



 [alecjacobson](#) / [geometry-processing-deformation](#)

Deformation assignment for Geometry Processing course

 MPL-2.0 License 48 stars  75 forks Star Watch ▼

< > Code

 Issues 2 Pull requests 34 Actions Projects Wiki Settings master ▼

...



psarahdactyl reduced paper pdf size ...

2 days ago  36[View code](#)

README.md

Geometry Processing - Deformation

To get started: Clone this repository by issuing

```
git clone --recursive http://github.com/alecjacobson/geometry-  
processing-deformation.git
```

Installation, Layout, and Compilation

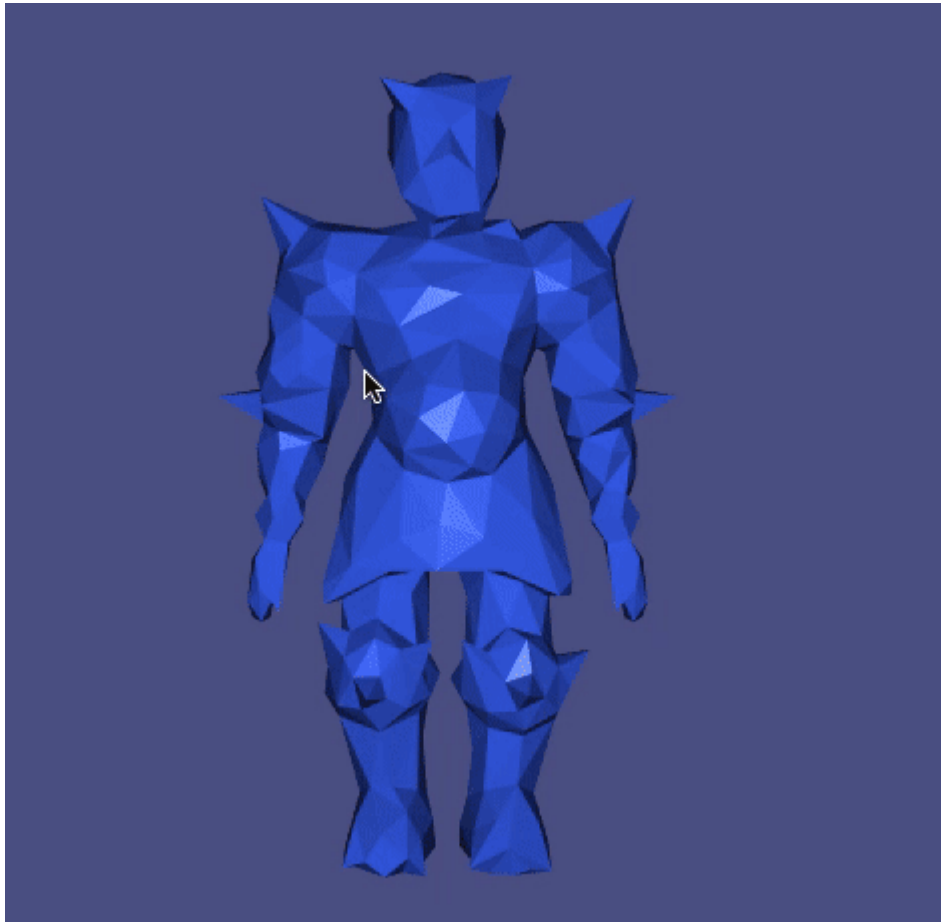
See [introduction](#).

Execution

Once built, you can execute the assignment from inside the `build/` by running on a given mesh:

```
./deformation [path to mesh.obj]
```

When the mesh is blue, the system is in "place handles" mode. Click on the mesh to select vertex locations for control point handles. After pressing space to switch to deformation mode, drag the handles. Pressing `m` will switch between different deformation methods.



Background

In this assignment we explore smooth deformation of an existing shape. [Shape deformation](#) has many applications in geometry process; we will focus on interactive **handle-based deformation**. In this setup, the user repositions a sparse set of points and the goal is to propagate the transformation at these "handles" to the rest of the shape. To be interactive we should aim for computing the new deformation of the shape at around 30 frames per second.

Shape deformation is the transformation from its *rest shape* to a new/current/deformed shape. If the position of a point on some 3D rest shape given by $\tilde{\mathbf{x}} \in \mathbb{R}^3$ then we will write that the unknown position on the deformed shape is given by $\mathbf{x} \in \mathbb{R}^3$. We can write this point's displacement vector as $\mathbf{x} - \tilde{\mathbf{x}} =: \mathbf{d} \in \mathbb{R}^3$.

The propagation of the handles' deformation can be thought of in two complementary ways:

1. as a **scattered data interpolation** problem, where handles provide sparse samples of an unknown **displacement field**, or
2. as a **shape optimization problem**, where we try to define a *new* shape that retains the details of the old shape but fulfills the handle constraints.

In the following discussion we will **take advantage of the ability to switch between thinking of the unknowns as the positions of the deformed shape (\mathbf{x}) and displacements (\mathbf{d})**. These views are equivalent, but often one or the other provides a better intuitive understanding.

Continuity

We will limit ourselves to **continuous deformations of shapes**. That is, **the shape will not tear, crack or change its topological features**.

If we represent our shape discretely as a **triangle mesh** (e.g., with *rest* vertices in $\tilde{\mathbf{V}} \in \mathbb{R}^{n \times 3}$ and faces in $F \in \{1, \dots, n\}^{m \times 3}$, **then we can trivially ensure a continuous deformation by determining new vertex positions \mathbf{V}** . The **topology (connectivity) of the mesh (F) will not change**.

