grav_project
Computational Science II

Denzel, Philipp and Vossoughi, Sara

Email:
phdenzel@hispeed.ch; sara.vossoughi@gmail.com

Main assignment

- test of a new algorithm
- gravitational force/acceleration

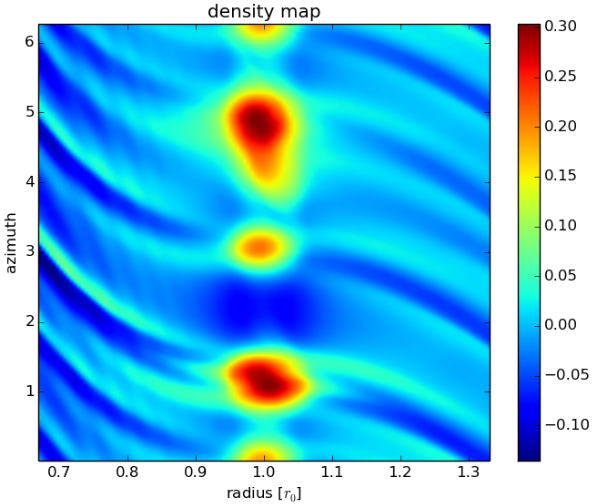
$$\overrightarrow{a_G}(r,\theta) = \int_{r'}\int_{\theta'} \frac{-\sigma(r',\theta')r'dr'd\theta'}{(r^2 + r'^2 - 2rr'\cos(\theta-\theta'))^{\frac{3}{2}}} \begin{pmatrix} r - r'\cos(\theta-\theta') \\ r'\sin(\theta-\theta') \end{pmatrix}$$

- things to be learned:
  - Fortran
  - General programming experience
  - Code optimization
  - Balancing speed vs. accuracy
- issues:
  - Time
  - Boundary conditions
  - Numerical stability

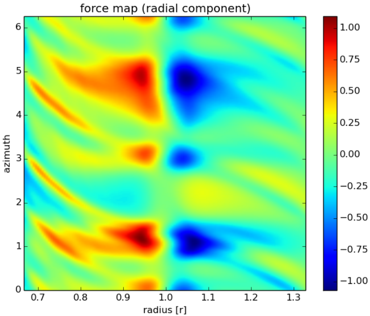
First Program

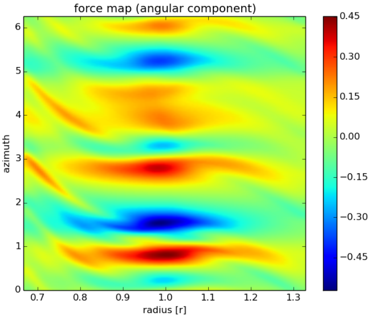
- most accurate calculation on level 0 as a reference
  - Open files
  - Read data into arrays
  - Loop through all grid points (i,j) (force)
  - Nested loop through all grid centres (i',j') (density)
  - Calculate formula inside inner loop
  - Write force into new file
- other versions (higher levels) have similar structure

Density map


density map
azimuth
radius [r_0]
6
5
4
3
2
1
0.7
0.8
0.9
1.0
1.1
1.2
1.3
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10

Level 0


force map (radial component)
azimuth
radius [r]
0.7
0.8
0.9
1.0
1.1
1.2
1.3
0
1
2
3
4
5
6
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00


force map (angular component)
azimuth
radius [r]
0.7
0.8
0.9
1.0
1.1
1.2
1.3
0
1
2
3
4
5
6
0.45
0.30
0.15
0.00
−0.15
−0.30
−0.45

Optimization Strategies

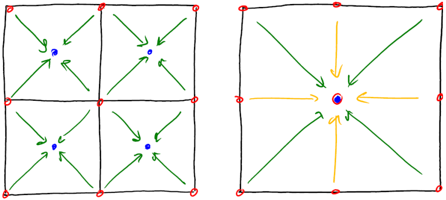
- Fortran array are column-major
- minimizing loop overhead:
  - Avoid subroutine calls in inner loops
  - Reduce repeat evalutions by shifting to outer loops
- Loop unrolling (if compiler optimization doesn't already do it)
- Faster arithmetic operations, e.g. a∗a instead of a∗∗2 (multiplications instead of exponentials and logarithms)
- Ideas: find faster methods, e.g. invsqrt using bit shifts and Newton iterations (additions and multiplications instead of divisions), use *cos* and *sin* identities to reduce lookup.
- Precalculations of lookup tables for grid given parameters (*cos*, *sin*, *r_prime*, *theta_prime* etc.)
- Using symmetries

