Linear Elasticity

Steve Rotenberg
CSE291: Physics Simulation
UCSD
Spring 2019

Homogeneous Transforms

- In 3D graphics and physics, we often need to move virtual objects around in a virtual space
- It is common to use 4x4 homogeneous matrix transforms to accomplish this

$$\mathbf{M} = \begin{bmatrix} a_x & b_x & c_x & d_x \\ a_y & b_y & c_y & d_y \\ a_z & b_z & c_z & d_z \\ a_w & b_w & c_w & d_w \end{bmatrix}$$

- With these transforms, we can combine an arbitrary sequence of rotations, translations, scales, and shears
- They can also be used for perspective viewing transforms or any combination of these different operations
- Note: we can also do reflections, which are just scales by negative values

Affine Transformations

- We will not be dealing with viewing projections, as we are mainly interested in using transforms to move things around
- Therefore, we will not be using the bottom row of the matrix and it will almost always be 0 0 0 1
- This will limit us to rotation, translation, scale, and shear

$$\mathbf{M} = \begin{bmatrix} a_x & b_x & c_x & d_x \\ a_y & b_y & c_y & d_y \\ a_z & b_z & c_z & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- The set of matrices in this form is known as the *affine group*
- An affine transformation will preserve parallel lines (i.e., two parallel lines will still be parallel after an affine transformation)

Vector Transformations

- Let's say we have 3D vector $\mathbf{v}$ representing a point on some object, defined in the objects local coordinate system
- We also have a 4x4 matrix $\mathbf{M}$ representing how the object is transformed into the global (world) coordinate system
- To transform local $\mathbf{v}$ into world space $\mathbf{v}'$, we have to momentarily turn $\mathbf{v}$ into a 4D vector $\mathbf{v}_4$ and place a 1 in the 4th coordinate

$$\begin{bmatrix} v'_x \\ v'_y \\ v'_z \\ 1 \end{bmatrix} = \mathbf{v'}_4 = \mathbf{M} \cdot \mathbf{v}_4 = \begin{bmatrix} a_x & b_x & c_x & d_x \\ a_y & b_y & c_y & d_y \\ a_z & b_z & c_z & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \\ 1 \end{bmatrix}$$

- If we always have 0 0 0 1 in the bottom row of **M**, then $\mathbf{v'}_4$ will always have a 1 in the 4th coordinate as well
- For this reason, we will bend notation rules by combining 3D vectors and 4x4 matrices and write transforms like this:

$$\mathbf{v}' = \mathbf{M} \cdot \mathbf{v}$$

Position vs. Direction Vectors

$$\mathbf{v}' = \mathbf{M} \cdot \mathbf{v} = \begin{bmatrix} a_x & b_x & c_x & d_x \\ a_y & b_y & c_y & d_y \\ a_z & b_z & c_z & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \\ 1 \end{bmatrix}$$

- We see that the transformed vector **v'** will be rotated, scaled, and sheared by the 3x3 **abc** portion of the matrix and will be translated by the **d** vector
- This works fine for vectors representing positions, but not for vectors representing directions (like an axis of rotation)
- For directional vectors, we only want the 3x3 portion to affect direction, and we want to ignore the translation
- For a directional vector **w**, we expand it with a 0 in the 4th coordinate:

$$\mathbf{w}' = \mathbf{M} \cdot \mathbf{w} = \begin{bmatrix} a_x & b_x & c_x & d_x \\ a_y & b_y & c_y & d_y \\ a_z & b_z & c_z & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} w_x \\ w_y \\ w_z \\ 0 \end{bmatrix}$$

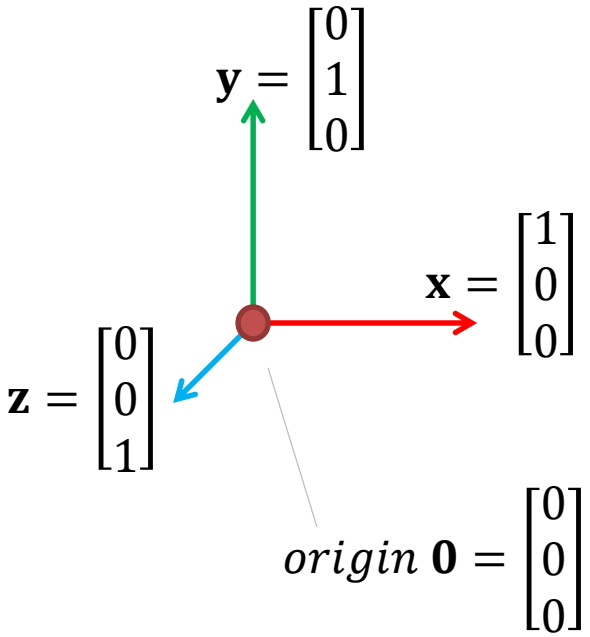
Column Vectors

$$\mathbf{M} = \begin{bmatrix} a_x & b_x & c_x & d_x \\ a_y & b_y & c_y & d_y \\ a_z & b_z & c_z & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- If **M** is a matrix that transforms from an object's local frame to the world frame, then the 4 column vectors (**a**, **b**, **c**, & **d**) will have a direct geometric interpretation
- **d** will be the translation vector
- **a**, **b**, and **c** are what you get if you transform the **x**, **y**, and **z** axes by matrix **M**
- In other words, they represent the object's local **x**, **y**, and **z** axes, transformed into world space

Coordinate System Visualization

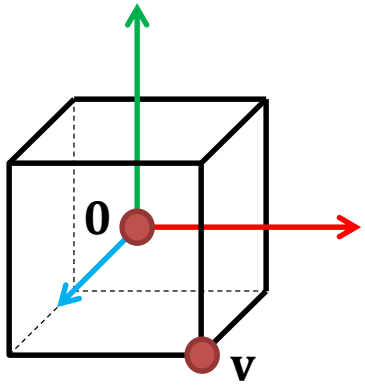
- We will start with a basic right-handed 3D coordinate system
- Direction vectors are represented with arrows and position vectors are small circles
- We will use an XYZ->RGB color scheme



