8

Water

We now have most of the machinery in place to simulate water as well. Our starting point is treating water as a fluid with a free surface boundary condition where it's in contact with air. The main new ingredient is geometric: the free surface condition allows the water to change shape, and thus we need to track or capture[1] that shape somehow.

Before jumping in, a note on terminology: the air-water surface or boundary is also often called an **interface**. Most of this chapter is about how to work with this interface.

8.1 Marker Particles and Voxels

We'll begin with the simplest possible water simulator. Here we'll use the voxelized pressure solve covered first in Chapter 5, which only required us to classify each grid cell as being fluid (i.e., water), empty (i.e., air), or solid. While the solid cells should be straightforward to identify, determining when to switch a cell to fluid or empty as water moves in or out of it due to advection under the velocity field is the tricky part.

One possibility is to define an initial level set of the water, and advect it using the sharp cubic interpolant. Even with the cubic, however, this tends to have a lot of problems. Water splashing very naturally creates thin structures, and as we noted before, fundamentally a level set on the grid can't reliable handle structures thinner than about two grid cells. Wherever the water thins out, it will tend to soon simply vanish from the grid, as if the water is rapidly evaporating. Droplets rarely can travel more than a few grid cells before disappearing.

This is where we turn instead to **marker particles**, used all the way back in Harlow and Welch's seminal marker-and-cell paper, which also introduced the MAC grid [HW65]. We begin the simulation by emitting

[1] In some numerical contexts, tracking and capturing are technical terms referring to different approaches to solving this problem: tracking loosely corresponds to methods using explicit descriptions of surfaces, such as meshes, whereas capturing refers to methods built on implicit descriptions of surfaces.

water particles to fill the volume of water (presumably defined by a level set) and viewing them as a sampling of the water geometry. If there are sources adding water during the course of the simulation, we emit particles from them as well. Within each advection step we move the particles according to the grid velocity field, so that they naturally follow where the water should be going—using RK2 or better, just like we did for semi-Lagrangian trajectory tracing.[2] Finally, we in turn use the marker particles to mark which grid cells are fluid for the pressure solve: any non-solid cell containing a marker particle is water, and the rest of the non-solid cells are left empty by default.

This algorithm can be justified conceptually by imagining taking it to the limit, where each individual molecule of water is identified with a marker particle: then the set of grid cells containing marker particles is exactly the set of grid cells containing water.

This raises the question of how densely to sample with marker particles —clearly one per molecule isn't practical! Obviously there should be at least one particle per water grid cell, and to avoid gaps in the fluid randomly opening up during advection we probably want at least double that resolution: four particles per cell in 2D and eight per cell in 3D. However, going above double won't necessarily give much improvement: ultimately these particles will be moved by velocities sampled at the grid resolution, which places a limit on how much detail we can expect in the water geometry. If higher-resolution geometry is required, the simulation grid must be refined too.

As we discussed in Chapter 7, it's a good idea to emit particles in a random jittered pattern, not just a regular lattice. This is super important for water marker particles: a shearing flow that compresses along one axis and stretches along another can turn a regular lattice into weird anisotropic stripe-like patterns, far from a good uniform sampling. Thus we require at least jittering the initial grid, as one might do for sampling patterns in rendering.

The first problem apparent with the marker particle approach is in rendering: ultimately we want a smooth surface, describing the boundary between water and air, but right now we only have a mass of points filling the water volume. Clearly we don't want to simply render the water-filled voxels as blocks.

One option is to construct a smooth implicit surface wrapped around the particles. For example, Blinn [Bli82] introduced **blobbies**: given the

[2] Just to remind you, for RK2 and many other integrators, we'll need to look up fluid velocity at locations that might not be inside the current water region, necessitating extrapolation. We'll get to that soon.

positions of the particles $\{\vec{x}_i\}$ define

$$F(\vec{x}) = \sum_i k\left(\frac{\|\vec{x} - \vec{x}_i\|}{h}\right),$$

where k is a suitable smooth kernel function and h is a user parameter intended to be the extent of each particle. A Gaussian might be a reasonable choice for k; a cheaper and simpler alternative would be a spline, such as

$$k(s) = \begin{cases} (1-s^2)^3 : s < 1, \\ 0 : s \geq 1. \end{cases}$$

This spline has the advantage that it depends only on s^2, not s, allowing one to avoid taking a square root when evaluating at $s = \|\vec{x} - \vec{x}_i\|/h$. The extent h should generally be several times the average inter-particle spacing r, for example $h = 3r$, but it can be tweaked as needed. (For our recommended sampling, r is half the grid spacing Δx.) The blobby surface is implicitly defined as the points $\vec{x}$ where $F(\vec{x}) = \tau$ for some threshold τ, or in other words the τ-isocontour of F. A reasonable default for τ is $k(r/h)$, which produces a sphere of radius r for an isolated particle, but this too can be a tweakable parameter.

Unfortunately, the blobby surface can have noticeable artifacts, chief among them that it can look, well, blobby. Many water scenarios include expanses of smooth water; after sampling with particles and then wrapping the blobby surface around the particles, generally bumps for each particle become apparent. This is especially noticeable from specular reflections on the water surface, though it can be masked by foam or spray. The bumps can be smoothed out to some extent by increasing the h-parameter—however, this also smooths out or even eliminates small-scale features we *want* to see in the render. Typically there is a hard trade-off involved.

An improvement on blobbies is given by Zhu and Bridson [ZB05], where instead the implicit surface function is given by

$$\phi(\vec{x}) = \|\vec{x} - \bar{X}\| - \bar{r},$$

where $\bar{X}$ is a weighted average of nearby particle locations:

$$\bar{X} = \frac{\sum_i k\left(\frac{\|\vec{x}-\vec{x_i}\|}{h}\right)\vec{x}_i}{\sum_i k\left(\frac{\|\vec{x}-\vec{x_i}\|}{h}\right)}$$

and $\bar{r}$ is a similar weighted average of nearby particle radii:

$$\bar{r} = \frac{\sum_i k\left(\frac{\|\vec{x}-\vec{x_i}\|}{h}\right) r_i}{\sum_i k\left(\frac{\|\vec{x}-\vec{x_i}\|}{h}\right)}.$$

For now we'll take all the particle radii to be the same r, though better results can be obtained if a particle's radius is actually its distance to the closest point on the desired water surface—see the paper by Adams et al. [APKG07] for an example of how to compute this practically, using a point-based fast marching algorithm similar to Corbett's [Cor05]. Finally, the surface is defined as the points $\vec{x}$ where $\phi(\vec{x}) = 0$, the zero isocontour or level set of ϕ. Once again, for an isolated particle this gives a perfect sphere of radius r. The advantage of this formula over regular blobbies is that it gives somewhat flatter, smoother results where the particles should be sampling smooth geometry, and it is less sensitive to non-uniformities in sampling; however, it does have the disadvantage of potentially introducing small-scale "chaff" in concavities (gaps) in the particle distribution. Sampling this on a grid and doing a small amount of smoothing can be helpful.

A faster and generally quite good option is to simply take a union-of-balls, but sampled on a grid and turned into a level set. This can be done efficiently by computing the distance to the particles on a grid, as in chapter 4, then subtracting the particle radius r (typically just a little less than the grid cell size Δx) from the distance to dilate those points by radius r. The interior is typically far from signed distance at this point, even though the exterior is fine: redistancing is needed where $\phi < 0$. The resulting level set can then be further processed, for example with a smoothing kernel, or a hole-filling step where $\phi_{i,j,k}$ is replaced with the average of its neighbors if and only if the average is less:

$$\phi^{\text{avg}}_{i,j,k} = \frac{\phi^{\text{old}}_{i-1,j,k} + \phi^{\text{old}}_{i+1,j,k} + \phi^{\text{old}}_{i,j-1,k} + \phi^{\text{old}}_{i,j-1,k} + \phi^{\text{old}}_{i,j,k-1} + \phi^{\text{old}}_{i,j,k+1}}{6}$$

$$\phi^{\text{new}}_{i,j,k} = \begin{cases} \phi^{\text{avg}}_{i,j,k} & : \quad \phi^{\text{avg}}_{i,j,k} < \phi^{\text{old}}_{i,j,k} \\ \phi^{\text{old}}_{i,j,k} & : \quad \text{otherwise.} \end{cases}$$

This causes small holes to be filled as if shrink-wrapped, but never erodes away the surface: its limit behaviour if you iterate for a long time is to reconstruct the convex hull of the original level set, so one or two iterations could be considered a "local" convex hull operation.

While a level set of the water—created either by sampling a blobby function or using the union-of-balls construction—can be directly raytraced, it is very common to then construct a mesh approximating the surface as we discussed in Chapter 4, and potentially run further mesh fairing algorithms to improve the quality of the result, or "shrink" it with averaging to get closer to the underlying water particles. These meshes can then be rendered very effectively.

One exciting direction taken in recent years is to skip the level set construction and directly use a triangle mesh to track the surface of the

water. While advecting a mesh through a velocity field is at first glance no harder than advecting particles—just move the vertices like particles—there are several problems that arise. First, the flow can deform the mesh significantly, stretching some triangles to be too big or compressing others to be too small, so remeshing operations are necessary, subdividing or collapsing edges if they are too big or small respectively. Second, when the water pinches off to separate a drop, or two parts of water are supposed to merge, those topology change operations have to be carried out in the mesh connectivity. Finally, numerical errors can cause the mesh to collide with itself, self-intersecting when it is not supposed to. Resolving all these issues robustly and efficiently is hard! There are several approaches to try [BB06,BB09,M̈09,BBB10,WTGT10,WMFB11,DBG14]. Their promise is providing much higher quality tracking of small features, like thin sheets and delicated ripples, but they haven't yet been as "battle tested" as marker particle methods, so I won't go further here.

Getting back to marker particles: if we are using particles to track where the water is, we may as well get the full benefit from them and use FLIP instead of semi-Lagrangian methods for velocity advection. This leads to a simulation step something like this:

1. From the particles, construct the level set for the liquid.
2. Transfer velocity (and any other additional state) from particles to the grid, and extrapolate from known grid values to at least one grid cell around the fluid, giving a preliminary velocity field $\vec{u}^*$.
3. Add body forces such as gravity or artistic controls to the velocity field.
4. Construct solid level sets and solid velocity fields.
5. Solve for and apply pressure to get a divergence-free velocity $\vec{u}^{n+1}$ that respects the solid boundaries.
6. Update particle velocities by interpolating the grid update $\vec{u}^{n+1} - \vec{u}^*$ to add to the existing particle velocities (FLIP), or simply interpolating the grid velocity (PIC), or a mix thereof.
7. Advect the particles through the divergence-free velocity field $\vec{u}^{n+1}$.

8.2 More Accurate Pressure Solves

The methods we have developed so far still fall short when it comes to visual plausibility. The culprit is the voxelized treatment of the free surface boundary condition $p = 0$. Even if we can track and render an accurate water surface, so far the core of the simulation—the pressure solve—only sees a block voxelized surface, with some cells marked as water and some as air. Thus the velocity field, and from there the motion and shape of the

surface itself, cannot avoid significant voxel artifacts. For example, small "ripples" less than a grid cell high do not show up at all in the pressure solve, and thus they aren't evolved correctly but rather persist statically in the form of a strange displacement texture. Somehow we need to inform the pressure solve about the location of the water-air interface within each grid cell. More precisely, we are going to modify how we compute the gradient of pressure near the water-air interface for updating velocities, which naturally will also change the matrix in the pressure equations.[3]

The standard solution is to use the ghost fluid method, as laid out by Gibou et al. [GFCK02]. We'll illustrate this by looking at the update to $u_{i+1/2,j,k}$, which in the interior of the water would be

$$u^{n+1}_{i+1/2,j,k} = u_{i+1/2,j,k} - \frac{\Delta t}{\rho_{i+1/2,j,k}} \frac{p_{i+1,j,k} - p_{i,j,k}}{\Delta x}.$$

Suppose further that (i,j,k) is in the water, i.e., $\phi_{i,j,k} \leq 0$, and that $(i+1,j,k)$ is in the air, i.e., $\phi_{i+1,j,k} > 0$ (treating the case where it's the other way around will be obvious). The simple solver before then set $p_{i+1,j,k} = 0$. However, it would be more accurate to say that $p = 0$ at the water-air interface, which is somewhere between (i,j,k) and $(i+1,j,k)$. Linearly interpolating between $\phi_{i,j,k}$ and $\phi_{i+1,j,k}$ gives the location of the interface at $(i + \theta\Delta x, j, k)$ where

$$\theta = \frac{\phi_{i,j,k}}{\phi_{i,j,k} - \phi_{i+1,j,k}}.$$

Linearly interpolating between the real pressure $p_{i,j,k}$ and a fictional "ghost" pressure $p^G_{i+1,j,k}$, then setting it equal to zero at the interface, gives

$$\begin{aligned}(1-\theta)p_{i,j,k} + \theta p^G_{i+1,j,k} &= 0 \\ \Rightarrow \qquad p^G_{i+1,j,k} &= -\frac{1-\theta}{\theta} p_{i,j,k} \\ &= \frac{\phi_{i+1,j,k}}{\phi_{i,j,k}} p_{i,j,k}.\end{aligned}$$

Now plug this into the velocity update:

$$\begin{aligned}u^{n+1}_{i+1/2,j,k} &= u_{i+1/2,j,k} - \frac{\Delta t}{\rho} \frac{\frac{\phi_{i+1,j,k}}{\phi_{i,j,k}} p_{i,j,k} - p_{i,j,k}}{\Delta x} \\ &= u_{i+1/2,j,k} - \frac{\Delta t}{\rho} \frac{\phi_{i+1,j,k} - \phi_{i,j,k}}{\phi_{i,j,k}} \frac{p_{i,j,k}}{\Delta x}.\end{aligned}$$

[3]Note that this is quite orthogonal to our previously covered accurate finite volume approach to solid wall boundaries, where we use face area fractions to get better estimates of fluid flow in and out of cells, independent of how the pressure updates velocity.

The end effect in the matrix is to increase the diagonal. At this point you should probably get worried about the case where $\phi_{i,j,k} = 0$ or is very close to zero—we are dividing by this quantity! This is a "safe" case, however, in that boosting the diagonal of the matrix makes it simpler to solve and (after Incomplete Cholesky preconditioning) better conditioned. Still, to limit the possibility of numerical strangeness it is still wise to limit the coefficient in the pressure update $(\phi_{i+1,j,k} - \phi_{i,j,k})/\phi_{i,j,k}$ and matrix to some maximum value, say 10^3.

A closely related solution, more in line with the volume fraction approach we've already used extensively, is to instead modify the fluid density $\rho_{i+1/2,j,k}$ to account for the fractions of the u-cell occupied by water versus air. Compute $\rho_{i+1/2,j,k}$ as the average fluid density in the u-cell, ignoring solids. The ghost fluid method above is in fact equivalent to using $\rho_{\text{air}} = 0$ and estimating the volume fraction from just $\phi_{i,j,k}$ and $\phi_{i+1,j,k}$. We can in fact track the velocity field in the air, or use a smooth extrapolation of the water velocity at each step, and solve for pressure in both water and air using the true non-zero density of air. This leads to a full two-phase flow simulator, though getting all the details right w.r.t. advection, level set estimation, etc. requires a lot more work: see Boyd and Bridson's article [BB12], for example, for an approach to extending FLIP to water and air mixes. Solving for pressure throughout water and air gives a much harder linear system, but it does mean that the implied velocity field in the air will be divergence-free so, in particular, air bubbles will be incompressible, conserving their volume instead of collapsing as they do with the usual $p = 0$ free surface condition. In fact, one can go even further and model the thermodynamics governing air compressibility in bubbles, as Patkar et al. showed [PAKF13].

8.3 Topology Change and Wall Separation

One of the trickiest parts of free surface flow, at least from a theoretical perspective, is how separation occurs. Assuming the water starts off as a path-connected component (meaning any two points in the water can be connected by a continuous path going only in the water region), and is advected only in a continuous velocity field, then it must remain path-connected. In particular, a drop cannot separate from the rest of the water: it will always remain connected by an ever-thinner tendril.

With level set and marker particle methods, it turns out this is not a problem in practice: droplets separate quite naturally. However, the theory indicates to us this is merely numerical error in fact, due to us only sampling the liquid's presence at a finite set of grid points and/or particles. Unfortunately, there is no simple resolution to this problem,

as the actual physics of topology change (separation, merging) is not yet properly modeled at the continuum level. We will leave this, therefore, as a problem to worry about in the middle of a sleepless night.

It is also worth pointing out, on the subject of topology change, that water merging can cause some trouble too. At low speeds, more specifically low velocities relative to the grid size divided by time step, two water components merge the instant they come within one grid cell of each other, due to our level set approach to describing the geometry to the pressure solve. In splashing scenarios, this may even cause appreciable volume gain. However, at high speeds (high velocities relative to grid cell divided by time step) we can have the opposite problem: a water drop can penetrate quite deeply into a solid wall or into another water region during advection, effectively losing volume in the process.

Another related issue is how liquid can separate from solid walls. In fact, exactly what happens at the **moving contact line** where air, water, and solid meet is again not fully understood. Complicated physical chemistry, including the influence of past wetting (leaving a nearly invisible film of water on a solid), is at play. What is clear is that our "no-stick" boundary condition $\vec{u} \cdot \hat{n} = \vec{u}_{\text{solid}} \cdot \hat{n}$ both prevents water from penetrating into a solid as well as *separating* from the surface of a solid. However, there is no denying we can observe water separating from surfaces, albeit usually leaving droplets or a wet film behind. Again, in practice using marker particles we find that numerical errors allow water to separate fairly naturally, while also modeling cohesion of water to solids that looks quite reasonable at small scales. However, the time of separation does depend critically and unphysically on grid size and time step size, and at large scales the degree of cohesion can look unnatural. Foster and Fedkiw [FF01] proposed a simple unsticking trick that only enforces $\vec{u} \cdot \hat{n} \geq \vec{u}_{\text{solid}} \cdot \hat{n}$, which unfortunately fails in several common scenarios; Batty et al. [BBB07] later demonstrated a more robust, physically consistent, though expensive treatment inspired by contact problems, extending the inequality constraint into a complementary condition with pressure (so pressure cannot exert more than a critical suction at solid walls). Chentanez and Müller illustrate how this condition can be efficiently solved with multigrid [CMF12].

8.4 Volume Control

Between iterative solves for pressure, truncation error in level set representations and errors in advection, as well as the topology change issues noted above, it is not surprising that a typical water simulation doesn't exactly conserve the volume of water it begins with. Typically the errors are biased to steadily lose volume, but noticeable volume gain can occur too. Kim et

al. [KLL+07] demonstrate a simple correction to this drift by enforcing a non-zero divergence control in the pressure projection step, similar to how we can enforce expansion when heating up smoke.

8.5 Surface Tension

Another subject we'll only briefly touch on is adding surface-tension forces for small-scale water phenomena. This is, it should be warned, a subject of ongoing research both within graphics and scientific computing. For example, the test case of running a sphere of water at rest in zero-G with surface tension (which should result in surface-tension forces exactly canceling pressure forces, so velocity remains zero) is surprisingly hard to get right.

The physical chemistry of surface tension is conceptually simple. Water molecules are more attracted to other water molecules than to air molecules, and vice versa. Thus, water molecules near the surface tend to be pulled in towards the rest of the water molecules and vice versa. In a way they are seeking to minimize the area exposed to the other fluid, bunching up around their own type. The simplest linear model of surface tension can in fact be phrased in terms of a potential energy equal to a surface-tension coefficient γ times the surface area between the two fluids (γ for water and air at normal conditions is approximately $0.073 J/\mathrm{m}^2$). The force seeks to minimize the surface area.

The surface area of the fluid is simply the integral of 1 on the boundary:

$$A = \iint_{\partial\Omega} 1.$$

Remembering our signed distance properties, this is the same as

$$A = \iint_{\partial\Omega} \nabla\phi \cdot \hat{n}.$$

Now we use the divergence theorem in reverse to turn this into a volume integral:

$$A = \iiint_{\Omega} \nabla \cdot \nabla\phi$$

Consider a virtual infinitesimal displacement of the surface, δx. This changes the volume integral by adding or subtracting infinitesimal amounts of the integrand along the boundary. The resulting infinitesimal change in surface area is

$$\delta A = \iint_{\partial\Omega} (\nabla \cdot \nabla\phi)\delta x \cdot \hat{n}.$$

Thus, the variational derivative of surface area is $(\nabla \cdot \nabla\phi)\hat{n}$. Our surface-tension force is proportional to this, and since it is in the normal direction we can think of it in terms of a pressure jump at the air-water interface (as pressure only applies in the normal direction). Since air pressure is zero in our free surface model, we have that the pressure at the surface of the water is

$$p = \gamma \nabla \cdot \nabla\phi.$$

It turns out that $\kappa = \nabla \cdot \nabla\phi$ is termed the **mean curvature** of the surface, a well-studied geometric quantity that measures how curved a surface is.

This property has been incorporated into the ghost fluid pressure discretization by Kang et al. [KFL00]. Here, we take the ghost pressures in the air so that we linearly interpolate to $\gamma\kappa$ at the point where $\phi = 0$, rather than interpolating to zero. The mean curvature κ can easily be estimated at that point by using the standard central difference for the Laplacian, on trilinearly interpolated values of ϕ. Though not immediately apparent, there is also a fairly severe stability restriction on the time step, $\Delta t \leq O(\Delta x^{3/2}\sqrt{\rho/\gamma})$, since this is essentially an explicit treatment of the surface-tension forces. Things are also rather more complicated at triple junctions between water, solid, and air; we refer to Wang et al. [WMT05] for more on this.

To ease the severe time step restriction, two interesting possibilities are out there. One is to make the surface tension treatment more implicit in time, solving for fluid motion which simultaneously will be incompressible and act to minimize surface area (in balance with inertia). With a tetrahedral mesh discretizing the fluid, Misztal et al. have shown the power of this approach [MEB+12], but it remains open how best to incorporate this idea into a more conventional water solver. Alternatively, Sussman and Ohta [SO09] have demonstrated a more stable evaluation of the surface tension force compared to using a finite difference estimate of curvature: essentially they run a smoothing PDE on the level set for a certain amount of time, and measure the difference in level set values to arrive at a more stable estimate of curvature.

9

Fire

This is a short chapter: while the physics and chemistry of combustion can be extraordinarily complicated, and to this day aren't fully understood, we will boil it down to a very simplified model. Our two main sources in graphics are the papers by Nguyen et al. [NFJ02] for thin flames, and those by Melek and Keyser [MK02] and Feldman et al. [FOA03] for volumetric combustion. There are of course many other approaches that avoid fluid simulation and directly model flames procedurally, but we will not cover them in this book.

Combustion is simply a chemical reaction[1] triggered by heat where an oxidizer (like oxygen gas) and a fuel (like propane) combine to form a variety of products, giving out more heat in the process. Thus, at a minimum, our fluid solver will need to be able to track fuel/oxidizer versus burnt products along with temperature—and if it's not a clean flame, we also need smoke concentration to track the resulting soot. In the following sections we'll take a look at two strategies for tracking the fuel/oxidizer appropriate for different classes of combustion.

Just to clarify before proceeding: throughout this chapter we are assuming that both fuel and oxidizer are either themselves gases, or suspended in the air. If you analyze even a wood fire or a candle, you'll find that the flames are the result of gaseous fuel—pyrolyzed from the wood as it heats up, or evaporated wax—not the solid itself directly. Thus when we model a solid (or even liquid) object that is on fire, we treat it as a source emitting gaseous fuel, which then burns. Part of the emission is enforcing a velocity boundary condition $\vec{u} \cdot \hat{n} = \vec{u}_{\text{solid}} \cdot \hat{n} + u_{\text{emission}}$ similar to the usual moving solid wall boundary condition: gaseous fuel is injected into the grid at this relative velocity u_{emission}. In addition, fields describing the presence of fuel (see the following sections) should have the appropriate boundary condition for advection. Various authors have looked at further

[1] One of the complexities hidden here is that in most situations in reality, there are actually hundreds of different chemical reactions in play, with a multitude of different chemical species. For example, we all know that the gasoline burned in a car engine doesn't simply combine with the oxygen in the air to produce water and carbon dioxide: a vast range of other chemicals from carbon monoxide to nitrous oxide to all sorts of carbon structures in soot particles end up in the exhaust.

eroding away the solid or liquid source as it emits gaseous fuel, which is particularly important for thin objects like sheets of paper—see for example the articles by Melek and Keyser [MK03] and Losasso et al. [LIGF06]. Whether or not (and where) a solid is emitting gaseous fuel is usually directly specified by the animator, often modulated by an animated texture to produce more interesting effects, though procedural models simulating the spread of fire according to burn rates or temperature thresholds can also easily be concocted.

9.1 Thin Flames

Nguyen et al. [NFJ02] detail an approach to fire, in which the region where combustion takes place is modeled as an infinitely thin flame front, i.e., a surface, not a volume. In addition, it's assumed that fuel and oxidizer are **premixed** before ignition, as in blow torches—while not really true for other phenomena where the mixing of fuel and oxidizer is an integral part of the fire (technically known as **diffuse flames**), the simplifying premixed assumption can still serve just fine for a visually plausible result.

The flame front divides the fluid region into two parts: premixed fuel/oxidizer on one side and burnt products (and/or background air) on the other side. To track the flame surface we model it with a level set ϕ sampled on the grid, as we did for water in Chapter 8.

The first problem to address is how to evolve the flame front: at what speed does it move? If no combustion were taking place, the flame front would be nothing more than the material interface between the two fluids—just like the free surface in water simulation. Thus our first velocity term is simply the fluid velocity: the flame front is advected along with the flow. For definiteness (this will be important in a moment) we'll use the velocity of the unburnt fuel, $\vec{u}_{\text{fuel}}$. However, when combustion is taking place, the flame front also is "moving:" fuel at the flame surface gets combusted into burnt products, effectively shrinking the surface inwards. The simplest model for this is to assume a constant burn speed S, the rate at which the flame surface burns into the fuel region along the normal direction. Nguyen et al. [NFJ02] suggest a default value of $S = 0.5m/s$. Assuming that ϕ is negative in the fuel region by convention, this gives our level set equation

$$\frac{\partial \phi}{\partial t} + \vec{u}_{\text{fuel}} \cdot \nabla \phi = S. \tag{9.1}$$

With $S \neq 0$, note that the volume of the fuel region is *not* conserved, and thus a lot of the worries for tracking level sets for water don't bother us here—just the usual advection approaches can be used as if ϕ were any other scalar, along with an addition of $S\Delta t$ each time step, and perhaps periodic

reinitialization to signed distance. Note also that if an object is supposed to be on fire, a boundary condition forcing $\phi \leq 0$ on the burning sections of its surface should be included; otherwise ϕ should be extrapolated into solids to avoid artifacts near the surface, as with liquids.

As an aside, Hong et al. [HSF07] have more recently added additional detail to this technique by using higher-order non-linear equations for the speed and acceleration of the flame front (rather than keeping it as a constant burn speed S). These promote the formation of "cellular patterns" characteristic of some fires.

The next problem to address is how the fire affects the velocity field. If the density of the burnt products is less than the density of the fuel mix (as is often the case, either due to differences in a specific gas constant or temperature), then conservation of mass demands that the fluid should instantaneously expand when going through the flame front from the fuel region to the burnt-products region. The rate of mass leaving the fuel region per unit area is $\rho_{\text{fuel}}S$, which must match the rate of mass entering the burnt region per unit area, $\rho_{\text{burnt}}(S + \Delta V)$, where ΔV is the jump in the normal component of fluid velocity across the interface. Solving gives

$$\Delta V = \left(\frac{\rho_{\text{fuel}}}{\rho_{\text{burnt}}} - 1 \right) S.$$

Since the tangential component of velocity is continuous across the flame, this means the full velocity satisfies the following jump at the flame front:

$$\begin{aligned} \vec{u}_{\text{burnt}} &= \vec{u}_{\text{fuel}} + \Delta V \hat{n} \\ &= \vec{u}_{\text{fuel}} + \left(\frac{\rho_{\text{fuel}}}{\rho_{\text{burnt}}} - 1 \right) S\hat{n}, \end{aligned} \tag{9.2}$$

where $\hat{n}$ is the normal pointing outward from the fuel region into the burnt region. This naturally defines a **ghost velocity** for use in advection: if you trace through the flame front and want to interpolate velocity on the other side, you need to add or subtract this change in normal velocity for the advection to make sense (where normal, in this case, might be evaluated from the normalized gradient of the level set ϕ wherever you happen to be interpolating). This is one example of the **ghost fluid method** developed by Fedkiw and coauthors.

This expansion also has to be modeled in the pressure projection step, enforcing a non-zero divergence at the flame front—indeed, this is one of the critical visual qualities of fire. The simplest discrete approach is to build Equation (9.2) into the evaluation of divergence that defines the right-hand side of the pressure equation. For example, when evaluating the divergence at a cell where $\phi \leq 0$ (i.e., in the fuel region) then any of the surrounding u-, v-, or w-velocity samples, where the averaged $\phi > 0$ (i.e., in the burnt

region) should be corrected by the x-, y- or z-component of $-\Delta V\hat{n}$, respectively. Similarly if $\phi > 0$ at the cell center, then, at surrounding velocity samples where the averaged $\phi \leq 0$, ϕ should be corrected by the appropriate components of $+\Delta V\hat{n}$. This can be interpreted as yet another use for divergence controls. When setting up the matrix in the pressure solve, greater fidelity can be obtained by accounting for the different densities of fuel and burnt products: this shows up as variable densities just as in water or the advanced smoke model discussed earlier in the book. We can estimate the density in, say, a u-cell $(i+1/2, j, k)$ as a weighted average of ρ_{fuel} and ρ_{burnt}:

$$\rho_{i+1/2,j,k} = \alpha\rho_{\text{fuel}} + (1-\alpha)\rho_{\text{burnt}},$$

where the weight α could be determined from the level set values $\phi_{i,j,k}$ and $\phi_{i+1,j,k}$:

$$\alpha = \begin{cases} 1: & \phi_{i,j,k} \leq 0 \text{ and } \phi_{i+1,j,k} \leq 0, \\ \dfrac{\phi_{i,j,k}}{\phi_{i,j,k} - \phi_{i+1,j,k}}: & \phi_{i,j,k} \leq 0 \text{ and } \phi i+1, j, k > 0, \\ 1 - \dfrac{\phi_{i,j,k}}{\phi_{i,j,k} - \phi_{i+1,j,k}}: & \phi_{i,j,k} > 0 \text{ and } \phi i+1, j, k \leq 0, \\ 0: & \phi_{i,j,k} > 0 \text{ and } \phi_{i+1,j,k} > 0. \end{cases}$$

This of course blends very elegantly with the variable density smoke solve, where density is a function of temperature T and the specific gas constant R; this constant R can be taken as different for the fuel region and the burnt-products region.

Speaking of temperature, we don't yet have a model for it. The simplest approach is to keep a constant $T = T_{\text{ignition}}$, the temperature at which combustion starts, inside the fuel region and establish a discontinuous jump to $T_{\max}$ on the burnt products side of the flame front. The temperature on the burnt side is advected and dissipated as before for smoke, but using the trick that—just as was done for velocity above—when crossing over the flame front and referring to a temperature on the fuel side, the "ghost" value of $T_{\max}$ is used. Let's make that clear: the actual temperature T in the fuel side is kept constant at T_{ignition}, but when advecting and diffusing T on the burnt side, if reference is made to a T value in the fuel (e.g., when doing interpolation in semi-Lagrangian advection) $T_{\max}$ is used instead. For the extreme temperatures apparent in fires, hot enough to make an incandescent glow, a black-body radiation formula for the decay of T might be used instead of the simple exponential decay mentioned earlier, where the rate of cooling is proportional to the fourth power of the temperature

difference

$$\frac{DT}{Dt} = -c\left(\frac{T - T_{\text{ambient}}}{T_{\max} - T_{\text{ambient}}}\right)^4$$

for some cooling constant c defined as a user parameter. After advecting temperature around to get an intermediate $\tilde{T}$ for a time step Δt, this cooling equation can be solved analytically to give

$$T^{n+1} = T_{\text{ambient}} + \left[\frac{1}{(\tilde{T} - T_{\text{ambient}})^3} + \frac{3c\Delta t}{(T_{\max} - T_{\text{ambient}})^4}\right]^{-\frac{1}{3}}.$$

Similar to this treatment of temperature, we can also feed a smoke concentration $s_{\max}$ into the burnt region from the flame front, allowing it to be advected and dissipated as usual. Temperature and smoke concentration can feed into either a buoyancy force (if we make the Boussinesq approximation) or modify density, as discussed in Chapter 6.

The final issue is rendering, which mostly lies outside the scope of this book. The actual flame front itself is sometimes referred to as the "blue core," referring to the spectral emissions made when burning typical hydrocarbons (other fuels give rise to different spectra, giving different colors): the level set itself is a light emitter. For a dirty flame, where soot is produced, the bulk of the visual effect though is the black-body incandescence of the soot particles. That is, light is emitted from the burnt region as well, proportional to the smoke concentration s and following the black-body spectrum for the temperature T. Simultaneously, the soot is somewhat opaque, so light emitted elsewhere should be absorbed at a rate proportional to s. Further scattering effects can be included of course.

9.2 Volumetric Combustion

We now turn to an alternate model where combustion may take place throughout the volume, loosely following Feldman et al. [FOA03]. This is appropriate particularly for modeling the fireballs due to deflagration of fuel suspended in the air, whether flammable powder or liquid-fuel mist. It's also slightly simpler to implement than the preceding thin-flame model, since level sets are not involved, yet it can still achieve some of the look of regular flames and thus might be preferred in some instances.

To a standard smoke simulation, which includes temperature T, smoke concentration s, and divergence controls, we add another field F, the concentration of unburnt fuel in each grid cell. The fuel gets advected along with the flow as usual, may be additional diffused, and may be seeded at fuel sources or emitted from boundaries. (In Feldman et al.'s work, fuel is instead represented as the unburnt mass of discrete fuel particles; we'll

come back to using particle systems in grid-based fluid solvers in Chapter 7.)

To this we add some simple rules. If the temperature T at a grid cell, following advection and dissipation steps, is above some ignition threshold T_{ignition} and the fuel concentration F is above zero, we burn some of the fuel. At the simplest we reduce F by $z\Delta t$, where z is the burn speed (volume fraction combusted per second), clamping it to zero to avoid negative F. Let ΔF be the change in F; we then increase the smoke concentration s by some amount proportional to ΔF, increase the temperature T also proportional to ΔF, and add an amount to the divergence control proportional to $\Delta F/\Delta t$ (recall the divergence is percent expansion per unit of time, thus we need to divide by Δt). Note that this rule burns the fuel at a constant rate z without regard to the availability of oxidizer; Feldman et al. argue that for suspended particle explosions (coal dust, sawdust, flour, etc.) oxygen availability in the air is never the limiting factor, and thus this rule is justified. However, if need be you could limit the rate based on $1 - s - F$, the volume fraction left for plain air.

Rendering is based on black-body radiation as in the previous section, though here we don't have any representation of the "blue core" available so the rendering possibilities are slightly more limited.

10

Viscous Fluids

After briefly discussing viscosity in Chapter 1, we dispensed with it and until now have only looked at inviscid simulations. In fact our major problem has been that our numerical methods have too much numerical dissipation which, in the velocity field, looks like viscosity. We now will turn our attention to simulating highly viscous fluids, like molasses, and even variable viscosity fluids where some parts are more viscous than others perhaps due to heating or cooling, or desired animation effects.

