\partial/\partial x, \partial/\partial y)$ in 2D or $(\partial/\partial x, \partial/\partial y, \partial/\partial z)$ in 3D.

Equation (1) results from conservation of mass, and equation (2) results from conservation of momentum. These equations describe the behavior of incompressible fluid dynamics. Actually, the pressure and velocity fields are related to each other. Therefore, equation (2) can be re-written with merely the velocity field:

$$\frac{\partial u}{\partial t} = \mathrm{P}(-(u \quad \nabla) \cdot u + \nu \cdot \nabla^2 \cdot u + f) \quad \textbf{(3)}$$

where P is a projector that projects any vector onto its divergence free space. In other words, for any vector w, by Helmholtz-Hodge Decomposition,

$w = u + \nabla q$ with $\nabla u = 0$,

where q is a scalar, then $u = Pw = w - \nabla q$

4. Numerical Solver

4.1 Semi-Lagragian Method

Stam [2] solved equation (3) implicitly with the semi-Lagrangian method. At each simulation step, we want to obtain the velocity and scalar fields at time ($t + \Delta t$). This method cuts an entire single step into 4 sub steps.

Let X be the spatial coordinate, representing (x,y) in 2D or (x,y,z) in 3D, and

$$w_0(x) = u(x,t), \quad w_4(x) = u(x, t + \Delta t)$$

A simulation over a period of time is simply a repetition of this process. The first sub-step adds forces onto the field over the time step Δt

$$w_1(x) = w_0(x) + \Delta t \cdot f(x,t)$$

In the liquid case, gravity should be added as external force at this sub-step.

The second sub-step carries out the effect of advection of the fluid on itself. For each cell, it traces (the center point of the cell) backwards in time to find the previous position at the last time step. Then, compute the previous value at that position by linear interpolation of values in its neighboring grids. This previous value determines the new value of this grid cell that we are computing.

$$w_2(x) = w_1(p(x, -\Delta t))$$

where p is the path corresponding to a partial streamline of the velocity field.

The third sub-step is to solve a partial differential equation and models the effect of diffusion. An implicit

method is used here to solve the system, which is sparse and can be solved efficiently.

$$(I - \nu \cdot \Delta t \cdot \nabla^2) w_3(x) = w_2(x)$$

The last sub-step constraints the mass conservation rule. It projects the previous vector onto the divergence free space ($\nabla \cdot \mathbf{u} = 0$).

$$\nabla^2 q = \nabla \quad w_3$$

$$w_4 = w_3 - \nabla q$$

In order to perform the last two sub-steps, I need the multi-grid algorithm to solve for the Poisson equation. The solver I chose is from the routine POIS3D in the library FISHPAK [12], as has been used in Stam's solver. The algorithm of the simulation, hence, looks like the following:

```
UpdateVelocity (U_new, U_old, visc, force, dt) {
      AddForce(U_new, U_old, force, dt);
      swap(U_old, U_new);
      Advect(U_new, U_old, U_old, dt);
      swap(U_old, U_new);
      Diffuse(U_new, U_old, visc, dt);
      ConserveMass(U1, dt); /* projection */
}
UpdateScalar (S_new, S_old, U, diff, source, dt)
{
      /* add source */
      AddForce(S_new, S_old, source, dt);
      swap(S_old, S_new);
      Advect(S_new, S_old, U, dt);
      swap(S_old, S_new);
      Diffuse(S_new, S_old, diff, dt);
}
while (simulating) {
      /* receive user input: F, source, etc. */
      swap(U_old, U_new);
      swap(S_old, S_new);
      UpdateVelocity(U_new, U_old, visc, fource,
      dt);
      UpdateScalar(S_new, S_old, U_new, diff,
      source, dt);
}
```

4.2 Boundary Condition

Foster and Metaxas proposed a clear idea about how to simulate boundaries and fixed objects in a scene, as described in [1]. First we approximate them by a subset of the grids in the scene. After the calculation of velocity fields and scalar fields (which is characterized by "temperature" in [2]) at any step, we simply reset the velocity and scalar value near an object boundary, according to some intuitive rules. That is, set zero to the component of velocity orthogonal to the boundary, and multiply a constant to the component of velocity parallel to the boundary. The constant multiplied clearly depends on the roughness of the boundary surface. Density values near boundary points are not modified, but those inside the boundary points should be kept zero (but it is different for temperature).