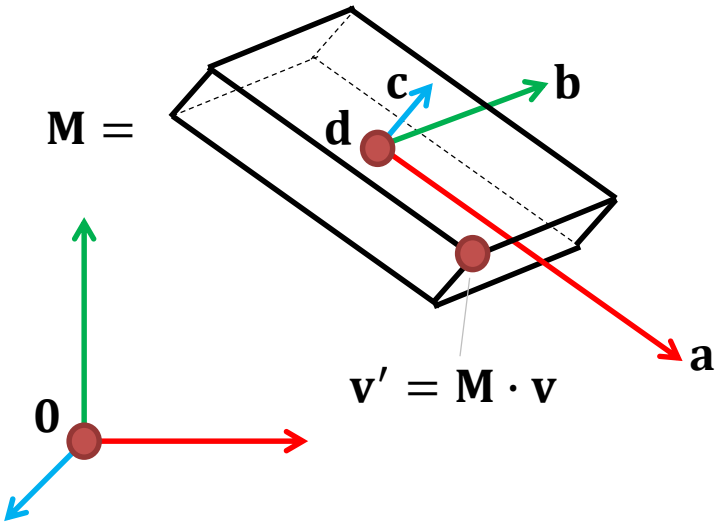
Matrix Visualization

- If we have a matrix $\mathbf{M}$ that transforms an object from its local (object space) coordinate system into the global (world space) coordinate system
- It's column vectors are $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$, and $\mathbf{d}$

$$\mathbf{M} = \begin{bmatrix} a_x & b_x & c_x & d_x \\ a_y & b_y & c_y & d_y \\ a_z & b_z & c_z & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



local frame



global frame

Rigid Transformations

- If our affine matrix **M** is further limited to only having rotations and translations (no scales or shears), it will be in the *rigid transformation group*
- In this case, the column vectors **a**, **b**, and **c** will be limited to unit length (no scaling) and they will be 90 degrees apart (the dot products **a·b**, **a·c**, and **b·c** will be 0)
- A rigid object has 6 degrees of freedom: 3 translations and 3 rotations
- The matrix **M** may still have 12 variable numbers, but there are 6 rigidity constraints applied:

$$\mathbf{a} \cdot \mathbf{a} = \mathbf{b} \cdot \mathbf{b} = \mathbf{c} \cdot \mathbf{c} = 1$$
$$\mathbf{a} \cdot \mathbf{b} = \mathbf{a} \cdot \mathbf{c} = \mathbf{b} \cdot \mathbf{c} = 0$$

- This also means that the upper 3x3 portion of the 4x4 matrix is *orthonormal*

Non-Rigid Transformations

- If our affine matrix **M** contains scaling and shearing, then it is a non-rigid transform and is effectively a linear deformation
- It has 12 degrees of freedom (3 rotations, 3 translations, 3 scales, 3 shears), and has 12 independent variables in the matrix

$$\mathbf{M} = \begin{bmatrix} a_x & b_x & c_x & d_x \\ a_y & b_y & c_y & d_y \\ a_z & b_z & c_z & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Keep in mind that although the translation can easily be extracted as the **d** vector, the rotations, scales, and shears are intertwined in the **a**, **b**, and **c** vectors and don't have a one-to-one correspondence

Linear Elasticity

Elasticity

- *Elasticity* refers to the property of a material to deform under force and return to its initial shape when the force is removed
- We've talked about elastic springs, which are a 1D idealization
- Today, we will be interested in full 3D elasticity in order to model deformable solids

Elastic Materials

- We can treat many materials (and the objects made from those materials) as purely elastic as long as we aren't deforming things too much
- This includes metals, concrete, plastic, glass, carbon fiber, cloth, paper, rubber, etc., etc.
- It doesn't include fluids like air or water
- Even foams and granular materials like mud can be approximated as elastic as long as we keep the deformation to a minimum

Non-Elastic Properties

- Let's say we have a metal object like a spoon
- If we bend it just a little bit and then let go, it will return back to its original shape. If we hold one end and bang it on the table, it will oscillate back and forth making an audible tone. These are purely elastic behaviors

- If we bend it too much, we will cause a permanent kink, known as a *plastic deformation*
- If we bend it back and forth quickly, we will generate heat, which involves coupling of mechanics and thermodynamics
- If we keep bending it, it will develop metal fatigue, resulting in permanent weakening
- If we pull or shear it hard enough it will eventually fracture
- These are all non-elastic behaviors of materials that we will cover in more detail in the next couple lectures
- For today, we will stick to elasticity and more specifically, linear elasticity

Hooke's Law

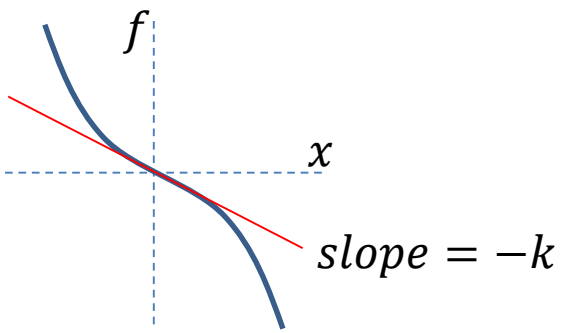
- We looked at springs in the first lecture:

$$f = -kx$$

- Where f is the resulting force, k is the spring constant and x is the displacement
- This is the 1D version of *Hooke's Law*
- We could write it in a 3D vector form as $\mathbf{f} = -k\mathbf{x}$, which is still a 1D spring but oriented in 3D space
- Note: this is sometimes written as $f = kx$ which is just another way of looking at it. If you think of *f* as the force required to cause the displacement, you would use this version. If you think of it as the force resulting from the displacement, you would use $f = -kx$. Remember Newton's Third Law says every action has an equal and opposite reaction, so both forms are valid

Linear vs. Nonlinear

- Hooke's Law assumes a linear relationship between force and displacement
- This is reasonable for small displacements, but clearly can't work forever
- A real spring, for example, will eventually get fully stretched and the force will grow much faster than linear
- Eventually, the spring will break and the force will go to 0 permanently
- If the spring is compressed too much, the coils will eventually touch and the force will once again grow much faster than linear until the metal in the spring gets squashed
- In other words, real springs are definitely nonlinear, but we can still treat them as linear if we keep the displacement within a small enough range
- Hooke's Law is effectively using the first two terms of a Taylor series to approximate a nonlinear spring function



Strain

- The term *strain* refers to the geometric measurement of deformation
- The displacement x in the linear spring equation is a 1D form of strain
- If the spring connects two particles at positions $\mathbf{r}_1$ and $\mathbf{r}_2$, then we can measure the strain x as the difference between the rest length l_0 and the current length:

$$x = |\mathbf{r}_1 - \mathbf{r}_2| - l_0$$

Stress

- The term *stress* refers to the physical forces resulting from deformation
- In other words, we could describe Hooke's Law as a type of *stress-strain relationship*
- We will extend the concepts of stress and strain into 3D, resulting in a generalized form of Hooke's Law

1D Elastic Simulation

- We talked about 1D springs that connect between two 3D particles
- A 1D linear elastic spring has two constants:
  - A spring constant k defining the stiffness
  - A rest length l_0 defining the rest shape (i.e., when it is not deformed)

