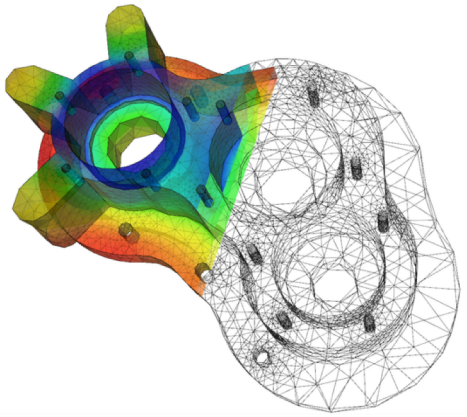
Lecture VIII: Finite Element Simulation

Motivation

- Continuous physics: rigid, deforming, attaching, detaching bodies.
  - “continuous” features
  - “infinite” resolution
- Computer simulation:
  - Discrete representation with limited memory.
  - Efficiency is key.
  - …and also consistency and stability.

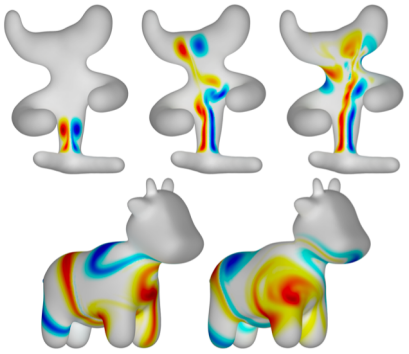


Partial Differential Equations

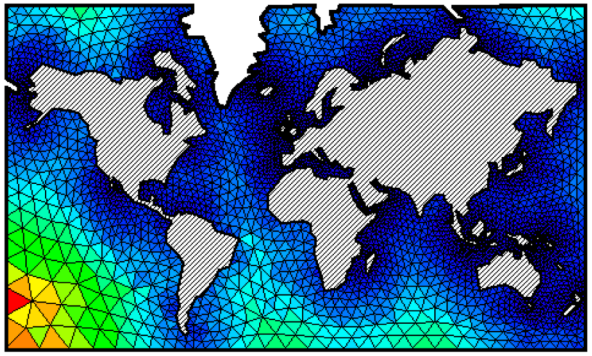
- Govern the connection between kinematics and dynamics.
- Examples:
  - $\vec{F} = m\vec{a} = m\frac{d\vec{v}}{dt} = -\vec{v}$ (drag).
  - $\vec{F} = -k\vec{x} = \frac{d\vec{x}}{dt}$ (spring; Hooke's law).
- We learned how to integrate in time.
- How to discretize in space?

Partial Differential Equations

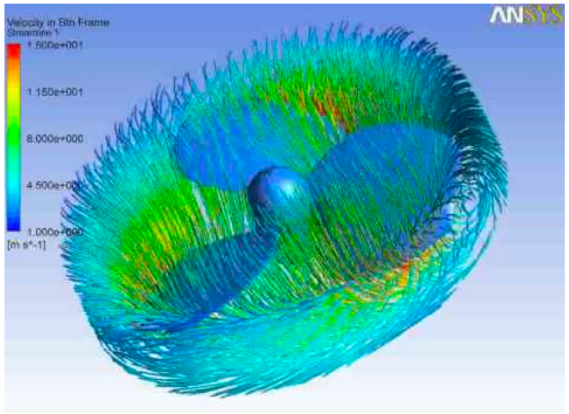
- Challenges:
  - Discretize (partial) derivatives
  - Obey conservation rules
  - Consistency
  - Good sampling
- Representation choices:
  - Where do functions\vectors\tensors “live”?



[Azencot *et al.* 2015]



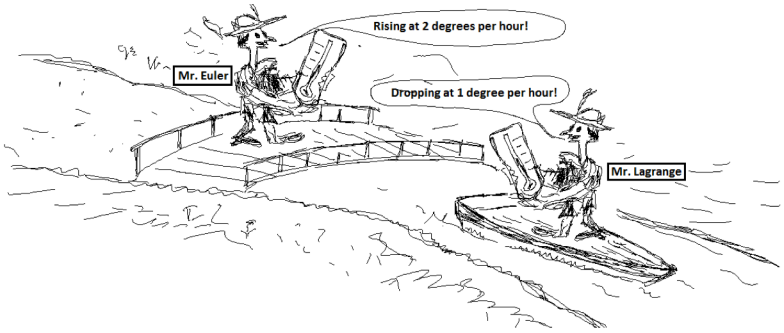
http://caewatch.com/top-5-misunderstandings-on-good-mesh/



https://www.youtube.com/watch?v=5mNn7csNGDk

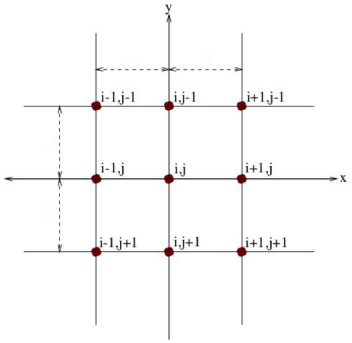
Numerical Simulation

- Lagrangian:
  - Representation: connected mesh or cloud of particles.
  - Examples: Finite methods, Mass-spring system, particle systems.
- Eulerian:
  - Representation: stationary point set or grid, where material properties change over time.
  - Boundary of object not explicitly defined.
  - Suitable for fluids.

