Notes

- Smoke:
  - Fedkiw, Stam, Jensen, SIGGRAPH'01
- Water:
  - Foster, Fedkiw, SIGGRAPH'01
  - Enright, Fedkiw, Marschner, SIGGRAPH'02
- Fire:
  - Nguyen, Fedkiw, Jensen, SIGGRAPH'02

Recall: plain CG

- CG is guaranteed to converge faster than steepest descent
  - Global optimality property
- But... convergence is determined by distribution of eigenvalues
  - Widely spread out eigenvalues means sloooow solution
- How can we make it efficient?

Speeding it up

- CG generally takes as many iterations as your grid is large
  - E.g. if 30x70x40 expect to take 70 iterations (or proportional to it)
  - Though a good initial guess may reduce that a lot
- Basic issue: pressure is globally coupled - information needs to travel from one end of the grid to the other
  - Each step of CG can only go one grid point: matrix-vector multiply is core of CG
- Idea of a “preconditioner”: if we can get a routine which approximately computes $A^{-1}$, call it $M$, then solve $MAx=Mb$
  - If $M$ has global coupling, can get information around faster
  - Alternatively, improve search direction by multiplying by $M$ to point it closer to negative error
  - Alternatively, cluster eigenvalues

Preconditioners

- Lots and lots of work on how to pick an $M$
- Examples: FFT, SSOR, ADI, multigrid, sparse approximate inverses
- We’ll take a look at Incomplete Cholesky factorization
- But first, how do we change CG to take account of $M$?
  - $M$ has to be SPD, but $MA$ might not be...
PCG

- \( r=b-Ap, \ z=Mr, \ s=z \)
- \( p=z^T r, \) check if already solved
- Loop:
  - \( t=As \)
  - \( \alpha = p/(s^T t) \)
  - \( x+=\alpha s, \ r-=\alpha t, \) check for convergence
  - \( z=Mr \)
  - \( \rho_{\text{new}}=z^T r \)
  - \( \beta = \rho_{\text{new}} / \rho \)
  - \( s=z+\beta s \)
  - \( \rho = \rho_{\text{new}} \)

Cholesky

- True Gaussian elimination, which is called Cholesky factorization in the SPD case, gives \( A=LL^T \)
- \( L \) is a lower triangular matrix
- Then solving \( Ap=b \) can be done by
  - \( Lx=p, \ L^T p=x \)
  - Each solve is easy to do - triangular
- But can’t do that here since \( L \) has many more nonzeros than \( A \) - EXPENSIVE!

Incomplete Cholesky

- We only need approximate result for preconditioner
- So do Cholesky factorization, but throw away new nonzeros (set them to zero)
- Result is not exact, but pretty good
  - Instead of \( O(n) \) iterations (for an \( n^3 \) grid) we get \( O(n^{1/2}) \) iterations
- Can actually do better:
  - Modified Incomplete Cholesky
  - Same algorithm, only when we throw away nonzeros, we add them to the diagonal - better behaviour with low frequency components of pressure
  - Gets us down to \( O(n^{1/4}) \) iterations

IC(0)

- Incomplete Cholesky level 0: IC(0) is where we make sure \( L=0 \) wherever \( A=0 \)
- For this \( A \) (7-point Laplacian) with the regular grid ordering, things are nice
- Write \( A=F+D+F^T \) where \( F \) is strictly lower triangular and \( D \) is diagonal
- Then IC(0) ends up being of the form \( L=(FE^{-1} + E) \) where \( E \) is diagonal
  - We only need to compute and store \( E! \)
Computing IC(0)

- Need to find diagonal \( E \) so that \((LL^T)_{ij}=A_{ij}\) wherever \( A_{ij}\neq 0\)
- Expand out:
  - \( LL^T=F+F^T+E^2+FE^2F^T \)
- Again, for this special case, can show that last term only contributes to diagonal and elements where \( A_{ij}=0\)
- So we get the off-diagonal correct for free
- Let’s take a look at diagonal entry for grid point \( ijk \)

Diagonal Entry

- Assume we order increasing in \( i, j, k \)
- Note \( F=A \) for lower diagonal elements
- Want this to match \( A \)’s diagonal
- Then solving for next \( E_{ijk} \) in terms of previously determined ones:
  \[
  E_{ijk} = \sqrt{A_{ijk,jk} - A_{jk,i-1,j}^2 E_{i-1,jk}^2 - A_{jk,i-1,k}^2 E_{i-1,jk}^2 - A_{ijk,jk-1}^2 E_{ijk-1,j}^2}
  \]

Practicalities

- Actually only want to store inverse of \( E \)
- Note that for values of \( A \) or \( E \) off the grid, substitute zero in formula
  - In particular, can start at \( E_{000,000}=\sqrt{A_{000,000}} \)
- Modified Incomplete Cholesky looks very similar, except instead of matching diagonal entries, we match row sums
- Can squeeze out a little more performance with the “Eisenstat trick”

