SOFT-BODY SIMULATION WITH THE FINITE-ELEMENT METHOD

AMIR VAXMAN, GAME PHYSICS (INFOMGP), PERIOD 3, 2018/2019

Last Update: 1/Apr/2019

Errata

Equation 13 should have μ on the latter three diagonal elements instead of λ (fixed in this version).

1. Introduction

These course notes will explain in details the finite-element approach to soft-body simulation. These notes will assume some knowledge of the continuous formulation, although such knowledge is not strictly necessary to reproduce a simulation with what is described here.

We work on a tetrahedral mesh $\mathbf{M} = \{\mathbf{V}, \mathbf{T}\}$, where the vertices $\mathbf{V}$ have rest positions in space $\mathbf{x_0} \in \mathbb{R}^{|\mathbf{V}|\times 3}$. The deformation of the mesh is described as new positions $\mathbf{x}(t)$, or alternatively a deformation field $\mathbf{u}(t) = \mathbf{x}(t) - \mathbf{x}_0$. The process of simulation computes the internal forces resulting from the deformation of the object with relation to its rest state and material properties, and integrates velocities and new deformation fields accordingly. we denote each tetrahedron by $e \in T$, short for "element" (rather that "t" so to not confuse it with the time variable).

2. Finite-Element Basics

A basic assumption in linear FEM is that functions on the mesh are defined as scalars on vertices, and interpolated linearly inside elements (in our case, tetrahedra). More formally, a function is represented as vector $f : \mathbf{V} \rightarrow \mathbf{R}$. The function values $f(p)$ for a point p inside tetrahedron e_{1234} are defined as:

$$f(p) = \sum_{i=1}^{4} B_i(p) f_i. \tag{1}$$

The functions $B_{1\ldots 4}(p)$, each associated to a single vertex and defined over the tetrahedron, are called *barycentric coordinates*, and encode the linear blending. To make that happen, they have the following properties:

- The hold the *Lagrange* property, that is:

$$B_i(v_j) = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

This property is necessary for interpolation, in order to get $f(v_i) = f_i$.

- They are linear inside the tetrahedron.

Due to the two properties above, the blending functions are also unique for tetrahedra in $\mathbf{R}^3$. That is, the properties above define them unambiguously.

2.1. **Piecewise-Linearity.** A function f that is interpolated in that manner, over each individual element, and consequently over the entire mesh, it is denoted as *piecewise-linear* (in short: PL). Consider a face f_{123} adjacent to two tetrahedra e_{1234} and e_{1235}. Due to the Lagrange property, vertices 4 and 5 have no effect on the function values at points $p \in f_{123}$, and they are a linear blend of f_1, f_2, f_3 alone. It is evident that the function is then continuous across elements. This logic follows again for function values on edges (v_1, v_2) that are a linear blend on the edge of values f_1 and f_2 and nothing else.

2.2. **Function Gradients.** The gradient of $f(p)$ at a point $p \in e_{1234}$ can then be simply written as:

$$\nabla f(p) = \sum_{i=1}^{4} \nabla B_i(p) f_i. \tag{2}$$

That is, the gradient $\nabla f(p)$ is a blend of the barycentric gradients $\nabla B_i(p)$. In fact, since the functions B are piecewise-linear, the gradients $\nabla B_i(p)$ are *piecewise-constant* (in short: PC). For intuition about how the gradient vectors look like, consider $B_1(p)$: since it 0 on the face f_{234}, then its gradient must be in the direction of the face normal $\hat{n}_{123}$; moreover, consider the perpendicular segment from v_1 to the face f_{234}, with height (length) h_1; the function changes on that height between 0 (on the face) on 1 (on the vertex) linearly, and the slope—or the magnitude of the gradient— is then $\frac{1}{h_1}$. Mark the area of f_{123} as A_{123}. Then, we have that $Vol\left(e\right) = \frac{h_1}{3} A_{234}$, and in addition: $\hat{n}_{234} = \frac{(v_3 - v_2) \times (v_4 - v_2)}{2A_{123}}$ (assuming the face is oriented counterclockwise towards inside the tetrahedra). Then we get:

$$\nabla B_1(p) = \frac{\hat{n}_{234}}{h_1} = \frac{v_{24} \times v_{23}}{2A_{234}} \cdot \frac{A_{234}}{3Vol\left(e\right)} = \frac{v_{23} \times v_{24}}{6Vol\left(e\right)} \tag{3}$$