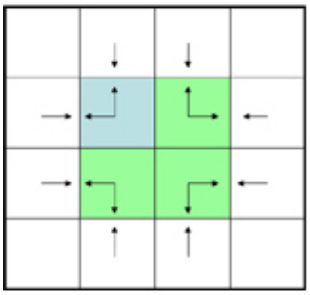


Figure 2. The blue cell is the defined boundary cell, but the effective boundaries also include those green

However, there is a critical difference when they are to be applied in Stam's scheme, since Stam defined the velocities at the center of each cell, whereas Foster and Metaxas defined them at the boundary of the cells. Therefore, when we place a wall (object) in the scene and reset the neighboring velocity fields, it turns out that the walls are effectively two grids thick, as shown

in Figure 2. The reason is that neighboring cells along two sides of the wall have to hold velocities in opposite directions.

5. Surface of Liquids

Since people are always more interested in the surface of liquids instead of density fields, the system has to form the surface to be rendered. While most previous work exploits dynamic level set ([8], [9], and [10]) or particles ([11]), I use marching cube method to directly build up the surface mesh.

5.1 Marching Cubes

The marching cubes algorithm [13] is used to construct an isosurface from a 3D field of values. In this project, the 3D field is just the density field evolved over time.

The 2D analog is illustrated in the Figure 3. For each pixel, set it to black if the value is below some threshold, and set it to white if it is above the threshold. Then smooth the jagged black outlines by skinning them with lines. This 2D case is also called marching squares.

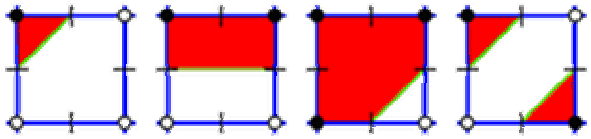


Figure 3. Marching Squres Illustration.

The marching cubes algorithm tests the corner of each cell in the scalar field as being either above or below a given threshold. This yields a collection of boxes with classified corners. Since there are eight corners with two possible states, there are 256 different combinations for each cube. Then, for each cube, we replace the cube with a surface that meets the classification of the cube. The result of the marching cubes algorithm is a piecewise planar surface that approximates the isosurface which has constant density value along a given threshold. We can resolve the original 256 combinations of cell state down to a total of 15 combinations, as described in [13]; with this number it is then easy to create predefined polygon sets for making the appropriate surface approximation.

5.2 Adaptive Marching Cube

We could at any stage apply conventional smoothing (e.g., subdivision) to the mesh, this would of course improve the look but what is more important at this stage is to improve the accuracy of the surface. If we look back at the original 2D test we can see that many of the assumed vertices are a significant distance from the actual surface. Since we know that all the vertices are on an edge between a corner known to be inside the shape and a corner known to be outside the shape we can move the vertex along this edge to the actual surface point. This extension to the algorithm is referred to as "Adaptive Marching Cubes" [14]. Figure 4 compares liquid surface constructed with original and with adaptive marching cubes.

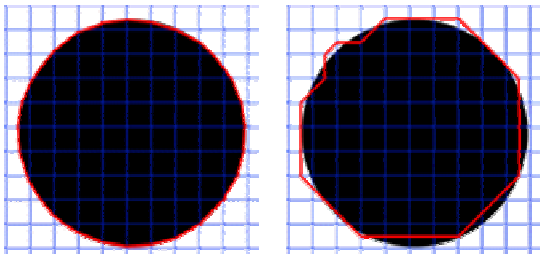
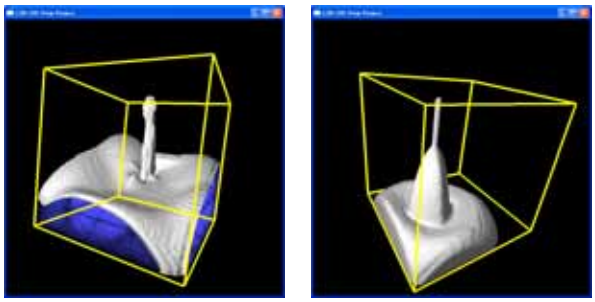


Figure 4. Comparison of Adaptive and original marching cubes

5.3 Surface Close to Container Boundary

In order to create convincingly realistic animation of fluids, when running marching cubes, we need to consider one more layer outside the density field, so that the boundary can be detected close to the wall of the container. Since it takes more triangles to form the boundary surface, computation is more expensive. Figure 5 suggests that we must form this boundary surface whenever possible.



(a) **(b)**

Figure5. (a) No boundary surface. **(b)** With boundary surface

6. Results

For two dimensional fluid simulations, I write an interactive program. Users can add liquid source using mouse right buttons, or apply external forces using left buttons. The system also provides users with an interface to place fixed objects into the scene, in order for liquids to interact with those objects.