Generic Distortion Minimization

A rest surface \tilde{S} **immersed** in \mathbb{R}^3 can be described as a mapping $\tilde{\mathbf{x}}$ from *some* 2D parametric domain Ω . For any parameters u and v , $\tilde{\mathbf{x}}$ describes the 3D position:

$$\tilde{\mathbf{x}}(u, v) \in \mathbb{R}^3.$$

Similarly the deformed surface can be represented as a position function $\mathbf{x} : \Omega \Rightarrow \mathbb{R}^3$. The displacement vector field is thus a function taking the difference:

$$\mathbf{d}(u, v) = \mathbf{x}(u, v) - \tilde{\mathbf{x}}(u, v).$$

For the **handle-based deformation problem** we would like to find a new surface (defined by \mathbf{x}) that:

1. adds as little *distortion* as possible to the shape, and
2. satisfies the users constraints at selected handle positions.

We can cast this as an energy optimization problem. Suppose we have energy **functional** $E(\mathbf{x})$ that measures the amount of distortion between the new shape (\mathbf{x}) and the rest shape $\tilde{\mathbf{x}}$, then we could optimize for the best possible shape \mathbf{x} by minimizing E :

$$\min_{\mathbf{x}} E(\mathbf{x}).$$

To ensure that the user's k handle points are interpolated we add the constraints:

$$\text{subject to } \mathbf{x}(u_i, v_i) = \mathbf{g}_i \quad \forall i = \{1, \dots, k\},$$

where \mathbf{g}_i is the position of the i th control point handle.

While the constraints are straightforward, we have many choices for how to formulate the energy function E . A natural choice is to measure distortion in an egalitarian way by integrating a *local* measure of distortion at all points on the surface:

$$\min_{\mathbf{x}} \int_{\Omega} \|e(\mathbf{x})\|^2 dA \quad \text{subject to } \mathbf{x}(u_i, v_i) = \mathbf{g}_i \quad \forall i = \{1, \dots, k\},$$

where e is a vector- or scalar- valued function measuring local (unsquared) distortion. We will now consider different choices for e .

Linear Methods

If we assume that the deformation between the rest shape given by $\tilde{\mathbf{x}}$ and the new shape given by \mathbf{x} is *small* then we can measure the distortion of the deformation in terms of the smoothness of the displacement field. This simplest methods will integrate the magnitude of derivatives of the displacement field (\mathbf{d}): if the displacement field has large variations or sudden changes then it is inducing a lot of distortion.

Gradient-based energy

Let us first consider minimizing the integral of squared variation of the displacement field:

$$\min_{\mathbf{d}} \int_{\Omega} \|\nabla \mathbf{d}\|_F^2 dA \quad \text{subject to } \mathbf{d}_i = \mathbf{g}_i - \tilde{\mathbf{x}}_i \quad \forall i = \{1, \dots, k\},$$

where $\nabla \mathbf{d} = \begin{pmatrix} \frac{\partial d^x}{\partial u} & \frac{\partial d^x}{\partial v} & \frac{\partial d^x}{\partial w} \\ \frac{\partial d^y}{\partial u} & \frac{\partial d^y}{\partial v} & \frac{\partial d^y}{\partial w} \\ \frac{\partial d^z}{\partial u} & \frac{\partial d^z}{\partial v} & \frac{\partial d^z}{\partial w} \end{pmatrix}$ the **Jacobian** matrix of the displacement field \mathbf{d}

Deformation Gradient

If $\mathbf{I} \in \mathbb{R}^{3 \times 3}$ is the identity matrix, then the quantity $\mathbf{F} := \mathbf{I} + \nabla \mathbf{d}$ is referred to as the **deformation gradient** in the **mechanics** community.

This is simply the familiar **Dirichlet energy** applied to each coordinate function of the displacement field *independently*.

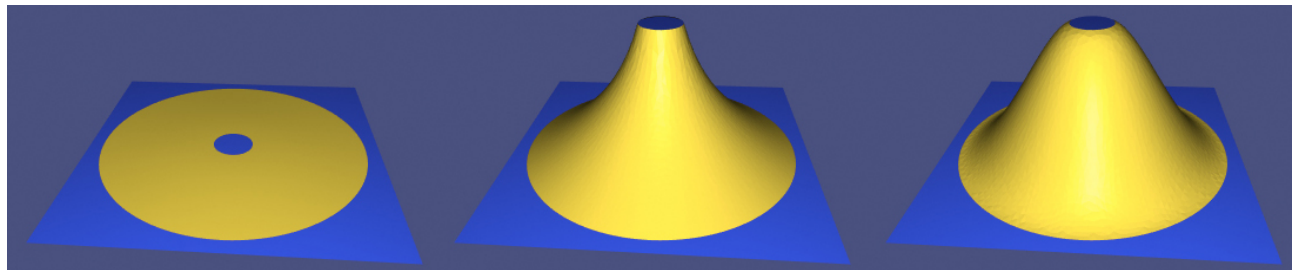
We can **discretize** this over our triangle mesh surface the same way we have in smoothing and parameterization assignments:

$$\min_{\mathbf{D}} \text{tr}(\mathbf{D}^T \mathbf{L} \mathbf{D}) \quad \text{subject to } \mathbf{D}_{\text{handles}} = \mathbf{g}_{\text{handles}} - \tilde{\mathbf{V}}_{\text{handles}},$$