Viscosity

- The viscosity update (if we really need it - highly viscous fluids) is just Backwards Euler:
  \[
  \left(I - \Delta t \nabla^2 \right)u^{(3)} = u^{(2)}
  \]
- Boils down to almost the same linear system to solve!
  - Or rather, 3 similar linear systems to solve - one for each component of velocity (NOTE: solve separately, not together!)
  - Again use PCG with Incomplete Cholesky
Staggered grid advection

- Problem: velocity on a staggered grid, don’t have components where we need it for semi-Lagrangian steps
- Simple answer
  - Average velocities to get flow field where you need it, e.g. $u_{ijk} = 0.5(u_{i+1/2,jk} + u_{i-1/2,jk})$
  - So advect each component of velocity around in averaged velocity field
- Even cheaper
  - Advect averaged velocity field around (with any other quantity you care about) --- reuse interpolation coefficients!
  - But - all that averaging smears u out… more numerical viscosity!
    - worse for small $\Delta t$]

Vorticity confinement

- The interpolation errors behave like viscosity, the averaging from the staggered grid behaves like viscosity…
  - Net effect is that interesting flow structures (vortices) get smeared out
- Idea of vorticity confinement - add a fake force that spins vortices faster
  - Compute vorticity of flow, add force in direction of flow around each vortex
  - Try to cancel off some of the numerical viscosity in a stable way

Smoke

- Smoke is a bit more than just a velocity field
- Need temperature (hot air rises) and smoke density (smoke eventually falls)
- Real physics - density depends on temperature, temperature depends on viscosity and thermal conduction, …
  - We’ll ignore most of that: small scale effects
  - Boussinesq approximation: ignore density variation except in gravity term, ignore energy transfer except thermal conduction
  - We might go a step further and ignore thermal conduction - insignificant vs. numerical dissipation - but we’re also ignoring sub-grid turbulence which is really how most of the temperature gets diffused

Smoke concentration

- There’s more than just air temperature to consider too
- Smoke weighs more than air - so need to track smoke concentration
  - Also could be used for rendering (though tracing particles can give better results)
  - Point is: physics depends on smoke concentration, not just appearance
- We again ignore effect of this in all terms except gravity force
Buoyancy

- For smoke, where there is no interface, we can add $\rho g y$ to pressure (and just solve for the difference) thus cancelling out g term in equation
- All that’s left is buoyancy -- variation in vertical force due to density variation
- Density varies because of temperature change and because of smoke concentration
- Assume linear relationship (small variations)
  \[ f_{\text{bouy}} = (-\alpha s + \beta T) \]
  - T=0 is ambient temperature; $\alpha$, $\beta$ depend on g etc.

Smoke equations

- So putting it all together…
  \[ u_t + u \cdot \nabla u + \nabla p = (-\alpha s + \beta T)(0,1,0) \]
  \[ \nabla \cdot u = 0 \]
  \[ T_t + u \cdot \nabla T = k \nabla^2 T \]
  \[ s_t + u \cdot \nabla s = 0 \]
- We know how to solve the u part, using old values for s and T
- Advecting s and T around is simple - just scalar advection
- Heat diffusion handled like viscosity

Notes on discretization

- Smoke concentration and temperature may as well live in grid cells same as pressure
- But then to add buoyancy force, need to average to get values at staggered positions
- Also, to maintain conservation properties, should only advect smoke concentration and temperature (and anything else - velocity) in a divergence-free velocity field
  - If you want to do all the advection together, do it before adding buoyancy force
  - I.e. advect; buoyancy; pressure solve; repeat

Water
**Water - Free Surface Flow**

- Chief difference: instead of smoke density and temperature, need to track a free surface
- If we know which grid cells are fluid and which aren’t, we can apply p=0 boundary condition at the right grid cell faces
  - First order accurate…
- Main problem: tracking the surface effectively

**Interface Velocity**

- Fluid interface moves with the velocity of the fluid at the interface
  - Technically only need the normal component of that motion…
- To help out algorithms, usually want to extrapolate velocity field out beyond free surface

**Marker Particle Issues**

- From the original MAC paper (Harlow + Welch ‘65)
- Start with several particles per grid cell
- After every step (updated velocity) move particles in the velocity field
  - $\frac{dx}{dt}=u(x)$
  - Probably advisable to use at least RK2
- At start of next step, identify grid cells containing at least one particle: this is where the fluid is

**Issues**

- Very simple to implement, fairly robust
- Hard to determine a smooth surface for rendering (or surface tension!)
  - Blobbies look bumpy, stair step grid version is worse
  - But with enough post-smoothing, ok for anything other than really smooth flow
Surface Tracking