$\longrightarrow$ time optimization from 255.95s to 9.87 s (v. 1)

$\longrightarrow$ time optimization from 69.0s to 11.8 s (v. 2)

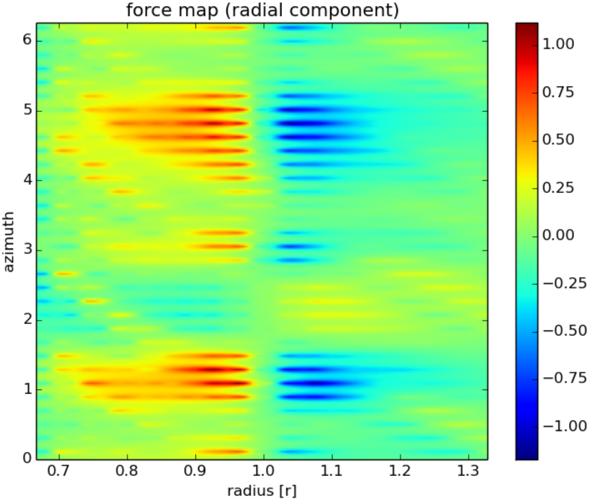
Higher levels

- Reducing operations by using fewer mass points
- 4^{LEVEL} cells into 1
- Problem: oscillations
- Cause: calculation not anymore only at corners of the grid cells
- First solution:
  - shift to corner
  - interpolate the masses
  - introduce ghost cells



point of calculation
position of mass point
distance from a corner
distance from a non-corner point

Oscillation pattern (Level 3)



Bilinear interpolation

for unit square:

$$f(x,y) \approx f(0,0)(1-x)(1-y) + f(1,0)x(1-y) + f(0,1)(1-x)y + f(1,1)xy$$

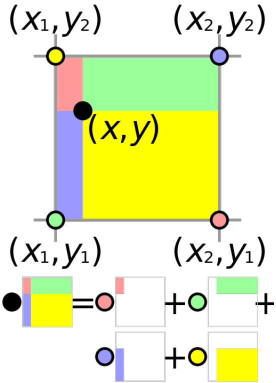
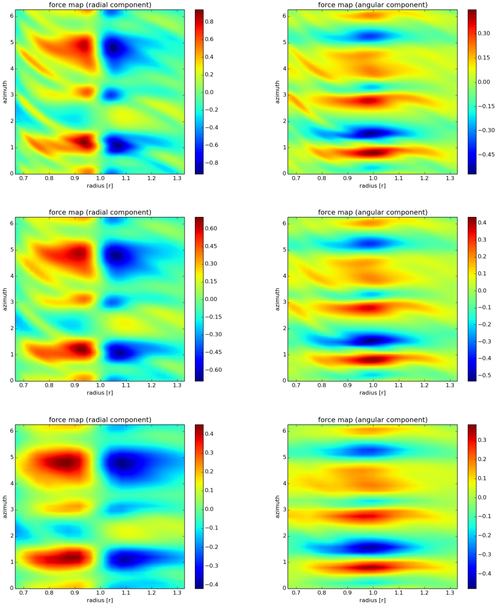


Figure : http://upload.wikimedia.org/wikipedia/commons/9/91/7 Bilinear_interpolation_visualisation.svg

Pure Levels 1, 2 and 3



Refinement with a single level

- task: calculate level 3 grid with refinement from level 2
- closest 4 by 4 cells calculated with one level below
- problem: periodic boundary conditions