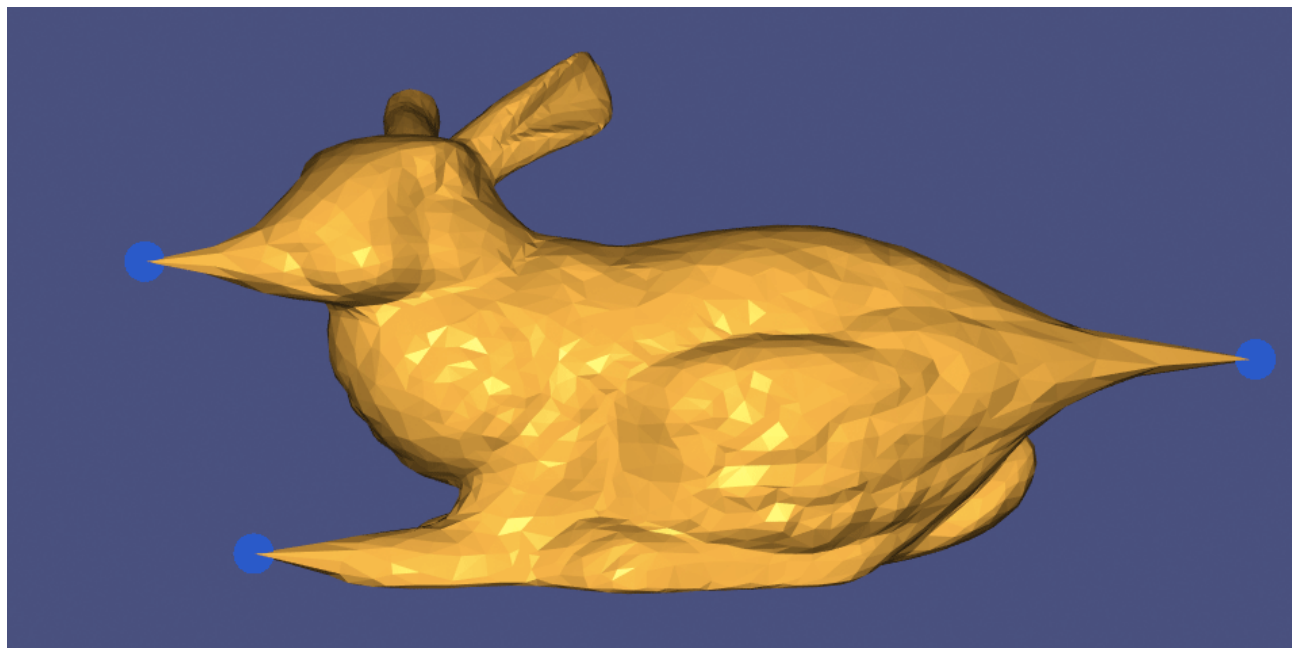
where the rows of $\mathbf{g}_{\text{handles}} \in \mathbb{R}^{k \times 3}$ contains the new positions of the k control point handles.

While easy to implement, this method suffers from a couple immediate problems:

1. it is not smooth at constraints, and
2. then *influence* of handles dies off too quickly.



By minimizing the Dirichlet energy, each coordinate of "displacement field" is a [harmonic function](#). Intuitively (however abstractly) we can think of each function as *diffusing* the user's constraints as if they were [heat](#) values. As such, each function diffuses quickly to an average, slowly varying value over most of the domain. As a displacement field a constant value for each coordinate function would mean a translation: most of the shape is simply translated.



The gradient operator (∇) is a [linear operator](#). We can alternatively view our minimization above in terms of the unknown positions \mathbf{x} :

$$\min_{\mathbf{d}} \int_{\Omega} \|\nabla \mathbf{d}\|_F^2 dA \Rightarrow \min_{\mathbf{x}} \int_{\Omega} \|\nabla(\mathbf{x} - \tilde{\mathbf{x}})\|_F^2 dA \Rightarrow \min_{\mathbf{x}} \int_{\Omega} \underbrace{\|\nabla \mathbf{x}\|_F^2}_{\text{after}} - \underbrace{\|\nabla \tilde{\mathbf{x}}\|_F^2}_{\text{before}} dA.$$

If we think of the gradient of the position function $\nabla \mathbf{x}$ (with respect to the underlying parameterization u, v) as a local geometric **feature descriptor** then this energy can be re-understood as measuring the difference in this feature before and after the deformation. This is very sensible as we are trying to measure distortion. We would expect that a low-distortion deformation would correspond with a small change to local features.

Unfortunately, the gradient of the position function \mathbf{x} is a *poor*, first-order local feature.

Laplacian-based energy

If we model distortion as the change in a local feature descriptor, then a natural local and *relative* descriptor would be one that compared the position of some point on the shape to the average of its local neighborhood. We have studied an operator that computes this in the smoothing assignment. The **Laplace(-Beltrami) operator** can be derived as taking exactly the difference of a functions value at a point and the average (i.e., **centroid**) of an infinitesimal region around that point:

$$\Delta f(\mathbf{x}) = \lim_{|B(\mathbf{x})| \rightarrow 0} \frac{1}{|B(\mathbf{x})|} \int_{B(\mathbf{x})} f(\mathbf{z}) d\mathbf{z} - f(\mathbf{x})$$

