Simulating Cloth Using Bilinear Elements (Supplement)

Eston Schweickart
Weta Digital
Wellington, New Zealand

Xiao Zhai
Weta Digital
Wellington, New Zealand

ACM Reference Format:
Eston Schweickart and Xiao Zhai. 2021. Simulating Cloth Using Bilinear Elements (Supplement). In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Talks (SIGGRAPH '21 Talks), August 09-13, 2021.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3450623.3464675

1 DERIVATION OF MEMBRANE ENERGY

In the following section, we describe a membrane deformation energy based on [Volino et al. 2009] that is consistent for both triangle elements and bilinear elements. The following section assumes we have an isotropic stretching model; an anisotropic extension is discussed in section 3.

Following [Volino et al. 2009], we assume that we have a mapping from a 2D material space to 3D world space for each element. From this mapping, we can calculate a 3×2 deformation gradient $\boldsymbol{F}$:

$$\boldsymbol{F} = \left[\boldsymbol{U} | \boldsymbol{V}\right]$$

The authors describe how to compute $\boldsymbol{U}, \boldsymbol{V} \in \mathbb{R}^3$ from the positions of the vertices of each triangle as well as material space coordinates. We make a slight modification: we assume that we have a rest mesh in world space, which is more convenient for artists to specify. Let $\boldsymbol{p}_i$ be the world space positions of an element, with corresponding rest positions $\overline{\boldsymbol{p}}_i$, where $i \in [0, 1, 2]$ if the element is a triangle or $i \in [0, 1, 2, 3]$ if the element is a quadrilateral. In the latter case, we define the associated bilinear element as:

$$\boldsymbol{p}(s,t) = (1-s)(1-t)\boldsymbol{p}_0 + (s)(1-t)\boldsymbol{p}_1 + (s)(t)\boldsymbol{p}_2 + (1-s)(t)\boldsymbol{p}_3$$

To construct a deformation gradient between rest space and world space, we find a mapping between the associated tangent spaces. We first define linear bases for each tangent space, represented by 3×2 matrices $\boldsymbol{T}$ and $\overline{\boldsymbol{T}}$. For triangles, we define:

$$\boldsymbol{T} = \left[(\boldsymbol{p}_1 - \boldsymbol{p}_0) | (\boldsymbol{p}_2 - \boldsymbol{p}_0)\right]$$

and similarly for $\overline{\boldsymbol{T}}$. For bilinear elements, the most natural choice is:

$$\boldsymbol{T}(s,t) = \left[\frac{\partial \boldsymbol{p}}{\partial s}(t) | \frac{\partial \boldsymbol{p}}{\partial t}(s)\right]$$

and similarly for $\overline{\boldsymbol{T}}(s,t)$. In either case, we can define the deformation gradient between the spaces by:

$$\boldsymbol{F} = \left[\boldsymbol{U} | \boldsymbol{V}\right] = \boldsymbol{T}\overline{\boldsymbol{R}}^{-1}$$

where $\overline{\boldsymbol{Q}}\overline{\boldsymbol{R}} = \overline{\boldsymbol{T}}$ is the thin $\boldsymbol{QR}$-decomposition of the rest tangent space matrix, i.e., $\overline{\boldsymbol{R}}^{-1}$ is a lower-triangular 2×2 matrix that orthonormalizes the rest tangent frame.

With these definitions, we return to the membrane deformation energy:

$$E_m = \frac{h}{2} \int_{\overline{\Omega}} \boldsymbol{\varepsilon}_m^T \boldsymbol{K} \boldsymbol{\varepsilon}_m d\overline{A}$$

where h is the shell thickness, $\overline{\Omega}$ is the rest surface with area element $\overline{A}$, ε_m is the membrane strain, and $\boldsymbol{K}$ is a stiffness matrix. Following the linear thin shell model, $\boldsymbol{K}$ can be defined from Young's Modulus E and Poisson's Ratio ν as follows:

$$\boldsymbol{K} = \frac{E}{1-\nu^2}\begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{bmatrix}$$

