Animation & CGI Motion: Elastic Body Simulation

Theme 4 Milestone 1

Academic Honesty Policy

You are permitted and encouraged to discuss your work with other students. You may work out equations in writing on paper or a whiteboard. You are encouraged to use the discussion boards to converse with other students, the TA, and the instructor.

HOWEVER, you may NOT share source code or hardcopies of source code. Refrain from activities or the sharing of materials that could cause your source code to APPEAR TO BE similar to another student's enrolled in this or previous years. We will be monitoring source code for individuality. Source code should be yours and yours only. Do not cheat.

1 Introduction

In Theme 4 we'll tackle the elastic body problem. Simulation of the elastic material is based on formulation of an elastic energy that is quadratic in a strain that describes how the object is deformed. Using different aspects of the deformation as our strain, we can capture different modes of deformation in the energies, and thus obtain the forces that counter-act these modes. In this milestone we'll examine three elastic forces: spring force, bending force, and the constant-strain-triangle (CST) force.

2 New XML Features

This milestone introduces the following new simulation features:

1. The *elasticbodyspringforce* feature specifies a spring force:

   ```
   <elasticbodyspringforce i1="0" i2="1" alhpa="1" l0="1.5"/>
   ```

   The *i1* and *i2* attributes specify between which particle this force acts; note that different than the spring force we implemented in the first theme, this force does not require an edge between the two endpoints. The *alpha* attribute specifies the material's elastic modulus (equivalent to EA where E is the young's modulus and A is the area of the cross-section of the spring). The *l0* attribute specifies the rest length which means the same as the spring in theme I.

2. The *elasticbodybendingforce* feature specifies a bending force:

   ```
   <elasticbodybendingforce i1="0" i2="1" i3="2" alhpa="1" theta0="1.5"/>
   ```

   The *i1*, *i2* and *i3* attributes specify between which particle this force acts. To represent a bending deformation we need three particles, where the second one is the hinge. The *alpha* attribute specifies the material's elastic modulus in resisting the bending mode. The *theta0* attribute specifies the rest angle.

3. The *elasticbodycstforce* feature specifies a CST force:

   ```
   <elasticbodycstforce i1="0" i2="1" i3="2" youngsmodulus="1" poissonratio="1" ebxx="0"
    ebyy="0" ebxy="0" xb1x="0" xb1y="0" xb2x="1" xb2y="0" xb3x="0" xb3y="1"/>
   ```

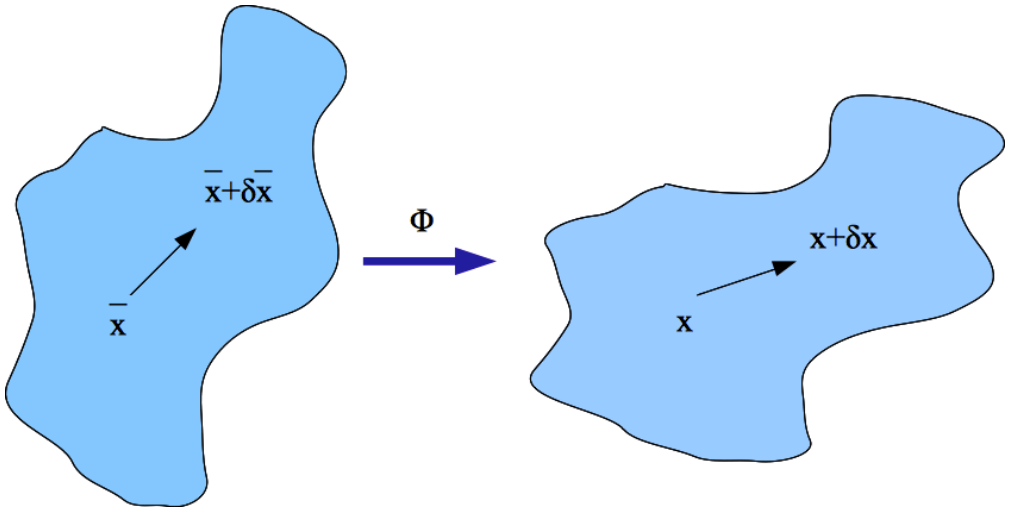


Figure 1: The deformation map.

