

Melting and Flowing

Mark Carlson, Peter J. Mucha, R. Brooks Van Horn III, Greg Turk

Georgia Institute of Technology

Abstract

We present a fast and stable system for animating materials that melt, flow, and solidify. Examples of real-world materials that exhibit these phenomena include melting candles, lava flow, the hardening of cement, icicle formation, and limestone deposition. We animate such phenomena by physical simulation of fluids – in particular the incompressible viscous Navier-Stokes equations with free surfaces, treating solid and nearly-solid materials as very high viscosity fluids. The computational method is a modification of the Marker-and-Cell (MAC) algorithm in order to rapidly simulate fluids with variable and arbitrarily high viscosity. This allows the viscosity of the material to change in space and time according to variation in temperature, water content, or any other spatial variable, allowing different locations in the same continuous material to exhibit states ranging from the absolute rigidity or slight bending of hardened wax to the splashing and sloshing of water. We create detailed polygonal models of the fluid by splatting particles into a volumetric grid and we render these models using ray tracing with sub-surface scattering. We demonstrate the method with examples of several viscous materials including melting wax and sand drip castles.

Keywords: melting, solidifying, animation, computational fluid dynamics.

1 Introduction

A major goal in animation research is to simulate the behavior of real-world materials, including such phenomena as draping cloth, breaking glass, and flowing and splashing liquids. The goals of the work presented in this paper are twofold: first, we develop a method for quickly simulating highly viscous liquids with free surfaces; second, we use this capability to animate materials that melt, flow, and harden.

The viscosity of a fluid describes how quickly the variations in the velocity of the fluid are damped out. We find viscous fluids everywhere: toothpaste, hand lotion, yogurt, ketchup, tar, wet cement, and glue are just a few examples. The computer graphics literature contains several methods of simulating fluids with relatively low viscosity. Absent, to our knowledge, is a method for simulating high viscosity liquids. The method that we present fills this gap in computer animation, allowing us to simulate material that varies in both space and time from absolutely rigid (treated as very high viscosity) to freely flowing (low viscosity). We have modified the Marker-and-Cell method from the computational fluid dynamics literature to incorporate an implicit scheme for calculating the diffusion component of the equations for viscous fluids. This implicit

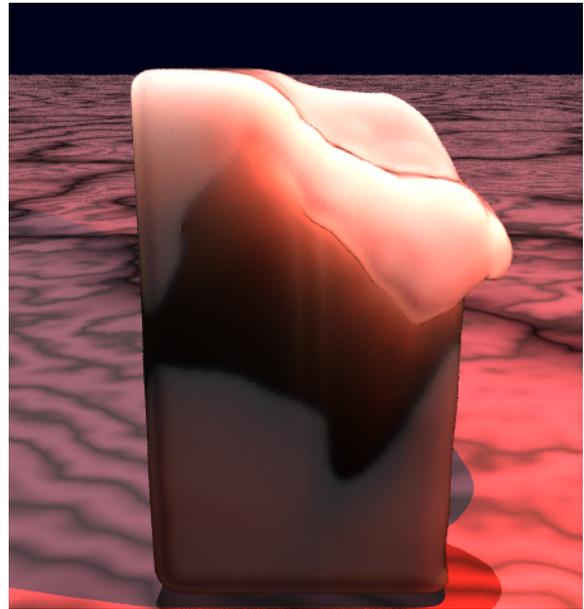


Figure 1: Melting wax.

approach allows us to take large time-steps even when the viscosity of the fluid is extreme.

Many materials exhibit variable viscosity depending on properties such as temperature and water content. Being able to simulate a wide range of viscosities allows us to achieve our second goal, which is to simulate materials that melt, flow, and harden. Many natural materials exhibit these properties, including wax, glass, cement, wet sand, stone (lava), and water (ice). We add several capabilities to the MAC method in order to animate such phenomena. First, we allow the viscosity of the animated material to vary from one position to the next, and we change the material equations of motion to address this variability. Second, we tie material properties such as temperature or water content to the viscosity in order to allow melting and hardening. We simulate heat diffusion, heat sources and heat sinks for the animation of material such as molten wax. Third, we extract surface models from the simulation by splatting particles into a high resolution volume and we then create polygons from this volume. Finally, we use ray tracing with subsurface scattering to render these materials.

The overarching theme of our work is that many materials that melt, flow and harden can be viewed *always* as a fluid, even when in solid form. This prevents us from ever having to set arbitrary thresholds when deciding whether a material should be treated as a solid or as a liquid. The methods that we use to implement this idea borrow from several techniques that have been published in the computer graphics and the computational fluid dynamics literature. We believe that our main contribution is in bringing these separate threads together and in demonstrating that a system built from these components can produce believable animation of melting and solidifying materials. Since the components of our system are based on well-understood numerical techniques, we anticipate that others will easily be able to replicate our results.

2 Previous Work

Animation of fluids have been approached in a number of ways in the computer graphics literature. We use the term *fluids* to encompass the motion of gases such as air (including simulating smoke), and liquids such as water.

Several graphics researchers studied the large-scale motion of water in waves [Fournier86; Peachy86]. These methods used elevation maps of the terrain underneath the water, and the line of waves were bent according to the variations in wave speed that the elevation profile induces. The simulation of breaking waves occurs at a particular sea floor elevation and wave velocity.

Kass and Miller took a different approach to the simulation of fluids [Kass90]. Like most of the earlier approaches, they used a height field to model water. In contrast to other methods, however, they used a partial differential equations formulation for the motion of the water. Their PDE's govern the amount of fluid that passes between columns of water. O'Brien and Hodgins used a hybrid height field and particle-based representation to simulate splashing water [O'Brien95].

Several groups of researchers have used physically-based particle models to represent fluids. Miller and Pearce create solids, deforming objects and fluids by tuning the manner in which particles interacted with one another [Miller89]. Their particle forces are similar to Lennard-Jones forces: particles very close together repel one another, but at moderate distances they are attracted to each other, with the attraction falling off with greater distances. Tonnesen, in addition to calculating inter-particle forces, uses a discrete approximation to the general heat transfer equation in order to modify a particle's behavior according to its thermal energy [Tonnesen91]. A similar approach is used by Terzopoulos et al., but they also allow pairs of particles to be explicitly attached to one another for modeling deformable objects [Terzopoulos89]. Desbrun and Gascuel also use Lennard-Jones style particle forces to create soft materials, but they maintain an explicit blending graph and perform particle size calculations in order to preserve volume [Desbrun95]. Stora et al. use particles and an approach to force calculations called *smoothed particle hydrodynamics* in order to simulate the flow of lava [Stora99]. Their simulator models heat diffusion and variable viscosity, and they demonstrate animations that use up to 3,000 particles.