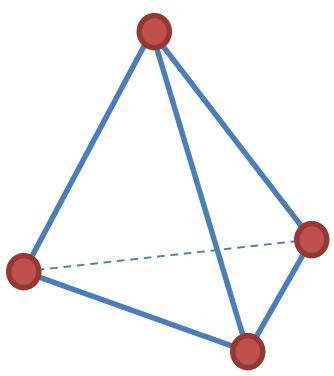
- When we simulate with a spring:
  1. We first measure the strain by comparing its current length to its rest length
  2. We then apply the stress-strain relationship (*f=-kx*) to compute the force
  3. We then turn this into the forces acting on the two particles
- The forces will be equal and opposite, and so will cancel out, causing no net change in momentum
- We do this for every spring in the system. When we're done computing all of the forces in the system, we can integrate to move the system to the next time step

rest state

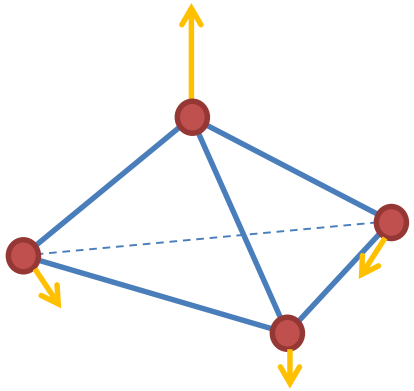
deformed state

3D Elastic Simulation

- 3D solid body simulation extends this exact same process into 3D
- Instead of 1D springs connecting two particles, we will use 3D *tetrahedral elements* to connect four particles
  1. We first compute the strain by comparing the current shape of a tetrahedron to its rest shape
  2. We then apply a stress-strain relationship to compute the internal stress
  3. We then turn this stress into forces at the four particles
- The forces will all add up to 0, thus causing no net change in momentum



rest state



deformed state

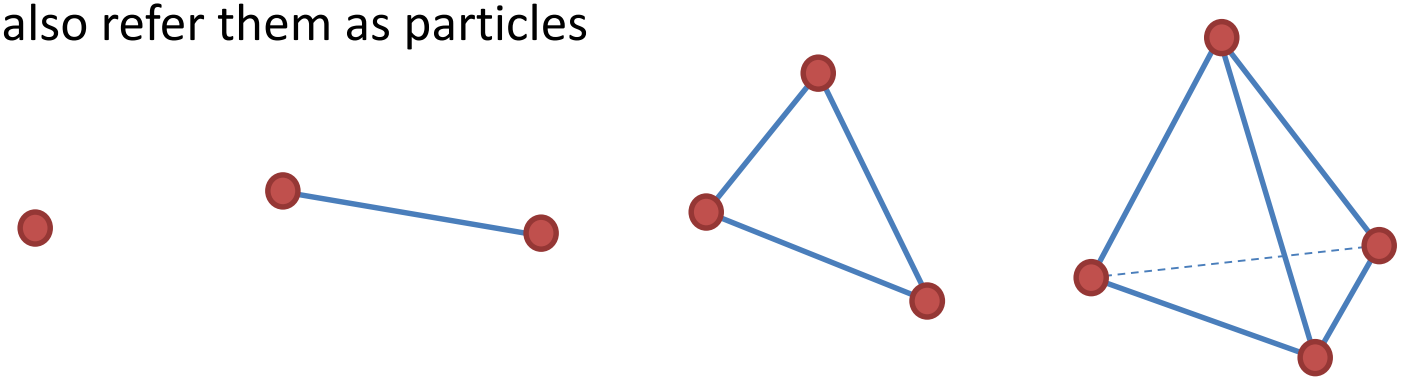
Linear Elasticity

- If we use a linear relationship between the strain and stress, we are operating in the domain of *linear elasticity theory*
- Linear elasticity is essentially a generalization of Hooke's Law to 3-dimensional solids

Finite Elements

Simplex Elements

- We can use the term *simplex* to describe these *n*-dimensional versions of a triangle:
  - 0-simplex: point
  - 1-simplex: segment
  - 2-simplex: triangle
  - 3-simplex: tetrahedron
  - etc.
- In physics simulation, we'll refer to these as *elements* and for today, we'll mainly use tetrahedral volume elements (3-simplex)
- When speaking more abstractly, we'll refer to the vertices as *nodes*, but we'll also refer them as particles



Finite Elements

- We can partition the volume of a 3-dimensional object into tetrahedral elements and then run physics on these individual elements to approximate the physics of the entire model
- We are essentially taking a continuous material with potentially infinitesimal detail and *discretizing* it into finite sized bits for practical computation
- If we use lots of small elements, we can capture more detail and increase the accuracy, but it will be more computationally expensive
- This is an example of the *finite element method* (FEM) used throughout physics and mathematical modeling
- Today, we will discuss finite elements in their simplest context, limiting them to *linear elastic simplex elements*
- In the next lecture, we will generalize this to allow for more complex material modeling with nonlinear, non-simplex elements and more…

Surface/Shell Elements

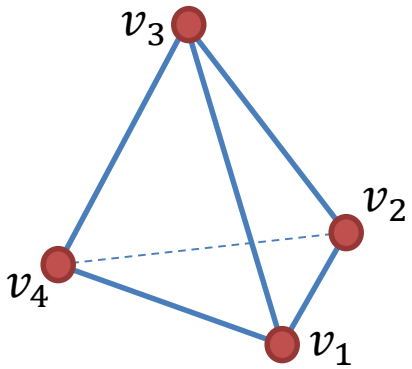
- We used 1D simplex elements already to model springs
- We can also model thin surfaces in 3D space using 2D triangular *shell* elements
- This is a better way to model cloth than using a network of 1D springs as we've previously done
- In fact, we can easily combine 1D, 2D and 3D elastic elements in the same simulation to model:
  - 1D ropes, cables, poles, beams
  - 2D cloth, paper, sheet metal, thin walls
  - 3D jello, concrete blocks, other solids

Mesh Representations

- There are different approaches to representing meshes of finite elements
- In the last lecture, we discussed architectural approaches to decoupling the integrator from the physics system
- It can also be a good idea to decouple the mesh from the physics
- This way, we can use the same mesh data structures for solid mechanics, thermodynamics, fluid dynamics, or other 3D field equations (electromagnetics, etc.)
- Also this makes it easy to separate out complex geometric operations such as mesh generation into their own library that only has to deal with meshing and not physics
- We will discuss this more when we talk about mesh generation in a later lecture
- For today, we'll keep it simple and just assume we have an array of particles and an array of elastic tetrahedra that connect 4 particles, similar to the way we described particles and springs in the first lecture

Vertex Numbering

- Let's assume we have a mesh of tetrahedra that we are loading from a file (or generating with a meshing algorithm…)
- At a minimum, we would represent a model as an array of position vectors and an array of tetrahedra, each defined with 4 vertices, which are just integers that index into the array of positions
- It is customary to define triangle vertices in a counter-clockwise order, so we will define our tetrahedron such that the first 3 vertices define one of the triangle faces in counter-clockwise order, and the fourth vertex would be the remaining one