The *i1*, *i2* and *i3* attributes specify the three vertices of the triangle where this force acts. The *youngsmodulus* and *poisonratio* attributes are the material parameters. *ebxx*, *ebxy* and *ebyy* encodes the "resting strain" $\bar{\epsilon}$ that is discussed in the next section. *xb1x* and *xb1y* specify the undeformed position of vertex 1, and *xb2x*, *xb2y*, *xb3x* and *xb3y* are similar.

3 Kinematics

3.1 The Deformation Map and Deformation Gradient

The kinematics of a deforming material are represented by a *deformation mapping* $\Phi : \mathbb{R}^2 \to \mathbb{R}^2$ taking every undeformed material point $\mathbf{x}$ on the plane to its corresponding deformed position $\Phi(\mathbf{x})$. In general, Φ is an arbitrary nonlinear map taking 2 coordinates denoting a position to a new set of 2 coordinates denoting some other position.

Now we'd like to know how much this deformation is stretching, compressing or shearing the material. Since a generic deformation is an arbitrary nonlinear map, we only look at each point locally (by examining its infinitesimal neighborhood). Consider a point $\bar{\mathbf{x}}$ in the undeformed space, and a nearby point $\bar{\mathbf{x}} + \delta\bar{\mathbf{x}}$. The deformed position are $\mathbf{x}$ and $\mathbf{x} + \delta\mathbf{x}$, and by definition of Φ we have:

$$\begin{aligned} \mathbf{x} &= \Phi(\bar{\mathbf{x}}) \\ \mathbf{x} + \delta\mathbf{x} &= \Phi(\bar{\mathbf{x}} + \delta\bar{\mathbf{x}}) \\ &= \Phi(\bar{\mathbf{x}}) + \nabla\Phi(\bar{\mathbf{x}})\delta\bar{\mathbf{x}} + O(\|\delta\bar{\mathbf{x}}\|^2) \\ \delta\mathbf{x} &= \nabla\Phi(\bar{\mathbf{x}})\delta\bar{\mathbf{x}} + O(\|\delta\bar{\mathbf{x}}\|^2) \end{aligned}$$

$\nabla\Phi(\bar{\mathbf{x}})$ is the Jacobian of the transformation Φ evaluated at $\bar{\mathbf{x}}$, often called the *Deformation Gradient*. The high order terms in the end can be omitted because we assume $\delta\bar{\mathbf{x}}$ is infinitesimal.

3.2 The Strain

Now we are ready to evaluate how much lengths change after such a transformation. Under the assumption that $\delta\bar{\mathbf{x}}$ is infinitesimal, the line from $\bar{\mathbf{x}}$ to $\bar{\mathbf{x}}+\delta\bar{\mathbf{x}}$ in the undeformed space remains a line after the deformation,

going from $\mathbf{x}$ to $\mathbf{x}+\delta\mathbf{x}$. Therefore the lengths of this piece of material before and after deformation are $\|\delta\bar{\mathbf{x}}\|$ and $\|\delta\mathbf{x}\|$ respectively. For the ease of computation we look at the change in squared length (so as to avoid taking square roots):

$$\begin{aligned}\delta\mathbf{x}^T\delta\mathbf{x} - \delta\bar{\mathbf{x}}^T\delta\bar{\mathbf{x}} &= (\nabla\Phi(\bar{\mathbf{x}})\delta\bar{\mathbf{x}})^T(\nabla\Phi(\bar{\mathbf{x}})\delta\bar{\mathbf{x}}) - \delta\bar{\mathbf{x}}^T\delta\bar{\mathbf{x}} \\ &= \delta\bar{\mathbf{x}}^T\nabla\Phi(\bar{\mathbf{x}})^T\nabla\Phi(\bar{\mathbf{x}})\delta\bar{\mathbf{x}} - \delta\bar{\mathbf{x}}^T\delta\bar{\mathbf{x}} \\ &= \delta\bar{\mathbf{x}}^T[\nabla\Phi(\bar{\mathbf{x}})^T\nabla\Phi(\bar{\mathbf{x}}) - I]\delta\bar{\mathbf{x}}\end{aligned}$$

The reason we can use change in squared length instead of change in length itself as a measure of stretching is that, to the first order, they are equivalent as long as the deformation is small. In order to make this change in squared length a scale-invariant representation of relative stretching/compression, we extract out the term in the middle, $[\nabla\Phi(\bar{\mathbf{x}})^T\nabla\Phi(\bar{\mathbf{x}}) - I]$, and call it the *strain*, or ϵ.