The most recent trend in the animation of liquid is to discretize the fluid into compact cells, rather than into columns of water, and then use PDE's to drive the motion of fluid between cells. This is the finite differences approach, and it is a commonly used method in the computational fluid dynamics literature. This approach is more computationally expensive than column-of-fluid PDE's, but has the advantage that it captures subtle motion effects that the other methods described above do not. The first use of this CFD approach for graphics was by Foster and Metaxis. In a series of several papers, they demonstrated how the Marker-and-Cell approach of Harlow and Welch [Harlow65] could be used to animate water [Foster96], animate smoke [Foster97a], and could be augmented to control the behavior of animated fluids [Foster97b]. A major strength of their method is that liquid is no longer constrained to be a height field, as demonstrated by their animations of pouring and splashing.

Witting demonstrated a system in which computational fluid dynamics was used in an animation environment [Witting99]. His system allows animators to create and control 2D effects such as water swirling and smoke rising. Witting uses a set of governing equations that includes heat diffusion and thermal buoyancy, and he uses a fourth-order Runge-Kutta finite differencing scheme for solving the equations. Yngve et al. demonstrated the animation of explosions using CFD based on the equations for compressible, viscous flow [Yngve00]. Their method takes care to properly model the shocks along blast wave fronts, and also models the interaction between the fluids and solid objects.

Stam uses a semi-Lagrangian method for fluid convection and an implicit integrator for diffusion so that large time steps can be used for animating smoke with no internal boundaries [Stam99]. He also

uses a projection method to satisfy the zero divergence condition. Fedkiw and Stam improved upon this method using vortex confinement to allow vortices to swirl indefinitely, and by using clamped cubic interpolation to prevent the dissipation of fine features [Fedkiw01]. Their improved technique allows solid boundaries, moving or stationary, but assumes a zero viscosity fluid [Fedkiw01]. Foster and Fedkiw recently re-visited the Marker-and-Cell method, and improved upon it in several ways [Foster01]. First, they replaced the forward Euler convection calculations with a semi-Lagrangian approach for greater stability. Second, and perhaps more important, they introduced use of the level set approach to computer graphics for the purpose of fluid simulations. Their level set approach results in considerably more finely resolved details on the liquid's surface.

We note that the CFD literature contains literally thousands of papers on simulating fluids, and there are a number of approaches such as spectral methods and finite elements that are virtually untried in computer graphics. The factors that guide researchers in selecting fluid simulation methods include: ease of programming, low computational overhead, controllability, the incorporation of obstacles, and (in the case of water and other liquids) the representation of free surfaces. Spectral methods do not easily represent complex boundaries or free surfaces, and finite element methods are computationally expensive. These factors are probably important for the prevalence of finite differences methods used for computer graphics. The Marker-and-Cell method is a finite difference approach; we describe it now and then later examine how it may be modified to handle high viscosity fluids.

3 MAC Method Overview

In this section we describe the equations for fluid motion and describe the MAC method of fluid simulation. In Sections 4 and 5 we will describe our modifications to this basic approach.

Our goal is to simulate incompressible, viscous fluids, and the equations that govern such fluids are the Navier-Stokes equations. In the following equations the vector-valued variable \mathbf{u} represents the velocity of the fluid, and it may be a 2D or 3D vector depending on the dimensionality of our simulations. Pressure will be represented by p , the density of the fluid is ρ (which we always take to be 1), and the kinematic viscosity is ν . Here are the Navier-Stokes equations:

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

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

Equation 1 states that the velocity field has zero divergence everywhere. This simply means that in any small region of fluid, the amount of fluid entering the region is exactly equal to the amount leaving the region. This is conservation of mass for incompressible fluids. Equation 2 describes conservation of momentum, and it has several components. Reading from left to right, it states that the instantaneous change in velocity of the fluid at a given position is the sum of four terms: convection, diffusion, pressure and body forces. The convection term accounts for the direction in which the surrounding fluid pushes a small region of fluid. The momentum diffusion term describes how quickly the fluid damps out variation in the velocity surrounding a given point. The parameter ν is the measure of kinematic viscosity of the fluid, and the higher its value, the faster the velocity variations are damped. For constant viscosity, the ν factors out yielding the more familiar $\nu \nabla^2 \mathbf{u}$ momentum diffusion form. The third term describes how a small parcel of fluid is pushed in a direction from high to low pressure. The final term \mathbf{f} contains the external forces (called *body forces*) such as gravity that act on the fluid.

The Marker-and-Cell method of simulating fluids with free surfaces was originally described by Harlow and Welch in 1965 [Harlow65]. This method allows fluids to be equally well simulated in

2D and 3D, but for clarity of exposition we will assume a 2D environment. Since the MAC approach is described well in several other publications [Welch66; Foster96; Griebel98], we will only give an overview of the method and refer the interested reader to these other publications for details. There are two major components to the MAC method: the cells in which fluid velocity is tracked, and a large collection of particles in the fluid that serve to mark which cells are filled with fluid near the surface (the air/fluid interface). One time-step in a fluid simulation is calculated in several stages. First, the velocity values in the fluid-filled cells are updated according to a forward Euler integration step based on Equation 2. The velocities in the cells are then modified to enforce the zero divergence condition of Equation 1. Next, the particles are moved according to the velocity field. Finally, each cell is marked as being fluid-filled or empty according to whether a given cell contains particles. These steps are repeated for each time-step of the simulation.

The cells of the simulation space are uniformly sized rectangular cells, and for simplicity we will assume square cells with side lengths h . Two variables are recorded for each cell, the velocity and the pressure. The velocity $\mathbf{u} = (u, v)$ is stored in a *staggered grid*, in which the x component of the velocity, u , is stored at the vertical boundaries between cells, and the y velocity component, v , is stored at the horizontal boundaries. The 3D case is similar, in which the three velocity components are stored at the faces of a cell. The pressure p is calculated and stored at the cell centers. The velocity and pressure are the only values that need be stored with the grid for the basic method, although later we will describe per-cell temperature (which then governs viscosity).

