Laplacian Mesh Deformation Documentation

Arnaud Gelas

June 15, 2013

Abstract

Deforming a 3D surface mesh while preserving its local detail is useful for editing anatomical atlases or for mesh based segmentation. This contribution introduces new classes for performing *hard* and *soft constraints* deformation in a flexible design which allows user to switch easily in between Laplacian discretization operators, area weighing and solvers. The usage of these new classes is demonstrated on a sphere.

Contents

Contents

1 Introduction

Manipulating and modifying surface while preserving geometric details has been an active area of research in geometric modeling due to their applications in design (e.g. computer graphics, animation), but also in biomedical imaging (e.g. deforming surface based atlas).

Here, we present one surface-based technique, as opposed to free-form deformations which deform the ambient 3D space, and based on differential representations (i.e. gradient based representation, Laplacian based representation, local frame representation). These techniques are becoming more and more popular over the last past years, most likely due to their robustness, speed and ease of implementation. The main idea behind these approaches relies on the use of a representation that focus on local differential properties and on preserving these ones when deforming. These approaches remain quite intuitive and preserve local details throughout the deformation. In the rest of the paper, we will focus on Laplacian based representation.

2 Background

Laplacian-based approaches represent the surface by the so-called differential coordinates or Laplacian coordinates [3], [59]. These coordinates are obtained by applying the Laplacian operator to the mesh vertices:

$$\delta_i = \Delta_S(p_i) = -H_i \cdot n_i$$

where H_i is the mean curvature ($\kappa_1 + \kappa_2$) at the vertex v_i.

The deformation can be formulated by minimizing the difference from the input surface coordinates δ_i. With a continuous formulation, this would lead to the minimization of the following energy:

$$\min_{p'} \int_\Omega \|\Delta p' - \delta\| dudv$$

The Euler-Lagrange equation derived:

$$\Delta^2 p' = \Delta\delta$$

When considering the input surface as the parameter domain, the Laplace operator turns out into the Laplace-Beltrami operator Δ_S:

$$L^2 p' = L\delta$$

which can be separated into 3 coordinate components. Then users can add positional constraints on some vertices:

$$p'_j = c_j$$

Note that the positional constraints can either be incorporated as *hard* or *soft constraints*.

2.1 Adding constraints

Hard constraints formulation

The problem can thus be expressed as follows:

$$\begin{aligned} L^2 p' &= L\delta \\ p'_j &= c_j \end{aligned}$$

where c_j are the positional constraints, or by constraining the displacement $d_i = c_i - p_i$.

which can be rewritten as:

$$\begin{pmatrix} L^2 & \\ 0 & I_k \end{pmatrix} \cdot \begin{pmatrix} d_1 \\ \vdots \\ d_n \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ c_{n'+1} - p_{n'+1} \\ \vdots \\ c_n - p_n \end{pmatrix}$$

where $L^2 \in \mathbb{R}^{n\times n}$ and I_k is the $k \times k$ identity matrix. By eliminating rows in the above linear system, we finally get a $n' \times n'$ sparse linear system where the unknown vector represents the 3D deformation of the n' unconstrained vertices.

Soft constraints formulation

In contrast soft constraints correspond to relax the previous constraint by incorporating the constraints into the enyergy minimization leading to

$$\min_{p'} \sum_{i=1}^{N} \|\Delta_S(p'_i - \delta_i\|^2 + \lambda \sum_{j=n'+1}^{n} \|p'_j - c_j\|^2$$

The minimum of this energy is can be found by solving the normal equations:

$$\left[L^t \cdot L + \lambda^2 \cdot \begin{pmatrix} 0 & 0 \\ 0 & I_k \end{pmatrix} \right] \cdot \begin{pmatrix} p_1 \\ \vdots \\ p_n \end{pmatrix} = L^t \cdot \begin{pmatrix} \delta'_0 \\ \vdots \\ \delta'_n \end{pmatrix} + \lambda^2 \cdot \begin{pmatrix} 0 \\ \vdots \\ 0 \\ c_{n'+1} \\ \vdots \\ c_n \end{pmatrix}$$

Depending on λ solution can be closed to an interpolation of the constraints (with large values), or an approximation of them (with low values).

2.2 Laplacian discretization

This approach requires a dicretization of the Laplacian operator, and results would highly depends on it. There exists several variations of the weights used in the typically used Laplacian discretization. Here we list few of them:

- The uniform weight: $w_i = 1$ and $w_{ij} = 1/N$;
- $w_i = 1$ and $w_{ij} = \frac{1}{2}(\cot\alpha_{ij} + \cot\beta_{ij})$;
- $w_i = 1/A_i$ and $w_{ij} = \frac{1}{2}(\cot\alpha_{ij} + \cot\beta_{ij})$ (where A_i is a local area corresponding for v_i)

3 Classes

There are 2 cases inheriting from a single base class (which is not meant to be instantiated).