This strain is a second-order tensor, unlike the 1D case where a single scalar fully describes how much a spring is stretched as shown in class two weeks ago. In 2D materials can be stretched differently in different directions, so a second-order tensor is needed. The physical meaning of this tensor is that when you hit it on both sides with a direction $\delta\bar{\mathbf{x}}$, it returns a change in squared length in that direction. In its matrix representation, the eigenvectors of this matrix are the two directions that are stretched or compressed the most, and the corresponding eigenvalues tell you how much the stretching/compression is in that direction.

Another interesting thing to note is that ϵ is rotation-invariant, even though $\nabla\Phi$ is not. As a measure of deformation, the strain should not depend on any rigid motion such as translation and rotation. It is obvious that $\nabla\Phi$, which is the result of a spatial differential operator, stays the same under translation, but what about rotation? To test this, we apply a constant rotation Q to any deformation Φ to obtain a new deformation $\tilde{\Phi}$, and see how ϵ responds:

$$\begin{aligned}\tilde{\Phi} &= Q\Phi \\ \nabla\tilde{\Phi} &= Q\nabla\Phi \\ \tilde{\epsilon} &= \nabla\tilde{\Phi}^T\nabla\tilde{\Phi} - I \\ &= (Q\nabla\Phi)^T(Q\nabla\Phi) - I \\ &= \nabla\Phi^T Q^T Q\nabla\Phi - I \\ &= \nabla\Phi^T\nabla\Phi - I \\ &= \epsilon\end{aligned}$$

Thus we've convinced ourselves that our strain ϵ is invariant under any rigid motions, therefore it qualifies as a measure of deformation.

3.3 Example Deformation

Let's look at an example with real numbers. Consideration an affine deformation of a rectangular sheet of rubber as in Figure 2. This rectangle is stretched along the green axis by 80%, and compressed along the red axis by 40%. This means our deformation map Φ is multiplication by a matrix with two eigenvalues: 1.8 corresponding to the eigenvector (3, 1) (green axis), and 0.6 corresponding to the eigenvector (-1, 3) (red axis). Therefore Φ looks like this:

$$\begin{aligned}\Phi(\mathbf{x}) &= \frac{1}{\sqrt{10}}\begin{pmatrix}3 & -1\\ 1 & 3\end{pmatrix}\begin{pmatrix}1.8 & 0\\ 0 & 0.6\end{pmatrix}\frac{1}{\sqrt{10}}\begin{pmatrix}3 & 1\\ -1 & 3\end{pmatrix}\mathbf{x} \\ &= \begin{pmatrix}1.68 & 0.36\\ 0.36 & 0.72\end{pmatrix}\mathbf{x}\end{aligned}$$

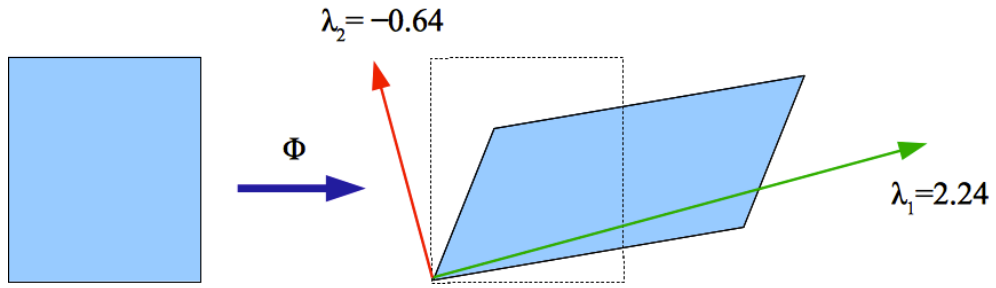


Figure 2: An example deformation of square, with stretching in one direction and compression in the other. The two arrows denote the two eigenvectors of the strain tensor with corresponding eigenvalues (λ_1 and λ_2) marked on the arrows.

We can compute the strain tensor ϵ from Φ:

$$\begin{aligned}\epsilon &= \nabla\Phi(\bar{\mathbf{x}})^T\nabla\Phi(\bar{\mathbf{x}}) - I \\ &= \begin{pmatrix}1.68 & 0.36\\ 0.36 & 0.72\end{pmatrix}^T\begin{pmatrix}1.68 & 0.36\\ 0.36 & 0.72\end{pmatrix} - I \\ &= \begin{pmatrix}1.952 & 0.864\\ 0.864 & 0.352\end{pmatrix}\end{aligned}$$

It is easy to verify that this 2x2 matrix has eigenvalues 2.24 and -0.64, corresponding to a 80% stretching ($1.8^2 - 1 = 2.24$) and a 40% compression ($0.6^2 - 1 = -0.64$).