To update the velocity after one time-step, the new velocities $u_{i,j}$ and $v_{i,j}$ at each position in the grid are calculated according to a finite difference approximation of Equation 2. As an example of these finite difference calculations for 2D animation, consider a simplification of Equation 2 that only accounts for the momentum diffusion term with constant viscosity:

$$\frac{\partial \mathbf{u}}{\partial t} = \nu \nabla^2 \mathbf{u}. \quad (3)$$

The new x component of the velocity at edge (i, j) , called $u_{i,j}$, after a time-step Δt can be calculated using the central differencing as follows:

$$u_{i,j}^{new} = u_{i,j} + \frac{\nu \Delta t}{h^2} (-4u_{i,j} + u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1}) \quad (4)$$

The above equation assumes that we are dealing with constant viscosity (a restriction that we will relax in Section 5.2). It makes use of the standard central differencing template for the Laplacian $\nabla^2 \mathbf{u}$ of u at the edge (i, j) , which is the sum of the four adjacent values $u_{i\pm 1, j\pm 1}$ minus four times the velocity at the edge $u_{i,j}$. For 3D animations, calculating the Laplacian requires the velocities at the six neighboring locations. A similar expression is used to update the y component of the velocity $v_{i,j}$. Note that as the product $(\nu \Delta t / h^2)$ increases, the influence of the diffusion term over the time step grows stronger, and this has implications for the stability of the solver for viscous fluids, a point which we will return to later. The other terms of Equation 2 are incorporated into this finite difference formulation similarly, and their exact forms may be found in [Welch66; Foster96; Griebel98].

After we have updated the velocities according to these finite difference approximations of Equation 2, the velocities do not necessarily satisfy the zero divergence requirements of Equation 1. We need to find another velocity field that is close to the current one, but that also satisfies the divergence-free condition. One way of doing this is known as relaxation, and this is the method used in the original work of Harlow and Welch [Harlow65] as well as others [Foster96]. We take a different approach, and make use of the Helmholtz-Hodges decomposition that states any vector field can be expressed as the sum of a divergence-free vector field and the

gradient of a scalar field. We find this scalar field (which is in fact the pressure gradient) and subtract it from the velocity field to make the result divergence-free. A nice discussion of this in the graphics literature is by Stam [Stam99], and we follow the approach he gives closely. Finding the pressure can be done by solving a Poisson equation. This linear equation can be solved iteratively by preconditioned conjugate gradient methods [Golub96; Barrett94].

Once the velocity field has been made divergence-free, the particles are updated. Particle positions are floating-point coordinates $\mathbf{p} = (p_x, p_y)$, and an individual particle may lie anywhere within a cell. A particle's position is updated by determining the velocity \mathbf{u} at the particle location (using bilinear interpolation for each velocity component) and then pushing the particle forward according to simple Euler integration: $\mathbf{p}^{new} = \mathbf{p} + \Delta t \mathbf{u}$. Note that more elaborate integration schemes such as fourth-order Runge-Kutta are possible, but we have found this to be unnecessary.

Recall that it is the purpose of the particles to mark which cells contain fluid, and ultimately this information is stored as a per-cell empty/surface/full flag. This flag is determined for each cell after all of the particles have been advected. If a previously empty cell acquires a particle, the velocities at the cell's boundaries acquire the velocity of the particle. In practice, it is only necessary to populate cells with particles near the air/fluid interface, although careless addition or removal of particles can result in visible artifacts. The positions of the particles near the surface give a highly resolved shape to the free surface, much more detailed than the cells alone. This allows the MAC method to create detailed fluid surfaces while using a relatively coarse cell grid.

4 High Viscosity Solver

The MAC method as described above is well-suited to simulating fluids with relatively low viscosity. This approach has become a favorite for computer graphics because of its ability to capture not only surface ripples and waves but full 3D splashes. Unfortunately, as it stands, the MAC method cannot simulate high viscosity fluids without introducing prohibitively many time steps. In order for the algorithm to remain stable, the method must respect a Courant-Friedrichs-Lewy (CFL) condition [Press93] describing the maximum speed with which information can be advected in one time step from a cell to its neighbors. Additionally, the explicit implementation of the MAC method must also obey a stability criterion imposed to prevent numerical instability in the calculation of the momentum diffusion contribution; at high viscosities, this second stability criterion for explicit solvers becomes more stringent than the CFL condition.

When the viscosity ν becomes large, the viscous diffusive part of the time evolution becomes stiff. The finite difference approximation to this contribution, as described for a simple forward Euler step in Equation 4, has eigenvalues between $(1 - 4\nu \Delta t / h^2)$ and 1, in d dimensions, as indicated by a straightforward von Neumann stability analysis [Press93]. Thus, to prevent numerical instability, the time step must remain small enough such that $\nu \Delta t / h^2 < \frac{1}{2d}$, which can become prohibitively small for large viscosity ν . Similarly, since there are no so-called "A-stable" explicit schemes, higher-order explicit time steps (e.g., fourth-order Runge-Kutta) meet with similarly prohibitive stability criteria at large viscosities, at only marginally different threshold values [Trefethen96]. Lowering the time-step size is one possible fix to this problem, but this quickly leads to prohibitively large number of time steps: even moderately viscous fluids with a viscosity of 10 require that 6000 time-steps be taken using forward Euler integration to simulate one second of fluid motion.¹ The required time-steps goes up linearly with viscosity – a viscosity of 100 would require 60,000 time-steps. The approach that we describe below allows fluids with 100 viscosity to be simulated using 30 time-steps per second, so long as the

¹For all the viscosities we quote we use a cell size $h = 0.1$, and our kinematic viscosity has units of $space^2 / time$.

containing or next to air are not represented in the equation. This makes the matrix much smaller than if we were to include every face in the grid. Even though the matrix does not include entries for surface cell faces (air on one side), faces that have an entry in \mathbf{U}^{new} will need the velocity values at those faces. We add these values into the \mathbf{U} vector because otherwise they would cause non-symmetries. In fact, after setting our boundary conditions we can hold constant any value that does not have an entry into the diagonal of the matrix. This allows us to move all the known values over to the \mathbf{U} vector and our matrix becomes symmetric.