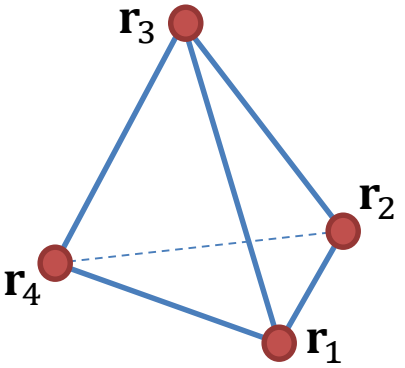


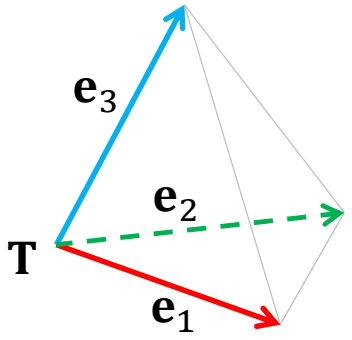
Tetrahedral Frame

- We can build a 3x3 matrix out of three edge direction vectors to create a *tetrahedral frame* $\mathbf{T}$
- If $\mathbf{r}_n$ is the 3D position of the vertex indexed by v_n, then we can compute the matrix $\mathbf{T}$ as:

$$\mathbf{e}_1 = \mathbf{r}_1 - \mathbf{r}_4$$
$$\mathbf{e}_2 = \mathbf{r}_2 - \mathbf{r}_4$$
$$\mathbf{e}_3 = \mathbf{r}_3 - \mathbf{r}_4$$

$$\mathbf{T} = \begin{bmatrix} \mathbf{e}_{1x} & \mathbf{e}_{2x} & \mathbf{e}_{3x} \\ \mathbf{e}_{1y} & \mathbf{e}_{2y} & \mathbf{e}_{3y} \\ \mathbf{e}_{1z} & \mathbf{e}_{2z} & \mathbf{e}_{3z} \end{bmatrix}$$

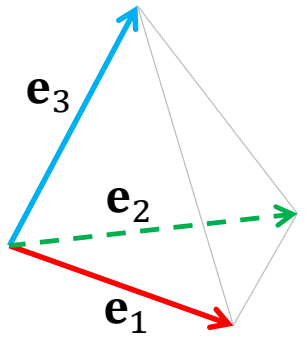




Volume of a Tetrahedron

- The volume of a tetrahedron is:

$$vol = \frac{1}{6}(\mathbf{e}_1 \times \mathbf{e}_2) \cdot \mathbf{e}_3$$



- We should get a positive number if we followed our numbering convention
- If we get zero or a negative number at any point in the simulation, it means the tetrahedron has been flattened or turned inside out, which can be a problem
- For linear elastic models, the volume change should be limited to a minimal amount during the simulation (maybe 5-10%)

Strain Computation

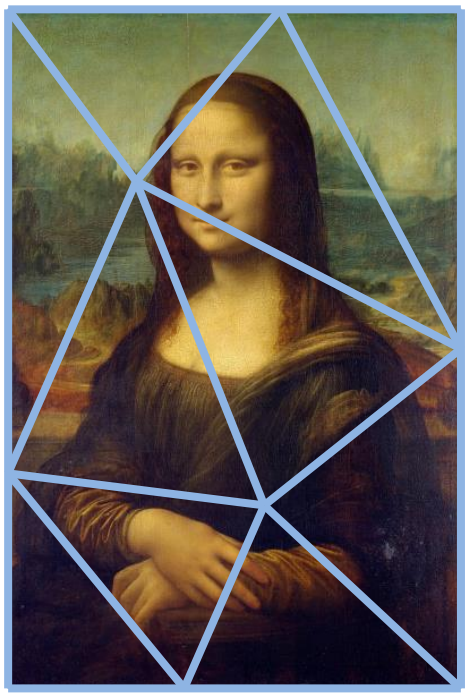
Strain Computation

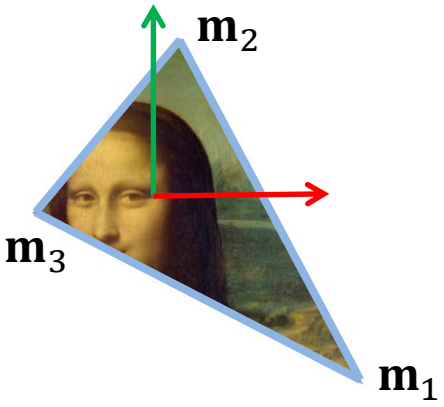
- We wish to partition a deformable volumetric object into a set of tetrahedra
- For each tetrahedron, we need to:
  1. Measure the deformation (strain)
  2. Relate that to the internal forces (stress)
  3. Turn the internal stress into forces on the particles
- We start by examining step 1 in detail

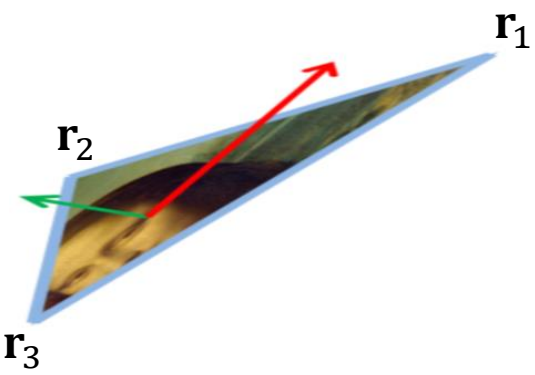
Material Space

- If we are to analyze the deformation of a tetrahedron, we need some representation of its undeformed state to compare to
- We consider that the undeformed tetrahedron is a finite piece from a larger undeformed block of material
- We represent the undeformed tetrahedron as four coordinates in this material space
- Each vertex v_n in the deformable model has a current position $\mathbf{r}_n$ in world space as well as its original position $\mathbf{m}_n$ in material space
- This is similar to the concept of texture coordinates in computer graphics

Material & World Space (2D)



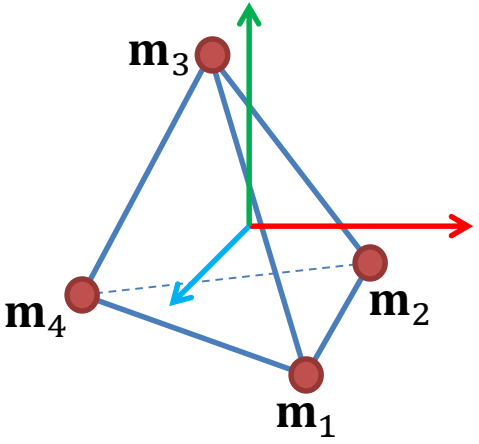




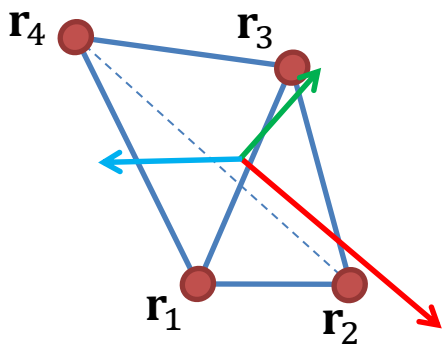
material space

world space

Material & World Space (3D)