4 Physics

The strain tensor gives us a complete description of the deformation of the material, but it doesn't tell us how the material will respond. That is out of the scope of kinematics, and we have to put in some ingredients from physics. Here in this section we'll attempt to write down the energy that arises from the deformation, which governs how the simulation will take place in time. For this purpose, we can proceed in either a mechanics-oriented way, or a mathematics-oriented way.

4.1 The Mathematical Way

Imagine we have somehow obtained the energy function w of strain ϵ, we can take its Taylor expansion:

$$w(\epsilon) = w(0) + \frac{\partial w(0)}{\partial \epsilon}\epsilon + \frac{1}{2}\frac{\partial^2 w(0)}{\partial \epsilon^2}\epsilon^2 + O(\|\epsilon\|^3)$$

w(0) is a constant independent of ϵ and can be conveniently assume to be 0. The linear term is also 0 because the energy should be minimized at zero strain (the definition of "resting shape"). If we adopt the *small strain assumption*, the higher order terms can be omitted since the energy is dominated by the quadratic term. This is basically saying, no matter how complicated the material laws are, under the small strain assumption we can always write down the energy as a quadratic function of the strain.

4.2 The Mechanical Way

Physically, the energy is equal to the product between the stress and the strain:

$$w(\epsilon) = \frac{1}{2}\sum_{i=1}^{2}\sum_{j=1}^{2}\epsilon_{ij}\sigma_{ij}$$

and the stress σ is defined as the elastic moduli times the strain:

$$\sigma_{kl} = \sum_{i=1}^{2}\sum_{j=1}^{2}c_{ijkl}\epsilon_{ij}$$

therefore

$$w(\epsilon) = \frac{1}{2}\sum_{i=1}^{2}\sum_{j=1}^{2}\sum_{k=1}^{2}\sum_{l=1}^{2}c_{ijkl}\epsilon_{ij}\epsilon_{kl}$$

Here we obtain an energy that is quadratic in strain again. c is a fourth-order tensor that encodes all the material properties such as stiffness upon stretching/compression in each direction and shearing etc. Although a generic fourth-order tensor requires a 16 scalars to represent, symmetry in both the strain tensor and the stress tensor reduces this number to 9. If we further assumes that the material is isotropic, then we only need two parameters to fully describe this material: *Young's modulus* E and *Poisson ratio* ν:

$$\begin{pmatrix}\sigma_{xx}\\ \sigma_{yy}\\ \sigma_{xy}\end{pmatrix} = \frac{E}{1-\nu^2}\begin{pmatrix}1 & \nu & 0\\ \nu & 1 & 0\\ 0 & 0 & \frac{1-\nu}{2}\end{pmatrix}\begin{pmatrix}\epsilon_{xx}\\ \epsilon_{yy}\\ 2\epsilon_{xy}\end{pmatrix}$$

4.3 Total Energy

The $w(\epsilon)$ defined above gives the energy density at a point in the material. Since ϵ is a function of undeformed space position $\bar{\mathbf{x}}$, so is w. To obtain the total elastic energy that is stored in an elastic body, we integrate this energy density over the entire body:

$$W = \int_{\Omega} w(\bar{\mathbf{x}})d\Omega$$

For this milestone, we simply assume that the strain is constant for any element (discussed in the next section), therefore the energy is simplified to $W = w(\bar{\mathbf{x}})A$ where A is the 2D element's area.