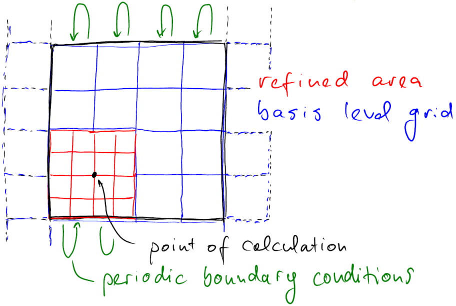


Figure : sketch by Philipp Denzel

Code

```
do i = 1, N_r0
   r_corner = r0(i)-.5*dr0(i)
   do j = 1, N_theta0
      theta_corner = theta0(j)-.5*dtheta0(j)
      call shift_param ! gives shifts: ishift(ref), jshift(ref), dr_shift(ref), dtheta_shift(ref)
      call ref_area ! gives refined area boundaries: iref_low/up, ix_low/up, periodic_low/up
      call intrpltn_coeff ! gives coefficients: c1, c2, c3, c4, c1ref, c2ref, c3ref, c4ref
      do iprime = iref_low, iref_up
         do jprime = jref_low, jref_up
         ...end do
      end do
      do iprime = 0, ix_low-1
         do jprime = 1, N_thetax
         ...end do
      end do
      do iprime = ix_up+1, N_rx
         do jprime = 1, N_thetax
         ...end do
      end do
      do iprime = ix_low, ix_up
         do jprime = 1+periodic_up, jx_low-1
         ...end do
      end do
      do iprime = ix_low, ix_up
         do jprime = jx_up+1, N_thetax-periodic_low
         ...end do
      end do
   end do
end do
```

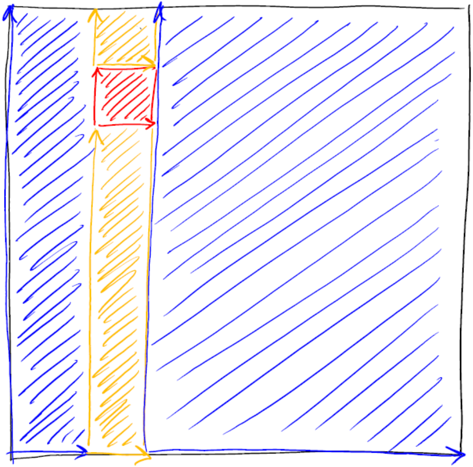
Inner loop calculation

```
vmass = ( c1 * massx(iprime, jprime) + c2 * massx(iprime+1, jprime) +&
         &c3 * massx(iprime, jprime+1)+ c4 * massx(iprime+1, jprime+1))&
         & * inv_factor2
! calculate shifted values
rprime = rx(iprime)+dr_shift
ratio = r_corner/rprime
delta_theta = theta_corner-thetax(jprime)-dtheta_shift
cosine = cos(delta_theta)
denom_point = 1.+ratio*ratio-2.*ratio*cosine
denom_point = sqrt(denom_point)*denom_point*rprime*rprime
force_point = vmass/denom_point
force_r(i,j) = force_r(i,j)+force_point*(ratio-cosine)
force_theta(i,j) = force_theta(i,j)+force_point*sin(delta_theta)
```

or for the refined area...

```
vmass = ( c1ref * massxref(iprime, jprime) + c2ref * massxref(iprime+1, jprime) +&
         &c3ref * massx(iprime, jprime+1) + c4ref * massxref(iprime+1, jprime+1))&
         & * inv_factor2ref
! calculate shifted values
rprime = rxref(iprime)+dr_shiftref
ratio = r_corner/rprime
delta_theta = theta_corner-thetaxref(jprime)-dtheta_shiftref
cosine = cos(delta_theta)
denom_point = 1.+ratio*ratio-2.*ratio*cosine
denom_point = sqrt(denom_point)*denom_point*rprime*rprime
force_point = vmass/denom_point
force_r(i,j) = force_r(i,j)+force_point*(ratio-cosine)
force_theta(i,j) = force_theta(i,j)+force_point*sin(delta_theta)
```