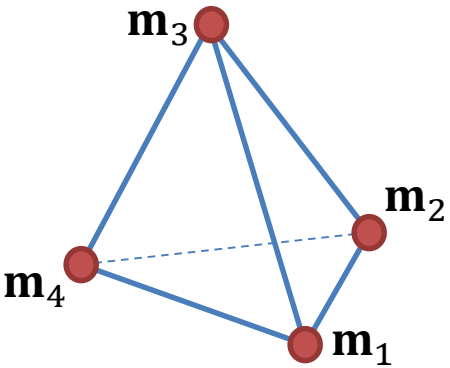
rest state (material space)

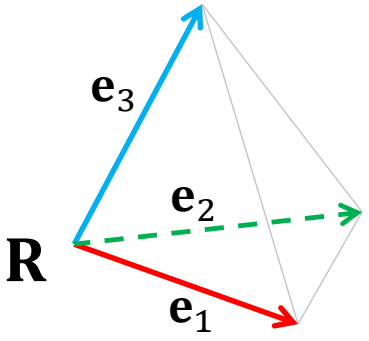


deformed state (world space)

Rest Tetrahedral Frame

- We can define a *rest tetrahedral frame* $\mathbf{R}$ as the 3x3 matrix built from the material coordinates $\mathbf{m}_n$ of the tetrahedron vertices
- This is just like the tetrahedral frame $\mathbf{T}$ calculation from few slides back, but applied specifically to the coordinates of the tetrahedron in its rest (underformed) state





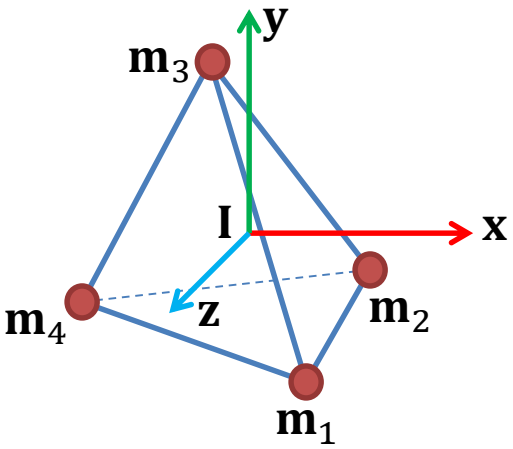
Deformation Gradient

- In the undeformed (rest) state, the material frame is lined up with the XYZ axes, and is represented as an identity matrix $\mathbf{I}$
- When the tetrahedron is deformed, this material frame deforms with it
- The deformed material frame is represented as matrix $\mathbf{F}$, which transforms into the deformed world space configuration
- $\mathbf{F}$ is also known as the *deformation gradient*

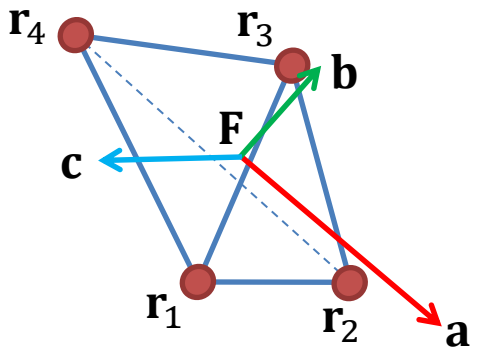
Deformation Gradient

- $\mathbf{F}$ represents the deformed material frame in world space. We will refer to its column vectors as $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{c}$:

$$\mathbf{F} = \begin{bmatrix} a_x & b_x & c_x \\ a_y & b_y & c_y \\ a_z & b_z & c_z \end{bmatrix}$$



rest state (material frame)



deformed state (global frame)

Deformation Gradient

- The deformation gradient is a transformation from the underformed material frame into the deformed frame. It will take a point relative to $\mathbf{I}$ and place it relative to $\mathbf{F}$
- We need to compute $\mathbf{F}$ from things we have available (such as the current and rest tetrahedral matrices)
- We can do this by transforming *into* the rest frame (i.e., by $\mathbf{R}^{-1}$) and then transforming by $\mathbf{T}$:

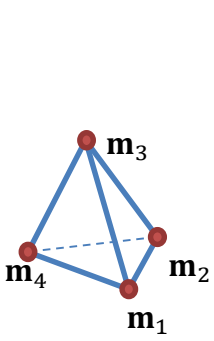
$$\mathbf{F} = \mathbf{T}\mathbf{R}^{-1}$$

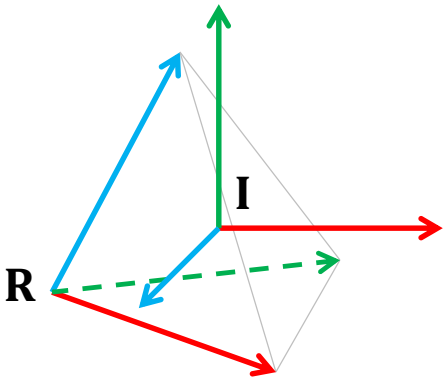
- Note that $\mathbf{R}^{-1}$ is constant, as it is based on the material coordinates. It can therefore be precomputed and stored for each element

Deformation Gradient

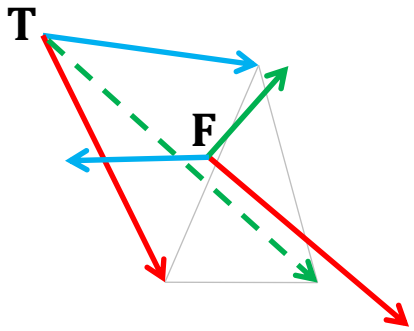
$$\mathbf{F} = \mathbf{T}\mathbf{R}^{-1}$$

- $\mathbf{I}$: underformed material frame (identity matrix)
- $\mathbf{F}$: deformed material frame
- $\mathbf{R}$: rest tetrahedral frame
- $\mathbf{T}$: current tetrahedral frame

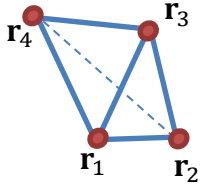




rest state



deformed state



Strain Computation

- The deformation gradient $\mathbf{F}$ represents the material frame deformed into world space
- It has 9 unique numbers that encode 3 rotations, 3 scales, and 3 shears
- We have already removed the 3 translations by using a 3x3 instead of a 4x4 matrix
- We want to remove the 3 rotations, as pure rotation won't cause any deformation, and therefore, no stress or forces
- We want to extract the 3 scale and 3 shear values and represent them in some useful form
- We refer to this general process as *strain computation*
- In order to compute the strain on an element, we first must agree as to exactly how we will represent that strain

Strain Tensor

- Most solid mechanics formulations represent the geometric strain measure as a symmetric *n* x *n strain tensor*, where *n* is the number of dimensions
- In 3D space, this leads to a 3x3 matrix, but because it is symmetric, it contains only 6 unique numbers (NOTE: an *n*-dimensional strain tensor will have $n(n+1)/2$ unique numbers)
- These 6 numbers measure the 6 degrees of deformation
- In general they will represent the 3 scales (squash/stretch along X, Y, or Z) and 3 shears (XY, XZ, or YZ plane)
- However, there are actually different methods for measuring them that can have different properties