4.3 Matrix Solver

Now that we have set up the matrix equation, let us examine the issue of what solver to use. As mentioned earlier, high viscosity fluid would require very small time steps if we use forward Euler integration. The diffusion step can be made stable even with large time steps by reformulating it using implicit backwards Euler integration (though any L-stable [Lambert91] method would be appropriate):

$$\mathbf{U}^{new} = \mathbf{U} + \mathbf{D}\mathbf{U}^{new} \quad (10)$$

If we define a new matrix $\mathbf{A} = \mathbf{I} - \mathbf{D}$, the above equation can be re-written as follows:

$$\mathbf{A}\mathbf{U}^{new} = \mathbf{U} \quad (11)$$

The above Equation 11 can now be solved using standard matrix techniques. After incorporating the boundary conditions described above (Section 4.2), we arrive at a matrix \mathbf{A} that is positive definite, symmetric, and banded. With a large range of viscosities, the condition number of the resulting matrix prevents an effective direct solve, and we solve the equation iteratively using the conjugate gradient method with a Jacobi preconditioner [Barrett94].

Neither operator splitting nor implicit integration are new to computer graphics. Stam [Stam99] used the operator splitting technique so that he could use a semi-Lagrangian method for calculating the convection term of the Navier-Stokes equation, thus making this component of the simulator stable even with very large time-steps. Moreover, Stam uses an implicit Euler integration scheme for calculating diffusion, just as we use method for solving the diffusion term. He used an FFT-based solver for diffusion, and thus the particular solver that he used will not be useful for problems with more complex boundary conditions. In particular, because we wish to capture complex fluid/air interfaces, his velocity diffusion solver cannot be used for our problem domain. Foster et al. used the semi-Lagrangian calculations of convection to make the MAC method stable at large time-steps [Foster01]. They must artificially decrease the fluid's viscosity, however, in order to keep the solver stable. We cannot use this approach since we use very high viscosities to model rigid solid materials. Our own work can be viewed as a marriage of a stable implicit Euler integration for diffusion (introduced to computer graphics by Stam for smoke) and the MAC method for representing liquid boundaries (brought to graphics by Foster and Metaxis). Because of the needs of representing complex boundary conditions, the particular implicit solver that we use to handle diffusion is different than any that we know of in computer graphics.

4.4 Fluid in Free Flight

At very high viscosities, the implicit calculation of the viscous dissipation effects over a time step has the detrimental effect of artificially slowing the motion of fluid that is in free flight. This artificial dissipation originates because the integrations representing the inverse of \mathbf{A} in Equation 11 become numerically ill-conditioned as the viscosity increases. Within this limit, any constant velocity added to a solution will yield an analytically acceptable new solution. At high viscosities the iterative conjugate gradient solution with Jacobi preconditioning converge to a solution with zero momentum. Therefore, an isolated splash of viscous fluid moving with a high velocity can, after one time step, have no velocity at all.

We prevent this non-physical behavior by identifying the connected regions of fluid that are in free flight and re-introduce the momentum that the diffusion solver removes. Specifically, we identify isolated fluid regions (surrounded by empty cells) and we determine the bulk motion of these fluid cells immediately prior to calculating velocity changes due to the momentum diffusion term. The viscous effects, which cannot change such freely-flying motion, are then calculated as described above, after which the bulk motion of the free-flight regions are added back in. The falling fluid in Figure 2 and the drops of sand shown in the video for the drip sand castle (still frame in Figure 4) are two examples in which this velocity re-introduction was necessary. In principle, this bulk motion should include both translational and rotational momenta. We have found, however, that using just the translational part of rigid body motion is sufficient for obtaining realistic-looking results in the animations described here. Adding a full six-degree-of-freedom rigid body simulator would be straightforward, should it be necessary for a particular animation sequence.

5 Heat and Viscosity

In order to simulate materials that melt and harden, it is necessary to vary the viscosity according to properties of the material. In particular, we simulate the temperature changes of the material and we vary the viscosity according to this temperature. Several other graphics researchers have incorporated thermal diffusion and the resulting changes to viscosity into their material models, usually with a particle-based approach [Miller89; Tonnesen91; Terzopoulos89; Stora99]. Incorporating these effects into the MAC framework is straightforward, and we give details of how to do so now.

5.1 Heat Equation

The change in heat is governed by an equation that is very similar to the second Navier-Stokes equation that we saw earlier. The heat diffusion equation that gives the change in temperature \mathbf{T} is:

$$\frac{\partial \mathbf{T}}{\partial t} = k \nabla^2 \mathbf{T} - (\mathbf{u} \cdot \nabla) \mathbf{T}. \quad (12)$$

This equation has two right-hand terms: the diffusion of heat and heat convection. The parameter k is called the *thermal diffusion constant*, and it takes on a small value for those materials that we simulate. Just as with Equation 2, we use operator splitting to solve for changes in temperature. We first use upwind differencing to determine an intermediate temperature due to convection. Then we use an implicit solver that operates on these intermediate values to account for thermal diffusion. We could use the same conjugate gradient solver for heat diffusion as we did for velocity diffusion, but because the thermal diffusion constant k is small we have the luxury of taking a different approach.

To solve for thermal diffusion we perform what is in fact another example of operator splitting. We set up three different matrix equations, each having the form:

$$\begin{bmatrix} \vdots \\ T_{i,j,k}^{new} \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ T_{i,j,k} \\ \vdots \end{bmatrix} + \frac{k\Delta t}{h^2} \begin{bmatrix} \vdots \\ \dots & 1 & -2 & 1 & \dots \\ \vdots \end{bmatrix} \begin{bmatrix} \vdots \\ T_{i-1,j,k} \\ T_{i,j,k} \\ T_{i+1,j,k} \\ \vdots \end{bmatrix} \quad (13)$$

Each of these three equations is set up to calculate diffusion only in a single direction, either x , y , or z . By solving them in sequence and passing each one's results to the next, we are performing three-way operator splitting. This time, however, instead of splitting one large PDE into separate ones, we are splitting the 3D Laplacian operator into three separate one-dimensional Laplacian operators. This will not yield the same exact answer as the full 3D Laplacian, but it gives a close approximation.

Our matrix is symmetric and positive-definite, and tri-diagonal solvers for such matrices are fast. In particular, doing so is substantially faster than using the conjugate gradient solver that we used to solve Equation 11. If our thermal diffusion constant k had been large, we would have been obliged to use the slower conjugate gradient solver to get accurate results. When we solved for velocity diffusion, the analogous material parameter was the viscosity ν , which can be quite high, so we had to use the computationally more expensive solver.

Operator splitting by dimensions is a common technique, and some of the more popular such methods are called alternating-direction implicit (ADI) methods [Press93; Morton94]. We actually use a closely related technique called the locally one-dimensional (LOD) method [Morton94] that is stable in 3D, but ADI techniques that are stable in 3D such as Douglas-Rachford would also be suitable for this task.

