Elastic Rod Networks

Julian Panetta

August 8, 2018

This document describes how to model the static equilibria of networks formed by elastic rods connecting at deformable joints. Contrary to the rod linkages also implemented in this repository, the rods meeting a joint in this model are not allowed to pivot freely, and the joint stores some elastic energy. We begin with the joint model from [2], and the elastic rod model is the one from [1] (with corrected gradients/Hessians).

Once we have implemented the joint model, we will run experiments to validate the accuracy of the displacements it computes against a volumetric (tetrahedral) finite element simulation of analogous joint geometry. If the accuracy is poor, we will attempt to invent a better joint model, either one that is better physically motivated or that is data driven (fit to data generated by the volumetric simulation). If the model is accurate, we will use the network simulation in some higher-level design or analysis tool. For instance, we might implement a homogenization tool for determining the effective material properties of a metamaterial (periodic microstructure) built from elastic rods.

1 Rod Network Graph

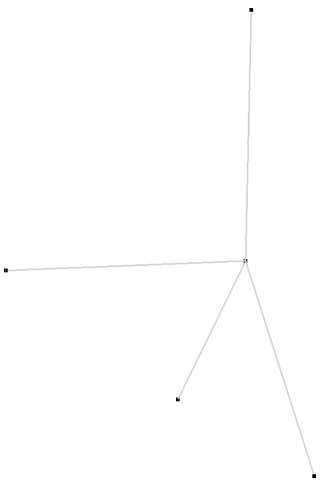

Figure 1 *An example rod network graph.*

The network's initial configuration is defined by an embedded graph (i.e., line mesh) with vertices and edges (V, E). Each edge of this graph is referred to as a rod *segment*. This embedded graph is stored in the `OBJ` format using line elements (using the `l` keyword instead of the usual `f`).

Each graph vertex represents either an elastic joint or a free rod end.

Each rod segment is modeled as a distinct discrete elastic rod with n_s subdivisions (this includes up to 2 "connection edges" described above). We label the n^{th} segment s_n for $n \in \{0 \dots |E| - 1\}$ and the joints $\jmath_i$ for $i \in \{0 \dots |J| - 1\}$, where $J \subset V$ is the set of vertices of valence 2–4.

2 Rod and Joint Representation

Recall that a discrete elastic rod's configuration is defined by its centerline positions and its material frame angles (expressed relative to the rod's hidden reference frame state). The material frame is used to express the orientation/twist of the rod's cross sections, which is particularly important for flat, anisotropic rods.

The network's rods meet at an elastic joint. Following [2], we define the energy stored in the joint by constructing a fictitious "connection edge" for each incident elastic rod. In the rest configuration, this connection edge continues straight across the joint (with the same tangent and material axes as the terminal edge incident the joint).

Conceptually, all connection edges for a given joint behave as if they are glued to a single rigid body at the joint. For a given deformed configuration of the incident elastic rods, the orientation of this body is chosen to approximately minimize the bending and twisting energy of the incident rods (by solving the least-squares fitting in Eq (4) of [2] and computing a polar decomposition). Once this "optimal" rigid body orientation is found, it determines the degrees of freedom of the connection edges (i.e., their orientations and material axes). Applying this orientation to each connection edge stores some twisting and bending energy in the incident rods; this energy is defined to be the joint's energy. TODO: draw a figure...

3 Network Degrees of Freedom

Our rod network model incorporates the constraints imposed by the joints by constructing a reduced set of variables that parametrize all admissible rod configurations. These reduced variables consist of (in order):

- For each rod segment $s \in E$:
  - Centerline positions for all interior and free-end nodes of s. The x, y, and z coordinates of the first interior/free node come first, followed by the coordinates for all subsequent nodes.
  - Material frame angles θ for all interior and free-end edges of s.
- The position of each joint $\jmath \in J$.

4 Elastic Energy, Gradients, and Hessians

The elastic energy of the full network is computed by summing the energy from each elastic rod and joint. However, because we implement the joint energies by explicitly representing the "connection edges" in the incident rod segments, the joint energy is effectively already contained in the energy computed for the elastic rod segments. In other words, we only need to sum the energy stored in each elastic rod segment.

