Notes

- How are projects going?

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
- $\rho=z^Tr$, check if already solved
- Loop:
  - t=As
  - $\alpha= \rho/(s^Tt)$
  - $x+= \alpha s$, $r-= \alpha t$, check for convergence
  - z=Mr
  - $\rho_{new}=z^Tr$
  - $\beta= \rho_{new} /\rho$
  - $s=z+ \beta s$
  - $\rho=\rho_{new}$

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, $L^Tp=x$
  - 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:
  - $LL^T=F+F^T+E^2+FE^{-2}F^T$
- 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

$$\left(LL^T\right)_{ijk,ijk} = E^2_{ijk} + A^2_{ijk,i-1\,jk}E^2_{i-1\,jk} + A^2_{ijk,ij-1k}E^2_{ij-1k} + A^2_{ijk,ijk-1}E^2_{ijk-1}$$

- Want this to match A's diagonal
  Then solving for next E_{ijk} in terms of previously determined ones:

$$E_{ijk} = \sqrt{A_{ijk,ijk} - A^2_{ijk,i-1\,jk}E^2_{i-1\,jk} - A^2_{ijk,ij-1k}E^2_{ij-1k} - A^2_{ijk,ijk-1}E^2_{ijk-1}}$$

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:

$$\left(I - \Delta t \nu \nabla^2\right)u^{(3)} = u^{(2)}$$

- 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 Δt]

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 ρgy to pressure (and just solve for the difference) thus cancelling out g term in equation
- υ All that's left is buoyancy -- variation in vertical force due to density variation
- υ Density varies because of temperature change and because of smoke concentration
- Assume linear relationship (small variations)

$$f_{bouy} = \left(-\alpha s + \beta T\right)$$

  - T=0 is ambient temperature; α, β depend on g etc.

Smoke equations

- So putting it all together…

$$u_t + u \cdot \nabla u + \nabla p = \left(-\alpha s + \beta T\right)(0,1,0)$$

$$\nabla \cdot u = 0$$

$$T_t + u \cdot \nabla T = k\nabla^2 T$$

$$s_t + u \cdot \nabla s = 0$$

- 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
  - 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
- Long-term behaviour may include settling: errors in interpolated velocity field, errors in particle motion, mean we don't quite preserve volume
- 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)

- Address the first issue: volume preservation
- 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