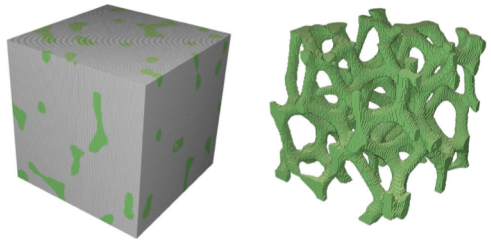


http://www.flowillustrator.com/wp-content/uploads/2015/02/MaterialDerivative.png

Finite Differences Method

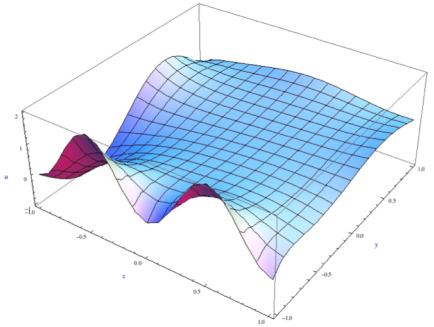


- Object sampled using regular spatial grid.
- PDE discretized using finite differences.
  - Pro: easier and more stable to implement than unstructured methods.
  - Con: difficult to approximate complex boundaries.
- Semi-implicit integration is used to move forward through time



(a) *Voxelized sample cube, consisting of voxels with material index 1 (aluminium) and 0 (void).*

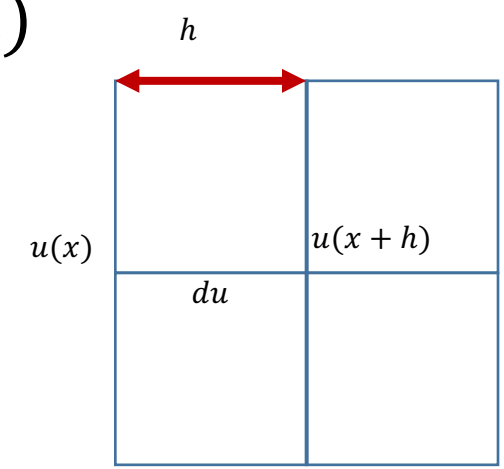
(b) *Voxels with material index 1, representing the foam geometry.*



http://www.tc.umn.edu/~dominik/sampling-of-research-activi.html

Finite Differences Method

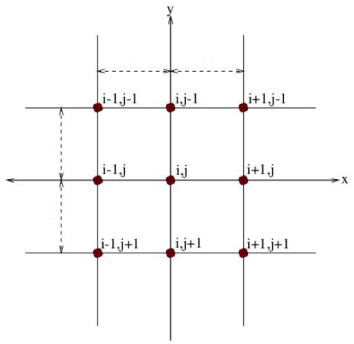
- Grid size h.
- Scalar functions: at vertices.
- Differentials: $du = u(x+h) - u(x)$
- Derivatives: $\frac{du}{dx} = \frac{u(x+h)-u(x)}{h} + O(h)$
  - ”Living” on edges



Example: Heat Equation

- $u_t = c\Delta u.$
  - In *2D*: $u_t = c(u_{xx} + u_{yy})$
- Forward discretization in time:

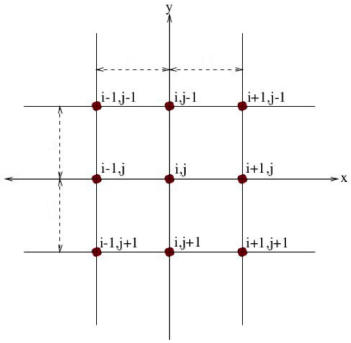
$$u_t \approx \frac{u_{i,j}^{t+\Delta t} - u_{i,j}^{t}}{\Delta t}$$



- 2nd order derivative central approximation:

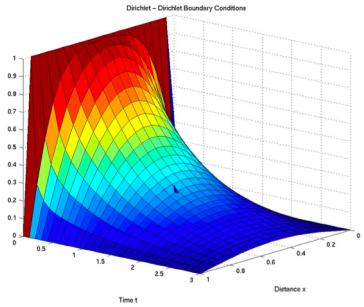
$$u_{xx} = (u_x)_x \approx \frac{\frac{u_{i+1}^t - u_i^t}{h} - \frac{u_i^t - u_{i-1}^t}{h}}{h} = \frac{u_{i+1}^t - 2u_i^t + u_{i-1}^t}{h^2}$$

Example: Heat Equation



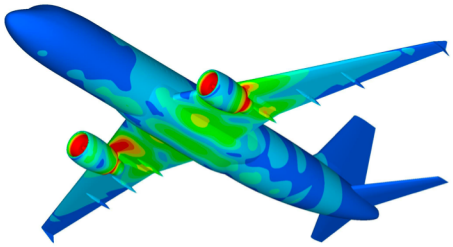
$$u_{xx} + u_{yy}$$
$$\approx \frac{u^t_{i+1,j} - 2u^t_{i,j} + u^t_{i-1,j}}{h^2} + \frac{u^t_{i,j+1} - 2u^t_{i,j} + u^t_{i,j-1}}{h^2}$$
$$= \frac{u^t_{i+1,j} + u^t_{i-1,j} + u^t_{i,j+1} + u^t_{i,j-1} - 4u^t_{i,j}}{h^2}$$

- Measuring difference of neighbor average from center!
- Dirichlet Boundary conditions: set constant values



http://www.math.oregonstate.edu/~show/images/Dd.jpg

Boundary Element Method



http://urbana.mie.uc.edu/yliu/

- Remember Stokes theorem:

$$\int_{\partial\Omega} \omega = \int_{\Omega} d\omega$$

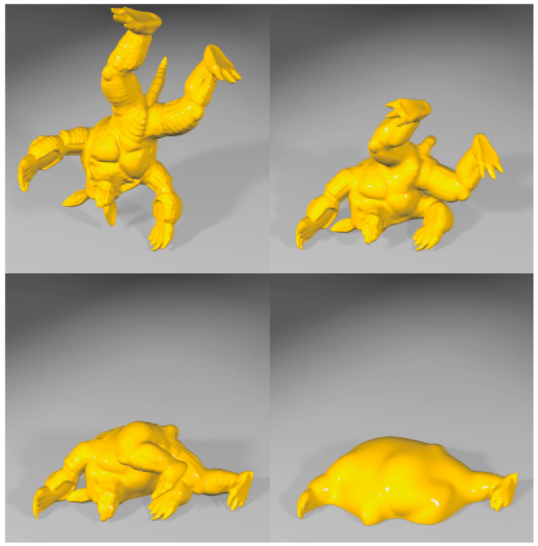
- Practical interpretation: the sum of what "happens inside" only depends on "what goes in and out".
- Discretizing PDE only on the boundary surface instead of the entire volume.
  - Only good for homogenous material.
  - Topological changes more difficult to handle.