We can then compute derivatives of this energy with respect to the degrees of freedom by assembling the (transformed) derivatives of the discrete elastic rod model.

4.1 Gradients and Hessian

First, we write the elastic energy with respect to the rod network's reduced variables

$$E(\mathbf{v}(\mathbf{r})),$$

where vectors $\mathbf{v}$ and $\mathbf{r}$ collect the network's full and reduced variables. By this, we mean $\mathbf{v}$ holds the full degrees of freedom of all elastic rods in the network (including the variables controlling the connection edges), while $\mathbf{r}$ holds only the non-connection edges' degrees of freedom and the joint positions.

The gradient and Hessian with respect to the reduced variables are now given by the chain rule:

$$\frac{\partial E}{\partial r_i} = \frac{\partial E}{\partial v_k}(\mathbf{v}(\mathbf{r}))\frac{\partial v_k}{\partial r_i}$$

$$\frac{\partial^2 E}{\partial r_i \partial r_j} = \frac{\partial v_k}{\partial r_i} \frac{\partial^2 E}{\partial v_k \partial v_l} \frac{\partial v_l}{\partial r_j} + \frac{\partial E}{\partial v_k} \frac{\partial^2 v_k}{\partial r_i \partial r_j}.$$

The terms $\frac{\partial v_k}{\partial r_i}$ and $\frac{\partial^2 v_k}{\partial r_i \partial r_j}$ involve the gradients and Hessians of the connection edge variables with respect to reduced variables, which in turn involves the gradients and Hessians of the polar decomposition.

4.2 Practical Implementation Steps

1. Create the files `RodNetwork.{hh,cc}` by copying `RodLinkage.{hh,cc}`.

2. Replace the existing `RodNetwork::Joint` class with one that supports an arbitrary number of incident rod segments (keeping track of which segments are incident and whether the joint appears at the beginning or end of the rod segment). This connectivity information can be stored in a pair of `std::vector`s. Most of the functions and data members brought over from `RodLinkage::Joint` should be removed (e.g., the joint normal and constraint stuff; also the joint should only have its position as the degrees of freedom–eA and eB stuff should be removed).

3. Write code to construct an elastic rod network from an OBJ file, building the elastic rod segments and joints based on the graph read from the file. The elastic rods should be built so that, for ends terminating in a joint, the last rod edge extends straight past the joint to form a "connection edge."

   Eventually, we will probably need an additional file that stores the orientation of material axis 1 (referred to as $\mathbf{n}$ in the rod mesh paper [2] and $\mathbf{d}_1$ in [1] and the code) in the rest configuration for all rod endpoints appearing in a joint. This orientation cannot in general be inferred from the graph (though [2] infers them with a heuristic that makes sense for networks lying on a surface, it doesn't make sense for "volumetric" networks). For now, we can stick with rotationally-symmetric cross-sections, where this orientation is irrelevant. `RodNetwork::saveVisualizationGeometry` should be useful for debugging this step.

4. Solve for the rigid transformation taking the fictitious rigid body at each joint from its rest configuration to its deformed configuration by performing the minimization in Eq (4) of [2] and computing a polar decomposition. Debug this orientation by outputting an oriented cube for each joint (add a `RodNetwork::saveJointDebuggingGeometry`).

5. Use this rigid transformation to orient the "connection edges" for each joint (transforming the first/last edges of the incident rod segments). At this point, the elastic energy of rod network should be computed properly.

6. Update the `RodLinkage::gradient` to transform the individual elastic rod segment energy gradients using the chain rule and sum them into the full gradient. This chain rule involves the derivative of the polar decomposition as discussed in Eq (6) of [2]. Validate the gradient using finite difference tests like in `test_elastic_rod_linkage.cc`.

7. Update the `RodLinkage::hessian` to transform the individual elastic rod segment energy Hessians and gradients using the chain rule and sum them into the full gradient. This chain rule involves the first and second derivatives of the polar decomposition; the latter is given in the appendix of [2]. Validate the Hessian using finite difference tests like in `test_elastic_rod_linkage.cc`.

