Basics of deformation of solids for computer animation

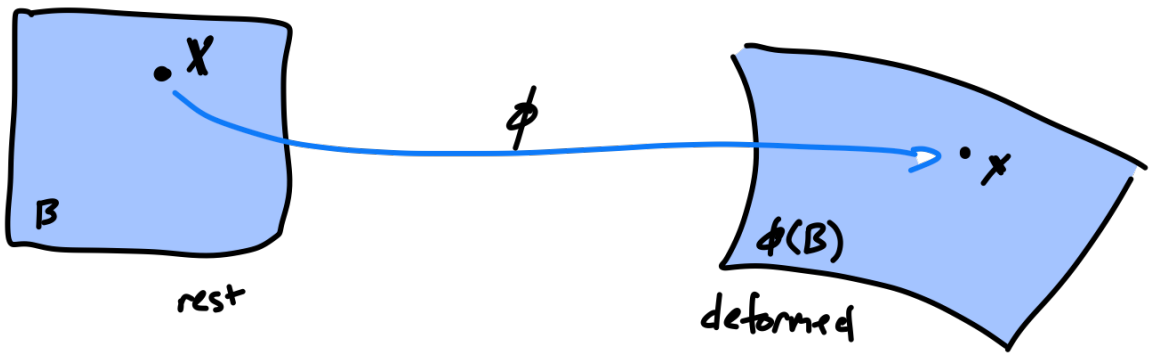
Steve Marschner
CS 5643 Spring 2025
Cornell University

Graphics is often concerned with simulating deformable objects, like blobs of Jell-O or (more often) the flesh of an animated character. It's a challenging problem about which a lot has been written, and this can be a really confusing part of the graphics literature to wade into because the ideas used in animation draw from and connect so many different fields that also model deformation, and there are many different ways of discussing deformation and the forces that arise from it. It's a field with a century of history and all the layers of notation and terminology that go with that.

In this course we will just scratch the surface, but we'll try to put together the basic tools needed to define and simulate some well-behaved deformation models.

Describing deformation

When a solid object deforms, the material it is made of moves through space, and what makes this deformation instead of just separate motions of different pieces is that it undergoes a *continuous* deformation: nearby points at one time remain nearby under deformation. Mathematically, there is a function $\mathbf{x} = \phi(\mathbf{X}, t)$ that takes in a *material point* $\mathbf{X}$, which identifies a little chunk of material in the *rest configuration* B of the object, and gives back the position $\mathbf{x}$ of the same piece of material at time t.



We assume the object is "at rest" in its rest configuration, meaning that it it will stay in that shape if no forces are applied to it, and it "wants" to return to that state, so that changes in shape should be referenced to the rest configuration. The function ϕ is the *deformation map* and it is continuous (as

long as the material is not tearing) and at least piecewise differentiable. We'll call the space where $\mathbf{X}$ lives *material space* and the space where $\mathbf{x}$ lives *world space*.

Measuring deformation

When we say that an object deforms, we usually mean it changes shape and/or size—that is, material points get farther apart or closer together. The change in shape in a small local region can be nicely summarized using the derivative of the deformation map, known as the *deformation gradient*:

$$\mathbf{F} = \frac{\partial \phi}{\partial \mathbf{X}} = \frac{\partial \mathbf{x}}{\partial \mathbf{X}}$$

Being the derivative of a vector with respect to a vector, $\mathbf{F}$ is a 2-tensor, also known as a matrix: 2x2 in a 2D simulation, or 3x3 in a 3D simulation. This matrix tells us all we need to know about what is happening locally in the material to simulate its response to deformation, and it is the starting point for a wide range of models for deformation.

The measures that we build from $\mathbf{F}$ are measures of *strain*—material deformation considered locally, in a way that is independent of rigid motion.

Infinitesimal vs. finite deformation and strain

In some cases objects deform a lot; in other cases just a little. A concrete column in a building or the steel body of a car deform just a little; if they change dimensions by more than a fraction of a percent things have probably gone very wrong. In this situation we have *infinitesimal* deformations (aka displacements) and *infinitesimal* strains, or sometimes just "small deformations" and "small strains." You can make accurate predictions by linearizing everything around the rest configuration, leading to the field of *linear elasticity*.

In computer animation we are interested in deformations we can see, so we generally are in the business of finite displacements. A cube of Jell-O, a rubber chicken, the muscles in your arm, or many other objects that might make an appearance in an animation are all things that deform a lot; they might easily change dimensions by a factor of 2 or more. This regime is called *finite strain* and *finite deformation*.

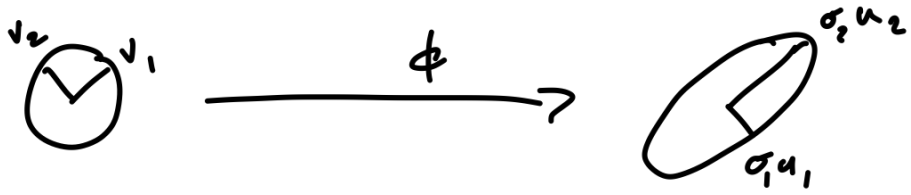
It's worth noting that the ideas of finite displacements and finite strains can be applied separately. Objects that are very thin, like a strand of hair or a sheet of plastic, can undergo large deformations while still undergoing very small strain—so it is not unreasonable to use an assumption of infinitesimal strain in such a case while at the same time working with large displacements.

Once you lose the assumption of infinitesimal displacements, you are in the business of *nonlinear elasticity*.