Finite Volume Methods

- Measure flux changes in small (simplicial) elements
  - Like a BEM on a FEM.

$$\frac{\partial}{\partial t}\iiint Q dV + \iint F dA = 0$$

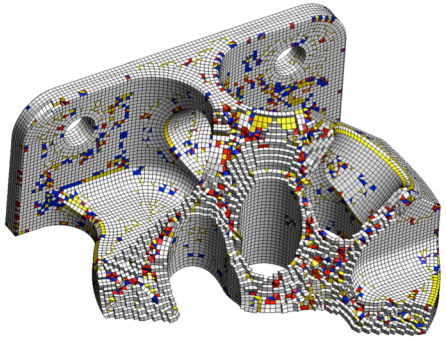
  - *Q:* variable inside the domain
  - *F*: flux on the boundary.
- Conserves volume by definition!
  - Good for CFD.

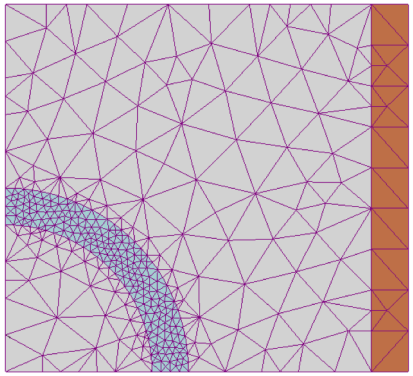


"A simple finite volume method for adaptive viscous liquids" by [Batti *et al.* 2011]

Finite Element Method (FEM)

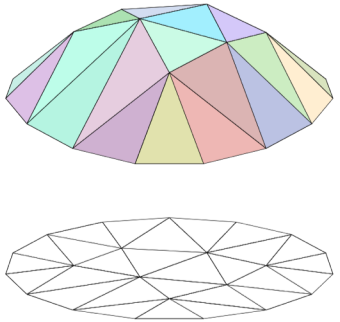
- Tessellating the volume into a large finite number of disjoint elements (3D volumetric/surface mesh).
- Usually: simplices. Sometimes quads/hexes.
- Scalar functions: at vertices.
- Differentials on edges e_{ij}: $du = u_j - u_i$.
- Derivatives?



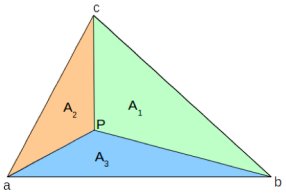


https://afinemesh.files.wordpress.com/2015/05/frontal-hex-mesh.png

Lagrange FEM Basis

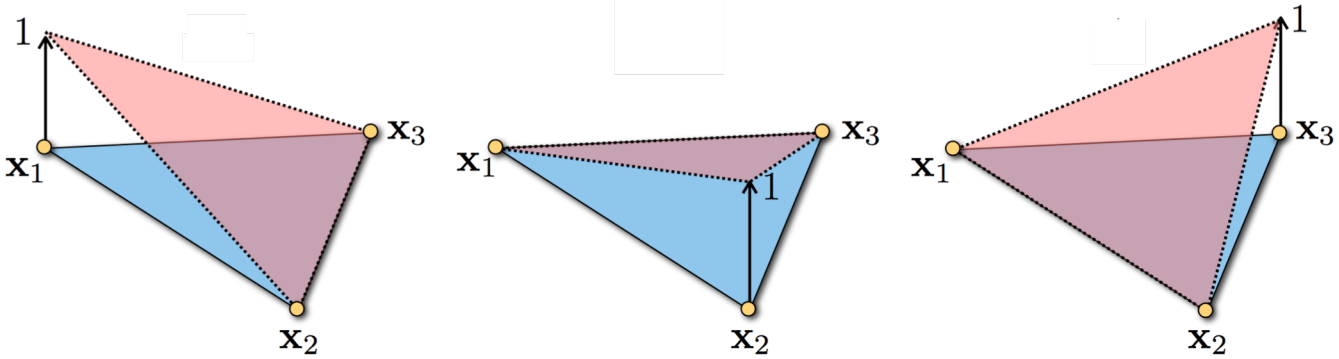


- Function values defined on vertices, but
  - Assumed to interpolate linearly on elements:



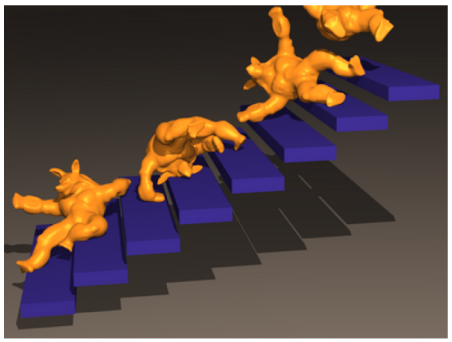
$$u(p) = \sum_{i=0}^{3} B_i(p) u_i$$

  - $B_i(p)$: Barycentric coordinates of *p* in element.



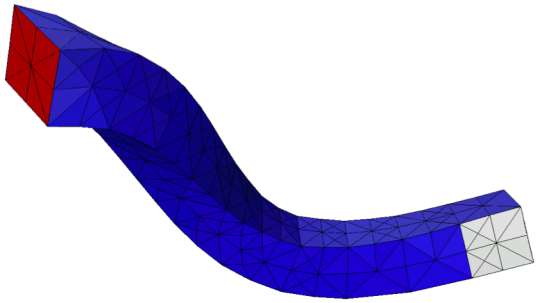
Motivation

- Discretize deformations, vectors, and tensors
- Derivative and integrals => linear operators (matrices)
  - Solving PDEs ⇔ solving linear equations
  - Either directly, or by iterations.