Schematics

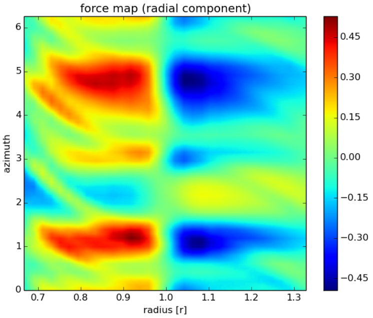


refined area

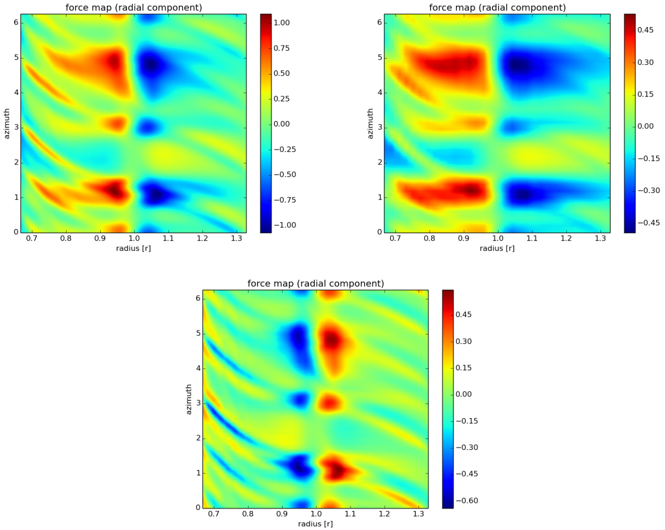
areas left and right

areas below and above

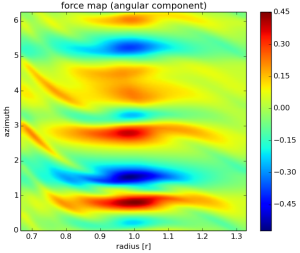
Results - Level 3 with refinement

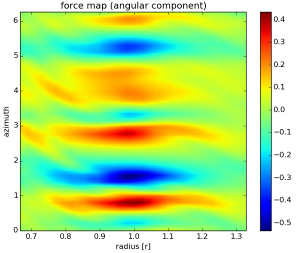


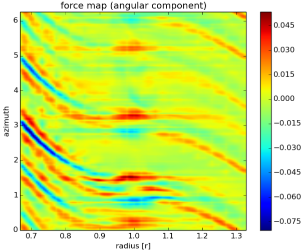
Differences to Level 0 - radial component



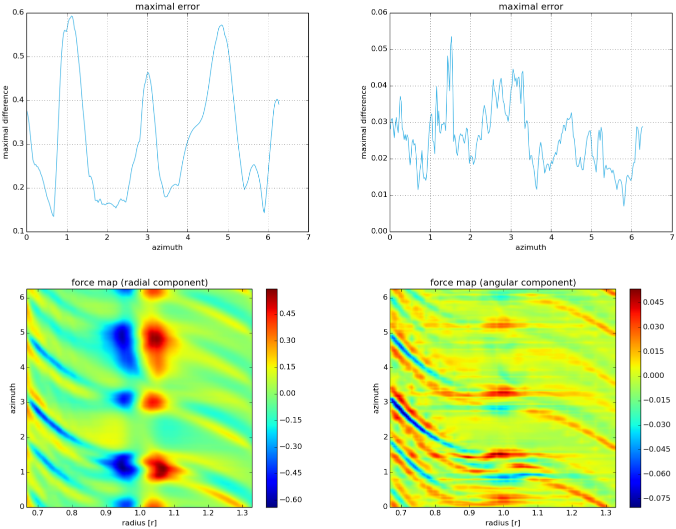
Differences to Level 0 - angular component



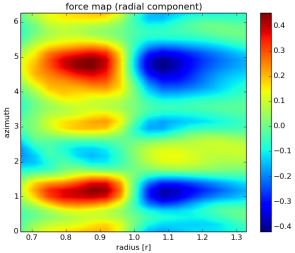


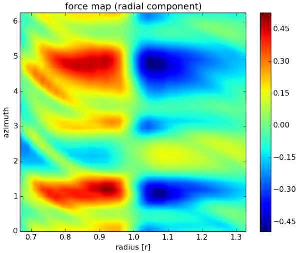


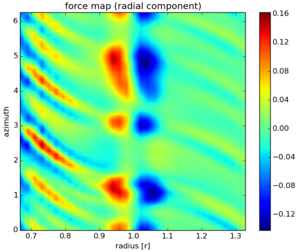
Maximal difference in a row



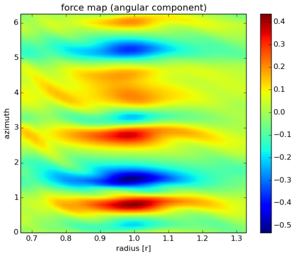
Improvement by refinement - radial component