We define $\boldsymbol{\varepsilon}_m$ to be the Voigt form of the St. Venant-Kirchhoff strain tensor:

$$\boldsymbol{\varepsilon}_m = \text{Voigt}\left(\frac{1}{2}(\boldsymbol{F}^T\boldsymbol{F} - \boldsymbol{I})\right) = \begin{bmatrix} \frac{1}{2}(\boldsymbol{U}^T\boldsymbol{U} - 1) \\ \frac{1}{2}(\boldsymbol{V}^T\boldsymbol{V} - 1) \\ \boldsymbol{U}^T\boldsymbol{V} \end{bmatrix}$$

This is straightforward to evaluate for linear surface elements with constant deformation gradients, but $\boldsymbol{\varepsilon}_m$ varies over a bilinear patch. In this case, we can instead approximate the energy over these faces using Gaussian quadrature:

$$E_m \approx \frac{h}{2} \sum_{q \in Q} \sum_{i=0}^{N_{qp}} g_i \boldsymbol{\varepsilon}_m(s_i,t_i)^T \boldsymbol{K} \boldsymbol{\varepsilon}_m(s_i,t_i) \left\| \frac{\partial \overline{\boldsymbol{p}}}{\partial s}(t_i) \times \frac{\partial \overline{\boldsymbol{p}}}{\partial t}(s_i) \right\|$$

where Q is the set of quadrilaterals in the mesh, and N_{qp} is the number of quadrature points with weights g_i and abscissae (s_i, t_i).

At this point it is helpful to take the Taylor expansions of the tangents at the center of the patch:

$$\frac{\partial \boldsymbol{p}}{\partial s}(t) = \frac{\partial \boldsymbol{p}}{\partial s}\left(\frac{1}{2}\right) + \left(\frac{1}{2} - t\right)\frac{\partial^2 \boldsymbol{p}}{\partial s \partial t}$$

$$\frac{\partial \boldsymbol{p}}{\partial t}(s) = \frac{\partial \boldsymbol{p}}{\partial t}\left(\frac{1}{2}\right) + \left(\frac{1}{2} - s\right)\frac{\partial^2 \boldsymbol{p}}{\partial s \partial t}$$

This gives us a definition of the tangents as a linear combination of vectors that are constant per patch. U and V are linear functions of these vectors, so the entries of $\boldsymbol{\varepsilon}_m$ can be written as linear functions of their dot products. We omit the full formulas here for brevity, but this implies that we can expand $\boldsymbol{\varepsilon}_m$ as follows:

$$\boldsymbol{\varepsilon}_m(s,t) = \boldsymbol{A}_q(s,t)\hat{\boldsymbol{\varepsilon}}_m$$

SIGGRAPH '21 Talks, August 09-13, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8373-8/21/08.
https://doi.org/10.1145/3450623.3464675

where $\boldsymbol{A}_q(s,t)$ is a 3×7 matrix function, and

$$\hat{\boldsymbol{\varepsilon}}_m = \begin{bmatrix} \frac{\partial \boldsymbol{p}}{\partial s}^T \frac{\partial \boldsymbol{p}}{\partial s} \\ \frac{\partial \boldsymbol{p}}{\partial s}^T \frac{\partial \boldsymbol{p}}{\partial t} \\ \frac{\partial \boldsymbol{p}}{\partial t}^T \frac{\partial \boldsymbol{p}}{\partial t} \\ \frac{\partial^2 \boldsymbol{p}}{\partial s \partial t}^T \frac{\partial \boldsymbol{p}}{\partial s} \\ \frac{\partial^2 \boldsymbol{p}}{\partial s \partial t}^T \frac{\partial \boldsymbol{p}}{\partial t} \\ \frac{\partial^2 \boldsymbol{p}}{\partial s \partial t}^T \frac{\partial^2 \boldsymbol{p}}{\partial s \partial t} \\ 1 \end{bmatrix}$$