Gradient Matrix. Since the gradients ∇B are constant, we can in fact get the constant gradient of a vertex-based function f inside tetrahedron e as follows (assume the gradients ∇B are columns vectors of 3×1:

$$\nabla_e f = \begin{pmatrix} \nabla B_1 & \nabla B_2 & \nabla B_3 & \nabla B_4 \end{pmatrix} \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{pmatrix} = G_e \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{pmatrix}. \tag{4}$$

where G_e is the *gradient matrix* of tetrahedron e. This demonstrates one of the core paradigms in finite-element formulations: they convert derivations into linear operations (=matrix multiplication), and differential equations into linear systems.

Constant and linear reproduction. Since the function is interpolated linearly from vertices inside tetrahedra, then it only makes sense that if the f_i values sample some ambient linear function in space, then that we get perfect reproduction of that function by this interpolation. Formally, consider a continuous linear function $g(x, y, z) : \mathbb{R}^3 \to \mathbb{R}$ so that $g(x, y, z) = ax + by + cz + d$. Sample the function at the vertices: $f_i = g(v_i)$. Then, for $p = (x, y, z)$ inside a tetrahedron e, we get:

$$f(p) = f(x, y, z) = \sum_{i=1}^{4} B_i(p) f_i = g(p). \tag{5}$$

And as a consequence we also get $\nabla f = (a, b, c)$ inside e. Note that ∇f is indeed constant. This property is called *linear reproduction*. Note that when $g = d$ is constant, we also get a constant f of the same value (and $\nabla f = 0$); this property is known as the weaker *constant reproduction*.

We can use linear reproduction to compute the gradient matrix G_e in an elegant way without resorting to the geometric formulation of Equation 3. Consider the function $f_i = x(v_i)$ (picking the x coordinate of the vertex), that gets interpolated into $f(x, y, z) = x$ in every inner point, and consequently $\nabla f = (1, 0, 0)$. Doing the same for all y and z, and for the constant function $f_i = 1$, we reach the following linear relation:

$$G_e \begin{pmatrix} 1 & x_1 & y_1 & z_1 \\ 1 & x_2 & y_2 & z_2 \\ 1 & x_3 & y_3 & z_3 \\ 1 & x_4 & y_4 & z_4 \end{pmatrix} = G_e P_e = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & I_{3\times 3} \end{pmatrix} \tag{6}$$

And we can obtain the gradient matrix as:

$$G_e = \begin{pmatrix} 0 & I_{3\times 3} \end{pmatrix} P_e^{-1}.$$

Which is equivalent to G_e being P_e^{-1} with the first row removed.

3. The FEM model of deformation

3.1. **Deformation Jacobians.** We model the deformation field $\mathbf{u}$ as a single vector per vertex. As such, the location $\mathbf{x}(\mathbf{t})$ is also naturally assigned per vertex. We treat such vectors as three different scalar functions x, y, z for the three coordinates. With this, we can easily define the Jacobian within each tetrahedron e:

$$\begin{pmatrix} \nabla u_x \\ \nabla u_y \\ \nabla u_z \end{pmatrix}_e = \begin{pmatrix} G_e & & \\ & G_e & \\ & & G_e \end{pmatrix} \begin{pmatrix} u_{x,1} \\ u_{x,2} \\ \cdots \\ u_{z,3} \\ u_{z,4} \end{pmatrix} = J_e u_e \tag{7}$$

The matrix J_e inputs the $3 \times 4 = 12$ deformation vectors on the tetrahedral vertices, and produces the $3 \times 3 = 9$ gradient coefficients for each deformation coordinate.