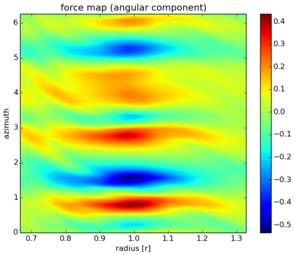


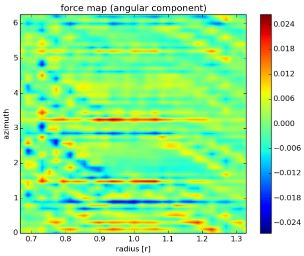




Comparison to Level 2 - angular component

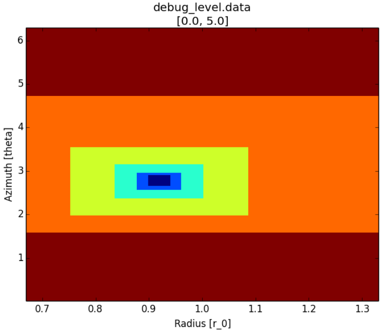


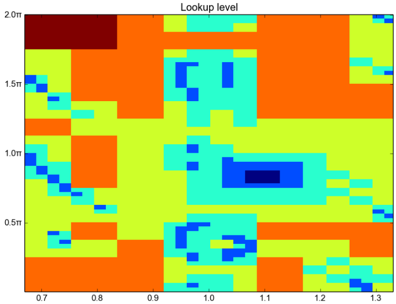




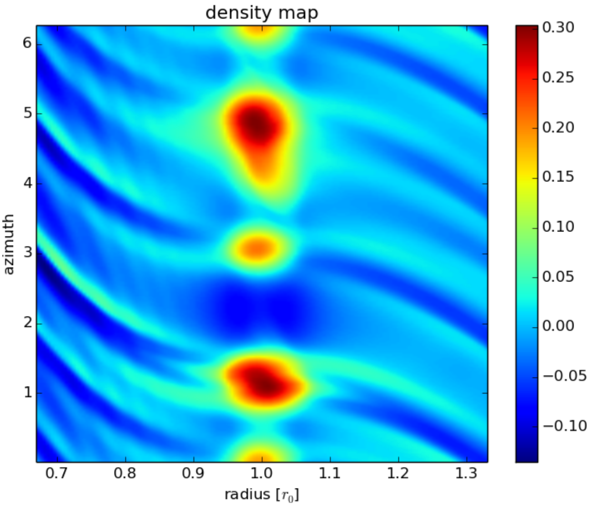
How can we improve this?

- Let's look at another strategy with an **adaptive lookup**
  - Increase level range [0,5]
  - Use distance to determine lookup level





Density map

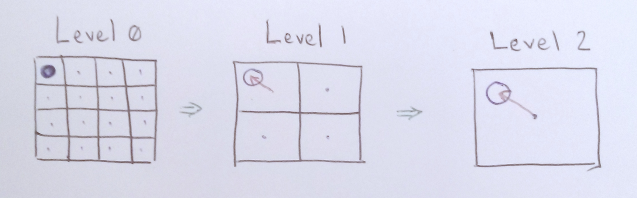

density map
6
5
4
3
2
1
azimuth
0.7
0.8
0.9
1.0
1.1
1.2
1.3
radius $[r_0]$
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10

Three Optimization Steps

1. Adaptive Lookup:
   - Increase levels with distance
2. Lookup Refinement by mass-spread:
   - For higher level cells, the distribution of mass at level 0 cells is an indicator for the accuracy of the higher level cells.
   - Measure "spread" as difference between max and min of level 0 masses of the cell.
   - User lower levels if cell "spread" is larger than an arbitrary predefined value.
   - This value has huge performance over accuracy implications.
3. Centre of Mass: Correction
   - Centre of mass usually is not at the centre of cell