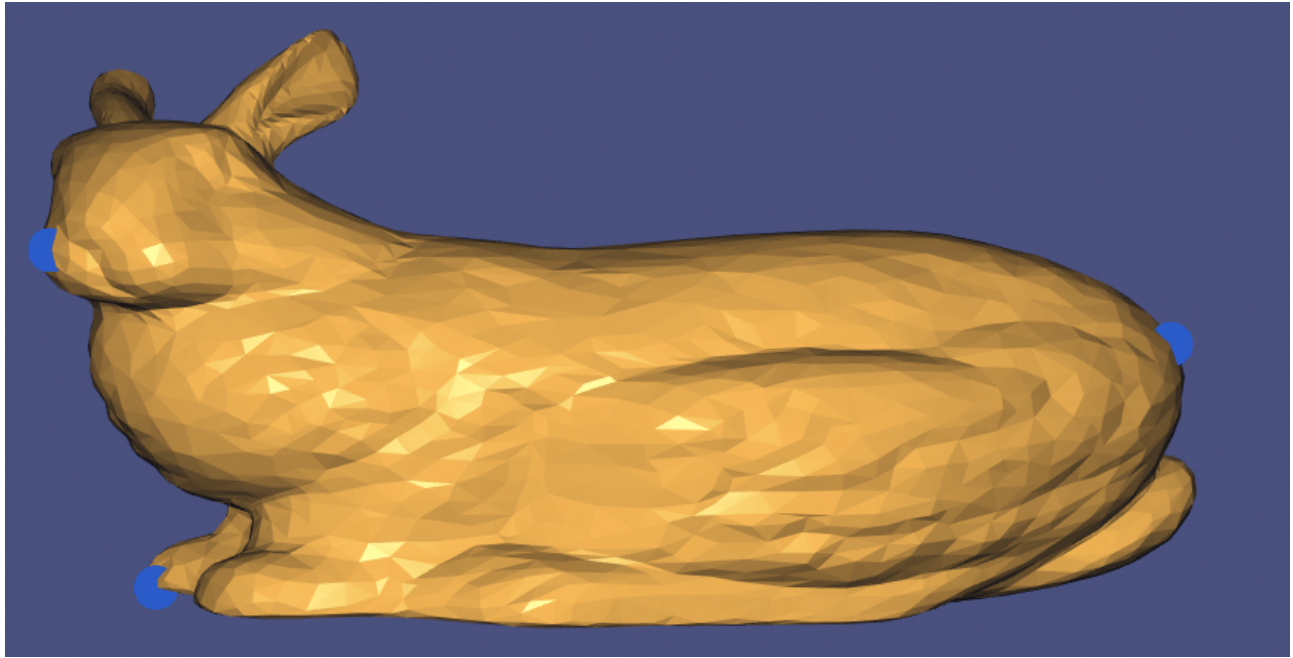
(see, e.g., "**Differential coordinates for local mesh morphing and deformation**" [Alexa et al. 2003] and expanded upon in "**Laplacian Surface Editing**" [Sorkine et al. 2004]).

When applied to the embedding function \mathbf{x} the Laplace operator computes the difference in *position* between a point and its local neighborhood. This *vector* points in the **normal** direction and its magnitude corresponds inversely with **how flat the surface is locally**.

Let's replace the gradient feature above with this *second-order* feature descriptor and massage our optimization problem back in terms of displacements:

$$\min_{\mathbf{x}} \int_{\Omega} \left\| \underbrace{\Delta \mathbf{x}}_{\text{after}} - \underbrace{\Delta \tilde{\mathbf{x}}}_{\text{before}} \right\|^2 dA \Rightarrow \min_{\mathbf{x}} \int_{\Omega} \|\Delta(\mathbf{x} - \tilde{\mathbf{x}})\|^2 dA \Rightarrow \min_{\mathbf{d}} \int_{\Omega} \|\Delta \mathbf{d}\|^2 dA.$$

Just as we can show that harmonic functions ($\Delta \mathbf{d} = 0$) minimize the Dirichlet energy, we can use **calculus of variations** apply **Green's identity** *twice* to show that minimizers of the squared-Laplacian energy are **_bi-_harmonic** functions ($\Delta \Delta \mathbf{d} = 0$ or $\Delta^2 \mathbf{d} = 0$). Obviously all harmonic functions are also biharmonic functions. This implies that the space of biharmonic functions is strictly *larger*. In particular, this will allow use to ensure continuity of first derivatives across constrained values at handles.



The fact that each coordinate of the displacement field \mathbf{d} will be a bi-harmonic function is not so elucidating, but by minimizing the squared-Laplacian energy integrated over the domain, we can say that it is *as-harmonic-as-possible*. Harmonic functions include constant functions (i.e., translations) but also any displacement that *bends* in one direction by an equal and opposite amount as it bends in the other direction:

$$\Delta \mathbf{d} = 0 \rightarrow \partial \mathbf{d} / \partial u + \partial \mathbf{d} / \partial v = 0 \rightarrow \partial \mathbf{d} / \partial u = -\partial \mathbf{d} / \partial v.$$

To discretize this energy we can make use of our discrete Laplacian operator \mathbf{L} . This matrix computes the locally *integrated* Laplacian of a given function specified by per-vertex values \mathbf{f} . Now we would like to integrate the square of the *point-wise* Laplacian. We can approximate the point-wise Laplacian by the local **integral average** of the Laplacian $\mathbf{M}^{-1}\mathbf{L}\mathbf{f}$. Integrating this over the mesh we have our complete approximate of the energy:

$$\begin{aligned} \int_{\Omega} \|\Delta \mathbf{d}\|^2 dA &\approx \text{tr} (\mathbf{D}^T \mathbf{L}^T \mathbf{M}^{-T} \mathbf{M} \mathbf{M}^{-1} \mathbf{L} \mathbf{D}) \\ &= \text{tr} \left(\mathbf{D}^T \underbrace{\mathbf{L}^T \mathbf{M}^{-1} \mathbf{L}}_{\mathbf{Q}} \mathbf{D} \right), \end{aligned}$$

where $\mathbf{M} \in \mathbb{R}^{n \times n}$ is the mass-matrix for the given mesh and $\mathbf{Q} \in \mathbb{R}^{n \times n}$ can be thought of as the bi-Laplacian matrix.

