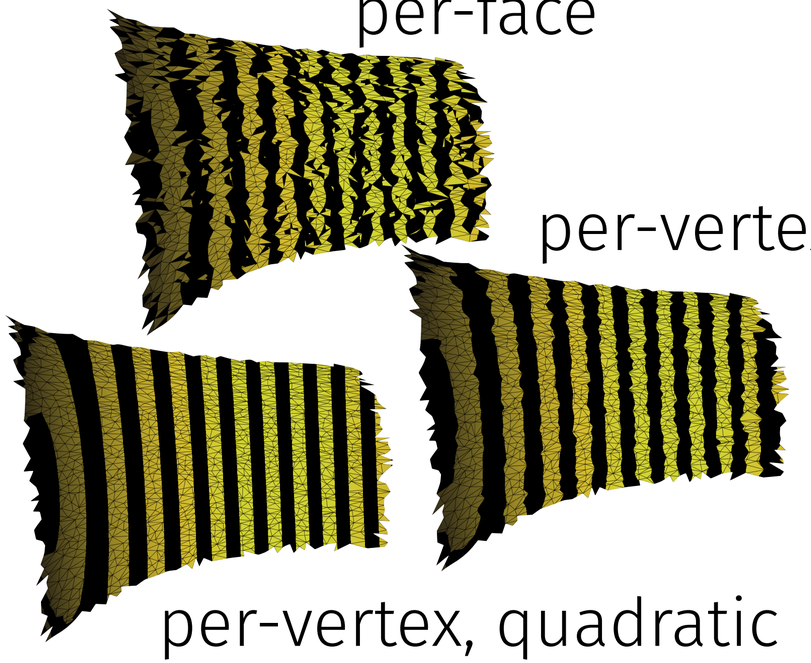
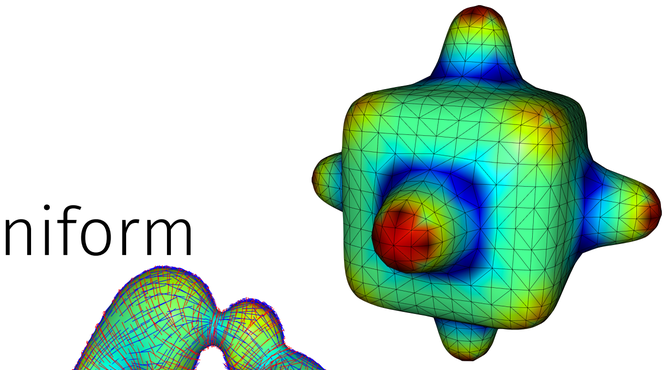
Geometric Modeling
Assignment 3: Discrete Differential Quantities

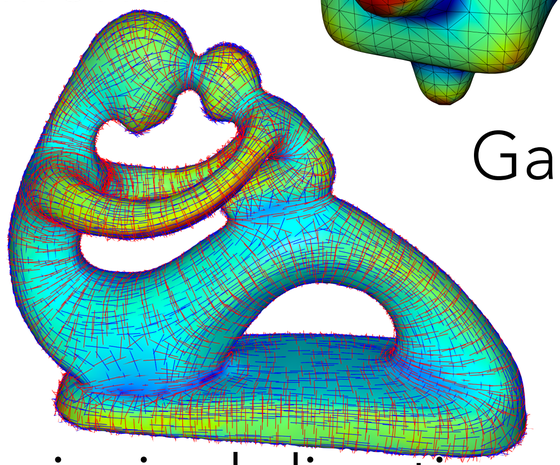
Acknowledgements: Daniele Panozzo, Julian Panetta, Olga Diamanti

