

Lecture 23: Bits and Bones

David Bindel

19 Apr 2010

Logistics

- ▶ Projects! Things to remember:
 - ▶ Stage your work to get *something* by end-of-semester
 - ▶ Important thing is reasoning about performance
 - ▶ Do come talk to me!
- ▶ No office hours this week
- ▶ SCAN seminar today (1:25 pm in 315 Upson)
- ▶ Wednesday guest lecture: Charlie Van Loan

Outline

Bone basics

Bone measurement and modeling

BoneFEA software

Conclusion

Outline

Bone basics

Bone measurement and modeling

BoneFEA software

Conclusion

Why study bones?

- ▶ Osteoporosis: 44M Americans, \$17B / year
- ▶ Over 55% of over 50 have osteoporosis or low bone mass
- ▶ 350K hip fractures / year; over \$10B / year
- ▶ A quarter of hip fracture patients die within a year
- ▶ ... and we're getting older

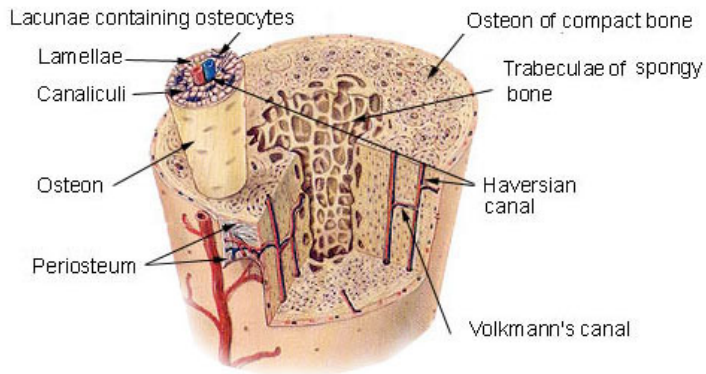
Bone basics: macrostructure



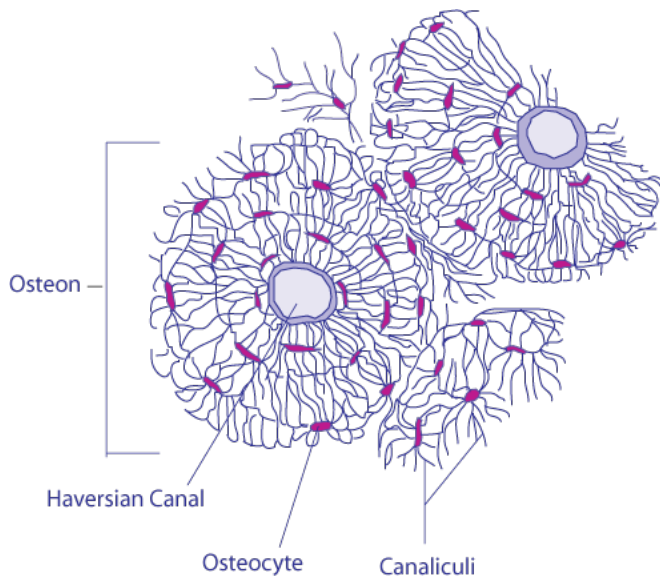
© ADAM, Inc.

Bone basics: microstructure

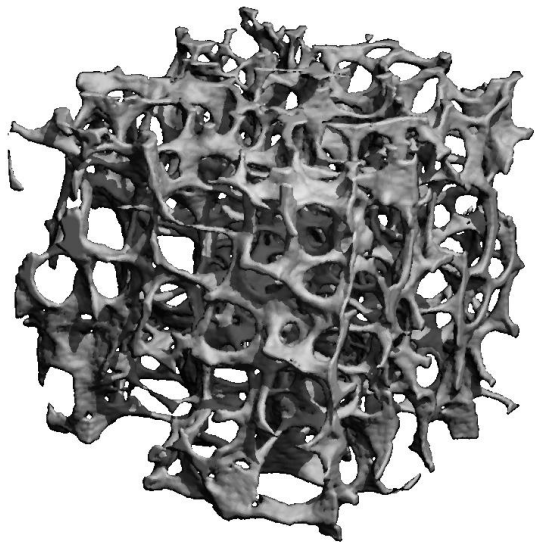
Compact Bone & Spongy (Cancellous Bone)