***k*-harmonic**

The logical continuation of harmonic and biharmonic deformation is to consider *triharmonic* ($\Delta^3 \mathbf{d} = 0$) and *tetraharmonic* ($\Delta^4 \mathbf{d} = 0$) and so on. It's straightforward to implement these, though there are diminishing returns and increasing costs.

Precomputation

With out loss of generality, assume that the rows of the unknown displacements \mathbf{D} have been sorted so that displacements corresponding to handle vertices are in the bottom part:

$$\mathbf{D} = \begin{pmatrix} \mathbf{D}_u \\ \mathbf{D}_h \end{pmatrix}$$

Since the displacements at handles are *known* before the optimization, we can separate the knowns and unknowns in the energy:

$$\begin{aligned} \min_{\mathbf{D}_u} \text{tr} \left((\mathbf{D}_u^T \ \mathbf{D}_h^T) \begin{pmatrix} \mathbf{Q}_{u,u} & \mathbf{Q}_{u,h} \\ \mathbf{Q}_{h,u} & \mathbf{Q}_{h,h} \end{pmatrix} \begin{pmatrix} \mathbf{D}_u \\ \mathbf{D}_h \end{pmatrix} \right) \\ \min_{\mathbf{D}_u} \text{tr} \left(\mathbf{D}_u^T \mathbf{Q}_{u,u} \mathbf{D}_u + 2 \mathbf{D}_u^T \mathbf{Q}_{u,h} \mathbf{D}_h + \underbrace{\mathbf{D}_h^T \mathbf{Q}_{h,h} \mathbf{D}_h}_{\text{constant}} \right) \\ \min_{\mathbf{D}_u} \text{tr} (\mathbf{D}_u^T \mathbf{Q}_{u,u} \mathbf{D}_u + 2 \mathbf{D}_u^T \mathbf{Q}_{u,h} \mathbf{D}_h) \end{aligned}$$

where $\mathbf{Q}_{u,u} \in \mathbb{R}^{(n-k) \times (n-k)}$ is the quadratic coefficients matrix corresponding to the unknown displacements.

This quadratic optimization problem may solved by setting all partial derivatives with respect to degrees of freedom in \mathbf{D}_u to zero:

$$2\mathbf{Q}_{u,u}\mathbf{D}_u + 2\mathbf{Q}_{u,h}\mathbf{D}_h = 0 \rightarrow \mathbf{D}_u = \mathbf{Q}_{u,u}^{-1}\mathbf{Q}_{u,h}\mathbf{D}_h$$

If we don't change *which* vertices are handles, but only change the positions of the selected handles, then only \mathbf{D}_h changes above. In particular, the matrix $\mathbf{Q}_{u,u}$ is unchanged. Therefore, we can **prefactorize** it so that *applying its inverse* is fast (`igl:min_quad_with_fixed` does this for you).

Actually the entire term $\mathbf{Q}_{u,u}^{-1}\mathbf{Q}_{u,h} =: \mathbf{W} \in \mathbb{R}^{(n-k) \times k}$ does not change. The columns of \mathbf{W} reveal how unknown displacements respond to each handle point's displacement (see also "An intuitive framework for real-time freeform modeling" [Botsch & Kobbelt 2004]). This provides a gateway to the relationship with linear blend skinning and automatic (biharmonic) weighting functions (see "Bounded Biharmonic Weights for Real-Time Deformation" [Jacobson et al. 2011]).

Trouble in paradise

Biharmonic displacements work well for small deformations and deformations that do not imply a large rotation. However, the Laplacian of the position function $\Delta \mathbf{x}$ as a feature descriptor is *not* rotation invariant. This problem is true for all linear differential features including the gradient of the embedding function $\nabla \mathbf{x}$ considered above (see "On linear variational surface deformation methods" [Botsch & Sorkine 2008]).



This means that if the user transforms all of the handle locations by a rigid transformation \mathbf{T} these energies will *not* measure zero for a displacement equivalent to applying the rigid transformation \mathbf{T} to the entire shape. We would like *global* rotation invariance, but we would also like this property to apply *locally* so that parts of the shape can easily rotate.

As-rigid-as-possible

In the scenario where each handle i are perfectly transformed by a [rigid transformation](#) $\mathbf{x}_i = \mathbf{R}\tilde{\mathbf{x}}_i + \mathbf{t}$, where $\mathbf{R} \in SO(3) \subset \mathbb{R}^{3 \times 3}$ is a rotation matrix and $\mathbf{t} \in \mathbb{R}^3$ is a translation vector. If an [oracle](#) could only tell us this particular rigid transformation then we could repair the gradient-based energy above by pre-rotating the rest shape by this transformation:

$$\begin{aligned} \int_{\Omega} \|\nabla \mathbf{x} - \nabla(\mathbf{R}\tilde{\mathbf{x}} + \mathbf{t})\|^2 dA &= \int_{\Omega} \|\nabla \mathbf{x} - \nabla(\mathbf{R}\tilde{\mathbf{x}}) - \nabla \mathbf{t}\|^2 dA \\ &= \int_{\Omega} \|\nabla \mathbf{x} - \mathbf{R}\nabla \tilde{\mathbf{x}}\|^2 dA, \end{aligned}$$

where the translation vector \mathbf{t} falls out because a translation has constant gradient.