Rotation invariance

When trying to see into what a matrix "does" it is often instructive to think about its SVD, so let's try that with $\mathbf{F}$, writing the 2D case:

$$\mathbf{F} = \mathbf{U}\Sigma\mathbf{V}^T = \begin{bmatrix}\mathbf{u}_1 & \mathbf{u}_2\end{bmatrix}\begin{bmatrix}\sigma_1 & 0\\ 0 & \sigma_2\end{bmatrix}\begin{bmatrix}\mathbf{v}_1 & \mathbf{v}_2\end{bmatrix}^T = \mathbf{R}_{\text{world}}\begin{bmatrix}\sigma_1 & 0\\ 0 & \sigma_2\end{bmatrix}\mathbf{R}_{\text{material}}$$



This says that if we appropriately rotate our coordinate systems in the reference space and the deformed space, we can think of the local deformation of a "blob" of material as just a nonuniform scaling: it stretches (or compresses) by the ratio σ_1 along one axis and by the ratio σ_2 along a perpendicular axis. In this equation the rotation $\mathbf{R}_{\text{material}}$ tells us how these *principal axes* are oriented relative to the material, and $\mathbf{R}_{\text{world}}$ tells us how they are oriented in the world. (I am making some assumptions when I claim that the two orthogonal matrices are rotations, not reflections, but I'll promise this is easy to fix and sweep it under the carpet for now.)

This decomposition is useful because there are some "deformations" we would expect the motion to be invariant to. For instance, rotating the whole object rigidly in the global space would not be expected to change anything about how it behaves; its behavior should be *rotation invariant* in world space, and for this reason we would expect material behavior to be invariant to $\mathbf{R}_{\text{world}}$.

Many materials can at least be approximated as *isotropic*, meaning that the material doesn't have any oriented structure that would make it stronger or weaker or otherwise different when stretched or compressed along different directions. Materials like rubber, plastic, many types of metal or glass, etc. can often be assumed isotropic since their molecular structure is randomly oriented; other materials like wood, fiber composites, etc. are strongly anisotropic. If a material is isotropic we also expect its behavior to be invariant to $\mathbf{R}_{\text{material}}$, which greatly simplifies the task of modeling it, and we'll stick to this case.

In fact, anything we compute from $\mathbf{F}$ that is supposed to be both rotation-invariant and isotropic can *only* be a function of the two (or three) singular values of $\mathbf{F}$! So that means many famous quantities with fancy names all really boil down to functions of two or three variables.

Hyperelastic materials and strain energy density

When a material deforms it can behave *elastically*, meaning that nothing changes about the material and it will return back to its original shape once all loads are removed, or *plastically*, meaning that the deformation causes permanent changes to the material so that it will not return back to the same shape or to the same material properties. Elastic behavior is much simpler to model, and most solid materials will behave elastically at least for small deformations, so it is the starting point (and in computer graphics, often also the ending point!) for deformable models.

A material that behaves elastically for all deformations is called *hyperelastic*. I think "hyper" here just means "more elastic than any real material" but it seems to me we should just call a material like this "elastic." In any case, a hyperelastic material can always be described using a potential energy known as *strain energy* which plays a similar role to the spring potential energy we used in mass-spring systems. The strain energy is a function of the entire deformation map ϕ, though because all interactions between bits of material are local (each material point can only "see" its infinitesimal neighborhood) the strain energy is a function only of the deformation gradient $\mathbf{F}$. If the material is homogeneous (another handy assumption we will always make), then it is an integral of some function $\psi(\mathbf{F})$ over the rest configuration:

$$E[\phi] = \int_B \psi(\mathbf{F}(\mathbf{X}))\, d\,\mathbf{X}$$

The function ψ is known as the *strain energy density*, and defining a model for a hyperelastic material boils down to cooking up a model for ψ.

Because of the rotation invariance in both material and world space that we expect from isotropic materials, ψ should not depend on $\mathbf{R}_{\text{material}}$ or $\mathbf{R}_{\text{world}}$ in the decomposition we wrote above; it is a function only of the singular values σ_i. The classical game of defining material models then is to devise functions of σ_1 and σ_2 (and σ_3 if we are in 3D) that are easy to work with analytically and also resemble the behavior of some real materials.

Aside: linear algebra and differentiation techniques

Matrices

Before we get into deriving nodal forces from these strain energies, I want to digress on notation so we have some tools available when we get to the derivative computations.

We are working with matrices that represent linear transformations in 3D space, and we'll brush up against some generalizations of this idea as we compute derivatives.

As practical-minded computer scientists, perhaps our most familiar way to think of a matrix is as a 2D array of numbers—after all that is what you will find in the code. You can also see this array as a 1D array of vectors in two ways: as a bunch of columns or as a bunch of rows. This is one perfectly good view of what a matrix is, but it's good to keep a couple other interpretations in mind.

A matrix also represents a linear transformation on vectors via matrix multiplication; if $\mathbf{A} \in \mathbb{R}^{m\times n}$ then the transformation is $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m : \mathbf{f}(\mathbf{x}) = \mathbf{A}\mathbf{x}$. In terms of matrix entries you can write $\mathbf{y} = \mathbf{A}\mathbf{x}$ in a couple of ways:

$y_i = \sum_j a_{ij} x_j = \mathbf{r}_i \cdot \mathbf{x}$ — each entry of $\mathbf{y}$ is a dot product of $\mathbf{x}$ with one row

$\mathbf{y} = \sum_j x_j \mathbf{c}_j$ — each entry of $\mathbf{x}$ is a coefficient in a linear combination of the columns

The row-wise mindset makes it clear that anything that is perpendicular to all the rows will be sent to zero by $\mathbf{f}$. The set of vectors with this property is the null space of $\mathbf{A}$.

The column-wise mindset makes it clear that only vectors that are in the span of the columns can be output by $\mathbf{f}$.

It's a theorem that the span of the rows and the span of the columns have the same dimension, and that dimension is known as the *rank* of the matrix. The rank clearly can't be more than $\min(m, n)$, since the n columns can't span more than n dimensions and likewise for the rows. If the matrix has the largest possible rank—that is, its rank is $\min(m, n)$—then we call the matrix *full rank*, and if it's less than full rank we use the slightly judgmental term "rank deficient" or the more neutral "singular."

