Fluid simulation

Valentin Miu

Abstract

This essay presents summarizes several basic concepts of fluid simulations, discusses treatment of different types of fluids, and ...

1 Introduction

...history

2 The Navier-Stokes equations

Liquid fluids, and to a lesser extend gases, can be considered incompressible in most simulations. The behaviour of incompressible fluids are given by the Navier-Stokes equations (1, 2):

$$\frac{\delta \vec{u}}{\delta t} + \vec{u} \cdot \nabla \vec{u} + \frac{1}{\rho} \nabla p = \vec{g} + \nu \nabla \cdot \nabla \vec{u} \tag{1}$$

$$\nabla \cdot \vec{u} = 0 \tag{2}$$

The Navier-Stokes equation can be solved from the Eulerian or Lagrangian viewpoints. In the former, the quantities are sampled at fixed points; in the latter, the quantities are sampled at the positions of particles moving with the flow (the "snowman POV" and the "snowflake POV", respectively). As a result, fluid simulations can be grid-based (for Eulerian) or particle-based (for Lagrangian). In many cases, the two are used in combination or in hybrid algorithms[2-way coupled SPH]. ()

3 A simple grid-based Navier-Stokes simulation method

The method used in [] is described. Due to the velocity divergence dependency, it is useful in grid-based simulations to use a staggered Marker-and-Cell (MAC) grid. This stores the pressure values at the centre of the grid cells, and the xyz-components of the velocity $\vec{u}$ (u, v, and w) at the centres of the grid walls (as in Figure 1). This allows for the numerical computation of velocity divergence at the center of the cell, exactly where the pressure is sampled (3. The velocity value itself at this point is determined by trilinear (linear in three dimensions) interpolation.

$$\nabla \vec{u}_{i,j,k} \approx \frac{u_{i+\frac{1}{2},j,k} - u_{i-\frac{1}{2},j,k}}{2} + \frac{v_{i,j+\frac{1}{2},k} - v_{i,j-\frac{1}{2},k}}{2} + \frac{w_{i,j,k+\frac{1}{2}} - w_{i,j,k-\frac{1}{2}}}{2} \tag{3}$$

Solving the Navier-Stokes equation can be simplified by breaking the equation down into three parts: the advection, body, and projection equations. Their respective processes `advect()`, `body()` and `project()` are applied in sequence for each time step.

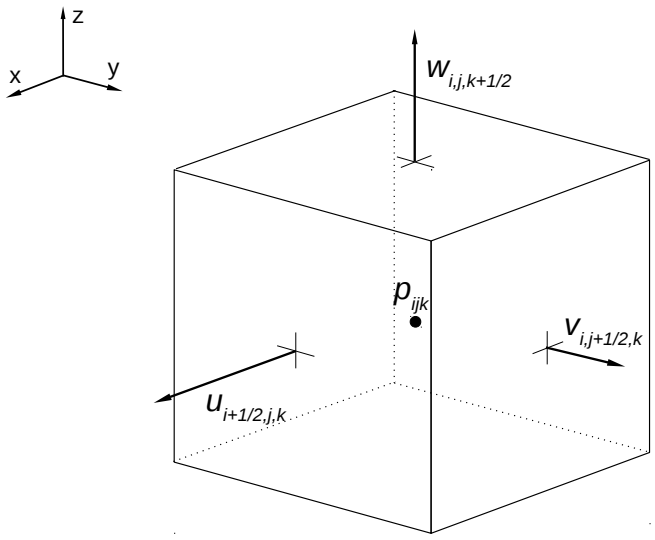

Figure 1: Data storage in one cell of a staggered MAC grid.

When applied to the entire fluid, the advection equation enforces the conservation of energy (the fluid mass undergoes no net acceleration without the application of a net force). The `advect()` process applies the velocity field to the quantity q for a time dt. This quantity may be velocity itself, pressure, or another quantity relevant to the particular fluid (such as temperature for smoke simulation).
Since in truth the fluid is acted on by the net force of gravity (and perhaps other forces), the body process simply adds $\vec{g}dt$ to the velocity field (all the members of the grid).
The incompressibility of the fluid is enforced by the `project()` process, which solves the Poisson equation by subtracting the correct pressure gradient from the output velocity grid of `body()` (4).