Irving *et al.* "Volume Conserving Finite Element Simulations of Deformable Models"

https://www.youtube.com/watch?v=Rbq2CdUIvw4



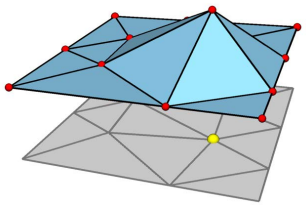
http://fetk.org/codes/mc/images/bar3ndef_wbg.gif

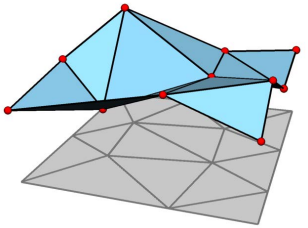
Lagrange FEM Basis

- Equivalent definition: function is defined by per-vertex u_v scalar values
- Entire space is spanned by Lagrange basis functions

$$u(p) = \sum_{v \in V} u_v \, \varphi_v(p)$$

- $\varphi_v(p) = 1$ on v, and 0 on other vertices.
  - Piecewise linear ("hat function").



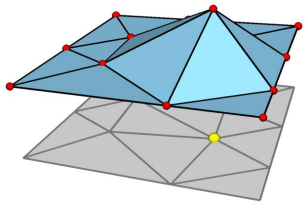


$$u(p) = \sum_{v \in V} u_i \, \varphi_v(p)$$

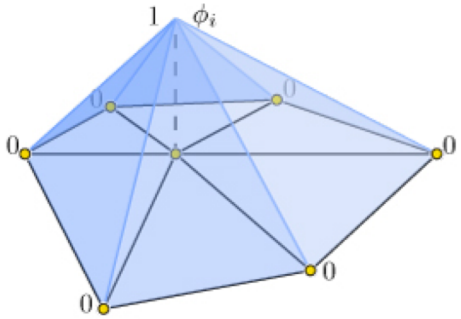
Lagrange FEM Basis

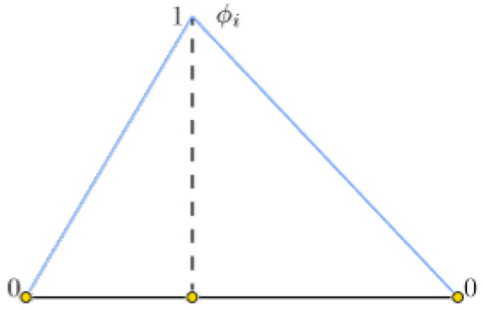
- If scalar functions are piecewise linear on vertices:
  - Gradients ∇u are piecewise constant on faces!

$$\nabla u(p) = \sum_{v \in V} u_i \, \nabla \varphi_v(p)$$



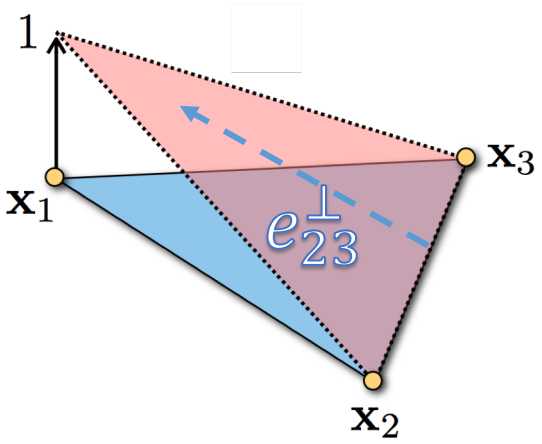
- Example: position function on vertices, deformation (velocity) on faces.





Gradients

- We want to compute $\nabla\varphi_1$ (w.l.o.g.) in face 123.
- Insight: in the direction of $e_{23}^{\perp} = \hat{n}\times e_{23}$.
- Result: $\nabla\varphi_1 = \frac{e_{23}^{\perp}}{2A_{123}}$
  - A_{123}: area of triangle 123.
- For a function u_1, u_2, u_3:



$$\nabla u_{123} = G u_{123} = \frac{1}{2A_{123}}\left(e_{23}^{\perp}u_1 + e_{31}^{\perp}u_2 + e_{12}^{\perp}u_3\right)$$

derived from geometry

Gradients

- In matrix form ($\vec{e}_{23}^{\perp}$ etc. are column vectors 1x3):
- $G_{123}u = \frac{1}{2A_{123}}(\vec{e}_{23}^{\perp} \quad \vec{e}_{31}^{\perp} \quad \vec{e}_{12}^{\perp}) \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix}$

  $= \begin{pmatrix} \partial u/\partial x \\ \partial u/\partial y \\ \partial u/\partial z \end{pmatrix}_{123}$
- Aggregating, we can get global matrix $G_{3|F|\times|V|}$
  - PL vertex functions -> face PC gradients.
  - $3|F|$ - one vector (xyz) in each face.