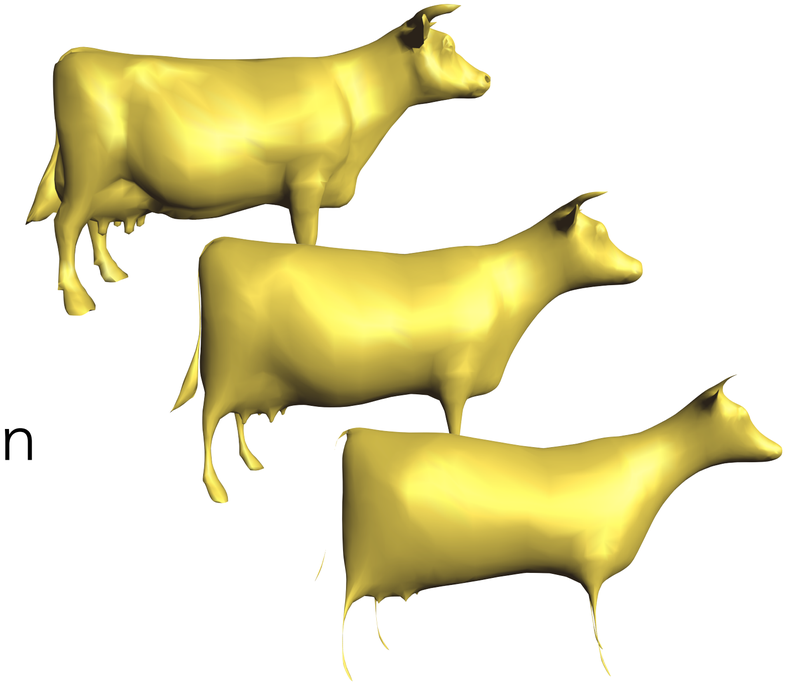
Assignment 3

- Topic: Discrete Differential Quantities
  - Vertex Normals, Curvature, Smoothing (igl tutorials 201-205)




Gaussian


principal directions


Laplacian smoothing

Vertex normals

1. Uniform average: compute the normal of each incident face and average such normals

2. Weighted average: same as above, but normals are weighted according to the areas of triangles

3. Mean curvature: compute the Laplacian of the coordinates function and normalize. Use 1 or 2 do set proper direction.

Vertex normals

4. PCA based: compute the PCA of the cloud of points consisting of the point itself and its k nearest neighbors:

   I. Find the k nearest neighbors

   II. Compute the mean of the cloud of points and center them about it

   III. Compute the covariance matrix $C = V^T V$ where V is the centered point cloud

   IV. Compute the eigen decomposition of matrix C

   V. Take the eigenvectors corresponding to the minimum eigenvalue as normal

   VI. Use 1 or 2 to set proper direction

Vertex normals

5. Quadratic fitting: fit a quadratic function to the *k* nearest neighbors and use its normal at the given vertex *p*:

   I. Find the *k* nearest neighbors of *p*

   II. Set a local frame with origin at *p* and axes *u,v,n* obtained with the PCA (axis *n* corresponds to the smallest eigenvalue)

   III. Express all *k* neighbors in this local frame (dot product with the axes)

   IV. Find a quadratic function in the local frame fitting this point cloud

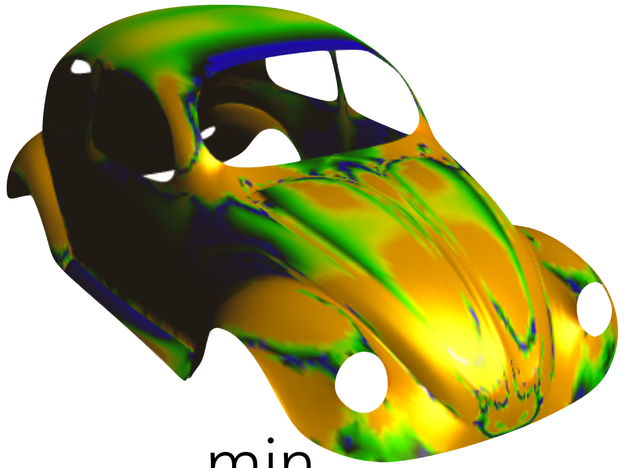
   V. Compute basis of tangent plane at the origin from the derivatives of the quadratic function