5.2 Variable Viscosity from Temperature

Once we have calculated temperature at each cell in the simulation we can use this temperature to determine the material's viscosity. We use a particularly simple relationship between temperature and viscosity: if the temperature is substantially below or above the melting point of the material, we leave the viscosity at a constant value. Within a temperature transition zone, we vary the viscosity as either a linear or quadratic function of temperature. Many materials, including wax, have a rapid transition from high to low viscosity when the material is heated to the melting point. Thus for our simulations of wax we make this transition zone quite narrow. This means that our materials remain rigid if they are cooler than the melting point, and then quickly liquefy at the appropriate temperature. So far as the solver goes, however, we could use almost any relationship between temperature and viscosity that we want. The key to simulating the proper behavior based on viscosity that changes spatially is to use the variable viscosity version of the diffusion term, and we now turn to this issue.

To understand the changes needed to allow variable viscosity, we return to the velocity diffusion equation. For expository purposes we will write these equations for a forward Euler integrator, and the appropriate changes to an implicit form are to be understood. Recall Equation 4 for the momentum diffusion contribution in 2D with constant viscosity:

$$u_{i,j}^{new} = u_{i,j} + \frac{\nu \Delta t}{h^2} (-4u_{i,j} + u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1}) \quad (14)$$

This equation assumes that viscosity ν is the same at all cells, so that this parameter may be placed outside the parenthesis. When viscosity varies across the fluid, viscosity should be considered a property of the boundaries that separate pairs of adjacent cells. Let $\nu_{i-1/2,j}$ represent the viscosity between cell $(i-1, j)$ and cell (i, j) , and $\nu_{i,j-1/2}$ will be viscosity between cell $(i, j-1)$ and cell (i, j) . The correct finite difference formulation, including variable viscosity [Press93], becomes:

$$u_{i,j}^{new} = u_{i,j} + \frac{\Delta t}{h^2} \begin{pmatrix} \nu_{i-1/2,j}(u_{i-1,j} - u_{i,j}) + \\ \nu_{i+1/2,j}(u_{i+1,j} - u_{i,j}) + \\ \nu_{i,j-1/2}(u_{i,j-1} - u_{i,j}) + \\ \nu_{i,j+1/2}(u_{i,j+1} - u_{i,j}) \end{pmatrix} \quad (15)$$

In the above equation each viscosity for the boundary between cells is paired with the difference in velocity between the cells. The resulting matrix equation stays symmetric, and after we make the appropriate changes to an implicit form we can use the same matrix solver as before.

The above viscosities at cell boundaries may be obtained by averaging the viscosities from the two cells separated by the boundary, since the material property controlling viscosity (e.g. temperature) is identified with the cells themselves, not the boundaries. When animating objects that melt, we get poor results when we use an

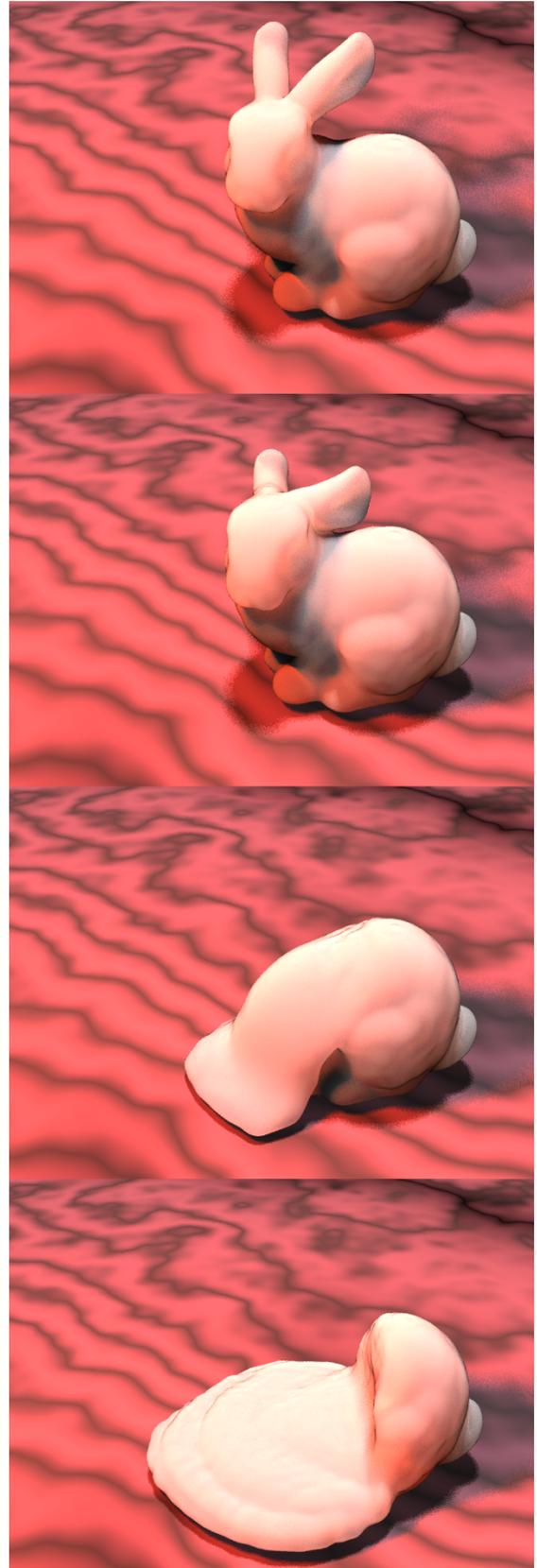


Figure 3: Melting bunny.

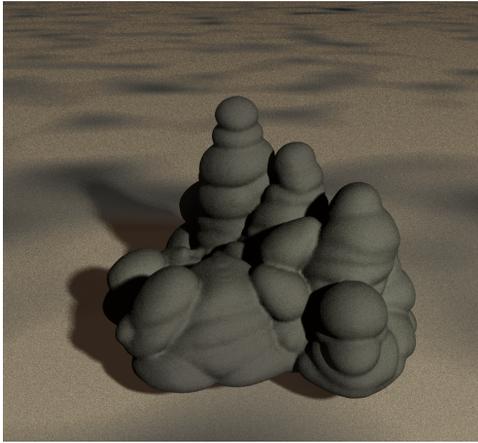


Figure 4: Drip sand castle.

arithmetic average between pairs of viscosities of adjacent cells. The reason for this is that melted material that is dripping down the side of a rigid object will slow down more quickly because arithmetic viscosity averaging is dominated by the very large viscosity of the solid material. Rather than resort to higher grid resolutions, we found that a simple change alleviates this problem: if we use the geometric average instead of the arithmetic average when combining the viscosities of adjacent cells, the lower viscosity dominates and the material continues to flow down the object’s side. Thus we advocate using $v_{i,j+1/2} = \sqrt{v_{i,j}v_{i,j+1}}$ to average the viscosities $v_{i,j}$ and $v_{i,j+1}$ of adjacent cells.