A matrix also represents a scalar-valued function of two vectors via $f(\mathbf{v}, \mathbf{w}) = \mathbf{v}^T\mathbf{A}\mathbf{w}$. This is a *bilinear* function because once you fix one argument, it's a linear function of the other argument.

In functional programming one often thinks of a function of two arguments (e.g. $f : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$) as a function that takes one argument and returns a new function one argument (e.g. $f : \mathbb{R} \to (\mathbb{R} \to \mathbb{R})$), and this idea also connects the notions of a matrix as a bilinear function or as a linear transformation. A vector can be considered a function $f : \mathbb{R}^n \to \mathbb{R} : f(\mathbf{w}) = \mathbf{v} \cdot \mathbf{w}$.

Products

With vectors we have a dot product which in the array view means "multiply and sum."

$$\mathbf{v} \cdot \mathbf{w} = \sum_i v_i w_i$$

One can see matrix multiplication as a generalization of this idea: to compute $\mathbf{A}\mathbf{B}$, identify the second (last) axis of $\mathbf{A}$ with the first axis of $\mathbf{B}$ and sum over that axis. This idea also generalizes to arrays with as many dimensions as you like:

both arguments 2D: $\left(\mathbf{A} \cdot \mathbf{B}\right)_{ij} = \sum_k a_{ik} b_{kj}$

both arguments 3D: $\left(\mathbf{A} \cdot \mathbf{B}\right)_{ijlm} = \sum_k a_{ijk} b_{klm}$

left argument 3D, right argument 1D: $\left(\mathbf{A} \cdot \mathbf{v}\right)_{ij} = \sum_k a_{ijk} v_k$

We can write matrix-matrix products with a dot if we like:

$\mathbf{A} \cdot \mathbf{B} = \mathbf{A}\mathbf{B}$; $\mathbf{A} \cdot \mathbf{w} = \mathbf{A}\mathbf{w}$; $\mathbf{v} \cdot \mathbf{A} = \mathbf{v}^T\mathbf{A}$; $\mathbf{v} \cdot \mathbf{w} = \mathbf{v}^T\mathbf{w}$.

With the matrix product notation we have to transpose the vector when it appears on the left because we insist on thinking of it as a 2D array with its second dimension equal to 1, rather than an array with just one dimension. The @ operator in NumPy works like this dot product when combining arrays of various dimensions.

One other type of product that is useful is a "double contraction," which is the same idea but with a double loop. If I have two multidimensional arrays $\mathbf{A}$ and $\mathbf{B}$, then to compute $\mathbf{A} : \mathbf{B}$, identify the last two axes of $\mathbf{A}$ with the first two axes of $\mathbf{B}$ and sum products over both of them. For some examples:

both arguments 2D: $\mathbf{A} : \mathbf{B} = \sum_{i,j} a_{ij} b_{ij}$

both arguments 3D: $\left(\mathbf{A} : \mathbf{B}\right)_{kl} = \sum_{i,j} a_{kij} b_{ijl}$

A handy identity for double contractions is

$$\mathbf{A} : \mathbf{BC} = \mathbf{B}^T\mathbf{A} : \mathbf{C} = \mathbf{AC}^T : \mathbf{B} \quad \text{(proof on homework)}$$

One place this figures prominently is in working with the Frobenius norm for matrices. You'll recall that the standard Euclidean norm (2-norm) of a vector can be defined

$$\|\mathbf{v}\|_2^2 = \mathbf{v} \cdot \mathbf{v} = \sum_k v_k^2$$

The Frobenius norm, which is computed in the same way by summing the squares of all the entries in a matrix, can be written analogously:

$$\|\mathbf{A}\|_F^2 = \mathbf{A} : \mathbf{A} = \sum_{i,j} a_{ij}^2$$

In both cases we'll see writing norms this way is helpful in computing their derivatives.

A couple other things to say about the Frobenius norm. In the same way that rotating a vector does not change its length, multiplying a matrix by an orthogonal matrix does not change its Frobenius norm. You can see this by thinking of $\mathbf{A}$ as a stack of column vectors:

$$\begin{aligned} \mathbf{A} &= [\mathbf{v}_1 \cdots \mathbf{v}_n] \\ \mathbf{QA} &= \left[\mathbf{Qv}_1 \cdots \mathbf{Qv}_n\right] \\ \|\mathbf{A}\|_F^2 &= \sum_k \|\mathbf{v}\|_2^2 = \sum_k \|\mathbf{Qv}\|_2^2 \end{aligned}$$

and a similar argument with row vectors works if the orthogonal matrix is on the right. The Frobenius norm is perhaps the best-behaved, simplest-to-compute matrix norm, so it often appears as a first pick when you want to measure the "size" of a matrix and want a smooth answer.

Matrix Decompositions

When you are trying to figure out what a matrix "does," there are a couple of great tools available in terms of decomposing a matrix as a product of simpler matrices. The first is the symmetric eigenvalue decomposition. A real-valued symmetric matrix always has a full set of real-valued eigenvalues and eigenvectors, and the eigenvectors are always orthogonal. These nice facts can be expressed as a decomposition theorem: for any symmetric matrix $\mathbf{S}$, there is an orthogonal matrix $\mathbf{V}$ (containing the eigenvectors of $\mathbf{S}$ as its columns) and a diagonal matrix $\mathbf{D}$ (containing the eigenvalues as its diagonal entries) such that

$$\mathbf{S} = \mathbf{V}\mathbf{D}\mathbf{V}^T \quad \text{—or, said differently:} \quad \mathbf{V}^T\mathbf{S}\mathbf{V} = \mathbf{D}$$

You can read this as "to multiply a vector $\mathbf{v}$ by $\mathbf{S}$, first rotate the vector into a coordinate system aligned with the eigenvectors, then scale the coordinates independently, then rotate back" or "If you choose the eigenvectors as your basis vectors when you write the matrix of your linear transformation, you will find it is diagonal."