10.1 Stress

To properly understand viscosity and avoid some potential mistakes when handling variable viscosity situations, we need to first understand the concept of stress.

In reality, at least as a first approximation, matter is composed of small particles with mass that interact by applying forces on each other. However, since we're not interested in phenomena at that microscopic scale, in fluid mechanics we make the continuum assumption, that matter is a continuous field with no discrete particles. One way of thinking about this assumption is that we're taking the limit as the particles become infinitesimally small and packed together with infinitesimal space between them. For dynamics this poses the problem that the masses drop to zero, and for accelerations to remain bounded the forces must drop to zero too. To get around this, we measure things in bulk: how much mass there is in a volume of space, or what the net force is on a volume.

While it makes no sense to ask what the mass of a continuum fluid is at a point in space (it has to be zero), we can define the density at any point, which is a useful quantity. By integrating density in a volume we get the total mass of the volume. Similarly it makes no sense to ask what the force on a continuum fluid is at a point in space (it has to be zero), but we can define quantities analogous to density: force densities in a sense, that when integrated over a region give a net force.

We've already seen two examples of force density, body forces (such as

gravity) and pressure. These are fundamentally different however: to get the net gravitational force on a region of fluid you integrate the gravitational body force density $\rho\vec{g}$ over the *volume*, but to get the net pressure force on the volume you integrate the pressure times the normal $p\hat{n}$ over the *surface* of the region. The difference is that a body force acts over a distance to influence everything inside the volume, whereas other forces (like pressure) can only act locally on the surface of contact. It is these local contact forces that we need to generalize from pressure to figure out viscosity and more exotic fluid effects.

An accepted assumption (verified to be accurate in many experiments), called Cauchy's Hypothesis, is that we can represent the local contact forces by a function of position and orientation only. That is, there is a vector field called **traction**, $\vec{t}(\vec{x}, \hat{n})$, a function of where in space we are measuring it and what the normal to the contact surface there is. It has units of force per area: to get the net contact force on a volume of fluid Ω, we integrate the traction over its surface:

$$\vec{F} = \iint_{\partial\Omega} \vec{t}(\vec{x}, \hat{n}).$$

I'd like to underscore here that this volume can be an arbitrary region containing fluid (or in fact, any continuum substance); e.g., it could be a tiny subregion in the interior of the fluid, or a grid cell, etc.

Once we accept this assumption, it can be proven that the traction must depend linearly on the normal; that is, the traction must be the result of multiplying some matrix by the normal. Technically speaking this is actually a rank-two tensor, not just a matrix, but we'll gloss over the difference for now and just call it a tensor from now on.[1] The tensor is called the **stress tensor**, or more specifically the **Cauchy stress tensor**,[2] which we label σ. Thus we can write

$$\vec{t}(\vec{x}, \hat{n}) = \sigma(\vec{x})\hat{n}.$$

Note that the stress tensor only depends on position, not on the surface normal. It can also be proven from conservation of angular momentum that the stress tensor must be symmetric: $\sigma = \sigma^T$.

[1] Basically a matrix is a specific array of numbers; a rank-two tensor is a more abstract linear operator that can be represented as a matrix when you pick a set of basis vectors with which to measure it. For the purposes of this book, you can think of them interchangeably, as we will always use a fixed Cartesian basis where y is in the vertical direction.

[2] There are other stress tensors, which chiefly differ in the basis in which they are represented. For elastic solids, where it makes sense to talk of a rest configuration to which the object tries to return, it can be convenient to set up a stress tensor in terms of rest-configuration coordinates, rather than the world-space coordinates in which Cauchy stress operates.

Since the unit normal has no units, the stress tensor is also measured as force per area, just like traction. However, it's a little harder to interpret; it's easier instead to think in terms of traction on a specific plane of contact.

As a concrete example using continuum materials that are a little easier to experience, put your hand down on a flat desk. The flesh of your hand and the wood of the desk are essentially each a continuum, and thus there is conceptually a stress tensor in each. The net force you apply on the desk with your hand is the integral of the traction over the area of contact. The normal in this case is the vertical vector $(0, 1, 0)$, so the traction at any point is

$$\vec{t} = \sigma\hat{n} = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_{33} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \sigma_{12} \\ \sigma_{22} \\ \sigma_{32} \end{bmatrix}.$$

Note that the normal force comes from the vertical component σ_{22} of traction—how hard you are pushing down on the desk. The other components of the traction, σ_{12} and σ_{32}, are tangential—how hard you are pushing the desk forwards, backwards, or to the side.

Those tangential forces are due to friction; without it there could only be a normal force. Viscosity is in many ways similar to friction, in particular that a fluid without viscosity only exerts forces in the normal direction. That is, the traction $\vec{t} = \sigma\hat{n}$ in an inviscid fluid is always in the normal direction: it must be parallel to $\hat{n}$. Since this is true for any normal vector, it can be proven that the stress tensor of an inviscid fluid must be a scalar times the identity. That scalar is, in fact, the negative of pressure. Thus, for the inviscid case we have considered up until now, the stress tensor is just

$$\sigma = -p\delta, \tag{10.1}$$

where we use δ to mean the identity tensor. When we model viscosity, we will end up with a more complicated stress tensor that can give tangential tractions.

10.2 Applying Stress

The net force due to stress on a volume Ω of fluid is the surface integral of traction:

$$\vec{F} = \iint_{\partial\Omega} \sigma\hat{n}.$$

We can use the divergence theorem to transform this into a volume integral:

$$\vec{F} = \iiint_{\Omega} \nabla \cdot \sigma.$$

Note that the notation $\nabla \cdot \sigma$ is the accepted short-hand for the vector whose elements are the divergences of the rows (or columns) of σ:

$$\nabla \cdot \sigma = \begin{bmatrix} \frac{\partial \sigma_{11}}{\partial x} + \frac{\partial \sigma_{12}}{\partial y} + \frac{\partial \sigma_{13}}{\partial z} \\ \frac{\partial \sigma_{21}}{\partial x} + \frac{\partial \sigma_{22}}{\partial y} + \frac{\partial \sigma_{23}}{\partial z} \\ \frac{\partial \sigma_{31}}{\partial x} + \frac{\partial \sigma_{32}}{\partial y} + \frac{\partial \sigma_{33}}{\partial z} \end{bmatrix}.$$

Ignoring body forces for simplicity, we set this net force equal to the mass times center-of-mass acceleration:

$$\vec{F} = M\vec{A} = \iiint_\Omega \rho \frac{D\vec{u}}{Dt},$$

i.e., we have an equality between the two volume integrals:

$$\iiint_\Omega \rho \frac{D\vec{u}}{Dt} = \iiint_\Omega \nabla \cdot \sigma.$$

Since this holds for any arbitrary volume, the integrands must be equal:

$$\rho \frac{D\vec{u}}{Dt} = \nabla \cdot \sigma.$$

Adding back in the body force term, we actually have the general momentum equation for a continuum (elastic solids as well as fluids):

$$\frac{D\vec{u}}{Dt} = \frac{1}{\rho}\vec{g} + \frac{1}{\rho}\nabla \cdot \sigma.$$

In the particular case of an inviscid fluid, as we discussed above, the stress tensor is just the negative of pressure times the identity—see Equation (10.1). In this case, it's not hard to see that $\nabla \cdot \sigma$ simplifies to $-\nabla p$, giving the familiar momentum equation.

For general fluid flow, pressure is still a very important quantity, so we will explicitly separate it out from the rest of the stress tensor:

$$\sigma = -p\delta + \tau,$$

where τ is also a symmetric tensor. We will let the pressure term handle the incompressibility constraint and model other fluid behavior with τ.

10.3 Strain Rate and Newtonian Fluids

Our model of viscosity is physically based on the fact that when molecules traveling at different speeds collide or closely interact, some energy may

be transferred to vibrational or rotational modes in the molecule—i.e., heat —and thus the difference in center-of-mass velocity between the two molecules is reduced. At the continuum level, the net effect of this is that as a region of fluid slips past another, momentum is transferred between them to reduce the difference in velocity, and the fluids get hotter. The critical thing to note is that this occurs when fluid moves *past* other fluid: in a rigid body rotation there are differences in velocity, but the fluid moves together and there is no viscous effect. Thus we really only care about how the fluid is deforming, that is how far from rigidly moving it is.

To measure differences in velocity locally, the natural quantity to consider is the gradient of velocity: $\nabla\vec{u}$. However, mixed up in the gradient is also information about the rigid rotation[3] as well as the deformation induced by the flow. We will want to separate out just the deformation part to define viscous stress.

One way to characterize rigid motion is that the dot-product of any two vectors remains constant. (If the two vectors are the same, this is just saying lengths remains constant; for different vectors we're saying the angle between them also stays the same.) How much the dot-product between two vectors changes is thus a measure of how fast the fluid is deforming. Let's look at a point $\vec{x}$: in a small time interval Δt it moves to approximately $\vec{x} + \Delta t\vec{u}(\vec{x})$. Now look at two nearby points, $\vec{x} + \Delta\vec{x}$ and $\vec{x} + \Delta\vec{y}$: linearizing appropriately, they approximately move to

$$\begin{aligned} &\vec{x} + \Delta\vec{x} + \Delta t(\vec{u}(\vec{x}) + \nabla\vec{u}\Delta\vec{x}) \\ \text{and} \quad &\vec{x} + \Delta\vec{y} + \Delta t(\vec{u}(\vec{x}) + \nabla\vec{u}\Delta\vec{y}), \end{aligned}$$

respectively. The dot-product of the vectors from $\vec{x}$ to these points begins as

$$[(\vec{x} + \Delta\vec{x}) - \vec{x}] \cdot [(\vec{x} + \Delta\vec{y}) - \vec{x}] = \Delta\vec{x} \cdot \Delta\vec{y}$$

and after the time interval is approximately

$$\begin{aligned} &[(\vec{x} + \Delta\vec{x} + \Delta t(\vec{u}(\vec{x}) + \nabla\vec{u}\Delta\vec{x})) - (\vec{x} + \Delta t\vec{u}(\vec{x}))] \\ &\quad \cdot [(\vec{x} + \Delta\vec{y} + \Delta t(\vec{u}(\vec{x}) + \nabla\vec{u}\Delta\vec{y})) - (\vec{x} + \Delta t\vec{u}(\vec{x}))] \\ &\qquad = [\Delta\vec{x} + \Delta t\nabla\vec{u}\Delta\vec{x}] \cdot [\Delta\vec{y} + \Delta t\nabla\vec{u}\Delta\vec{y}]\,. \end{aligned}$$

Then the change in the dot-product, ignoring $O(\Delta t^2)$ terms, is

$$\Delta t\,(\Delta\vec{x} \cdot \nabla\vec{u}\Delta\vec{y} + \Delta\vec{y} \cdot \nabla\vec{u}\Delta\vec{x}) = \Delta t\Delta\vec{x}^T \left(\nabla\vec{u} + \nabla\vec{u}^T\right) \Delta\vec{y}.$$

That is, the rate of change of dot-products of vectors in the flow is determined by the symmetric part of the velocity gradient, the matrix $D =$

[3] Later in the book, we will take a look at the curl of velocity which is called vorticity, $\vec{\omega} = \nabla \times \vec{u}$; it measures precisely the rotational part of the velocity field.

$\frac{1}{2}(\nabla\vec{u}+\nabla\vec{u}^T)$. This is called the **strain rate tensor** or rate of strain, since it's measuring how fast **strain**—the total deformation of the continuum—is changing.

Incidentally, the rest of the velocity gradient, the skew-symmetric part $\frac{1}{2}(\nabla\vec{u}-\nabla\vec{u}^T)$ naturally has to represent the other source of velocity differences in the flow: rotation. We'll explore this further later in the book.

We'll also immediately point out that for incompressible fluids, which is all we focus on in this book, the trace of D (the sum of the diagonal entries, denoted $\mathrm{tr}(D)$) is simply $\nabla\cdot\vec{u}=0$.

We are looking for a symmetric tensor τ to model stress due to viscosity; the rate of strain tensor D is symmetric and measures how fast the fluid is deforming. The obvious thing to do is assume τ is proportional to D. Fluids for which there is a simple linear relationship are called **Newtonian**. Air and water are examples of fluids which are, to a very good approximation, Newtonian. However, there are many liquids (generally with a more complex composition) where a non-linear relationship is essential; they go under the catch-all category of **non-Newtonian** fluids.[4] We won't go into any more detail, except to say that two classes of non-Newtonian fluids, **shear-thickening** and **shear-thinning** fluids, can be easily modeled with a viscosity coefficient μ that is a function of $\|D\|_F$, the Frobenius norm of the strain rate:[5]

$$\|D\|_F = \sqrt{\sum_{i,j=1}^{3} D_{i,j}^2}.$$

Often a power-law is assumed:

$$\mu = K\|D\|_F^{n-1}, \tag{10.2}$$

where $n=1$ corresponds to a Newtonian fluid, $n>1$ a shear-thickening fluid where apparent viscosity increases as you try to deform the fluid faster (e.g., cornstarch suspended in water), and $0<n<1$ a shear-thinning

[4] Also sometimes included in the non-Newtonian class are **viscoelastic** fluids, which blur the line between fluid and solid as they can include elastic forces that seek to return the material to an "undeformed" state—in fact these are sometimes best thought of instead as solids with permanent (**plastic**) deformations. You might refer to the article by Goktekin et al. [GBO04] and Irving's thesis [Irv07] for a fluid-centric treatment in graphics.

[5] Technically this is assuming again that the fluid is incompressible, so the trace of D is zero, which means it represents only **shearing** deformations, not expansions or contractions.

fluid where apparent viscosity increases as the fluid comes to rest (e.g., paint). Granular materials such as sand can even be modeled as the limit $n = 0$ of shear-thinning where the magnitude of "viscous" stress depends instead on pressure, not the magnitude of the strain rate, making it more akin to dry Coulomb friction; see Zhu and Bridson [ZB05] for more on this subject.

Getting back to simple Newtonian fluids, the relationship for incompressible flow is

$$\tau = 2\mu D + \lambda \mathrm{tr}(D)\delta, \tag{10.3}$$

where μ is the **coefficient of dynamic viscosity**. The second term, involving $\mathrm{tr}(D) = \nabla \cdot \vec{u}$, is of course zero for incompressible flow for any λ (which is termed the **second coefficient of viscosity**, and is also associated with the term "bulk viscosity," i.e. viscous resistance to changing bulk a.k.a. volume). For compressible flow λ is often taken to be $-\frac{2}{3}\mu$, though theoretically this is only an idealization of monatomic gases. However, for an incompressible fluid we are free to choose λ as we please,[6] and thus for simplicity's sake we'll set $\lambda = 0$ for now.

Plugging this into the momentum equation, we get

$$\frac{D\vec{u}}{Dt} + \frac{1}{\rho}\nabla p = \frac{1}{\rho}\nabla \cdot \left(\mu(\nabla \vec{u} + \nabla \vec{u}^T)\right). \tag{10.4}$$

You may notice that this isn't quite the same as our first statement of the momentum equation, Equation (1.1). It turns out that for the common case where μ is constant, the correct equation (10.4) does in fact simplify to Equation (1.1): but be aware, for simulations with variable viscosity, only Equation (10.4) is correct. For example, Equation (1.1) doesn't conserve angular momentum in the variable viscosity case. The simplification also is unhelpful in applying the correct free surface boundary condition (see below), which is critical for viscous liquid simulations, and since highly viscous gases are rarely of interest for graphics, I actually recommend **not** bothering with the simplification, since it's hardly ever useful.

However, for completeness' sake, let's work through the simplification so it at least doesn't remain a mystery. If μ is constant, then we can take

[6] This isn't quite true: for $\lambda < -\frac{2}{3}\mu$, basic thermodynamics are violated, with viscosity actually accelerating expansion or contraction, increasing the energy of the system. In a numerical method, where divergence probably isn't exactly zero even for incompressible flow, problems are bound to arise.

it out from under the divergence:

$$\begin{aligned}
\frac{D\vec{u}}{Dt} + \frac{1}{\rho}\nabla p &= \frac{\mu}{\rho}\nabla\cdot(\nabla\vec{u} + \nabla\vec{u}^T) \\
&= \frac{\mu}{\rho}\left[\nabla\cdot\nabla\vec{u} + \nabla\cdot(\nabla\vec{u}^T)\right] \\
&= \frac{\mu}{\rho}\left[\nabla\cdot\nabla\vec{u} + \begin{pmatrix} \frac{\partial}{\partial x}\frac{\partial u}{\partial x} + \frac{\partial}{\partial y}\frac{\partial v}{\partial x} + \frac{\partial}{\partial z}\frac{\partial w}{\partial x} \\ \frac{\partial}{\partial x}\frac{\partial u}{\partial y} + \frac{\partial}{\partial y}\frac{\partial v}{\partial y} + \frac{\partial}{\partial z}\frac{\partial w}{\partial y} \\ \frac{\partial}{\partial x}\frac{\partial u}{\partial z} + \frac{\partial}{\partial y}\frac{\partial v}{\partial z} + \frac{\partial}{\partial z}\frac{\partial w}{\partial z} \end{pmatrix}\right] \\
&= \frac{\mu}{\rho}\left[\nabla\cdot\nabla\vec{u} + \begin{pmatrix} \frac{\partial}{\partial x}(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}) \\ \frac{\partial}{\partial y}(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}) \\ \frac{\partial}{\partial z}(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}) \end{pmatrix}\right].
\end{aligned}$$

In the last step we simply changed the order of partial derivatives in the last term and regrouped. But now we see that the last term is simply the gradient of $\nabla\cdot\vec{u}$:

$$\frac{D\vec{u}}{Dt} + \frac{1}{\rho}\nabla p = \frac{\mu}{\rho}\left[\nabla\cdot\nabla\vec{u} + \nabla(\nabla\cdot\vec{u})\right].$$

If the flow is incompressible, i.e., $\nabla\cdot\vec{u} = 0$. Finally, substituting the **kinematic viscosity** $\nu = \mu/\rho$ in, we end up back at Equation (1.1). I emphasize that this only happens when both the viscosity is constant through the flow and the velocity field is incompressible. The second point also becomes important numerically since at intermediate stages in our time integration our velocity field may not be discretely incompressible—and then this last term can't be blithely ignored.

Getting back to variable viscosity, some formula needs to be decided for μ. For a non-Newtonian fluid, it might be a function of the magnitude of the strain rate, as we have seen. For regular Newtonian fluids it might instead be a function of temperature—assuming we're tracking temperature in the simulation as we saw how to do with smoke and fire—which is most important for liquids. Carlson et al. [CMIT02] suggest that modeling melting and solidifying (freezing) can be emulated by making the viscosity a low constant for temperatures above a transition zone (centered on the melting point) and a high constant for temperatures below the transition zone (thus giving near-rigid behavior), and smoothly varying between the two in the narrow transition zone itself.

10.4 Boundary Conditions

The two types of boundaries considered in this book are free surfaces and solid walls, and each has particular conditions associated with viscosity.

In the case of a free surface, things are fairly straightforward. On the other side of the boundary there is a vacuum, or another fluid of much smaller density whose effect we assume is negligible. Thus there is nothing with which to transfer momentum: there can be no traction at the free surface. In other words, the boundary condition for the stress at the free surface is

$$\sigma\hat{n} = -p\hat{n} + \tau\hat{n} = 0.$$

Note that if the viscous stress τ is zero, this reduces to $p = 0$ as before; however, this becomes significantly more complex when τ isn't zero. In terms of the velocity, and assuming no special treatment of bulk viscosity, this condition is:

$$-p\hat{n} + (\nabla\vec{u} + \nabla\vec{u}^T)\hat{n} = 0.$$

In particular, the true free surface condition is quite different from separately specifying $p = 0$ and $\nabla\vec{u} \cdot \hat{n} = 0$, an erroneous and physically meaningless boundary condition which unfortunately crops up regularly in research papers. Batty and Bridson [BB08] worked through the first correct treatment of a viscous free surface in graphics, illustrating how this error destroys many of the critical visual features in highly viscous flow.

At solid walls, things are also a little different. Physically speaking once we model viscosity, it turns out the velocity field must be continuous everywhere: if it weren't, viscous transfer of momentum would in the next instant make it continuous again. This results in the so-called **no-slip** boundary condition:

$$\vec{u} = \vec{u}_{\text{solid}},$$

which of course simplifies to $\vec{u} = 0$ at stationary solids. Recall that in the inviscid case, only the normal component of velocities had to match: here we are forcing the tangential components to match as well.

The no-slip condition has been experimentally verified to be more accurate than the inviscid no-stick condition. However, the caveat is that in many cases something called a **boundary layer** develops. Loosely speaking, a boundary layer is a thin region next to a solid where the tangential velocity rapidly changes from $\vec{u}_{\text{solid}}$ at the solid wall to $\vec{u}^\star$ at the other side of the layer, where $\vec{u}^\star$ is the velocity the inviscid no-stick boundary condition ($\vec{u} \cdot \hat{n} = \vec{u}_{\text{solid}} \cdot \hat{n}$) would have given. That is, the effect of viscous drag at the surface of the solid is restricted to a very small region next to the solid, and can be ignored elsewhere. When we discretize the fluid flow on a relatively coarse grid, that boundary layer may be much thinner than a

grid cell, and thus it's no longer a good idea to implement the no-slip condition numerically—we would artificially be expanding the boundary layer to at least a grid cell thick, which would be a much worse approximation than going back to the inviscid no-stick boundary condition. In this case, we have

$$\vec{u} \cdot \hat{n} = \vec{u}_{\text{solid}} \cdot \hat{n},$$
$$(\tau \hat{n}) \times \hat{n} = 0,$$

where the second boundary equation is indicating that the viscous stress causes no tangential traction. There is even an intermediate case, the Navier slip condition, which allows some tangential slip but also some tangential "drag" at the solid surface. However, for simplicity's sake, we will stick with the simpler no-slip condition for the rest of the chapter.

10.5 Implementation

The first simplification we will make is to use time-splitting again, handling viscosity in a separate step from advection and body forces. It can be profitably combined with the pressure solve, enforcing incompressibility while simultaneously integrating viscous effects, which sometimes is called an "unsteady Stokes" solve.[7] Batty and Bridson [BB10] show how, if you solve for p and the components of τ you can still do this with an SPD matrix, and use the same linear solver code as we used for pressure.

However, for this book, we'll take one more common simplification, time splitting viscosity from pressure projection. That is, we'll project out pressure to make the flow incompressible, and then solve for the effect of viscosity on velocity. For one time step Δt, this update is

$$\frac{\partial \vec{u}}{\partial t} = \frac{1}{\rho} \nabla \cdot \left(\mu (\nabla \vec{u} + \nabla \vec{u}^T) \right),$$

with any or all of the boundary conditions given above. Although we will use the full form of viscosity in equation (10.4), which can help damp away any remaining divergence in the field, this step can also trade reduced a shear rate for an increased divergence, so it's generally necessary to perform a second pressure projection afterwards, before continuing with advection in the resulting velocity field.[8]

[7] Stokes flow, as opposed to Navier-Stokes, is used to model slow-moving and viscous incompressible flow; it is what you get when you drop the nonlinear $\vec{u} \cdot \nabla \vec{u}$ term from the momentum equation arguing it is a negligible high-order effect, reducing the equations to much simpler linear equations.

[8] Increasing the bulk viscosity coefficient λ to a substantial positive multiple of μ can lessen the need for a second projection, but also causes convergence troubles for the linear solver in an implicit step, so it's probably cheaper nevertheless to stick with a second projection.

Likewise, we will time split the boundary conditions, enforcing $p = 0$ at free surfaces and $\vec{u} \cdot \hat{n} = \vec{u}_{\text{solid}} \cdot \hat{n}$ at solid boundaries in the pressure solve, and then $(\nabla\vec{u} + \nabla\vec{u}^T)\hat{n} = 0$ at free surfaces and $\vec{u} = \vec{u}_{\text{solid}}$ at solid boundaries in the viscous solve.

Staggered grids make life easier for viscosity, as they did for pressure. Let's take a look at the contribution to the horizontal component of velocity, given the viscous stress tensor τ:

$$u^{n+1} = u^P + \frac{\Delta t}{\rho}\left(\frac{\partial \tau^{11}}{\partial x} + \frac{\partial \tau^{12}}{\partial y} + \frac{\partial \tau^{13}}{\partial z}\right).$$

(Here I am using u^P as the velocity field *after* pressure projection, and u^{n+1} is the finalized velocity field after viscosity is included.) Since u is located in the grid at, say, $(i+1/2, j, k)$, it's natural to ask for τ^{11} to be at grid cell centers (i, j, k), for τ^{12} to be at the edge-center $(i+1/2, j+1/2, k)$, and for τ^{13} at $(i+1/2, j, k+1/2)$. This gives an elegant discretization:

$$u^{n+1}_{i+1/2,j,k} = u^P_{i+1/2,j,k} + \frac{\Delta t}{\rho}\left(\frac{\tau^{11}_{i+1,j,k} - \tau^{11}_{i,j,k}}{\Delta x} + \frac{\tau^{12}_{i+1/2,j+1/2,k} - \tau^{12}_{i+1/2,j-1/2,k}}{\Delta x} + \frac{\tau^{13}_{i+1/2,j,k+1/2} - \tau^{13}_{i+1/2,j,k-1/2}}{\Delta x}\right).$$

Similarly for the other components of velocity:

$$v^{n+1}_{i,j+1/2,k} = v^P_{i,j+1/2,k} + \frac{\Delta t}{\rho}\left(\frac{\tau^{12}_{i+1/2,j+1/2,k} - \tau^{12}_{i-1/2,j+1/2,k}}{\Delta x} + \frac{\tau^{22}_{i,j+1,k} - \tau^{22}_{i,j,k}}{\Delta x} + \frac{\tau^{23}_{i,j+1/2,k+1/2} - \tau^{23}_{i,j+1/2,k-1/2}}{\Delta x}\right),$$

$$w^{n+1}_{i,j,k+1/2} = w^P_{i,j,k+1/2} + \frac{\Delta t}{\rho}\left(\frac{\tau^{13}_{i+1/2,j,k+1/2} - \tau^{13}_{i-1/2,j,k+1/2}}{\Delta x} + \frac{\tau^{23}_{i,j+1/2,k+1/2} - \tau^{23}_{i,j-1/2,k+1/2}}{\Delta x} + \frac{\tau^{33}_{i,j,k+1} - \tau^{33}_{i,j,k}}{\Delta x}\right).$$

Note that these formulas make use of the symmetry of τ, e.g., $\tau^{12} = \tau^{21}$. In 2D, they simplify the obvious way. But how do we determine the values of τ on the staggered grid?

10.5.1 Explicit Treatment

The simplest thing of all is to just use central differences on the given velocity field. From the definition $\tau = \mu(\nabla\vec{u} + \nabla\vec{u}^T)$ we get

$$\tau^{11}_{i,j,k} = 2\mu_{i,j,k}\frac{u^P_{i+1/2,j,k} - u^P_{i-1/2,j,k}}{\Delta x}$$

$$\tau^{12}_{i+1/2,j+1/2,k} = \mu_{i+1/2,j+1/2,k}\left(\frac{u^P_{i+1/2,j+1,k} - u^P_{i+1/2,j,k}}{\Delta x} + \frac{v^P_{i+1,j+1/2,k} - v^P_{i,j+1/2,k}}{\Delta x}\right)$$

$$\tau^{13}_{i+1/2,j,k+1/2} = \mu_{i+1/2,j,k+1/2}\left(\frac{u^P_{i+1/2,j,k+1} - u^P_{i+1/2,j,k}}{\Delta x} + \frac{w^P_{i+1,j,k+1/2} - w^P_{i,j,k+1/2}}{\Delta x}\right)$$

$$\tau^{22}_{i,j,k} = 2\mu_{i,j,k}\frac{v^P_{i,j+1/2,k} - v^P_{i,j-1/2,k}}{\Delta x}$$

$$\tau^{23}_{i,j+1/2,k+1/2} = \mu_{i,j+1/2,k+1/2}\left(\frac{v^P_{i,j+1/2,k+1} - v^P_{i,j+1/2,k}}{\Delta x} + \frac{w^P_{i,j+1,k+1/2} - w^P_{i,j,k+1/2}}{\Delta x}\right)$$

$$\tau^{33}_{i,j,k} = 2\mu_{i,j,k}\frac{w^P_{i,j,k+1/2} - w^P_{i,j,k-1/2}}{\Delta x}$$

This can be simplified in the obvious way for 2D.

Beyond boundaries, we need ghost velocity values to plug into these formulas. For viscous solid walls with the no-slip condition, it's natural to simply use the solid velocity itself. For free surfaces we have to be a little more careful. The obvious first approach to try is to just extrapolate the velocity into the air, as we have done before, effectively setting $\vec{u}$ at a point in the air to the velocity at the closest point on the surface.

However, this simple extrapolation induces a non-negligible error. As a thought experiment, imagine a blob of fluid moving rigidly in free flight: it has zero deformation since its internal velocity field is rigid, therefore it should experience no viscous forces—i.e., τ should evaluate to zero, even at the boundary. However, if a rigid rotation is present, τ only evaluates

to zero if the ghost velocities keep that same rotation: extrapolating as a constant doesn't, and will induce erroneous viscous resistance at the boundary. Ideally a more sophisticated extrapolation scheme such as linear extrapolation should be used. That said, at present our first-order time-splitting of advection from pressure also will induce a similar erroneous drag on rotational motion, which we'll discuss in Chapter 11. Reducing one error but not the other is probably not worth the bother, and thus we'll leave this question open for further research.

The chief problem with the method as presented is stability. Unfortunately this method is liable to blow up if Δt is too large. Let's examine a simple 1D model diffusion problem to understand why,

$$\frac{\partial q}{\partial t} = k\frac{\partial^2 q}{\partial x^2},$$

where q models a velocity component and k models μ/ρ, the kinematic viscosity. Our explicit discretization would give

$$q_i^{n+1} = q_i^n + \Delta t k \frac{q_{i+1}^n - 2q_i^n + q_{i-1}^n}{\Delta x^2}.$$

Consider the highest spatial-frequency component possible in the numerical solution, say $q_i^n = Q^n(-1)^i$. Here Q^n is the time n scalar coefficient multiplying the ± 1 underlying basis function, with grid index i the exponent of (-1). Plug this in to see what Q^{n+1} is:

$$\begin{aligned}
Q^{n+1}(-1)^i &= Q^n(-1)^i + \Delta t k \frac{Q^n(-1)^{i+1} - 2Q^n(-1)^i + Q^n(-1)^{i-1}}{\Delta x^2} \\
&= Q^n(-1)^i \left(1 + \frac{\Delta t k}{\Delta x^2}(-1-2-1)\right) \\
&= Q^n(-1)^i \left(1 - \frac{4\Delta t k}{\Delta x^2}\right), \\
Q^{n+1} &= \left(1 - \frac{4\Delta t k}{\Delta x^2}\right) Q^n.
\end{aligned}$$

This can only exponentially and monotonically decay, as we would expect viscosity to do, if

$$\Delta t < \frac{\Delta x^2}{4k}.$$

Otherwise we end up with q oscillating in time—a physically incorrect vibration—and possibly even exponential *increase*, which is thermodynamically impossible and potentially disastrous numerically.

For the full 3D viscous problem, a similar time step restriction applies:

$$\Delta t < \frac{\Delta x^2 \rho}{12\mu_{\max}}. \tag{10.5}$$

This is pretty severe: while we discussed the merit of restricting Δt to be $O(\Delta x)$ to control errors in advection, this possibly reduces it a whole order of magnitude further. More to the point, there is no accuracy requirement to keep Δt this small: the physics of viscous dissipation essentially boils down to the exponential decay of deformation modes, which can be well approximated even with large time steps. In numerical lingo, this means that the problem is **stiff**: accuracy is saying it should be fine to take large Δt, but stability is requiring punishingly small Δt. The usual numerical solution is to use **implicit** time integration.

10.5.2 Implicit Treatment

The simplest implicit time integration scheme is called **backward Euler**. In this case it means rather than evaluating the stress tensor based on the old velocities $\vec{u}^n$, we'll base it on the new velocities $\vec{u}^{n+1}$, which of course, we don't know yet, until we get the stress tensor—which again depends on knowing the new velocities. This isn't a paradox: it's merely an *implicit* definition of the new velocities, giving us simultaneous equations we must solve to find them.

Let's first do it with the 1D model problem from the last section. The backward Euler discretization is

$$q_i^{n+1} = q_i^n + \Delta t k \frac{q_{i+1}^{n+1} - 2q_i^{n+1} + q_{i-1}^{n+1}}{\Delta x^2}.$$

This is the ith linear equation: we can complete the system by enforcing, say, no-slip boundary conditions $q_0 = q_{m+1} = 0$, leaving m equations in m unknowns $q_1^{n+1}, \ldots q_m^{n+1}$. Rearranging gives

$$-\frac{\Delta t k}{\Delta x^2} q_{i+1}^{n+1} + \left(1 + \frac{2\Delta t k}{\Delta x^2}\right) q_i^{n+1} - \frac{\Delta t k}{\Delta x^2} q_{i-1}^{n+1} = q_i^n.$$

This can now be thought of as a classic matrix-times-unknown-vector-equals-known-vector problem,

$$\left(I + \frac{\Delta t k}{\Delta x^2} A\right) q^{n+1} = q^n,$$

where I is the identity matrix, and A is a tridiagonal matrix with 2 down the main diagonal and -1 along the sub- and super-diagonals:

$$A = \begin{pmatrix} 2 & -1 & & \\ -1 & 2 & -1 & \\ & \ddots & \ddots & \ddots \\ & & -1 & 2 \end{pmatrix}.$$

This is almost the same, up to scaling, as a 1D version of the Poisson problem for pressure, except now we are increasing the positive diagonal entries even further. The matrix is symmetric positive definite—in fact slightly better conditioned than the pressure matrix thanks to the incremented diagonal—and so solving it with PCG works very efficiently.

Does this solve the stability problem? Well, let's rewrite the ith equation again:

$$\left(1 + \frac{2\Delta t k}{\Delta x^2}\right) q_i^{n+1} = \frac{\Delta t k}{\Delta x^2} q_{i+1}^{n+1} + \frac{\Delta t k}{\Delta x^2} q_{i-1}^{n+1} + q_i^n$$

$$\Rightarrow \qquad q_i^{n+1} = \left(\frac{\Delta t k / \Delta x^2}{1 + 2\Delta t k / \Delta x^2}\right) q_{i+1}^{n+1} + \left(\frac{\Delta t k / \Delta x^2}{1 + 2\Delta t k / \Delta x^2}\right) q_{i-1}^{n+1} + \left(\frac{1}{1 + 2\Delta t k / \Delta x^2}\right) q_i^n.$$

That is, the new value at a grid point is a weighted average (with guaranteed positive weights, summing to 1) of its neighbors' new values and the old value at that grid point. Here we're including ghost values of q in the boundaries for some of those averages. Therefore, the maximum new value of q has to be less than or equal to the maximum old value of q, and similarly the minimum new value of q has to be greater than or equal to the minimum old value. So unstable growth is impossible. A more detailed analysis further can prove that spurious oscillations (the unphysical vibration we could hit before) are ruled out as well. This is all true no matter how large Δt is taken: it's unconditionally stable and monotone![9]

10.5.3 Variational Form of Implicit Integration

Before we get back to the full 3D problem of viscosity, we will take one more step with this 1D model problem. The issue is that the free surface boundary condition in 3D, $(\nabla \vec{u} + \nabla \vec{u}^T)\hat{n} = 0$, is difficult to directly incorporate into finite differences. To make progress, we need another tool in our belt: the calculus of variations. We hinted at this before in Chapter 8, discussing surface tension. Here we will use it to rephrase an implicit, Backwards Euler step as the search for a velocity field which minimizes a special quantity. Batty and Bridson [BB08] introduced this numerical approach to viscosity specifically because it naturally captures the free surface boundary condition without any extra work.

