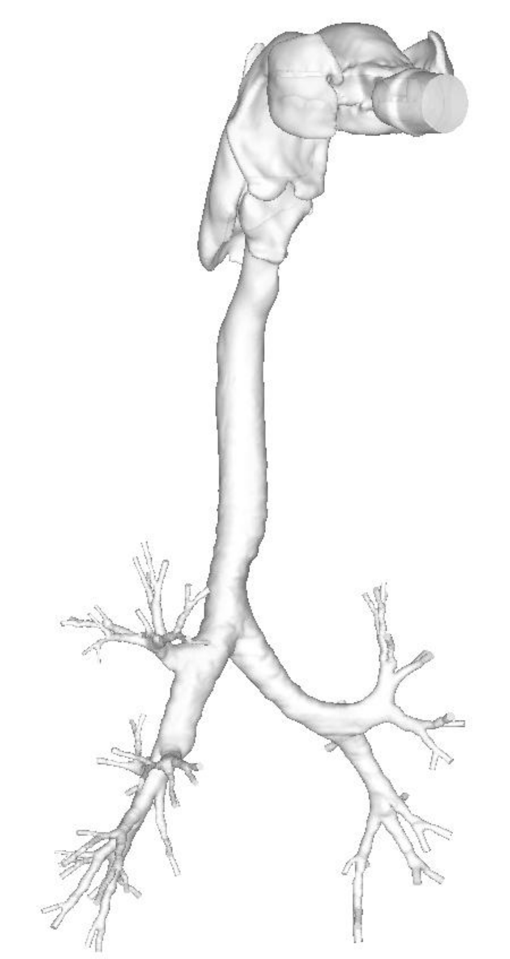
‘Regional Deposition of Particles in an Image Based Airway Model of the Human Lung’

NATHAN ELLINGWOOD
UNIVERSITY OF IOWA
ADVISOR: CHING-LONG LIN

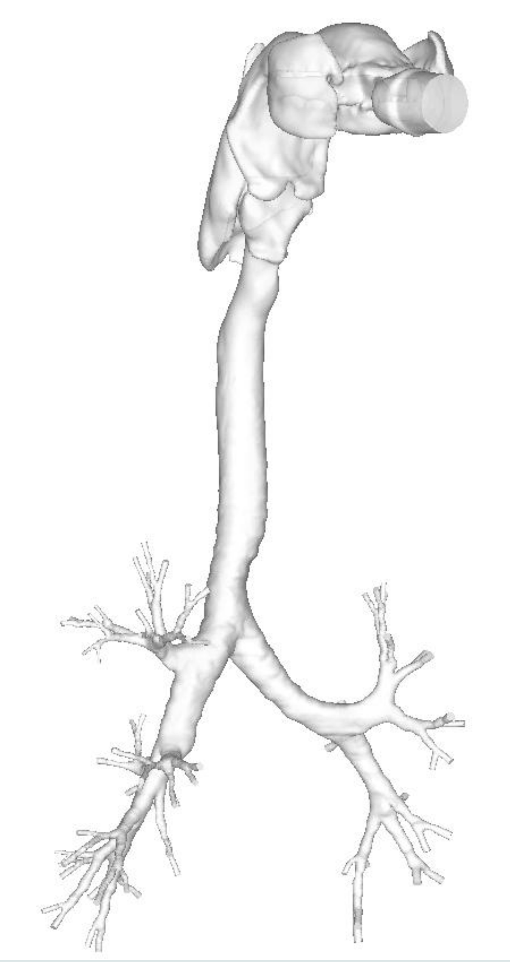
8/07/10

CFD and Lagrangian Particle Tracking



- CFD utilized to simulate pulmonary air flow in a multi-scale model of the human lungs
- CT and MRI data are used to construct realistic model of lung airway geometry
- LES simulation run with CT based lung geometry of tracheo-bronchial region, from the upper airways to the 6th generation of the central conducting airways
- Image-based velocity boundary conditions at terminal exits

CFD and Lagrangian Particle Tracking



- Original mesh geometry consists of 899,465 nodes and 4,644,447 tetrahedral elements, partitioned into 65 sub-volumes
- Refined mesh geometry consists of 1,528,932 nodes and 8,063,559 tetrahedral elements
- 2.16 s total simulation time for one full breath at 85% TLC; fluid data in time-steps of 0.048 s
- Data files are ~350MB for each 0.048s interval of fluid and mesh/node data
- Neighbor element data ~280MB