There is a generalization of this idea for nonsymmetric matrices in terms of complex-valued eigenvalues and eigenvectors, which is a great tool in some situations. But in many cases an easier generalization to work with is the Singular Value Decomposition (SVD), which is just like the symmetric eigenvalue decomposition except that the two orthogonal matrices are not the same:

$$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^T \quad \text{—or, said differently:} \quad \mathbf{U}^T\mathbf{A}\mathbf{V} = \Sigma$$

You can read this as "to multiply a vector $\mathbf{v}$ by $\mathbf{A}$, first rotate the vector into a coordinate system aligned with the right singular vectors, then scale the coordinates independently; what you get is the result of the transformation in coordinates of the left singular vectors, so rotate back out of that coordinate system" or "If you choose the right singular vectors as your basis vectors for the *input* to $\mathbf{A}$ and the left singular vectors as your basis for the *output* of $\mathbf{A}$, then when you write the matrix of your linear transformation, you will find it is diagonal."

Derivatives

When we differentiate a function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$ the result is a linear transformation from $\mathbb{R}^n$ to $\mathbb{R}^m$, which as an array of numbers we could describe as the matrix of partial derivatives of $\mathbf{f}$. That is,

$$\left(\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right)_{ij} = \frac{\partial f_i}{\partial x_j}$$

This matrix can also be interpreted as a linear transformation, but what does the transformation do? One answer is that it takes a vector representing an infinitesimal change in the input and maps it to the corresponding differential change in the output:

$$\delta\mathbf{f} = \left(\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right)\delta\mathbf{x}$$

A more solid way to say this is that the derivative maps a vector $\delta\mathbf{x}$ to the directional derivative of $\mathbf{f}$ in the direction $\delta\mathbf{x}$—it answers the question "if $\mathbf{x}$ is moving with velocity $\delta\mathbf{x}$, how fast is $\mathbf{f}(\mathbf{x})$ changing?"

This notion of a derivative as a linear transformation makes for a useful way to write down derivatives; for instance if $\mathbf{f}(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}$ then we could write $\delta\mathbf{f} = \mathbf{A}\delta\mathbf{x}$ as an alternative to $\partial\mathbf{f}/\partial\mathbf{x} = \mathbf{A}$. Sometimes it is easier to write down an expression for the result of applying the derivative to a vector than it is to write down an expression for the derivative matrix.

Some handy differentiation techniques using this notation: Differentiating a dot product:

$$\delta[\mathbf{v} \cdot \mathbf{w}] = \delta[\mathbf{v}] \cdot \mathbf{w} + \mathbf{v} \cdot \delta[\mathbf{w}]$$

$$\delta[\|\mathbf{v}\|_2^2] = \delta[\mathbf{v} \cdot \mathbf{v}] = 2\mathbf{v} \cdot \delta[\mathbf{v}] \quad \text{(can combine terms because the dot product is commutative)}$$

Differentiating a product of two matrices goes like this:

$$\delta[\mathbf{AB}] = \delta[\mathbf{A}]\,\mathbf{B} + \mathbf{A}\,\delta[\mathbf{B}]$$

This looks just like the product rule for scalars, but note the order of multiplication needs to be preserved. It's worth unpacking this a bit. This is talking about the product AB as a function of the matrices A and B. The product changes when the matrices change. And this formula tells us the rate of change of the product in terms of the rates of change of the matrices. It's a linear transformation from matrices to matrices, which is awkward to write down in general, but here it is simple to write the result as a function of the inputs.

This same product rule also works for double contractions:

$$\delta[\mathbf{A} : \mathbf{B}] = \delta[\mathbf{A}] : \mathbf{B} + \mathbf{A} : \delta[\mathbf{B}]$$

...except here the result is a scalar (if $\mathbf{A}$ and $\mathbf{B}$ are matrices), so we can swap the order if we want, which makes it neat to write the derivative of the Frobenius norm:

$$\delta[\|\mathbf{A}\|_F^2] = \delta[\mathbf{A} : \mathbf{A}] = 2\mathbf{A} : \delta\mathbf{A}$$

A useful identity about derivatives and determinants:

$$\delta[\det \mathbf{A}] = (\det \mathbf{A})\,\mathbf{A}^{-T} : \delta\mathbf{A}$$

This (or a slightly different version that also works for singular $\mathbf{A}$) is known as Jacobi's formula.

Measuring strain

The name for a local measure of deformation is *strain*. For nonlinear elasticity (finite displacements) we want strain to be invariant to rotation in the world. Let me show you two ways you can use simple matrix math to build strain measures. The very first goal here is to write some function of $\mathbf{F}$ that is not going to be too hard to compute with and is invariant to rotations in world space. That is, in the decomposition $\mathbf{F} = \mathbf{U}\Sigma\mathbf{V}^T$ we want to compute things which do not depend on $\mathbf{U}$.

Green's strain

We want to work with simple and cheap matrix operations, so one idea you might think of is to cancel out the orthogonal matrix $\mathbf{U}$ by multiplying it with its transpose. We can make this happen by using the product $\mathbf{F}^T\mathbf{F}$; expanding this using the SVD of $\mathbf{F}$ leads to:

$$\mathbf{F}^T\mathbf{F} = (\mathbf{V}\Sigma\mathbf{U}^T)(\mathbf{U}\Sigma\mathbf{V}^T) = \mathbf{V}\Sigma^2\mathbf{V}^T$$

Since $\mathbf{U}$ is gone, this means $\mathbf{F}^T\mathbf{F}$ is invariant to world-space rotation. This matrix gets the name "right Cauchy-Green deformation tensor" and it is the starting point for many deformation models. When there is no stretching at all, just rotation in world space, we'll find that $\mathbf{F}^T\mathbf{F} = \mathbf{I}$. This gives rise to the idea of measuring the amount of deformation as the difference between $\mathbf{F}^T\mathbf{F}$ and the identity:

$$\text{Green's strain tensor: } \mathbf{E} = \frac{1}{2}\left(\mathbf{F}^T\mathbf{F} - \mathbf{I}\right)$$

and this tensor is the classical way to measure strain in the finite-deformation setting. Some kind of squared norm of this tensor $\mathbf{E}$ seems like a good candidate for defining a strain energy density. Something like

$$\psi_G = \tfrac{1}{2}\|\mathbf{E}\|_F^2.$$

This idea forms the basis for the well known St. Venant–Kirchhoff (StVK) constitutive model, which we'll come back to in a minute.