This might just be the most intimidating bit of math in this book. I'm including it because I think it is one of the most wonderful tools in applied

[9] This of course doesn't mean we're necessarily getting the correct answer if Δt is very large—there is still an approximation error. However, a Fourier analysis can show that for large Δt the only "problem" is that the diffusion, the exponential decay of all the deformation modes, is effectively slower than it should be, though it still takes place.

mathematics, and worth seeing, but if you have a tough time following this, don't worry: you can skip on to the next section and just trust that the math there can be justified.

Instead of discretizing first in space and then in time, let's discretize the 1D problem $\partial q/\partial t = k\partial^2 q/\partial x^2$ first in time, with Backwards Euler:

$$q^{n+1} = q^n + \Delta t k \frac{\partial^2 q^{n+1}}{\partial x^2}.$$

We will want the analog of free-surface boundaries, by setting $\partial q/\partial x = 0$ at $x = 0, 1$. We already have an idea that the solution q^{n+1} will be like a smoothed out version of q^n: the smaller $\Delta t k$ is the closer q^{n+1} will be to q^n, and the bigger $\Delta t k$ is the more it will be smoothed out to a fully diffused constant average value.

For any 1D function q, take a look at this special combination of integrals on the interval $[0, 1]$:

$$E[q] = \int_0^1 \frac{1}{2}\Big(q(x) - q^n(x)\Big)^2 \, dx + \int_0^1 \frac{\Delta t k}{2} \left(\frac{\partial q(x)}{\partial x}\right)^2 \, dx.$$

This "functional" $E[q]$ takes a function q and returns a single real number which is a sum of two terms. The first term measures how far q is from q_n, by integrating the squared difference; the second term measures how big the first spatial derivative of q is, i.e. how far from being constant q is, and is weighted by $\Delta t k$. Assuming $k > 0$ and $\Delta t > 0$, also note that $E[q]$ is always non-negative, since it's integrating positive factors times squares.

We can then ask for what function q does $E[q]$ take a minimal value? If E were a regular function that took numbers instead of functions as its argument, we could use the usual calculus trick of looking for where the derivative of E is zero—but it's a bit of a leap to take the derivative of a functional with respect to another function!

There is a clever end-run around this though. Suppose that $q(x)$ actually was the function for which $E[q]$ is minimized.[10] Now say that $r(x)$ is any nonzero smooth function on $[0, 1]$, and define the *regular* function $g(s)$ as follows:

$$\begin{aligned} g(s) &= E[q + sr] \\ &= \int_0^1 \frac{1}{2}\Big(q(x) + sr(x) - q^n(x)\Big)^2 \, dx + \int_0^1 \frac{\Delta t k}{2} \left(\frac{\partial q(x)}{\partial x} + s\frac{\partial r(x)}{\partial x}\right)^2 \, dx. \end{aligned}$$

[10] We actually are assuming a lot here, both that a minimum is achieved by any function, and that it is unique: some higher-powered math is required to make this all rigorous, but the derivation we're following gives the right impression of what's going on nevertheless.

The important thing to keep in mind is that $g(s)$ just maps a single real number s to another real number: the definition is a bit complicated, but our usual calculus approach can be used on it. In fact, let's rewrite $g(s)$ a bit to show that it's nothing more complicated than a quadratic equation, with some fancy but constant coefficients. We'll multiply out the squared terms in the integrals, and then move the factor s out of the integrals since it is just a real number, not a function of x:

$$\begin{aligned}
g(s) &= \int_0^1 \frac{1}{2}\Big(\left(q(x) - q^n(x)\right)^2 + 2\left(q(x) - q^n(x)\right) r(x)s + r(x)^2 s^2 \Big)\, dx \\
&\quad + \int_0^1 \frac{\Delta t k}{2} \left(\left(\frac{\partial q(x)}{\partial x}\right)^2 + 2\frac{\partial q(x)}{\partial x}\frac{\partial r(x)}{\partial x}s + \left(\frac{\partial r(x)}{\partial x}\right)^2 s^2 \right) dx \\
&= \left[\int_0^1 \frac{1}{2}\Big(q(x) - q^n(x)\Big)^2 dx + \int_0^1 \frac{\Delta t k}{2} \left(\frac{\partial q(x)}{\partial x}\right)^2 dx \right] \\
&\quad + \left[\int_0^1 \Big(q(x) - q^n(x)\Big) r(x)\, dx + \int_0^1 \Delta t k \left(\frac{\partial q(x)}{\partial x}\frac{\partial r(x)}{\partial x}\right) dx \right] s \\
&\quad + \left[\int_0^1 \frac{1}{2} r(x)^2\, dx + \int_0^1 \frac{\Delta t k}{2} \left(\frac{\partial r(x)}{\partial x}\right)^2 dx \right] s^2 \\
&= C + Bs + As^2.
\end{aligned}$$

Now, under the assumption that $q(x)$ is the true minimum of $E[q]$, it must be the case that $g(s) = E[q + sr]$ is minimized at $s = 0$. Otherwise, the function $q(x) + sr(x)$ would have an E value even lower than the minimum! If $s = 0$ is the minimum of the quadratic function $g(s) = C + Bs + As^2$, then $g'(0) = B$ has to be zero.

That is, we have argued that if $q(x)$ minimizes the functional $E[q]$, then for any smooth function $r(x)$, we must have that

$$\left[\int_0^1 \Big(q(x) - q^n(x)\Big) r(x)\, dx + \int_0^1 \Delta t k \left(\frac{\partial q(x)}{\partial x}\frac{\partial r(x)}{\partial x}\right) dx \right] = 0.$$

It's nice to have an equation, but it doesn't tell us much about the minimizing function $q(x)$ yet. Let's use another calculus trick, integration-by-parts, on the second integral:

$$\int_0^1 \Big(q(x) - q^n(x)\Big) r(x)\, dx - \int_0^1 \Delta t k \left(\frac{\partial^2 q(x)}{\partial x^2} r(x)\right) dx + \left[\frac{\partial q(x)}{\partial x} r(x)\right]_0^1 = 0.$$

We can now unite the integrals, identifying a common factor of $r(x)$:

$$\int_0^1 \left(q(x) - q^n(x) - \Delta t k \frac{\partial^2 q(x)}{\partial x^2}\right) r(x)\, dx + \left[\frac{\partial q(x)}{\partial x} r(x)\right]_0^1 = 0.$$

This last equation has to hold no matter what we choose for the nonzero function $r(x)$. With each choice of $r(x)$, of course the quadratic $g(s)$ has different coefficients, but the argument still holds giving us this equation. The only way this integral, plus the boundary terms, can always evaluate to zero regardless of the choice of $r(x)$ is if

$$\left(q(x) - q^n(x) - \Delta t k \frac{\partial^2 q(x)}{\partial x^2}\right) = 0$$

for all $x \in (0,1)$, and at the boundary if

$$\left.\frac{\partial q(x)}{\partial x}\right|_{x=0} = 0 \quad \left.\frac{\partial q(x)}{\partial x}\right|_{x=1} = 0.$$

That is, the function q which minimizes $E[q]$ has to satisfy this differential equation in $(0,1)$,

$$q(x) = q^n(x) + \Delta t k \frac{\partial^2 q(x)}{\partial x^2},$$

and also is subject to the "free" boundary condition $\partial q/\partial x = 0$ at the ends of the domain.

This gives us a new way to express Backwards Euler: instead of working straight from the PDE, we can instead say the next time step function q^{n+1} is whatever minimizes $E[q]$. The marvellous bonus is that this automatically implies the free boundary conditions without any extra work.

In fact, we can discretize $E[q]$ in space, approximating the integrals with sums over n discrete intervals:

$$E[q] \approx E_{\Delta x}[q] = \sum_{i=0}^{n} \frac{\omega_i}{2}\left(q_i - q_i^n\right)^2 \Delta x + \sum_{i=0}^{n-1} \frac{\Delta t k}{2}\left(\frac{q_{i+1} - q_i}{\Delta x}\right)^2 \Delta x$$

The special weight ω_i is 1 except at the ends of the interval $i = 0, n$ where $\omega_0 = \omega_n = 1/2$, in order to give a consistent estimate of the integral.

Once we have a discrete $E_{\Delta x}[q]$, our actual Backwards Euler step is defined by minimizing it. With some effort (which we won't go through here), you can show that $E_{\Delta x}[q]$ is just a big quadratic in q, and setting its gradient with respect to q_i to zero gives the i'th linear equation we derived for Backwards Euler already. However, the important point is that the discrete solution will again automatically satisfy free boundary conditions $\partial q/\partial x = 0$ without us having to explicitly handle them.

10.5.4 Implicit Viscosity with Free Surfaces

At last we have the tools we need to do an accurate viscous solve, with free surfaces included. Following Batty and Bridson [BB08], we look for

the new velocity field which minimizes the energy

$$E[\vec{u}] = \iiint_\Omega \frac{\rho}{2}\|\vec{u} - \vec{u}^P\|^2 + \iiint_\Omega \Delta t\mu \left\| \frac{\nabla\vec{u} + \nabla\vec{u}^T}{2} \right\|_F^2$$

The new velocity field (which will be $\vec{u}^{n+1}$, the finalized velocity at the end of the time step) is thus a balance between staying close to $\vec{u}^P$ (the velocity after pressure projection) and trying to eliminate all deformation to get just rigid motion, with the exact balance decided by the density, the time step, and the viscosity coefficient. At solid boundaries we enforce $\vec{u} = \vec{u}_{\text{solid}}$, but at free surfaces we don't have to do anything special, thanks to the variational magic. This will give us a stable Backwards Euler step of viscosity.

To discretize in space, we just approximate $E[\vec{u}]$ with sums and finite differences as above. For a basic first order accurate solution, which nevertheless gives all the right visual behavior of viscous flow, it suffices to simply include in the sums those staggered grid points which are inside the fluid, without any special weights. The first integral then breaks up into three sums:

$$\begin{aligned}
&\iiint_\Omega \frac{\rho}{2}\|\vec{u} - \vec{u}^P\|^2 \\
&\quad = \iiint_\Omega \frac{\rho}{2}(u - u^P)^2 + \iiint_\Omega \frac{\rho}{2}(v - v^P)^2 + \iiint_\Omega \frac{\rho}{2}(w - w^P)^2 \\
&\quad \approx \sum_{(i+1/2,j,k)\in\Omega} \frac{\rho_{i+1/2,j,k}}{2}(u_{i+1/2,j,k} - u^P_{i+1/2,j,k})^2\Delta x^3 + \\
&\qquad + \sum_{(i,j+1/2,k)\in\Omega} \frac{\rho_{i,j+1/2,k}}{2}(v_{i,j+1/2,k} - v^P_{i,j+1/2,k})^2\Delta x^3 + \\
&\qquad + \sum_{(i,j,k+1/2)\in\Omega} \frac{\rho_{i,j,k+1/2}}{2}(w_{i,j,k+1/2} - w^P_{i,j,k+1/2})^2\Delta x^3
\end{aligned}$$

Here the sums are over staggered grid points where the fluid velocity is going to be defined, say where the liquid-air level set interpolates to negative but the solid level set interpolates to positive. Note that the fluid density ρ can vary across the grid.

The second integral similarly breaks up into six sums, one for each of the distinct components of the Frobenius norm, taking into account symmetry. We switch to short-hand derivative notation again for brevity:

$$\begin{aligned}
&\left\| \frac{\nabla\vec{u} + \nabla\vec{u}^T}{2} \right\|_F^2 = \\
&\qquad u_x^2 + \tfrac{1}{4}(u_y + v_x)^2 + \tfrac{1}{4}(u_z + w_x)^2 + v_y^2 + \tfrac{1}{4}(v_z + w_y)^2 + w_z^2
\end{aligned}$$

Let's look at the first two of these for example, u_x^2 and $\frac{1}{4}(u_y+v_x)^2$, approximating their integrals separately as a sum over the staggered grid points (i,j,k) and $(i+1/2,j+1/2,k)$ respectively, inside the fluid domain. By *inside* we mean that all the nearby velocity values used in the finite difference are either real fluid velocities (appearing in the sums above) or are solid velocities (which we use from the solid, respecting the no-slip boundary condition). The first:

$$\iiint_\Omega \Delta t\mu u_x^2 \approx \sum_{i,j,k} \Delta t\mu_{i,j,k} \left(\frac{u_{i+1/2,j,k} - u_{i-1/2,j,k}}{\Delta x} \right)^2 \Delta x^3.$$

The second:

$$\iiint_\Omega \frac{\Delta t\mu}{4}(u_y+v_x)^2 \approx$$
$$\sum \tfrac{\Delta t\mu}{4} \left(\frac{u_{i+1/2,j+1,k} - u_{i+1/2,j,k} + v_{i+1,j+1/2,k} - v_{i,j+1/2,k}}{\Delta x} \right)^2 \Delta x^3.$$

I left the indices off the dynamic viscosity coefficient $\mu_{i+1/2,j+1/2,k}$ in the second to save space, but as with ρ it can vary across the domain for variable or non-Newtonian viscosity simulations. If the viscosity coefficient is only specified at grid points, then the appropriate averages can be used to estimate it at staggered locations.

Once we have these discrete sums in hand, we can differentiate with respect to the unknown velocities and set the gradient to zero to find the minimizer. This will be a system of linear equations (SPD as before, so PCG works) to solve for the new velocity field. Here is the equation taken from differentiating $E_{\Delta x}[\vec{u}]$ by $u_{i+1/2,j,k}$, for example, and dropping the Δx^3 common factor:

$$\begin{aligned}
&\rho_{i+1/2,j,k}(u_{i+1/2,j,k} - u^P_{i+1/2,j,k}) \\
&\quad + 2\Delta t\mu_{i,j,k} \left(\frac{u_{i+1/2,j,k} - u_{i-1/2,j,k}}{\Delta x^2} \right) \\
&\quad - 2\Delta t\mu_{i+1,j,k} \left(\frac{u_{i+3/2,j,k} - u_{i+1/2,j,k}}{\Delta x^2} \right) \\
&\quad + \tfrac{\Delta t\mu_{i+1/2,j-1/2,k}}{2} \left(\frac{u_{i+1/2,j,k} - u_{i+1/2,j-1,k} + v_{i+1,j-1/2,k} - v_{i,j-1/2,k}}{\Delta x^2} \right) \\
&\quad - \tfrac{\Delta t\mu_{i+1/2,j+1/2,k}}{2} \left(\frac{u_{i+1/2,j+1,k} - u_{i+1/2,j,k} + v_{i+1,j+1/2,k} - v_{i,j+1/2,k}}{\Delta x^2} \right)
\end{aligned}$$

$$
\begin{aligned}
&+ \tfrac{\Delta t \mu_{i+1/2,j,k-1/2}}{2} \left(\frac{u_{i+1/2,j,k} - u_{i+1/2,j,k-1} + w_{i+1,j,k-1/2} - w_{i,j,k-1/2}}{\Delta x^2} \right) \\
&- \tfrac{\Delta t \mu_{i+1/2,j,k+1/2}}{2} \left(\frac{u_{i+1/2,j,k+1} - u_{i+1/2,j,k} + w_{i+1,j,k+1/2} - w_{i,j,k+1/2}}{\Delta x^2} \right) \\
&\qquad = 0.
\end{aligned}
$$

Here any velocity values that lie inside the solid should be replaced by the known solid velocity. Entire terms that involve one or more velocity samples "in the air", neither in the solid nor the fluid, should be dropped entirely, even if some of the velocities are fluid velocities. This is a consequence of how the energy is defined (summing just over grid points where the deformation rate is well-defined) and is what gives us the correct free-surface boundary condition.

The equations corresponding to $v_{i,j+1/2,k}$ and $w_{i,j,k+1/2}$ can similarly be derived to complete the system of equations:

$$
\begin{aligned}
&\rho_{i,j+1/2,k}(v_{i,j+1/2,k} - v^P_{i,j+1/2,k}) \\
&\quad + 2\Delta t \mu_{i,j,k} \left(\frac{v_{i,j+1/2,k} - v_{i,j-1/2,k}}{\Delta x^2} \right) \\
&\quad - 2\Delta t \mu_{i,j+1,k} \left(\frac{v_{i,j+3/2,k} - v_{i,j+1/2,k}}{\Delta x^2} \right) \\
&\quad + \tfrac{\Delta t \mu_{i-1/2,j+1/2,k}}{2} \left(\frac{u_{i-1/2,j+1,k} - u_{i-1/2,j,k} + v_{i,j+1/2,k} - v_{i-1,j+1/2,k}}{\Delta x^2} \right) \\
&\quad - \tfrac{\Delta t \mu_{i+1/2,j+1/2,k}}{2} \left(\frac{u_{i+1/2,j+1,k} - u_{i+1/2,j,k} + v_{i+1,j+1/2,k} - v_{i,j+1/2,k}}{\Delta x^2} \right) \\
&\quad + \tfrac{\Delta t \mu_{i,j+1/2,k-1/2}}{2} \left(\frac{v_{i,j+1/2,k} - v_{i,j+1/2,k-1} + w_{i,j+1,k-1/2} - w_{i,j,k-1/2}}{\Delta x^2} \right) \\
&\quad - \tfrac{\Delta t \mu_{i,j+1/2,k+1/2}}{2} \left(\frac{v_{i,j+1/2,k+1} - v_{i,j+1/2,k} + w_{i,j+1,k+1/2} - w_{i,j,k+1/2}}{\Delta x^2} \right) \\
&\qquad = 0,
\end{aligned}
$$

$$
\begin{aligned}
&\rho_{i,j,k+1/2}(w_{i,j,k+1/2} - w^P_{i,j,k+1/2}) \\
&\quad + 2\Delta t \mu_{i,j,k} \left(\frac{w_{i,j,k+1/2} - w_{i,j,k-1/2}}{\Delta x^2} \right) \\
&\quad - 2\Delta t \mu_{i,j,k+1} \left(\frac{w_{i,j,k+3/2} - w_{i,j,k+1/2}}{\Delta x^2} \right) \\
&\quad + \frac{\Delta t \mu_{i-1/2,j,k+1/2}}{2} \left(\frac{u_{i-1/2,j,k+1} - u_{i-1/2,j,k} + w_{i,j,k+1/2} - w_{i-1,j,k+1/2}}{\Delta x^2} \right) \\
&\quad - \frac{\Delta t \mu_{i+1/2,j,k+1/2}}{2} \left(\frac{u_{i+1/2,j,k+1} - u_{i+1/2,j,k} + w_{i+1,j,k+1/2} - w_{i,j,k+1/2}}{\Delta x^2} \right) \\
&\quad + \frac{\Delta t \mu_{i,j-1/2,k+1/2}}{2} \left(\frac{v_{i,j-1/2,k+1} - v_{i,j-1/2,k} + w_{i,j,k+1/2} - w_{i,j-1,k+1/2}}{\Delta x^2} \right) \\
&\quad - \frac{\Delta t \mu_{i,j+1/2,k+1/2}}{2} \left(\frac{v_{i,j+1/2,k+1} - v_{i,j+1/2,k} + w_{i,j+1,k+1/2} - w_{i,j,k+1/2}}{\Delta x^2} \right) \\
&\qquad = 0.
\end{aligned}
$$

Rearranging these into sparse matrix and vector form is left to you.

Part III

More Algorithms

11

Turbulence

This chapter takes a look at methods aimed to capture more of the fine-scale swirly motion characteristic of turbulence. This is far from a scientific examination of turbulence, and in fact scientific work on the subject tends to concentrate on averaging or smoothing over the details of turbulent velocity fields—whereas we want to get to those details as cheaply as possible, even if they fall short of true accuracy.

11.1 Vorticity

Our first stop is getting at a precise measurement of the "swirliness" characteristic of turbulent flow. That is, at any point in space, we would like to measure how the fluid is rotating. In Chapter 10 on viscosity we saw how the gradient of the velocity field gives a matrix whose symmetric part measures deformation—independent of rigid body motions. It's not surprising then that what's left over, the skew-symmetric part, gives us information about rotation. (And of course, $\vec{u}$ itself without any derivatives tells us the translational component of the motion.)

Let's take a look at a generic rigid motion velocity field in 3D:

$$\vec{u}(\vec{x}) = \vec{U} + \vec{\Omega} \times \vec{x}.$$

Here $\vec{U}$ is the translation, and $\vec{\Omega}$ is the angular velocity measured around the origin. Let's work out the gradient of this velocity field in three dimensions to see how we can extract the angular velocity:

$$\begin{aligned} \frac{\partial \vec{u}}{\partial \vec{x}} &= \frac{\partial}{\partial \vec{x}} \begin{pmatrix} U_1 + \Omega_2 z - \Omega_3 y \\ U_2 + \Omega_3 x - \Omega_1 z \\ U_3 + \Omega_1 y - \Omega_2 x \end{pmatrix} \\ &= \begin{pmatrix} 0 & -\Omega_3 & \Omega_2 \\ \Omega_3 & 0 & -\Omega_1 \\ -\Omega_2 & \Omega_1 & 0 \end{pmatrix}. \end{aligned}$$

Thus for a rigid body rotation, the gradient has no symmetric part—there's no deformation after all—and the skew-symmetric part lets us read out the components of angular velocity directly.

Take a look at the skew-symmetric part of the velocity gradient (in general, not just for rigid body motions):

$$\frac{1}{2}\left(\frac{\partial \vec{u}}{\partial \vec{x}} - \frac{\partial \vec{u}}{\partial \vec{x}}^T\right) = \frac{1}{2}\begin{pmatrix} 0 & \frac{\partial u}{\partial y} - \frac{\partial v}{\partial x} & \frac{\partial u}{\partial z} - \frac{\partial w}{\partial x} \\ \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} & 0 & \frac{\partial v}{\partial z} - \frac{\partial w}{\partial y} \\ \frac{\partial w}{\partial x} - \frac{\partial u}{\partial z} & \frac{\partial w}{\partial y} - \frac{\partial v}{\partial z} & 0 \end{pmatrix}.$$

Reading off the local measure of angular velocity this represents, just as we saw in the rigid case, we get

$$\vec{\Omega}(\vec{x}) = \frac{1}{2}\left(\frac{\partial w}{\partial y} - \frac{\partial v}{\partial z}, \quad \frac{\partial u}{\partial z} - \frac{\partial w}{\partial x}, \quad \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}\right).$$

This is exactly half the curl of the velocity field.

In fact we define **vorticity** $\vec{\omega}$ to be the curl of the velocity field, which will then be twice the local angular velocity. Again, in three dimensions this is a vector:

$$\begin{aligned} \vec{\omega} &= \nabla \times \vec{u} \\ &= \left(\frac{\partial w}{\partial y} - \frac{\partial v}{\partial z}, \quad \frac{\partial u}{\partial z} - \frac{\partial w}{\partial x}, \quad \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}\right). \end{aligned}$$

In two dimensions it reduces to a scalar:

$$\omega = \nabla \times \vec{u} = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}.$$

This turns out to be one of the most useful "derived" quantities for a fluid flow.

Take the curl of the momentum equation (1.1), assuming constant viscosity:

$$\nabla \times \frac{\partial \vec{u}}{\partial t} + \nabla \times (\vec{u} \cdot \nabla \vec{u}) + \nabla \times \left(\frac{1}{\rho}\nabla p\right) = \nabla \times \vec{g} + \nabla \times \nu \nabla \cdot \nabla \vec{u}.$$

Switching the order of some derivatives, and assuming that density ρ is constant so it can be brought outside the curl for the pressure term, gives

$$\frac{\partial \nabla \times \vec{u}}{\partial t} + \nabla \times (\vec{u} \cdot \nabla \vec{u}) + \frac{1}{\rho}\nabla \times \nabla p = \nabla \times \vec{g} + \nu \nabla \cdot \nabla (\nabla \times \vec{u}).$$

Recalling that the curl of a gradient is automatically zero (see Appendix A for identities such as this) and assuming that $\vec{g}$ is constant or the gradient of some potential, and substituting in vorticity, reduces this to

$$\frac{\partial \vec{\omega}}{\partial t} + \nabla \times (\vec{u} \cdot \nabla \vec{u}) = \nu \nabla \cdot \nabla \vec{\omega}.$$

The advection term can be simplified with some work (and exploiting the divergence-free condition $\nabla \cdot \vec{u} = 0$) to eventually get, in three dimensions:

$$\frac{\partial \vec{\omega}}{\partial t} + \vec{u} \cdot \nabla \vec{\omega} = -\omega \cdot \nabla \vec{u} + \nu \nabla \cdot \nabla \vec{\omega}. \tag{11.1}$$

This is known as the **vorticity equation**, which you can see has the material derivative $D\vec{\omega}/Dt$ on the left-hand side, a viscosity term on the right-hand side, and a new term $\vec{\omega} \cdot \nabla \vec{u}$, which we can write in components as

$$\vec{\omega} \cdot \nabla \vec{u} = \begin{pmatrix} \omega_1 \frac{\partial u}{\partial x} + \omega_2 \frac{\partial v}{\partial x} + \omega_3 \frac{\partial w}{\partial x} \\ \omega_1 \frac{\partial u}{\partial v} + \omega_2 \frac{\partial v}{\partial v} + \omega_3 \frac{\partial w}{\partial v} \\ \omega_1 \frac{\partial u}{\partial z} + \omega_2 \frac{\partial v}{\partial z} + \omega_3 \frac{\partial w}{\partial z} \end{pmatrix}.$$

This term is sometimes called the **vortex-stretching** term from a geometric point of view which we won't get into in this book. In two dimensions, the vorticity equation actually simplifies further: the vortex-stretching term is automatically zero. (This is easy to verify if you think of a 2D flow as being a slice through a 3D flow with u and v constant along the z-direction and $w = 0$.) Here it is in 2D, now written with the material derivative to emphasize the simplicity:

$$\frac{D\omega}{Dt} = \nu \nabla \cdot \nabla \omega. \tag{11.2}$$

In fact, if we are talking about inviscid flow where viscosity is negligible (as we have done throughout this book except in Chapter 10 on highly viscous flow), the 2D vorticity equation reduces simply to $D\omega/Dt = 0$. That is, vorticity doesn't change, but is just advected with the flow.

It turns out you can build an attractive fluid solver based on vorticity, particularly in 2D where the equation is even simpler, though there are decidedly non-trivial complications for boundary conditions and reconstructing the velocity field for advection from vorticity (more on this in Chapter 14). For example, Yaeger et al. [YUM86], Gamito et al. [Gam95], Angelidis et al. [AN05, ANSN06], Park and Kim [PK05], and Elcott et al. [ETK+07] have all taken this route. However, our chief concern now is in what happens to vorticity in our regular fluid solver, based on velocity and pressure.

We already know that we run the risk of numerical dissipation in our Eulerian advection schemes: we saw that for first-order linear interpolation, the error behaves like additional viscosity, and so it should be no surprise that the vorticity of the velocity field similarly gets dissipated. So far we've dealt with this by increasing the sharpness of the interpolation—though even this doesn't fully avoid dissipation—or switching to particles with FLIP. However, this is not the only cause of vorticity dissipation.

The other big source lies in the time-splitting algorithm itself. We mentioned before that our algorithm for separately advecting and then projecting velocity is only first-order accurate in time; it turns out this can be a fairly problematic error when attempting to capture small-scale vortices. As a motivating example, imagine starting with just a 2D rigid rotation of constant-density fluid around the origin, say

$$\vec{u}^0 = (-y, x).$$

Ignoring boundary conditions and body forces, the exact solution of the Navier-Stokes equations, given this initial velocity field, is for $\vec{u}$ to stay constant—the rotation should continue at exactly the same speed. However, if we advance it with our time-splitting algorithm, things go wrong. Even with *perfect* error-free advection, for a time step of $\Delta t = \frac{1}{2}\pi$ which corresponds in this velocity field to a counterclockwise rotation of 90°, we get this intermediate velocity field:

$$\vec{u}^A = (x, y).$$

It no longer has any vorticity (easy to check) and moreover is divergent: our advection step transferred all the energy from rotation to expansion. It's not hard to verify that the pressure solution is

$$p = \frac{x^2 + y^2}{2\pi\rho}.$$

After updating the intermediate velocity field with this pressure, we end up with

$$\vec{u}^{n+1} = 0.$$

Oops! The flow comes to a standstill, never to move again. If we had taken a smaller time step it would have just been slowed down, but only in the limit $\Delta t \to 0$ does it approach the vorticity it should be preserving. (If we had taken a larger time step, the fluid might even have reversed direction and rotated the other way!)

In fact, at least when density is constant, since the curl of a gradient is automatically zero the pressure projection stage can't affect the fluid's vorticity: the damage is already done when we advect velocities. With a little more effort, it's not hard to verify that starting with a rigid rotation of vorticity ω, one time step of perfect advection will change that to vorticity $\omega \cos(\omega \Delta t/2)$. If Δt is small enough, this is approximately

$$\omega_{n+1} \approx \left(1 - \frac{\omega_n^2 \Delta t^2}{8}\right) \omega_n.$$

Thus our next step is to look at a way of adding back some of the missing vorticity.

11.2 Vorticity Confinement

The **vorticity confinement** technique developed by Steinhoff and Underhill [SU94] is a modification of the Navier-Stokes equations by a term that tries to preserve vorticity. Fedkiw et al. [FSJ01] introduced it to graphics, introducing a Δx factor so that in the limit (as the grid is refined) the term disappears and we get back the true fluid solution. The underlying idea is to detect where vortices are located and add a body force to boost the rotational motion around each vortex.

In this context, a vortex is loosely speaking a peak in the vorticity field, a place that's spinning faster than all the fluid nearby. We can construct unit vectors $\vec{N}$ that point to these maximum points simply by normalizing the gradient of $\|\vec{\omega}\|$:

$$\vec{N} = \frac{\nabla\|\vec{\omega}\|}{\|\nabla\|\vec{\omega}\|\|}.$$

Now $\vec{N}$ points toward the center of rotation of a vortex, and $\vec{\omega}$ itself points along the axis of rotation, so to get a force vector that increases the rotation, we just take a cross-product:

$$f_{\text{conf}} = \epsilon\Delta x(\vec{N} \times \vec{\omega}).$$

The ϵ here is a parameter that can be adjusted to control the effect of vorticity confinement. The Δx factor, as mentioned above, makes this physically consistent: as we refine the grid and Δx tends to zero, the erroneous numerical dissipation of vorticity also tends to zero, so our fix should too.

Let's step through numerically implementing this. We begin by averaging velocities from the MAC grid to the cell centers (as discussed in Chapter 2) and then use central derivatives to approximate the vorticity:[1]

$$\vec{\omega}_{i,j,k} = \left(\frac{w_{i,j+1,k} - w_{i,j-1,k}}{2\Delta x} - \frac{v_{i,j,k+1} - v_{i,j,k-1}}{2\Delta x}, \right.$$
$$\frac{u_{i,j,k+1} - u_{i,j,k-1}}{2\Delta x} - \frac{w_{i+1,j,k} - w_{i-1,j,k}}{2\Delta x},$$
$$\left. \frac{v_{i+1,j,k} - v_{i-1,j,k}}{2\Delta x} - \frac{u_{i,j+1,k} - u_{i,j-1,k}}{2\Delta x} \right).$$

The gradient of $\|\vec{\omega}\|$ is similarly estimated with central differences at the

[1] Note that the null-space problem we discussed earlier isn't particularly alarming here: we just will lack the ability to "see" and boost the very smallest vortices. We still get the benefit of boosting slightly larger ones.

grid cell centers, for use in defining $\vec{N}$:

$$\nabla\|\vec{\omega}\|_{i,j,k} = \left(\frac{\|\vec{\omega}|_{i+1,j,k} - \|\vec{\omega}|_{i-1,j,k}}{2\Delta x}, \frac{\|\vec{\omega}|_{i,j+1,k} - \|\vec{\omega}|_{i,j-1,k}}{2\Delta x}, \frac{\|\vec{\omega}|_{i,j,k+1} - \|\vec{\omega}|_{i,j,k-1}}{2\Delta x}\right).$$

When we normalize this to get $\vec{N}$, we should of course guard against a divide-by-zero by using, for example,

$$\vec{N}_{i,j,k} = \frac{\nabla\|\vec{\omega}\|_{i,j,k}}{\|\nabla\|\vec{\omega}\|_{i,j,k}\| + 10^{-20}M},$$

where M is a characteristic value of units $\mathrm{m}^{-1}\mathrm{s}^{-1}$ for the simulation—nothing to be too concerned about; $M = 1/(\Delta x \Delta t)$ is fine just to make sure this scales properly. Finally, we take the cross-product to get f_{conf} at the grid cell centers; we can take the appropriate averages to apply this to the different components of velocity on the MAC grid.

Ideally we would connect the confinement parameter ϵ with the expected numerical dissipation of vorticity. However, this has yet to be done, but in the meantime serves as another tweakable parameter for the simulation. If set too high, the simulation can go quasi-unstable, reducing the velocity field to essentially just random chaos; more moderate values encourage fine-scale vortices and keep the flow more lively.

Finding a single number which restores some of the lost vorticity where we feel the flow is too boring, but which doesn't break up coherent features like smoke rings which we like, is tricky. Often we want a more localized way of injecting apparent turbulence into only some regions. Selle et al. [SRF05] provide just such a tool. The core of their idea is to seed extra "spin particles"[2] in regions where we want turbulence, which are advected with the flow, and whose strength is added into a local per-grid-cell vorticity confinement parameter.

11.3 Procedural Turbulence

Ultimately, there is a practical limit to the resolution a fluid simulator can run at. For an $n \times n \times n$ grid, we obviously require $O(n^3)$ memory—and the hidden constant is a bit hefty, as you can see when you add up all the additional arrays needed for time integration, pressure solves, etc. Furthermore, if we keep Δt proportional to Δx as recommended, and use the MICCG(0)

[2]Selle et al. call these vortex particles, but this is a bit confusing compared to the usual meaning of vortex particles, coming up in Chapter 14, so I prefer to call them spin particles.

linear solver developed in Chapter 5 which requires $O(\sqrt{n})$ iterations to converge, we end up with the total cost of a simulation scaling like $O(n^{4.5})$. That puts a pretty severe bottleneck on going to higher resolution.

However, real turbulence can show features—vortices—on a huge range of length scales. As a point of reference, for example, turbulence in the atmosphere can span from kilometers down to millimeters. There simply is no practical way to directly simulate with a grid capable of capturing that range ($n \sim 10^5$). However, the turbulent features below a certain length scale tend to lose structure and become isotropic and easily described statistically: if you filter out the large-scale motion of the fluid and zoom in on just the small-scale turbulence, any region looks pretty much like any other region. This is our saving grace. Below the scale of the actual simulation, we can add in procedural models of turbulent velocity fields to fake additional detail. For turbulent smoke, instead of tracking the grid-level smoke concentration field, we instead trace and render millions of marker particles running through this enhanced velocity field (see Rasmussen et al. [RNGF03], for example).