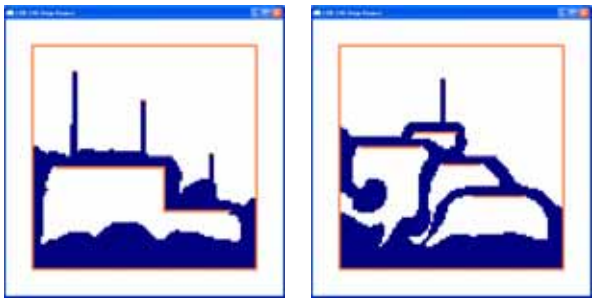


Figure 6. Interactive liquid simulator in 2D

For three dimensional fluid simulations, it is difficult to run the simulation real-time. Therefore I made three clips of videos to demonstrate the system works correctly. In two of the them I do not form boundary surfaces as described in 5.3 since it is time-consuming. To quickly visualize the result, first I render the three-dimensional scene with OpenGL, which does not guarantee excellent quality. If normal vector per face is used, then the result looks piecewise planar. However, using normal vector per vertex smoothes the surface and considerably improves the quality.

As far as rendering is concerned, I use Blue Moon Rendering Tool (BMRT) to carry out ray tracing. The BMRT package is a freeware that implements global illumination with Renderman standard. It is easy to adjust parameters of the material of the liquid surface to simulate different types of liquids.

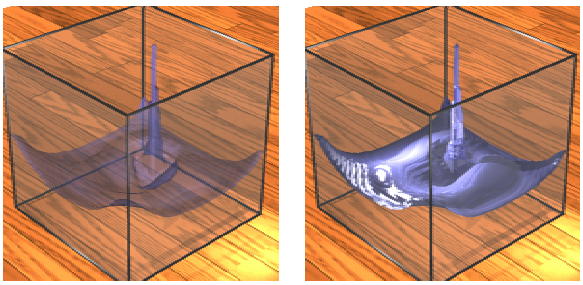
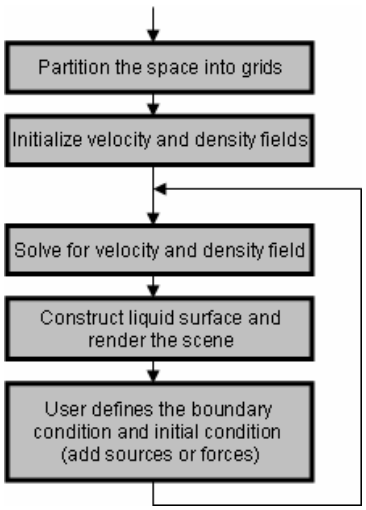


Figure 7. Different rendering styles

7. Analysis

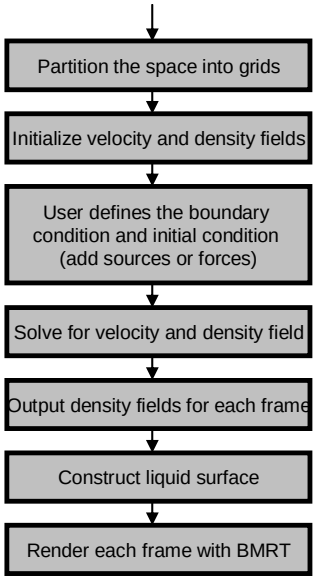
7.1 System Process

The process for simulating fluids in 2D in my program is as follows:



This process can run in real time with N < 100, where N represents the number of grids we subdivide in each dimension. When N becomes larger, the program slows down, due to the O(N^2) algorithm that computes the next-step properties for each cell.

The process for simulating fluids in 3D in my program is as follows:



7.2 Running time

The running time for the last two phases (surface construction and rendering) are related to the number of surface faces, which depend on the density field at each frame. The statistics in the following tables are roughly measured in seconds.

frame	simulation	surface construction	rendering
1st	~10	< 10	4
50th	~10	22	8
100th	~10	25	14
150th	~10	47	17

Table 1. Running time for Video 1

frame	simulation	surface construction	rendering
1st	~10	< 10	4
50th	~10	28	10
100th	~10	45	17
150th	~10	58	25

Table 2. Running time for Video 2

frame	simulation	surface construction	rendering
1st	~10	< 10	4
50th	~10	30	12
100th	~10	43	18
150th	~10	58	22

Table 3. Running time for Video 3

Platform:

Pentium 4 800MHZ

512 MB RAM

NVIDIA Geforce FX5900

8. Conclusion and Future Work

I have satisfactorily completed this project. I solved the Navier Stokes equation on liquids with gravity. Then marching cubes algorithm is used to form the isosurface of the density fields of liquids, and a further smoothing is achieved by computing normal vectors of each vertex. Finally, the three dimensional simulations are rendered out using Renderman standard (BMRT) and video clips are generated.

Here is some special points that I learn in the project:

1. To perform the boundary or object interaction, the resetting of nearby grids must be carried out at each sub-step, instead of only once for the whole step.
2. The Semi-Lagrangian method indeed tends to result in highly viscous fluid. Even if the viscosity term is low and diffuse rate is high, when being looked at carefully, liquids tend to be gluey comparing to the results in [8].