- Actually build a mesh of the surface
- Move it with the velocity field
- Rendering is trivial
- Surface tension - well studied digital geometry problem
- But: fluid flow distorts interface, needs adaptivity
- Worse: topological changes need “mesh surgery”
  - Break a droplet off, merge a droplet in...
  - Very challenging in 3D

Volume of Fluid (VOF)

- Work in a purely Eulerian setting - maintain another variable “volume fraction”
  \[ \frac{\partial f}{\partial t} + \nabla \cdot (fu) = 0 \]
- Update conservatively (no semi-Lagrangian) so discretely guarantee sum of fractions stays constant (in discretely divergence free velocity field)

VOF Issues

- Difficult to get second order accuracy -- smeared out a discontinuous variable over a few grid cells
  - May need to implement variable density
- Volume fraction continues to smear out (numerical diffusion)
  - Need high-resolution conservation law methods
  - Need to resharpen interface periodically
- Surface reconstruction not so easy for rendering or surface tension

Level Set

- Maintain signed distance field for fluid-air interface
  \[ \frac{\partial \phi}{\partial t} + u \cdot \nabla \phi = 0 \]
- Gives smooth surface for rendering, curvature estimation for surface tension is trivial
- High order notion of where surface is
Level Set Issues

- Numerical smearing even with high-resolution methods
  - Interface smoothes out, small features vanish

Level Set Distortion

- Assuming even no numerical diffusion problems in level set advection (e.g. well-resolved on grid), level sets still have problems
- Initially equal to signed distance
- After non-rigid motion, stop being signed distance
  - E.g. points near interface will end up deep underwater, and vice versa

Fixing Distortion

- Remember it’s only zero isocontour we care about - free to change values away from interface
- Can reinitialize to signed distance (“redistance”)
  - Without moving interface, change values to be the signed distance to the interface
- Fast Marching Method
- Fast Sweeping Method

Problems

- Reinitialization will unfortunately slightly move the interface (less than a grid cell)
- Errors look like, as usual, extra diffusion or smoothing
  - In addition to the errors we’re making in advection…
**Velocity extrapolation**

- We can exploit level set to extrapolate velocity field outside water
  - Not a big deal for pressure solve - can directly handle extrapolation there
  - But a big deal for advection - with semi-Lagrangian method might be skipping over, say, 5 grid cells
  - So might want velocity 5 grid cells outside of water
- Simply take the velocity at an exterior grid point to be interpolated velocity at closest point on interface
  - Alternatively, propagate outward to solve $\nabla u \cdot \nabla \phi = 0$
  - similar to reinitialization methods

**Particle-Level Set**

- Marker particle (MAC) method great for rough surfaces
- But if we want surface tension (which is strongest for rough flows!) or smooth water surfaces, we need a better technique
- Hybrid method: particle-level set
  - [Fedkiw and Foster], [Enright et al.]
  - Level set gives great smooth surface - excellent for getting mean curvature
  - Particles correct for level set mass (non-)conservation through excessive numerical diffusion

**Level set advancement**

- Put marker particles with values of $\phi$ attached in a band near the surface
  - We’re also storing $\phi$ on the grid, so we don’t need particles deep in the water
  - For better results, also put particles with $\phi>0$ (“air” particles) on the other side
- After doing a step on the grid and moving $\phi$, also move particles with (extrapolated) velocity field
- Then correct the grid $\phi$ with the particle $\phi$
- Then adjust the particle $\phi$ from the grid $\phi$

**Level set correction**

- Look for escaped particles
  - Any particle on the wrong side (sign differs) by more than the particle radius $\phi_0$
    - Rebuild $\phi<0$ and $\phi>0$ values from escaped particles (taking min/max’s of local spheres)
    - Merge rebuilt $\phi<0$ and $\phi>0$ by taking minimum-magnitude values
    - Reinitialize new grid $\phi$
    - Correct again
    - Adjust particle $\phi$ values within limits (never flip sign)
Fire

- [Nguyen, Fedkiw, Jensen '02]
- Gaseous fuel/air mix (from a burner, or a hot piece of wood, or ...) heats up
- When it reaches ignition temperature, starts to burn
  - “blue core” - see the actual flame front due to emission lines of excited hydrocarbons
- Gets really hot while burning - glows orange from blackbody radiation of smoke/soot
- Cools due to radiation, mixing
  - Left with regular smoke

Defining the flow

- Inside and outside blue core, regular incompressible flow with buoyancy
- But an interesting boundary condition at the flame front
  - Gaseous fuel and air chemically reacts to produce a different gas with a different density
  - Mass is conserved, so volume has to change
  - Gas instantly expands at the flame front
- And the flame front is moving too
  - At the speed of the flow plus the reaction speed