We now take a look at two approaches to generating the required procedural velocity fields. The critical requirements are allowing control over the spectrum of the velocity (i.e., looking at the velocity variation over different length-scales) and making sure the velocity is still divergence-free.

11.3.1 Fourier Synthesis

One of the simpler methods for generating plausible turbulent velocity fields is to do it in Fourier space. If we take the Fourier transform of a velocity field $\vec{u}(\vec{x})$, which we'll assume is periodic over a cube of side length L, we can write it as

$$\vec{u}(\vec{x}) = \sum_{i,j,k=-\infty}^{\infty} \hat{u}_{ijk} e^{\sqrt{-1}2\pi(ix+jy+kz)/L}.$$

Here we're using $\sqrt{-1}$ as a symbol to denote an imaginary number, instead of the more common i or j since this book uses i and j as indices. This Fourier series is also obviously using complex exponentials instead of sines and cosines, which implies the Fourier coefficients $\hat{u}_{ijk}$ may be complex even though $\vec{u}$ is real-valued: this helps to simplify some of the other notation, and more to the point, matches the API of most Fast Fourier Transform packages. Note also that the Fourier coefficients $\hat{u}_{ijk}$ are 3D vectors of complex numbers, not just scalars: you can think of this really being the Fourier transform of u, the separate Fourier transform of v, and the further separate Fourier transform of w all wrapped up into one equation.

In practice, of course, we'll use just a discrete Fourier transform, over an $m \times m \times m$ array of Fourier coefficients. The length L should be chosen

large enough that the periodic tiling isn't too conspicuous, but not too large relative to the simulation grid spacing Δx—after all, we won't be able to afford to take m too large, and we want the Fourier grid spacing L/m (the smallest details we'll procedurally add) to be a lot smaller than Δx.

Shinya and Fournier [SF92] and Stam and Fiume [SF93] introduced to graphics perhaps the simplest physically reasonable turbulence model, the Kolmogorov "5/3-law." This states that for fully developed steady-state turbulence, the kinetic energy contained in all the Fourier modes of spatial frequency around ω should scale like $\omega^{-5/3}$. This means that the (i, j, k) Fourier coefficient (with spatial frequency $\omega = \sqrt{i^2 + j^2 + k^2}$) should have magnitude on the order of

$$\|\hat{u}_{ijk}\| \sim (i^2 + j^2 + k^2)^{-11/12}$$

for i, j, k large enough (the low spatial frequencies are assumed to not belong to the isotropic turbulence regime.) We can take it, in fact, to be a random vector uniformly sampled from a ball of radius $C(i^2+j^2+k^2)^{-11/12}$, for some user-tunable constant C. The cut-off frequency, below which we keep $\hat{u}_{ijk}$ zero, should be on the order of $L/\Delta x$ or more, so that we don't add procedural details at scales that were captured in the simulation.

The divergence of velocity becomes a simple algebraic operation on the Fourier series:

$$\nabla\vec{u}(\vec{x}) = \sum_{i,j,k=-\infty}^{\infty} \frac{\sqrt{-1}2\pi}{L} \left[(i, j, k) \cdot \hat{u}_{ijk}\right] e^{\sqrt{-1}2\pi(ix+jy+kz)/L}.$$

Therefore, $\nabla \cdot \vec{u} = 0$ is equivalent to requiring that each Fourier coefficient $\hat{u}_{ijk}$ is perpendicular to its **wave vector** (i, j, k). Making a velocity field divergence-free then simplifies to just fixing each coefficient individually, subtracting off their component in the radial direction—a simple 3D projection.

Finally, once the Fourier coefficients have been determined, an inverse FFT can be applied on each of the u-, $v-$, and w-components to get a grid of plausible velocities. Note that the velocity vectors are sampled at the grid points, all components together—this is not a staggered MAC grid. Trilinear interpolation can be used between grid points for particle advection.

To animate this velocity field in time, the simplest technique (proposed by Rasmussen et al. [RNGF03]) is just to construct two such velocity fields and then cross-fade back and forth between them. The key observation is that while on its own this method of animation falls short of plausibility (and for that matter, the periodicity of the field is objectionable too), this only is added on top of an already detailed simulation. The extra detail is just needed to break up the smooth interpolation between simulation grid points, not to behave perfectly.

11.3.2 Noise

While Fourier synthesis has many advantages—it's fairly efficient and has a nice theoretical background—it has a few problems too, chief among them being the problem of how to control it in space. If you want the turbulence to be stronger in one region than another, or to properly handle a solid wall somewhere in the flow, simultaneously meeting the divergence-free constraint becomes difficult.

An alternative is to forget about Fourier transforms and instead directly construct divergence-free velocity fields from building blocks such as Perlin noise. We get the divergence-free condition by exploiting vector calculus identities. For example, the divergence of the curl of a vector field is always zero:

$$\nabla \cdot (\nabla \times \vec{\psi}) = 0 \quad \text{for all vector fields } \vec{\psi}$$

and the cross-product of two gradients is always divergence free as well:

$$\nabla \cdot (\nabla\phi \times \nabla\psi) = 0 \quad \text{for all scalar fields } \phi \text{ and } \psi.$$

Kniss and Hart [KH04] and Bridson et al. [BHN07] used the first of these formulas, with $\vec{\psi}$ a vector-valued noise function, and DeWolf [DeW05] used the second with ϕ and possibly ψ scalar noise functions (see also von Funck et al. [vFTS06] for an application of this identity in geometric modeling).

To get full turbulence, several octaves of noise can be added in either formula, with an appropriate power-law scaling of magnitudes. For example, using the first formula, **curl-noise**, we might take for the vector potential

$$\vec{\psi}(\vec{x}) = \sum_{p=1}^{m} A_p \vec{N}\left(\frac{C2^p\vec{x}}{\Delta x}\right)$$

and then get the velocity field as

$$\vec{u} = \nabla \times \vec{\psi}.$$

The curl here can be approximated with finite differences for convenience, rather than evaluated exactly. The simulation grid Δx appears here to emphasize that this should be done only for length scales below the simulation.

In addition, the amplitude A_p of each octave of noise can be modulated in space, allowing full control over where turbulence should appear. Bridson et al. [BHN07] in addition show that ramping A_p down to zero at the boundary of a solid causes the velocity field to meet the solid wall boundary condition.

All of these noise formulations can be animated in time, either using 4D noise functions or, more intriguingly, the FlowNoise method of Perlin and Neyret [PN01].

11.4 Simulating Sub-Grid Turbulence

The previous section gives us some tools to add extra turbulent detail beyond the resolution of the core simulation. However, it relies on artistic intervention to decide how much to add. With Fourier synthesis, it is added globally which only really helps if all of the domain should have uniform extra turbulent noise and solid boundaries aren't important. With curl-noise the artist has very detailed control, allowing them to modulate the amplitude of noise, include different scales in varying amounts, and respect solid boundaries with control over the bandwidth of the their effect—but all this control can be overwhelming! Ideally we want a way to automatically modulate the curl-noise, simulating the turbulence we expect should be present in a flow beneath the grid resolution.

This is by no means a solved problem, and in fact touches on a very hard topic in computational fluid dynamics, Large Eddy Simulation (LES), which attempts to model the effect of unresolved small turbulent scales on the fluid simulated on a grid. At this point in time, we just don't know how to do this, and for the most part rely on heuristic formulas backed up by experimental validation, both measured in real fluids and also tested with extremely high resolution, fully resolved computer simulations which go by the name "Direct Numerical Simulation" (DNS).

The two main ideas in sub-grid turbulence modeling are:

- nonlinear effects in the Navier-Stokes equations, like vortex-stretching, cause small grid-scale vortices to evolve at least partly into even smaller vortices below the scale of the grid;
- turbulence below the scale of the grid mostly effects grid-scale features by mixing them up in space, like viscosity would, but at a rate derived from the amplitude of turbulence which is much larger than the fluid's true viscosity.

The last point is sometimes called **turbulent mixing**, and modeling its effect on the grid by a viscosity-like term leads to something called **effective viscosity**. Unfortunately the truth is that while these two effects are important, turbulence can also work the other way: sub-grid vortices can evolve into grid-scale features, and turbulent mixing doesn't always diffuse grid-scale features like viscosity would. However, engineers and scientists regularly get good-enough results with these sorts of subgrid models.

In graphics, there have been a few attempts at simulating turbulence with this. At the simplest end of the spectrum, Kim et al. [KTJG08] modulate several octaves of wavelet-noise-based curl-noise with amplitudes scaled through the octaves like the Kolmogorov fully-developed turbulence spectrum (used above for Fourier synthesis). The amplitudes are chosen

automatically by estimating the rate of transfer of energy from the grid-scale simulation into sub-grid modes. Other researchers have looked at tracking and evolving the sub-grid energy as extra fields in the simulation (e.g. [SB08, NSCL08, PTC+10]), drawing inspiration from $k - \epsilon$ models in computational fluid dynamics [JL72, LS74]. A good book (albeit not for the faint of engineering heart) to get up to speed on turbulence is Pope's *Turbulent Flows* [Pop00]. This is an ongoing area of research, both in CFD and graphics; there's a lot to read, a lot still to do, and this is where we will leave it.

12

Shallow Water

In this chapter and the next we will turn to special cases of water simulation that allow much faster and simpler algorithms. In both cases, we will use the simplifying assumption that the water surface can be represented as a **height field** $y = h(x, z)$: the water region is all of the points where $y < h(x, z)$, excluding solids. The most important solid is of course the bottom, which we also represent as a height field $y = b(x, z)$, giving a water region defined by

$$b(x, z) < y < h(x, z),$$

and thus the water depth is $d(x, z) = h(x, z) - b(x, z)$. We will actually use depth d as a primary simulation variable, reconstructing the height $h = b+d$ as needed. This geometric simplification rules out many interesting effects such as convincing splashes, breaking waves, droplets or sprays, but still allows many interesting wave motions.[1] For the purposes of the book, we'll also restrict the bottom to be stationary—$b(x, z)$ remains constant—though allowing it to move is a fairly easy generalization if you follow the modeling steps in this chapter.

For the height field assumption to remain a good approximation for the water throughout the simulation, we also need to restrict our attention to height fields that aren't too steep and velocities which aren't too extreme: mainly we will be looking at fairly calm waves. For example, a tall column of water can be represented with a height field in the first frame, but when it starts to collapse it is almost bound to start splashing around in more general ways that will rule it out.

While you can of course use the height field representation to track the water surface in conjunction with a full three-dimensional solver as detailed earlier in the book—see Foster and Metaxas [FM96]—we'll make some further approximations to reduce the complexity of the equations. In this chapter we'll look at the case where the water is shallow, i.e., the depth $d = h - b$ is very small compared to the horizontal length scale of waves

[1] Many authors have worked out ways to bring back some of these features, usually by way of adding a particle system for the extra effects. For example, see the articles by O'Brien and Hodgins [OH95] and Thürey et al. [TMSG07].

or other flow features, and in the next chapter we'll instead consider the water very deep relative to this scale.

12.1 Deriving the Shallow Water Equations

12.1.1 Assumptions

The shallowness assumption means essentially that we can ignore vertical variations in the velocity field: the fluid doesn't have "room" for vertical features like vortices stacked on top of each other. We'll then just track the **depth-averaged** horizontal velocities, $u(x, z)$ and $w(x, z)$, which are the average of u and w for y varying along the depth of the water. For the inviscid flow we're modeling in this chapter, you can think of u and w as constant along y; for viscous flow a better model would be that u and w vary linearly from zero at the bottom to some maximum velocity at the free surface.

Just as an interjection: the process of depth averaging is used in many more contexts than the one here. For example, avalanches have been modeled this way (the snow layer is thin compared to the extent of the mountainside it flows along) as well as large-scale weather patterns (the atmosphere and oceans are extremely thin compared to the circumference of the Earth). The resulting systems of equations are still commonly called "shallow water equations," even if referring to fluids other than water.

The other fundamental simplification we'll make is assuming hydrostatic pressure. That is, if we look at the vertical component of the momentum equation

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + w\frac{\partial v}{\partial z} + \frac{1}{\rho}\frac{\partial p}{\partial y} = -g,$$

(where $g \approx 9.81\text{m/s}$ is the magnitude of acceleration due to gravity) we will assume that the dominant terms are the pressure gradient and gravity, with the rest much smaller. This is consistent with the requirement that the water is shallow and relatively calm, with accelerations in the fluid much smaller than g. Dropping the small terms gives the equation for hydrostatic pressure:

$$\frac{1}{\rho}\frac{\partial p}{\partial y} = -g.$$

Combining that with the free surface boundary condition $p = 0$ at $y = h$ gives

$$p(x, y, z, t) = \rho g \left(h(x, z, t) - y \right). \tag{12.1}$$

Again, this isn't strictly true if the water is moving, but it's a good approximation. The fact that we can directly write down the pressure in the shallow water case, as opposed to solving a big linear system for pressure as we had to in fully three-dimensional flow, is one of the key speed-ups.

12.1.2 Velocity

Assuming that u and w are constant along y means $\partial u/\partial y = \partial w/\partial y = 0$, which means the horizontal parts of the momentum equation are reduced to

$$\begin{aligned} \frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + w\frac{\partial u}{\partial z} + \frac{1}{\rho}\frac{\partial p}{\partial x} = 0, \\ \frac{\partial w}{\partial t} + u\frac{\partial w}{\partial x} + w\frac{\partial w}{\partial z} + \frac{1}{\rho}\frac{\partial p}{\partial z} = 0. \end{aligned}$$

This is just two-dimensional advection along with the horizontal parts of the pressure gradient. Note that although pressure varies linearly in y, the horizontal components of its gradient are in fact constant in y; substituting in Equation (12.1) gives

$$\begin{aligned} \frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + w\frac{\partial u}{\partial z} + g\frac{\partial h}{\partial x} = 0, \\ \frac{\partial w}{\partial t} + u\frac{\partial w}{\partial x} + w\frac{\partial w}{\partial z} + g\frac{\partial h}{\partial z} = 0. \end{aligned} \tag{12.2}$$

That is, the horizontal velocity components are advected in the plane as usual, with an additional acceleration proportional to gravity that pulls water down from higher regions to lower regions.

What about vertical velocity v? It turns out this is fully determined from the "primary" shallow water variables (u, w and d) that we will be simulating. We won't actually need v in the simulation, unless for some reason you need to evaluate it for, say, particle advection in the flow, but it will come in handy to figure out how the surface height evolves in a moment.

First take a look at the incompressibility condition:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0 \tag{12.3}$$

$$\Leftrightarrow \quad \frac{\partial v}{\partial y} = -\frac{\partial u}{\partial x} - \frac{\partial w}{\partial z}. \tag{12.4}$$

The right-hand side of this equation doesn't depend on y, so $\partial v/\partial y$ must be a constant along the y-direction too—which implies v has to be a linear function of y. It's fully determined from its value at the bottom $y = b(x, z)$ and the gradient we just derived.

The bottom velocity comes from the boundary condition $\vec{u} \cdot \hat{n} = 0$, remembering again that we're assuming the bottom is stationary. Recalling some basic calculus, the normal at the bottom is proportional to $(-\partial b/\partial x, 1, -\partial b/\partial z)$, so at the bottom $y = b(x, z)$:

$$\begin{aligned} & -u\frac{\partial b}{\partial x} + v - w\frac{\partial b}{\partial z} = 0 \\ \Leftrightarrow \quad & v = u\frac{\partial b}{\partial x} + w\frac{\partial b}{\partial z}. \end{aligned}$$

Note that if the bottom is flat, so the partial derivatives of b are zero, this reduces to $v = 0$ as expected. Combined with Equation (12.3) we get the following vertical velocity at any point in the fluid:

$$v(x, y, z, t) = u\frac{\partial b}{\partial x} + w\frac{\partial b}{\partial z} - \left(\frac{\partial u}{\partial x} + \frac{\partial w}{\partial z}\right)(y - b). \tag{12.5}$$

In other words, for shallow water we take v to be whatever it requires for the flow to be incompressible and to satisfy the bottom solid boundary condition, given the horizontal velocity.

12.1.3 Height

We can also describe the vertical velocity at the free surface, in a different way. Note that the function $\phi(x, y, z) = y - h(x, z)$ implicit defines the free surface as its zero isocontour—similar to how we tracked general liquid surfaces back in Chapter 8. We know that the free surface, i.e., the zero isocontour, moves with the velocity of the fluid, and so ϕ should satisfy an advection equation

$$\begin{aligned} & \frac{D\phi}{Dt} = 0 \\ \Leftrightarrow \quad & \frac{\partial \phi}{\partial t} + u\frac{\partial \phi}{\partial x} + v\frac{\partial \phi}{\partial y} + w\frac{\partial \phi}{\partial z} = 0 \\ \Leftrightarrow \quad & -\frac{\partial h}{\partial t} + u\left(-\frac{\partial h}{\partial x}\right) + v(1) + w\left(-\frac{\partial h}{\partial z}\right) = 0 \end{aligned}$$

at least at the surface $y = h$ itself. Plugging in what we derived for the velocity in Equation (12.5) at $y = h$ gives us an equation for the rate of

change of height:

$$\begin{aligned} -\frac{\partial h}{\partial t} - u\frac{\partial h}{\partial x} + \left[u\frac{\partial b}{\partial x} + w\frac{\partial b}{\partial z} - \left(\frac{\partial u}{\partial x} + \frac{\partial w}{\partial z}\right)(h-b)\right] - w\frac{\partial h}{\partial z} = 0, \\ \frac{\partial h}{\partial t} + u\frac{\partial (h-b)}{\partial x} + w\frac{\partial (h-b)}{\partial z} = -(h-b)\left(\frac{\partial u}{\partial x} + \frac{\partial w}{\partial z}\right). \end{aligned} \tag{12.6}$$

Using the depth $d = h - b$, and remembering that b is stationary, this can be simplified to

$$\frac{\partial d}{\partial t} + u\frac{\partial d}{\partial x} + w\frac{\partial d}{\partial z} = -d\left(\frac{\partial u}{\partial x} + \frac{\partial w}{\partial z}\right). \tag{12.7}$$

That is, the water depth is advected by the horizontal velocity and, in addition, increased or decreased proportional to the depth and the two-dimensional divergence.

We can simplify Equation (12.7) even further, putting it into what's called **conservation law form**:

$$\frac{\partial d}{\partial t} + \frac{\partial}{\partial x}(ud) + \frac{\partial}{\partial z}(wd) = 0. \tag{12.8}$$

This can in fact be directly derived from conservation of mass, similar to the approach in Appendix B. It's significant here because it leads to numerical methods that exactly conserve the total volume of water in the system—avoiding the mass-loss problems we saw earlier with three-dimensional free surface flow.[2] However, discretizing this accurately enough (to avoid numerical dissipation) is a topic that lies outside the scope of this book.

12.1.4 Boundary Conditions

Equations (12.2) and (12.7) or (12.8) also need boundary conditions, where the water ends (in the x–z horizontal plane) or the simulation domain ends. The case of a solid wall is simplest: if $\hat{n}$ is the two-dimensional normal to the wall in the x–z plane, then we require

$$(u, w) \cdot \hat{n} = 0.$$

[2]This conservation law form can also be applied in three dimensions, leading to a **volume-of-fluid** or **VOF** simulation that exactly conserves volume as well. However, in three dimensions, VOF techniques have their own share of problems in terms of accurately localizing the surface of the fluid and requiring small time steps.

Of course, for a moving solid wall, this should instead be $(u_{\text{solid}}, w_{\text{solid}}) \cdot \hat{n}$. To maintain that velocity in the normal direction, following the velocity equations (12.2), we also need

$$\left(\frac{\partial h}{\partial x}, \frac{\partial h}{\partial z}\right) \cdot \hat{n} = 0.$$

This also applies at an inflow/outflow boundary, where we pump water in or out of the simulation. The utility of such a boundary may be enhanced by adding a source term to the height equation, directly adding (or subtracting) water in some regions; such source terms are also perfect for modeling vertical sinks or sources of water (such as a drop falling from above, perhaps in a particle system, or a drainage hole).

It's much more difficult dealing with the edge of the simulation domain, if it's assumed that the water continues on past the edge. If you expect all the waves in the system to travel parallel to the edge, it's perfectly reasonable to put an invisible solid wall boundary there. If you determine waves should be entering along one edge, perhaps from a simple sinusoid model (see the next section for how to choose such a wave), you can further specify normal velocity and height. However, if you also expect waves to leave through the edge, things are much, much trickier: solid walls, even if invisible or specifying fancy normal velocities and heights, reflect incoming waves. Determining a **non-reflecting** (or **absorbing**) boundary condition is not at all simple and continues as a subject of research in numerical methods. The usual approach taken is to gradually blend away the simulated velocities and heights with a background field (such as a basic sinusoid wave, or flat water at rest), over the course of many grid cells: if the blend is smooth and gradual enough, reflections should be minimal.

Finally one boundary condition of prime importance for many shallow water simulations is at the **moving contact line**: where the depth of the water drops to zero, such as where the water ends on a beach. In fact, no boundary conditions need to be applied in this case: if desired for a numerical method, the velocity can be extrapolated to the dry land as usual, and the depth is zero ($h = b$).

12.2 The Wave Equation

Before jumping to numerical methods for solving the shallow water equations, it's worth taking a quick look at a further simplification. For very

calm water we can completely neglect the advection terms, leaving us with

$$\begin{aligned}
\frac{\partial u}{\partial t} + g\frac{\partial h}{\partial x} &= 0, \\
\frac{\partial w}{\partial t} + g\frac{\partial h}{\partial z} &= 0, \\
\frac{\partial h}{\partial t} &= -d\left(\frac{\partial u}{\partial x} + \frac{\partial w}{\partial z}\right).
\end{aligned}$$

Divide the height equation through by the depth d and differentiate in time:

$$\frac{\partial}{\partial t}\left(\frac{1}{d}\frac{\partial h}{\partial t}\right) = -\frac{\partial}{\partial x}\frac{\partial u}{\partial t} - \frac{\partial}{\partial z}\frac{\partial w}{\partial t}.$$

Then substitute in the simplified velocity equations to get

$$\frac{\partial}{\partial t}\left(\frac{1}{d}\frac{\partial h}{\partial t}\right) = \frac{\partial}{\partial x}\left(g\frac{\partial h}{\partial x}\right) + \frac{\partial}{\partial z}\left(g\frac{\partial h}{\partial z}\right).$$

Expanding the left-hand side, but further neglecting the quadratic term as being much smaller, gives

$$\frac{\partial^2 h}{\partial t} = gd\nabla \cdot \nabla h,$$

where the Laplacian $\nabla \cdot \nabla$ here is just in two dimensions (x and z). Finally, with the assumption that the depth d in the right-hand side remains near enough constant, this is known as the **wave equation**.

The wave equation also pops up naturally in many other phenomena—elastic waves in solid materials, electromagnetic waves, acoustics (sound waves), and more—and has been well studied. Fourier analysis can provide a full solution, but to keep things simple let's just try a single sinusoid wave.[3] Take a unit-length vector $\hat{k}$ (in two dimensions) which will represent the direction of wave motion; the peaks and troughs of the waves will lie on lines perpendicular to $\hat{k}$. Let λ be the wavelength, A the amplitude, and c the speed of the wave. Putting this all together gives

$$A\sin\left(\frac{2\pi(\hat{k}\cdot(x,z) - ct)}{\lambda}\right).$$

[3]In fact, any wave shape will do—we pick sinusoids simply out of convention and to match up with the ocean wave modeling in the next chapter where sinusoids are critical.

If you plug this in as a possible $h(x, z, t)$ in the wave equation, we get the following equation:

$$-A\frac{4\pi^2 c^2}{\lambda^2}\sin\left(\frac{2\pi(\hat{k}\cdot(x,z)-ct)}{\lambda}\right) = -gdA\frac{4\pi^2}{\lambda^2}\sin\left(\frac{2\pi(\hat{k}\cdot(x,z)-ct)}{\lambda}\right).$$

This reduces to

$$c^2 = gd.$$

In other words, the wave equation has solutions corresponding to waves moving at speed $\sqrt{gd}$. Yuksel et al. [YHK07] directly made a very fast wave solver from this observation, using "wave particles" traveling at this speed which locally change the height of the water.

The key insight to glean from all these simplifications and models is that shallow water waves move at a speed related to the depth: the deeper the water, the faster the waves move. For example as a wave approaches the shore, the depth decreases and the wave slows down. In particular, the front of the wave slows down earlier, and so water from the back of the wave starts to pile up as the wave front slows down. Waves near the shore naturally get bigger and steeper, and if conditions are right, they will eventually crest and overturn. The shallow water equations we've developed in this chapter do contain this feature, though of course the height field assumption breaks down at the point of waves breaking: we won't be able to quite capture that look, but we'll be able to come close.

12.3 Discretization

There are many possibilities for discretizing the shallow water equations, each with its own strengths and weaknesses. You might in particular take a look at Kass and Miller's introduction of the equations to animation [KM90], and Layton and van de Panne's unconditionally stable method [LvdP02]. Here we'll provide a small variation on the Layton and van de Panne method that avoids the need for a linear solver at the expense of having a stability restriction on the time step.

We begin with the two-dimensional staggered MAC grid as usual, storing the velocity components u and w at the appropriate edge midpoints and the depth d at the cell centers. Where needed, the height h is reconstructed from the depth as $h = b + d$. We also use the usual time-splitting approach of handling advection in an initial stage, perhaps with the semi-Lagrangian

method we've been using so far:

$$\begin{aligned} u^A &= \text{advect}(\vec{u}^n, \Delta t, u^n), \\ w^A &= \text{advect}(\vec{u}^n, \Delta t, w^n), \\ d^A &= \text{advect}(\vec{u}^n, \Delta t, d^n). \end{aligned}$$

We then compute the intermediate height field $h^A = b + d^A$ and extrapolate it to non-fluid cells, i.e., setting h equal to the value in the nearest fluid cell. Note that it is important to extrapolate height h, not depth d, as we want to make sure water sitting still on a sloped beach, for example, will remain still. We then update the velocities with the pressure acceleration:

$$\begin{aligned} u^{n+1}_{i+1/2,k} &= u^A_{i+1/2,k} - \Delta t\, g\, \frac{h^A_{i+1,k} - h^A_{i,k}}{\Delta x}, \\ w^{n+1}_{i,k+1/2} &= w^A_{i,k+1/2} - \Delta t\, g\, \frac{h^A_{i,k+1} - h^A_{i,k}}{\Delta x}. \end{aligned}$$

We extrapolate these velocities to non-fluid cells as usual and finally update the depth with the divergence term:

$$d^{n+1}_{i,k} = d^A_{i,k} - \Delta t d^A_{i,k} \left(\frac{u^{n+1}_{i+1/2,k} - u^{n+1}_{i-1/2,k}}{\Delta x} + \frac{w^{n+1}_{i,k+1/2} - w^{n+1}_{i,k-1/2}}{\Delta x} \right).$$

That's all there is to it!

There is a stability time-step restriction here, however. A simple analysis in the same vein as the approximations made in Section 12.2 to get to the wave equation can be made, showing that for stability we require

$$\Delta t \lesssim \frac{\Delta x}{\sqrt{gD}},$$

where D is the maximum depth value in the simulation. For safety a fraction of this quantity, such as 0.2, should be used.

13

Ocean Modeling

Simulating the ocean is an ongoing challenge in computer animation. This chapter will demonstrate a series of simplifications that allow relatively calm ocean surfaces to be efficiently simulated; efficiently handling rough oceans, or large-scale interactions between the ocean and solid objects immersed or floating in it, is still an open research problem. The chief resource in graphics for relatively calm ocean waves is by Tessendorf [Tes04].

The main difficulty in the ocean setting is scale. Essential to the look of waves are both large-scale swells and small ripples, and as we'll see in a moment, to get the relative speeds of these different sizes of waves correct, a simulation needs to take into account the true depth of the water. (In particular, the shallow water model of the previous chapter is completely wrong.) A naïve brute-force approach of just running a 3D fluid simulator like the ones we've looked at so far would result in an excessively and impractically large grid. Therefore we'll take a look at changing the equations themselves.

13.1 Potential Flow

Recall the vorticity equation (11.1) from Chapter 11, and since we're dealing with large-scale water, drop the small viscosity term:

$$\frac{\partial \vec{\omega}}{\partial t} + \vec{u} \cdot \nabla \vec{\omega} = -\omega \cdot \nabla \vec{u}.$$

It's not hard to see that if vorticity starts at exactly zero in a region, it has to stay zero unless modified by boundary conditions. Since the ocean at rest (with zero velocity) has zero vorticity, it's not too much of a stretch to guess that vorticity should stay nearly zero once calm waves have developed, as long as boundaries don't become too important—i.e., away from the shoreline or large objects, and assuming the free surface waves don't get too violent. That is, we will model the ocean as **irrotational**, meaning the vorticity is zero: $\nabla \times \vec{u} = \vec{\omega} = 0$.

A basic theorem of vector calculus tells us that if a smooth vector field has zero curl in a simply-connected region, it must be the gradient of some scalar potential:

$$\vec{u} = \nabla\phi.$$

Note that the ϕ used here has nothing to do with the signed distance function or any other implicit surface function we looked at earlier. Combining this with the incompressibility condition, $\nabla \cdot \vec{u} = 0$, indicates that the potential ϕ must satisfy Laplace's equation:

$$\nabla \cdot \nabla\phi = 0.$$

This is the basis of **potential flow**: instead of solving the full non-linear Navier-Stokes equations, once we know the fluid is irrotational and the region is simply-connected, we only need solve a single linear PDE.

The boundary conditions for potential flow are where it gets interesting. For solid walls the usual $\vec{u} \cdot \hat{n} = \vec{u}_{\text{solid}} \cdot \hat{n}$ condition becomes a constraint on $\nabla\phi \cdot \hat{n}$. Free surfaces, where before we just said $p = 0$, are a bit trickier: pressure doesn't enter into the potential flow equation directly. However, there is a striking resemblance between the PDE for the potential and the PDE for pressure in the projection step: both involve the Laplacian $\nabla \cdot \nabla$. We'll use this as a clue in a moment.

The equation that pressure does appear in is momentum: let's substitute $\vec{u} = \nabla\phi$ into the inviscid momentum equation and see what happens:

$$\frac{\partial \nabla\phi}{\partial t} + (\nabla\phi) \cdot (\nabla\nabla\phi) + \frac{1}{\rho}\nabla\phi = \vec{g}.$$

Exchanging the order of the space and time derivatives in the first term, and assuming ρ is constant so it can be moved inside the gradient in the pressure term, takes us to

$$\nabla\frac{\partial\phi}{\partial t} + (\nabla\phi) \cdot (\nabla\nabla\phi) + \nabla\frac{p}{\rho} = \vec{g}.$$

Seeing a pattern start to form, we can also write the gravitational acceleration as the gradient of the gravity potential, $\vec{g} \cdot \vec{x} = -gy$ where $g = 9.81\ \text{m/s}^2$ and y is height (for concreteness, let's take $y = 0$ at the average sea level).

$$\nabla\frac{\partial\phi}{\partial t} + (\nabla\phi) \cdot (\nabla\nabla\phi) + \nabla\frac{p}{\rho} + \nabla(gy) = 0.$$

Only the advection term is left. Writing it out in component form

$$(\nabla\phi) \cdot (\nabla\nabla\phi) = \begin{pmatrix} \frac{\partial\phi}{\partial x}\frac{\partial^2\phi}{\partial x^2} + \frac{\partial\phi}{\partial y}\frac{\partial^2\phi}{\partial x\partial y} + \frac{\partial\phi}{\partial z}\frac{\partial^2\phi}{\partial x\partial z} \\ \frac{\partial\phi}{\partial x}\frac{\partial^2\phi}{\partial x\partial y} + \frac{\partial\phi}{\partial y}\frac{\partial^2\phi}{\partial y^2} + \frac{\partial\phi}{\partial z}\frac{\partial^2\phi}{\partial y\partial z} \\ \frac{\partial\phi}{\partial x}\frac{\partial^2\phi}{\partial x\partial z} + \frac{\partial\phi}{\partial y}\frac{\partial^2\phi}{\partial y\partial z} + \frac{\partial\phi}{\partial z}\frac{\partial^2\phi}{\partial z^2} \end{pmatrix},$$

and it becomes clear that this is actually the same as

$$(\nabla\phi)\cdot(\nabla\nabla\phi) = \begin{pmatrix} \frac{\partial}{\partial x}\frac{1}{2}\left(\frac{\partial\phi}{\partial x}\right)^2 + \frac{\partial}{\partial x}\frac{1}{2}\left(\frac{\partial\phi}{\partial y}\right)^2 + \frac{\partial}{\partial x}\frac{1}{2}\left(\frac{\partial\phi}{\partial z}\right)^2 \\ \frac{\partial}{\partial y}\frac{1}{2}\left(\frac{\partial\phi}{\partial x}\right)^2 + \frac{\partial}{\partial y}\frac{1}{2}\left(\frac{\partial\phi}{\partial y}\right)^2 + \frac{\partial}{\partial y}\frac{1}{2}\left(\frac{\partial\phi}{\partial z}\right)^2 \\ \frac{\partial}{\partial z}\frac{1}{2}\left(\frac{\partial\phi}{\partial x}\right)^2 + \frac{\partial}{\partial z}\frac{1}{2}\left(\frac{\partial\phi}{\partial y}\right)^2 + \frac{\partial}{\partial z}\frac{1}{2}\left(\frac{\partial\phi}{\partial z}\right)^2 \end{pmatrix}$$

$$= \nabla\left(\tfrac{1}{2}\|\nabla\phi\|^2\right).$$

Using this now brings the momentum equation to

$$\nabla\frac{\partial\phi}{\partial t} + \nabla\left(\tfrac{1}{2}\|\nabla\phi\|^2\right) + \nabla\frac{p}{\rho} + \nabla(gy) = 0$$

$$\Rightarrow \qquad \nabla\left[\frac{\partial\phi}{\partial t} + \left(\tfrac{1}{2}\|\nabla\phi\|^2\right) + \frac{p}{\rho} + gy\right] = 0.$$

The only function whose gradient is everywhere zero is a constant, and since constants added to ϕ (a theoretical abstraction) have no effect on the velocity field (the real physical thing), we can assume that the constant is just zero for simplicity. This gives **Bernoulli's equation**:

$$\frac{\partial\phi}{\partial t} + \left(\tfrac{1}{2}\|\nabla\phi\|^2\right) + \frac{p}{\rho} + gy = 0.$$

You may have already heard of this. For example, in the steady-state case, where $\partial\phi/\partial t = 0$, and after subtracting out the hydrostatic component of pressure, we end up with the pressure variation $\Delta p = -\frac{1}{2}\rho\|\vec{u}\|^2$. Many simple experiments, such as blowing over the top of a sheet of paper held along one edge, verify how fast-moving air can induce a pressure drop which sucks things toward it.[1]

Bernoulli's equation gives us a relationship, admittedly non-linear, between pressure and the potential ϕ. In the interior of the fluid this can be used to get pressure from our solution for ϕ. At a free surface where $p = 0$ is known, we can instead use it as a boundary condition for ϕ:

$$\frac{\partial\phi}{\partial t} + \left(\tfrac{1}{2}\|\nabla\phi\|^2\right) + gy = 0.$$

[1] It also unfortunately figures in a bogus explanation of the lift on an airplane wing, namely that due to the curved shape of the airfoil the air has to go faster over the top than over the bottom to meet at the other side; hence there is lower pressure on the top surface, which gives lift. It's not hard to see this is almost completely wrong: angle of attack is the biggest factor in determining lift, allowing airplanes to fly even when upside-down, and allowing flat fan blades with no fancy curvature to effectively push air around.