6 Model Creation by Particle Splatting

A vital component of a fluid animation system is to produce 3D models that are suitable for high-quality rendering. As noted by previous researchers, the grid of cells used with the MAC method is not fine enough to produce smooth surfaces for rendering [Foster96; Foster01]. Foster et al. recognized, however, that the particle positions contain high resolution information about the location of the fluid’s surface. They make use of a level set in order to more finely resolve the air/fluid interface, and their method produces wonderfully detailed models [Foster01]. Their level set is calculated using information both from the particle positions and the cell velocities. We, too, make use of particle positions in order to create detailed 3D surfaces, but we take quite a different approach.

We begin by noting that particle advection is quite inexpensive computationally compared to the velocity solver, so we can afford to keep a fairly dense collection of particles at the surface cells. Recall as well that no particles are kept for cells that are far from the fluid surface. Our approach is to create a volumetric model of the fluid that is four times higher resolution in each spatial dimension than the velocity grid. Thus each parent cell $C(i, j, k)$ of the velocity grid corresponds to 64 daughter voxels in the volumetric model $V(i, j, k)$ that we are creating. We adopt the convention that $V(i, j, k) = 1$ represents a fluid filled voxel, a value $V(i, j, k) = 0$ is empty, and intermediate values are used at the air/fluid interface.

We make use of the status of cells in C to create the high resolution volumetric model. If a cell in C is entirely filled with fluid, all 64 of its daughter cells are given a value of 1. If a cell C is on the surface of the fluid, we turn to the particles for higher resolution information. Specifically, we “splat” each particle into the high resolution volume V . For splatting into a 3D grid, we use the three-dimensional version of a tent function (a separable filter) that has a width of 2.5 voxels and with a maximum value of 1. In order to avoid aliasing, it is important to use sub-voxel precision about the location of the particle during splatting. After splatting all of the particles, we clamp all voxels to a value of 1. When all of the cells (both full and surface) have been processed, we low-pass filter

the volumetric grid to smooth away small gaps due to irregularity in particle density. Finally, to create a model for rendering we use an iso-surface extraction method to create polygons from the volume grid. Previous researchers have proposed creating an implicit surface with a Gaussian radial basis function centered at each particle. The splatting and low-pass filtering operations have much the same effect as this, but with a lower computational cost. Since the fluids in our animations move relatively slowly we have found our splatting approach to be quite satisfactory. For higher velocity fluids in which the particle distribution is likely to vary greatly, the level-set approach [Foster01] would produce significantly better results.

We can reverse this process of creating models from simulation data. If we have a particular 3D polygon model that we wish to input to our simulator, we scan convert the model into a voxel grid. It is important that the interior of the model in the voxel grid is filled, not empty. We then sweep through this volume, identifying those groups of $4 \times 4 \times 4$ voxels that are entirely filled. We then make another sweep, writing out each voxel as a particle position. We do not create a particle, however, if the voxel is part of a 64-voxel block that is entirely inside the model. In addition we write out an identifier for each partially or entirely filled 64-voxel block, and these become the fluid cells in the simulation. The bunny of Figure 3 is a model that was brought into the simulator and melted.

7 Rendering

Once the particle and cell data have been used to create a polygonal model of the fluid, we are prepared for rendering. All of the rendered images in this paper were created using a ray tracer that handles translucent materials. Some of the most common viscous fluids are semi-transparent and much of their appearance is due to sub-surface scattering of light. We were inspired by the beautiful images of Jensen et al. from their model of sub-surface scattering in translucent materials [Jensen01]. We follow their approach closely, and we refer the interested reader to their paper for details. We use this method of computing sub-surface scattering to render images of wax. Figures 1 and 3 are images created using this approach.

8 Results

We have used our fluid simulator to create several animations of viscous fluid and materials that melt or harden. Examples can be seen in the figures and the color plates. Figure 1 shows a block of wax that is being melted by a heat source near its upper right corner. Figure 3 shows a similar wax-like simulation, but this time the model is the Stanford Bunny. This example demonstrates that our models may be given detailed geometry. Note that in a single time-step during this animation, one portion is entirely liquid (near the head) while an adjacent part is solid (the tail). Our solver gracefully handles such variations in viscosity.

The snapshots in Figure 2 demonstrate the behavior of fluid over a wide range of viscosity. Each column represents a different viscosity, from left to right: 0.1, 1, 10 and 100. Fluid has been thrown from above into a complex free-slip container that already holds a shallow pool of fluid. (The container walls are not rendered.) This example not only shows the difference between fluids with varying viscosity, but also demonstrates splashing and high velocity fluids.

Figure 4 shows an example of model creation. When very wet sand is dripped down onto the ground by a child at the beach, the drips of sand pile up to form sand castles. To simulate this, a user indicated locations and times for viscous spheres of sand to drop onto a ground plane. Because of the high viscosity, the simulated drops of sand do not melt together to form a large pool, but instead they pile up and retain their individual shapes.

Because the implicit integrator for velocity diffusion is stable even with large time steps, our simulation times are fast. Table 1 shows simulation times for entire animation sequences. The melting bunny, for example, required about .55 seconds per frame of simulation time. This is dramatically faster than what a forward Euler technique would allow.

Animation	frame count	simulation time	viscosity	grid size
Green Liquid 1	300	145	0.1	32 × 32 × 32
Green Liquid 2	300	104	1	32 × 32 × 32
Green Liquid 3	300	94	10	32 × 32 × 32
Green Liquid 4	300	94	100	32 × 32 × 32
Toothpaste	330	108	10,000	42 × 33 × 18
Bunny Melt	600	330	0.1 - 10,000	35 × 28 × 38
Drip Sand	750	397	50,000	48 × 48 × 48

Table 1: Simulation times (in seconds) are for entire animations, not for each frame. Simulations were run on a 2.0 GHz Pentium 4.