   VI. Normal is orthogonal to the tangent plane

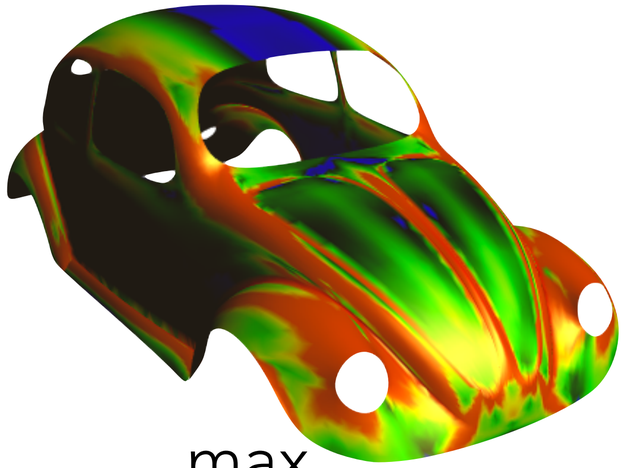
   VII. Use 1 or 2 to set proper direction

Curvature

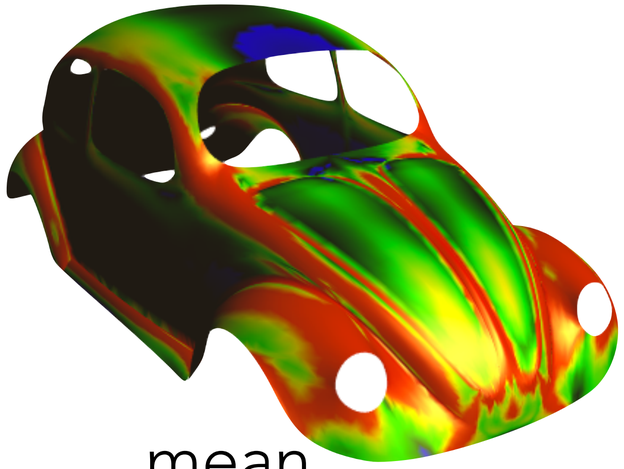
libigl tutorials
Chapter 1



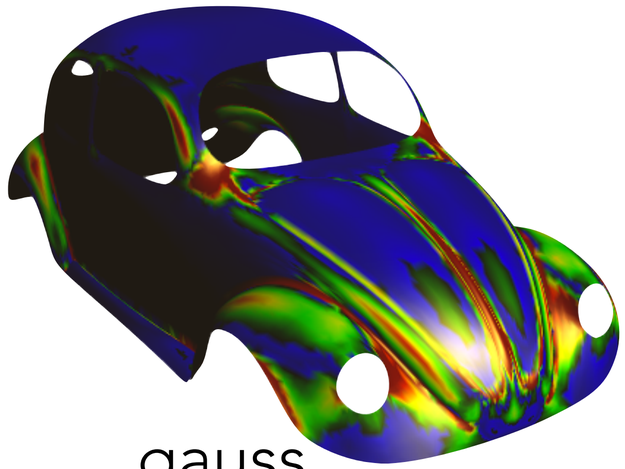
min



max



mean



gauss

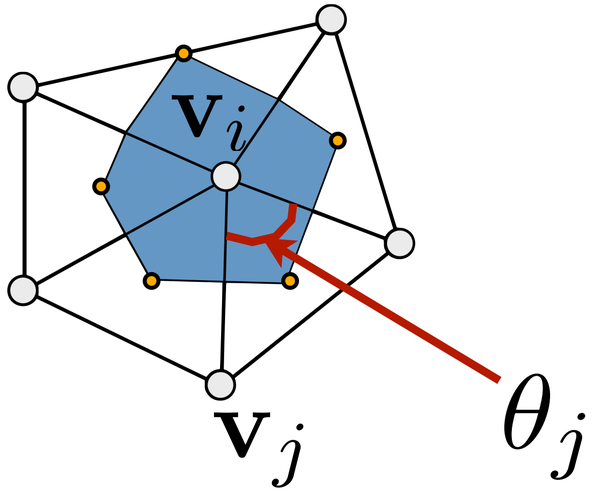
Curvature

- Mean Curvature $\Delta_S \mathbf{p} = -2H\mathbf{n}$

$$H(\mathbf{v}_i) = 0.5\|L_c(\mathbf{v}_i)\|$$

- Gaussian Curvature

$$G(\mathbf{v}_i) = \frac{2\pi - \sum_j \theta_j}{A(\mathbf{v}_i)}$$



- Min Curvature $\kappa_1 = H + \sqrt{H^2 - G}$
- Max Curvature $\kappa_1 = H - \sqrt{H^2 - G}$