Green's Strain Tensor

- A common, classical way to measure strain is with *Green's strain tensor,* $\boldsymbol{\varepsilon}$
- The symmetric 3x3 Green strain tensor is based on the (asymmetric) 3x3 deformation gradient $\mathbf{F}$:

$$\boldsymbol{\varepsilon} = \frac{1}{2}(\mathbf{F}^T \cdot \mathbf{F} - \mathbf{I})$$

$$\boldsymbol{\varepsilon} = \frac{1}{2}\begin{bmatrix} \mathbf{a}\cdot\mathbf{a} - 1 & \mathbf{a}\cdot\mathbf{b} & \mathbf{a}\cdot\mathbf{c} \\ \mathbf{b}\cdot\mathbf{a} & \mathbf{b}\cdot\mathbf{b} - 1 & \mathbf{b}\cdot\mathbf{c} \\ \mathbf{c}\cdot\mathbf{a} & \mathbf{c}\cdot\mathbf{b} & \mathbf{c}\cdot\mathbf{c} - 1 \end{bmatrix}$$

- The diagonal values measure the scale deformation along the **a**, **b**, and **c** vectors: 0 for no scaling, positive for stretching, negative for compressing
- The off-diagonal values measure the shears in the **ab**, **ac**, and **bc** planes: 0 for no shear, positive when the two axes form an acute angle, negative when the two axes form an obtuse angle. The off diagonal values are symmetric, and so there are only 3 unique values (i.e., $\mathbf{a}\cdot\mathbf{b} = \mathbf{b}\cdot\mathbf{a}$, etc.)

Strain Computation

- There are various other strain tensors used in solid modeling and we will discuss a couple of them in the next lecture
- For today, we'll use Green's strain tensor
- Once again, we will try to limit the amount of deformation, as it is not intended to handle large deformations

Strain Computation Summary

- For each element:
  - Gather the 4 vertex positions into vectors $\mathbf{r}_1 \ldots \mathbf{r}_4$
  - Compute edge vectors $\mathbf{e}_1 \ldots \mathbf{e}_3$ and build tetrahedral frame $\mathbf{T}$

$$\begin{aligned} \mathbf{e}_1 &= \mathbf{r}_1 - \mathbf{r}_4 \\ \mathbf{e}_2 &= \mathbf{r}_2 - \mathbf{r}_4 \\ \mathbf{e}_3 &= \mathbf{r}_3 - \mathbf{r}_4 \end{aligned} \qquad \mathbf{T} = \begin{bmatrix} \mathbf{e}_{1x} & \mathbf{e}_{2x} & \mathbf{e}_{3x} \\ \mathbf{e}_{1y} & \mathbf{e}_{2y} & \mathbf{e}_{3y} \\ \mathbf{e}_{1z} & \mathbf{e}_{2z} & \mathbf{e}_{3z} \end{bmatrix}$$

  - Compute deformation gradient $\mathbf{F}$ using precomputed $\mathbf{R}^{-1}$

$$\mathbf{F} = \mathbf{T}\mathbf{R}^{-1}$$

  - Compute Green's strain tensor $\boldsymbol{\varepsilon}$

$$\boldsymbol{\varepsilon} = \frac{1}{2}(\mathbf{F}^T \cdot \mathbf{F} - \mathbf{I})$$

Stress Tensor

Cauchy Stress Tensor

- Stress is a measure of the physical forces within a deformed solid and is ultimately due to intermolecular attraction/repulsion forces
- There are different ways to represent it, but a common method used when deformations are small is with the *Cauchy stress tensor:* $\boldsymbol{\sigma}$
- The Cauchy stress is a measure of the force acting on a differential area in a deformed solid
- It is often written as a 3x3 symmetric tensor (matrix):

$$\boldsymbol{\sigma} = \begin{bmatrix} \sigma_{xx} & \sigma_{xy} & \sigma_{xz} \\ \sigma_{xy} & \sigma_{yy} & \sigma_{yz} \\ \sigma_{xz} & \sigma_{yz} & \sigma_{zz} \end{bmatrix}$$

- It is actually a measure of pressure and has units of force per area, or N/m^2
- If an object is not deformed, then the stress is 0 everywhere
- If it is deformed in some way, the stress will vary throughout the solid

Pressure on a Plane

$$\boldsymbol{\sigma} = \begin{bmatrix} \sigma_{xx} & \sigma_{xy} & \sigma_{xz} \\ \sigma_{xy} & \sigma_{yy} & \sigma_{yz} \\ \sigma_{xz} & \sigma_{yz} & \sigma_{zz} \end{bmatrix}$$

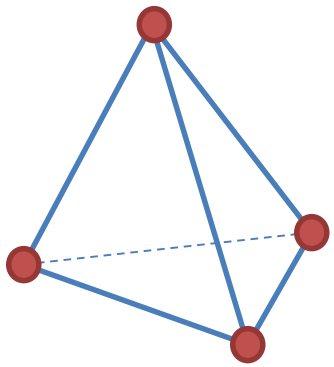
- The *traction pressure* $\mathbf{t}$ is the vector force per unit area on an arbitrary plane with normal $\mathbf{n}$ within the material. It is equal to:

$$\mathbf{t} = \boldsymbol{\sigma} \cdot \mathbf{n}$$

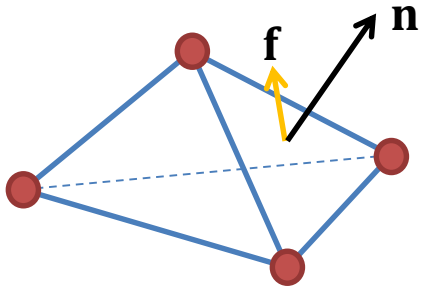
Force on a Tetrahedral Face

- Let's say we have a deformed tetrahedron that has a constant stress tensor $\boldsymbol{\sigma}$ throughout
- To compute the total force on one of the triangular faces, we compute the unit-length normal $\mathbf{n}$ in the material space, then compute the traction force per area $\mathbf{t} = \boldsymbol{\sigma} \cdot \mathbf{n}$, multiply by the total area a of the triangle, and then transform into world space by $\mathbf{F}$ resulting in:

$$\mathbf{f} = \mathbf{F} \cdot a\boldsymbol{\sigma} \cdot \mathbf{n}$$



rest state



deformed state

Force on a Triangular Face

- Say we have a triangular face with three material coordinates $\mathbf{m}_1$, $\mathbf{m}_2$, and $\mathbf{m}_3$, arranged in counter-clockwise order when viewed from the outside of the tetrahedron. The unit-length normal would be:

$$\mathbf{n} = \frac{(\mathbf{m}_2 - \mathbf{m}_1) \times (\mathbf{m}_3 - \mathbf{m}_1)}{|(\mathbf{m}_2 - \mathbf{m}_1) \times (\mathbf{m}_3 - \mathbf{m}_1)|}$$

- And the area would be:

$$a = \frac{|(\mathbf{m}_2 - \mathbf{m}_1) \times (\mathbf{m}_3 - \mathbf{m}_1)|}{2}$$