Well, almost—this is more of a boundary condition on $\partial\phi/\partial t$, not ϕ itself. But it's not hard to see that as soon as we discretize in time, this will end up as a boundary condition on the new value of ϕ that happens to also depend on old values of ϕ.

13.2 Simplifying Potential Flow for the Ocean

Unfortunately as it stands we still have to solve a three-dimensional PDE for the potential ϕ, and though it's a much simpler linear problem in the interior of the water, it now has a fairly nasty non-linear boundary condition at the free surface. In this section we'll go through a series of simplifications to make it solvable in an efficient way. The critical assumption underlying all the simplifications is that we're only going to look at fairly calm oceans.

The first step is to rule out breaking waves so the geometry of the free surface can be described by a height field, just like the previous chapter:

$$y = h(x, z).$$

Of course h is also a function of time, $h(x, z, t)$, but we'll omit the t to emphasize the dependence on just two of the spatial variables. For a perfectly calm flat ocean we'll take $h(x, z) = 0$; our chief problem will be to solve for h as a function of time. In fact, given the velocity field $\vec{u}$ we know that the free surface should follow it—and in fact viewing the surface as implicit defined as the zero level set of $h(x, z) - y$, we already know the advection equation it should satisfy:

$$\begin{aligned} & \frac{D}{Dt}(h(x,z) - y) = 0 \\ \Rightarrow \quad & \frac{\partial h}{\partial t} + u\frac{\partial h}{\partial x} - v + w\frac{\partial h}{\partial z} = 0 \\ \Leftrightarrow \quad & \frac{\partial h}{\partial t} + (u, w) \cdot \left(\frac{\partial h}{\partial x}, \quad \frac{\partial h}{\partial z}\right) = v. \end{aligned}$$

Just as with shallow water, this looks like a two-dimensional material derivative of height, with vertical velocity v as an additional term.

We'll also make the assumption that the ocean floor is flat, at depth $y = -H$ for some suitably large H. While this is almost certainly false, the effect of variations in the depth will not be apparent for the depths and the wavelengths we're considering.[2] The solid "wall" boundary condition at the bottom, where the normal is now $(0, 1, 0)$, becomes $\partial\phi/\partial y = 0$.

[2]Once you get to truly big waves, tsunamis, variation in ocean depth becomes important: for a tsunami the ocean looks shallow, and so the previous chapter actually

We can now write down the exact set of differential equations we want to solve:

$$\begin{aligned}
&\nabla \cdot \nabla \phi = 0 && \text{for } -H \le y \le h(x,z),\\
&\frac{\partial \phi}{\partial y} = 0 && \text{at } y = -H,\\
&\frac{\partial \phi}{\partial t} + \left(\tfrac{1}{2}\|\nabla\phi\|^2\right) + gh(x,z) = 0 && \text{at } y = h(x,z),\\
&\frac{\partial h}{\partial t} + (u,w) \cdot \left(\frac{\partial h}{\partial x}, \quad \frac{\partial h}{\partial z}\right) = v.
\end{aligned}$$

This is still hard to deal with, thanks to the non-linear terms at the free surface. We will thus use the clever mathematical trick of ignoring them—in effect, assuming that $\vec{u}$ is small enough and h is smooth enough that all the quadratic terms are negligible compared to the others. This cuts them down to

$$\begin{aligned}
&\frac{\partial \phi}{\partial t} = -gh(x,z) && \text{at } y = h(x,z),\\
&\frac{\partial h}{\partial t} = v.
\end{aligned}$$

However, it's still difficult to solve since the location at which we are applying the free surface boundary condition moves according to the solution of h. Assuming that the waves aren't too large, i.e., h is small, we can cheat and instead put the boundary condition at $y = 0$, leading to a new simplified problem:

$$\begin{aligned}
&\nabla \cdot \nabla \phi = 0 && \text{for } -H \le y \le 0,\\
&\frac{\partial \phi}{\partial y} = 0 && \text{at } y = -H,\\
&\frac{\partial \phi}{\partial t} = -gh(x,z) && \text{at } y = 0,\\
&\frac{\partial h}{\partial t} = \frac{\partial \phi}{\partial y} && \text{at } y = 0.
\end{aligned}$$

This now is a perfectly linear PDE on a simple slab $-H \le y \le 0$. (We also swapped in $\partial\phi/\partial y$ for v, to make it clear how h and ϕ are coupled.) One of the great properties we now have is that we can add (superimpose) two solutions to get another solution, which we'll exploit to write the general exact solution as a linear combination of simpler solutions.

provides a better model. However, in the deep ocean these waves are practically invisible, since they tend to have wavelengths of tens or hundreds of kilometers but very small heights on the order of a meter.

We haven't yet touched on boundary conditions along x and z: we've implicitly assumed so far that the ocean is infinite, stretching out forever horizontally. We can actually solve this analytically, using Fourier integrals, but clearly this raises some problems when it comes to implementation on a finite computer. Instead we will assume the ocean is *periodic* in x and z, with some suitably large length L being the period. From a graphics standpoint, L should be large enough that the periodicity is inconspicuous—an $L \times L$ ocean "tile" should probably fill a reasonable fraction of the screen. (We'll later discuss a few other tricks to further obscure periodicity.) Any reasonable periodic function can be represented as a Fourier series, which for computer implementation we'll simply truncate to a finite number of terms.

Before jumping to the full Fourier series, let's take a look at a single Fourier mode. Though it may be a little mind-bending to try and visualize it, we'll actually use a complex exponential for this: ultimately this is more convenient mathematically and corresponds best to what a typical Fast Fourier transform library offers in its API, even though the PDE as we have written it only involves real numbers.

Let's start with a generic Fourier component of the height field:

$$h(x, z, t) = \hat{h}_{ij}(t) e^{\sqrt{-1} 2\pi (ix + jz)/L}.$$

The **Fourier coefficient** is $\hat{h}_{ij}(t)$; the t is included to emphasize that it depends on time, but not on spatial variables. In general $\hat{h}_{ij}(t)$ will be a complex number, even though in the end we'll construct a real-valued height field—more on this in a minute. I use the notation $\sqrt{-1}$ instead of the usual i (for mathematicians) or j (for engineers) since i and j are reserved for integer indices. Speaking of which, the integers i and j are the indices of this Fourier component—they may be negative or zero as well as positive. The vector $(i, j)/L$ gives the spatial frequency, and the vector $\vec{k} = 2\pi(i, j)/L$ is called the **wave vector**. Define the **wave number** k as

$$k = \|\vec{k}\| = \frac{2\pi\sqrt{i^2 + j^2}}{L}.$$

The **wavelength** is $\lambda = 2\pi/k = L/\sqrt{i^2 + j^2}$. As you can probably guess, this corresponds precisely to what we would physically measure as the length of a set of waves that this Fourier mode represents.

We'll now make the guess, which will prove to be correct, that when we plug this in for the height field, the corresponding solution for $\phi(x, y, z, t)$ will be in the following form:

$$\phi(x, y, z, t) = \hat{\phi}_{ij}(t) e^{\sqrt{-1} 2\pi (ix + jz)/L} d_{ij}(y).$$

We don't yet know what the depth function $d_{ij}(y)$ should be. Let's first try this guess in the interior of the domain, where $\nabla \cdot \nabla\phi = 0$ should hold:

$$\begin{aligned}
& \nabla \cdot \nabla\phi = 0 \\
\Leftrightarrow \quad & \frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} + \frac{\partial^2\phi}{\partial z^2} = 0 \\
\Leftrightarrow \quad & \frac{-4\pi^2 i^2}{L^2}\phi + \frac{d_{ij}''}{d_{ij}}\phi + \frac{-4\pi^2 j^2}{L^2}\phi = 0 \\
\Leftrightarrow \quad & \frac{d_{ij}''}{d_{ij}}\phi = k^2\phi \\
\Leftrightarrow \quad & d_{ij}'' = k^2 d_{ij}.
\end{aligned}$$

We now have an ordinary differential equation for d_{ij}, with the general solution being a linear combination of e^{ky} and e^{-ky}. Note that the bottom boundary condition, $\partial\phi/\partial y = 0$ at $y = -H$, reduces to $d_{ij}'(-H) = 0$. Since our guess at ϕ already has a yet-to-be-determined factor $\hat{\phi}_{ij}(t)$ built in, we take

$$d_{ij}(y) = e^{ky} + e^{-2kH}e^{-ky}.$$

With this choice, ϕ now satisfies the Laplace equation in the interior of the fluid and the bottom boundary condition. Let's write out what we have for this Fourier mode so far:

$$\phi(x, y, z, t) = \hat{\phi}_{ij}(t)e^{\sqrt{-1}2\pi(ix+jz)/L}\left(e^{ky} + e^{-2kH}e^{-ky}\right) \tag{13.1}$$

All that's left to determine is the time dependence of the potential $\hat{\phi}_{ij}(t)$ and the height field $\hat{h}_{ij}(t)$, and the only equations we have left are the boundary conditions at $y = 0$: $\partial\phi/\partial t = -gh$ and $\partial h/\partial t = \partial\phi/\partial y$. These two boundary equations at the free surface become (after cancelling out the common $e^{\sqrt{-1}2\pi(ix+jz)/L}$ factor):

$$\begin{aligned}
\frac{\partial\hat{\phi}_{ij}}{\partial t}(1 + e^{-2kH}) &= -g\hat{h}_{ij} \\
\frac{\partial\hat{h}_{ij}}{\partial t} &= \hat{\phi}_{ij}(k - ke^{-2kH})
\end{aligned} \tag{13.2}$$

Now, differentiate the second equation with respect to time, and replace the $\partial\hat{\phi}_{ij}/\partial t$ term with the first equation, to get:

$$\frac{\partial^2\hat{h}_{ij}}{\partial t^2} = -kg\frac{1 - e^{-2kH}}{1 + e^{-2kH}}\hat{h}_{ij} \tag{13.3}$$

This is another simple ordinary differential equation, with general solution consisting of yet more Fourier sinusoids, $e^{\sqrt{-1}\omega_k t}$ and $e^{-\sqrt{-1}\omega_k t}$ where the

wave frequency ω_k (how fast the wave is going up and down, no relation at all to vorticity) is given by:

$$\omega_k = \sqrt{kg\frac{1 - e^{-2kH}}{1 + e^{-2kH}}} \tag{13.4}$$

Before going on to the full solution, and accompanying numerical method, it's instructive to pause a moment and reinterpret this height field solution.

Writing out the one of the components of the height field gives:

$$\begin{aligned} h(x, z, t) &= e^{-\sqrt{-1}\omega_k t} e^{\sqrt{-1}2\pi(ix+jz)/L} \\ &= e^{\sqrt{-1}(\vec{k}\cdot(x,z)-\omega_k t)} \\ &= e^{\sqrt{-1}\vec{k}\cdot[(x,z)-c_k t\hat{k}]} \end{aligned} \tag{13.5}$$

where $\hat{k} = \vec{k}/k$ is the unit-length direction of the wave (normal to the crests and troughs) and c_k is the wave speed, defined as:

$$\begin{aligned} c_k &= \frac{\omega_k}{k} \\ &= \sqrt{\frac{g(1 - e^{-2kH})}{k(1 + e^{-2kH})}} \end{aligned} \tag{13.6}$$

Equation (13.5) tells us that the value at a horizontal position (x, z) and time t is the same as the initial wave form at time t and position $(x, z) - c_k t\hat{k}$. It's a lot like the advection equations we have seen over and over again, only this time it's just the wave moving at that speed, not the individual molecules of water. Equation (13.6) is called the **dispersion relation**, giving the speed of a wave as a function of its wave number k. Remembering that wavelength is inversely proportional to k, the dispersion relation shows that waves of different sizes will travel at different speeds—in particular if they all start off in the same patch of ocean, as time progresses they will disperse apart, hence the name. This fact is probably the most crucial visual element in a convincing ocean: it's what communicates to the audience that there is significant depth below the water, whether or not they know the physics underlying it.

In fact, what we have derived so far is just as valid for shallow water (H small) as deep water (H big). If the waves are shallow, i.e. H is small compared to the wavelength so that kH is small, then asymptotically $c \sim \sqrt{gH}$. That is, the speed depends on gravity and depth but not wavelength, which is exactly what we saw for shallow water in chapter 12. On the other hand, for deep water and moderate sized waves, i.e. where kH is very large,

to a very good approximation we have

$$c_k \approx \sqrt{\frac{g}{k}}, \qquad \omega_k \approx \sqrt{gk} \tag{13.7}$$

which is in fact the simplified formula normally used in the ocean—beyond a certain depth the dependence on H doesn't really matter. This form makes it clear that longer waves (k small) move faster than short waves (k large) in the ocean, which again is a very characteristic look: big swells rushing past underneath slow moving ripples.[3]

13.3 Evaluating the Height Field Solution

We derived the wave speed using only one component of the general solution in time for the height field. You can double check that the other component gives a wave moving at the same speed, but in the opposite direction $-\hat{k}$. This leads to some redundancy with the Fourier mode associated with wave vector $-\vec{k}$, which has the same wave number k, the same wave speed, and the same directions. We'll sort this out now, and also clean up the issue of how to make sure the height field is only real-valued despite all the complex numbers flying around. We now build the general real-valued solution from a collection of real cosine waves:

$$h(x, z, t) = \sum_{i,j} A_{ij} \cos(\vec{k} \cdot (x, z) - \omega_k t + \theta_{ij}) \tag{13.8}$$

Here A_{ij} is the real-valued constant amplitude of the wave, $\vec{k} = 2\pi(i, j)/L$ is the wave vector as before and now points in the direction of the wave's motion, ω_k is the time frequency of the wave from equation (13.7), and θ_{ij} is a constant phase shift. We'll get to picking A_{ij} and θ_{ij} later on in the chapter.

Equation (13.8) can be used directly to evaluate the height field at any point in space and time, but if a lot of waves are involved the summation becomes expensive. However, a cheaper alternative exists by way of the Fast Fourier Transform (FFT). Let $n = 2^m$ be a power of two—this isn't essential as FFT algorithms exist for any n, but typically the transform is fastest for powers of two—and restrict the wave indices to

$$-n/2 + 1 \leq i, j \leq n/2 \tag{13.9}$$

[3]Incidentally, the full spectrum of dispersion is also very easily seen in boat wakes: if you look far enough away from the boat, the wake will have separated into big wavelengths at the leading edge and smaller scale stuff behind.

We thus have an $n \times n$ grid of wave parameters. We'll additionally specify that the constant term is zero, $A_{00} = 0$, as this is not a wave but the average sea level, and to simplify life also zero out the highest positive frequency (which doesn't have a matching negative frequency): $A_{n/2,j} = A_{i,n/2} = 0$. The sum will then actually only be up to $i = n/2 - 1$ and $j = n/2 - 1$.

We'll now show how to evaluate $h(x, z, t)$ for any fixed time t on the $n \times n$ regular grid of locations $0 \le x, z < L$, i.e. where $x_p = pL/n$ and $z_q = qL/n$ for integer indices p and q. This grid of h values can then be fed directly into a renderer. Note that this in fact gives an $L \times L$ tile which can be periodically continued in any direction: we'll talk more about that at the end of the chapter.

The problem is to determine, for a fixed time t and all integer indices $0 \le p, q < n$, the height values as specified by:

$$h_{pq} = \sum_{i=-n/2+1}^{n/2-1} \sum_{j=-n/2+1}^{n/2-1} A_{ij} \cos(\vec{k} \cdot (x_p, z_q) - \omega_k t + \theta_{ij}) \tag{13.10}$$

This is not quite in the form that an FFT code can handle, so we will need to manipulate it a little, first by substituting in the wave vector $\vec{k} = 2\pi(i, j)/L$ and the coordinates of the grid $x_p = pL/n$, $z_q = qL/n$:

$$h_{pq} = \sum_{i=-n/2+1}^{n/2-1} \sum_{j=-n/2+1}^{n/2-1} A_{ij} \cos(2\pi(ip + jq)/n - \omega_k t + \theta_{ij}) \tag{13.11}$$

Next we write the cosine in terms of complex exponentials:

$$\begin{aligned} h_{pq} = & \sum_{i=-n/2+1}^{n/2-1} \sum_{j=-n/2+1}^{n/2-1} \tfrac{1}{2} A_{ij} e^{\sqrt{-1}(2\pi(ip+jq)/n - \omega_k t + \theta_{ij})} \\ & \qquad + \tfrac{1}{2} A_{ij} e^{-\sqrt{-1}(2\pi(ip+jq)/n - \omega_k t + \theta_{ij})} \\ = & \sum_{i=-n/2+1}^{n/2-1} \sum_{j=-n/2+1}^{n/2-1} \tfrac{1}{2} e^{\sqrt{-1}(\theta_{ij} - \omega_k t)} A_{ij} e^{\sqrt{-1}(2\pi(ip+jq)/n)} \\ & \qquad + \tfrac{1}{2} e^{-\sqrt{-1}(\theta_{ij} - \omega_k t)} A_{ij} e^{\sqrt{-1}(2\pi(-ip-jq)/n)} \end{aligned} \tag{13.12}$$

Finally we shuffle terms around in the sum to get

$$\begin{aligned} h_{pq} &= \sum_{i=-n/2+1}^{n/2-1} \sum_{j=-n/2+1}^{n/2-1} \Big[\tfrac{1}{2} e^{\sqrt{-1}(\theta_{ij} - \omega_k t)} A_{ij} \\ &\qquad + \tfrac{1}{2} e^{-\sqrt{-1}(\theta_{-i,-j} - \omega_k t)} A_{-i,-j} \Big] e^{\sqrt{-1}(2\pi(ip+jq)/n)} \\ &= \sum_{i=-n/2+1}^{n/2-1} \sum_{j=-n/2+1}^{n/2-1} Y_{ij}(t) e^{\sqrt{-1}(2\pi(ip+jq)/n)} \end{aligned} \tag{13.13}$$

where the complex Fourier coefficients $Y_{ij}(t)$ are defined as:

$$\begin{aligned} Y_{ij}(t) &= \tfrac{1}{2} e^{\sqrt{-1}(\theta_{ij} - \omega_k t)} A_{ij} + \tfrac{1}{2} e^{-\sqrt{-1}(\theta_{-i,-j} - \omega_k t)} A_{-i,-j} \\ &= \tfrac{1}{2} \left[\cos(\theta_{ij} - \omega_k t) + \sqrt{-1} \sin(\theta_{ij} - \omega_k t) \right] A_{ij} \\ &\quad + \tfrac{1}{2} \left[\cos(\theta_{-i,-j} - \omega_k t) - \sqrt{-1} \sin(\theta_{ij} - \omega_k t) \right] A_{-i,-j} \\ &= \left[\tfrac{1}{2} \cos(\theta_{ij} - \omega_k t) A_{ij} + \tfrac{1}{2} \cos(\theta_{-i,-j} - \omega_k t) A_{-i,-j} \right] \\ &\quad + \sqrt{-1} \left[\tfrac{1}{2} \sin(\theta_{ij} - \omega_k t) A_{ij} - \tfrac{1}{2} \sin(\theta_{-i,-j} - \omega_k t) A_{-i,-j} \right] \end{aligned} \tag{13.14}$$

In the last line it's spelled out in real and imaginary parts. Evaluating Equation (13.13) is exactly what FFT software is designed to do: all you need to do is evaluate $Y_{ij}(t)$ for each i and j and pass in that 2D array of Y values, getting back the heights. The results should be real, up to round-off errors—you can safely ignore the imaginary parts, though a good bug check is to make sure they all are zero or very close to zero. Some FFT libraries allow you to specify that the result should be real-valued, and then allow you to define and pass in only half the Y coefficients: this can certainly be a worthwhile optimization, but the specifics of how to do it vary from library to library.

13.4 Unsimplifying the Model

We made a lot of simplifying assumptions to get to an easily solved fully linear PDE. Unfortunately, the resulting height field solution isn't terribly convincing beyond very small amplitudes. In this section we'll try to boost our solution to look better even at larger amplitudes, by compensating for some of the terms we dropped earlier.

The first order of business is looking at the solution for the potential ϕ that accompanies our height field solution in Equation (13.8). We left this hanging before, rushing on to the height field instead, but ϕ offers some extremely useful information: in particular, $\nabla\phi$ gives us the implied

velocity field. As we'll see in a moment, the plain height field solution is what you get when you ignore horizontal motion, letting the water bob up and down but not side to side; the characteristic look of larger waves, with wide flat troughs and sharper peaks, is largely due to this horizontal motion so we will bring it back.

It's not hard to verify that the potential ϕ which matches the height field in Equation (13.8) is as follows, building on our earlier incomplete form and taking the limit as $H \to \infty$ as we did for the wave speed c_k and time frequency ω_k:

$$\phi(x, y, z, t) = \sum_{i,j} \frac{A_{ij}\omega_k}{k} \sin(\vec{k} \cdot (x, z) - \omega_k t + \theta_{ij}) e^{ky} \tag{13.15}$$

Taking the gradient gives us the complete velocity field, both at the surface ($y \approx 0$) and even far below:

$$\begin{aligned} u(x, y, z, t) &= \frac{\partial \phi}{\partial x} = \sum_{i,j} \frac{A_{ij}\omega_k 2\pi i}{kL} \cos(\vec{k} \cdot (x, z) - \omega_k t + \theta_{ij}) e^{ky} \\ v(x, y, z, t) &= \frac{\partial \phi}{\partial y} = \sum_{i,j} A_{ij}\omega_k \sin(\vec{k} \cdot (x, z) - \omega_k t + \theta_{ij}) e^{ky} \\ w(x, y, z, t) &= \frac{\partial \phi}{\partial z} = \sum_{i,j} \frac{A_{ij}\omega_k 2\pi j}{kL} \cos(\vec{k} \cdot (x, z) - \omega_k t + \theta_{ij}) e^{ky} \end{aligned} \tag{13.16}$$

These formulas are themselves fairly useful if you want velocity vectors at arbitrary points, say to aid in simulating the motion of a small solid in the water.

However we'll go one step further. Imagine tracking a blob of water starting at some initial position. The velocity field implied by a single wave, evaluated at that fixed point in space, is just a periodic sinusoid in time. As long as these velocities are small enough, the particle can't stray too far, so to a good approximation the velocity of the particle itself will be that periodic sinusoid. This means its position, the integral of velocity, will also be a periodic sinusoid: the particle will follow an elliptic orbit round and round as waves pass by. Experimental observation confirms this is a fairly accurate description of the motion; it's not perfect—there is a very small net movement in the direction of the wave propagation, termed **Stokes drift**—but it's pretty good.

Solving for the motion of a blob of water starting at position $\vec{x}_0$, from the simplified equation $d\vec{x}/dt = \vec{u}(\vec{x}_0, t)$, gives this general solution for the

displacement from $\vec{x}_0$:

$$\begin{aligned}
\Delta x &= \sum_{i,j} \frac{-2\pi A_{ij} i}{kL} \sin(\vec{k} \cdot (x_0, z_0) - \omega_k t + \theta_{ij}) e^{k y_0} \\
\Delta y &= \sum_{i,j} A_{ij} \cos(\vec{k} \cdot (x_0, z_0) - \omega_k t + \theta_{ij}) e^{k y_0} \\
\Delta z &= \sum_{i,j} \frac{-2\pi A_{ij} j}{kL} \sin(\vec{k} \cdot (x_0, z_0) - \omega_k t + \theta_{ij}) e^{k y_0}
\end{aligned} \tag{13.17}$$

This displacement field can be evaluated anywhere for $y_0 \leq 0$ to give a particle that's moving with the water should be displaced to at any given time. For example, for a solid floating on the surface of the water you can get its position at any time by plugging in $y_0 = 0$ and its "resting" horizontal x_0 and z_0 coordinates. Objects suspended underneath the surface are the same, just with an exponential reduction of the motion by $e^{k y_0}$ for each component. (Note that the components with large k will be nearly zero deep enough down, so they can be dropped for more efficient evaluation—as you go deeper, only the large wavelength, small wave number k, waves have an effect.)

In fact this displacement field also tells us how to get a more accurate free surface: we use it to deform the $y = 0$ plane. The technical term for this, when applied to enrich a single Fourier component, is a **Gerstner wave**, first introduced to graphics by Fournier and Reeves [FR86]. Our earlier height field solution only included the vertical displacement (notice $h(x, z, t)$ and Δy at $y = 0$ are identical), and now we will add in the matching horizontal displacement. Just as with the height field vertical displacement, we can evaluate this horizontal displacement on a regular grid efficiently with the FFT. Using the same process as we did before with the height field to reduce it the desired complex exponential form, we get

$$\begin{aligned}
\Delta x_{pq} &= \sum_{i=-n/2+1}^{n/2-1} \sum_{j=-n/2+1}^{n/2-1} X_{ij}(t) e^{\sqrt{-1}(2\pi(ip+jq)/n)} \\
\Delta z_{pq} &= \sum_{i=-n/2+1}^{n/2-1} \sum_{j=-n/2+1}^{n/2-1} Z_{ij}(t) e^{\sqrt{-1}(2\pi(ip+jq)/n)}
\end{aligned} \tag{13.18}$$

where the Fourier coefficients are defined from

$$\begin{aligned} X_{ij}(t) = & \left[-\frac{\pi A_{ij} i}{kL} \sin(\theta_{ij} - \omega_k t) + \frac{\pi A_{-i,-j} i}{kL} \sin(\theta_{-i,-j} - \omega_k t) \right] \\ & + \sqrt{-1} \left[\frac{\pi A_{ij} i}{kL} \cos(\theta_{ij} - \omega_k t) + \frac{\pi A_{-i,-j} i}{kL} \cos(\theta_{-i,-j} - \omega_k t) \right] \\ Z_{ij}(t) = & \left[-\frac{\pi A_{ij} j}{kL} \sin(\theta_{ij} - \omega_k t) + \frac{\pi A_{-i,-j} j}{kL} \sin(\theta_{-i,-j} - \omega_k t) \right] \\ & + \sqrt{-1} \left[\frac{\pi A_{ij} j}{kL} \cos(\theta_{ij} - \omega_k t) + \frac{\pi A_{-i,-j} j}{kL} \cos(\theta_{-i,-j} - \omega_k t) \right] \end{aligned} \tag{13.19}$$

or more simply:

$$(X_{ij}(t),\, Z_{ij}(t)) = \sqrt{-1} \frac{\vec{k}}{k} Y_{ij}(t). \tag{13.20}$$

These can be evaluated just like the vertical component Fourier coefficients $Y_{ij}(t)$, and for each component a call to the FFT library will then return that component of the displacement evaluated on a regular grid. Adding this fully 3D displacement to the coordinates of a regular grid at $y = 0$ gives a much more convincing ocean surface. Tessendorf recommends including a tunable choppiness parameter $\lambda \in [0, 1]$ to scale down the horizontal displacement, allowing more range in the look: $\lambda = 0$ gives the soft look of the pure height field solution, and $\lambda = 1$ the choppy full displacement solution. We can go even further, in fact, getting a bit of a wind-blown look by displacing in the horizontal direction of the wind, by a small amount proportional to the height—more tunable parameters.

However, all the approximations we've taken to get to this point aren't entirely self-consistent. If the amplitudes A_{ij} are too large the full displacement field might actually cause self-intersections, giving strange loopy inside-out artifacts. This is fairly easy to detect—if a surface normal is pointing downwards (has a negative y component) anywhere, it's a self-intersection. These spots could be interpreted as points where the waves got so steep and pointed that this solution breaks down, i.e. the wave is breaking; the problem can then be plausible covered up with a procedural foam shader, or used as an emitter for a spray and mist particle system.

13.5 Wave Parameters

We turn now to the selection of the wave parameters, the amplitudes A_{ij} and phase shifts θ_{ij}, which so far have been left as unspecified constants. The phase shifts are straightforward: they have no special significance, and

so each θ_{ij} may be chosen as an independent uniform random number from $[0, 2\pi]$. The amplitudes, however, are a little more interesting.

The first point to make follows from the nature of the solution method itself: if the ratio of amplitude to wavelength is too large, the approximations we made are unjustifiable and the result looks unconvincing—real waves simply don't get that steep, and simulating very rough violent oceans is a continuing research problem beyond this method. Therefore it makes sense to put a limit of, say, $A_{ij} \lesssim O(1/k)$. However, beyond this our physical model doesn't give us much guidance for automatically picking a convincing set of amplitudes; ultimately waves are driven by the wind or by ocean currents which we are not even considering. Tessendorf recommends instead turning to phenomological models garnered from observations, such as the **Phillips spectrum**, which biases waves to align with some chosen wind direction, but there is a lot of freedom to experiment. The FFTs can run fast enough to give interactive feedback even on half-decent grids (say 128^2), allowing you to tune the amplitudes effectively. Horvath recently gave an excellent review of existing models and further synthesized a highly usable model to easily get good looking waves [Hor15].

13.6 Eliminating Periodicity

The ocean model we've defined so far often works just fine for ocean shots. However, it is periodic: if the perspective of a shot allows the audience to see many of these tiles, there is a chance the periodicity will be visible and distracting. One way of overcoming this, which in fact is a simple trick to turn any unstructured periodic texture into a nonperiodic pattern, is to superimpose two repeating tiles of different sizes. That is, add to our $L \times L$ repeating ocean tile another one of dimension $\alpha L \times \alpha L$. (In practice, this means evaluating both tiles, then interpolating values from both onto one master grid.) If α is an irrational number, the sum of the two is nonperiodic. You can see if α is rational, it's not hard to prove the sum *is* periodic, in fact if $\alpha = r/s$ for integers r and s then the period is L times the least common multiple of r and s, divided by s. If α is irrational but very close to a rational number r/s with small integers r and s, the sum will not be exactly periodic but look very close to it, which still might appear objectionable. One of the best choices then is the golden ratio $(\sqrt{5}+1)/2 = 1.61803\ldots$ which (in a sense we will not cover in this book) is as far as possible from small integer fractions.

Other possible techniques involve layering in further effects based on nonperiodic noise. For example, combinations of upwelling currents in the sea and wind gusts above often give the ocean a patchy look, where in some regions there are lots of small ripples to reflect light but in others the

surface is much smoother. This can be modeled procedurally, but here we stop as it lies outside the realm of simulation.

14

Vortex Methods

Chapter 11 began with a discussion of vorticity, and highlighted its importance for turbulent, lively detail, especially in smoke and fire simulations. We will focus just on these scenarios in this chapter, looking at alternative simulation methods which directly work with vorticity. While some of this work is equally applicable to water simulation, experience shows most water simulations don't involve a lot of vorticity, or at least vorticity variation, in the fluid (e.g. the last chapter on ocean waves used the assumption of zero vorticity in the water), and the free surface boundary condition is considerably trickier when working with vorticity. Compared to other topics in this book, this chapter is also a bit less "finished": I'm including some basics on vortex methods because they show incredible promise for smoke, but there are still tricky issues left to work out for practical fluid simulation in graphics.

Readers interested in going further with vortex methods may want to follow up with Chorin's paper which started it all [Cho73], some of the graphics papers on this topic (e.g. [YUM86, GLG95, AN05, PK05, ANSN06, ETK+07, WP10, BKB12, PTG12, ZB14]), the somewhat-harder-to-crack but authoritative book by Cottet and Koumoutsakos [CK08], the SIGGRAPH course notes by Koumoutsakos et al. [KCR08], or Mark Stock's summary [Sto07].

Recall again that vorticity is defined as the curl of velocity,

$$\vec{\omega} = \nabla \times \vec{u},$$

and measures locally how much the flow is rotating. For a rigid body motion, it is exactly twice the angular velocity. It also can be directly extracted from the skew-symmetric part of the velocity gradient $\nabla \vec{u}$.

Slightly differently from Chapter 11, we take the vorticity equation with a body force $\vec{b}$ included:

$$\frac{\partial \vec{\omega}}{\partial t} + \vec{u} \cdot \nabla \vec{\omega} = -\vec{\omega} \cdot \nabla \vec{u} + \nu \nabla \cdot \nabla \vec{\omega} + \nabla \times \vec{b}.$$

If we are going to work primarily with vorticity for smoke, then $\vec{g}$ won't represent gravity but instead buoyancy, which is mostly zero but shows

up at hot sources. Assuming negligible viscosity, the vorticity equation simplifies further,

$$\frac{\partial \vec{\omega}}{\partial t} + \vec{u} \cdot \nabla \vec{\omega} = -\vec{\omega} \cdot \nabla \vec{u} + \nabla \times \vec{b},$$

and in two-dimensions (where vorticity is just a scalar) even further:

$$\frac{D\vec{\omega}}{Dt} = \nabla \times \vec{b}.$$

Particularly given just how simple this last form is—tracking it almost perfectly with particles is nearly trivial—and knowing that vorticity is essential to the look of detailed smoke, it is only natural to ask if we can directly track vorticity as a primary fluid variable.

The major problem to overcome with this, however, is that we still need to know the fluid velocity $\vec{u}$ to move vorticity, not to mention smoke concentration, temperature, etc. Luckily, if we know vorticity (the curl of velocity), it is possible to solve for the velocity whose curl matches that as closely as possible. Let's see how.

14.1 Velocity from Vorticity

As a first attempt, we could consider the definition of vorticity $\vec{\omega} = \nabla \times \vec{u}$ as an equation to solve for velocity given vorticity. However, life is not so simple. First of all, even if we directly discretized this on a grid to form a system of linear equations, the matrix we would be solving with is pretty horrible: PCG definitely can't work. However, more importantly, given an arbitrary velocity field $\vec{\omega}$, which might have resulted from approximately solving the vorticity equation forward in time, there is no guarantee that it is exactly the curl of *any* velocity field. For example, the divergence of the curl of a vector field is always zero (a fact we used in Chapter 11 for curl-noise) so if $\nabla \cdot \vec{\omega} \neq 0$, it cannot be exactly the curl of any velocity field. Unfortunately, most of our methods for solving the vorticity equation forward in time will not exactly keep a zero divergence vorticity field, due to numerical errors.

Instead, we can ask for the velocity field $\vec{u}$ whose curl is as close as possible to the given vorticity ω. Ignoring boundaries for now, and assuming the fields all decay to zero far enough from the origin, we can express this by integrating the squared magnitude of the difference between $\nabla \times \vec{u}$ and $\vec{\omega}$ over all space:

$$\vec{u} = \arg\min_{\vec{u}} \iiint \|\nabla \times \vec{u} - \vec{\omega}\|^2$$

If you followed the calculus of variations derivation in Chapter 10, you can guess what's coming next. We'll use the same approach to get a handle on what equation $\vec{u}$ actually solves.

Suppose $\vec{u}$ is the optimal velocity field to match up against the given vorticity $\vec{\omega}$, and let $\vec{p}$ be any other smooth function which decays to zero far enough from the origin. Then define the regular function of one variable, $g(s)$, as

$$g(s) = \iiint \|\nabla \times (\vec{u} + s\vec{p}) - \vec{\omega}\|^2.$$

Since $\vec{u}$ is optimal, $g(s)$ must have a minimum at $s = 0$. Therefore $g'(0) = 0$.