Strain via polar decomposition

Another way to get rid of the rotation is more explicit: numerically factor $\mathbf{F}$ to pull out the world-space rotation. This idea is known as the polar decomposition and it can be readily computed from the SVD:

$$\mathbf{F} = \mathbf{RS} = (\mathbf{U}\mathbf{V}^T)(\mathbf{V}\Sigma\mathbf{V}^T)$$

(with some easy-to-fix brushing of signs under the rug to assume $\mathbf{R}$ is a rotation.) This says that we can always write $\mathbf{F}$ (or any other matrix) in the form of a rotation times a symmetric matrix, and when we do this we separate the ideas of how much the material is rotating from how much it is changing dimensions. For measuring strain we can ignore $\mathbf{R}$ and focus on $\mathbf{S}$. If $\mathbf{F}$ is a rotation then $\mathbf{S} = \mathbf{I}$, so applying the same idea as with Green's strain we can measure stretching in terms of the difference between $\mathbf{S}$ and $\mathbf{I}$:

$$\text{Corotational strain tensor: } \boldsymbol{\epsilon}_c = \mathbf{S} - \mathbf{I}$$

and define an energy density:

$$\psi_{\text{ARAP}} = \|\boldsymbol{\epsilon}_c\|_F^2$$

In graphics this is energy is called the As Rigid As Possible (ARAP) energy because it measures the difference between $\mathbf{F}$ and the nearest rigid motion, which is $\mathbf{R}$ (since the Frobenius norm is invariant to rotation, $\|\mathbf{F} - \mathbf{R}\|_F = \|\mathbf{R}^T(\mathbf{F} - \mathbf{R})\|_F = \|\mathbf{S} - \mathbf{I}\|_F$), and by minimizing it you can find a deformation which is as close as possible to being rigid. This energy is the first part of the corotational linear elasticity model we'll describe in a minute.

Linear strain

We won't really use infinitesimal-displacement models for animation, but it's useful to write down the strain measure that is used for linear elasticity. If you take the Green's strain and compute its first-order Taylor expansion with respect to $\mathbf{F}$ around the identity, you have

$$\mathbf{E}(\mathbf{F}) \approx \mathbf{E}(\mathbf{I}) + \delta\,\mathbf{E}(\mathbf{I})\,|_{\delta\mathbf{F}=\mathbf{F}-\mathbf{I}} + \dots$$

The first term is zero; working with the other one,

$$\begin{aligned}\delta\,\mathbf{E}(\mathbf{F}) &= \frac{1}{2}\delta[\mathbf{F}^T\mathbf{F} - \mathbf{I}] \\ &= \frac{1}{2}\left(\delta[\mathbf{F}]^T\mathbf{F} + \mathbf{F}^T\delta[\mathbf{F}]\right)\end{aligned}$$

evaluating this differential at the identity,

$$\begin{aligned}\delta\mathbf{E}(\mathbf{I}) &\approx \frac{1}{2}\left(\delta[\mathbf{F}]^T\mathbf{I} + \mathbf{I}^T\delta[\mathbf{F}]\right)\\ &= \frac{1}{2}\left(\delta[\mathbf{F}]^T + \delta[\mathbf{F}]\right)\end{aligned}$$

and plugging into the expansion,

$$\begin{aligned}\mathbf{E}(\mathbf{F}) \approx \delta\mathbf{E}(\mathbf{I})|_{\delta\mathbf{F}=\mathbf{F}-\mathbf{I}} &= \frac{1}{2}\left((\mathbf{F}-\mathbf{I})^T + (\mathbf{F}-\mathbf{I})\right)\\ &= \frac{1}{2}\left(\mathbf{F}+\mathbf{F}^T\right) - \mathbf{I}\end{aligned}$$

Using this approximation to compute strain, you wind up with

$$\text{Infinitesimal strain tensor: } \boldsymbol{\epsilon} = \frac{1}{2}\left(\mathbf{F}+\mathbf{F}^T\right) - \mathbf{I}$$

This strain measure is *not* rotation invariant, but it matches the Green's strain for very small displacements.

Some basic constitutive models

A *constitutive model* is a model that connects the strain that a material undergoes to the forces that push the material back towards its rest shape. In the continuum setting the restorative force is *stress*—a tensor that describes the continuous distribution of internal forces within the material, and the constitutive model provides a way to compute stress from strain. Since we will always be working with discretized models, we can view the stress as a quantity that appears in computing the nodal forces on the vertices of our mesh. In either case, the way to build these models is to start with an energy and differentiate it to learn about forces.

To measure stress we will use the derivative of the strain energy density ψ with respect to the deformation gradient $\mathbf{F}$:

$$\mathbf{P}(\mathbf{F}) = \frac{\partial\psi(\mathbf{F})}{\partial\mathbf{F}}$$

This tensor is known as the *first Piola-Kirchhoff stress tensor*, hence the name $\mathbf{P}$. For any constitutive model we need to work out expressions we can evaluate to compute both ψ and $\mathbf{P}$. We will work through three of these that all have quite a similar form and are built on the three strain measures we just defined.

Linear elasticity

For the case of infinitesimal strains the relationship between stress and strain can be modeled as linear. The requirements of isotropy and coordinate independence bring the range of possible linear models down to just two parameters—so we just need two numbers to describe any isotropic linear elastic material. Different fields use different pairs of parameters; confusing but fundamentally all that matters

is agreeing on the definitions of the ones you use! We'll use two pairs, one that is conventional for describing the material in the input file and one that makes the equations nice.