3.1.1. *About indexing.* Note that the arrangement in the equation (coordinate-dominant order $xxxyyyzzz$) is made so that the matrix J_e can be written with blocks of G_e per coordinate; in practice (and in your practical), it is often the case that the deformation vectors are expressed in vertex-dominant order (coordinates given in xyz order sequentially per vertex); it is comfortable to build J_e in the coordinate-dominant order per tetrahedron, and multiply it with *permutation matrices* that rearrange vectors from one order to another.

3.2. **The strain tensor.** The strain tensor $\boldsymbol{\epsilon}$ is a symmetric tensor expressed by a symmetric 3×3 matrix. That means that it has only 6 independent elements: the diagonal elements $\epsilon_{xx}, \epsilon_{yy}, \epsilon_{zz}$ and the off-diagonal elements $\epsilon_{xy}, \epsilon_{yz}, \epsilon_{zx}$. In matrix form, we use the linear elasticity (or: infinitesimal strain) approximation:

$$\boldsymbol{\epsilon} = \frac{1}{2}\left(Ju + Ju^T\right). \tag{8}$$

In the FEM deformation model, since the Jacobian is piecewise constant, the stress tensor is PC as well. We can write it in concise form:

$$\boldsymbol{\epsilon}_e = \begin{pmatrix} \epsilon_{xx} \\ \epsilon_{yy} \\ \epsilon_{zz} \\ \epsilon_{xy} \\ \epsilon_{yz} \\ \epsilon_{zx} \end{pmatrix}_e = DJ_e u_e, \tag{9}$$

where D is a matrix that adds and rearranges the elements of $J_e u_e$ so that they fit to the respective values of $\boldsymbol{\epsilon}$. We have that:

$$J_e u = \begin{pmatrix} \frac{\partial u_x}{\partial x} \\ \frac{\partial u_x}{\partial y} \\ \frac{\partial u_x}{\partial z} \\ \frac{\partial u_y}{\partial x} \\ \frac{\partial u_y}{\partial y} \\ \frac{\partial u_y}{\partial z} \\ \frac{\partial u_z}{\partial x} \\ \frac{\partial u_z}{\partial y} \\ \frac{\partial u_z}{\partial z} \end{pmatrix}_e = \begin{pmatrix} \nabla u_x \\ \nabla u_y \\ \nabla u_z \end{pmatrix}_e \tag{10}$$

And we need to get for $\boldsymbol{\epsilon}$:

$$\begin{aligned}
\epsilon_{xx} &= \frac{\partial u_x}{\partial x} \\
\epsilon_{yy} &= \frac{\partial u_y}{\partial y} \\
\epsilon_{zz} &= \frac{\partial u_z}{\partial z} \\
\epsilon_{xy} &= \frac{1}{2}\left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x}\right) \\
\epsilon_{yz} &= \frac{1}{2}\left(\frac{\partial u_y}{\partial z} + \frac{\partial u_z}{\partial y}\right) \\
\epsilon_{zx} &= \frac{1}{2}\left(\frac{\partial u_z}{\partial x} + \frac{\partial u_x}{\partial z}\right)
\end{aligned} \tag{11}$$

As such, we get that:

$$D = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0.5 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0.5 & 0 \\
0 & 0 & 0.5 & 0 & 0 & 0 & 0.5 & 0 & 0
\end{pmatrix} \tag{12}$$

Note that D is purely made out of combination and addition values and therefore does not depend on anything in the geometry of e. However, we can combine the matrices to get the strain as a direct function of deformation:

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

.

For sanity check, B_e is of dimensions 6×12, from deformation vectors to strain coefficients.