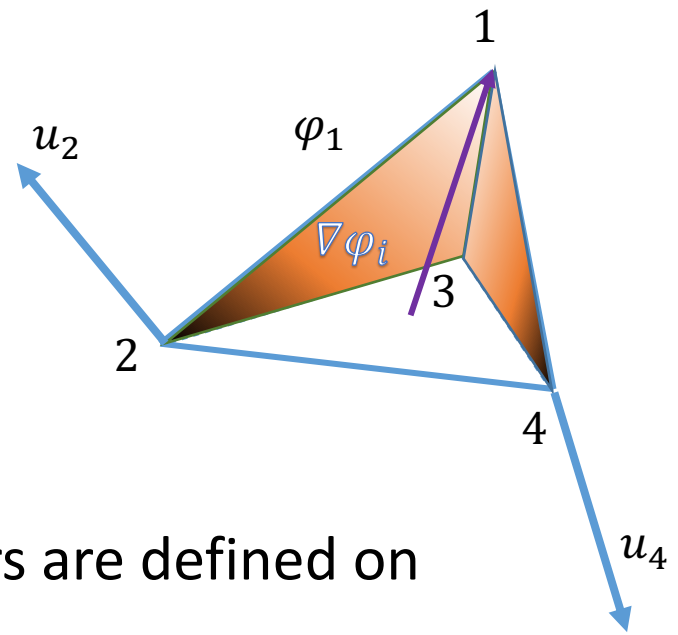
Linear Elasticity in FEM

- Discretization: deformation field on vertices
  - Where each vertex moves.
  - PL inside each tetrahedron\triangle.
- Reminder: Jacobian: $Ju(p)$: $\mathbb{R}^{3\times 3} = \begin{pmatrix} \nabla u_x(p) \\ \nabla u_y(p) \\ \nabla u_z(p) \end{pmatrix}$
- We simply work with three scalar functions: u_x, u_y, u_z.
- Denoting: $\vec{u}_x = \begin{pmatrix} u_{x,1} \\ u_{y,1} \\ u_{z,1} \end{pmatrix}$

$$\begin{pmatrix} \nabla u_x{}^T \\ \nabla u_y{}^T \\ \nabla u_z{}^T \end{pmatrix} = J_{123} \begin{pmatrix} \vec{u}_x \\ \vec{u}_y \\ \vec{u}_z \end{pmatrix} = \begin{pmatrix} G_{123} & & \\ & G_{123} & \\ & & G_{123} \end{pmatrix} \begin{pmatrix} \vec{u}_x \\ \vec{u}_y \\ \vec{u}_z \end{pmatrix}$$

- Sanity check: J_{123} is a 9×9 matrix.

Tetrahedral Meshes



- Elements $e = [1234]$.
- "straightforward" generalization
  - Scalar functions/deformation vectors are defined on vertices
  - Piecewise-linear inside each tet volume.
  - Gradients piecewise-constant in each tet.
  - ➔Gradients orthogonal to opposite face normal.
- Single tet Jacobian: J_e: 9×12 matrix
  - Input vertex deformation field $3\times4 = 12$.
  - Output tet jacobian: $3\times3 = 9$.

Linear Elasticity in FEM

- Full Lagrangian strain tensor:

$$\mathbf{E} = \frac{1}{2}(Ju^T Ju + Ju + Ju^T)$$

- Not linear inside tets!
- Linear elasticity approximation:

$$\boldsymbol{\varepsilon} \approx \frac{1}{2}(Ju + Ju^T)$$

- Good for small deformations without much rotation.

Discrete Strain Tensor

- Written explicitly:

$$\boldsymbol{\varepsilon} = \frac{1}{2}(Ju + Ju^T) = \frac{1}{2}\begin{pmatrix} 2\frac{\partial u_x}{\partial x} & \frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} & \frac{\partial u_x}{\partial z} + \frac{\partial u_z}{\partial x} \\ \frac{\partial u_y}{\partial x} + \frac{\partial u_x}{\partial y} & 2\frac{\partial u_y}{\partial y} & \frac{\partial u_y}{\partial z} + \frac{\partial u_z}{\partial y} \\ \frac{\partial u_z}{\partial x} + \frac{\partial u_x}{\partial z} & \frac{\partial u_z}{\partial y} + \frac{\partial u_y}{\partial z} & 2\frac{\partial u_z}{\partial z} \end{pmatrix}$$

- Only 6 relevant elements (rest are symmetric):

$$\begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \\ \varepsilon_{xy} \\ \varepsilon_{yz} \\ \varepsilon_{zx} \end{pmatrix} = \begin{pmatrix} \frac{\partial}{\partial x} & & \\ & \frac{\partial}{\partial y} & \\ & & \frac{\partial}{\partial z} \\ 0.5\frac{\partial}{\partial y} & 0.5\frac{\partial}{\partial x} & \\ & 0.5\frac{\partial}{\partial z} & 0.5\frac{\partial}{\partial y} \\ 0.5\frac{\partial}{\partial z} & & 0.5\frac{\partial}{\partial x} \end{pmatrix} \begin{pmatrix} u_x(p) \\ u_y(p) \\ u_z(p) \end{pmatrix}$$

Discrete Strain Tensor

- In a single tet: $J_e u$ is constant➔ $\boldsymbol{\varepsilon}_e$ is constant.
- We want to get $\boldsymbol{\varepsilon}_e$ as a direct function of u.
- We have:

$$\boldsymbol{\varepsilon}_e = \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \\ \varepsilon_{xy} \\ \varepsilon_{yz} \\ \varepsilon_{zx} \end{pmatrix} = DJ_e \begin{pmatrix} \vec{u}_x \\ \vec{u}_y \\ \vec{u}_z \end{pmatrix},$$

Where D is a permutation matrix that "chooses" the corresponding values of $J_e u$

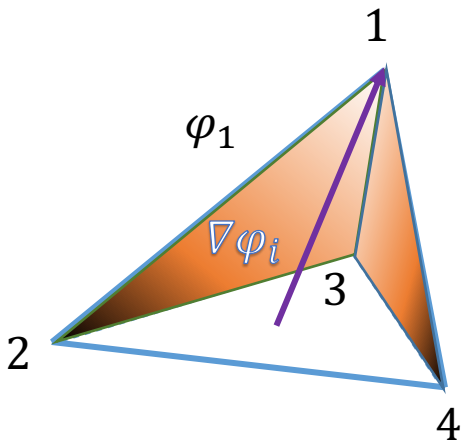
- Sanity check: D is of size 6×9 and does not depend on the shape of any tet (only contains non-zeros values "1" or "0.5").