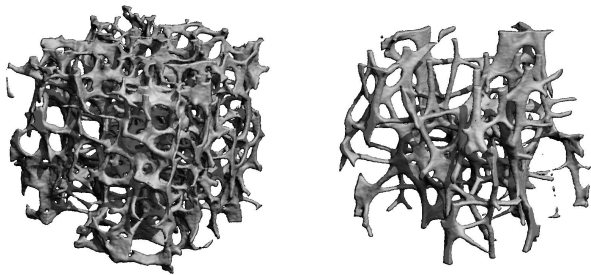
Bone basics: microstructure



Bone basics: trabecular microstructure

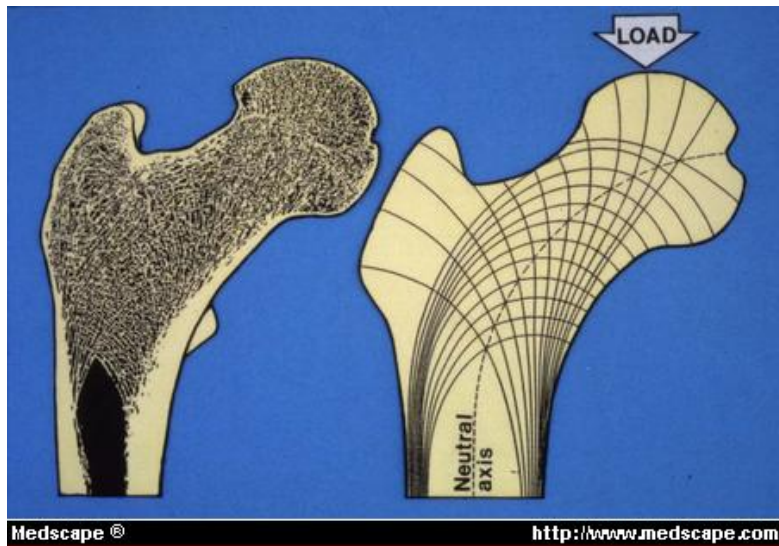


Bone basics: trabecular microstructure



(Scans from 23 and 85 year old females)

Bone basics: orientation and remodeling



Why study bones?

... because bone is a fascinating material!

- ▶ Structurally complicated across length scales
- ▶ Structure adapts to loads and changes over time
- ▶ inhomogeneous, anisotropic, asymmetric, often nonlinear

Outline

Bone basics

Bone measurement and modeling

BoneFEA software

Conclusion

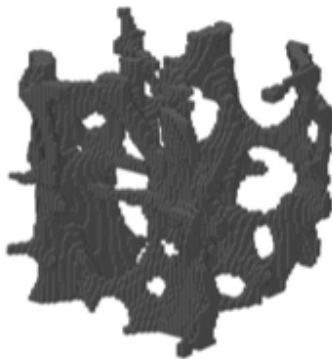
Bone measurement

- ▶ Diagnostic for osteoporosis: T-scores from DXA
- ▶ Ordinary microscopy on extracted cores
- ▶ QCT software: density profile, about 3 mm scale
- ▶ Micro-CT and micro-MRI: $O(10 \text{ micron})$

Micro-FE bone modeling



micro-CT @ 22 μm resolution
(8 mm H: 15 mm D)

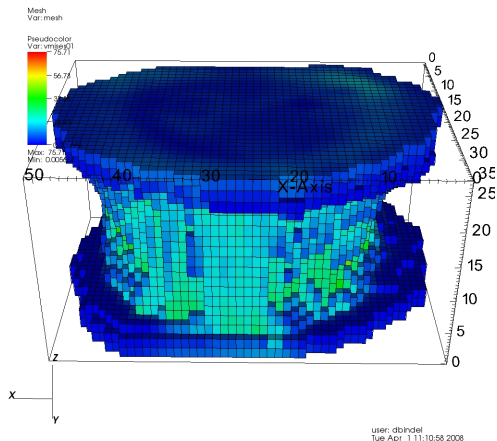


micro-FE mesh
 $2.5 \times 2.5 \times 2.5 \text{ mm}^3$
44 μm hexahedral elements

One vertebrate = 57M+ elements at 40 microns

Whole bone modeling

DB: vb.vtk



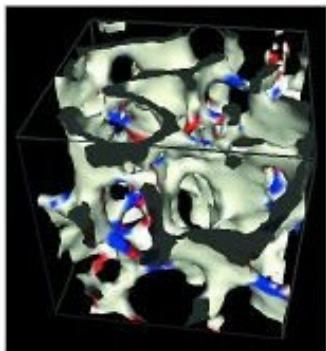
- ▶ Density only weakly predicts strength
- ▶ Wanted: Good effective constitutive relation

Difficulties

Bone is:

- ▶ Variable over time and between individuals
- ▶ Inhomogeneous and anisotropic
- ▶ Different in tension and compression