Lagrangian Particle Tracking

- Particle tracking is a post-processing step after the fluid solver obtains the CFD solution

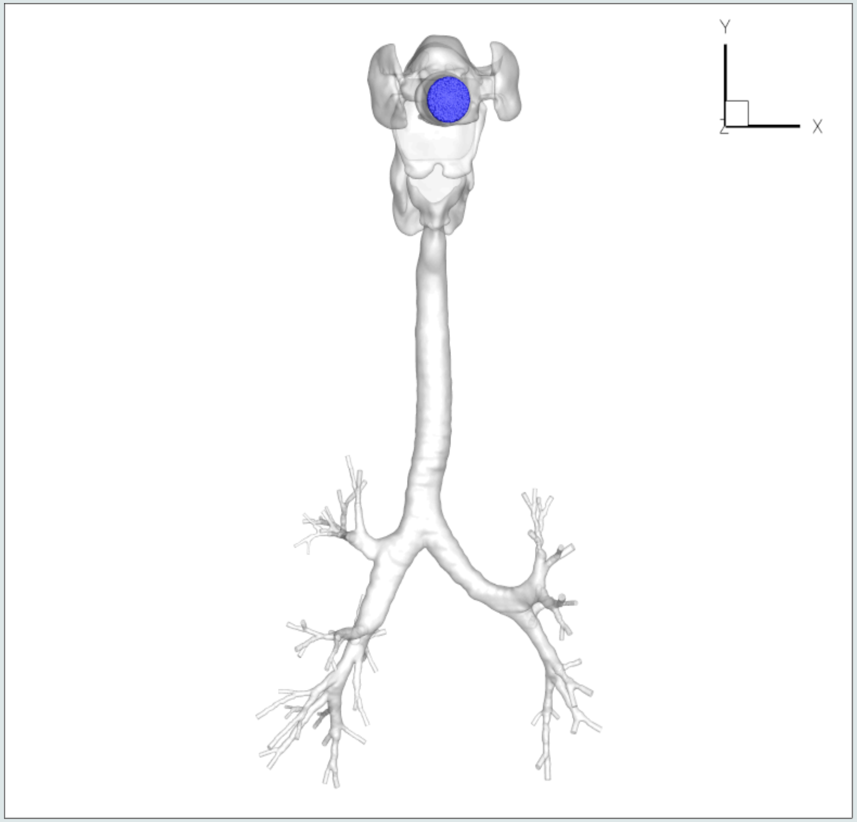
- Assumptions:
  - Brownian motion of particles not considered in airway
  - One-way coupling (no Coulomb interactions)
  - Particles are initialized as a cylindrical bolus that consists of 10,000 perfectly spherical particles located at the mouth inlet
  - Radii of 2.5, 10 and 30 um used in regional deposition simulations

Lagrangian Particle Tracking Code

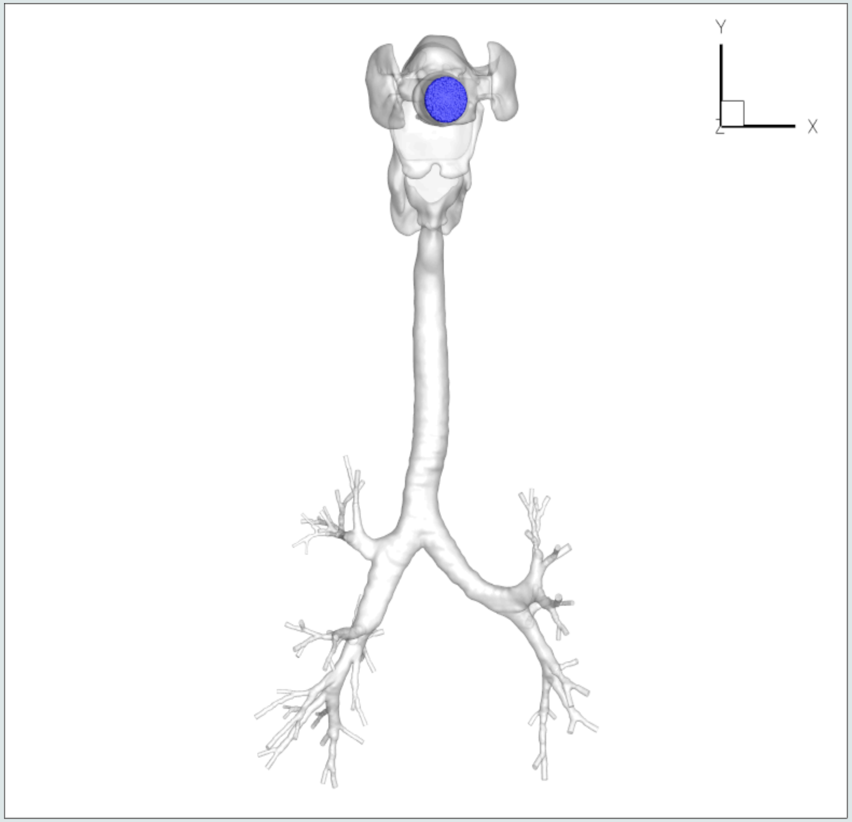
- 1. Reads in and stores a 3D dataset ~350MB, and initialize
- 2. Linear interpolation of particle velocity (see 3.)
- 3. Integrates equation of motion forward in time for same particle (from tn to tn+1, in terms of fluid data)
- 4. SALT algorithm searches for the tetrahedral element particle resides in, after integration
- 5. Repeat for each particle
- 6. Check if deposited and store locations and velocities of all particles
- As time progress load next 3D data set and repeat
    - Tecplot used for visualization after completion