with all vectors evaluated at $(s,t) = \left(\frac{1}{2}, \frac{1}{2}\right)$. Note that the final entry (the constant 1) isn't a strain, but it is needed in order to subtract 1 from the first two entries of $\boldsymbol{\varepsilon}_m$, similar to how homogenous coordinates are used to translate points in transformation matrices. Since $\hat{\boldsymbol{\varepsilon}}_m$ no longer depends on st coordinates, we can move it outside of the sum over Gaussian quadrature points:

$$E_m \approx \frac{h}{2} \sum_{q \in Q} \hat{\boldsymbol{\varepsilon}}_m^T \left(\sum_{i=0}^{N_{qp}} g_i \boldsymbol{A}_q(s_i,t_i)^T \boldsymbol{K} \boldsymbol{A}_q(s_i,t_i) \left\| \frac{\partial \overline{\boldsymbol{p}}}{\partial s}(t_i) \times \frac{\partial \overline{\boldsymbol{p}}}{\partial t}(s_i) \right\| \right) \hat{\boldsymbol{\varepsilon}}_m$$

It is now possible to evaluate the Gaussian quadrature sum with data known at the start of the simulation, i.e., rest positions, the stiffness matrix, and Gaussian quadrature weights. We use this to precompute a new 7×7 stiffness matrix $\hat{\boldsymbol{K}}_q$ per patch, resulting in our final membrane energy formula:

$$E_m \approx \frac{h}{2} \sum_{q \in Q} \hat{\boldsymbol{\varepsilon}}_m^T \hat{\boldsymbol{K}}_q \hat{\boldsymbol{\varepsilon}}_m$$

We can now calculate the membrane energy over both triangles and bilinear patches, using 3 and 7 strain values respectively.

2 DERIVATION OF BENDING ENERGY

In our framework, we use the following bending energy:

$$E_b = \frac{h^3}{48} \int_{\overline{\Omega}} k_1 \mathrm{Tr}(\boldsymbol{S} - \overline{\boldsymbol{S}})^2 + k_2 \mathrm{Tr}\big((\boldsymbol{S} - \overline{\boldsymbol{S}})^2\big) d\overline{A}$$

where $\boldsymbol{S}$ and $\overline{\boldsymbol{S}}$ are the shape operators for our current and rest surfaces respectively, $k_1 = \frac{E\nu}{1-\nu^2}$, and $k_2 = \frac{E(1-\nu)}{1-\nu^2}$.

To calculate it on our discrete surface of triangles and quadrilaterals, we divide it into two parts: an *in-element* term that penalizes bending for each individual quadrilateral, and a *cross-element* term that penalizes bending between adjacent elements, assuming that each face is planar. We now discuss how to calculate both.

In-element Shape Operator. For a parameterized surface, we can calculate the shape operator with the first and second fundamental forms, which are calculated from the first and second derivatives of the surface. For the bilinear patch, this gives the following:

$$\begin{aligned} \boldsymbol{S}_q &= \begin{bmatrix} 0 & \frac{\partial^2 \boldsymbol{p}}{\partial s \partial t}^T \boldsymbol{n} \\ \frac{\partial^2 \boldsymbol{p}}{\partial s \partial t}^T \boldsymbol{n} & 0 \end{bmatrix} \begin{bmatrix} \frac{\partial \boldsymbol{p}}{\partial s}^T \frac{\partial \boldsymbol{p}}{\partial s} & \frac{\partial \boldsymbol{p}}{\partial s}^T \frac{\partial \boldsymbol{p}}{\partial t} \\ \frac{\partial \boldsymbol{p}}{\partial s}^T \frac{\partial \boldsymbol{p}}{\partial t} & \frac{\partial \boldsymbol{p}}{\partial t}^T \frac{\partial \boldsymbol{p}}{\partial t} \end{bmatrix}^{-1} \\ &= \frac{\frac{\partial^2 \boldsymbol{p}}{\partial s \partial t}^T \boldsymbol{n}}{\left\| \frac{\partial \boldsymbol{p}}{\partial s} \times \frac{\partial \boldsymbol{p}}{\partial t} \right\|^2} \begin{bmatrix} -\frac{\partial \boldsymbol{p}}{\partial s}^T \frac{\partial \boldsymbol{p}}{\partial t} & \frac{\partial \boldsymbol{p}}{\partial t}^T \frac{\partial \boldsymbol{p}}{\partial t} \\ \frac{\partial \boldsymbol{p}}{\partial s}^T \frac{\partial \boldsymbol{p}}{\partial s} & -\frac{\partial \boldsymbol{p}}{\partial s}^T \frac{\partial \boldsymbol{p}}{\partial t} \end{bmatrix} \end{aligned}$$