9 Conclusion and Future Work

We have presented a technique for simulating and rendering materials that vary in viscosity from absolute rigidity to water-like. One possible way of doing so would be to have an arbitrary threshold between liquid and solid, and to treat these two cases individually. Instead, our approach is to model the range of material behaviors as variations in the viscosity of the material. We feel that this unified treatment of materials is the main contribution of our work to computer animation. The changes needed to implement this approach are straightforward to make to a MAC method fluid simulator, and because of this we believe that others will have no trouble using our approach. Here are the key ideas that we use in our system:

- Stable integration of diffusion for highly viscous fluids
- Re-introduction of damped out free-flight velocity
- Simulation of heat diffusion that is coupled with viscosity
- Creation of detailed models for rendering by particle splatting

Our approach allows us to rapidly animate liquids that are considerably more viscous than previously published graphics methods have allowed. By coupling the viscosity of a material to its temperature, we can animate objects that heat up, melt, flow, and harden.

There are several topics that we are interested in pursuing in the future. One near-term topic is the texturing of models as they deform and flow. The sand texture of Figure 4 does not move with the surface, and we seek a method of making the texture “stick” to the model. Another extension would be to use the level-set method of [Foster01] instead of particle splatting to define the surface of the fluid. Their method will produce higher quality results than splatting for splashing fluids. To allow large time-steps for fast moving fluid, it would also be necessary to use another method for the convection term such as the semi-Lagrangian approach given in [Stam99; Foster01]. Both the level-set method and the semi-Lagrangian convection calculations deal with aspects of fluid simulation that can be decoupled from the diffusion of high viscosity fluids, so such modifications should be straightforward. A more long-term research question is how to allow the cracking of material that has melted and then hardened, perhaps through the inclusion of surface tension or viscoelastic forces. Many materials such as mud and lava crack while they harden, and it would be wonderful to animate this process.

References

Barrett, R., M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine and H. Van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*, SIAM Press, Philadelphia, PA, 1994.

Desbrun, Mathieu, Marie-Paule Gascuel, “Animating Soft Substances with Implicit Surfaces”, *Computer Graphics Proceedings*, Annual Conference Series (SIGGRAPH 95), August 1995, pp. 287–290.

Fedkiw, Ronald, Jos Stam and Henrik Wann Jensen, “Visual Simulation of Smoke”, *Computer Graphics Proceedings*, Annual Conference Series (SIGGRAPH 2001), August 2001, pp. 15–22.

Fournier, Alain and William T. Reeves, “A Simple Model of Ocean Waves,” *Computer Graphics*, Vol. 20, No. 4, (SIGGRAPH 86), August 1986, pp. 75–84.

Foster, Nick and Dimitri Metaxis, “Realistic Animation of Liquids,” *Graphical Models and Image Processing*, Vol. 58, No. 5, 1996, pp. 471–483.

Foster, Nick and Dimitri Metaxis, “Modeling the Motion of a Hot, Turbulent Gas,” *Computer Graphics Proceedings*, Annual Conference Series (SIGGRAPH 97), August 1997, pp. 181–188.

Foster, Nick and Dimitri Metaxis, “Controlling Fluid Animation,” *Computer Graphics International '97*, Kinopolis, Belgium, June 23–27, 1997, pp. 178–188.

Foster, Nick and Ronald Fedkiw, “Practical Animation of Liquids,” *Computer Graphics Proceedings*, Annual Conference Series (SIGGRAPH 2001), August 2001, pp. 23–30.

Griebel, M., T. Dornseifer and T. Neunhoffer, *Numerical Simulation in Fluid Dynamics, a Practical Introduction*, SIAM Press, Philadelphia, PA, 1998.

Golub, Gene H. and Charles F. Van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, Maryland, 1996.

Harlow, F. H. and J. E. Welch, “Numerical Calculation of Time-Dependent Viscous Incompressible Flow of Fluid with a Free Surface,” *The Physics of Fluids*, Vol. 8, 1965, pp. 2182–2189.

Jensen, Henrik Wann, Stephen R. Marschner, Marc Levoy and Pat Hanrahan, “A Practical Model for Subsurface Light Transport,” *Computer Graphics Proceedings*, Annual Conference Series (SIGGRAPH 2001), August 2001, pp. 511–518.

Kass, Michael and Gavin Miller, “Rapid, Stable Fluid Dynamics for Computer Graphics,” *Computer Graphics*, Vol. 24, No. 4 (SIGGRAPH 90), August 1990, pp. 49–57.

Lambert, J. D., *Numerical Methods for Ordinary Differential Systems*, John Wiley & Sons Ltd., West Sussex, 1991.

Miller, Gavin and A. Pearce, “Globular Dynamics: A Connected Particle System for Animating Viscous Fluids,” *Computers and Graphics*, Vol. 13, 1989, pp. 305–309.

Morton, K. W. and D. F. Mayers (editors), *Numerical Solution of Partial Differential Equations*, Cambridge University Press, 1994.

O’Brien, James and Jessica Hodgins, “Dynamic Simulation of Splashing Fluids,” *Computer Animation 95*, 1995, pp. 198–205.

Peachy, Darwyn, “Modeling Waves and Surf,” *Computer Graphics*, Vol. 20, No. 4, (SIGGRAPH 86), August 1986, pp. 65–74.

Press, William H., Brian P. Flannery, Saul A. Teukolsky and William T. Vetterling, *Numerical Recipes in C: The Art of Scientific Computing*, Cambridge University Press, Cambridge, 1993.

Stam, Jos, “Stable Fluids,” *Computer Graphics Proceedings*, Annual Conference Series (SIGGRAPH 99), August 1999, pp. 121–128.

Stora, Dan, Pierre-Olivier Agliati, Marie-Paule Cani, Fabrice Neyret and Jean-Dominique Gascuel, “Animating Lava Flows”, *Graphics Interface '99*, Kingston, Ontario, Canada, June 1999, pp. 203–210.

Terzopoulos, Dimitri, John Platt and Kurt Fleischer, “Heating and Melting Deformable Models (From Goop to Glop),” *Graphics Interface '89*, June 1989, pp. 219–226.

Tonnesen, D., “Modeling Liquids and Solids using Thermal Particles”, *Graphics Interface '91*, Calgary, Canada, June 1991, pp. 255–262.

Trefethen, Lloyd N., *Finite Difference and Spectral Methods for Ordinary and Partial Differential Equations*, unpublished text, 1996, available at <http://web.comlab.ox.ac.uk/oucl/work/nick/trefethen/pdetext.html>.