- Since we're just going to multiply these together as $\mathbf{f} = a\boldsymbol{\sigma} \cdot \mathbf{n}$, we can save some work and use the *area weighted normal* $\mathbf{n}^*$ to compute the total force on the triangle face as:

$$\mathbf{n}^* = \frac{1}{2}(\mathbf{m}_2 - \mathbf{m}_1) \times (\mathbf{m}_3 - \mathbf{m}_1)$$

$$\mathbf{f} = \mathbf{F} \cdot \boldsymbol{\sigma} \cdot \mathbf{n}^*$$

- Note: because $\mathbf{n}$ is always length 1.0, it has no units. The area weighted normal $\mathbf{n}^*$ has units of length^2, or area

Finite vs. Infinitesimal Strain

- The Cauchy stress tensor is intended for small deformations
- The Green strain tensor we looked at earlier is actually meant for moderate deformations
- There are other strain and stress tensors and relationships that are designed for larger deformations
- We will look at these further in the next lecture, but today we'll assume fairly small deformations

Stress-Strain Relationship

Stress Vector

- The 3x3 symmetric stress tensor $\boldsymbol{\sigma}$ has 6 unique numbers

$$\boldsymbol{\sigma} = \begin{bmatrix} \sigma_{xx} & \sigma_{xy} & \sigma_{xz} \\ \sigma_{xy} & \sigma_{yy} & \sigma_{yz} \\ \sigma_{xz} & \sigma_{yz} & \sigma_{zz} \end{bmatrix}$$

- We can rearrange this into a stress vector $[\boldsymbol{\sigma}]$

$$[\boldsymbol{\sigma}] = \begin{bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{zz} \\ \sigma_{yz} \\ \sigma_{xz} \\ \sigma_{xy} \end{bmatrix} = \begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_3 \\ \sigma_4 \\ \sigma_5 \\ \sigma_6 \end{bmatrix}$$

- This is known as *Voigt notation*

*Strain Vector

- The same can be done with the strain tensor $\boldsymbol{\varepsilon}$

$$\boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_{xx} & \varepsilon_{xy} & \varepsilon_{xz} \\ \varepsilon_{xy} & \varepsilon_{yy} & \varepsilon_{yz} \\ \varepsilon_{xz} & \varepsilon_{yz} & \varepsilon_{zz} \end{bmatrix}$$

- We can rearrange this into a strain vector $[\boldsymbol{\varepsilon}]$

$$[\boldsymbol{\varepsilon}] = \begin{bmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \\ 2\varepsilon_{yz} \\ 2\varepsilon_{xz} \\ 2\varepsilon_{xy} \end{bmatrix} = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \varepsilon_4 \\ \varepsilon_5 \\ \varepsilon_6 \end{bmatrix}$$

- In this case, we multiply three of the vector elements by 2 (why?)

Stress-Strain Relationship

- If we assume a linear relationship between the 6 strains and 6 stresses, we could use as many as 36 constants to represent the stress-strain relationship
- This would be expressed as a 6x6 matrix known as a *stiffness tensor* $\mathbf{K}$

$$[\boldsymbol{\sigma}] = \mathbf{K} \cdot [\boldsymbol{\varepsilon}]$$

- It turns out that this matrix will be symmetric, so the number of unique constants is actually 21 for a general linear elastic stress-strain relationship

Stiffness Tensor

- Our generalized Hooke's Law relates stresses $\boldsymbol{\sigma}$ to strains $\boldsymbol{\varepsilon}$:

$$[\boldsymbol{\sigma}] = \mathbf{K} \cdot [\boldsymbol{\varepsilon}]$$

- Expanded out, we can see the general form of the stiffness tensor $\mathbf{K}$ has 21 unique constants:

$$\begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_3 \\ \sigma_4 \\ \sigma_5 \\ \sigma_6 \end{bmatrix} = \begin{bmatrix} k_{11} & k_{12} & k_{13} & k_{14} & k_{15} & k_{16} \\ k_{12} & k_{22} & k_{23} & k_{24} & k_{25} & k_{26} \\ k_{13} & k_{23} & k_{33} & k_{34} & k_{35} & k_{36} \\ k_{14} & k_{24} & k_{34} & k_{44} & k_{45} & k_{46} \\ k_{15} & k_{25} & k_{35} & k_{45} & k_{55} & k_{56} \\ k_{16} & k_{26} & k_{36} & k_{46} & k_{56} & k_{66} \end{bmatrix} \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \varepsilon_4 \\ \varepsilon_5 \\ \varepsilon_6 \end{bmatrix}$$

Isotropic Materials

- An *isotropic* material is a material whose properties are invariant with respect to rotation
- In other words, the material does not have any inherent orientation
- For example, metals, plastic, concrete, and glass are generally isotropic
- Wood, however, is *anisotropic* in that it has a grain direction, and so the orientation of the grain will affect the material properties
- Crystals are also anisotropic as are some construction materials like fiberglass and carbon fiber

Anisotropic Materials

- A general anisotropic material could have a full 21 unique constants in the stress-strain relationship
- There are several sub-groups of these however, with fewer constants:
  - Orthotropic: these have 3 orthogonal planes of symmetry and requires 9 unique constants
  - Transverse isotropic: these are isotropic around a single axis, like a parallel bundle of fibers and requires 5 unique constants
- Isotropic materials require only 2 unique constants

Isotropic Stiffness Tensor

- The isotropic version of the stiffness tensor looks like this:

$$\mathbf{K} = \begin{bmatrix} 2\mu+\lambda & \lambda & \lambda & 0 & 0 & 0 \\ \lambda & 2\mu+\lambda & \lambda & 0 & 0 & 0 \\ \lambda & \lambda & 2\mu+\lambda & 0 & 0 & 0 \\ 0 & 0 & 0 & \mu & 0 & 0 \\ 0 & 0 & 0 & 0 & \mu & 0 \\ 0 & 0 & 0 & 0 & 0 & \mu \end{bmatrix}$$

- Where μ and λ are the *Lamé constants*

Isotropic Material Properties

- We saw how to build the K matrix based on the isotropic Lamé constants μ and λ, however these constants don't have a very intuitive meaning
- We can rearrange some of the equations to base these on some more intuitive constants:
  - Young's modulus E: This is similar to the original 1D spring constant *k* in that it relates the uniaxial stress σ to the strain ε with: $\sigma = E\varepsilon$
  - Poisson ratio ν: This is a measure of how much the material resists changes in volume and ranges from -1 to 0.5 (usually from 0 to 0.5). A perfectly incompressible material will have a $\nu = 0.5$ and a perfectly compressible material will have $\nu = 0$
- To compute the Lamé constants from the Young's modulus and Poisson ratio:

$$\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)}$$

$$\mu = \frac{E}{2(1+\nu)}$$

Isotropic Stress-Strain Relationship

- If we are using isotropic materials, we can actually represent the stress-strain relationship a little easier as:

$$\boldsymbol{\sigma} = 2\mu\boldsymbol{\varepsilon} + \lambda \mathrm{trace}(\boldsymbol{\varepsilon})\mathbf{I}$$