3.3. **The stress tensor.**

The stress tensor is related to the strain tensor by the *stiffness tensor*:

$$\begin{pmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{zz} \\ \sigma_{xy} \\ \sigma_{yz} \\ \sigma_{zx} \end{pmatrix} = C \begin{pmatrix} \epsilon_{xx} \\ \epsilon_{yy} \\ \epsilon_{zz} \\ \epsilon_{xy} \\ \epsilon_{yz} \\ \epsilon_{zx} \end{pmatrix} = \begin{pmatrix}
\lambda + 2\mu & \lambda & \lambda & & & \\
\lambda & \lambda + 2\mu & \lambda & & & \\
\lambda & \lambda & \lambda + 2\mu & & & \\
& & & \mu & & \\
& & & & \mu & \\
& & & & & \mu
\end{pmatrix} \begin{pmatrix} \epsilon_{xx} \\ \epsilon_{yy} \\ \epsilon_{zz} \\ \epsilon_{xy} \\ \epsilon_{yz} \\ \epsilon_{zx} \end{pmatrix}, \tag{13}$$

where the $\boldsymbol{\sigma}$ values are piecewise constant per tetrahedron, paralleling those of $\boldsymbol{\epsilon}$. λ and μ are the material coefficients known as the *Lamé parameters*. They are computed as:

$$\mu=\frac{Y}{2\left(1+\nu\right)},\ \lambda=\frac{\nu Y}{\left(1+\nu\right)\left(1-2\nu\right)}. \tag{14}$$

Y is Young's modulus (1D stress for change in length), and is usually very large: in the Giga Pascals (Gpa) or Mega Pascals (MPa). ν is Poisson's ratio (the transversal compression to axis extension rate and vice versa; or rather, the compressibility of the material). ν is then between $[0,0.5)$, where $\nu=0.5$ is perfect incompressibility, as expected from a perfectly rigid body, and therefore not a physical actuality. Note that this would lead to an infinite stress for every strain.

3.4. **The strain energy.** The strain energy is the potential energy within the material that is gained from creating strain. For the entire mesh M, It is defined as:

$$U_M=\frac{1}{2}\int_M\langle\boldsymbol{\sigma},\boldsymbol{\epsilon}\rangle dV=\frac{1}{2}\int_M\langle C\boldsymbol{\epsilon},\boldsymbol{\epsilon}\rangle dV. \tag{15}$$

For intuition on this definition, consider the simplest deformable body: the spring. The potential energy within the spring is defined (remember the first lecture) as the integrated force applied per distance, or formally: $U_{spring}=\int_0^{\Delta x}F\cdot dx$. Since we have $F=k\Delta x$, we get $U_{spring}=\frac{1}{2}k\Delta x^2$. Since x is the strain, and k is the (1D) stiffness tensor, we get the same.

Note that there is no minus sign to the force! this is the force to change the spring *away* from its rest state, not the force the spring exerts to return to rest state, as we learned. Intuitively, but not trivially, they are of opposite signs. More on that in the next section.

In the discrete setting, and as $\boldsymbol{\epsilon}$ and $\boldsymbol{\sigma}$ are piecewise constant, U_M is actually a sum over the energies of each tetrahedron, in the following manner:

$$U_M=\sum_{e\in T}U_e=\sum_{e\in T}\frac{1}{2}Vol(e)\cdot\boldsymbol{\sigma}_e\cdot\boldsymbol{\epsilon}_e \tag{16}$$

The $Vol\left(e\right)$ factor comes from the trivial integration of the constant $\boldsymbol{\sigma}_e\cdot\boldsymbol{\epsilon}_e$ per tetrahedron.