4.4 Discretization

All the analysis up to now has been performed on a continuum, in the form of differential equations. Continua have an infinite number of degrees of freedom and cannot be manipulated directly in computers - we need to *discretize*. The discretization here is in the spatial dimensions, but it's in the same spirit with the temporal discretization we've implemented in the first Theme with integrators. By spatial discretization we represent the body of interest with a set of nodes, connected by edges, polygon faces or polyhedron volumes, depending on the dimension of the body. These building blocks are called *elements*. There are a fixed number of nodes in each element, and due to the limited amount of information these nodes can provide, we have to assume certain properties on the element itself. For example, a triangle element with only one node in each corner

(one of the simplest and most commonly used element types) simply does not have any information about how the deformation mapping changes over the triangle to the second or higher order, so we have to assume that the deformation Φ is linear, and when we need information about a point inside the triangle, we linearly interpolate between the three nodes. Linear deformation map implies constant deformation gradient, and as a result this element type is called *constant strain triangle* (discussed later in more details). Of course the interpolation won't be able to faithfully reproduce the true property values at an arbitrary point in the element, so a *discretization error* is introduced; but since our hardware technology isn't advanced enough to handle infinite degrees of freedom, this error is inevitable. As long as the discretization error approaches zero as we refine our mesh further and further, we say the discretization *converges* and thus is valid. For our milestone, we do not consider more complicated (but potentially more accurate) elements.

5 Required Features

5.1 Elastic Body Spring Force

The spring force is similar to the spring force we implemented in Theme 1, with the only difference being that we use a shape-independent material property as the stiffness. Following the concept from above, we define the strain as:

$$\epsilon = \frac{\|\mathbf{x}_i - \mathbf{x}_j\| - l_0}{l_0}$$

The energy density is:

$$\begin{aligned} w &= \frac{1}{2}\alpha\epsilon^2 \\ &= \frac{\alpha}{2l_0^2}(\|\mathbf{x}_i - \mathbf{x}_j\| - l_0)^2 \end{aligned}$$

Assuming that the strain is constant along the spring, the total energy is

$$\begin{aligned} W &= wl_0 \\ &= \frac{\alpha}{2l_0}(\|\mathbf{x}_i - \mathbf{x}_j\| - l_0)^2 \end{aligned}$$

Taking the gradient of this energy gives the force. Please implement the computation for energy, force, and hessian in *addEnergyToTotal()*, *addGradEToTotal()* and *addHessEToTotal()* in *ElasticBodySpringForce.cpp*. Note that implementation of the energy is optional; it won't be graded. Implementing the energy will help implementing the force by providing more intuitions and more debugging information.

5.2 Elastic Body Bending Force

When a thin rod is bent, why does it tries to restore its initial straight configuration? It is because the bending introduces differential compression/stretching along the cross section of the rod, and thus increases the elastic energy. If we explicitly model the cross section of the rod, we will have an array of axial springs side-by-side, connected by short and highly stiff radial-directed springs at endpoints. When bent springs on the inner layer are compressed and springs on the outer layer are stretched, therefore creating the restoring force. However this modeling scheme has two serious disadvantages: it is too stiff and has bad aspect ratios, both of which lead to numerical difficulties. In practice we model thin rods as a single polyline, augmented with a bending force to resist bending.

Since the bending energy arises from axial compression and stretching of the material, it is naturally quadratic in the curvature κ of the material centerline:

$$w = \frac{\alpha}{2}\kappa^2$$

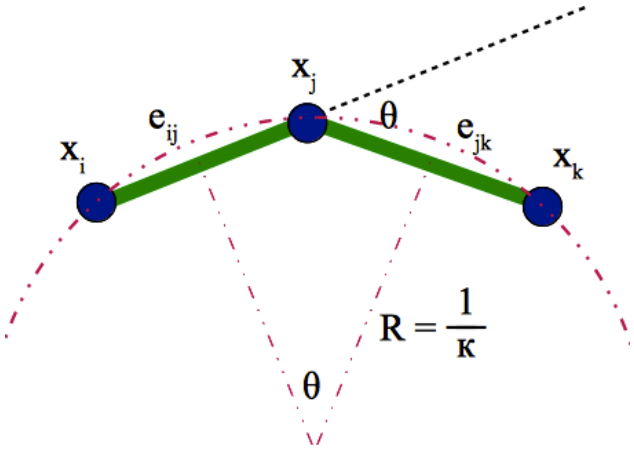


Figure 3: The bending at the common node between two edges

where α captures all other constants, including the material's elasticity modulus and cross section area. As our discretization refines, curvature κ can be approximated by $\frac{\theta}{\|\bar{\mathbf{e}}_{ij}\| + \|\bar{\mathbf{e}}_{jk}\|}$, where θ is the angle by which the next edge ($\mathbf{e}_{ij}$) "deviates" from the direction of the previous edge ($\mathbf{e}_{jk}$):

$$\theta = atan2(\mathbf{e}_{ij} \times \mathbf{e}_{jk}, \mathbf{e}_{ij} \cdot \mathbf{e}_{jk})$$

where function *atan2(y, x)* returns the direction angle of vector (x, y).