We do not know the rotation \mathbf{R} ahead of time, but we could be as generous as possible and use the "best" rotation $\mathbf{R} \leftarrow \operatorname{argmin}_{\mathbf{R}} \int_{\Omega} \|\nabla \mathbf{x} - \mathbf{R}\nabla \tilde{\mathbf{x}}\|^2 dA$:

$$\int_{\Omega} \left\| \nabla \mathbf{x} - \left(\operatorname{argmin}_{\mathbf{R}} \int_{\Omega} \|\nabla \mathbf{x} - \mathbf{R}\nabla \tilde{\mathbf{x}}\|^2 dA \right) \nabla \tilde{\mathbf{x}} \right\|^2 dA.$$

If we treat \mathbf{R} as a degree of freedom along with the unknown positions \mathbf{x} , we can unify this into an optimization over \mathbf{x} and \mathbf{R} :

$$\min_{\mathbf{x}, \mathbf{R} \in SO(3)} \int_{\Omega} \|\nabla \mathbf{x} - \mathbf{R}\nabla \tilde{\mathbf{x}}\|^2 dA.$$

Optimizing this energy will ensure *global* rotation invariance. To ensure *local* rotation invariance, we can replace $\mathbf{R} \in SO(3)$ with a spatially varying *function* $\mathbf{R} : \Omega \rightarrow SO(3)$ that outputs a "best" rotation for any point on the shape (see ["A simple geometric model for elastic deformations" \[Chao et al. 2010\]](#)). In this way, the optimal rotation will be locally rigid everywhere, or *as-rigid-as-possible* (ARAP).



For embedded solid shapes, we can take the rest shape given by $\tilde{\mathbf{x}}$ as the parameterization Ω , so that $\nabla \tilde{\mathbf{x}} = \mathbf{I}$. This allows us to rewrite the as-rigid-as-possible energy as the square of the difference between the [deformation gradient](#) and the closest rotation:

$$\begin{aligned} & \int_{\Omega} \|\nabla \mathbf{x} - \mathbf{R} \nabla \tilde{\mathbf{x}}\|^2 dA \\ & \int_{\Omega} \|(\nabla \mathbf{x} + \mathbf{I} - \mathbf{I}) - \mathbf{R} \mathbf{I}\|^2 dA \\ & \int_{\Omega} \|(\mathbf{I} + \nabla \mathbf{x} - \nabla \tilde{\mathbf{x}}) - \mathbf{R}\|^2 dA \\ & \int_{\Omega} \|(\mathbf{I} + \nabla \mathbf{d}) - \mathbf{R}\|^2 dA \\ & \int_{\Omega} \|\mathbf{F} - \mathbf{R}\|^2 dA \end{aligned}$$

This form provides a bridge between the as-rigid-as-possible energy common in geometry processing to *corotated linear elasticity* used in mechanics/physically-based simulation (made explicit in "A simple geometric model for elastic deformations" [Chao et al. 2010]). See Section 3.4 of "FEM Simulation of 3D Deformable Solids" [Sifakis 2012] for a graphics-mechanics perspective.

Discrete as-rigid-as-possible energy

For a triangle mesh with displacing vertices, the gradient of the embedding function is constant inside each triangle. In this way we can write the raw gradient energy above as a double sum over all half-edges ij of all faces f in the mesh:

$$\frac{1}{2} \int_{\Omega} \|\nabla \mathbf{x} - \nabla \tilde{\mathbf{x}}\|^2 dA = \frac{1}{2} \sum_{f \in F} \sum_{ij \in f} c_{ij} \|(\mathbf{v}_i - \mathbf{v}_j) - (\tilde{\mathbf{v}}_i - \tilde{\mathbf{v}}_j)\|^2,$$

where c_{ij} is cotangent of the angle opposite half-edge ij .

To inject localized best fit rotations, we will assign an unknown rotation matrix \mathbf{R}_k to each vertex k of the mesh and accounts for a third of the energy integrated over incident triangles:

$$\frac{1}{2} \int_{\Omega} \|\nabla \mathbf{x} - \mathbf{R} \nabla \tilde{\mathbf{x}}\|^2 dA = \frac{1}{6} \sum_{k=1}^n \sum_{ij \in F(k)} c_{ij} \|(\mathbf{v}_i - \mathbf{v}_j) - \mathbf{R}_k(\tilde{\mathbf{v}}_i - \tilde{\mathbf{v}}_j)\|^2,$$

where $F(k)$ is the set of all faces incident on the k -th vertex.

Where do rotations live?

We have assigned a rotation for each vertex: there are n rotation matrices as auxiliary degrees of freedom. This is in contrast to assigning rotations per face (as in "A Local/Global Approach to Mesh Parameterization" [Liu et al. 2008]). Per-face--or more generally per-element--rotations work well for **codimension** zero objects (triangle meshes in \mathbb{R}^2 or tetrahedral meshes in \mathbb{R}^3). But for triangle mesh surfaces in \mathbb{R}^3 (i.e., codimension 1), per-face rotations would lead to "crumpling" because *bending* along edges would not be measured.

Optimization

The simplest method for optimizing the ARAP energy is by alternating between

1. finding the optimal rotations \mathbf{R}_k assuming the vertex positions \mathbf{V} are fixed, and
2. finding the **optimal vertex positions \mathbf{V}** assuming all rotations \mathbf{R}_k are fixed.