Particle Tracking Visuals

2.5 um particles



30 um particles



Possible impact of work

- CFD was able to predict lobar volume changes consistent with observed physiology
- Modeling methods could help explain why a tumor might appear in the left lung or determine the cause of asthma in certain individuals
- With increased degree of accuracy can help to improve drug delivery

Goals for GPU Implementation

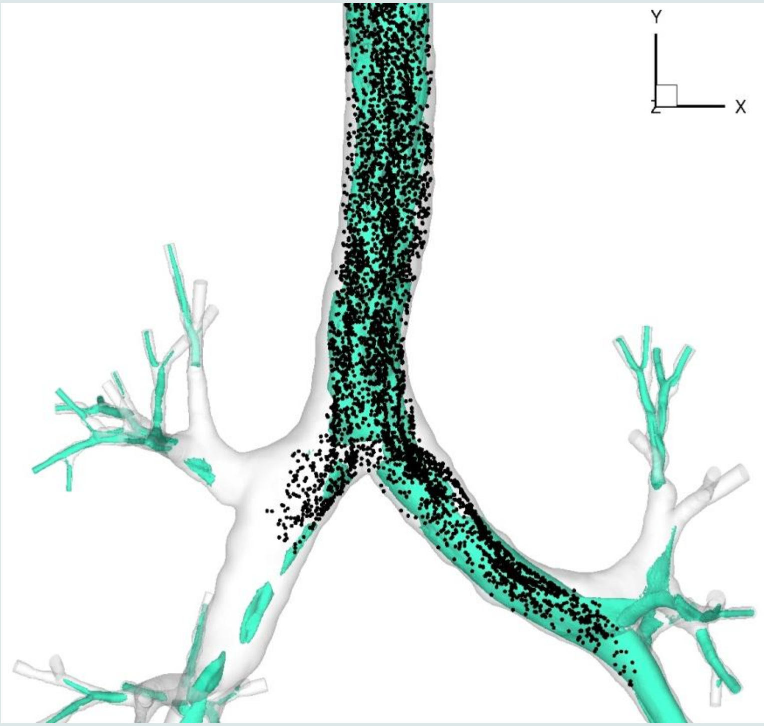
- Improve run speed from several hours to …
- Track hundreds of thousands to millions of particles
- Track particles throughout the entire airway tree, not just 6^{th} generation (3D data sets estimated to be several GB for each 0.48s time interval of fluid/mesh data)
- Real-time interaction

Lagrangian Particle Tracking Code

- Load fluid data at time-step
- (Initialize particle location/velocity; overwritten)
- Do ip=1, 10000
  - Velocity Verlet Scheme
  - SALT
  - Check if particle deposited
  - Write particle data, element particle resides in
  - End do
- Load next set of fluid data. Repeat

Key algorithms in CPU approach

- Velocity Verlet Scheme
- SALT (Search and Locate Algorithm for Linear Tetrahedra by Allievi and Bermejo)