$$\vec{u}^{n+1} = \vec{u}_B^n - \Delta t \frac{1}{\rho} \nabla p \tag{4}$$

In numerical form, this results in a series of n equations, where n is the number of cells in the grid. If $\vec{p}$ is a column vector containing all the pressure unknowns, and $\vec{d}$ is a vector of the corresponding divergence values, the series of equations can be written as

$$Ap = d, \tag{5}$$

where A is a symmetric sparse 3D matrix known as the seven-point Lagrangian matrix. This can be solved using the Modified Incomplete Cholesky Conjugate Gradient, Level Zero algorithm (MICCG(0)). This involves using the Preconditioned Conjugate Gradient algorithm (or PCG, shown in Figure A1),
with the `applyPreconditioner()` function shown in Figure A2. Since A is a symmetric sparse matrix with large-weight components only near and on the main diagonal, its storage can be optimized to $O(xSize * ySize * zSize)$ space.
If a fluid moves into a new space where the velocity has not been previously defined, it needs to be extrapolated from the nearest valid fluid cell. It is common for velocity to be sampled within a certain distance of the fluid surface in order to reduce such problems.

4 Boundary conditions

Fluid simulations are commonly expected to account for obstacles in the fluid volume.

5 Smoke

Different types of fluids require specific considerations. For smoke, the body force is replaced with a temperature-dependent buoyancy force. Using the Boussinesq approximation of constant pressure, and assuming a linear dependence on only the temperature and smoke concentration. (6

$$1 + 1 = 2 \tag{6}$$

When using the semi-Lagrangian advection scheme, numerical dissipation leads to rapid destruction of the interesting features of smoke, such as vortices. This can be counteracted by creating a dependence on the vorticity $\vec{\omega} = \nabla \times \vec{u}$ (to boost vortex strength), or by using a sharper tricubic interpolant instead of the trilinear one.

6 Water

For the simulation of water, the issue of keeping track of the surface arises. The most basic method, suggested by Marker-and-Cell in 1965, involves using marker particles inside the grid volume. The presence of such a particle marks the presence of water at that point as the particles are moved by advection in the velocity field of the grid. However, this leads to poorly-defined, "blobby" water surfaces.
A superior method is the use of level sets. In their case, a function $\phi(x)$ is defined by values sampled at grid cells; the surface is described by $\phi(x) = 0$. This allows for easy modelling of smooth surfaces, as well as determining whether a point is inside the fluid volume, by simple interpolation to find the value of $\phi(x)$ at that point.
Since a given surface may be defined by multiple level sets, the function $\phi(x)$ offering the most stability is chosen. It is given by the gradient condition 7.

$$\|\nabla\phi(\vec{x})\| = 1 \tag{7}$$

This is enforced by using the signed distance function for $\phi(x)$. (what is that?)
A caveat of level sets is that when combined with the advection process, numerical dissipation leads to the rapid destruction of small features (like small droplets or ripples). This can be solved by using par

7 Smoothed Particle Hydrodynamics

Although not nearly as computationally efficient in dealing with incompressible fluid volume, Smoothed Particle Hydrodynamics is far better than grid-based methods at creating visually pleasing fluid surfaces. While small interesting features like fine droplets and thin films are removed by numerical dissipation and limited resolution in grids, SPH is unaffected by these issues. As a result, modern fluid simulations use grid-based Navier-Stokes combined with SPH with particles seeded near fluid boundaries.
In SPH, every particle has an associated normalized smoothing kernel function $W(r)$, where r is the distance from the particle. To reduce computational cost, the function is given a cutoff point

r_c beyond which it is zero. Therefore, for one particle, only the particles in a vicinity of radius r_c need to be considered.
A commonly used smoothing kernel is the poly6 kernel (8):

$$W_{poly6}(r) = \frac{315}{64\pi d^9} \cdot \begin{cases} (r_c^2 - r^2) & 0 \leq r \leq r_c \\ 0 & r > r_c \end{cases} \tag{8}$$

SPH makes the Navier-Stokes equations trivial, as mass conservation can be enforced simply by keeping the number of particles constant. For liquid fluids, the boundary is again found through the level set method applied to the particle field.
SPH is also useful in modelling compressible fluids such as sea foam, leading to impressive results if properly coupled with