Each rotation \mathbf{R}_k only affects the local energy and doesn't interact with the *other* rotations. So each can be optimized *locally*. In contrast, the mesh vertex positions \mathbf{V} depend on each other requiring a *global* solve. In the geometry processing literature, this is known as a local-global optimization (see "[As-rigid-as-possible surface modeling](#)" [Sorkine & Alexa 2007]). It is also known as "alternating block coordinate descent" because we have separated the variables into disjoint sets $(\mathbf{V}, \mathbf{R}_1, \dots, \mathbf{R}_n)$ and taking the optimal descent direction for each independently.

Observing the discrete energy above we can see that the energy is quadratic in \mathbf{V} and quadratic in each \mathbf{R}_k . Let's start by separating the terms that are quadratic and linear in \mathbf{V} :

$$\underbrace{\frac{1}{6} \sum_{k=1}^n \sum_{ij \in F(k)} c_{ij} (\mathbf{v}_i - \mathbf{v}_j)^T (\mathbf{v}_i - \mathbf{v}_j)}_{\text{quadratic}} + \underbrace{\frac{1}{6} \sum_{k=1}^n \sum_{ij \in F(k)} c_{ij} (\mathbf{v}_i - \mathbf{v}_j)^T \mathbf{R}_k (\tilde{\mathbf{v}}_i - \tilde{\mathbf{v}}_j)}_{\text{linear}}$$

if we stack the rotation matrices \mathbf{R}_k into large matrix $\mathbf{R} \in \mathbb{R}^{3n \times 3}$ then we can write this energy in matrix form as:

$$\text{tr}(\mathbf{V}^T \mathbf{L} \mathbf{V}) + \text{tr}(\mathbf{V}^T \mathbf{K} \mathbf{R}),$$

where $\mathbf{L} \in \mathbb{R}^{n \times n}$ is the familiar cotangent discrete Laplacian matrix and $\mathbf{K} \in \mathbb{R}^{n \times 3n}$ sparse matrix containing cotangents multiplied against differences across edges in the rest mesh (e.g., $\tilde{\mathbf{v}}_i - \tilde{\mathbf{v}}_j$).

I'm so confused. What's in the \mathbf{K} matrix?

Let's take it slow. The \mathbf{K} matrix is represents the [bilinear form](#) that combines unknown vertex positions and unknown rotations. We have identified above that we can write this in summation or matrix form:

$$\frac{1}{6} \sum_{k=1}^n \sum_{ij \in F(k)} c_{ij} (\mathbf{v}_i - \mathbf{v}_j)^T \mathbf{R}_k (\tilde{\mathbf{v}}_i - \tilde{\mathbf{v}}_j) = \text{tr}(\mathbf{V}^T \mathbf{K} \mathbf{R}),$$

but how did we get here?

Let's start with the summation form. The *constants* of this formula are the c_{ij} terms and the $(\tilde{\mathbf{v}}_i - \tilde{\mathbf{v}}_j)$ terms. Since these always appear together, let us merge them into weighted edge difference vectors $c_{ij}(\tilde{\mathbf{v}}_i - \tilde{\mathbf{v}}_j) =: \tilde{\mathbf{e}}_{ij} \in \mathbb{R}^3$:

$$\frac{1}{6} \sum_{k=1}^n \sum_{ij \in F(k)} \underbrace{(\mathbf{v}_i - \mathbf{v}_j)^T \mathbf{R}_k}_{\in \mathbb{R}} \tilde{\mathbf{e}}_{ij},$$

the inner term in the summation is an **inner product**; that is, a **scalar**. Let's expose this by expanding the matrix-vector products of the inner-product:

$$\frac{1}{6} \sum_{k=1}^n \sum_{ij \in F(k)} \sum_{\alpha=1}^3 \sum_{\beta=1}^3 (v_i^\alpha - v_j^\alpha) R_k^{\alpha\beta} \tilde{e}_{ij}^\beta.$$

If our mesh is stored as a vertex list and face list, it's not easy/efficient to loop over per-vertex rotations (outer sum) and then over all half-edges of incident faces (second sum). Instead, let's rearrange these sums to loop over all faces first, then the half-edges of that face, and then over all per-vertex rotations that involve this half-edge:

$$\frac{1}{6} \sum_{f=1}^m \sum_{ij \in E(f)} \sum_{k|ij \in F(k)} \sum_{\alpha=1}^3 \sum_{\beta=1}^3 (v_i^\alpha - v_j^\alpha) R_k^{\alpha\beta} \tilde{e}_{ij}^\beta,$$

where the third sum is over all rotations k such that the half-edge ij belongs to the half-edges of the faces incident on the k -th vertex: $k|ij \in F(k)$. Well, this means k can either be i or j or the third vertex of the f -th face.

Now let's turn our attention back to the **summand**. The terms indexed by α never *mix*. That is, we never add/multiply v_i^α , v_j^γ , and $R_k^{\delta\beta}$ unless $\alpha = \gamma = \delta$. This implies that we can write this summation in matrix form as:

$$\mathbf{V}_1^\top \mathbf{K}_1 \mathbf{R}_1 + \mathbf{V}_2^\top \mathbf{K}_2 \mathbf{R}_2 + \mathbf{V}_3^\top \mathbf{K}_3 \mathbf{R}_3,$$

where $\mathbf{V}_\alpha \in \mathbb{R}^n$ is α -th column of \mathbf{V} , $\mathbf{R}_\alpha \in \mathbb{R}^{3n}$ is the α -th column of \mathbf{R} and $\mathbf{K}_1, \mathbf{K}_2, \mathbf{K}_3 \in \mathbb{R}^{n \times 3n}$ are sparse matrices.