4.3 Hessian Implementation Steps

At a high level, the subtasks to complete are:

1. Compute $\frac{\partial^2 DC^T}{\partial u \partial v}$ in `RodNetwork::Joint::d2_DC_dkdl` using the second derivative formulas for the tangent and normal vectors.

2. Validate `RodNetwork::Joint::d2_DC_dkdl` by comparing against a finite difference of the matrices computed by `RodNetwork::Joint::DCdots`.
3. Compute $\frac{\partial S}{\partial u}$ in `RodNetwork::Joint::Sdots` (should be relatively easy).
4. Compute $\frac{\partial^2 R}{\partial u \partial v}$ in `RodNetwork::Joint::d2_R_dkdl` using the formulas for the second derivative of the polar decomposition. You will need to use the values computed by `RodNetwork::Joint::DCdots`, `RodNetwork::Joint::d2_DC_dkdl`, `RodNetwork::Joint::Sdots`, and `RodNetwork::Joint::Rdots`.
5. Validate `RodNetwork::Joint::d2_R_dkdl` by comparing against a finite difference of the matrices computed by `RodNetwork::Joint::Rdots`.
6. Implement `RodNetwork::Joint::hessian`. This should be similar to the `RodNetwork::Joint::jacobian` implementation, except you need to use `d2_R_dkdl` instead of `d_R_dk`. You'll also need to use the product rule and the derivative of $\mathbf{d}_1$ for the line that looks like:

   ```
   Jac(3 * N + lsi, k) = -d_1.dot(d_R_dk[k] * rod.restMaterialFrameD2(edgeIdx));
   ```

7. Validate `RodNetwork::Joint::hessian` by comparing against a finite difference of the matrix `RodNetwork::Joint::jacobian`.
8. Implement the chain rule expression for `RodNetwork::hessian` using `RodNetwork::Joint::Rdots`, `RodNetwork::Joint::d2_R_dkdl`, `ElasticRod::hessian`, and `ElasticRod::gradient`. It may be helpful to read `RodLinkage::hessian`.
9. Validate `RodNetwork::hessian` by comparing against a finite difference of `RodNetwork::gradient`.

4.4 Hessian representation

The Hessian matrix of the elastic energy is quite large (it has as many rows and columns as variables in the network), but most of its entries are 0. To store the matrix efficiently, we use a sparse representation. Specifically, when constructing the Hessian, we use a "sparse triplet" representation which simply stores a large list of the row index, column index, and value of each nonzero entry ((i, j, v) "triplets"). Its fine to have multiple entries with the same row and column index in this list: their values will be summed together to produce the final sparse matrix. This fact will be useful when implementing all the sums in the chain rule (1) below (we can just push all the values summing to a given (i, j) entry into the list, and they will be automatically summed for us at the end).

This sparse triplet representation is implemented by the `TMatrix` type, which has inside it a `std::vector` named `nz` holding the matrix's nonzero entries.

4.5 Chain rule implementation

Now let's look at how to implement the two terms appearing in the Hessian's chain rule:

$$\frac{\partial^2 E}{\partial r_i \partial r_j} = \frac{\partial v_k}{\partial r_i} \frac{\partial^2 E}{\partial v_k \partial v_l} \frac{\partial v_l}{\partial r_j} + \frac{\partial E}{\partial v_k} \frac{\partial^2 v_k}{\partial r_i \partial r_j}.$$

First, we rewrite this Hessian in terms of a sum over the segments, since the network's elastic energy is the sum of the elastic energy stored in each segment:

$$\frac{\partial^2 E}{\partial r_i \partial r_j} = \sum_s \left(\frac{\partial v_k^s}{\partial r_i} \frac{\partial^2 E^s}{\partial v_k^s \partial v_l^s} \frac{\partial v_l^s}{\partial r_j} + \frac{\partial E^s}{\partial v_k^s} \frac{\partial^2 v_k^s}{\partial r_i \partial r_j} \right). \tag{1}$$

Now $\frac{\partial v_k^s}{\partial r_i}$ is the Jacobian of segment s's variables (i.e. red and blue vertex/edge quantities) with respect to all the variables of the network. This is a large sparse matrix with identity blocks for the blue/joint quantities' rows and with entries from the `RodNetwork::Joint::jacobian` matrix in the red quantities' rows.