Interface speed

- Interface = flame front = blue core surface
- $D = V_f - S$ is the speed of the flame front
  - It moves with the fuel flow, and on top of that, moves according to reaction speed $S$
  - $S$ is fixed for a given fuel mix
- We can track the flame front with a level set $\phi$
  - Level set moves by
    $\phi_t + D|\nabla\phi| = 0$
    $\phi_t + u_{LS} \cdot \nabla \phi = 0$
- Here $u_{LS}$ is $u_f - S_n$
Numerical method

- For water we had to work hard to move interface accurately
- Here it’s ok just to use semi-Lagrangian method (with reinitialization)
- Why?
  - We’re not conserving volume of blue core - if reaction is a little too fast or slow, that’s fine
  - Numerical error looks like mean curvature
  - Real physics actually says reaction speed varies with mean curvature!

Conservation of mass

- Mass per unit area entering flame front is $\rho_f(V_f - D)$ where
  - $V_f = u_f \cdot n$ is the normal component of fuel velocity
  - $D$ is the (normal) speed of the interface
- Mass per unit area leaving flame front is $\rho_h(V_h - D)$ where
  - $V_h = u_h \cdot n$ is the normal component of hot gaseous products velocity
- Equating the two gives:
  $$\rho_f(V_f - D) = \rho_h(V_h - D)$$

Velocity jump

- Plugging interface speed $D$ into conservation of mass at the flame front gives:
  $$\rho_f S = \rho_h(V_h - V_f + S)$$
  $$\rho_h V_h = \rho_h V_f + \rho_f S - \rho_h S$$
  $$V_h = V_f + \left(\frac{\rho_f}{\rho_h} - 1\right)S$$

Ghost velocities

- This is a “jump condition”: how the normal component of velocity jumps when you go over the flame interface
- This lets us define a “ghost” velocity field that is continuous
  - When we want to get a reasonable value of $u_h$ for semi-Lagrangian advection of hot gaseous products on the fuel side of the interface, or vice versa (and also for moving interface)
  - When we compute divergence of velocity field
- Simply take the velocity field, add/subtract $(\rho_f/\rho_h - 1)S$
Conservation of momentum

- Momentum is also conserved at the interface
- Fuel momentum per unit area “entering” the interface is 
  \[ \rho_f V_f (V_f - D) + p_f \]
- Hot gaseous product momentum per unit area “leaving” the interface is 
  \[ \rho_h V_h (V_h - D) + p_h \]
- Equating the two gives 
  \[ \rho_f V_f (V_f - D) + p_f = \rho_h V_h (V_h - D) + p_h \]

Simplifying

- Make the equation look nicer by taking conservation of mass: 
  \[ \rho_f (V_f - D) = \rho_h (V_h - D) \]
  multiplying both sides by -D: 
  \[ \rho_f (-D)(V_f - D) = \rho_h (-D)(V_h - D) \]
  and adding to previous slide’s equation: 
  \[ \rho_f (V_f - D)^2 + p_f = \rho_h (V_h - D)^2 + p_h \]

Pressure jump

- This gives us jump in pressure from one side of the interface to the other
- By adding/subtracting the jump, we can get a reasonable continuous extension of pressure from one side to the other
  - For taking the gradient of p to make the flow incompressible after advection
- Note when we solve the Poisson equation density is NOT constant, and we have to incorporate jump in p (known) just like we use it in the pressure gradient

Temperature

- We don’t want to get into complex (!) chemistry of combustion
- Instead just specify a time curve for the temperature
  - Temperature known at flame front \( T_{\text{ignition}} \)
  - Temperature of a chunk of hot gaseous product rises at a given rate to \( T_{\max} \) after it’s created
  - Then cools due to radiation
Temperature cont’d

- For small flames (e.g. candles) can model initial temperature rise by tracking time since reaction: \( Y_t + u \cdot \nabla Y = 1 \) and making \( T \) a function of \( Y \)
- For large flames ignore rise, just start flame at \( T_{\text{max}} \) (since transition region is very thin, close to blue core)
- Radiative cooling afterwards:

\[
T_t + u \cdot \nabla T = -c_f \left( \frac{T - T_{\text{air}}}{T_{\text{max}} - T_{\text{air}}} \right)^t
\]

Smoke concentration

- Can do the same as for temperature: initially make it a function of time \( Y \) since reaction (rising from zero)
  - And ignore this regime for large flames
  - Then just advect without change, like before
- Note: both temperature and smoke concentration play back into velocity equation (buoyancy force)

Note on fuel

- We assumed fuel mix is magically being injected into scene
  - Just fine for e.g. gas burners
  - Reasonable for slow-burning stuff (like thick wood)
- What about fast-burning material?
  - Can specify another reaction speed \( S_{\text{fuel}} \) for how fast solid/liquid fuel turned into flammable gas (dependent on temperature)
  - Track level set of solid/liquid fuel just like we did the blue core