where $\boldsymbol{n}$ is the normal of the surface. This is one definition of the shape operator, but it is not unique; we have chosen the tangents of the bilinear patch to be the basis for our tangent space, but we are free to choose a different basis instead. In fact, we should; subtracting shape operators as discussed above only makes sense if they use the same tangent space basis, and there is no guarantee that our bilinear patch tangents will be aligned between the rest positions and the current position. Therefore, we perform a change of basis: we orthonormalize the tangent space, keeping the direction of $\frac{\partial \boldsymbol{p}}{\partial s}$ consistent. We can do this by taking the $\boldsymbol{QR}$ decomposition of our tangent vectors and using the thin version of $\boldsymbol{Q}$ as our new basis. $\boldsymbol{R}$ then defines how to transform our shape operator. In summary:

$$\left[\frac{\partial \boldsymbol{p}}{\partial s} \middle| \frac{\partial \boldsymbol{p}}{\partial t} \right] = \boldsymbol{QR}$$

$$\hat{\boldsymbol{S}}_q = \boldsymbol{R} \boldsymbol{S}_q \boldsymbol{R}^{-1}$$

Note that there are multiple valid definitions of the reparameterized shape operator $\hat{\boldsymbol{S}}_q$, since the $\boldsymbol{QR}$-decomposition is not unique, but all result in an equivalent bending energy. We choose the following definition:

$$\hat{\boldsymbol{S}}_q = \frac{\frac{\partial^2 \boldsymbol{p}}{\partial s \partial t}^T \boldsymbol{n}}{\left\| \frac{\partial \boldsymbol{p}}{\partial s} \times \frac{\partial \boldsymbol{p}}{\partial t} \right\|^2} \begin{bmatrix} 0 & \left\| \frac{\partial \boldsymbol{p}}{\partial s} \times \frac{\partial \boldsymbol{p}}{\partial t} \right\| \\ \left\| \frac{\partial \boldsymbol{p}}{\partial s} \times \frac{\partial \boldsymbol{p}}{\partial t} \right\| & -2 \frac{\partial \boldsymbol{p}}{\partial s}^T \frac{\partial \boldsymbol{p}}{\partial t} \end{bmatrix}$$

With this formulation, we can calculate the shape operators using the current and rest positions of each bilinear patch and subtract them without issue. We have found that calculating these shape operators once at the center of the patch is sufficient to approximate the energy integral.

Cross-element Shape Operator. To measure bending across faces in a way that is consistent with our in-element shape operator, we calculate a discrete shape operator that assumes all faces are planar. De Goes et al. [2020] describe how to calculate such a shape operator on a polygonal mesh. Their first formulation describes how to find the shape operator at the center of each face using surrounding vertex normals; however, each vertex normal depends on all of its adjacent faces, so the computational footprint for this calculation is fairly large. Instead, we opt for the adjoint version, which calculates a discrete shape operator for every vertex given the normals of the surrounding faces. This implies that the bending energy at a particular vertex depends on the surrounding 1-ring of vertices, which is consistent with the membrane energy.