In fact, $g(s)$ is actually just a quadratic! We can expand it out to see:

$$\begin{aligned} g(s) &= \iiint \Big(\nabla \times (\vec{u} + s\vec{p}) - \vec{\omega}\Big) \cdot \Big(\nabla \times (\vec{u} + s\vec{p}) - \vec{\omega}\Big) \\ &= \iiint \|\nabla \times \vec{u} - \vec{\omega}\|^2 + 2\Big(\nabla \times \vec{u} - \vec{\omega}\Big) \cdot \Big(\nabla \times \vec{p}\Big)s + \|\nabla \times \vec{p}\|^2 s^2 \end{aligned}$$

The condition $g'(0) = 0$ is equivalent to the middle term, the linear coefficient, being zero:

$$\iiint 2\Big(\nabla \times \vec{u} - \vec{\omega}\Big) \cdot \Big(\nabla \times \vec{p}\Big) = 0$$

Dividing by two and using a generalized integration-by-parts, this is equivalent to

$$-\iiint \nabla \times \Big(\nabla \times \vec{u} - \vec{\omega}\Big) \cdot \vec{p} = 0$$

This integral should be exactly zero no matter which smooth vector field $\vec{p}$ (that decays to zero) we use. Therefore, it must be the case that

$$\nabla \times \Big(\nabla \times \vec{u} - \vec{\omega}\Big) = 0$$

everywhere in space. We have an equation for $\vec{u}$ again:

$$\nabla \times \nabla \times \vec{u} = \nabla \times \vec{\omega}.$$

This one should always have a solution.

However, it turns out we still have a problem: there are many different solutions! In fact, if you take any solution $\vec{u}$, and add the gradient of any scalar function ∇f to it, the curl is unchanged, since $\nabla \times \nabla f = 0$. (As a reminder, this and other vector calculus identities are reviewing in Appendix A.) This is actually a fundamental point about reconstructing velocity from vorticity: the answer is not unique. In fact, we will exploit this a little bit later.

For now we really would like to pin down one particular solution among the infinitely many. We can start by rewriting the $\nabla \times \nabla \times$ operator, which a little bit of differentiation reveals is the same as:

$$-\nabla \cdot \nabla \vec{u} + \nabla(\nabla \cdot \vec{u}) = \nabla \times \vec{\omega}.$$

That is, curl-curl is the same thing as the negative Laplacian (applied to each component of $\vec{u}$ separately) plus the gradient of the divergence of $\vec{u}$. Well, we know the velocity field we want at the end should be incompressible, i.e. divergence-free, so the second term should just be zero. (We had better check this later, of course!) Throwing it away, we are left with a much nicer problem:

$$-\nabla \cdot \nabla \vec{u} = \nabla \times \vec{\omega}.$$

This is our dear old Poisson problem yet again! It's actually three independent Poisson problems, one for each component of velocity.

In this case, since we don't have boundaries, we can actually write down the solution by way of the **fundamental solution**. The fundamental solution of the Laplacian is a radially symmetric function Φ whose Laplacian is zero everywhere except at the origin, where it is singular—in fact, so singular it is the Dirac delta δ. Without going through the pain of deriving it, the fundamental solution in 2D is

$$\Phi_2(\vec{x}) = \frac{1}{2\pi} \log \|\vec{x}\|,$$

and in 3D it is

$$\Phi_3(\vec{x}) = -\frac{1}{4\pi \|\vec{x}\|}.$$

You can easily check, albeit with some heavy-duty differentiation, that the Laplacian of each, in their respective dimension, is indeed zero when $\vec{x} \neq 0$. Obviously they are not differentiable in the usual sense at $\vec{x} = 0$: in a fuzzy sense their Laplacian at the origin is so infinitely huge that if you integrated around it you would get the value 1 instead of 0. You can check this relatively easily too, say in 3D, by integrating the fundamental solution over the sphere S of radius one centered at the origin:

$$\iiint_S \nabla \cdot \nabla \Phi_3(\vec{x}).$$

First use the divergence theorem (fundamental theorem of calculus) to convert this into an integral over the surface of the sphere:

$$\iiint_S \nabla \cdot \nabla \Phi_3(\vec{x}) = \iint_{\partial S} \nabla \Phi_3(\vec{x}) \cdot \hat{n}.$$

A little bit of calculation shows the gradient of $\Phi_3(\vec{x})$ is just

$$\nabla\Phi_3(\vec{x}) = \nabla\left(-\frac{1}{4\pi\|\vec{x}\|}\right) = \frac{\vec{x}}{4\pi\|\vec{x}\|^3}.$$

On the surface of the unit sphere, $\|vecx\| = 1$, so this simplifies, and in fact the normal $\hat{n}$ is the same as the position $\vec{x}$. Our integral becomes:

$$\iiint_S \nabla\cdot\nabla\Phi_3(\vec{x}) = \iint_{\partial S} \frac{\vec{x}}{4\pi}\cdot\vec{x}.$$

But $\vec{x}\cdot\vec{x}$ is just $\|\vec{x}\|$ on the surface of the unit sphere, which again is 1. So we have

$$\iiint_S \nabla\cdot\nabla\Phi_3(\vec{x}) = \iint_{\partial S} \frac{1}{4\pi}.$$

The surface area of the unit sphere is 4π, so integrating the constant $1/4\pi$ over it gives us the answer 1:

$$\iiint_S \nabla\cdot\nabla\Phi_3(\vec{x}) = 1.$$

This is exactly what defines the Dirac delta, which technically is a "distribution" rather than a normal function:

$$\nabla\cdot\nabla\Phi(\vec{x}) = \delta(\vec{x})$$

It is zero everywhere except at 0, but it's so singular at the origin that its integral over a region containing the origin is 1.

The fundamental solution is called fundamental because it gives us an easy way to write down the solution for any Poisson problem, modulo boundary conditions. For our problem, that solution is

$$\vec{u}(\vec{x}) = \iiint \Phi(\vec{x}-\vec{p})\nabla\times\vec{\omega}(\vec{p})\,d\vec{p}.$$

That is, the velocity at a point $\vec{x}$ is a weighted integral of $\nabla\times\omega(\vec{p})$ over all of space, where the weight comes from the fundamental solution applied to the vector between them $\vec{x}-\vec{p}$.

Let's check this actually is a solution, by taking the Laplacian with respect to $\vec{x}$:

$$\nabla_x\cdot\nabla_x\vec{u}(\vec{x}) = \nabla_x\cdot\nabla_x\iiint \Phi(\vec{x}-\vec{p})\nabla\times\vec{\omega}(\vec{p})\,d\vec{p}.$$

Differentiating with $\vec{x}$ can be brought inside the integral with $\vec{p}$:

$$\begin{aligned}\nabla_x\cdot\nabla_x\vec{u}(\vec{x}) &= \iiint \nabla_x\cdot\nabla_x\Phi(\vec{x}-\vec{p})\nabla\times\vec{\omega}(\vec{p})\,d\vec{p}\\ &= \iiint \delta(\vec{x}-\vec{p})\nabla\times\vec{\omega}(\vec{p})\,d\vec{p}.\end{aligned}$$

This integral with the Dirac delta is special. Since $\delta(\vec{x}-\vec{p})$ is zero except at $\vec{x}-\vec{p}=0$, i.e. except when $\vec{x}=\vec{p}$, only $\nabla\times\vec{\omega}(\vec{x})$ can contribute to the answer. Since the integral of the Dirac delta over the origin is one, the contribution is direct:

$$\nabla\cdot\nabla\vec{u}(\vec{x}) = \nabla\times\vec{\omega}(\vec{x}).$$

So we know this is the solution we want.

Let's massage the expression just a little bit, using integration-by-parts:

$$\begin{aligned}\vec{u}(\vec{x}) &= \iiint \Phi(\vec{x}-\vec{p})\nabla\times\vec{\omega}(\vec{p})\,d\vec{p}\\ &= -\iiint \nabla_p\Phi(\vec{x}-\vec{p})\times\vec{\omega}(\vec{p})\,d\vec{p}\end{aligned}$$

Now, since $\Phi(\vec{x}-\vec{p}) = \Phi(\vec{p}-\vec{x})$, the gradient with resepct to $\vec{p}$ is the same as the gradient with respect to $\vec{x}$:

$$\begin{aligned}\vec{u}(\vec{x}) &= -\iiint \nabla_x\Phi(\vec{x}-\vec{p})\times\vec{\omega}(\vec{p})\,d\vec{p}\\ &= -\iiint \nabla_x\times\Big(\Phi(\vec{x}-\vec{p})\vec{\omega}(\vec{p})\Big)\,d\vec{p}\\ &= -\nabla_x\times\iiint \Phi(\vec{x}-\vec{p})\vec{\omega}(\vec{p})\,d\vec{p}\\ &= \nabla_x\times\vec{\psi}(\vec{x})\end{aligned}$$

In the last line we introduce a new function $\vec{\psi}$ defined by the integral of the fundamental solution with $\vec{\omega}$. Our velocity is the curl of $\vec{\psi}$ so it has to divergence-free, justifying our step way back at the beginning of this odyssey.

14.2 Biot-Savart and Streamfunctions

Our last little derivation actually breezed through two very important equations. The first is called the Biot-Savart law:

$$\vec{u}(\vec{x}) = \iiint -\nabla\Phi(\vec{x}-\vec{p})\times\vec{\omega}\,d\vec{p}.$$

This gives the velocity at any point in space given the vorticity, or at least (as we saw from our derivation in the last section) the velocity whose curl is as close as possible to the given $\vec{\omega}$ in a least-squares sense.

The other equation expresses the velocity as the curl of a function $\vec{\psi}$. The definition of this function is

$$\vec{\psi}(\vec{x}) = -\iiint \Phi(\vec{x} - \vec{p})\vec{\omega}(\vec{p})\, d\vec{p},$$

which you can recognize, since it's an integral with the fundamental solution, as really saying that $\vec{\psi}$ is the solution to this Poisson problem:

$$\nabla \cdot \nabla\vec{\psi} = -\vec{\omega}.$$

This gives an alternative interpretation of the velocity we reconstructed from $\vec{\omega}$: we first solve a Poisson problem for $\vec{\psi}$, then take its curl to get the velocity. In fact, if $\vec{\omega}$ happened to be divergence-free itself (so it could possibly be the curl of some velocity), a little effort can show that $\nabla\times\nabla\vec{\psi} = \vec{\omega}$ exactly, and likewise $\nabla \times \vec{u} = \vec{\omega}$.

In 2D, this function like the vorticity is actually just a scalar, ψ, and has a special name: the **streamfunction**. The name derives from the fact that its level set isocontours are in fact the streamlines of the velocity field. This isn't hard to see: the curl in 2D is just the gradient rotated by 90° (see Appendix A), and the gradient points 90° out from the level set isocontours, therefore the velocity is parallel to the level sets.

In 3D, there isn't such a nice geometric property for $\vec{\psi}$, but some people still call it the streamfunction. Others prefer to describe it as a vector-valued potential, but streamfunction is easier to say, so I will go with that.

14.3 Vortex Particles

At last, we can arrive at an interesting numerical method. Let's begin in 2D. We know there that the inviscid vorticity equation is especially simple, even more so if we drop buoyancy for now:

$$\frac{D\omega}{Dt} = 0.$$

Solving this with particles is as easy as it gets! Suppose we approximate the vorticity of the flow as being zero everywhere except at a set of points (our particles), where it's concentrated. Each particle at position $\vec{x}_i$ will have vorticity ω_i, and the above equation says the particles should just flow through the velocity field while never changing their vorticity values.

How do we get the velocity? More precisely we will model the vorticity field as a sum of weighted Dirac delta spikes:

$$\omega(\vec{x}) = \sum_{i=1}^{n} \omega_i \delta(\vec{x} - \vec{x}_i).$$

Such singularities can't exist in reality, of course, so think of each particle as a simplifying approximation to a vortex where the total integral of its vorticity is ω_i, concentrated in a small region of space around $\vec{x}_i$.

The Biot-Savart law then allows us to compute the velocity at any point in space (written here in 2D with $\nabla\Phi^\perp$ meaning the 90° rotation of the gradient, in lieu of a 3D cross-product):

$$\vec{u}(\vec{x}) = \iiint -\nabla\Phi(\vec{x}-\vec{p})^\perp \left(\sum_{i=1}^{n} \omega_i \delta(\vec{p}-\vec{x}_i)\right) d\vec{p}.$$

Integrals with Dirac deltas simplify:

$$\vec{u}(\vec{x}) = \sum_{i=1}^{n} -\nabla\Phi(\vec{x}-\vec{x}_i)^\perp \omega_i.$$

We can evaluate this at each particle itself to find the velocity there, with the proviso that $\nabla\Phi(\vec{x}-\vec{x}_i)$ isn't yet defined at $\vec{x}=\vec{x}_i$: we set the value to be zero, so a particle's own vorticity does not influence its velocity.

With this simple formula, we have a 2D smoke simulator of surprising power. It has zero dissipation, zero numerical viscosity: the only numerical error arises from the choice of time integrator to update positions, and rounding error in the calculations. There is also modeling error, of course: real fluids don't have vorticity concentrated in Dirac deltas, and real fluids generally have some nonzero viscosity. It still gives amazingly good results with surprisingly few vortex particles—for a simple 2D smoke simulation, this is a far more effective approach than what we took earlier in the book.

But wait, if it's so good, why did we even bother with solving pressure on grids and all that earlier in the book? Well, there are a few flies in the ointment. For one, we're still in 2D and we also haven't seen how to incorporate solid boundaries, or even buoyancy. The Biot-Savart formula above is also a bit scary: to find the velocity of one particle, we have to sum over all n particles, and therefore the cost of evaluating all velocities is $O(n^2)$, which doesn't scale well. Unlike our solvers up to this point, it turns out the inviscid assumption together with the Dirac deltas takes us into dangerous ill-posed territory: if two particles get close to each other, their velocities can blow up. And finally, tracking vorticity on a few scattered particles doesn't give us detailed smoke concentration, temperature, etc.

14.3.1 Tracking Smoke

Let's start with the easiest issue, the last. Part of the power of vortex particle methods is that relatively few particles are needed to generate very rich motion for a large region of space—in fact, unlike grid methods

with velocity and pressure, by default our simulation is running in all of space without any artificial domain boundaries! However, this does mean the few vortex particles we have are not going to cut it for tracking smoke, temperature, and other related fields. The vortex particles and the Biot-Savart formula can provide us velocity anywhere, however, so we can track other fields with that however we want—on grids with semi-Lagrangian advection, on particles, or with particle-grid hybrids where we splat values down on to the grid for further processing.

14.3.2 Buoyancy

Buoyancy is also simple. Recall the 2D vorticity equation with body force $\vec{b}$ included is

$$\frac{D\vec{\omega}}{Dt} = \nabla \times \vec{b}.$$

We need to estimate the curl of the buoyancy vector field at the particles to see how to update their vorticity. This could be as simple as specifying a buoyancy field and taking its curl. For example, you could specify an upward buoyancy in a region around where smoke is being emitted, possibly with an animated volumetric texture to make it more interesting, modeling the idea that the air will cool off to ambient temperature by the time it leaves that region, more or less. If we are tracking temperature on a grid (or splatting it from particles to a grid), we can also compute buoyancy on the grid and then use finite differences to estimate the curl, interpolating that curl at our vortex particles.

14.3.3 Mollification

Dealing with the blow-up when particles get close is a little more involved. Instead of insisting ω is concentrated in unrealistic Dirac delta spikes, we can instead assume it is spread out more smoothly. Coming up with a smoothed kernel function, like a Gaussian, and then computing the integral with the fundamental solution is usually intractable, but instead we can just directly "mollify" the fundamental solution itself, avoiding the singularity. There's no single correct way to do this. The guiding principle is to keep the asymptotic behavior the same, i.e. the mollified version should converge to the true fundamental solution as you go far away from the origin. This will keep the implied vorticity distribution concentrated around the particle, and it also makes sure that the Laplacian of the mollified version still integrates to 1. For example, one might use

$$\tilde{\Phi}_2(\vec{x}) = \frac{1}{2\pi} \log\left(\|\vec{x}\| + he^{-\|\vec{x}\|/h}\right),$$

where h is a small length which you can think of as the characteristic radius of the particles.

14.3.4 Undoing the inviscid assumption

Accurately simulating viscous diffusion of vorticity is hard with just the particle representation. However, we can at least hack in a very simple dissipation of vorticity. Assume that the sum of all the vorticities stored on the particles is zero, so there isn't a net rotation for the entire gas—indeed it could be useful to check this in the simulation, and if the sum is nonzero, subtract the mean vorticity from each particle to bring it back to zero. A very simple model of viscously transferring vorticity to the rest of the fluid averaged out over all of space is then to use a simple exponential decay. In each time step, reduce the vorticity of each particle by the factor $e^{-k\Delta t}$, where k controls the speed of dissipation.

14.3.5 Jumping to 3D

Switching all of this to three dimensions is straightforward: just swap in the 3D fundamental solution, or a mollified version like

$$\tilde{\Phi}_3(\vec{x}) = -\frac{1}{4\pi(\|\vec{x}\| + he^{-\|\vec{x}\|/h}}.$$

However, the vortex-stretching term in the 3D vorticity equation needs careful treatment:

$$\frac{D\vec{\omega}}{Dt} = -\vec{\omega} \cdot \nabla \vec{u}.$$

Trying to directly evaluate the gradient of velocity and plugging it in is not a good idea—most schemes that start out that way tend to be vulnerable to instability.

It's better to begin by understanding the physical meaning of this term. In 3D, rotations don't just exist around a point, but instead are always around an axis. Likewise, it turns out that vorticity in 3D can never actually be isolated to separate points, but always is arranged in at least "filaments" (curves), if not vortex sheets (surfaces) or spread across volumes. Even if we take isolated vortex particles as our approximate representation, the velocity field produced by Biot-Savart must have a true vorticity which is nonzero along curves though the space. Perhaps a better model to have in mind for a vortex particle in 3D is that it (approximately) represents a little portion of a longer vortex filament, like a segment, aligned in the direction of $\vec{\omega}$. In a sense, it is almost a rotating cylinder of fluid of very small radius, spinning around its axis with angular velocity proportional to $\vec{\omega}$.

The vortex filaments, made up of these tiny rotating cylinders, are advected by the velocity field. The velocity field can deform them, if the velocity field varies along the tangential direction of the filament, as measured by the directional derivative of $\vec{u}$ in the direction of $\vec{\omega}$, namely $\vec{\omega} \cdot \nabla \vec{u}$. In particular, the tangential direction along the filament has to change accordingly, i.e. $\vec{\omega}$ has to change direction.

Thinking again of just one spinning cylinder of fluid, if the velocity field moves its endpoints at different speeds, the cylinder can change direction and/or change its length. If it gets stretched longer, it has to get thinner (since the flow is incompressible, conserving volume). If it gets thinner, it has to spin faster, by conservation of angular momentum—just like ice skaters will spin faster if they bring their limbs in close to their body.

The vortex stretching term exactly accounts for both of these effects: the axis of vorticity changing direction as the velocity field deforms the vortex filaments, and the magnitude of vorticity changing to preserve angular momentum if the filament is stretched or compressed in length.

This suggests that perhaps instead of vortex particles, for 3D we should instead use vortex filaments or vortex sheets, whose deformation under the flow can be tracked exactly, and using conservation of angular momentum and their geometry to avoid needing to directly approximate the vortex-stretching term—and indeed, many graphics papers do just this. However, there are then heavier geometric problems.

If we stick to plain vortex particles it's still not clear what the best way to handle vortex-stretching is. Geometric approaches which take into account exactly how the time integrator would deform a cylinder lined up with $\vec{\omega}$, then appealing to conservation of angular momentum, are probably the best bet.

14.3.6 Speeding up Biot-Savart

For a very detailed simulation with tens of thousands of vortex particles, or more, and perhaps also orders of magnitude more smoke particles to track as well for rendering, the cost of the Biot-Savart summation over particles is huge. The main saving grace is that it at least is embarrassingly parallel: evaluating the velocity at one point in space can be computed completely independently of any other point. However, it's still worrisome.

There are several algorithms available to speed this up. Two of the most famous, in the "tree code" family, are the Barnes-Hut algorithm [BH86] and the Fast Multipole Method [GR87]. These are complicated algorithms, particularly the latter, and I won't dive into a full explanation here. But let's look at an outline of these methods.

Suppose k of the particles, $\vec{x}_1, \ldots, \vec{x}_k$, are in a cluster. For example, they all lie inside a sphere of radius R with center $\vec{x}_C$. Look at evaluting the

velocity at some point $\vec{x}$ far away from the cluster, i.e. with $\|\vec{x} - \vec{x}_C\| \gg R$. The regular Biot-Savart formula would include their contribution with this sum:

$$\sum_{i=1}^{k} -\nabla\Phi(\vec{x} - \vec{x}_i) \times \vec{\omega}_i.$$

However, $\nabla\Phi$ decays rapidly and smoothly to zero at distances like this, and is basically almost the same for all of the particles in this sum. Therefore we would not commit very much error if we changed the sum to just use the center point of the cluster:

$$\sum_{i=1}^{k} -\nabla\Phi(\vec{x} - \vec{x}_C) \times \vec{\omega}_i.$$

In fact, a little math with a Taylor series can give us a good bound of the worst-case error involved, but I will spare you that detour. The main thing is that we can now factor out the fundamental solution from the sum like this:

$$-\nabla\Phi(\vec{x} - \vec{x}_C) \times \left(\sum_{i=1}^{k} \vec{\omega}_i \right).$$

If we compute and store the sum of vorticity in the cluster just once, then finding the cluster's contribution to velocity far away drops to $O(1)$ time from $O(k)$ time.

This idea can be taken several steps further, for example approximating the contribution of one cluster on another well-separated cluster all in one go, by considering just the separation of the cluster centers, or using a more accurate approximation (akin to taking more terms in a Taylor series).

The next big idea is to construct a bounding-volume hierarchy around the particles, for example using octrees or kd-trees to divide up the space and cluster nearby particles together in the tree. Each node in the tree then becomes a candidate cluster. We can efficiently compute the vorticity sums and so forth in each tree node (for the particles contained in that branch of the tree) bottom-up from the leaves. Then when we need to evaluate velocity, we can traverse the tree starting at the root, and stop traversal as soon as we find a cluster that is tight enough and far enough away from the evaluation point that we can get a good approximation without descending further.

14.3.7 Solid Boundaries

Finally we turn to solid boundaries. There are actually two solid boundary conditions we have seen so far:

- no-stick $\vec{u} \cdot \hat{n} = \vec{u}_{\text{solid}} \cdot \hat{n}$, and
- no-slip $\vec{u} = \vec{u}_{\text{solid}}$.

The first is natural for inviscid flow, the second for viscous flow.

Let's look at no-stick first, because it is a bit simpler. The key thing to remember is that when we reconstructed velocity from vorticity, you may recall we computed only one of infinitely many possible velocities whose curls are as close as possible to a given $\vec{\omega}$. Call that choice, from the Biot-Savart law, $\vec{u}_0$. For any scalar field ϕ, the velocity field

$$\vec{u} = \vec{u}_0 + \nabla\phi$$

will have exactly the same curl, so is just as good a solution. We can use this freedom to enforce no-stick boundaries with the right choice of ϕ.[1]

As we saw, the Biot-Savart velocity $\vec{u}_0$ is automatically divergence-free, because it is the curl of the streamfunction. Even if we use a mollified fundamental solution, this will still be true—it will be the curl of a somewhat blurred streamfunction instead. However, $\nabla\phi$ isn't necessarily divergence-free. We can actually use this freedom to inject nonzero divergence into the flow (e.g. an expansion in a combustion region), but for now we will want to impose zero divergence as a constraint:

$$\nabla \cdot \nabla\phi = 0.$$

Surprise! Here's yet another Laplacian popping up. This equation is another way of saying ϕ has to be a **harmonic** function.

At any point on a solid boundary with normal $\hat{n}$, we want to impose the no-stick condition,

$$(\vec{u}_0 + \nabla\phi) \cdot \hat{n} = \vec{u}_{\text{solid}} \cdot \hat{n},$$

which we can rearrange as a boundary condition on ϕ:

$$\nabla\phi \cdot \hat{n} = (\vec{u}_{\text{solid}} - \vec{u}_0) \cdot \hat{n}.$$

At each solid boundary, we can evaluate the Biot-Savart velocity, subtract it from the solid velocity, and the dot-product with the normal gives us the boundary condition for ϕ.

This may seem a little familiar: it's actually just like the potential flow we saw in Chapter 13. There we imposed a similar Neumann boundary condition on the bottom of the ocean, while using Bernoulli's theorem to get the boundary condition at the free surface.

[1] Incidentally, this also closely relates to the famous Helmholtz decomposition; many people prefer to start from Helmholtz to get into vortex methods.

Writing down the PDE for ϕ is a far cry from solving for it, and then evaluating $\nabla\phi$ wherever we need it, of course. There are at least two interesting possibilities for general solid geometry. The first is to actually use the fundamental solution and summation again, but with "particles" embedded on the solid boundary: this leads to Boundary Integral Equations a.k.a. the Boundary Element Method. Brochu et al. [BKB12] used this approach together with vortex sheets for the flow, if you want to read more on that. The other approach to consider is to discretize the problem onto a grid. We already know how to solve the Poisson problem for pressure, with Neumann solid boundary conditions, on a grid quite well: this is hardly any different.

Let's now turn to no-slip viscous boundaries. Given that we are ostensibly looking at inviscid flow, it may be mysterious why we even consider them. However, remember that air really does have a small amount of viscosity, so no-slip is correct at some level. Moreover, inviscid no-stick boundaries have some strange, even paradoxical features. It can be shown that, with truly inviscid flow and boundaries, it is impossible for a solid object to experience lift or drag or any interesting net force in the air if it starts at rest—and yet we are all familiar with lift and drag, not the least in airplanes which can in fact fly. With our earlier velocity-pressure solvers, we always had enough numerical dissipation in the system that this theoretical problem doesn't bother us, but vortex particles eliminate numerical dissipation to the extent that we have to worry about it here.

In reality, the flow near a solid tends to exhibit "boundary layers," where the velocity changes from matching the solid at its surface to slipping past tangentially a very short distance away. That kind of layer with rapidly changing tangential velocity naturally has very strong vorticity. This guides us to the idea that to enforce no-slip boundaries, we should add vortex particles (or similar vortex sources) on the surface of the solid. Where it gets particularly interesting is in deciding when and where the vorticity computed at the solid can detach from the solid and flow into the air as more vortex particles. This is critical to the right look for flow past solids, but is still very much an active area of research: I refer you to the papers and books mentioned above to learn more.

14.3.8 Vortex-in-Cell

Continuing with the idea of solving for ϕ on a grid to enforce no-stick solid boundaries, also remember the Biot-Savart velocity itself is derived from solving a Poisson problem (either directly from the curl of vorticity, or through a streamfunction): why not solve it on a grid too? This leads to the Vortex-in-Cell (VIC) method (begun by Christiansen [Chr73]).

The essence of VIC, much like PIC and FLIP, is to transfer vorticity

stored on particles to a background grid of the whole domain, solve for the velocity field on the grid, and then interpolate back to the particles. This immediately avoids the cost of Biot-Savart summation—instead we just need to solve a few Poisson problems on a grid, which we can do very efficiently already, plus some cheap finite differences and interpolation. No complicated tree codes are needed.

The main downside to VIC, compared to Biot-Savart-based methods, is that we lose some of the extraordinary detail which vortex particles can provide. To match the same detail requires a very fine grid, which may end up being more expensive again.

15

Coupling Fluids and Solids

We have covered in some detail earlier (Chapter 5) how to incorporate moving solid-wall boundary conditions in a simulation. The assumption there was that the solids followed an immutable scripted path: the fluid can't push back on them to change their motion. This chapter is focused on providing this two-way coupling.

15.1 One-Way Coupling

However, before going into two-way coupling, let's take a quick look at the other one-way coupling: solids that take their motion from the fluid, but don't affect it. This is particularly useful for solids that are much lighter than the fluid or much smaller than the features of the flow. In fact we have already seen this, tracing marker particles in the velocity field of the fluid for smoke animation. Here the position $\vec{x}_i$ of each particle simply followed the fluid velocity field:

$$\frac{d\vec{x}_i}{dt} = \vec{u}(\vec{x}_i, t). \tag{15.1}$$

This is also useful for small objects, or even particle systems representing foam, drifting in water, perhaps with a step projecting them to stay on the water surface.

One step up from marker particles are rigid bodies that take their motion from the fluid. In addition to moving their centers of mass with Equation (15.1), we need to update their orientations. Recalling that the vorticity of the flow $\vec{\omega} = \nabla \times \vec{u}$ is twice the angular velocity of the fluid at any given point, we simply integrate the rigid body position using $\frac{1}{2}\vec{\omega}$. For example, using a unit-length quaternion $\hat{q}_i$ to represent the orientation, we could update it over a time step Δt with

$$\begin{aligned} \tilde{q}_i &= \left(1, \tfrac{1}{4}\Delta t\vec{\omega}\right) \hat{q}_i^n, \\ \hat{q}_i^{n+1} &= \frac{\tilde{q}_i}{\|\tilde{q}_i\|}. \end{aligned}$$

Take note of the factor of $\frac{1}{4}$: this is $\frac{1}{2}$ from the quaternion integration formula and another $\frac{1}{2}$ to get angular velocity from vorticity. Advancing orientations in this manner is useful not just for actual rigid bodies but also for oriented particles that carry a local coordinate system—see Rasmussen et al. [RNGF03] for an example in constructing highly detailed smoke plumes from explosions, where each particle carries a volumetric texture.

More generally, we might want solids to have some inertia, with the effect of the fluid felt in terms of force, not velocity. As we know, there are two forces in effect in a fluid: pressure and viscous stress. The second is perhaps more important for very small objects.

The net force due to viscosity is the surface integral of viscous traction, the viscous stress tensor times the surface normal:

$$\vec{F} = -\iint_{\partial S} \tau \hat{n}.$$

Here I take S to be the volume of the solid and ∂S to be its boundary—this is a slight change of notation from earlier chapters where S represented the solid surface. The normal here points *out* of the solid and *into* the fluid, leading to the negative sign. In one-way coupling, the viscous boundary condition $\vec{u} = \vec{u}_{\text{solid}}$ isn't present in the simulation and thus the fluid's viscous stress tensor isn't directly usable. Indeed, the assumption underlying one-way coupling is that the solid objects don't have an appreciable effect on the fluid at the resolution of the simulation. However, we can imagine that *if* the solid were in the flow, there would be a small **boundary layer** around it in which the velocity of the fluid rapidly alters to match the solid velocity: the gradient of velocity in this region gives us the viscous stress tensor. The actual determination of this boundary layer and exactly what average force results is in general unsolved. We instead boil it down to simple formulas, with tunable constants. For small particles in the flow, we posit a simple drag force of the form:

$$\vec{F}_i = D(\vec{u} - \vec{u}_i).$$

Here D is proportional to the fluid's dynamic viscosity coefficient, and might be a per-particle constant, or involve the radius or cross-sectional area of the object, or might even introduce a non-linearity such as being proportional to $\|\vec{u} - \vec{u}_i\|$—in various engineering contexts all of these have been found to be useful. For flatter objects, such as leaves or paper, we might constrain the normal component of the velocity to match the fluid and only apply a weak (if not zero) viscous force in the tangential direction.

If we are further interested in solids with orientation, the net torque on

an object due to viscosity is likewise

$$\vec{T} = -\iint_S (\vec{x} - \vec{x}_i) \times (\tau \hat{n}),$$

where $\vec{x}_i$ is the center of mass of the object, and similarly we can't hope to derive a perfect physical formula for it. Instead we can posit simple formulas now based on the difference between the angular velocity of the solid and half the vorticity of the fluid:

$$\vec{T} = E\left(\tfrac{1}{2}\vec{\omega} - \vec{\Omega}\right).$$

The proportionality E can be tuned similar to D and may even be generalized to a matrix incorporating the current rotation matrix of the object if the solids are far from round.

The effect of pressure is a little simpler. The net force in this case is

$$\vec{F} = -\iint_{\partial S} p\hat{n} = -\iiint_S \nabla p,$$

where we have used the divergence theorem to convert it into an integral of the pressure gradient over the volume occupied by the solid. For small objects, we can evaluate ∇p from the simulation at the center of mass and multiply by the object's volume to get the force. Note that for water sitting still (and assuming a free surface pressure of zero), the hydrostatic pressure is equal to $\rho_{\text{water}}|g|d$ where d is depth below the surface, giving a gradient of $-\rho_{\text{water}}\vec{g}$. Multiplying this by the volume that the object displaces, we get the mass of displaced water, leading to the usual buoyancy law.

The torque due to pressure is

$$T = -\iint_{\partial S} (\vec{x} - \vec{x}_i) \times (p\hat{n}).$$

If p is smooth enough throughout the volume occupied by the solid—say it is closely approximated as a constant or even linear function—this integral vanishes, and there is no torque on the object; we needn't model it. Do note that in the case of water sitting still, the pressure is *not* smooth across the surface—it can be well approximated as a constant zero above the surface, compared to the steep linear gradient below—and thus a partially submerged object can experience considerable torque from pressure. In the partially submerged case, the integral should be taken (or approximated) over the part of the solid below the water surface.

15.2 Weak Coupling

For objects large or heavy enough to significantly affect the fluid flow, but light enough to be affected in turn by the fluid, we need methods for

simulating both in tandem. One common approach to implementing this two-way coupling is sometimes termed **weak coupling**. In this scheme, we interleave the solid- and fluid-simulation steps. At its simplest, we get the following algorithm for each time step:

- Advect the fluid, and update the solid positions (and orientations if relevant).
- Integrate non-coupled forces into all velocities (e.g., gravity, internal elasticity forces).
- Solve for the pressure to make the fluid incompressible, enforcing the solid-wall boundary condition with the current solid velocities held fixed.
- Update the solid velocities from forces due to the new fluid pressure and from contact/collision.

More complicated schemes are of course possible, e.g., with repeated alternations between fluid and solid or with substeps to get higher accuracy for the internal elastic forces, but the essence of weak coupling remains: one pressure solve for fluid treats the solid velocities as fixed, and one update to solid velocities treats the fluid pressure as fixed.

In terms of implementation, we have already covered the fluid aspect of this problem since, from the point of view of the fluid solver, the solid is always treated as fixed as before.[1] All that needs to be added is the fluid-to-solid stage, where fluid forces are applied to the solid.

For a rigid object, the fluid-to-solid coupling amounts to finding the net force and torque due to the fluid, which we have seen in surface integral form in the previous section. If the geometry of the solid objects is tesselated finely enough (i.e., on a scale comparable to the grid spacing Δx) these surface integrals can be directly approximated with numerical quadrature. For example, if the object surface is represented by a triangle mesh, the force could be as simple as summing over the triangles the product of triangle area with pressure interpolated at the centroid of the triangle, and if relevant, the viscous stress tensor times the triangle normal. The torque can similarly be approximated. However, in other circumstances (e.g., objects tesselated at a very different scale) directly approximating these surface integrals can be inconvenient. Luckily the face area fractions we compute for the pressure solve can directly be used here too with a little effort.

[1] Though as Guendelman et al. [GSLF05] point out, if the solids are thin, care must be taken in advection—in the semi-Lagrangian approach, if a particle trajectory is traced back *through* a solid, the fluid velocity at the interfering solid wall should be used, interpolated in a one-sided way only from the correct side of the solid; for particle methods, collision detection should be used to ensure particles don't pass through solid walls.