We already effectively used this matrix to implement the `RodNetwork::gradient`, but we didn't explicitly construct it (we just implemented the matrix multiplication it appeared in by using the entries we knew to be identity blocks and the entries we knew came from `RodNetwork::Joint::jacobian`), and won't construct it to build the Hessian either. Tensor $\frac{\partial^2 v_k^s}{\partial r_i \partial r_j}$ is the Hessian of segment s's variables with respect to all the variables of the network—the nonzero entries of this large matrix will come from `RodNetwork::Joint::hessian`.

Matrix $\frac{\partial^2 E^s}{\partial v_k^s \partial v_l^s}$ is the Hessian of segment s's elastic energy, which can be computed just by calling `ElasticRod::hessian`. This is a sparse matrix in the sparse triplet format described above. To implement the first term of the Hessian chain rule (1), you should loop over the nonzero entries stored in this triplet matrix's `nz` array. Each of these entries gives you row and column indices—we'll call them k and l here—as well as the element's value. This rod energy Hessian entry will contribute a whole list of nonzero entries to the network energy Hessian: for each (i, j) such that $\frac{\partial v_k^s}{\partial r_i}$ and $\frac{\partial v_l^s}{\partial r_j}$ are nonzero, you will add an (i, j) entry to the network Hessian with value $\frac{\partial^2 E^s}{\partial v_k^s \partial v_l^s} * \frac{\partial v_k^s}{\partial r_i} \frac{\partial v_l^s}{\partial r_j}$. Essentially, this is just the sparse outer product of the vectors $\frac{\partial v_k^s}{\partial r_i}$ and $\frac{\partial v_l^s}{\partial r_j}$ (these are vectors because indices k and l are fixed) multiplied by the current entry of the rod energy Hessian.

To implement the second term of (1), you can loop over the pairs (i, j) of indices of network variables influencing either the start or end joint of the current segment (entries from `RodNetwork::jointVars`). For each of these (i, j), you find the k indices for which $\frac{\partial^2 v_k^s}{\partial r_i \partial r_j}$ is nonzero (this should just be the red vertices/edge variables of the current segment's joints). Then, for each of these k, you multiply the derivative of the segment's rod energy with respect to variable k (an entry from the `ElasticRod::gradient`) by the nonzero entry $\frac{\partial^2 v_k^s}{\partial r_i \partial r_j}$ (an entry from `RodNetwork::hessian`) to obtain an output value for entry (i, j) in the network's Hessian.

4.6 Computing $\frac{\partial^2 \mathbf{t}}{\partial u \partial v}$ and $\frac{\partial^2 \mathbf{d}_2}{\partial u \partial v}$

To compute $\frac{\partial^2 DC^T}{\partial u \partial v}$ for a particular joint, we mainly need to evaluate $\frac{\partial^2 \mathbf{t}}{\partial u \partial v}$ and $\frac{\partial^2 \mathbf{d}_2}{\partial u \partial v}$ for each blue edge incident the joint. We do this by differentiating $\frac{\partial \mathbf{t}}{\partial u}$ and $\frac{\partial \mathbf{d}_2}{\partial u}$ with respect to another of the joint's variables, v. We'll get different expressions depending on the types of variables u and v (whether they are vertex coordinates or θs).