We would like to express the energy component per tetrahedron using only the deformation fields. In that case, we get:

$$\frac{1}{2}Vol\left(e\right)\cdot\boldsymbol{\sigma}_e\cdot\boldsymbol{\epsilon}_e=\frac{1}{2}Vol\left(e\right)\cdot u_e^T\left(Be\right)^TCB_eu_e=\frac{1}{2}u_e^TK_eu_e,$$

where K_e is the *stiffness* matrix of the tetrahedron, and is sized 12×12 (taking the local deformation vectors on both sides. the matrix K_e is in fact *symmetric positive definite* (SPD): for every u_e, we get $u_e^TK_eu_e\geq0$, where we get strict 0 for constant u alone (creating no deformation). We can then aggregate the stiffness matrices and define the energy on the entire mesh using the entire deformation field:

$$U_M=\frac{1}{2}u^TKu.$$

where now K is of size $3|V| \times 3|V|$.

3.4.1. *Creating K.* While creating K_e for each element might be relatively simple, creating the global K can be tricky, in sense of "which index goes where". The best advice is to create K not directly, but as a product of matrices, each more comfortable to make. Consider matrix Q to be the permutation matrix that does the following operation:

$$\begin{pmatrix} u_{e,1} \\ u_{e,2} \\ \dots \end{pmatrix} = Q \begin{pmatrix} u_{v_1} \\ u_{v_2} \end{pmatrix}$$

In words, arranging the global vertex-dominant order we use to encode to total deformation field, and where each u_{v_m} is the xyz coordinates of vertex v_m, into a comfortable element-dominant order, where each u_{e_m} is the 12 coordinates of the deformation field in each tet, in sequence. Note that the column vector resulting from the vertex-dominant vector is much smaller than the element-dominant column vector, as the deformation vector in each vertex gets copied to each elements it is adjacent to.

Consequently, the matrix Q^T takes element-dominant fields into vertex-dominant again, where the reduction is by summing all the values contributed to a vertex from all the adjacent tets.

For the element-dominant order, the stiffness matrix has a very simple form: this K' is simply a $12|T| \times 12|T|$ huge matrix made of 12×12 block matrices of each K_e on its main diagonal—each operating individually on every element independently of the others. Finally, we get:

$$K = Q^T K' Q$$

.

3.5. **Deformation forces.** Out next step is to figure out the forces that result from the object trying to get back to its rest state, denoted as *internal forces*. Such forces are by definition in the direction that minimizes the strain energy—think about free falling minimizing the height potential energy as fast as possible. For reference, this principle is called *Hamilton's principle*, and a special case of it, *Fermat's principle* tells us that light always travels the shortest route.

The direction that minimizes the energy the most is the negative gradient. As such, the internal forces are simply:

$$f_{int} = -Ku \tag{17}$$

We can also look at the geometric interpretation of K this way: it is a sort of *deformation Laplacian*, measuring how much the u at each vertex is different from its vertex neighbors, where the result gives back the force needed to equalize the deformation so it doesn't deform the rest state. If and only if $u = const$ (in every coordinate individually), we get $f = 0$.

3.6. **The soft-body equation of motion.** We can now assemble the final equation of motion for the finite-element deformation model:

$$Ma(t) + Dv(t) + K\left(x(t) - x(0)\right) = f_{ext}, \tag{18}$$

where:

(1) M is the $3|V| \times 3|V|$ *mass matrix*, which is a diagonal matrix with mass value per coordinate in $a(t)$ (that means repeating the mass of each vertex three times).
(2) D is the $3|V| \times 3|V|$ *damping matrix* which slows down (damps) the movement. It is an abstraction of physical damping, which is mostly here for the sake of stabilizing the discrete simulation.
(3) Note that the external forces is on the right-hand side, while the internal forces are on the left-hand side (but with the proper sign, as $f_{int} = -Ku$): that is because the equation solves for the deformation at every time step, while external forces are supposedly independent (like gravity).