For example, the net pressure force is

$$\vec{F} = -\iint_{\partial S} p\hat{n}.$$

Breaking this up into a sum over grid cells intersected by the surface of the solid, and assuming pressure is constant in each such cell, gives

$$\vec{F} = -\sum_{i,j,k} p_{i,j,k} \iint_{\partial S_{i,j,k}} \hat{n}.$$

Consider one such grid cell, and suppose the fluid volume in the cell is $F_{i,j,k}$, while the parts of the cell faces in the fluid are $\partial F_{i,j,k}$. The Fundamental Theorem of Calculus applied to the constant 1 shows that

$$0 = \iiint_{F_{i,j,k}} \nabla 1 = \iint_{\partial F_{i,j,k}} 1\hat{n} + \iint \partial S_{i,j,k} 1\hat{n},$$

or rearranged,

$$-\iint_{\partial S_{i,j,k}} \hat{n} = \iint_{\partial S_{i,j,k}} \hat{n}.$$

The latter is just an easy sum over the face fractions we compute for our accurate pressure solve anyhow, scaled by Δx^2.

Likewise the net torque is

$$\vec{T} = -\iint_{\partial S} (\vec{x} - \vec{x}_C) p\hat{n},$$

where $\vec{x}_C$ is the center of mass. The same approach can be taken to compute this in terms of face area fractions of grid cells..

This general approach has met with success in many graphics papers (e.g., [THK02,GSLF05]) and is quite attractive from a software architecture point of view—the internal dynamics of fluids and solids remain cleanly separated, with new code only for integrating fluid forces applied to solids—but does suffer from a few problems that may necessitate smaller than desirable time steps. For example, if we start with a floating solid initially resting at equilibrium: after adding acceleration due to gravity all velocities are $\Delta t\vec{g}$, the fluid pressure solve treats this downward velocity at the solid surface as a constraint and thus leads to non-zero fluid velocities, and finally the pressure field (perturbed from hydrostatic equilibrium) doesn't quite cancel the velocity of the solid; the solid sinks to some extent, and the water starts moving. These errors are proportional to the time-step size and thus of course can be reduced, but at greater expense.

15.3 The Immersed Boundary Method

A somewhat stronger coupling scheme is epitomized by the immersed boundary method (the classic reference is the review article by Peskin [Pes02]). Here we give the fluid pressure solve leeway to change the solid velocity by, in effect, momentarily pretending the solid is also fluid (just of a different density). In particular, rather than impose the solid velocity as boundary conditions for the fluid pressure solve, we add the mass and velocity of the solid to the fluid grid and then solve for pressure throughout the whole domain. The usual fluid fractions are used as weights in determining the average density and average velocity in each u-, v-, and w-cell, and then the fractions actually used in determining the pressure equations are full. (Incidentally, this approach was in fact combined with the approach of the previous section in the paper of Guendelman et al. [GSLF05], where this pressure is used instead of the less accurate pressure of the classic voxelized pressure solve to update the solid's velocity.)

A related method, the rigid fluid approach of Carlson et al. [CMT04], simplifies the solve somewhat by moreover assuming the density of the solid to be the same as the fluid and adding a corrective buoyancy force as a separate step, recovering a rigid body's velocity directly from averaging the velocity on the grid after the pressure solve (i.e., finding the average translational and angular velocity of the grid cells the solid occupies) rather than integrating pressure forces over the surface of the body. This can work extremely well if the ratio of densities isn't too large.

For inviscid flow, simply averaging the solid and fluid velocities in mixed cells as is typically done in the immersed boundary method may lead to excessive numerical dissipation. Recall that the tangential velocity of the solid is not coupled to the tangential velocity of the fluid: only the normal components are connected for inviscid flows. When averaging the full velocities together we are, in essence, constraining the fluid to the viscous boundary condition $\vec{u} = \vec{u}_{\text{solid}}$. Therefore it is recommended if possible to extrapolate the tangential component of fluid velocity into the cells occupied by the solid and only average the normal component of the solid's velocity onto the grid. For very thin solids, such as cloth, this is particularly simple since extrapolation isn't required—just a separation of the solid velocities into normal and tangential components.

This approach helps reduce some of the artifacts of the previous weak-coupling method, but it doesn't succeed in all cases. For example, starting a simulation with a floating object resting at equilibrium still ends up creating false motion, since in the pressure solve the solid object appears to be an odd-shaped wave on the fluid surface.

15.4 General Sparse Matrices

Before getting into strong coupling, where we compute fluid and solid forces simultaneously, we need to take a brief diversion to generalize our sparse matrix capabilities: the regular structure of the matrices used up until now will not accommodate the addition of solids.

There are several possible data structures for storing and manipulating general sparse matrices. The one we will focus on is sometimes called the **compressed sparse row** (or CSR) format. Here each row of the matrix is stored as an array of non-zero values and their associated column indices. We'll actually use two variations of CSR, a simple dynamic version (that makes adding new non-zeros when dynamically constructing a matrix fairly efficient) and a static version that gives better performance in PCG.

In dynamic CSR, the array for each sparse row is stored independently with an associated length. To support adding new non-zeros relatively efficiently, we may allocate extra storage for these arrays and keep track of the total available; when the extra space runs out, we can reallocate the array with double the size (or some other multiplier). This is the strategy taken in the C++ STL `vector` container for example. Often people will further maintain the arrays in order sorted by column index, making it more efficient to find entries or add two sparse rows together.

However, the core of PCG, multiplying the sparse matrix and a dense vector, loses some efficiency with this approach: each sparse row might be scattered in memory leading to poor cache usage. Therefore, after constructing a matrix using the dynamic CSR structure, we convert it to a static CSR structure. Here just three arrays are defined, ensuring that all matrix non-zeros are contiguous in memory:

- a floating-point array `value` containing all non-zero values ordered row by row,
- an integer array `colindex` of the same length containing the corresponding column indices, and
- an integer array `rowstart` of length $n+1$ (for an $n \times n$ matrix) indicating where each sparse row begins in `value` and `colindex`—an extra entry at the end points just past the end of the `value` and `colindex` arrays (i.e., contains the number of non-zeros in the matrix).

The small overhead of converting a dynamic CSR matrix to the static format is generally well worth it for PCG, bringing the cost back in line with our earlier grid-based method.

You may recall that with our grid version of the matrix we optimized the storage by exploiting symmetry, in some sense just storing the upper

```
for i = 0 to n-1
   y(i) = 0
   for j = rowstart(i) to rowstart(i+1)-1
      y(i) += value(j)*x(colindex(j))
```

Figure 15.1. Pseudocode for multiplying an $n \times n$ static CSR matrix with a dense vector, $y = Ax$.

(or lower) triangle of the matrix. This is generally not worthwhile with CSR: it considerably complicates matrix operations.

It is fairly straightforward to multiply a static CSR sparse matrix with a dense vector; see Figure 15.1 for pseudocode. It's another matter to generalize the incomplete Cholesky preconditioner and associated triangular solves. It turns out, for general sparsity patterns, that the previous simplification of only having to compute diagonal entries (and reusing the off-diagonal entries of A) doesn't hold, and furthermore the modified incomplete Cholesky factorization cannot be computed with the same loop-ordering presented earlier. It is more natural, in fact, to compute $R = L^T$ in CSR format (or equivalently, L in a compressed sparse column format). All that said, we can still define the regular incomplete factor from the previous properties:

- R is upper triangular, and $R_{i,j} = 0$ wherever $A_{i,j} = 0$,

- Set tuning constant $\tau = 0.97$ and safety constant $\sigma = 0.25$.
- Copy the upper triangle of A into R (including the diagonal).
- For $k = 0$ to $n - 1$ where $R_{k,k} \neq 0$:
  - If $R_{k,k} < \sigma A_{k,k}$ then set $R_{k,k} \leftarrow \sqrt{A_{k,k}}$, otherwise set $R_{k,k} \leftarrow \sqrt{R_{k,k}}$.
  - Rescale the rest of the k'th row of R: $R_{k,j} \leftarrow \frac{R_{k,j}}{R_{k,k}}$ for stored entries with $j > k$.
  - Loop over $j > k$ where $R_{k,j}$ is stored:
    - Set $\delta = 0$ (where we keep a sum of the elements we drop).
    - Loop over $i > k$ where $R_{k,i}$ is stored:
      - If $R_{j,i}$ is stored, set $R_{j,i} \leftarrow R_{j,i} - R_{k,i}R_{k,j}$, otherwise $\delta \leftarrow \delta + R_{k,i}R_{k,j}$.
  - Set $R_{j,j} \leftarrow R_{j,j} - \tau\delta$.

Figure 15.2. The calculation of the MIC(0) preconditioner R for general matrices A, using CSR format.

- (First solve $R^T z = r$).
- Copy $z \leftarrow r$.
- For $i = 0$ to $n - 1$ where $R_{i,i} \neq 0$:
  - Set $z_i \leftarrow \frac{z_i}{R_{i,i}}$.
  - Loop over $j > i$ where $R_{i,j}$ is stored:
    - Set $z_j \leftarrow z_j - R_{i,j} z_i$.
- (Next solve $Rz^{\text{new}} = z$ in place).
- For $i = n - 1$ down to 0, where $R_{i,i} \neq 0$:
  - Loop over $j > i$ where $R_{i,j}$ is stored:
    - Set $z_i \leftarrow z_i - R_{i,j} z_j$.
  - Set $z_i \leftarrow \frac{z_i}{R_{i,i}}$.

Figure 15.3. Applying the MIC(0) preconditioner in CSR format to get $z = (R^T R)^{-1} r$.

- $(R^T R)_{i,j} = A_{i,j}$ wherever $A_{i,j} \neq 0$,

and the modified factor from:

- R is upper triangular, and $R_{i,j} = 0$ wherever $A_{i,j} = 0$,
- $(R^T R)_{i,j} = A_{i,j}$ wherever $A_{i,j} \neq 0$ with $i < j$ (i.e. off the diagonal),
- each row sum $\sum_j (R^T R)_{i,j}$ matches the row sum $\sum_j A_{i,j}$ of A.

Without going into the picky but obvious details of using the format, Figure 15.2 presents pseudocode to construct R, with the same parameters as before, and Figure 15.3 demonstrates how to apply the preconditioner by solving with R and R^T.

15.5 Strong Coupling

Strong coupling has been most thoroughly worked out for the rigid body case, with an inviscid fluid (just pressure); this is where we will end the chapter. Let's work out the equations for the continuous case before proceeding to discretization.

First we'll define some notation for a rigid body:

- $\vec{X}$ the center of mass of the rigid body,
- $\vec{V}$ its translation velocity,

- $\vec{\Omega}$ its angular velocity,
- $\vec{L}$ its angular momentum,
- m its mass, and
- I its inertia tensor.

(Extending this to multiple rigid bodies is straightforward.) We saw the net force and torque on the body due to pressure above, which gives us the following updates:

$$\vec{V}^{n+1} = \vec{V} - \frac{\Delta t}{m} \iint_{\partial S} p\hat{n}, \tag{15.2}$$

$$\vec{\Omega}^{n+1} = \vec{\Omega} - \Delta t I^{-1} \iint_{\partial S} \left(\vec{x} - \vec{X}\right) \times p\hat{n}. \tag{15.3}$$

The velocity of the solid at a point $\vec{x}$ on the surface is

$$\vec{u}_{\text{solid}}(\vec{x}) = \vec{V} + \Omega \times \left(\vec{x} - \vec{X}\right),$$

which then appears in the boundary conditions for the usual pressure problem:

$$\begin{aligned} \nabla \cdot \frac{\Delta t}{\rho} \nabla p &= \nabla \cdot \vec{u} && \text{in the fluid,} \\ \left(\vec{u} - \frac{\Delta t}{\rho} \nabla p\right) \cdot \hat{n} &= \vec{u}_{\text{solid}}^{n+1} \cdot \hat{n} && \text{on } \partial S. \end{aligned}$$

A free surface boundary condition $p = 0$ may also be included.

As an aside, though not immediately apparent in this form, the linear operator that maps $(\vec{V}, \vec{\Omega})$ to the normal velocity field on the boundary can be shown to be the adjoint[2] of the operator that maps pressures on the boundary to net force and torque on the solid. It's this property that will end up, after discretization, giving us a symmetric matrix to solve for pressure.

A simpler form of the equations are given by Batty et al. [BBB07]. We keep the rigid body update Equations (15.2) and (15.3), but avoid the boundary conditions by instead seeking a pressure that minimizes the kinetic energy of the entire system. This is quite compatible with the more accurate FVM pressure solve in this book, only where Batty et al. use

[2] Adjoint simply means transpose when talking about matrices but is also used for operators involving infinite-dimensional spaces such as the space of normal velocity fields here.

volume fractions as coefficients in the matrix, we use face fractions. Recall that the kinetic energy of the rigid body is just $\frac{1}{2}m\|\vec{V}\|^2 + \frac{1}{2}\vec{\Omega}^T I\vec{\Omega}$.

It is this variational form of the equations that we choose to discretize. We already have discussed how to approximate the pressure update to the rigid body (i.e., the net force and torque) in the earlier weak-coupling section; we need only make this concrete with a sparse matrix J which, when multiplied with a vector containing the grid pressure values yields the force and torque. Examine the first three rows of J that correspond to the net force. For example, the x-component F_1 (the first row) is determined, with a little rearrangement of the summation, from

$$F_1 = \sum_{i,j,k} \Delta x^2 (V_{i+1/2,j,k} - V_{i-1/2,j,k}) p_{i,j,k},$$

where $V_{i+1/2,j,k}$ is the face area fraction of the solid in u-cell $(i+1/2, j, k)$—note that this is the complement of the face area fractions for the fluid! This gives us

$$J_{1,(i,j,k)} = \Delta x^2 (V_{i+1/2,j,k} - V_{i-1/2,j,k}),$$

Similarly, the next two rows of J, corresponding to the y- and z-components of net force, are

$$\begin{aligned} J_{2,(i,j,k)} &= \Delta x^2 (V_{i,j+1/2,k} - V_{i,j-1/2,k}), \\ J_{3,(i,j,k)} &= \Delta x^2 (V_{i,j,k+1/2} - V_{i,j,k-1/2}). \end{aligned}$$

Similarly we can get the other three rows of J that correspond to the net torque. The first component T_1 is approximated by

$$\begin{aligned} T_1 = &\sum_{i,j,k} \Delta x^2 V_{i,j+1/2,k} (z_k - Z)(p_{i,j+1,k} - p_{i,j,k}) \\ &- \sum_{i,j,k} \Delta x^2 V_{i,j,k+1/2} (y_j - Y)(p_{i,j,k+1} - p_{i,j,k}), \end{aligned}$$

where x_i, y_j and z_k give the coordinates of the center of grid cell (i, j, k). From this we see that the fourth row of J is given by

$$\begin{aligned} J_{4,(i,j,k)} = &- \Delta x^2 (z_k - Z)(V_{i,j+1/2,k} - V_{i,j-1/2,k}) \\ &+ \Delta x^2 (y_j - Y)(V_{i,j,k+1/2} - V_{i,j,k-1/2}). \end{aligned}$$

Similarly, the last two rows of J, corresponding to the second and third components of net torque, are

$$\begin{aligned} J_{5,(i,j,k)} &= -\Delta x^2 (x_i - X)(V_{i,j,k+1/2} - V_{i,j,k-1/2}) \\ &\quad + \Delta x^2 (z_k - Z)(V_{i+1/2,j,k} - V_{i-1/2,j,k}), \\ J_{6,(i,j,k)} &= -\Delta x^2 (y_j - Y)(V_{i+1/2,j,k} - V_{i-1/2,j,k}) \\ &\quad + \Delta x^2 (x_i - X)(V_{i,j+1/2,k} - V_{i,j-1/2,k}). \end{aligned}$$

Note that away from the boundary of the solid, all of these differences are just zero, so J is quite sparse: it has non-zero columns only for cells near the boundary of the solid.

To be perfectly consistent, and thus be able to get exact hydrostatic rest for neutrally buoyant bodies fully immersed in fluid, we can also use the same fractions to approximate the rigid body's inertia tensor—however, outside of this particular scenario this is probably unnecessary work, and thus we leave it to the reader to derive it if interested.

For notational convenience, we'll put the rigid body's translational velocity $\vec{V}$ and angular velocity Ω together into one six-dimensional vector $\vec{U}$. Similarly we can construct a 6×6 mass matrix M from the mass and inertia tensor:

$$M = \begin{pmatrix} m & 0 & 0 & \vec{0} \\ 0 & m & 0 & \vec{0} \\ 0 & 0 & m & \vec{0} \\ \vec{0} & \vec{0} & \vec{0} & I \end{pmatrix}.$$

Then the kinetic energy of the body is $\frac{1}{2}\vec{U}^T M \vec{U}$ and the pressure update is $\vec{U}^{n+1} = \vec{U} + \Delta t M^{-1} J p$.

Finally, we are ready for the discrete minimization. Taking the gradient of the new kinetic energy with respect to pressure gives

$$\frac{\partial}{\partial p}\left[\frac{1}{2}\left(\vec{U} + \Delta t M^{-1} J p\right)^T M \left(\vec{U} + \Delta t M^{-1} J p\right)\right] = \Delta t^2 J^T M^{-1} J p + \Delta t J^T \vec{U}.$$

Batty et al. show how to rephrase the pressure solve as a kinetic energy minimization; without going into those details, the upshot is we just add $\Delta t J^T M^{-1} J$ to the appropriately scaled pressure matrix and $-J^T \vec{U}$ to the right-hand side.

This is precisely where we need more general sparse matrices: $\Delta t J^T M^{-1} J$ doesn't correspond to a simple grid structure. In fact, it forms a dense submatrix, connecting up all the cells near the boundary of the object. If you're interested in the linear algebra, it's also simple to see that it is (at most) rank six and must be symmetric positive semi-definite—so PCG still works! The density may, however, cause memory and performance problems: these can mostly be overcome by keeping the extra terms separate in factored form. The matrix-vector multiplies in PCG can be significantly accelerated then by multiplying $J^T M^{-1} J s$ as $J^T(M^{-1}(J^T s))$ and, since they are only rank six, can be ignored in constructing the preconditioner without too big a penalty.

For large objects, which overlap many grid cells, this is actually a considerable problem: a large amount of memory will be required to store it,

and PCG will run slowly due to all the work multiplying with this dense submatrix. One possibility for improvement is to add the new rigid body velocity $\vec{U}^{n+1}$ as an auxiliary variable in the pressure solve, giving the following slightly larger but much sparser system:

$$\begin{pmatrix} A & J^T \\ J & -\frac{1}{\Delta t}M \end{pmatrix} \begin{pmatrix} p \\ \vec{U}^{n+1} \end{pmatrix} = \begin{pmatrix} -d \\ -\frac{1}{\Delta t}M\vec{U} \end{pmatrix}.$$

While this still leads to a symmetric matrix, it is unfortunately now indefinite, which means PCG no longer can work. Nevertheless, this is in a well-studied class of matrices, sometimes termed "saddle-point" matrices (in fact, apart from the constant pressure null-space, it would be a "symmetric quasi-definite" matrix), and it seems promising to solve it as such. For example, it would be worthwhile trying an iterative method such as MINRES in conjunction with an incomplete Cholesky preconditioner in LDL^T form (where L has unit diagonal, and D is a diagonal matrix with positive entries for the pressure unknowns and negative entries for the rigid body unknowns). For more on solving this class of matrix problems, see the review article by Benzi et al. [BGL05] for example.

Another interesting direction to consider is to generalize this approach to include strong coupling of articulated rigid bodies: for example, the Lagrange multiplier approach to constraints can also be phrased as a minimization of kinetic energy. Frictional contact forces fall in the same category, albeit with inequality constraints that complicate the minimization considerably.

Turning to deformable objects, the energy minimization framework falters. Strong coupling in this context means combining an implicit time step of the internal dynamics of the deformable object with the pressure solve; however, implicitly advancing elastic forces (involving potential energy) apparently cannot be interpreted as a minimization of energy with respect to forces. At the time of writing, within graphics only the work of Chentanez et al. [CGFO06] has tackled this problem, discretizing the fluid on an unstructured tetrahedral mesh that conforms to the boundary mesh of the deformable object; so far the generalization to regular grids with volume fractions hasn't been made. We thus end the chapter here.

A

Background

A.1 Vector Calculus

The following three differential operators are fundamental to vector calculus: the gradient ∇, the divergence $\nabla\cdot$, and the curl $\nabla\times$. They occasionally are written in equations as grad, div, and curl instead.

A.1.1 Gradient

The gradient simply takes all the spatial partial derivatives of the function, returning a vector. In two dimensions,

$$\nabla f(x,y) = \left(\frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}\right),$$

and in three dimensions,

$$\nabla f(x,y,z) = \left(\frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z}\right).$$

It can sometimes be helpful to think of the gradient operator as a symbolic vector, e.g., in three dimensions:

$$\nabla = \left(\frac{\partial}{\partial x}, \quad \frac{\partial}{\partial y}, \quad \frac{\partial}{\partial z}\right).$$

The gradient is often used to approximate a function locally:

$$f(\vec{x} + \Delta\vec{x}) \approx f(\vec{x}) + \nabla f(\vec{x}) \cdot \Delta\vec{x}.$$

In a related vein we can evaluate the **directional derivative** of the function; that is, how fast the function is changing when looking along a particular vector direction, using the gradient. For example, if the direction is $\hat{n}$,

$$\frac{\partial f}{\partial n} = \nabla f \cdot \hat{n}.$$

Occasionally we will take the gradient of a vector-valued function, which results in a matrix (sometimes called the **Jacobian**). For example, in three dimensions,

$$\nabla \vec{f} = \nabla(f, g, h) = \begin{pmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} & \frac{\partial f}{\partial z} \\ \frac{\partial g}{\partial x} & \frac{\partial g}{\partial y} & \frac{\partial g}{\partial z} \\ \frac{\partial h}{\partial x} & \frac{\partial h}{\partial y} & \frac{\partial h}{\partial z} \end{pmatrix}.$$

Note that each *row* is the gradient of one component of the function. One way to remember that it's the rows and not the columns is that it should work with the approximation

$$\vec{f}(\vec{x} + \Delta\vec{x}) \approx \vec{f}(\vec{x}) + \nabla\vec{f}(\vec{x})\Delta\vec{x}.$$

The matrix-vector product is just computing the dot-product of each row of the matrix with the vector, and so each row should be a gradient of the function:

$$\nabla(f, g, h) = \begin{pmatrix} \nabla f \\ \nabla g \\ \nabla h \end{pmatrix}.$$

An alternative notation for the gradient that is sometimes used is

$$\nabla f = \frac{\partial f}{\partial \vec{x}}.$$

Using a vector in the denominator of the partial derivative indicates we're taking derivatives with respect to each component of $\vec{x}$.

A.1.2 Divergence

The divergence operator only is applied to vector fields and measures how much the vectors are converging or diverging at any point. In two dimensions it is

$$\nabla \cdot \vec{u} = \nabla \cdot (u, v) = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y},$$

and in three dimensions,

$$\nabla \cdot \vec{u} = \nabla \cdot (u, v, w) = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}.$$

Note that the input is a vector and the output is a scalar.

The notation $\nabla\cdot$ is explained by thinking of it as symbolically taking a dot-product between the gradient operator and the vector field, e.g., in three dimensions,

$$\begin{aligned} \nabla \cdot \vec{u} &= \left(\frac{\partial}{\partial x}, \quad \frac{\partial}{\partial y}, \quad \frac{\partial}{\partial z}\right) \cdot (u, v, w) \\ &= \frac{\partial}{\partial x}u + \frac{\partial}{\partial y}v + \frac{\partial}{\partial z}w. \end{aligned}$$

A.1.3 Curl

The curl operator measures how much a vector field is rotating around any point. In three dimensions this is a vector:

$$\nabla \times \vec{u} = \nabla \times (u, v, w) = \left(\frac{\partial w}{\partial y} - \frac{\partial v}{\partial z}, \quad \frac{\partial u}{\partial z} - \frac{\partial w}{\partial x}, \quad \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} \right).$$

We can reduce this formula to two dimensions in two ways. The curl of a two-dimensional vector field results in a scalar, the third component of the expression above, as if we were looking at the three-dimensional vector field $(u, v, 0)$:

$$\nabla \times \vec{u} = \nabla \times (u, v) = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}.$$

The curl of a two-dimensional scalar field results in a vector field, as if we were looking at the three-dimensional field $(0, 0, w)$:

$$\nabla \times w = \left(\frac{\partial w}{\partial y}, \quad -\frac{\partial w}{\partial x} \right).$$

The simple way to remember these formulas is that the curl is taking a symbolic cross-product between the gradient operator and the function. For example, in three dimensions,

$$\begin{aligned} \nabla \times \vec{u} &= \left(\frac{\partial}{\partial x}, \quad \frac{\partial}{\partial y}, \quad \frac{\partial}{\partial z} \right) \times (u, v, w) \\ &= \left(\frac{\partial}{\partial y} w - \frac{\partial}{\partial z} v, \quad \frac{\partial}{\partial z} u - \frac{\partial}{\partial x} w, \quad \frac{\partial}{\partial x} v - \frac{\partial}{\partial y} u \right). \end{aligned}$$

The curl is a way of measuring how fast (and in three dimensions along what axis) a vector field is rotating locally. If you imagine putting a little paddle wheel in the flow and letting it be spun, then the curl is twice the angular velocity of the wheel. You can check this by taking the curl of the velocity field representing a rigid rotation.

A vector field whose curl is zero is called curl-free, or **irrotational** for obvious reasons.

A.1.4 Laplacian

The Laplacian is usually formed as the divergence of the gradient (as it repeatedly appears in fluid dynamics). Sometimes it is written as ∇^2 or

Δ, but since these symbols are occasionally used for other purposes, I will stick to writing it as $\nabla \cdot \nabla$. In two dimensions,

$$\nabla \cdot \nabla f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

and in three dimensions,

$$\nabla \cdot \nabla f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} + \frac{\partial^2 f}{\partial z^2}.$$

The Laplacian can also be applied to vector or even matrix fields, and the result is simply the Laplacian of each component.

Incidentally, the partial differential equation $\nabla \cdot \nabla f = 0$ is called Laplace's equation, and if the right-hand side is replaced by something non-zero, $\nabla \cdot \nabla f = q$ we call it the Poisson equation. More generally, you can multiply the gradient by a scalar field a (such as $1/\rho$), like $\nabla \cdot (a \nabla f) = q$ and still call it a Poisson problem.

A.1.5 Differential Identities

There are several identities based on the fact that changing the order of mixed partial derivatives doesn't change the result (assuming reasonable smoothness), e.g.,

$$\frac{\partial}{\partial x}\frac{\partial}{\partial y} f = \frac{\partial}{\partial y}\frac{\partial}{\partial x} f.$$

Armed with this, it's simple to show that for any smooth function,

$$\nabla \cdot (\nabla \times \vec{u}) \equiv 0,$$
$$\nabla \times (\nabla f) \equiv 0.$$

Another identity that shows up in vorticity calculations is

$$\nabla \times (\nabla \times \vec{u}) \equiv \nabla (\nabla \cdot \vec{u}) - \nabla \cdot \nabla \vec{u}.$$

The Helmholtz or Hodge decomposition is the result that any smooth vector field $\vec{u}$ can be written as the sum of a divergence-free part and a curl-free part. In fact, referring back to the first two identities above, the divergence-free part can be written as the curl of something and the curl-free part can be written as the gradient of something else. In three dimensions,

$$\vec{u} = \nabla \times \vec{\psi} - \nabla \phi,$$

where $\vec{\psi}$ is a vector-valued potential function and ϕ is a scalar potential function. In two dimensions this reduces to ψ being a scalar potential function as well:

$$\vec{u} = \nabla \times \psi - \nabla \phi.$$

This decomposition is highly relevant to incompressible fluid flow, since we can interpret the pressure projection step as decomposing the intermediate velocity field $\vec{u}^{n+1}$ into a divergence-free part and something else which we throw away, just keeping the divergence-free part. When we express a divergence-free velocity field as the curl of a potential ψ, we call ψ the **streamfunction**.

Some more useful identities are generalizations of the product rule:

$$\nabla(fg) = (\nabla f)g + f\nabla g,$$
$$\nabla \cdot (f\vec{u}) = (\nabla f) \cdot \vec{u} + f\nabla \cdot \vec{u}.$$

A.1.6 Integral Identities

The Fundamental Theorem of Calculus (that the integral of a derivative is the original function evaluated at the limits) can be generalized to multiple dimensions in a variety of ways.

The most common generalization is the divergence theorem discovered by Gauss:

$$\iiint_\Omega \nabla \cdot \vec{u} = \iint_{\partial\Omega} \vec{u} \cdot \hat{n}.$$

That is, the volume integral of the divergence of a vector field $\vec{u}$ is the boundary integral of $\vec{u}$ dotted with the unit outward normal $\hat{n}$. This actually is true in any dimension (replacing volume with area or length or hypervolume as appropriate). This provides our intuition of the divergence measuring how fast a velocity field is expanding or compressing: the boundary integral above measures the net speed of fluid entering or exiting the volume.

Stokes' Theorem applies to the integral of a curl. Suppose we have a bounded surface S with normal $\hat{n}$ and with boundary curve Γ whose tangent vector is τ. Then,

$$\iint_S (\nabla \times \vec{u}) \cdot \hat{n} = \int_\Gamma \vec{u} \cdot \tau.$$

This can obviously be restricted to two dimensions with $\hat{n} = (0, 0, 1)$. The curve integral is called the **circulation** in the context of a fluid velocity field.

We can also integrate a gradient:

$$\iiint_\Omega \nabla f = \iint_{\partial\Omega} f\hat{n}.$$

Some of the most useful identities of all are ones called **integration by parts**, which is what we get when we combine integration identities based

on the Fundamental Theorem of Calculus with the product rule for derivatives. They essentially let us move a differential operator from one factor in a product to the other. Here are some of the most useful:

$$\iiint_\Omega (\nabla f)g = \iint_{\partial\Omega} fg\hat{n} - \iiint_\Omega f(\nabla g),$$
$$\iiint_\Omega f\nabla\cdot\vec{u} = \iint_{\partial\Omega} f\vec{u}\cdot\hat{n} - \iiint_\Omega (\nabla f)\cdot\vec{u},$$
$$\iiint_\Omega (\nabla f)\cdot\vec{u} = \iint_{\partial\Omega} f\vec{u}\cdot\hat{n} - \iiint_\Omega f\nabla\cdot\vec{u}.$$

Replacing $\vec{u}$ by ∇g in the last equation gives us one of Green's identities:

$$\begin{aligned}\iiint_\Omega (\nabla f)\cdot(\nabla g) &= \iint_{\partial\Omega} f\nabla g\cdot\hat{n} - \iiint_\Omega f\nabla\cdot\nabla g \\ &= \iint_{\partial\Omega} g\nabla f\cdot\hat{n} - \iiint_\Omega g\nabla\cdot\nabla f.\end{aligned}$$

A.1.7 Basic Tensor Notation

When you get into two or three derivatives in multiple dimensions, it can get very confusing if you stick to using the ∇ symbols. An alternative is to use tensor notation, which looks a little less friendly but makes it trivial to keep everything straight. Advanced differential geometry is almost impossible to do without this notation. We'll present a simplified version that is adequate for most of fluid dynamics.

The basic idea is to label the separate components of a vector with subscript indices 1, 2, and in three dimensions, 3. Usually we'll use variables i, j, k, etc., for these indices. Note that this can get very confusing if you also are thinking of discretizing on a grid—if you want to avoid that confusion, it's often a good idea to only use Greek letters for your tensor indices, e.g., α, β, γ instead.

The gradient of a function is $(\partial f/\partial x_1, \partial f/\partial x_2, \partial f/\partial x_3)$. This is still a bit longwinded, so we instead use the generic $\partial f/\partial x_i$ without specifying what i is: it's a "free" index.

We could then write the divergence, for example, as

$$\sum_i \frac{\partial u_i}{\partial x_i}.$$

This brings us to the **Einstein summation convention**. It's tedious to have to write the sum symbol Σ again and again. Thus we just won't bother writing it: instead, we will assume that in any expression that contains the

index i twice, there is an implicit sum over i in front of it. If we don't want a sum, we use different indices, like i and j. For example, the dot-product of two vectors $\vec{u}$ and $\hat{n}$ can be written very succinctly as

$$u_i n_i.$$

Note that by expression I mean a single term or a product—it does not include addition. So this

$$u_i + r_i$$

is a vector, $\vec{u} + \vec{r}$, not a scalar sum.

Einstein notation makes it very simple to write a matrix-vector product, such as $A\vec{x}$:

$$A_{ij} x_j.$$

Note that the free index in this expression is j: this is telling us the jth component of the result. This is also an introduction to second-order tensors, which really are a fancy name for matrices: they have two indices instead of the one for a vector (which can be called a first-order tensor). We can write matrix multiplication just as easily: the product AB is

$$A_{ij} B_{jk}$$

with free indices i and k: this is the i, k entry of the result. Similarly, the outer-product matrix of vectors $\vec{u}$ and $\hat{n}$ is

$$u_i n_j.$$

Other useful symbols for tensor expressions are the Kronecker delta δ_{ij} and the Levi-Civita symbol ϵ_{ijk}. The Kronecker delta is δ_{ij}, which is actually just the identity matrix in disguise: $\delta_{ij} x_j = x_i$. The Levi-Civita symbol has three indices, making it a third-order tensor (kind of like a three-dimensional version of a matrix!). It is zero if any of the indices are repeated, $+1$ if (i, j, k) is just a rotation of $(1, 2, 3)$, and -1 if (i, j, k) is a rotation of $(3, 2, 1)$. What this boils down to is that we can write a cross-product using it: $\vec{r} \times \vec{u}$ is just

$$\epsilon_{ijk} r_j u_k,$$

which is a vector with free index i.

Putting all this together, we can translate the definitions, identities and theorems from before into very compact notation. Furthermore, just by keeping our indices straight, we won't have to puzzle over what an expression like $\nabla\nabla\vec{u} \cdot \hat{n} \times \nabla f$ might actually mean. Here are some of the

translations that you can check:

$$(\nabla f)_i = \frac{\partial f}{\partial x_i},$$

$$\nabla = \frac{\partial}{\partial x_i},$$

$$f(x_i + \Delta x_i) \approx f(x_i) + \frac{\partial f}{\partial x_i}\Delta x_i,$$

$$\frac{\partial f}{\partial n} = \frac{\partial f}{\partial x_i} n_i,$$

$$(\nabla \vec{f})_{ij} = \frac{\partial f_i}{\partial x_j},$$

$$\nabla \cdot \vec{u} = \frac{\partial u_i}{\partial x_i},$$

$$(\nabla \times \vec{u})_i = \epsilon_{ijk} \frac{\partial u_k}{\partial x_j},$$

$$\nabla \cdot \nabla f = \frac{\partial^2 f}{\partial x_i \partial x_i},$$

$$\frac{\partial}{\partial x_i} \epsilon_{ijk} \frac{\partial u_k}{\partial x_j} = 0,$$

$$\epsilon_{ijk} \frac{\partial}{\partial x_j} \frac{\partial f}{\partial x_k} = 0.$$

The different versions of the product rule for differentiation, in tensor notation, all just fall out of the regular single-variable calculus rule. For example,

$$\frac{\partial}{\partial x_i}(fg) = \frac{\partial f}{\partial x_i}g + f\frac{\partial g}{\partial x_i},$$
$$\frac{\partial}{\partial x_i}(fu_i) = \frac{\partial f}{\partial x_i}u_i + f\frac{\partial u_i}{\partial x_i}.$$

The integral identities also simplify. For example,

$$\iiint_\Omega \frac{\partial u_i}{\partial x_i} = \iint_{\partial\Omega} u_i n_i,$$
$$\iiint_\Omega \frac{\partial f}{\partial x_i} = \iint_{\partial\Omega} f n_i,$$
$$\iiint_\Omega \frac{\partial f}{\partial x_i}g = \iint_{\partial\Omega} f g n_i - \iiint_\Omega f\frac{\partial g}{\partial x_i}.$$

A.2 Numerical Methods

This book concentrate on methods based on finite differences, which themselves boil down simply to applications of the Taylor series.