For describing materials we use Young's modulus E and Poisson's ratio ν. These can be described as the results of a simple experiment where we stretch a bar of material from length 1 to length $1+\epsilon$. The force required to do this is proportional to the area and Young's modulus is the ratio: $f = EA$. This means E has units of force per unit area, so you see it quoted with pressure units like Pascals. When you stretch the bar it also generally changes its width, in most cases getting narrower as you stretch it out. Poisson's ratio is the ratio of the transverse contraction to the lengthwise extension in this experiment. A material like foam may compress without bulging and have $\nu \approx 0$. A material like muscle, containing lots of nearly incompressible water, might maintain its volume almost exactly, so that in 2D $\nu \approx 1$ or in 3D $\nu \approx 0.5$ (since in 3D there are two transverse widths that change in the same way). Some exotic materials actually get wider as you stretch them, and therefore have $\nu < 0$.

Because E and ν have nice concrete interpretations, usually we use them to talk about material properties. But when we write down the constitutive relations we use other parameters. In particular, for a linear isotropic material the strain energy density is

$$\psi(\mathbf{F}) = \mu \|\boldsymbol{\epsilon}\|_F^2 + \frac{\lambda}{2}(\operatorname{tr}\boldsymbol{\epsilon})^2$$

where the *Lamé parameters* μ and λ are

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

$$\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)} \text{ (in 3D) or } \lambda = \frac{E\nu}{(1+\nu)(1-\nu)} \text{ (in 2D)}$$

It's the constraints of linearity and isotropy that lead to there being two parameters, and the relationship between (E, ν) and (μ, λ) comes from a geometric argument.

The stress tensor $\mathbf{P}$ is the derivative of ψ with respect to $\mathbf{F}$. Let's work this out with the variational notation:

$$\begin{aligned}\delta\psi(\mathbf{F}) &= \delta[\mu\|\boldsymbol{\epsilon}(\mathbf{F})\|_F^2 + \frac{\lambda}{2}(\operatorname{tr}\boldsymbol{\epsilon}(\mathbf{F}))^2] \\ &= 2\mu\boldsymbol{\epsilon} : \delta\boldsymbol{\epsilon} + \lambda(\operatorname{tr}\boldsymbol{\epsilon})\mathbf{I} : \delta\boldsymbol{\epsilon}\end{aligned}$$

Working with the definition of $\boldsymbol{\epsilon}(\mathbf{F})$:

$$\delta\boldsymbol{\epsilon} = \delta[\operatorname{sym}\mathbf{F}] = \operatorname{sym}\delta\mathbf{F} \quad \text{where } \operatorname{sym}\mathbf{A} = \tfrac{1}{2}\left(\mathbf{A} + \mathbf{A}^T\right)$$

For any symmetric matrix $\mathbf{S}$, it's easy to show that

$$\mathbf{S} : \operatorname{sym}\mathbf{A} = \mathbf{S} : \mathbf{A}$$

and since the matrices $\boldsymbol{\epsilon}$ and $\mathbf{I}$ are symmetric,

$$\boldsymbol{\epsilon} : \delta\boldsymbol{\epsilon} = \boldsymbol{\epsilon} : \delta\mathbf{F} \text{ and } \mathbf{I} : \delta\boldsymbol{\epsilon} = \mathbf{I} : \delta\mathbf{F}$$

so

$$\delta\psi(\mathbf{F}) = \left(2\mu\boldsymbol{\epsilon} + \lambda(\operatorname{tr}\boldsymbol{\epsilon})\mathbf{I}\right) : \delta\mathbf{F}$$

and since $\mathbf{P} = \partial\psi/\partial\mathbf{F}$,

$$\mathbf{P} = 2\mu\boldsymbol{\epsilon} + \lambda(\operatorname{tr}\boldsymbol{\epsilon})\mathbf{I}$$

This is not much use to us because it is for infinitesimal displacements, but I write it down because it is the prototype for the next two models.

St. Venant–Kirchhoff

This one comes from building an analogous energy from the Green's strain:

$$\psi = \mu\|\mathbf{E}\|_F^2 + \frac{\lambda}{2}(\operatorname{tr}\mathbf{E})^2$$

This is a simple and useful model that is often used when simplicity is key. It really only applies to the large-displacement, small-strain case because the volume preservation is a linear approximation that becomes quite inaccurate for large strains. In particular, it will permit objects to collapse entirely.

Its derivative, which is the stress tensor, follows a similar pattern as in linear elasticity.

$$\delta\psi = 2\mu\mathbf{E} : \delta\mathbf{E} + \lambda(\operatorname{tr}\mathbf{E})\mathbf{I} : \delta\mathbf{E}$$

$$\delta\mathbf{E} = \operatorname{sym}(\mathbf{F}^T\delta\mathbf{F}) \quad \text{so} \quad \mathbf{E} : \delta\mathbf{E} = \mathbf{E} : \mathbf{F}^T\delta\mathbf{F} = \mathbf{F}\mathbf{E} : \delta\mathbf{F} \quad \text{and} \quad \mathbf{I} : \delta\mathbf{E} = \mathbf{F} : \delta\mathbf{F}$$

$$\delta\psi = 2\mu\mathbf{F}\mathbf{E} : \delta\mathbf{F} + \lambda(\operatorname{tr}\mathbf{E})\mathbf{F} : \delta F = \mathbf{F}\left(2\mu\mathbf{E} + \lambda(\operatorname{tr}\mathbf{E})\mathbf{I}\right) : \delta F$$

...and this shows the derivative that defines $\mathbf{P}$:

$$\mathbf{P} = \mathbf{F}\left[2\mu\mathbf{E} + \lambda(\operatorname{tr}\mathbf{E})\mathbf{I}\right]$$