Further, the constant term \tilde{e}_{ij}^β in the summation acts the same on $(v_i^\alpha - v_j^\alpha) R_k^{\alpha\beta}$ for any α value. This implies that $\mathbf{K}_1 = \mathbf{K}_2 = \mathbf{K}_3$, so we can reduce the matrix form to:

$$\text{tr}(\mathbf{V} \mathbf{K} \mathbf{R}).$$

Finally, we can answer what is in each entry K_{vw} , where v chooses the row and $w = 3k + \beta$ chooses the column of \mathbf{K} . Treating our nested summation as nested for-loops, we can increment entries in \mathbf{K}

$$\begin{aligned} K_{i \ 3k+\beta} &+= \tilde{e}_{ij}^\beta, \\ K_{j \ 3k+\beta} &+= -\tilde{e}_{ij}^\beta, \end{aligned}$$

for each half-edge encountered.

Local step

Minimizing this energy with respect \mathbf{R} corresponds to minimizing:

$$\text{tr} \left(\underbrace{\mathbf{V}^T \mathbf{K}}_{\mathbf{C}^T} \mathbf{R} \right),$$

where $\mathbf{C} \in \mathbb{R}^{3n \times 3}$ stacks weighted covariance matrices $\mathbf{C}_k \in \mathbb{R}^{3 \times 3}$ for each region covered by the corresponding rotation \mathbf{R}_k . We have seen this problem before in the registration assignment. For each \mathbf{C}_k , \mathbf{R}_k will be the closest rotation matrix solved via [singular value decomposition](#).

Global step

Minimizing the energy above with respect to \mathbf{V} corresponds to solving a Dirichlet energy-like minimization problem:

$$\min_{\mathbf{V}} \text{tr} (\mathbf{V}^T \mathbf{L} \mathbf{V}) + \text{tr} (\mathbf{V}^T \mathbf{B})$$

where $\mathbf{K} \mathbf{R} =: \mathbf{B} \in \mathbb{R}^{n \times 3}$ is a matrix of rotated vertex gradients. Adding the handle constraints to the corresponding rows of \mathbf{V} this is easily minimized by setting all partial derivatives with respect to the unknowns in \mathbf{V} equal to zero (as in the linear methods above).

Implementation

In order to facilitate interactive deformation we would like our local and global iterations to be computed as quickly as possible. Since the quadratic form in the global step is the *same* regardless of the current rotations or current handle positions, we can [prefactorize](#) it (again, as above). The matrix \mathbf{K} also does not depend on the rotations, current positions or handle positions, so we can pre-build this matrix. This way computing \mathbf{C} and \mathbf{B} is a simple matrix-matrix multiplication.

Note: When constructing \mathbf{K} it's easiest to iterate over *all* half-edges in the mesh (by iterating over all faces and then each of the three edges). Each half-edge ij contributes terms tying $\mathbf{v}_i, \mathbf{v}_j$ to *each* of the (three) rotations \mathbf{R}_k that apply against their difference (see "Fast Automatic Skinning Transformations" [Jacobson et al. 2012]).

Still trouble in paradise



The as-rigid-as-possible deformation method for surfaces described above has a number of remaining problems:

1. like its gradient-based energy ancestor the deformation is not smooth at constraints;
2. the energy punishes *bending* of the surface--which is good--but does so in a way that diminishes as the mesh becomes higher and higher resolution, in otherwords, the discrete energy is mesh-resolution dependent; and
3. the energy is biased by the original combinatorics of the mesh (even in flat regions).

Tasks

Blacklist

- `igl::arap`
- `igl::arap_linear_block`
- `igl::covariance_scatter_matrix`
- `igl::harmonic`

Whitelist

- `igl::polar_svd3x3` (or your previous assignment's `closest_rotation`)
- `igl::min_quad_with_fixed`

- `igl::cotmatrix_entries`
- `igl::cotmatrix` (or your previous implementation)
- `igl::massmatrix` (or your previous implementation)

source/biharmonic_precompute.cpp

Precompute data needed to efficiently solve for a biharmonic deformation given a mesh with vertices v and faces F and a list of selected vertices as indices b into v . The output should be a prefactorized system using the `data` struct employed by `igl::min_quad_with_fixed`.

source/biharmonic_solve.cpp

Given precomputation `data` and a list of handle *displacements* determine *displacements* for all vertices in the mesh.

source/arap_precompute.cpp

Precompute data needed to efficiently conduct local-global iterations for an arap deformation. This includes the `data` struct employed by `igl::min_quad_with_fixed` to solve the global step" and constructing the bilinear form κ that mixes rotation degrees of freedom with unknown positions for preparing the covariance matrices of the local step and the linear term of the global step.

source/arap_single_iteration.cpp

Given precomputed data (`data` and κ), handle *positions* b_c and current positions of all vertices u , conduct a *single* iteration of the local-global solver for minimizing the as-rigid-as-possible energy. Output the *positions* of all vertices of the mesh (by overwriting u).

Releases

No releases published

Packages

No packages published

Contributors 3



alecjacobson Alec Jacobson



ErisZhang Jiayi Eris Zhang



psarahdactyl Sarah Kushner

Languages

● **C++** 86.5% ● **CMake** 13.5%