Our cross-element shape operator is:

$$\boldsymbol{S}_v = \frac{1}{2a_v} \boldsymbol{T}_v^T \left(\sum_{f \in F(v)} \boldsymbol{Q}_f^v \boldsymbol{g}_f^v \boldsymbol{n}_f^T + \left(\boldsymbol{Q}_f^v \boldsymbol{g}_f^v \boldsymbol{n}_f^T \right)^T \right) \boldsymbol{T}_v$$

where v is the vertex where we are evaluating the shape operator, a_v is the voronoi area associated with vertex v, $\boldsymbol{T}_v$ is an orthonormal tangent frame at vertex v, $F(v)$ is the set of faces surrounding vertex v, $\boldsymbol{Q}_f^v$ is the linear operator that rotates the normal at f to the normal at v, $\boldsymbol{g}_f^v$ is the face area gradient vector, and $\boldsymbol{n}_f$ is the normal of the face. At a high level, this operator describes how much the normals at each face surrounding v need to be rotated to align with the vertex normal. We will now discuss how to compute each of these pieces in more detail.

For each face f, we calculate its *area vector* $\boldsymbol{a}_f$, i.e., the vector in the direction of the face's normal with a length equal to its area. For quadrilateral elements, we calculate this as $\boldsymbol{a}_f = \frac{1}{2}(\boldsymbol{p}_0 - \boldsymbol{p}_2) \times (\boldsymbol{p}_1 - \boldsymbol{p}_3)$, and denote the planar area as $a_f = \|\boldsymbol{a}_f\|$ and face normal as $\boldsymbol{n}_f = \boldsymbol{a}_f / a_f$. Note that $\boldsymbol{a}_f = \frac{\partial \boldsymbol{p}}{\partial s}(\frac{1}{2}) \times \frac{\partial \boldsymbol{p}}{\partial t}(\frac{1}{2})$, and that a_f is the area of the bilinear patch when projected to the tangent space at the center of the patch. Effectively, this is the closest linear approximation of the bilinear patch. Then, for each vertex v, we can compute its voronoi area as $a_v = \sum_{f \in F(v)} \frac{a_f}{n_f}$ where n_f is the number of vertices of f, and the vertex normal as the normalized sum of surrounding face area vectors.

We define the gradient vector as $\boldsymbol{g}_f^v = \frac{1}{2}\boldsymbol{n}_f \times (\boldsymbol{p}_{v-1} - \boldsymbol{p}_{v+1})$ where $\boldsymbol{p}_{v-1}$ and $\boldsymbol{p}_{v+1}$ denote the positions of vertices adjacent to v with respect to face f. (Note that this definition differs slightly from [De Goes et al. 2020] thanks to simplifications in our case.)

The vector tangent frame $\boldsymbol{T}_v$ is arbitrary as long as it is consistent between the rest position and the current position. For every vertex, we deterministically choose an adjacent edge; we orthonormalize the vertex normal and this edge vector to construct a consistently oriented tangent basis.

3 ANISOTROPIC STIFFNESS

We can extend our definitions of membrane energies with anisotropic stiffness. In particular, we implement *orthotropic* stiffness, implying that separate Young's modulus values are specified along orthogonal directions in rest tangent space. We allow the user to specify uv-coordinates for each vertex, but unlike other frameworks, these coordinates do not specify the mapping to 2D material space, but rather its orientation—the rest space positions define the scale of material space. User-provided uv-coordinates may have some shearing that is not respected by the rest positions, so we prioritize aligning the u-axis in rest space with the provided coordinates, and choose the v-axis direction by enforcing orthonormality.

Suppose a user has specified uv-coordinates $\boldsymbol{w}_i$ for a triangle or bilinear element. We define a 2×2 basis from these coordinates, denoted $\boldsymbol{W}$, calculated for triangles as:

$$\boldsymbol{W} = \left[(\boldsymbol{w}_1 - \boldsymbol{w}_0) | (\boldsymbol{w}_2 - \boldsymbol{w}_0)\right]$$

and for bilinear elements as:

$$\boldsymbol{W}(s,t) = \left[\frac{\partial \boldsymbol{w}}{\partial s}(t) | \frac{\partial \boldsymbol{w}}{\partial t}(s)\right]$$

When calculating the membrane energy, we multiply the rest-space tangent basis by $\boldsymbol{W}^{-1}$ before orthonormalizing, implying $\overline{\boldsymbol{QR}} = \overline{\boldsymbol{T}}\boldsymbol{W}^{-1}$. We must also take this into account when computing the deformation gradient: $\boldsymbol{F} = \boldsymbol{T}\boldsymbol{W}^{-1}\overline{\boldsymbol{R}}^{-1}$.