3.1 Hard Constraints

For hard constraints, one should first choose the type of sparse solver to be used. For performance purpose we strongly advise to use *VNLSparseLUSolverTraits*, but it is possible to use a custom one with external libraries not delivered in ITK (such as TAUCS, CHOLMOD, etc.).

```
typedef VNLSparseLUSolverTraits< CoordType > SolverType;
typedef itk::LaplacianDeformationQuadEdgeMeshFilterWithHardConstraints< MeshType, MeshType, SolverType >
```

The next step is then to allocate the filter and set the input mesh.

```
FilterType::Pointer filter = FilterType::New();
filter->SetInput( reader->GetOutput() );
```

Then set the order (note that **we recommend using order 2**, but it is possible to use higher-orders).

```
filter->SetOrder( 2 );
```

Then set the method to compute the coefficient. Although theoretically it is recommended to use *itk::ConformalMatrixCoefficients*, it is possible to use other weights to speed up the deformation, or for other reasons...

```
typedef itk::ConformalMatrixCoefficients< MeshType > CoefficientType;
CoefficientType coeff;
filter->SetCoefficientsMethod( &coeff );
```

Then set the constraints, either by the means of *SetDisplacement*, either by the means of *SetConstrainedNode*

```
MeshType::VectorType null( 0. );
filter->SetDisplacement( 150, null );

MeshType::VectorType d( null );
d[2] = -5;

filter->SetDisplacement( 2030, d );
```

Finally call the *Update* method and get the output mesh

```
filter->Update();

MeshType::Pointer output = filter->GetOutput();
```

3.2 Soft Constraints

As above (for hard constraints), one should first choose the type of sparse solver to be used. For performance purpose we strongly advise to use *VNLSparseLUSolverTraits*, but it is possible to use a custom one with external libraries not delivered in ITK (such as TAUCS, CHOLMOD, etc.).

```
typedef VNLSparseLUSolverTraits< CoordType > SolverType;
typedef itk::LaplacianDeformationQuadEdgeMeshFilterWithSoftConstraints< MeshType, MeshType, SolverType >
```

The next step is then to allocate the filter and set the input mesh.

```
FilterType::Pointer filter = FilterType::New();
filter->SetInput( reader->GetOutput() );
```

Then set the order (note that **we recommend using order 1**, but it is possible to use higher-orders), and the lambda value which balance in between interpolation and approximation.

```
filter->SetOrder( 1 );
filter->SetLambda( 1. );
```

Then set the method to compute the coefficient. Although theoretically it is recommended to use *itk::ConformalMatrixCoefficients*, it is possible to use other weights to speed up the deformation, or for other reasons...

```
typedef itk::ConformalMatrixCoefficients< MeshType > CoefficientType;
CoefficientType coeff;
filter->SetCoefficientsMethod( &coeff );
```

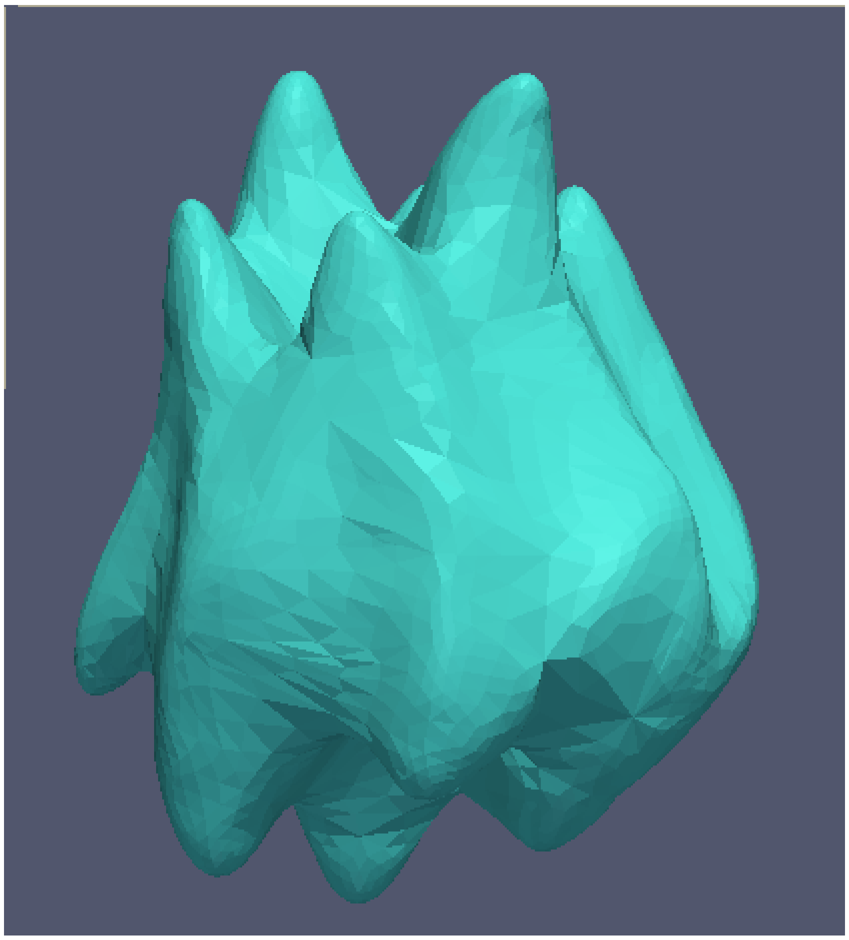
Then set the constraints, either by the means of *SetDisplacement*, either by the means of *SetConstrainedNode*

```
MeshType::VectorType null( 0. );
filter->SetDisplacement( 150, null );

MeshType::VectorType d( null );
d[2] = -5;

filter->SetDisplacement( 2030, d );
```

Finally call the *Update* method and get the output mesh

```
filter->Update();

MeshType::Pointer output = filter->GetOutput();
```

4 Example



Left: Input mesh. **Middle:** Resulting mesh with hard constraints deformation. **Right:** Resulting mesh with soft constraints deformation.