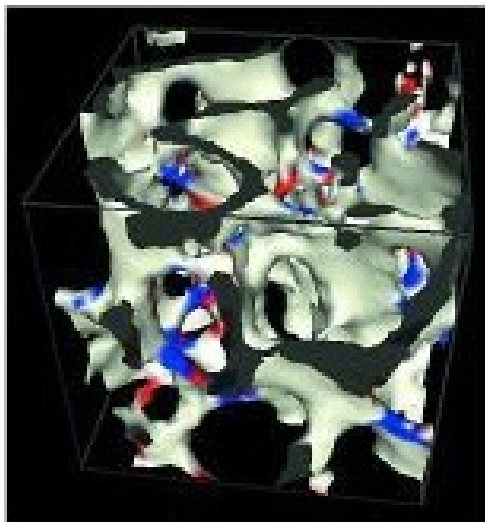
Yielding and nonlinearity



Example difficulty:

- ▶ Trabecular network has beam and plate elements
- ▶ Small macro strains yield much larger micro strains
- ▶ Small-scale geometric nonlinearity a significant effect

Yielding and nonlinearity



An approach

- ▶ Micro-CT structure scans for orientation
- ▶ Use orientation indices + density to approximate material parameters
- ▶ Proceed phenomenologically

Outline

Bone basics

Bone measurement and modeling

BoneFEA software

Conclusion

Diagnostic toolchain

- ▶ Micro-CT scan data from patient
- ▶ Inference of material properties
- ▶ Construction of coarse FE model (voxels)
- ▶ Simulation under loading
- ▶ Output of stress fields, displacements, etc.

Software strategies

Two basic routes:

- ▶ Discretize microstructure to get giant FE model
 - ▶ Prometheus (Mark Adams)
 - ▶ ParFE (Arbenz and Sala)
- ▶ Approximate microstructure with constitutive model
 - ▶ Can do with commercial FEM codes
 - ▶ Less compute time
 - ▶ Less detail required in input?
 - ▶ Hard to get the right constitutive model

A little history

BoneFEA started as a consulting gig

- ▶ Code for ON Diagnostics (Keaveny and Kopperdahl)
- ▶ Developed jointly with P. Papadopoulos
- ▶ Meant to replace ABAQUS in overall system
- ▶ Initial goal: some basic simulations in under half an hour
- ▶ Development work on and off 2006–2008
- ▶ More recent revisitings (trying to rebuild)

BoneFEA

- ▶ Standard displacement-based finite element code
- ▶ Elastic and plastic material models (including anisotropy and asymmetric yield surfaces)
- ▶ High-level: incremental load control loop, Newton-Krylov solvers with line search for nonlinear systems
- ▶ Library of (fairly simple) preconditioners; default is a two-level geometric multigrid preconditioner
- ▶ Input routines read ABAQUS decks (and native format)
- ▶ Output routines write requested mesh and element quantities
- ▶ Visualization routines write VTK files for use with VisIt

Basic principles

- ▶ This sort of programming seems hard (?)
 - ▶ How many man-hours went into ABAQUS?
 - ▶ Easy to lose sleep to an indexing error
- ▶ Want to reduce the *accidental* complexity
 - ▶ Express as much as possible at a high level
 - ▶ Use C++/Fortran (and libraries) for performance-critical stuff
 - ▶ Make trying new things out easy

Enabling technology

Three separate language-based tools:

- ▶ Matexpr for material model computations
- ▶ Lua-based system for scripting simulations and solvers
- ▶ Lua-based system for mesh and boundary conditions

In progress: solver scripting via PyTrilinos (Sandia)

Solver quandries

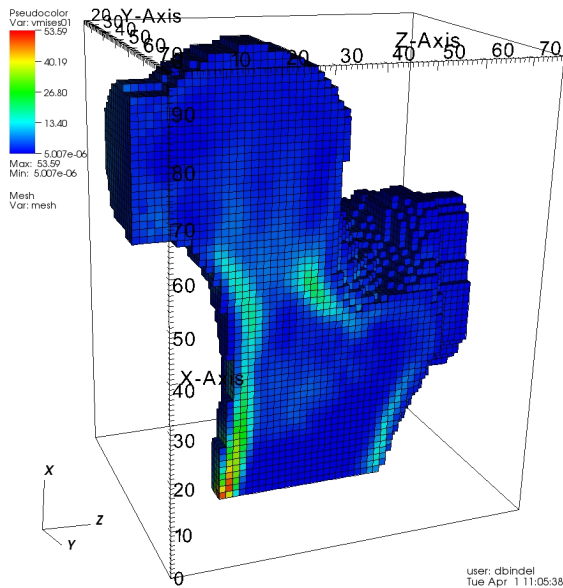
A simple simulation involves *lots* of choices:

- ▶ Load stepping strategy?
- ▶ Nonlinear solver strategy?
- ▶ Linear solver strategy?
- ▶ Preconditioner?
- ▶ Solvers in multilevel preconditioner?

Want a simple framework for playing with options.