I plan to implement Loop's subdivision in order to further smooth the liquid surface. Sharp features close to boundary surfaces need to be preserved with attention. In addition, I hope to write a volume ray

tracer by myself so that I have full control over the rendering. I also need to try some other methods for liquid animation such as dynamic level set so that I will be able to compare performance of different approaches. Finally, I am interested in the interaction between deformable objects and liquids, which I did not implement in this project. I will try to follow [11] to take it into consideration, and generate some video clips with movie-production quality.

References

[1] Foster, N. and Metaxas, D., "Modeling the Motion of a Hot Turbulent Gas". ACM SIGGRAPH 97.

[2] Stam, J., "Stable Fluids". *In Proceedings of SIGGRAPH 1999.*

[3] Fedkiw, R., Stam, J., and Jensen H. W., "Visual Simulation of Smoke", *In Proceedings of SIGGRAPH 2001*.

[4] Fournier, A., and Reeves, W. T., "A Simple Model of Ocean Waves". *In Computer Graphics (Proceedings of SIGGRAPH 86).*

[5] Peachey, D. R. "Modeling waves and Surf". *In Computer Graphics (Proceedings of SIGGRAPH 86).*

[6] Kass, M., and Miller, G. "Rapid, Stable Fluid Dynamics for Computer Graphics". *In Computer Graphics (Proceedings of SIGGRAPH 90).*

[7] Chen, J., and Lobo, N. "Toward interactive-rate simulation of fluids with moving obstacles using the Navier-Stokes equations." *Computer Graphics and Image Processing 57.*

[8] Foster, N., and Fedkiw, R., "Practical animation of liquids". *In Proceedings of SIGGRAPH 2001*.

[9] Enright, D., Marschner, S., Fedkiw, R., "Animation and Rendering of Complex Water Surfaces", *ACM SIGGRAPH 2003.*

[10] Ginzburg, I., and Steiner, K., "Lattice Boltzmann Model for Free-Surface flow and Its Application to Filling Process in Casting", *Journal of Computational Physics, Volume 185, Issue 1.*

[11] Muller, M., Schirm, S., Teschner, M., Heidelberger, B., and Gross, M., "Interaction of Fluids with Deformable Solids", *in Journal of Computer Animation and Virtual Worlds (CAVW), vol 15, no. 3-4*

[12] SCD's FISHPAK source directory,
ftp://ftp.ucar.edu/dsl/lib/fishpak/

[13] Lorensen, E. W., and Cline, H. E., "Marching Cubes: A High Resolution 3D Surface Construction Algorithm", *In Computer Graphics (Proceedings of SIGGRAPH 87).*

[14] Hall, M., and Warren, J., "Adaptive Polygonalization of Implicitly Defined Surfaces", *IEEE Journal of Computer Graphics and Applications Nov., 1990.*

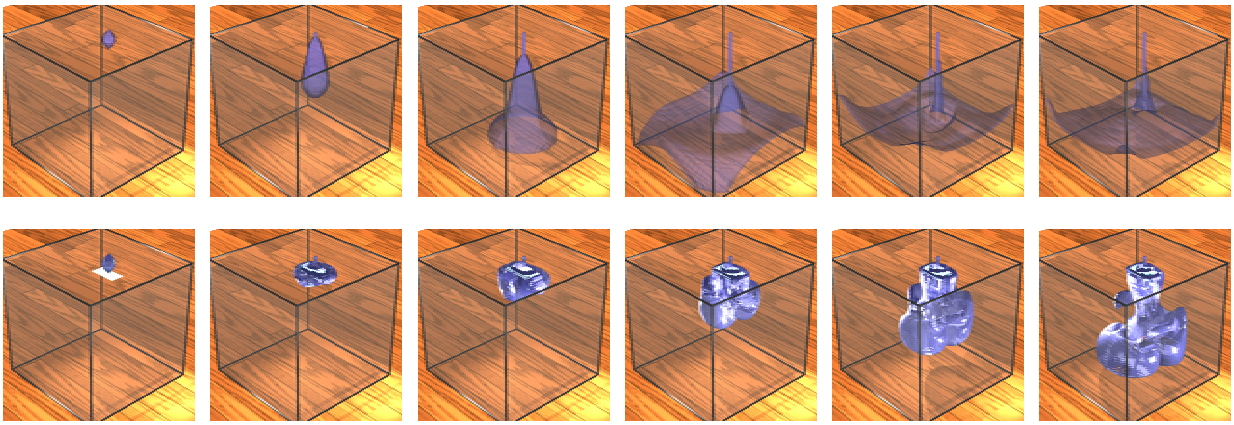


Figure 8. Two image sequences of liquid animation