$$G = \kappa_1 \kappa_2$$
$$2H = \kappa_1 + \kappa_2$$

Curvature

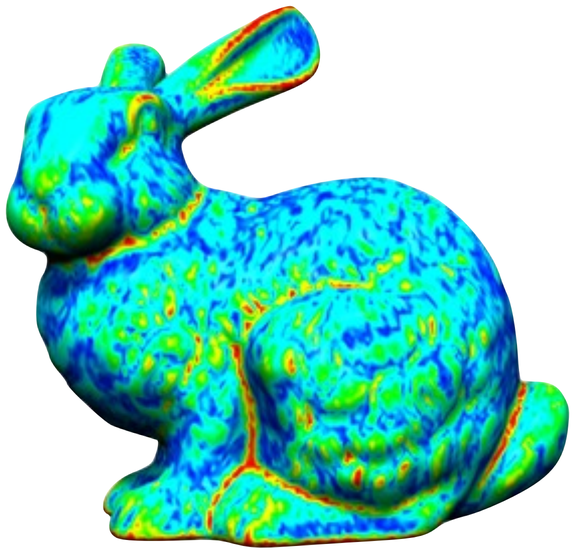
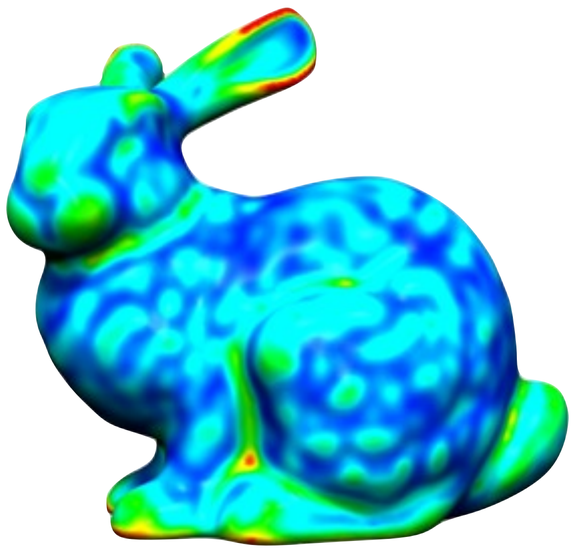
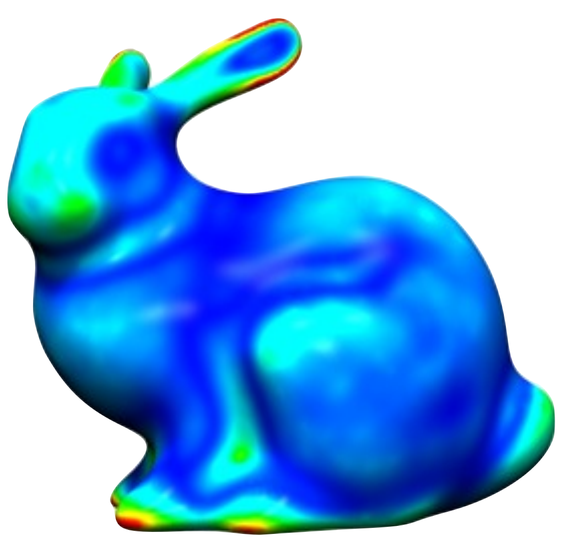
Igl provides functions for computing all such curvatures:

- Mean curvature from Laplace-Beltrami operator (as in Method 3 for normal): functions `cotmatrix` and `massmatrix`
- Gaussian curvature: function `gaussian_curvature`
- Principal curvatures: function `principal_curvature` (based on shape operator from quadratic fitting)