Corotational linear elasticity

This one applies the same exact formula to the corotational strain. If we carry through the same program as above, but starting with the corotational strain $\boldsymbol{\epsilon}_c$, we start with:

$$\psi(\mathbf{F}) = \mu\|\boldsymbol{\epsilon}_c\|_F^2 + \frac{\lambda}{2}(\operatorname{tr}\boldsymbol{\epsilon}_c)^2$$

This is still something we can compute readily from $\mathbf{F}$ (using the polar decomposition) and it will behave more like the linear model (in the absence of a global rotation) than StVK does (since the strain varies linearly rather than quadratically). The derivative is a bit more tricky to work out, because there is no formula for the polar decomposition to differentiate, but it is not hard; let's start with a couple of simple lemmas about derivatives of matrix products. First, for a rotation matrix $\mathbf{R}$, the product $\mathbf{R}^T\delta\mathbf{R}$ is skew symmetric (meaning it is the opposite of its transpose):

$$\delta(\mathbf{R}^T\mathbf{R}) = 0 = \delta\mathbf{R}^T\mathbf{R} + \mathbf{R}^T\delta\mathbf{R} = \mathbf{R}^T\delta\mathbf{R} + \left(\mathbf{R}^T\delta\mathbf{R}\right)^T$$

Second, the differential of the $\mathbf{S}$ term of the polar decomposition can be written:

$$\begin{aligned}\delta\mathbf{F} &= \delta[\mathbf{R}]\mathbf{S} + \mathbf{R}\delta[\mathbf{S}] \\ \mathbf{R}\delta[\mathbf{S}] &= \delta\mathbf{F} - \delta[\mathbf{R}]\mathbf{S} \\ \delta\mathbf{S} &= \mathbf{R}^T\delta\mathbf{F} - \left(\mathbf{R}^T\delta\mathbf{R}\right)\mathbf{S}\end{aligned}$$

Now going to the computation of $\delta\psi$, the first step is the same as the other models:

$$\delta\psi = 2\mu\boldsymbol{\epsilon}_c : \delta\boldsymbol{\epsilon}_c + \lambda(\operatorname{tr}\boldsymbol{\epsilon}_c)\mathbf{I} : \delta\boldsymbol{\epsilon}_c$$

Since $\boldsymbol{\epsilon}_c = \mathbf{S} - \mathbf{I}$, $\delta\boldsymbol{\epsilon}_c = \delta\mathbf{S}$. Filling in $\delta\mathbf{S}$ from our lemma we get

$$\begin{aligned}\delta\psi &= \left(2\mu\boldsymbol{\epsilon}_c + \lambda(\operatorname{tr}\boldsymbol{\epsilon}_c)\mathbf{I}\right) : \mathbf{R}^T\delta\mathbf{F} + \left(2\mu\boldsymbol{\epsilon}_c + \lambda(\operatorname{tr}\boldsymbol{\epsilon}_c)\mathbf{I}\right) : \left(\mathbf{R}^T\delta\mathbf{R}\right)\mathbf{S} \\ &= \mathbf{R}\left(2\mu\boldsymbol{\epsilon}_c + \lambda(\operatorname{tr}\boldsymbol{\epsilon}_c)\mathbf{I}\right) : \delta\mathbf{F} + \left(2\mu\boldsymbol{\epsilon}_c\mathbf{S} + \lambda(\operatorname{tr}\boldsymbol{\epsilon}_c)\mathbf{S}\right) : \left(\mathbf{R}^T\delta\mathbf{R}\right) \\ &= \mathbf{R}\left(2\mu\boldsymbol{\epsilon}_c + \lambda(\operatorname{tr}\boldsymbol{\epsilon}_c)\mathbf{I}\right) : \delta\mathbf{F}\end{aligned}$$

The second term is zero in the last step because it is the double-dot product of a symmetric and a skew-symmetric matrix. From this we can read off $\mathbf{P}(\mathbf{F}) = \partial\psi/\partial\mathbf{F}$:

$$\mathbf{P}(\mathbf{F}) = \mathbf{R}\left[2\mu\boldsymbol{\epsilon}_c + \lambda(\operatorname{tr}\boldsymbol{\epsilon}_c)\mathbf{I}\right]$$

The result is remarkably similar to the previous two — all this work just to determine what the matrix on the left is! In any case, this is easy to compute as long as you are willing to compute the polar decomposition.

Neo-Hookan material

Going beyond these three models that all use the same formula for strain energy, here is one more model that is popular in graphics. A problem with the linear models is that they don't perform well when there is a lot of compression, and are unable to maintain volume and prevent elements from inverting. To explicitly preserve volume across the full range of deformations, a neo-Hookean uses an energy based on the determinant of $\mathbf{F}$, which accurately measures volume change even for large strains. There are quite a few models called "neo-Hookean" and one commonly used in graphics is:

$$\psi(\mathbf{F}) = \tfrac{\mu}{2}(\|\mathbf{F}\|_F^2 - 3) - \mu\log\det\mathbf{F} + \tfrac{\lambda}{2}(\log\det\mathbf{F})^2$$

Note that rather than building on a strain, this model defines ψ straight from $\mathbf{F}$; it manages to have zero energy when $\mathbf{F} = \mathbf{I}$ by subtracting $\|\mathbf{I}\|_F^2 = 3$ from $\|\mathbf{F}\|_F^2$ and by taking the logarithm of the determinant of $\mathbf{F}$ (so that $\log\det\mathbf{I} = 0$).