4 ENERGY DERIVATIVES

Thus far, we've only discussed the deformation energies themselves, but we need their gradients with respect to the degrees of freedom to calculate forces, and their Hessians to build the system matrix used for implicit integration. Additionally, we would like to ensure that the Hessians can be made positive semi-definite to prevent instabilities when using Newton's method to perform implicit integration. We will now discuss the details of these derivatives.

With the assumption that our shape operators are symmetric (which we have ensured in the derivations above), each of our energies can be written as a sum of quadratic forms: $E = \frac{1}{2}\sum_i \boldsymbol{\varepsilon}_i^T \boldsymbol{K}_i \boldsymbol{\varepsilon}_i$ where i iterates over faces for the membrane or in-element bending energies, or vertices for the cross-element bending energy. In each of these cases, $\boldsymbol{K}_i$ is independent of the degrees of freedom, so the gradient and Hessian of E can be written as follows:

$$\frac{\partial E}{\partial \boldsymbol{p}_j} = \sum_i \frac{\partial \boldsymbol{\varepsilon}_i}{\partial \boldsymbol{p}_j} \boldsymbol{K}_i \boldsymbol{\varepsilon}_i$$

$$\frac{\partial^2 E}{\partial \boldsymbol{p}_j \partial \boldsymbol{p}_k} = \sum_i \frac{\partial^2 \boldsymbol{\varepsilon}_i}{\partial \boldsymbol{p}_j \partial \boldsymbol{p}_k} \boldsymbol{K}_i \boldsymbol{\varepsilon}_i + \frac{\partial \boldsymbol{\varepsilon}_i}{\partial \boldsymbol{p}_j} \boldsymbol{K}_i \frac{\partial \boldsymbol{\varepsilon}_i}{\partial \boldsymbol{p}_k}^T$$

This implies that we need to find derivatives with respect to our strains. For the membrane strains, this is straightforward, since all of our strain measures are quadratic expressions of our degrees of freedom, implying the Hessian tensor is constant. It has a simple structure of blocks of 3×3 scaled identity matrices, and it is straightforward to find the eigenvalues and eigenvectors. We keep either the positive semi-definite or negative semi-definite components based on the sign of the associated stress value to ensure that $\frac{\partial^2 \boldsymbol{\varepsilon}_i}{\partial \boldsymbol{p}_j \partial \boldsymbol{p}_k} \boldsymbol{K}_i \boldsymbol{\varepsilon}_i$ is always positive semi-definite. $\frac{\partial \boldsymbol{\varepsilon}_i}{\partial \boldsymbol{p}_j} \boldsymbol{K}_i \frac{\partial \boldsymbol{\varepsilon}_i}{\partial \boldsymbol{p}_k}^T$ is positive semi-definite as long as $\boldsymbol{K}_i$ is, which is always true in our system.

Our bending energy derivatives are significantly more complicated, but in our experiments we have found that it is sufficient to use a quasi-Newton approximation for these Hessians by omitting the second derivative of the strain. This is acceptable since the bending stiffness is generally small compared to the stretching stiffness for cloth and thin shells, and our time step sizes are largely limited by our collision algorithm.

REFERENCES

Fernando De Goes, Andrew Butts, and Mathieu Desbrun. 2020. Discrete differential operators on polygonal meshes. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 110–1.

Pascal Volino, Nadia Magnenat-Thalmann, and Francois Faure. 2009. A simple approach to nonlinear tensile stiffness for accurate cloth simulation. *ACM Transactions on Graphics* 28, 4, Article 105 (2009).