Explicit Smoothing

- In matrix form: $\frac{\partial \mathbf{v}}{\partial t} = \lambda \mathbf{L}(\mathbf{v})$
- Intuition: vertices moving toward neighbor average (“Forward Euler”)

$$\mathbf{v}^{n+1} - \mathbf{v}^n = \lambda dt \mathbf{L}\mathbf{v}^n \qquad \mathbf{v}^{n+1} = (I + \lambda dt \mathbf{L})\mathbf{v}^n$$

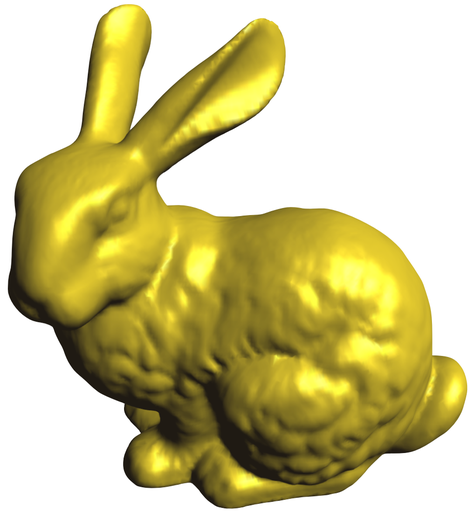




Implicit Smoothing

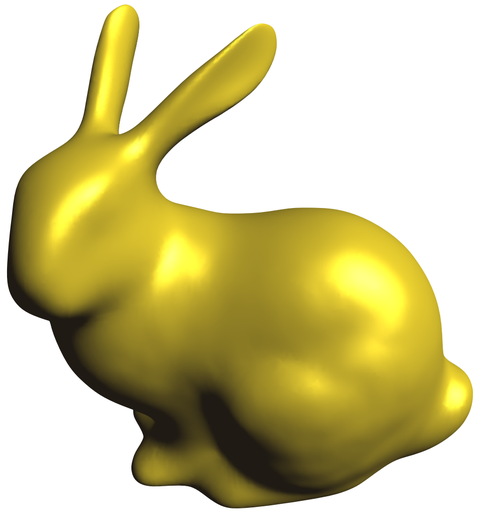
- “Backward Euler” is more stable
- Take larger time steps without mesh “blowing up” (becoming jagged)

$$\mathbf{v}^{n+1} = \mathbf{v}^n + \lambda dt \mathbf{L}\mathbf{v}^{n+1}$$

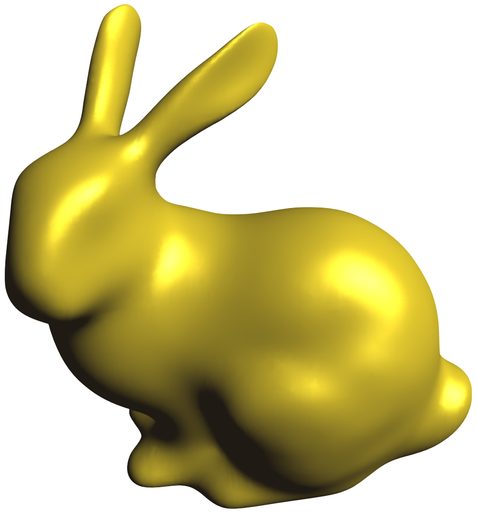
$$(I - \lambda dt \mathbf{L})\mathbf{v}^{n+1} = \mathbf{v}^n$$



original



explicit,1k iterations,
λ = 0.01



implicit, 1 iteration
λ = 20

libigl tutorial
Chapter 1

SciPy Sparse Matrix

- Full-sized Laplacian matrix can be huge for large meshes
  - but most elements are zero!
- Instead, only store non-zero elements: **Sparse Matrix**
- https://docs.scipy.org/doc/scipy/reference/sparse.html

Linear Systems with SciPy

```
from scipy.sparse.linalg import spsolve

A = … # a SciPy sparse matrix
b = … # a NumPy array
x = spsolve(A, b)
```

- Cholesky factorization works only for symmetric, positive definite A!
- Other solvers available, e.g. SparseLU:
  - http://eigen.tuxfamily.org/dox/TutorialSparse.html

Questions?