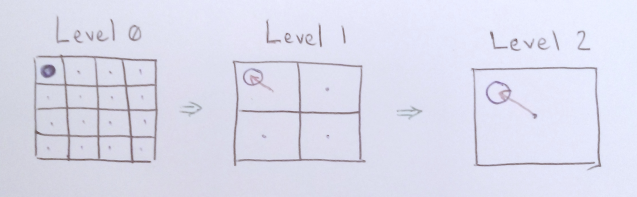
More about adaptive Code - Centre of Mass Correction

- A cell's centre of mass can vary wildly as the levels increase



More about adaptive Code - Centre of Mass Correction

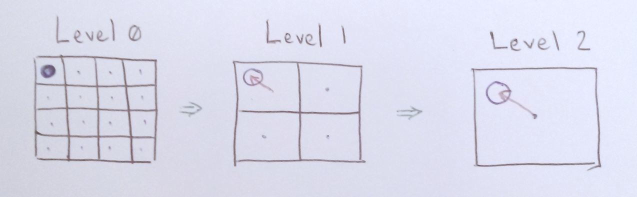
- Correcting for the centre of mass



- Improvement: error reduction from 1.8% $\rightarrow$ 1.4% in the radial component

More about adaptive Code - Centre of Mass Correction

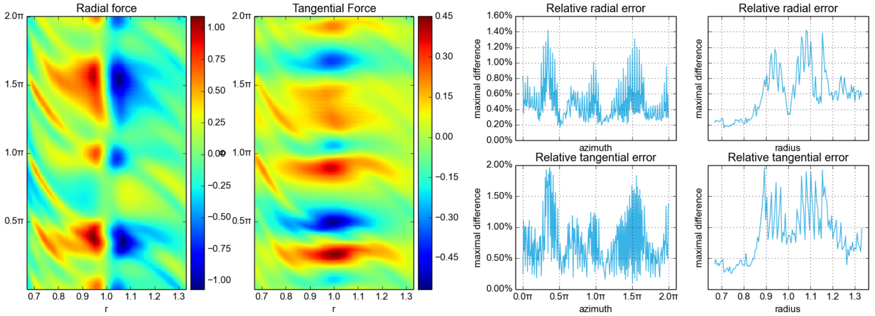
- Correcting for the centre of mass



- Improvement: error reduction from 1.8% $\rightarrow$ 1.4% in the radial component
- Further Work: Carthesian math is an approximation, should work better in polar coordinates?

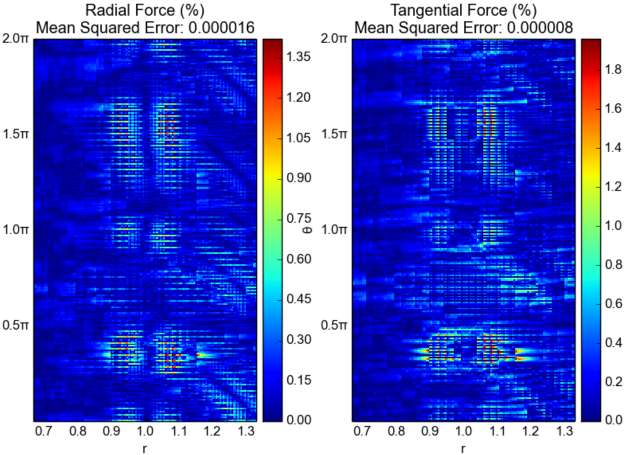
Result

- Simulation time: Level 0: 11s, Level 5-0: 4s
- Error bound: <2% at peaks
- Error patterns still exist



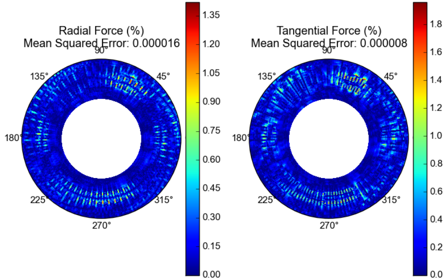
Results - Relative Diff with Level 0 vs. Adaptive