Differentiating the three terms (recalling Jacobi's formula $\delta[\det\mathbf{A}] = (\det\mathbf{A})\,\mathbf{A}^{-T} : \delta\mathbf{A}$ from above):

$$\frac{\mu}{2}\delta[\|\mathbf{F}\|_F^2] = \mu\mathbf{F} : \delta\mathbf{F}$$

$$\mu\delta[\log\det\mathbf{F}] = \frac{\mu}{\det\mathbf{F}}\delta[\det\mathbf{F}] = \frac{\mu}{\det\mathbf{F}}(\det\mathbf{F})\,\mathbf{F}^{-T} : \delta\mathbf{F} = \mu\mathbf{F}^{-T} : \delta\mathbf{F}$$

$$\frac{\lambda}{2}\delta\left[(\log\det\mathbf{F})^2\right] = \lambda\log\det\mathbf{F}\,\delta[\log\det\mathbf{F}] = \lambda(\log\det\mathbf{F})\,\mathbf{F}^{-T} : \delta\mathbf{F}$$

and putting it back together the derivative, and hence the formula for $\mathbf{P}$, is surprisingly simple:

$$\begin{aligned}\delta\psi &= \mu\mathbf{F} : \delta\mathbf{F} - \mu\mathbf{F}^{-T} : \delta\mathbf{F} + \lambda(\log\det\mathbf{F})\,\mathbf{F}^{-T} : \delta\mathbf{F} \\ &= \left(\mu\mathbf{F} - \mu\mathbf{F}^{-T} + \lambda(\log\det\mathbf{F})\,\mathbf{F}^{-T}\right) : \delta\mathbf{F}\end{aligned}$$

so

$$\mathbf{P}(\mathbf{F}) = \mu(\mathbf{F} - \mathbf{F}^{-T}) + \lambda(\log\det\mathbf{F})\,\mathbf{F}^{-T}$$

Computing nodal forces

This has been a lot of derivation and you might ask what it gained us. A lot, it turns out! We now have all the pieces we need to compute the forces on the vertices of a deforming mesh. Rewinding back to the beginning of the story: we have a deformable object whose current shape is defined by the deformation gradient ϕ, and we have a strain energy that depends on ϕ:

$$E[\phi] = \int_B \psi(\mathbf{F}(\mathbf{X}))\,d\mathbf{X}$$

For an object that is approximated by a triangle or tetrahedral mesh, we can choose to model ϕ as a piecewise linear function that is defined by the vertex positions $\mathbf{x}_i$. Then this integral becomes a sum over the elements of the mesh:

$$E[\phi] = \sum_k \int_{T_k} \psi(\mathbf{F}(\mathbf{X}))\,d\mathbf{X} = \sum_k |T_k|\,\psi(\mathbf{F}_k)$$

where T_k is the (rest) volume or area belonging to element k. Since $\mathbf{F}$ is constant over each element, ψ is also constant, and the integral is just the size of T_k, times ψ evaluated for the constant value of $\mathbf{F}$ on element k, $\mathbf{F}_k$.

The forces are the derivatives of the potential energy

$$\mathbf{f}_i = -\frac{\partial E}{\partial \mathbf{x}_i}$$

and since each vertex affects the energy of all the adjacent triangles, each vertex has a contribution to its force related to each of its adjacent triangles. Turning our attention to one triangle, which is one of the terms in the sum defining $E[\phi]$ above, we now have a chain of functions that we have derivatives for: $\mathbf{x} \to \mathbf{F} \to \boldsymbol{\epsilon} \to \psi \to E$. The above models all let us write the derivative $\delta\psi = \mathbf{P} : \delta\mathbf{F}$, which takes care of the middle part of the chain. You'll recall earlier we were able to write $\mathbf{F}$ in terms of the triangle edges as

$$\mathbf{F} = \mathbf{D}\mathbf{D}_0^{-1}$$

where $\mathbf{D}$ is the matrix whose columns are the triangle edge vectors (and $\mathbf{D}_0$ is the same matrix for the rest shape). Since $\mathbf{D}_0$ is constant the derivative here is

$$\delta \mathbf{F} = \delta[\mathbf{D}]\,\mathbf{D}_0^{-1}$$

Substituting this into the expression for $\delta\psi$,

$$\begin{aligned}\delta\psi &= \mathbf{P} : (\delta[\mathbf{D}]\,\mathbf{D}_0^{-1}) \\ &= \mathbf{P}\mathbf{D}_0^{-T} : \delta\mathbf{D}\end{aligned}$$

Finally, since we are focused on just one triangle, we have $\delta E = |T_k|\,\delta\psi$, so

$$\delta E = |T_k|\,\mathbf{P}\mathbf{D}_0^{-T} : \delta\mathbf{D}$$

Defining a name for that matrix, $\mathbf{H} = -|T_k|\,\mathbf{P}\mathbf{D}_0^{-T}$, we have

$$\delta E = -\,\mathbf{H} : \delta\mathbf{D}$$

This is the derivative of the energy with respect to the triangle edge vectors $\mathbf{x}_1 - \mathbf{x}_0$ and $\mathbf{x}_2 - \mathbf{x}_0$, which are the two columns of the matrix $\mathbf{D}$. This means the columns of $\mathbf{H}$ are the forces due to this triangle on vertices $\mathbf{x}_1$ and $\mathbf{x}_2$:

$$\mathbf{H} = \begin{bmatrix}\mathbf{f}_1 & \mathbf{f}_2\end{bmatrix} \quad \text{and} \quad \mathbf{f}_0 = -(\mathbf{f}_1 + \mathbf{f}_2)$$

Hooray! This is what we wanted! So now we can write the forces as a function of the positions. To bring it all together the process to compute the forces due to one triangle is:

1. Ahead of time compute $\mathbf{D}_0^{-1}$ and $|T_k|$.
2. Compute $\mathbf{F} = \mathbf{D}\mathbf{D}_0^{-1}$ from the current vertex positions.
3. Compute the strain from $\mathbf{F}$ using the formulas appropriate to your model.
4. Compute the stress $\mathbf{P}$ from the strain using the formulas appropriate to your model.
5. Compute $\mathbf{H} = -|T_k|\,\mathbf{P}\mathbf{D}_0^{-T}$, and the forces are sitting in the columns of $\mathbf{H}$.

To compute the total force on each vertex you need to loop over all the triangles and accumulate their contributions. That's all there is to it!