Welch, J. Eddie, Francis H. Harlow, John P. Shannon and Bart J. Daly, “The MAC Method: A Computational Technique for Solving Viscous, Incompressible, Transient Fluid-Flow Problems Involving Free Surfaces,” Los Alamos Scientific Laboratory of the University of California, Technical Report LA-3425, March 1966, 146 pages.

Witting, Patrick, “Computational Fluid Dynamics in a Traditional Animation Environment,” *Computer Graphics Proceedings*, Annual Conference Series (SIGGRAPH 99), August 1999, pp. 129–136.

Yngve, Gary, James O’Brien and Jessica Hodgins, “Animating Explosions,” *Computer Graphics Proceedings*, Annual Conference Series (SIGGRAPH 2000), July 2000, pp. 29–36.

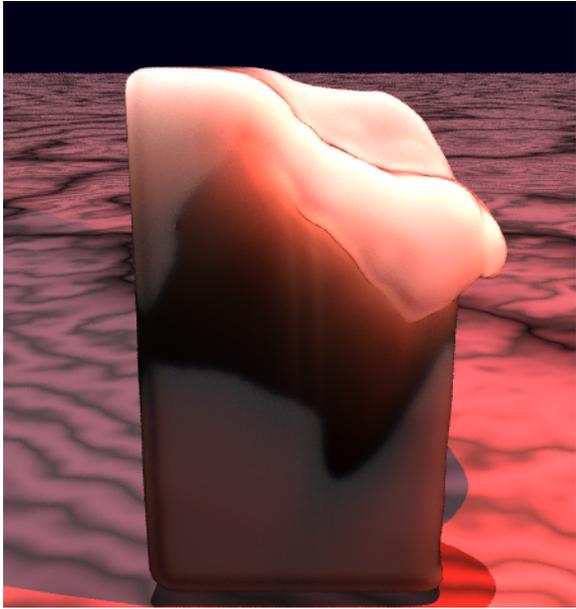


Figure 5: Melting wax.

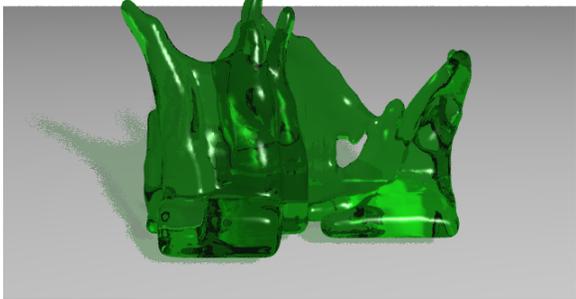


Figure 6: Splashing fluid.

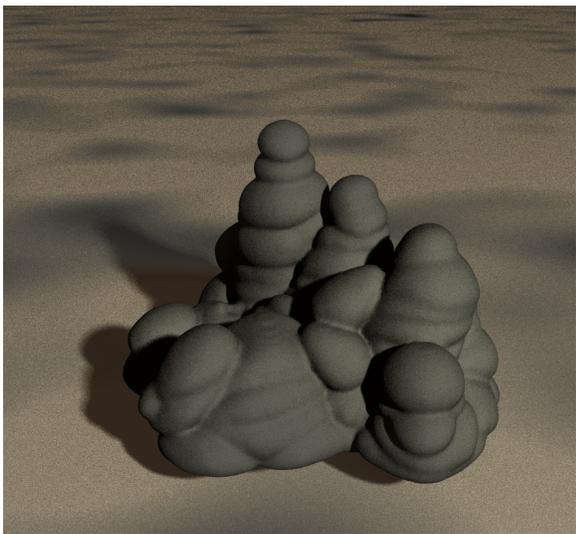


Figure 7: Drip sand castle.

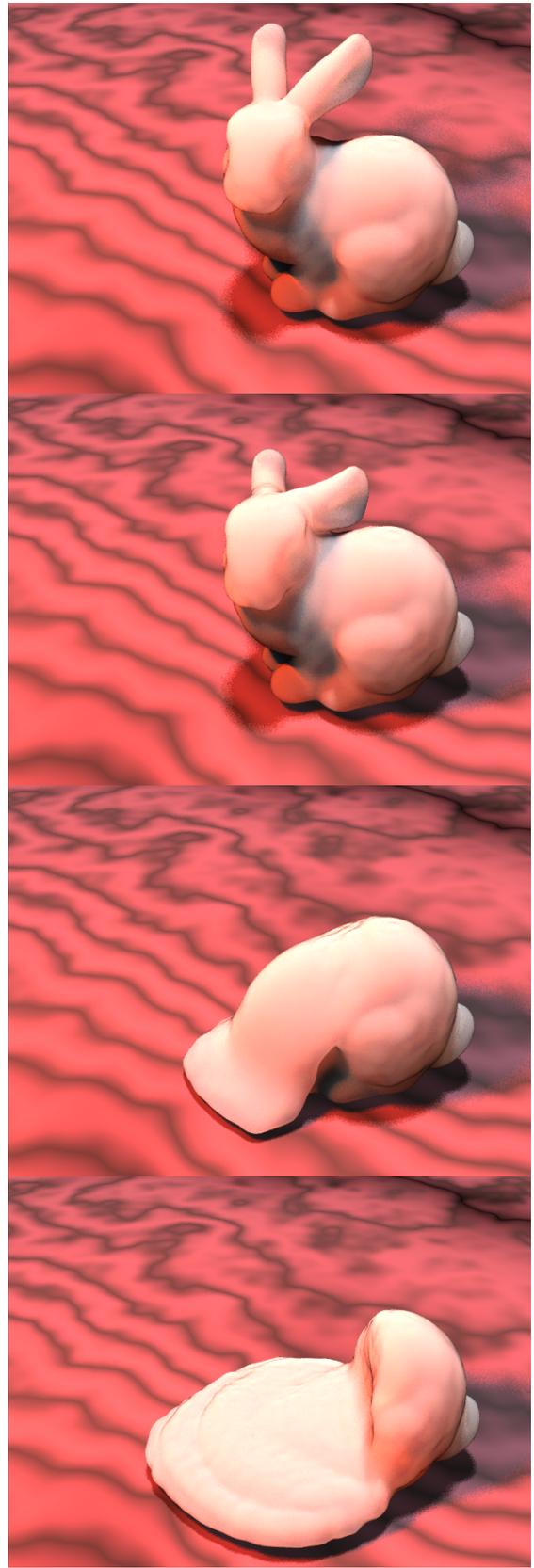


Figure 8: Melting bunny.