Discrete Strain Tensor

- Strain tensor per face:

$$\boldsymbol{\varepsilon}_e = DJ_e u_e = B_e u_e$$

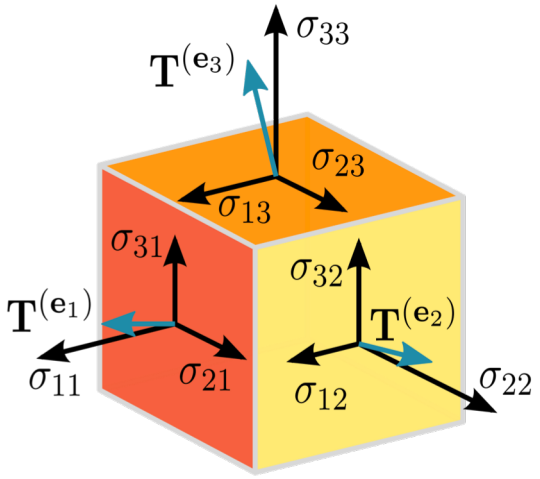
- u_e: all $u_{xyz,1234}$ values (12 in total).
- Note: B_e: $\mathbb{R}^{6x12}$.
- B_e contains derivatives of $\varphi_{1,2,3,4}(p)$
  - $\varphi_{1,2,3,4}$ are linear inside e.
  - Derivatives of φ_i are constant inside e.
- B_e is constant inside the element!



Discrete Stress Tensor

- Stress and strain are related by Hooke's law
  - Remember $\vec{F} = -k\vec{x}$?
- Again, because of symmetry, we use the compact form:

$$\boldsymbol{\sigma}_e = \begin{pmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{zz} \\ \sigma_{xy} \\ \sigma_{yz} \\ \sigma_{zx} \end{pmatrix}$$



Discrete Stress Tensor

- Stress and strain are related by discrete stiffness tensor $\mathbf{C}_e$: $\mathbb{R}^{6\times 6}$.

$$\sigma_e = \mathbf{C}_e \varepsilon_e = \mathbf{C}_e B_e u_e$$

- Relatively easy structure in the discrete case:

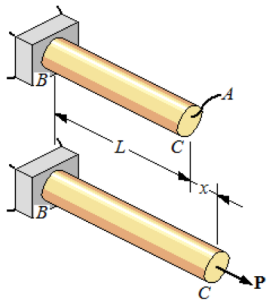
$$\mathbf{C}_e = \begin{pmatrix} \lambda+2\mu & \lambda & \lambda & & & \\ \lambda & \lambda+2\mu & \lambda & & & \\ \lambda & \lambda & \lambda+2\mu & & & \\ & & & \mu & & \\ & & & & \mu & \\ & & & & & \mu \end{pmatrix}$$

- Where $\mu = \frac{Y}{2(1+\upsilon)}, \lambda = \frac{\upsilon Y}{(1+\upsilon)(1-2\upsilon)}$
  - Y – Young's modulus.
  - υ – Poisson's ratio.
- μ and λ are known as Lamé coefficients.

Strain Energy

- Potential energy gained when applying strain to object:

$$U_e = \frac{1}{2}\int_T \langle \boldsymbol{\sigma}_e, \boldsymbol{\varepsilon}_e \rangle dV$$



http://www.engineeringarchives.com/img/les_mom_strainenergydensity_1.png

- We have that $\sigma_e = \mathbf{C}_e B_e u_e$ and $\boldsymbol{\varepsilon}_e = B_e u_e$.
- Then:

$$\langle \boldsymbol{\sigma}_e, \boldsymbol{\varepsilon}_e \rangle = (\boldsymbol{\sigma}_e)^T \boldsymbol{\varepsilon}_e = {u_e}^T {B_e}^T C_e B_e u_e$$

- Both are constant inside volume, so:

$$U_e = \frac{1}{2} Vol(e) \times {u_e}^T {B_e}^T C_e B_e u_e = \boxed{\frac{1}{2} {u_e}^T K_e u_e}$$

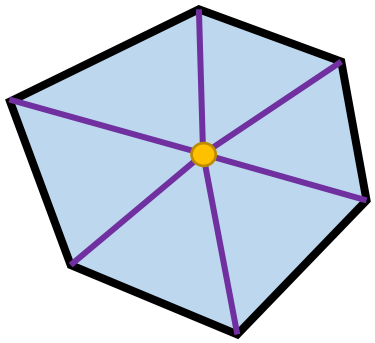
- K_e: stiffness matrix of 12×12
  - Only depends on the original geometry and material properties!

Elastic Forces

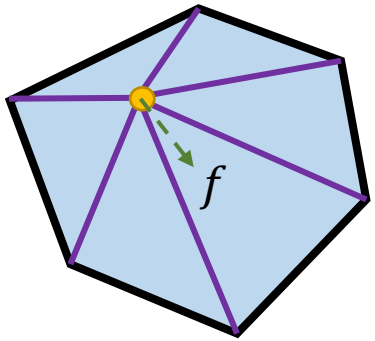
- Derivatives of the potential energy:

$$f_e = \frac{\partial U_e}{\partial u_e} = K_e u_e$$

- Each element of f_e corresponds to an element of u_e
  - Inner force acting on that coordinate due to deformation.



Rest State



Deformation

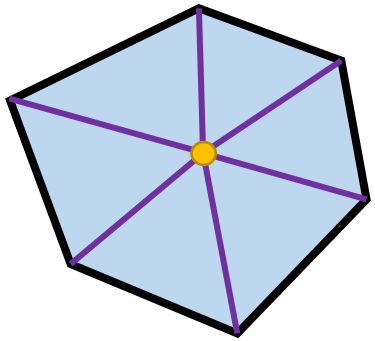
$$f_e = \frac{\partial U_e}{\partial u_e} = K_e u_e$$

Elastic Forces

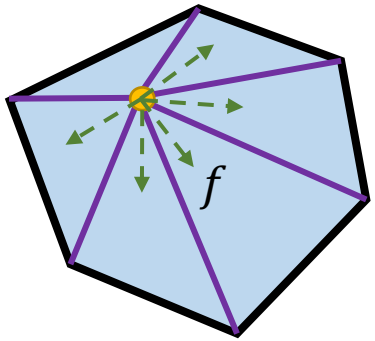
- Net force on each vertex: sum of forces from all adjacent elements.
- Aggregated into a big matrix:

$$f = Ku$$

  - $u_{1\times 3|v|}, f_{1\times 3|v|}, K_{3|v|\times 3|v|}$
- Interpretation: a “force Laplacian”.
- Pulling vertex toward “rest-state equilibrium”.