- Where $\mathrm{trace}(\boldsymbol{\varepsilon})$ is the sum of the diagonal elements of $\boldsymbol{\varepsilon}$
- This skips the Voigt notation and the 6x6 matrix

Materials

- The stiffness tensor $\mathbf{K}$ can be precomputed and stored for each unique material in the system
- There are several different ways to construct the $\mathbf{K}$ matrix depending on the material properties
- For this and other reasons, it is nice to have a material class that stores all material properties (like density and the Lamé, Young, or Poisson constants) and also stores a precomputed $\mathbf{K}$ matrix
- Individual elements can reference a material instead of storing the properties directly
- This also makes it easier to add a user interface for tuning material properties
- Also, as one adds more complex physics such as plasticity and fracture, additional properties can be added to the material

Stress Computation Summary

- To summarize the isotropic stress computation:
  1. Compute stress tensor using:

$$\boldsymbol{\sigma} = 2\mu\boldsymbol{\varepsilon} + \lambda \mathrm{trace}(\boldsymbol{\varepsilon})\mathbf{I}$$

- To summarize the anisotropic stress computation:
  1. Arrange the strain tensor $\boldsymbol{\varepsilon}$ into vector (Voigt) form $[\boldsymbol{\varepsilon}]$
  2. Compute the stress vector $[\boldsymbol{\sigma}]$ using:

$$[\boldsymbol{\sigma}] = \mathbf{K} \cdot [\boldsymbol{\varepsilon}]$$

  3. Arrange vector $[\boldsymbol{\sigma}]$ into matrix form $\boldsymbol{\sigma}$

Node Force Computation

Force Computation

- For each tetrahedron in our deformable object, we need to:
  1. Compute the strain tensor
  2. Compute the stress tensor
  3. Compute the forces on the nodes
- So far, we covered 1 & 2 in detail

Node Forces

- Once we have the stress tensor, we can proceed to computing the forces on the nodes (particles)
- For small deformations, this is actually pretty easy
- We already showed how to compute the total force on a triangular face of a tetrahedron
- By Newton's Third Law, if this force is distributed to the three nodes making up the triangle, then the equal and opposite force must apply to the fourth node, (the one that isn't on the triangle)

Node Force Computation

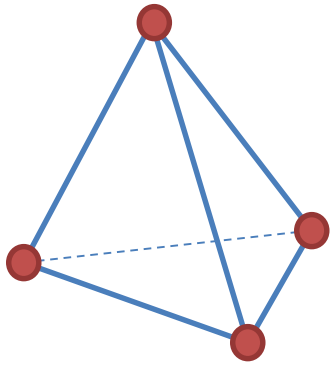
- We saw how to compute the total force on the triangle face as:

$$\mathbf{n}^* = \frac{1}{2}(\mathbf{m}_2 - \mathbf{m}_1) \times (\mathbf{m}_3 - \mathbf{m}_1)$$

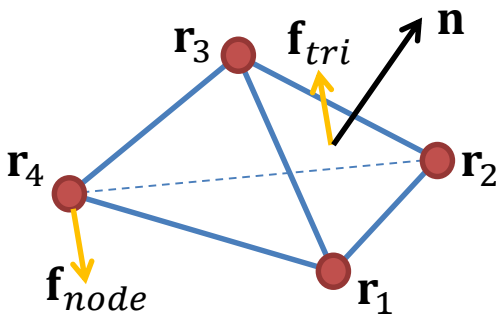
$$\mathbf{f}_{tri} = \mathbf{F} \cdot \boldsymbol{\sigma} \cdot \mathbf{n}^*$$

- Then the force on the opposite node $\mathbf{r}_4$ is therefore:

$$\mathbf{f}_{node} = -\mathbf{F} \cdot \boldsymbol{\sigma} \cdot \mathbf{n}^*$$



rest state



deformed state

Node Force Summary

- To summarize the node force computation for a tetrahedron:
  - For each of the four nodes:
    1. Select $\mathbf{m}_1'$, $\mathbf{m}_2'$, and $\mathbf{m}_3'$ from the original $\mathbf{m}_1$-$\mathbf{m}_4$ as the three nodes on the opposite triangle face arranged in counter-clockwise order
    2. Compute node force:

$$\mathbf{n}^* = \frac{1}{2}(\mathbf{m}_2' - \mathbf{m}_1') \times (\mathbf{m}_3' - \mathbf{m}_1')$$

$$\mathbf{f}_{node} = -\mathbf{F} \cdot \boldsymbol{\sigma} \cdot \mathbf{n}^*$$

    3. Apply (add) force $\mathbf{f}_{node}$ to the particle associated with the node

- Note: $\mathbf{n}^*$ can be precomputed for each face
- Note: this method is only valid for small deformations, where $\mathbf{F}$ is nearly orthonormal. Larger deformations will require some modifications to this

Elastic Body Simulation

Elastic Simulation

- We've now seen the whole process of how to compute the forces acting on the nodes of a deformed tetrahedron
- If our model connects up many tetrahedra, we just do this for each one, adding up all the forces, and then integrate the motion of the particles using the integrator of our choice
- There are still a few more details to cover regarding the initialization of the model

Particle Masses

- Let's assume that each tetrahedral element is made a material with density ρ (kg/m^3)
- The mass of an element would be the volume of the undeformed tetrahedron times ρ
- As there are 4 particles to a tetrahedron, this mass should be distributed equally to the 4 particles
- To initialize all particle masses in the entire model, we first set them to 0. We then loop through all elements. We compute the element mass and add ¼ to each of its 4 particles. When we finish looping through the elements, all particle masses in the model will be set
- While we're looping through the elements, we can also precompute and store $\mathbf{R}^{-1}$ for each one, as well as the area weighted normal $\mathbf{n}^*$ for each face

Summary

- Here is a summary of the process. Note, this uses a simple built-in forward Euler integration. To adapt to more complex integrators, separate out the force computation and use a more general integration architecture

- Initialize
  - Load/generate mesh & material properties
  - Precompute element rest properties (volume, $\mathbf{R}^{-1}$, $\mathbf{n}^{*}$)
  - Initialize particle masses
  - Set initial particle positions & velocities
- Simulate
  - While(not finished)
    - ComputeForces() {
      - For each element
        - » Compute strain tensor
        - » Compute stress tensor
        - » Compute and apply node forces
    - IntegrateForwardEuler()

Limits of Linear Elasticity

- The linear elastic model we looked at today works OK as long as we minimize the deformation of any one tetrahedron
- It doesn't handle large deformations very well, but we can still get some nice behaviors with the model
- As mentioned in the beginning, it leaves out a lot of interesting material behaviors (plasticity, fatigue, fracture, thermodynamics, etc.)
- We also haven't talked about collisions
- But it is a great place to start and a foundation for more complex modeling