4.6.1 Case 1: u and v both vertex coordinates

We first consider the case where both variables control vertex coordinates–either a coordinate of a blue vertex or the joint position. In this case the derivatives with respect to u are:

$$\begin{aligned}
\frac{\partial \mathbf{t}}{\partial u} &= s_u \frac{I[:,u] - \mathbf{t}\,\mathbf{t}[u]}{\|\mathbf{e}\|} \\
\frac{\partial \mathbf{d}_2}{\partial u} &= s_u \left[\frac{(\widehat{\mathbf{d}_1} \cdot \mathbf{t}) \left(\widehat{\mathbf{t}} \times \frac{\partial \mathbf{t}}{\partial u} \right) + (\widehat{\mathbf{t}} \times \mathbf{t}) \left(\widehat{\mathbf{d}_1} \cdot \frac{\partial \mathbf{t}}{\partial u} \right)}{1 + \widehat{\mathbf{t}} \cdot \mathbf{t}} - \frac{(\widehat{\mathbf{d}_1} \cdot \mathbf{t})(\widehat{\mathbf{t}} \times \mathbf{t}) \left(\widehat{\mathbf{t}} \cdot \frac{\partial \mathbf{t}}{\partial u} \right)}{(1 + \widehat{\mathbf{t}} \cdot \mathbf{t})^2} - \widehat{\mathbf{d}_1} \times \frac{\partial \mathbf{t}}{\partial u} \right],
\end{aligned} \tag{2}$$

where $I[:, u]$ means the u^{th} column of the 3×3 identity matrix, and we've defined the sign multiplier:

$$s_i = \begin{cases} \texttt{orientation\_case} & \text{if } i \text{ is a joint position variable of the current segment} \\ \texttt{-orientation\_case} & \text{if } i \text{ is a blue vertex position variable of the current segment} \\ 0 & \text{otherwise} \end{cases}.$$

Now, we can differentiate with respect to v, starting with $\mathbf{t}$:

$$\frac{\partial^2 \mathbf{t}}{\partial u \partial v} = s_u \left[-s_v \frac{I[:,u] - \mathbf{t}\,\mathbf{t}[u]}{\|\mathbf{e}\|^2} \mathbf{t}[v] - s_v \frac{(I[:,v] - \mathbf{t}\,\mathbf{t}[v])\,\mathbf{t}[u] + \mathbf{t}\,(I[u,v] - \mathbf{t}[u]\mathbf{t}[v])}{\|\mathbf{e}\|^2} \right]$$

$$= s_u s_v \frac{\mathbf{t}\left(3\mathbf{t}[u]\mathbf{t}[v] - I[u,v]\right) - I[:,v]\mathbf{t}[u] - I[:,u]\mathbf{t}[v]}{\|\mathbf{e}\|^2}.$$

Notice that $I[u,v] = \delta_{uv}$ is 1 if $u = v$ and 0 otherwise.

Next, we differentiate $\frac{\partial \mathbf{d}_2}{\partial u}$. To simplify the expression, we assume that we only evaluate the second derivative immediately after the source frame has been updated (so $\widehat{\mathbf{d}_1} = \mathbf{d}_1$ and $\widehat{\mathbf{t}} = \mathbf{t}$). This means that our final Hessian expression will only be correct with an up-to-date source frame, but the elastic rods hessian code already makes this assumption. With this simplification, all the terms containing $\widehat{\mathbf{d}_1} \cdot \mathbf{t}$ and $\widehat{\mathbf{t}} \times \mathbf{t}$ vanish. In particular, we can pretend the entire second term is constant (any term of the product rule for it will contain at least one of these two vanishing expressions), as well as $\left(\widehat{\mathbf{t}} \times \frac{\partial \mathbf{t}}{\partial u}\right)$, $\left(\widehat{\mathbf{d}_1} \cdot \frac{\partial \mathbf{t}}{\partial u}\right)$, and the denominator of the first term (which actually will equal 2). Also, recall that the source quantities are all constant as the edge changes, by definition. This leaves us with:

$$\frac{\partial^2 \mathbf{d}_2}{\partial u \partial v} = s_u \left[\frac{\left(\widehat{\mathbf{d}_1} \cdot \frac{\partial \mathbf{t}}{\partial v}\right)\left(\widehat{\mathbf{t}} \times \frac{\partial \mathbf{t}}{\partial u}\right) + \left(\widehat{\mathbf{t}} \times \frac{\partial \mathbf{t}}{\partial v}\right)\left(\widehat{\mathbf{d}_1} \cdot \frac{\partial \mathbf{t}}{\partial u}\right)}{2} - \widehat{\mathbf{d}_1} \times \left(\frac{\partial^2 \mathbf{t}}{\partial u \partial v}\right) \right],$$

where we can reuse our previous expression for the derivatives and Hessian of the tangent.