8 Other fluid simulation types

For real-time simulations in games, the aforementioned methods are generally much too computationally expensive, although small-scale SPH methods are possible (the chinese dude citation). Heightfield approximations use the two-dimensional wave equation as a displacement map in the z direction. This however does not allow for multiple z solutions per (x,y) point, as a breaking wave would for example require.
Ocean surfaces can also be generated using procedural animation, as the sum of various sinusoidal functions of varying direction, amplitude, and frequency. This is not well suited for interaction with boundaries or solid objects in the water; this shortcoming can be obfuscated by texturing foam onto the mesh near obstacles and/or using particle emitters.
These types of simulation are far less computationally expensive than 3D Navier-Stokes methods; this makes them suitable for real-time simulations in games. Still, small-scale SPH can also be used in real-time [chinese guy citation], and so can grid-based methods (by using the Jacobi method in the projection step.

9 Fluid simulations through deep learning

The most computationally demanding step of fluid simulations is projection. This has recently been shown to be surprisingly well handled by neural networks trained on exact fluid solvers, even when the initial conditions of fluid in the solver and network differ. Furthermore, unlike traditional exact solvers, the complexity of this approach is not dependent on boundary conditions.
[maybe put a picture here reflecting it]

Annex

```
// "·" is the dot product of two vectors; "*" is scalar multiplication
p = 0
r = d
z = applyPreconditioner(r)
s = z
σ = z · r
iterations = 0
while (p not within tolerance) and (iterations < maximumIterations)
    z = A · s
    α = ρ / (z · s)
    p = p + α * s
    r = r - α * z
    // if the element of r with the greatest absolute value is within
    // tolerance, take current p as the solution
    if (max(|r|) ≤ tol) return p
    σ_new = z · r
    β = σ_new / ρ
    s = z + β * s
    σ = σ_new
// maximum iterations have been exceeded
return r
```

Figure A1: The PCG algorithm.

```
//set tuning constant
τ = 0.97

//calculate the preconditioner
for i = 1 to xSize, j = 1 to ySize, k = 1 to zSize
  if (grid cell has fluid in it)
    e = A_(i,j,k)(i,j,k) - (A_(i+1,j,k)(i-1,j,k) * precon_i-1,j,k)^2
                         - (A_(i,j+1,k)(i,j-1,k) * precon_i,j-1,k)^2
                         - (A_(i,j,k+1)(i,j,k-1) * precon_i,j,k-1)^2
                - τ * (A_(i+1,j,k)(i-1,j,k) * (A_(i,j+1,k)(i-1,j,k) + A_(i,j,k+1)(i-1,j,k)) * precon^2_i-1,j,k
                - τ * (A_(i,j+1,k)(i,j-1,k) * (A_(i+1,j,k)(i,j-1,k) + A_(i,j,k+1)(i,j-1,k)) * precon^2_i,j-1,k
                - τ * (A_(i,j,k+1)(i,j,k-1) * (A_(i+1,j,k)(i,j,k-1) + A_(i,j+1,k)(i,j,k-1)) * precon^2_i,j,k-1

//Solve Lq = r
for i = 1 to xSize, j = 1 to ySize, k = 1 to zSize
  if (grid cell has fluid in it)
    t = r_i,j,k - A_(i+1,j,k)(i-1,j,k) * precon_i-1,j,k * q_i-1,j,k
                - A_(i,j+1,k)(i,j-1,k) * precon_i,j-1,k * q_i,j-1,k
                - A_(i,j,k+1)(i,j,k+1) * precon_i,j,k-1 * q_i,j,k-1
    q_i,j,k = t * precon_i,j,k

//Solve L^T z = q
for i = xSize to 1, j = ySize to 1, k = zSize to 1
  if (grid cell has fluid in it)
    t = r_i,j,k - A_(i,j,k)(i,j,k) * precon_i,j,k * z_i+1,j,k
                - A_(i,j,k)(i,j,k) * precon_i,j,k * q_i,j+1,k
                - A_(i,j,k)(i,j,k) * precon_i,j,k * q_i,j,k+1
    z_i,j,k = t * precon_i,j,k
```

Figure A2: An optimized preconditioner corresponding to the weighted average (τ=97%) between MICCG(0) and regular Incomplete Cholesky (ICCG(0)).