Example analyses

DB: femur.vtk



Example analysis loop

```
mesh:rigid(mesh:numnp()-1, {z='min'},  
  function()  
    return 'uuuuuu', 0, 0, bound_disp  
  end)
```

```
pc = simple_msm_pc(mesh,20)  
mesh:set_cg{M=pc, tol=1e-6, max_iter=1000}  
for j=1,n do  
  bound_disp = 0.2*j  
  mesh:step()  
  mesh:newton{max_iter=6, Rtol=1e-4}  
end
```

Analysis innards

- ▶ `rigid` ties a specified part of the mesh to a rigid body (and applies boundary conditions to that rigid body)
- ▶ `step` swaps history, updates load, computes predictor
- ▶ `newton` does Newton iteration with line search; specify
 - ▶ Max iterations
 - ▶ Residual tolerance
 - ▶ Line search parameters (Armijo constant α)
 - ▶ What linear solver to use
 - ▶ Whether to update the preconditioner
- ▶ Also have `mnewton` (modified Newton)

Preconditioning

- ▶ Accelerate iterative solver with *preconditioner*
- ▶ Often built from simpler blocks
 - ▶ Basic iterative solver passes
 - ▶ Block solves
 - ▶ Coarse grid solves
- ▶ Want a simple way to assemble these blocks

Preconditioner specification (library code)

```
function simple_msm_pc(mesh, ncgrid, nsmooth, omega)
    local pcc = form_coarse_pc2(mesh, ncgrid)
    local pc  = {}
    local K   = mesh.K
    nsmooth = nsmooth or 1
    function pc:solve(x,b) ... end
    function pc:update() pcc:update() end
    function pc:delete() ... end
    return pc
end
```

Preconditioner specification (library code)

```
function pc:solve(x,b)
    self.r = self.r or QArray:new(x:m(),1)
    self.dx = self.dx or QArray:new(x:m(),1)

    mesh_bgs(mesh.mesh,mesh.K,x,b,nsmooth)
    K:apply(x,self.r)
    self.r:sub(b)

    pcc:solve(self.dx,self.r)
    x:sub(self.dx)
    K:apply(x,self.r)
    self.r:sub(b)

    mesh_bgs(mesh.mesh,mesh.K,self.dx,self.r,nsmooth)
    x:sub(self.dx)
end
```

Preconditioning triumphs and failures

- ▶ We do pretty well with two-level Schwarz
 - ▶ 18 steps, 15 s to solve femur model on my laptop
- ▶ ... up until plasticity starts to kick in
- ▶ Needed: a better (physics-based) preconditioner
- ▶ Usual key: physical insight into macroscopic behavior

Material modeling

BoneFEA provides general plastic element framework; specific material model provided by an object. Built-in:

- ▶ Isotropic elastic
- ▶ Orthotropic elastic
- ▶ Simple plastic
- ▶ Anisotropic elastic / isotropic plastic
- ▶ Isotropic elastic / asymmetric plastic yield surface

How do we make it simplify to code more?

Example: Plasticity modeling (no hardening)

Basic idea: push until we hit the yield surface. Push the yield surface around as needed. Can also change shape of yield surface (hardening/softening effects)

$$\begin{aligned}\dot{\epsilon}_{ij}^p &= \lambda \frac{s_{ij} - a_{ij}}{\|s_{ij} - a_{ij}\|} \\ \dot{\kappa} &= \sqrt{\frac{2}{3}} \lambda \\ \dot{\alpha}_{ij} &= \frac{2}{3} (1 - \eta) q'(k) \dot{\epsilon}_{ij}^p\end{aligned}$$

Return map algorithm – take a step, project back to yield surface

More complicated with anisotropy. I don't like writing this in C!

Partial solution: Matexpr

- ▶ Relatively straightforward in `MATLAB` – but slow
- ▶ Use `Matexpr` to translate `MATLAB`-like code to C
- ▶ Supports basic matrix expressions, symbolic differentiation, etc.
- ▶ Takes advantage of symmetry, sparsity, etc. to optimize generated code
- ▶ Does not provide control flow (that's left to C)

Matexpr in action

```
void ME::plastic_DG(double* DG, double* Cd,
                   double* n, double qp)
{
    /* <generator matexpr>
    input Cd(9,9), n(9), qp;
    inout DG(9,9);

    m = [1; 1; 1; 0; 0; 0; 0; 0; 0];
    Iv = m*m'/3.0;
    Id = eye(9) - Iv;

    CIdn = DG*(Id*n);
    con = n'*Cd*n + 2*qp/3.0;
    DG = DG - CIdn*CIdn'/con;
    */
}
```

Outline

Bone basics

Bone measurement and modeling

BoneFEA software

Conclusion

Conclusion

- ▶ Bones are interesting as well as important!
- ▶ Initial BoneFEA work done, in use by ON Diagnostics
- ▶ Possible follow-up work for diagnostic tool
- ▶ Plenty of interesting research directions