- MSE is quite small
  - Large areas have almost no error
- Problem: Some error patterns still exist (higher values?)
- Huge error spike in COM correction (lower MSE @ 0.000012)



Future Work with Adaptive Approach

- Address errors
  - Bilinear interpolation with ghost cells
  - Overlaping adaptive higher levels
- Address centre of mass correction errors
  - Address spikes (not just by increasing epsilon)
  - Polar-corrected approximation of centre of mass
- Many early errors in "array arithmetic"
  - Create an array API as a container?



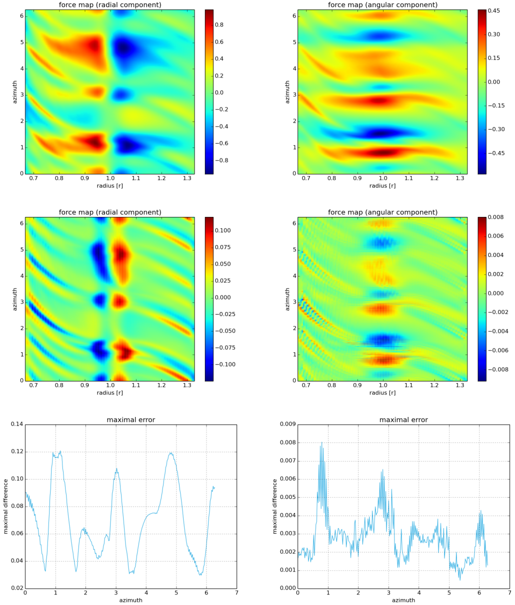
Thanks for listening!

Questions?

Simple Implementation

```
! write force components for every corner in grid
do i = 1, N_r
   r_i = r(i)-.5*dr(i) ! shift r to the corners
   do j = 1, N_theta
      theta_j = theta(j)-.5*dtheta(j) ! shift theta to the corners
      ! sum up the forces onto the point (i, j)
      do iprime = 1, N_r
         do jprime = 1, N_theta
            force_point = -sigma(iprime,jprime)*r(iprime)*dr(iprime)*dtheta(jprime)/(r_i**2+r(iprime)
                 **2-2.*r_i*r(iprime)*cos(theta_j-theta(jprime)))**(1.5)
            f_r = f_r + force_point * (r_i-r(iprime)*cos(theta_j-theta(jprime)))
            f_theta = f_theta + force_point * r(iprime)*sin(theta_j-theta(jprime))
         end do
      end do
   write(11, '(e20.10)') f_r
   write(12, '(e20.10)') f_theta
   f_r = 0.
   f_theta = 0.
   end do
end do
```

Results for Level 1 with refinement



InvSqrt

http://en.wikipedia.org/wiki/Fast_inverse_square_root

Custom InvSqrt

```
  REAL(8) FUNCTION InvSqrt (x)
   IMPLICIT NONE
   TYPE casting
      REAL(8) :: x
   END TYPE casting
   REAL(8), INTENT(in) :: x
   ! casting
   TYPE(casting), TARGET :: pointerTo
   ! Encode data as an array of integers
   INTEGER(8), DIMENSION(:), ALLOCATABLE :: enc
   INTEGER(8) :: length
   INTEGER(8) :: magic_number = 6910469410427058089
   REAL(8) :: xhalf
   xhalf = .5*x
   ! transfer to heap
   pointerTo%x = x
   ! encode a memory section from a type to other
   length = size(transfer(pointerTo, enc))
   allocate(enc(length))
   ! encoded to integer
   enc = transfer(pointerTo, enc) ! evil floating point bit level hacking
   enc(1) = magic_number - rshift(enc(1),1) ! wtf! (for int64: 0x5fe6eb50c7b537a9 = 6910469410427058089)
   ! decode
   pointerTo = transfer(enc, pointerTo)
   ! dealloc
   deallocate(enc)

   InvSqrt = pointerTo%x*(1.5 - xhalf*pointerTo%x*pointerTo%x)
END FUNCTION InvSqrt
```