SALT (code excerpt)

```
...delta=(xx(2)-xx(1))*(yy(3)-yy(1))*(zz(4)-zz(1))
  &     -(xx(2)-xx(1))*(zz(3)-zz(1))*(yy(4)-yy(1))
  &     -(xx(3)-xx(1))*(yy(2)-yy(1))*(zz(4)-zz(1))
  &     +(xx(3)-xx(1))*(zz(2)-zz(1))*(yy(4)-yy(1))
  &     +(xx(4)-xx(1))*(yy(2)-yy(1))*(zz(3)-zz(1))
  &     -(xx(4)-xx(1))*(zz(2)-zz(1))*(yy(3)-yy(1))

    p = 1.0/6.0
    q = 1.0/2.0
    r = 1.0/6.0

    Gfx = rx(ip)-xx(1)*(1.0-p-q-r)-xx(2)*p-xx(3)*q-xx(4)*r
    Gfy = ry(ip)-yy(1)*(1.0-p-q-r)-yy(2)*p-yy(3)*q-yy(4)*r
    Gfz = rz(ip)-zz(1)*(1.0-p-q-r)-zz(2)*p-zz(3)*q-zz(4)*r
    p1 = p + 1.d0/delta*Gfx*
  & ((yy(3)-yy(1))*(zz(4)-zz(1))-(zz(3)-zz(1))*(yy(4)-yy(1)))
  &      + 1.d0/delta*Gfy*
  & ((xx(4)-xx(1))*(zz(3)-zz(1))-(xx(3)-xx(1))*(zz(4)-zz(1)))
  &      + 1.d0/delta*Gfz*
  & ((xx(3)-xx(1))*(yy(4)-yy(1))-(xx(4)-xx(1))*(yy(3)-yy(1)))

    q1 = q + 1.d0/delta*Gfx*
  & ((zz(2)-zz(1))*(yy(4)-yy(1))-(zz(4)-zz(1))*(yy(2)-yy(1)))
  &      + 1.d0/delta*Gfy*
  & ((xx(2)-xx(1))*(zz(4)-zz(1))-(zz(2)-zz(1))*(xx(4)-xx(1)))
  &      + 1.d0/delta*Gfz*
  & ((yy(2)-yy(1))*(xx(4)-xx(1))-(xx(2)-xx(1))*(yy(4)-yy(1)))

    r1 = r + 1.d0/delta*Gfx*
  & ((yy(2)-yy(1))*(zz(3)-zz(1))-(zz(2)-zz(1))*(yy(3)-yy(1)))
  &      + 1.d0/delta*Gfy*
  & ((zz(2)-zz(1))*(xx(3)-xx(1))-(xx(2)-xx(1))*(zz(3)-zz(1)))
  &      + 1.d0/delta*Gfz*
  & ((xx(2)-xx(1))*(yy(3)-yy(1))-(xx(3)-xx(1))*(yy(2)-yy(1)))
```

```
    basismax = max(1.d0-p1-q1-r1,p1)
    basismax = max(basismax,q1)
    basismax = max(basismax,r1)

    basismin = min(1.d0-p1-q1-r1,p1)
    basismin = min(basismin,q1)
    basismin = min(basismin,r1)

    signmax = (basismax-1.d0)/abs(basismax-1.d0)
    signmin = basismin/abs(basismin)
    if(signmax.le.0.d0) then
    if(signmin.ge.0.d0) then
     ixfz(ip) = 1
     incell(ip) = ie
     call involume(ip,ie)
     call Map_vel(ip,ie)
     goto 2766
    endif
    endif

    basismin = min(1.d0-p1-q1-r1,p1)
    basismin = min(basismin,q1)
    basismin = min(basismin,r1)
    if(basismin.eq.1.d0-p1-q1-r1) lside = 1
    if(basismin.eq.p1) lside = 2
    if(basismin.eq.q1) lside = 3
    if(basismin.eq.r1) lside = 4
...
```

List of challenges expected for a GPU implementation

- The I/O problem and data storage in GPU/CPU in step 1
- The parallel version of SALT search algorithm in GPU (step 2)
- Steps 3 and 4 require parallelization as well
- Partition of the total particles into subunits running in different GPU/CPU cores (step 5)
- I/O for step 6
- Code is in Fortran, not C ☹

Acknowledgements

- Ching-Long Lin
- Merryn Tawhai
- Eric Hoffman
- Haribalan Kumar
- Jiwoong Choi
- Youbing Yin
- Andrew Lambert