Therefore the total bending energy is:

$$\begin{aligned} W &= w(\|\bar{\mathbf{e}}_{ij}\| + \|\bar{\mathbf{e}}_{jk}\|) \\ &= \frac{\alpha}{2}\kappa^2(\|\bar{\mathbf{e}}_{ij}\| + \|\bar{\mathbf{e}}_{jk}\|) \\ &\approx \frac{\alpha}{2}(\frac{\theta}{\|\bar{\mathbf{e}}_{ij}\| + \|\bar{\mathbf{e}}_{jk}\|})^2(\|\bar{\mathbf{e}}_{ij}\| + \|\bar{\mathbf{e}}_{jk}\|) \\ &= \frac{\alpha}{2(\|\bar{\mathbf{e}}_{ij}\| + \|\bar{\mathbf{e}}_{jk}\|)}\theta^2 \end{aligned}$$

Allowing non-straight (non-zero curvature or non-zero bending angle) resting shape, we define the energy as

$$W = \frac{\alpha}{2(\|\bar{\mathbf{e}}_{ij}\| + \|\bar{\mathbf{e}}_{jk}\|)}(\theta - \theta_0)^2$$

Please calculate the gradient and hessian, and implement *addEnergyToTotal(),anddGradEToTotal()* and *addHessEToTotal()* in *ElasticBodyBendingForce.cpp*. Note that implementation of the energy is optional; it won't be graded. Implementing the energy will help implementing the force by providing more intuitions and more debugging information. The following formula may be useful:

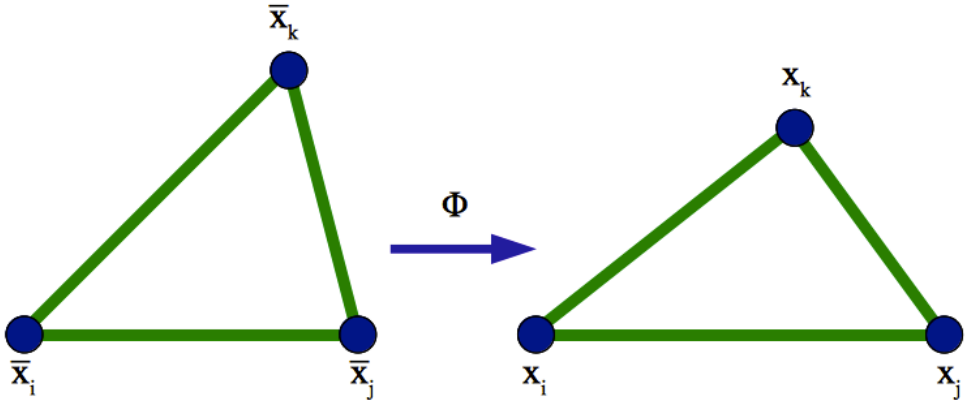


Figure 4: A triangle undergoing a deformation Φ

$$\frac{\partial arc\ tan(x)}{\partial x} = \frac{1}{1+x^2}$$
$$(\frac{\partial a \times b}{\partial a})^T = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} b$$

5.3 Elastic Body Constant Strain Triangle Force

Both the spring force and the bending force essentially assumes an infinitely thin rod-like structure, where the strain is fully captured by a single scalar (either the edge length change or the bending angle). For 2D elastic bodies we need to use the full-fledged treatment in the previous sections. Now consider the triangle involving particle i, j and k:

We assume that the strain is constant over the interior of the triangle (hence the name "constant strain triangle"), and so the deformation map Φ is affine (without loss of generality, from here on we assume i = 1, j = 2 and k = 3):

$$\mathbf{x} = \nabla\Phi(\bar{\mathbf{x}} - \bar{\mathbf{x}}_1) + \mathbf{x}_1$$
$$\nabla\Phi = \begin{pmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{pmatrix} \begin{pmatrix} \bar{x}_2 - \bar{x}_1 & \bar{x}_3 - \bar{x}_1 \\ \bar{y}_2 - \bar{y}_1 & \bar{y}_3 - \bar{y}_1 \end{pmatrix}^{-1}$$