Assuming a function f has at least k smooth derivatives, then

$$f(x+\Delta x) = f(x) + \frac{\partial f}{\partial x}(x)\Delta x + \frac{1}{2}\frac{\partial^2 f}{\partial x^2}(x)\Delta x^2 + \frac{1}{6}\frac{\partial^3 f}{\partial x^3}(x)\Delta x^3 + \cdots + \frac{1}{(k-1)!}\frac{\partial^{k-1} f}{\partial x^{k-1}}(x)\Delta x^{k-1} + R_k.$$

The remainder term R_k can be expressed in several ways, for example,

$$R_k = \int_x^{x+\Delta x} \frac{1}{k!}\frac{\partial^k f}{\partial x^k}(s)s^{k-1}ds,$$
$$R_k = \frac{1}{k!}\frac{\partial^k f}{\partial x^k}(s)\Delta x^k \quad \text{for some } s \in [x, x+\Delta x],$$
$$R_k = O(\Delta x^k).$$

Note that Δx could be negative, in which case the second form of the remainder uses the interval $[x+\Delta x, x]$. We'll generally stick with the last

form, using the simple $O()$ notation, but do remember that the hidden constant is related to the kth derivative of f—and if f isn't particularly smooth, that could be huge and the Taylor series (taken up to that term) isn't particularly useful.

A.2.1 Finite Differences in Space

Partial derivatives of smooth functions sampled on a grid can be estimated using Taylor's theorem. For example, for a function $q(x)$ sampled at grid points spaced Δx apart, i.e., $q_i = q(x_i) = q(i\Delta x)$, Taylor's theorem gives

$$q_{i+1} = q_i + \Delta x \frac{\partial q}{\partial x}(x_i) + O(\Delta x^2).$$

We can rearrange this to get an estimate of $\partial q/\partial x$ at x_i:

$$\frac{\partial q}{\partial x}(x_i) = \frac{q_{i+1} - q_i}{\Delta x} + O(\Delta x).$$

Note that after dividing through by Δx, the error term was reduced to **first order**, i.e., the exponent of Δx in the $O()$ notation is one.

Of course, you can also estimate the same derivative from q_{i-1}, using Taylor's theorem for $q_{i-1} = q(x_{i-1})$:

$$\frac{\partial q}{\partial x}(x_i) = \frac{q_i - q_{i-1}}{\Delta x} + O(\Delta x).$$

This is also only first-order accurate. Both this and the previous finite difference are **one-sided**, since values of q only to one side of the approximation point are used.

We can get second-order accuracy by using both q_{i+1} and q_{i-1}, for a **centered** or **central** finite difference: the value we're approximating lies in the center of the points we use. Write down the Taylor series for both these points:

$$q_{i+1} = q_i + \Delta x \frac{\partial q}{\partial x}(x_i) + \frac{\Delta x^2}{2}\frac{\partial^2 q}{\partial x^2}(x_i) + O(\Delta x^3),$$
$$q_{i-1} = q_i - \Delta x \frac{\partial q}{\partial x}(x_i) + \frac{\Delta x^2}{2}\frac{\partial^2 q}{\partial x^2}(x_i) + O(\Delta x^3).$$

Now subtract them to get, after cancellation,

$$q_{i+1} - q_{i-1} = 2\Delta x \frac{\partial q}{\partial x}(x_i) + O(\Delta x^3).$$

Dividing through gives the second-order accurate central finite difference:

$$\frac{\partial q}{\partial x}(x_i) = \frac{q_{i+1} - q_{i-1}}{2\Delta x} + O(\Delta x^2).$$

Similar reasoning also shows that the first formula we saw is a second-order accurate central finite difference for the point $x_{i+1/2} = (i + 1/2)\Delta x$:

$$\frac{\partial q}{\partial x}(x_{i+1/2}) = \frac{q_{i+1} - q_i}{\Delta x} + O(\Delta x^2).$$

Throughout this book we also deal with functions sampled at midpoints, $q_{i+1/2} = q(x_{i+1/2})$, for which we can similarly write down

$$\frac{\partial q}{\partial x}(x_i) = \frac{q_{i+1/2} - q_{i-1/2}}{\Delta x} + O(\Delta x^2).$$

Higher derivatives can also be estimated. In particular, we can get a second-order accurate central finite difference for the second derivative $\partial^2 q/\partial x^2$ by writing down the Taylor series yet again:

$$q_{i+1} = q_i + \Delta x \frac{\partial q}{\partial x}(x_i) + \frac{\Delta x^2}{2}\frac{\partial^2 q}{\partial x^2}(x_i) + \frac{\Delta x^3}{6}\frac{\partial^3 q}{\partial x^3}(x_i) + O(\Delta x^4),$$

$$q_{i-1} = q_i - \Delta x \frac{\partial q}{\partial x}(x_i) + \frac{\Delta x^2}{2}\frac{\partial^2 q}{\partial x^2}(x_i) + \frac{\Delta x^3}{6}\frac{\partial^3 q}{\partial x^3}(x_i) + O(\Delta x^4).$$

The following combination cancels out most of the terms:

$$q_{i+1} - 2q_i + q_{i-1} = \Delta x^2 \frac{\partial^2 q}{\partial x^2}(x_i) + O(\Delta x^4).$$

Dividing through by Δx^2 gives the finite difference formula,

$$\frac{\partial^2 q}{\partial x^2}(x_i) = \frac{q_{i+1} - 2q_i + q_{i-1}}{\Delta x^2} + O(\Delta x^2).$$

A.2.2 Time Integration

Solving differential equations in time generally revolves around the same finite difference approach. For example, to solve the differential equation

$$\frac{\partial q}{\partial t} = f(q)$$

with initial conditions $q(0) = q^0$, we can approximate q at discrete times t^n, with $q^n = q(t^n)$. The **time step** Δt is simply the length of time between these discrete times: $\Delta t = t^{n+1} - t^n$. This time-step size may or may not stay fixed from one step to the next. The process of **time integration** is determining the approximate values $q^1, q^2, \ldots$ in sequence; it's called integration since we are approximating the solution

$$q(t) = \int_0^t f(q(\tau))\, d\tau,$$

which has an integral in it.

The simplest time integration method is **forward Euler**, based on the first one-sided finite difference formula we saw:

$$\frac{q^{n+1} - q^n}{\Delta t} = \frac{\partial q}{\partial t}(t^n) + O(\Delta t).$$

Plugging in the differential equation and rearranging gives the formula for the new value q^{n+1} based on the previous value q^n:

$$q^{n+1} = q^n + \Delta t f(q^n)$$

This is only first-order accurate, however.

This book makes use of a few more advanced time integration schemes, such as Runge-Kutta methods. The Runge-Kutta family gets higher-order accuracy and other numerical advantages by evaluating f at several points during a time step. For example, one of the classic second-order accurate Runge-Kutta methods can be written as

$$\begin{aligned} q^{n+1/2} &= q^n + \tfrac{1}{2}\Delta t f(q^n), \\ q^{n+1} &= q^n + \Delta t f(q^{n+1/2}). \end{aligned}$$

Probably the best third-order accurate Runge-Kutta formula is due to Ralston [Ral62]:

$$\begin{aligned} k_1 &= f(q^n), \\ k_2 &= f(q^n + \tfrac{1}{2}\Delta t k_1), \\ k_3 &= f(q^n + \tfrac{3}{4}\Delta t k_2), \\ q^{n+1} &= q^n + \tfrac{2}{9}\Delta t k_1 + \tfrac{3}{9}\Delta t k_2 + \tfrac{4}{9}\Delta t k_3. \end{aligned}$$

These schemes are not easy to derive in general!

Many time integration schemes come with a caveat that unless Δt is chosen small enough, the computed solution exponentially blows up despite the exact solution staying bounded. This is termed a **stability time-step restriction**. For some problems, a time integration scheme may even be unstable no matter how small Δt is: both forward Euler and the second-order accurate Runge-Kutta scheme above suffer from this flaw in some cases. The third-order accurate Runge-Kutta scheme may be considered the simplest general-purpose method as a result.

B

Derivations

B.1 The Incompressible Euler Equations

The classic derivation of the incompressible Euler equations is based on conservation of mass and momentum. Consider an arbitrary but fixed region of space Ω, in the fluid. The mass of the fluid in Ω is

$$M = \iiint_\Omega \rho,$$

and the total momentum of the fluid in Ω is

$$\vec{P} = \iiint_\Omega \rho\vec{u}.$$

The rate of change of M, as fluid flows in or out of Ω, is given by the integral around the boundary of the speed at which mass is entering or exiting, since mass cannot be created or destroyed inside Ω:

$$\frac{\partial M}{\partial t} = -\iint_{\partial\Omega} \rho\vec{u}\cdot\hat{n}.$$

Here $\hat{n}$ is the outward-pointing normal. We can transform this into a volume integral with the divergence theorem:

$$\frac{\partial M}{\partial t} = -\iiint_\Omega \nabla\cdot(\rho\vec{u}).$$

Expanding M and differentiating with respect to time (recalling that Ω is fixed) gives

$$\iiint_\Omega \frac{\partial\rho}{\partial t} = -\iiint_\Omega \nabla\cdot(\rho\vec{u}).$$

Since this is true for any region Ω, the integrands must match:

$$\frac{\partial\rho}{\partial t} + \nabla\cdot(\rho\vec{u}) = 0.$$

This is called the **continuity equation**. For an incompressible fluid the material derivative of density $D\rho/Dt$ is zero, i.e.,

$$\frac{\partial \rho}{\partial t} + \vec{u} \cdot \nabla \rho = 0.$$

Subtracting this from the continuity equation gives $\rho \nabla \cdot \vec{u} = 0$, or more simply

$$\nabla \cdot \vec{u} = 0,$$

which is termed the **incompressibility condition**. Note that this is independent of density, even for problems where fluids of different densities mix together.

We can apply the same process to the rate of change of momentum:

$$\frac{\partial \vec{P}}{\partial t} = \iiint_\Omega \frac{\partial (\rho \vec{u})}{\partial t}.$$

Momentum can change in two ways: the transport of fluid across the boundary and a net force $\vec{F}$ applied to region. The transport of momentum with the fluid is the boundary integral of momentum $\rho\vec{u}$ times the speed of the fluid through the boundary:

$$-\iint_{\partial\Omega} (\rho \vec{u}) \vec{u} \cdot \hat{n}.$$

(The negative sign comes from the fact that the normal is outward-pointing.) There are two forces in play for an inviscid fluid: pressure p on the boundary and gravity ρg throughout the region:

$$\vec{F} = -\iint_{\partial\Omega} p\hat{n} + \iiint_\Omega \rho g.$$

(Again, we get a negative sign in front of the pressure integral since the normal is outward-pointing.) Combining all these terms we get

$$\iiint_\Omega \frac{\partial (\rho \vec{u})}{\partial t} = -\iint_{\partial\Omega} (\rho \vec{u}) \vec{u} \cdot \hat{n} - \iint_{\partial\Omega} p\hat{n} + \iiint_\Omega \rho g.$$

Transforming the boundary integrals into volume integrals with the Fundamental Theorem of Calculus and rearranging gives

$$\iiint_\Omega \frac{\partial (\rho \vec{u})}{\partial t} + \iiint_\Omega \nabla \cdot (\rho \vec{u} \otimes \vec{u}) + \iiint_\Omega \nabla p = \iiint_\Omega \rho g.$$

Here $\vec{u} \otimes \vec{u}$ is the 3×3 outer-product matrix. Again, since this is true for any arbitrary region Ω, the integrands must be equal:

$$\frac{\partial (\rho \vec{u})}{\partial t} + \nabla \cdot (\rho \vec{u} \times \vec{u}) + \nabla p = \rho g.$$

This is the **conservation law** form of the **momentum equation**. Using the product rule of differentiation, exploiting $D\rho/Dt = 0$ for an incompressible fluid, and dividing through by ρ, this can readily be reduced to the form of the momentum equation used in the rest of this book.

Bibliography

[AN05] A. Angelidis and F. Neyret. Simulation of smoke based on vortex filament primitives. In *Proc. ACM SIGGRAPH/Eurographics Symp. Comp. Anim.*, 2005.

[ANSN06] A. Angelidis, F. Neyret, K. Singh, and D. Nowrouzezahrai. A controllable, fast and stable basis for vortex based smoke simulation. In *Proc. ACM SIGGRAPH/Eurographics Symp. Comp. Anim.*, 2006.

[APKG07] B. Adams, M. Pauly, R. Keiser, and L. Guibas. Adaptively sampled particle fluids. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 26(3), 2007.

[BB06] T. Brochu and R. Bridson. Fluid animation with explicit surface meshes. In *Proc. ACM SIGGRAPH/Eurographics Symp. Comp. Anim. (posters)*, 2006.

[BB08] Christopher Batty and Robert Bridson. Accurate viscous free surfaces for buckling, coiling, and rotating liquids. In *Proc. ACM SIGGRAPH/Eurographics Symp. Comp. Anim.*, SCA '08, pages 219–228, 2008.

[BB09] Tyson Brochu and Robert Bridson. Robust topological operations for dynamic explicit surfaces. *SIAM J. Sci. Comput.*, 31(4):2472–2493, 2009.

[BB10] Christopher Batty and Robert Bridson. A simple finite difference method for time-dependent, variable coefficient stokes flow on irregular domains. Technical report, arXiv:1010.2832, 2010.

[BB12] Landon Boyd and Robert Bridson. Multiflip for energetic two-phase fluid simulation. *ACM Trans. Graph.*, 31(2):16:1–16:12, 2012.

[BBB07] C. Batty, F. Bertails, and R. Bridson. A fast variational framework for accurate solid-fluid coupling. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 26(3), 2007.

[BBB10] Tyson Brochu, Christopher Batty, and Robert Bridson. Matching fluid simulation elements to surface geometry and topology. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 29(4):47:1–47:9, 2010.

[BGL05] M. Benzi, G. H. Golub, and J. Liesen. Numerical solution of saddle point problems. *Acta Numerica*, pages 1–137, 2005.

[BH86] J. Barnes and P. Hut. A hierarchical O(N log N) force-calculation algorithm. *Nature*, 324(4):446–449, 1986.

[BHN07] R. Bridson, J. Hourihan, and M. Nordenstam. Curl-noise for procedural fluid flow. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 26(3), 2007.

[BKB12] Tyson Brochu, Todd Keeler, and Robert Bridson. Linear-time smoke animation with vortex sheet meshes. In *Proc. ACM SIGGRAPH/Eurographics Symp. Comp. Anim.*, SCA '12, pages 87–95, 2012.

[Bli82] J. Blinn. A generalization of algebraic surface drawing. *ACM Trans. Graph.*, 1(3):235–256, 1982.

[Bon07] M. Bonner. Compressible subsonic flow on a staggered grid. Master's thesis, UBC Dept. Computer Science, 2007.

[BR86] J. U. Brackbill and H. M. Ruppel. FLIP: a method for adaptively zoned, particle-in-cell calculuations of fluid flows in two dimensions. *J. Comp. Phys.*, 65:314–343, 1986.

[Bri02] Robert Bridson. Digital sculpture. Cs348b course project, Stanford University, 2002. http://graphics.stanford.edu/courses/cs348b-competition/cs348b-02/bridson/.

[Bri03] R. Bridson. *Computational Aspects of Dynamic Surfaces*. PhD thesis, Stanford University, June 2003.

[BT00] Robert Bridson and Wei-Pai Tang. A structural diagnosis of some ic orderings. *SIAM J. Sci. Comput.*, 22(5):1527–1532, May 2000.

[CFL28] R. Courant, K. Friedrichs, and H. Lewy. Über die partiellen differenzengleichungen der mathematischen physik. *Mathematische Annalen*, 100(1):32–74, 1928.

[CGFO06] N. Chentanez, T. G. Goktekin, B. E. Feldman, and J. F. O'Brien. Simultaneous coupling of fluids and deformable bodies. In *Proc. ACM SIGGRAPH/Eurographics Symp. Comp. Anim.*, pages 83–89, 2006.

[Cho73] A. J. Chorin. Numerical study of slightly viscous flow. *J. Fluid Mechanics*, 57(4):785–796, 1973.

[Chr73] I. P. Christiansen. Numerical simulation of hydrodynamics by the method of point vortices. *J. Comp. Phys.*, 13(3):363–379, 1973.

[CK08] Georges-Henri Cottet and Petros D. Koumoutsakos. *Vortex Methods: Theory and Practice*. Cambridge University Press, 2008.

[CMF12] Nuttapong Chentanez and Matthias Mueller-Fischer. A multigrid fluid pressure solver handling separating solid boundary conditions. *IEEE Trans. Vis. Comp. Graph.*, 18(8):1191–1201, 2012.

[CMIT02] M. Carlson, P. Mucha, R. Van Horn III, and G. Turk. Melting and flowing. In *Proc. ACM SIGGRAPH/Eurographics Symp. Comp. Anim.*, pages 167–174, 2002.

[CMT04] M. Carlson, P. J. Mucha, and G. Turk. Rigid fluid: animating the interplay between rigid bodies and fluid. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 23:377–384, 2004.

[Cor05] R. Corbett. Point-based level sets and progress towards unorganised particle based fluids. Master's thesis, UBC Dept. Computer Science, 2005.

[DBG14] Fang Da, Christopher Batty, and Eitan Grinspun. Multimaterial mesh-based surface tracking. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 33(4):112:1–112:11, 2014.

[DC96] M. Desbrun and M.-P. Cani. Smoothed particles: A new paradigm for animating highly deformable bodies. In *Comput. Anim. and Sim. '96 (Proc. of EG Workshop on Anim. and Sim.)*, pages 61–76, 1996.

[DeW05] I. DeWolf. Divergence-free noise. 2005.

[DM89] Iain Duff and Gerard Meurant. The effect of ordering on preconditioned conjugate gradients. *BIT*, 29:635–637, 1989.

[Doi91] S. Doi. On parallelism and convergence of incomplete lu factorizations. *Appl. Numer. Math.*, 7(5):417–436, June 1991.

[Dür88] M. J. Dürst. Additional reference to "marching cubes" (letters). *Comput. Graph*, 22(4):72–73, 1988.

[EB15] Essex Edwards and Robert Bridson. The discretely-discontinuous Galerkin coarse grid for domain decomposition. Technical report, arXiv:1504.00907, 2015.

[Eij91] Victor Eijkhout. Analysis of parallel incomplete point factorizations. *Lin. Alg. Appl.*, 154:154–156, 1991.

[EM90] Herbert Edelsbrunner and Ernst Peter Mcke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Graph.*, 9(1):66–104, 1990.

[ETK+07] S. Elcott, Y. Tong, E. Kanso, P. Schröder, and M. Desbrun. Stable, circulation-preserving, simplicial fluids. *ACM Trans. Graph.*, 26(1), 2007.

[FF01] N. Foster and R. Fedkiw. Practical animation of liquids. In *Proc. SIGGRAPH*, pages 23–30, 2001.

[FL04] R. Fattal and D. Lischinski. Target-driven smoke animation. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 23(3):441–448, 2004.

[FM96] N. Foster and D. Metaxas. Realistic animation of liquids. *Graph. Models and Image Processing*, 58:471–483, 1996.

[FM97] N. Foster and D. Metaxas. Modeling the motion of a hot, turbulent gas. In *Proc. SIGGRAPH*, pages 181–188, 1997.

[FOA03] B. E. Feldman, J. F. O'Brien, and O. Arikan. Animating suspended particle explosions. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 22:708–715, 2003.

[FR86] A. Fournier and W. T. Reeves. A simple model of ocean waves. In *Proc. SIGGRAPH*, pages 75–84, 1986.

[FSJ01] R. Fedkiw, J. Stam, and H. W. Jensen. Visual simulation of smoke. In *Proc. SIGGRAPH*, pages 15–22, 2001.

[Gam95] M. N. Gamito. Two dimensional Simulation of Gaseous Phenomena Using Vortex Particles. In *Proc. of the 6th Eurographics Workshop on Comput. Anim. and Sim.*, pages 3–15. Springer-Verlag, 1995.

[GBO04] T. G. Goktekin, A. W. Bargteil, and J. F. O'Brien. A method for animating viscoelastic fluids. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 23:463–468, 2004.

[GFCK02] F. Gibou, R. Fedkiw, L.-T. Cheng, and M. Kang. A second-order-accurate symmetric discretization of the Poisson equation on irregular domains. *J. Comp. Phys.*, 176:205–227, 2002.

[GLG95] M. N. Gamito, P. F. Lopes, and M. R. Gomes. Two-dimensional simulation of gaseous phenomena using vortex particles. In Dimitri Terzopoulos and Daniel Thalmann, editors, *Computer Animation and Simulation '95*, pages 2–15. Springer-Verlag, 1995.

[GR87] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *J. Comput. Phys.*, 73(2):325–348, 1987.

[GSLF05] E. Guendelman, A. Selle, F. Losasso, and R. Fedkiw. Coupling water and smoke to thin deformable and rigid shells. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 24(3):973–981, 2005.

[Har63] F. H. Harlow. The particle-in-cell method for numerical solution of problems in fluid dynamics. In *Experimental arithmetic, high-speed computations and mathematics*, 1963.

[Har04] F. H. Harlow. Fluid dynamics in group T-3 Los Alamos National Laboratory. *J. Comput. Phys.*, 195(2):414–433, 2004.

[Hor15] Christopher Horvath. Empirical directional wave spectra for computer graphics. In *Proc. Digital Production Symposium*, 2015.

[HSF07] J.-M. Hong, T. Shinar, and R. Fedkiw. Wrinkled flames and cellular patterns. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 26(3), 2007.

[HW65] F. Harlow and J. Welch. Numerical Calculation of Time-Dependent Viscous Incompressible Flow of Fluid with Free Surface. *Phys. Fluids*, 8:2182–2189, 1965.

[Irv07] G. Irving. *Methods for the Physically Based Simulation of Solids and Fluids*. PhD thesis, Stanford University Dept. Computer Science, 2007.

[JBS06] M. Jones, A. Bærentzen, and M. Sramek. 3D distance fields: A survey of techniques and applications. *IEEE Trans. Vis. Comp. Graphics*, 2006.

[JKSH13] Alec Jacobson, Ladislav Kavan, , and Olga Sorkine-Hornung. Robust inside-outside segmentation using generalized winding numbers. *ACM Transactions on Graphics (proceedings of ACM SIGGRAPH)*, 32(4):33:1–33:12, 2013.

[JL72] W. P. Jones and B. E. Launder. The prediction of laminarization with a two-equation model of turbulence. *Int. J. Heat Mass Transfer*, 15:301–314, 1972.

[Jon95] Mark W. Jones. 3d distance from a point to a triangle. Technical report, Department of Computer Science, University of Wales, 1995.

[KCR08] Petros Koumoutsakos, Georges-Henri Cottet, and Diego Rossinelli. Flow simulations using particles: Bridging computer graphics and cfd. In *ACM SIGGRAPH 2008 Classes*, SIGGRAPH '08, pages 25:1–25:73, 2008.

[KFL00] M. Kang, R. Fedkiw, and X.-D. Liu. A boundary condition capturing method for multiphase incompressible flow. *J. Sci. Comput.*, 15:323–360, 2000.

[KH04] J. Kniss and D. Hart. Volume effects: modeling smoke, fire, and clouds, 2004. Section from ACM SIGGRAPH 2004 courses, *Real-Time Volume Graphics*, http://www.cs.unm.edu/ jmk/sig04_modeling.ppt.

[KLL+07] B. Kim, Y. Liu, I. Llamas, X. Jiao, and J. Rossignac. Simulation of bubbles in foam with the volume control method. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 26(3), 2007.

[KM90] Michael Kass and Gavin Miller. Rapid, stable fluid dynamics for computer graphics. In *Proc. ACM SIGGRAPH*, pages 49–57, 1990.

[KTJG08] Theodore Kim, Nils Thürey, Doug James, and Markus Gross. Wavelet turbulence for fluid simulation. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 27(3):50:1–50:6, 2008.

[LC87] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Proc. ACM SIGGRAPH*, pages 163–169, 1987.

[LGF04] F. Losasso, F. Gibou, and R. Fedkiw. Simulating water and smoke with an octree data structure. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 23:457–462, 2004.

[LIGF06] F. Losasso, G. Irving, E. Guendelman, and R. Fedkiw. Melting and burning solids into liquids and gases. *IEEE Trans. Vis. Graph.*, 12:343–352, 2006.

[LKHW03] A.E. Lefohn, J.M. Kniss, C.D. Hansen, and R.T. Whitaker. Interactive deformation and visualization of level set surfaces using graphics hardware. In *IEEE Visualization 2003*, pages 75–82, October 2003.

[LS74] B. E. Launder and B. I. Sharma. Application of the energy-dissipation model of turbulence to the calculation of flow near a spinning disc. *Lett. Heat Mass Trans.*, 1:131–138, 1974.

[LvdP02] A. T. Layton and M. van de Panne. A numerically efficient and stable algorithm for animating water waves. *The Visual Computer*, 18(1):41–53, 2002.

[MÖ9] Matthias Müller. Fast and robust tracking of fluid surfaces. In *Proc. ACM SIGGRAPH/Eurographics Symp. Comp. Anim.*, SCA '09, pages 237–245, 2009.

[MCG03] M. Müller, D. Charypar, and M. Gross. Particle-based fluid simulation for interactive applications. In *Proc. ACM SIGGRAPH/Eurographics Symp. Comp. Anim.*, pages 154–159, 2003.

[MCZ07] Ken Museth, Michael Clive, and Nafees Bin Zafar. Blobtacular: surfacing particle system in "pirates of the caribbean 3". In *ACM SIGGRAPH 2007 sketches*, SIGGRAPH '07, 2007.

[MEB+12] M. K. Misztal, K. Erleben, A. Bargteil, J. Fursund, B. Bunch Christensen, J. A. Bærentzen, and R. Bridson. Multiphase flow of immiscible fluids on unstructured moving meshes. In *Proc. ACM SIGGRAPH/Eurographics Symp. Comp. Anim.*, SCA '12, pages 97–106, 2012.

[MK02] Z. Melek and J. Keyser. Interactive simulation of fire. In *Proc. Pacific Graphics*, pages 431–432, 2002.

[MK03] Z. Melek and J. Keyser. Interactive simulation of burning objects. In *Proc. Pacific Graphics*, pages 462–466, 2003.

[Mus11] Ken Museth. Db+grid: a novel dynamic blocked grid for sparse high-resolution volumes and level sets. In *ACM SIGGRAPH 2011 Talks*, SIGGRAPH '11, pages 51:1–51:1, 2011.

[Mus13] Ken Museth. Vdb: High-resolution sparse volumes with dynamic topology. *ACM Trans. Graph.*, 32(3):27:1–27:22, 2013.

[MW99] Heinrich Müller and Michael Wehle. Visualization of implicit surfaces using adaptive tetrahedrizations. In *Dagstuhl '97, Scientific Visualization*, pages 243–250, 1999.

[NFJ02] D. Q. Nguyen, R. Fedkiw, and H. W. Jensen. Physically based modeling and animation of fire. *ACM Trans. Graph. (Proc. SIGGRAPH)*, pages 721–728, 2002.

[NMG09] Yen Ting Ng, Chohong Min, and Frédéric Gibou. An efficient fluidsolid coupling algorithm for single-phase flows. *J. Comp. Phys.*, 228(23):8807 – 8829, 2009.

[NSCL08] Rahul Narain, Jason Sewall, Mark Carlson, and Ming C. Lin. Fast animation of turbulence using energy transport and procedural synthesis. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*, 27(5):166:1–166:8, 2008.

[OF02] S. Osher and R. Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces*. Springer-Verlag, 2002. New York, NY.

[OH95] J. F. O'Brien and J. K. Hodgins. Dynamic simulation of splashing fluids. In *Proc. Computer Animation*, page 198, 1995.

[PAKF13] Saket Patkar, Mridul Aanjaneya, Dmitriy Karpman, and Ronald Fedkiw. A hybrid lagrangian-eulerian formulation for bubble generation and dynamics. In *Proc. ACM SIGGRAPH/Eurographics Symp. Comp. Anim.*, SCA '13, pages 105–114, 2013.

[Pes02] C. S. Peskin. The immersed boundary method. *Acta Numerica*, 11:479–517, 2002.

[PK05] S. I. Park and M. J. Kim. Vortex fluid for gaseous phenomena. In *Proc. ACM SIGGRAPH/Eurographics Symp. Comp. Anim.*, 2005.

[PN01] K. Perlin and F. Neyret. Flow noise. In *ACM SIGGRAPH Technical Sketches and Applications*, page 187, 2001. http://www-evasion.imag.fr/Publications/2001/PN01/.

[Pop00] S. B. Pope. *Turbulent flows*. Cambridge University Press, 2000.

[PTB+03] S. Premoze, T. Tasdizen, J. Bigler, A. Lefohn, and R. Whitaker. Particle–based simulation of fluids. In *Comp. Graph. Forum (Eurographics Proc.)*, volume 22, pages 401–410, 2003.

[PTC+10] Tobias Pfaff, Nils Thuerey, Jonathan Cohen, Sarah Tariq, and Markus Gross. Scalable fluid simulation using anisotropic turbulence particles. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*, 29(6):174:1–174:8, 2010.

[PTG12] Tobias Pfaff, Nils Thuerey, and Markus Gross. Lagrangian vortex sheets for animating fluids. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 31(4):112:1–112:8, 2012.

[Ral62] A. Ralston. Runge-Kutta methods with minimum error bounds. *Mathematics of Computation*, 16(80):431–437, 1962.

[RNGF03] N. Rasmussen, D. Nguyen, W. Geiger, and R. Fedkiw. Smoke simulation for large scale phenomena. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 22:703–707, 2003.

[SB08] H. Schechter and R. Bridson. Evolving sub-grid turbulence for smoke animation. In *Proc. ACM SIGGRAPH/Eurographics Symp. Comp. Anim.*, SCA '08, pages 1–7, 2008.

[SBG04] B. Smith, P. Bjorstad, and W. Gropp. *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, 2004.

[Set96] J. Sethian. A fast marching level set method for monotonically advancing fronts. *Proc. Natl. Acad. Sci.*, 93:1591–1595, 1996.

[SF92] M. Shinya and A. Fournier. Stochastic motion: Motion under the influence of wind. In *Proc. Eurographics*, pages 119–128, 1992.

[SF93] J. Stam and E. Fiume. Turbulent wind fields for gaseous phenomena. In *Proc. ACM SIGGRAPH*, pages 369–376, 1993.

[She97] Jonathan Richard Shewchuk. Adaptive precision floating-point arithmetic and fast robust geometric predicates. *Discrete & Computational Geometry*, 18:305–363, 1997.

[SO09] Mark Sussman and Mitsuhiro Ohta. A stable and efficient method for treating surface tension in incompressible two-phase flow. *SIAM J. Sci. Comput.*, 31(4):2447–2471, 2009.

[SRF05] A. Selle, N. Rasmussen, and R. Fedkiw. A vortex particle method for smoke, water and explosions. *ACM Trans. Graph. (Proc. SIGGRAPH)*, pages 910–914, 2005.

[Sta99] J. Stam. Stable fluids. In *Proc. SIGGRAPH*, pages 121–128, 1999.

[Sto07] Mark J. Stock. Summary of vortex methods literature. Technical report, 2007.

[SU94] J. Steinhoff and D. Underhill. Modification of the Euler Equations for "Vorticity Confinement": Application to the Computation of Interacting Vortex Rings. *Phys. of Fluids*, 6(8):2738–2744, 1994.

[Tes04] J. Tessendorf. Simulating ocean water. *SIGGRAPH Course Notes*, 1999–2004.

[THK02] T. Takahashi, U. Heihachi, and A. Kunimatsu. The simulation of fluid-rigid body interaction. In *Proc. SIGGRAPH Sketches & applications*, 2002.

[TMSG07] N. Thuerey, M. Müller, S. Schirm, and M. Gross. Real-time breaking waves for shallow water simulations. In *Proc. Pacific Graphics*, 2007.

[Tsa02] Y.-H. R. Tsai. Rapid and accurate computation of the distance function using grids. *J. Comput. Phys.*, 178(1):175–195, 2002.

[Tsi95] J. Tsitsiklis. Efficient algorithms for globally optimal trajectories. *IEEE Trans. on Automatic Control*, 40:1528–1538, 1995.

[Ü01] Alper Üngör. Tiling 3D euclidean space with acute tetrahedra. In *Proc. 13th Canadian Conference on Computational Geometry (CCCG'01)*, pages 169–172, 2001.

[vFTS06] W. von Funck, H. Theisel, and H.-P. Seidel. Vector field based shape deformations. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 25(3):1118–1125, 2006.

[Wil08] Brent Williams. Fluid surface reconstruction from particles. Master's thesis, University of British Columbia, 2008.

[WMFB11] Chris Wojtan, Matthias Müller-Fischer, and Tyson Brochu. Liquid simulation with mesh-based surface tracking. In *ACM SIGGRAPH 2011 Courses*, SIGGRAPH '11, pages 8:1–8:84, 2011.

[WMT05] H. Wang, P. J. Mucha, and G. Turk. Water drops on surfaces. *ACM Trans. Graph. (Proc. SIGGRAPH)*, pages 921–929, 2005.

[WMW86] Geoff Wyvill, Craig McPheeters, and Brian Wyvill. Data Structure for Soft Objects. *The Visual Computer*, 2(4):227–234, February 1986.

[WP10] Steffen Weißmann and Ulrich Pinkall. Filament-based smoke with vortex shedding and variational reconnection. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 29(4):115:1–115:12, 2010.

[WTGT10] Chris Wojtan, Nils Thürey, Markus Gross, and Greg Turk. Physics-inspired topology changes for thin fluid features. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 29(4):50:1–50:8, 2010.

[YHK07] Cem Yuksel, Donald H. House, and John Keyser. Wave particles. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 26(3), 2007.

[YUM86] L. Yaeger, C. Upson, and R. Myers. Combining physical and visual simulation—creation of the planet jupiter for the film 2010. In *Proc. ACM SIGGRAPH*, pages 85–93, 1986.

[ZB05] Y. Zhu and R. Bridson. Animating sand as a fluid. *ACM Trans. Graph. (Proc. SIGGRAPH)*, pages 965–972, 2005.

[ZB14] Xinxin Zhang and Robert Bridson. A pppm fast summation method for fluids and beyond. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*, 33(6):206:1–206:11, 2014.

[Zha05] H. Zhao. A fast sweeping method for Eikonal equations. *Math. Comp.*, 74:603–627, 2005.