Rest State



Deformation

Dynamic Deformation Equation

- Computing new positions $x(t)$:

$$Mx''(t) + Dx'(t) + K(x - x_0) = f_{ext}$$

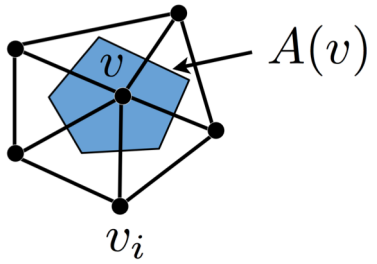
  - M : Mass matrix
  - D: Damping matrix
  - K: stiffness matrix
  - f_{ext}: other forces (gravity etc.)

$$Mx''(t) + Dx'(t) + K(x - x_0) = f_{ext}$$

Mass Matrix

- Suppose we are given density ρ for each tetrahedron.
- We need mass per vertex (where acceleration is).
- Solution: use Voronoi area:

$$m_v = \sum_{e \in N(v)} \frac{1}{4} \rho V(e)$$



- Assuming
- Mass matrix is then

$$M = diag\{m\}$$

  - Each element in the diagonal corresponds to which vertex is represented in the respective column of x.

Dampening Matrix

- We often use Rayleigh Dampening:

$$D = \alpha M + \beta K$$

- α, β are dampening parameters.
  - A whole theory of how to compute them and to what effect…
- For our purposes: keep as parameters, limit by small values $0 \leq \alpha, \beta \leq 0.02$.

Time Integration

- Best practice: a variant on implicit backward integration
- Solve for:

$$x(t + \Delta t) = x(t) + \Delta t \cdot v(t + \Delta t)$$

And similarly for $v(t + \Delta t)$ and $a(t + \Delta t)$.

- We need to solve the linear system:

Constant matrix in time!

$$[M + \Delta t D + \Delta t^2 K] v(t + \Delta t) = M v(t) - \Delta t [K(x(t) - x_0) - f_{ext}]$$

variables

Right-hand side, changes with time.

- Integrating for position with the formula above.

$$[M + \Delta t D + \Delta t^2 K]v(t + \Delta t) = Mv(t) - \Delta t[K(x(t) - x_0) - f_{ext}]$$

Solving the system

- We need to solve the linear system in every time step.
- Abstract form: $Ax(t) = b(t)$, A time independent.
- Sounds slow; however, we should use the fact that A does not change!
  - Reduce A to some convenient form in the beginning of time
  - In each iteration: change $b(t)$ and solve for $x(t)$.

LU Decomposition

- Every square matrix A can be decomposed into $A = L \cdot U$, where:
  - L lower triangular
  - U upper triangular
  - $A = \begin{pmatrix} A_{11} & \cdots & A_{1n} \\ \vdots & \ddots & \vdots \\ A_{n1} & \cdots & A_{nn} \end{pmatrix} =$

$$\boldsymbol{L} \begin{pmatrix} L_{11} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ L_{n1} & \cdots & L_{nn} \end{pmatrix} \cdot \begin{pmatrix} U_{11} & \cdots & U_{1n} \\ \vdots & \ddots & \vdots \\ 0 & \cdots & U_{nn} \end{pmatrix} \boldsymbol{U}$$

Solving systems with LU Decomposition

- Decompose $Ax = LUx$.
- Solve in two steps:
  - Solve $Ly = b$ (one forward substitution)
  - Solve $Ux = y$ (one backward substitution)
- Decomposition is relatively expensive.
- But solving for a given RHS is cheap.
- Consequence: decomposition is very beneficial for fixed A and varying b!

Cholesky Decomposition

- If M is symmetric positive definite, LU reduces to:

$$M = LL^*$$

- $*$ means *"conjugate transpose"*
  - For real matrices: just LL^T.
  - For complex matrices: every number $a + ib$ becomes $\overline{a + ib} = a - ib$.
- Cheaper, more stable decomposition.

Sparse Representation

- M, D, K have few elements w.r.t. to their sizes
  - M just diagonal.
  - K creates forces for vertices only from their neighbors.
- Efficiency and memory requirements: use sparse matrices!
- Direct Representation:
  - List of triplets (row, col, value).
- Example:

$$A = \begin{pmatrix} 1 & & \\ & & -5 \\ & 4 & \end{pmatrix}$$

Represented as $\{(1,1,1), (2,3,-5), (3,2,4)\}$.

- The common way to feed sparse matrices.
- Sometimes called COOrdinate format (COO)

Compressed Sparse Row/Column

- The common way to represent and manipulate in memory.
- For CSR we have 3 arrays A,IA,JA
  - A: list of non-zeros (NZ) row-by-row
  - IA: defined recursively:
    - IA[0]=0
    - IA[i]=IA[i-1]+NNZ(row(i-1))
  - JA: column value of each element in A.
- CSC: the same with columns instead of rows and vice versa.

$$\begin{pmatrix} 10 & 20 & 0 & 0 & 0 & 0 \\ 0 & 30 & 0 & 40 & 0 & 0 \\ 0 & 0 & 50 & 60 & 70 & 0 \\ 0 & 0 & 0 & 0 & 0 & 80 \end{pmatrix}$$

A = [10 20 30 40 50 60 70 80]
IA = [0 2 4 7 8]
JA = [0 1 1 3 2 3 4 5]