where x_i denotes the x component of node $\mathbf{x}_i$, y_i denotes its y component, and the overhead bar denotes the undeformed configuration as usual. Denoting the components in $\nabla\Phi$ as $\nabla\Phi = \begin{pmatrix} a & c \\ b & d \end{pmatrix}$, we define the strain:

$$\begin{aligned}\epsilon &= \begin{pmatrix}\epsilon_{xx} & \epsilon_{xy}\\ \epsilon_{xy} & \epsilon_{yy}\end{pmatrix}\\ &= \nabla\Phi^T\nabla\Phi - I\\ &= \begin{pmatrix}a^2+b^2-1 & ac+bd\\ ac+bd & c^2+d^2-1\end{pmatrix}\end{aligned}$$

and the energy:

$$\begin{aligned}W &= \frac{\bar{A}}{2}\begin{pmatrix}\sigma_{xx}\\ \sigma_{yy}\\ \sigma_{xy}\end{pmatrix}^T\begin{pmatrix}\epsilon_{xx}\\ \epsilon_{yy}\\ 2\epsilon_{xy}\end{pmatrix}\\ &= \frac{\bar{A}}{2}(\frac{E}{1-\nu^2}\begin{pmatrix}1 & \nu & 0\\ \nu & 1 & 0\\ 0 & 0 & \frac{1-\nu}{2}\end{pmatrix}\begin{pmatrix}\epsilon_{xx}\\ \epsilon_{yy}\\ 2\epsilon_{xy}\end{pmatrix})^T\begin{pmatrix}\epsilon_{xx}\\ \epsilon_{yy}\\ 2\epsilon_{xy}\end{pmatrix}\\ &= \frac{E\bar{A}}{2(1-\nu^2)}[\epsilon_{xx}^2+2\nu\epsilon_{xx}\epsilon_{yy}+\epsilon_{yy}^2+2(1-\nu)\epsilon_{xy}^2]\end{aligned}$$

where $\bar{A}$ is the undeformed triangle area:

$$\bar{A} = \frac{1}{2}\|(\bar{\mathbf{x}}_3-\bar{\mathbf{x}}_1)\times(\bar{\mathbf{x}}_2-\bar{\mathbf{x}}_1)\|$$

In order to find the force, we take the gradient of the energy above:

$$\begin{aligned}-F &= \nabla_{\mathbf{x}}W\\ &= \frac{E\bar{A}}{1-\nu^2}[(\epsilon_{xx}+\nu\epsilon_{yy})\nabla_{\mathbf{x}}\epsilon_{xx}+(\epsilon_{yy}+\nu\epsilon_{xx})\nabla_{\mathbf{x}}\epsilon_{yy}+2(1-\nu)\epsilon_{xy}\nabla_{\mathbf{x}}\epsilon_{xy}]\\ \nabla_{\mathbf{x}}\epsilon_{xx} &= 2a\nabla_{\mathbf{x}}a+2b\nabla_{\mathbf{x}}b\\ \nabla_{\mathbf{x}}\epsilon_{yy} &= 2c\nabla_{\mathbf{x}}c+2d\nabla_{\mathbf{x}}d\\ \nabla_{\mathbf{x}}\epsilon_{xy} &= a\nabla_{\mathbf{x}}c+c\nabla_{\mathbf{x}}a+b\nabla_{\mathbf{x}}d+d\nabla_{\mathbf{x}}b\end{aligned}$$

Now letting

$$\begin{pmatrix}\bar{x}_{2x}-\bar{x}_{1x} & \bar{x}_{3x}-\bar{x}_{1x}\\ \bar{x}_{2y}-\bar{x}_{1y} & \bar{x}_{3y}-\bar{x}_{1y}\end{pmatrix}^{-1} = \begin{pmatrix}e & g\\ f & h\end{pmatrix}$$