4.6.2 Case 2: u is a vertex coordinate and v is a blue θ

Next, we differentiate our expressions in (2) with respect to the θ variable *from the same blue edge we're differentiating* (the other blue θs don't affect this edge's quantites). We refer to this θ as v below. In this case:

$$\frac{\partial^2 \mathbf{t}}{\partial u \partial v} = 0$$
$$\frac{\partial^2 \mathbf{d}_2}{\partial u \partial v} = s_u \left[\frac{(\widehat{\mathbf{d}_2} \cdot \mathbf{t})\left(\widehat{\mathbf{t}} \times \frac{\partial \mathbf{t}}{\partial u}\right) + (\widehat{\mathbf{t}} \times \mathbf{t})\left(\widehat{\mathbf{d}_2} \cdot \frac{\partial \mathbf{t}}{\partial u}\right)}{1 + \widehat{\mathbf{t}} \cdot \mathbf{t}} - \frac{(\widehat{\mathbf{d}_2} \cdot \mathbf{t})(\widehat{\mathbf{t}} \times \mathbf{t})\left(\widehat{\mathbf{t}} \cdot \frac{\partial \mathbf{t}}{\partial u}\right)}{(1 + \widehat{\mathbf{t}} \cdot \mathbf{t})^2} - \widehat{\mathbf{d}_2} \times \frac{\partial \mathbf{t}}{\partial u} \right],$$

where we used the fact that $\frac{\partial \widehat{\mathbf{d}_1}}{\partial v} = \widehat{\mathbf{d}_2}$ in this case where v is the edge's θ variable.

4.6.3 Case 3: u is a blue theta and v is a vertex coordinate

By symmetry of the Hessian, this case is the same as the one above with u and v labels exchanged.

$$\frac{\partial^2 \mathbf{t}}{\partial u \partial v} = 0$$
$$\frac{\partial^2 \mathbf{d}_2}{\partial u \partial v} = s_v \left[\frac{(\widehat{\mathbf{d}_2} \cdot \mathbf{t})\left(\widehat{\mathbf{t}} \times \frac{\partial \mathbf{t}}{\partial v}\right) + (\widehat{\mathbf{t}} \times \mathbf{t})\left(\widehat{\mathbf{d}_2} \cdot \frac{\partial \mathbf{t}}{\partial v}\right)}{1 + \widehat{\mathbf{t}} \cdot \mathbf{t}} - \frac{(\widehat{\mathbf{d}_2} \cdot \mathbf{t})(\widehat{\mathbf{t}} \times \mathbf{t})\left(\widehat{\mathbf{t}} \cdot \frac{\partial \mathbf{t}}{\partial v}\right)}{(1 + \widehat{\mathbf{t}} \cdot \mathbf{t})^2} - \widehat{\mathbf{d}_2} \times \frac{\partial \mathbf{t}}{\partial v} \right].$$

4.6.4 Case 4: u and v are both blue θs

When both u and v are the blue θ variable for this edge, we have:

$$\frac{\partial^2 \mathbf{t}}{\partial u \partial v} = 0$$
$$\frac{\partial^2 \mathbf{d}_2}{\partial u \partial v} = \frac{\partial}{\partial v}\Big(- \mathbf{d}_1\Big) = -\mathbf{d}_2.$$

References

[1] Miklós Bergou, Basile Audoly, Etienne Vouga, Max Wardetzky, and Eitan Grinspun. Discrete viscous threads. *ACM Trans. Graph.*, 29(4):116:1–116:10, July 2010.

[2] Jesús Pérez, Bernhard Thomaszewski, Stelian Coros, Bernd Bickel, José A. Canabal, Robert Sumner, and Miguel A. Otaduy. Design and fabrication of flexible rod meshes. *ACM Trans. Graph.*, 34(4):138:1–138:12, July 2015.