Sparse Representation

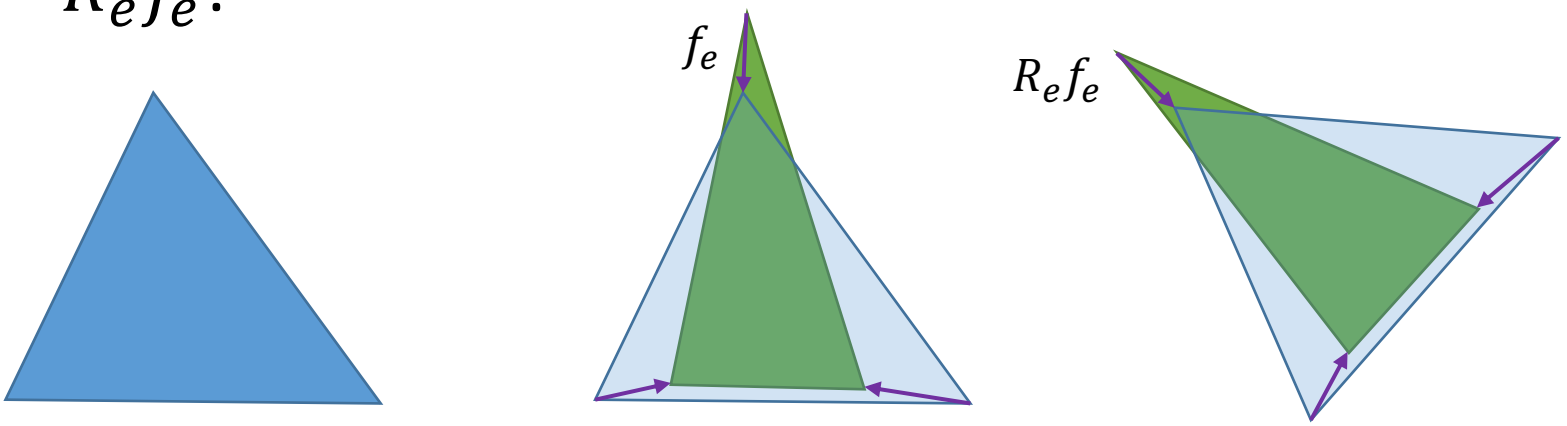
- Cheap:
  - Sparse * vector for CSR.
  - Vector * sparse for CSC.
  - Slicing (but don't use that).
- What's Relatively cheap:
  - Decompositions (sparse LU, Cholesky, etc.)
- What's Expensive:
  - Transpose (conversion from CSC to CSR)
  - Random memory access.

"Index Hell"

- Figuring out the matrices can be quite a job.
  - "which vertex, which coordinate xyz to which element multiplies with what" ➔ index hell.
  - Easy to get wrong!
- Life-(and sanity-; and grade-) saving tip:
  - Create Complicated matrices (such as *K*) through the composition of easier matrices.
- Example: compute a huge K' that is just made of blocks of the much easier K_e per-element.
- Compose with a matrix M that averages numbers from elements to adjacent vertices to get $K = MK'M^T$.
  - Similar with the matrices G, J, etc.

Corotational Elements

- Insight: rotating an element should not change the strain energy.
- Conclusion: Given an element in position x, with strain energy U_e, and consequent elastic forces f_e, the forces on a rotated element $R_e x$ should be $R_e f_e$!



Müller and Gross, "Interactive Virtual Materials"

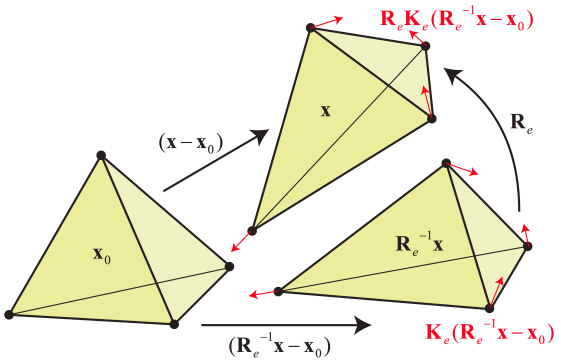
Corotational Elements

- Method:
  - Estimate rotation R_e, and factor out rotation from the deformed object x: ${R_e}^{-1}\mathrm{x}$
  - Compute elastic forces of unrotated object:

$$K_e\left({R_e}^{-1}\mathrm{x} - x_0\right)$$

  - Rotate back to deformed state to get actual forces:

$$f_e = R_e K_e\left({R_e}^{-1}\mathrm{x} - x_0\right) = \boxed{K_e' u_e}$$



Corotational Elements

- Advantages: able to work with large rotations
- Disadvantages: stiffness matrix not constant anymore.
  - Have to solve system using conjugate gradients…
- How to estimate rotation R_e?

Finding Best-Fit Rotation

- Original positions *x,* deformed positions *x'*.
- *Create* stacked coordinates of edges of original points:

$$P = \begin{pmatrix} x_1 - x_2 \\ \\ x_1 - x_d \end{pmatrix}, Q = \begin{pmatrix} {x_1}' - {x_2}' \\ \\ {x_1}' - {x_d}' \end{pmatrix}$$

- Compute matrix: $S = P^T Q \in \mathbb{R}^{3\times 3}$
- Singular value decomposition (SVD) extracts rotation from S

$$\mathbf{S} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T \longrightarrow \mathbf{R} = \mathbf{U}\mathbf{V}^T$$

Singular Value Decomposition

- Every linear operator (=matrix $M_{n\times m}$) can be decomposed to:
  - Rotation (Change of basis): $V_{m\times m}$.
  - Stretch in the new basis: $\Sigma_{n\times m}$
    - Note (possible) change in dimension.
  - Rotation (another change of basis): $U_{n\times n}$

- For vector p we get $Mp = U\Sigma V^T p$.

Examples

https://www.youtube.com/watch?v=4Wl0ksysYKM
https://www.youtube.com/watch?v=6f3UYHnR4zU
https://www.youtube.com/watch?v=p5uhnSw8_Xw