Assignment 3: Discrete Normals, Curvatures, and Smoothing

In this exercise you will

- Experiment with different ways to compute surface normals for triangle meshes.
- Calculate curvatures from a triangle mesh.
- Perform mesh smoothing.
- Familiarize yourselves with the relevant implementations in igl

1. Vertex Normals

Starting from the igl tutorial, experiment with different ways to compute per-vertex normals. A good starting point is the documentation inside function `per_vertex_normals`. Also check this tutorial for the discrete Laplacian calculation. You will need to compute and switch between the following types of normals:

- **Standard vertex normals:**These are computed as the unweighted average of surrounding face normals.
- **Area-weighted normals:** Same as above, but the average is weighted by face area.
- **Mean-curvature normals:** Apply the cotangent-weighted Laplacian to the mesh vertex positions to compute the normal weighted proportionally to mean curvature as shown in class. If this vector field's magnitude at a particular vertex is greater than some threshold, `eps`, use the normalized vector as the vertex's normal. Otherwise, (when the surface is locally flat or possibly saddle-shaped), compute the vertex's normal using the area-weighted average above.
- **PCA computation:** At each vertex `vi`, fit a plane to the `k` nearest neighbors (with `k` configurable) using Principal Component Analysis. The vertex normal is then the principal component with the smallest eigenvalue (i.e., the fit plane's normal). The neighbors can be collected by running breadth-first search or by `np.argpartition`.
- **Normals based on quadratic fitting:** Using the local frame computed at a given vertex with PCA as above, the positions of the vertex and its `k` neighbours can be represented as `[u, v, f(u, v)]` using a height field function `f(u, v)`. This parametric surface is expressed in the local frame's basis (i.e. the principal component vectors), with `u` and `v` giving the tangential coordinates and `f(u, v)` giving the height. Recall, the height axis is the principal component with smallest eigenvalue. Compute the vertex normal by (a) fitting a quadratic bivariate polynomial to the height samples `fi`, then (b) computing the polynomial surface's normal at `u = v = 0` (i.e., at the vertex) by differentiating the surface with respect to `(u, v)` to compute tangent vectors and then taking their cross product.

Note: apart from the first two computations, the normals are defined up the the sign. For the last three computations, align the normal using the normals computed from standard vertex normals.

Relevant igl functions: `per_vertex_normals`, `cotmatrix`, `massmatrix`, `principal_curvature` (look inside for quadric fitting).

Relevant numpy functions: `cross`, `argpartition`, `mean`, `linalg.eig`, `argmin`, `sign`, `column_stack`, `row_stack`, `linalg.lstsq`.

2. Curvature

Compute discrete mean, Gaussian, and principal curvatures using the discrete definitions from class. Color the mesh by curvature with a color map of your choice. Check out this tutorials to begin.

Relevant igl functions: `gaussian_curvature`, `principal_curvature`, `cotmatrix`, `massmatrix`

3. Smoothing with the Laplacian

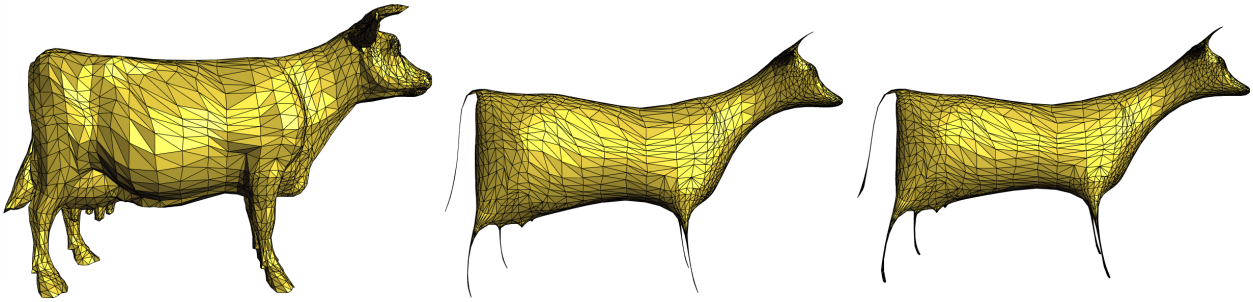

Figure 1. Input mesh, explicit smoothing lambda = 0.01 and 1000 itertions, and implicit smoothing lambda = 20 and single iteration.

Perform both explicit and implicit Laplacian mesh smoothing. Experiment with uniform and cotangent weights. Before you begin, check out this tutorial, which implements implicit smoothing.

Relevant igl functions: `cotmatrix`, `massmatrix`, `grad`, `doublearea`.

Note 1: for explicit smoothing you need to invert the *diagonal* mass matrix, you can do it with

```
import scipy.sparse as sp
M = ...
Minv = sp.diags(1 / m.diagonal())
```

Note 2: for implicit smoothing you need to solve a sparse linear system of the form `Ax = b`, you can use

```
from scipy.sparse.linalg import spsolve
x = spsolve(A, b)
```