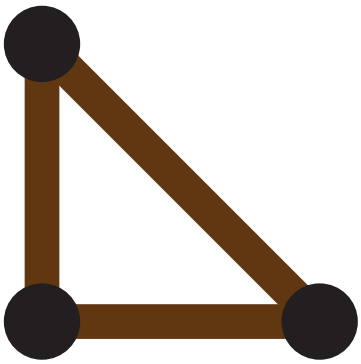
we have

$$\begin{aligned}\nabla\Phi &= \begin{pmatrix}(-e-f)x_1+ex_2+fx_3 & (-g-h)x_1+gx_2+hx_3\\ (-e-f)y_1+ey_2+fy_3 & (-g-h)y_1+gy_2+hy_3\end{pmatrix}\\ \nabla a &= \begin{pmatrix}-e-f\\ 0\\ e\\ 0\\ f\\ 0\end{pmatrix}, \nabla b = \begin{pmatrix}0\\ -e-f\\ 0\\ e\\ 0\\ f\end{pmatrix}, \nabla c = \begin{pmatrix}-g-h\\ 0\\ g\\ 0\\ h\\ 0\end{pmatrix}, \nabla d = \begin{pmatrix}0\\ -g-h\\ 0\\ g\\ 0\\ h\end{pmatrix},\end{aligned}$$

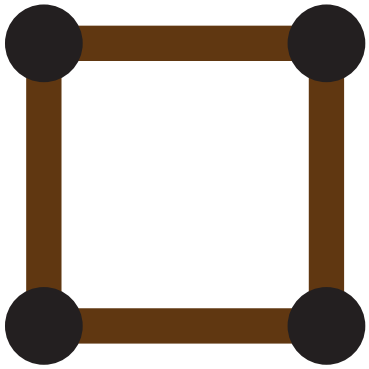
With these derivation, you should be able to implement the elastic energy, force, and hessian of a constant strain triangle. Please do so in *addEnergyToTotal()*, *addGradEToTotal()*, and *addHessEToTotal()* in *ElasticBodyCSTForce.cpp*. Note that implementation of the energy is optional; it won't be graded. Implementing the energy will help implementing the force by providing more intuitions and more debugging information.

Debugging help

If you have problems implementing the elasticity, try debugging a few simple cases where you know the exact solution. Then test the output of your algorithm for the deformation of each point to the exact solution that you know to be true.
An easy example to use for this is the triangle with vertices $(0,0)$, $(0,1)$ and $(1,0)$ (ref. the first image below). Another simple test case is the square with vertices $(0,0)$, $(0,1)$, $(0,1)$ and $(1,1)$ (ref. the second image below)



A triangle with vertices $(0,0)$, $(0,1)$ *and* $(1,0)$



A square with vertices $(0,0)$, $(0,1)$, $(0,1)$ *and* $(1,1)$

6 Creative Scene

As part of your final submission for this milestone, please include a scene of your design that best shows off your program. Your scene will be judged by a secret of committee of top scientists using the highly refined criteria of:

1. How well the scene shows off this milestone's "magic ingredients" (a la Iron Chef).
2. Aesthetic considerations. The more beautiful, the better.
3. Originality.

Top examples will be posted to the discussion board. Please remember to generate this scene before submitting, as the Codio box will become read-only after you submit.

6.1 Making Movies

Please remember to generate this scene before submitting, as the Codio box will become read-only after you submit. The FOSSSim starter code comes with a PNG outputting utility that can help you create movies from your simulation. To enable it, create a folder named "pngs" in the directory that you are calling your executable from (i.e. your current directory) and run your code with the

```
-g 1
```

flag enabled. This will prompt your program to automatically save a PNG file for each frame of the simulation. This won?t work if the "pngs" folder does not exist so you need to create it first before you run the program.

After the simulation finishes, you'll find all the frames in the pngs folder. Then you can make videos out of them with command-line tools such as mencoder and ffmpeg. Both mencoder and ffmpeg are easy-to-use tools available on the Codio boxes. You can execute this command:

```
mencoder mf://pngs/*.png -mf fps=24 -ovc lavc -lavcopts vcodec=msmpeg4v2 -oac copy -o output.avi
```

followed by:

```
ffmpeg -i output.avi -vcodec libx264 -crf 25 output.mp4
```

in order to create an mp4 video you may upload for the Creative portion of the assignment.

Note, ffmpeg can be used to create a video from the png images in one step, although sometimes this will give an error depending on the count and dimensions of input files. Nevertheless, the following command can take pngs and convert them to an mp4 movie in one:

```
ffmpeg -r 24 -f image2 -i ./pngs/frame%05d.png -vcodec libx264 -crf 25 -pix_fmt yuv420p test.mp4
```

Lastly, you may preview your movie by running the following command and looking at the virtual desktop:

```
mplayer <movie_name>
```

Please submit the mp4 file to the Peer Review Assignment portion of this week. An explanation for arguments to both mencoder and ffmpeg can be found online.