3.6.1. *The mass matrix.* We need a notion of the "mass" of each vertex. Given the density ρ of the vertex (often uniform for the entire mesh), the mass element is $m = \rho dV$, so we need a notion of the local volume dV of a vertex. For this, we define the notion of *Voronoi area*. In fact, we will use a simpler version of it—the Barycentric area. It is defined by dividing each tetrahedron in equal four parts, and contributing each quarter to each of its vertices.

3.6.2. *The damping matrix.* Damping matrices are a huge subject of its own, and they are usually designed to subdue unwanted high-frequency movement that results from the discretization, while not harming the physical accuracy so much. For our purposes, we use the well-known *Rayleigh dampening*, where we set $D = \alpha M + \beta K$. A reasonable setting of the damping parameters α and β is between 0 and 0.02.

Note that our matrices D, M, K are all sparse: they only have values that deformation vectors contribute to their neighbors, and the rest is zeros. Moreover, they are all symmetric semi-positive definite; we will use that when we design the time-step solutions in the next section.

4. Simulating the equation motion

Our modus operandi is to include the soft-body solver within the velocity and position integration step in the game engine. Then is is combined with later correction of velocity and position by any other method, due to constraints and collisions.

4.1. **Time Integration.** The integration method that is best fit for this type of simulation is the backward implicit Euler method. Namely, we solve for:

$$\begin{aligned} v(t+\Delta t) &= v(t) + \Delta t a(t+\Delta t), \\ x(t+\Delta t) &= x(t) + \Delta t v(t+\Delta t). \end{aligned} \tag{19}$$

Assigning these in the system, we get:

$$M\frac{v\left(t+\Delta t\right)-v\left(t\right)}{\Delta t}+Dv\left(t+\Delta t\right)+K\left(x\left(t+\Delta t\right)-x\left(0\right)\right)=f_{ext}\Rightarrow \tag{20}$$
$$Mv\left(t+\Delta t\right)-Mv\left(t\right)+\Delta tDv\left(t+\Delta t\right)+\Delta tK\left(x\left(t\right)+\Delta tv\left(t+\Delta t\right)-x\left(0\right)\right)=\Delta tf_{ext}\Rightarrow$$
$$\left[M+\Delta tD+\Delta t^2K\right]v\left(t+\Delta t\right)=Mv\left(t\right)-\Delta t\left[K\left(x\left(t\right)-x\left(0\right)\right)-f_{ext}\right].$$

We then have to solve a large $(3\left|V\right|\times 3\left|V\right|)$ linear system of the form $Av\left(t+\Delta t\right)=b$ in every time step. This seems much more complicated then the simple integration methods we had for rigid body. However, as we see next, the structure of this system can be exploited to create cheap real-time solutions.

4.2. **Cholesky factorization.** Note that $A=\left[M+\Delta tD+\Delta t^2K\right]$ is not time dependent. That is, we can manipulate it in the beginning of time to facilitate the solution at each time step (or when a new time step interval Δt is chosen, which is hopefully not often). Moreover, A is symmetric and positive definite. As such, it can be *factorized* in the beginning of time to the form:

$$A=LL^T, \tag{21}$$

where L is a *lower-triangular* matrix, and its transpose is consequently upper-triangular. Solving linear systems with triangular matrices is very easy, since we can just solve them the equations by one by one; this is called *forward substitution* for lower-triangular matrices, and *back substitution* for upper triangular ones. For a given right-hand side $b=Mv\left(t\right)-\Delta t\left[K\left(x\left(t\right)-x\left(0\right)\right)-f_{ext}\right]$, we then need to solve two systems at each time step (the "$(t+\Delta t)$" is omitted):

$$\begin{aligned}Ly&=b\\L^Tv&=y\end{aligned} \tag{22}$$

Having pre-factorized A to L, these two system are cheap even as b is time-varying.

4.2.1. *Sparse Matrices.* The second concern is whether the system is too huge to fit in memory. Luckily, as the matrices are sparse, and as Cholesky factorization preserves sparsity, we never run into any memory problems.