CS468: Machine Learning for 3D Data Handout # 1
Spring 2017
Stanford University Friday, 14 April 2017

Homework #1: Surface Representations; Converting between representations; Basics of Discrete Geometry : Laplace-Beltrami operator. [100 points]
Due Date: Wednesday, April 26

Homework policies

CS468 is a highly technical course, so doing the homework is the only way to acquire a working knowledge of the material presented. We encourage you strongly to start working on the homework problems right away—the problems below, as well as those to follow, have considerable technical depth and you are unlikely to be able to solve them if you wait until the evening before the due date.

Collaboration in solving the problems is encouraged in this class—you have a lot to learn from your fellow students. However, in order to make grading the homeworks a meaningful way to measure your effort and your understanding of the material, we must ask that:

- *On programming problems [such as in this homework], groups of up to three students can work together as a team, handing in a single body of code and documentation for their joint effort.*

It is very important in this course that every homework be turned in on time. We recognize that occasionally there are circumstances beyond your control that prevent an assignment from being completed on time. You will be allowed two classes of grace during the quarter. This means that you can either hand in two assignments each late by one class, or one assignment late by two classes. Any further assignments handed in late will be penalized by 20% for each class that they are late, unless special arrangements have been made previously with the instructor or the CA(s).

Software

We ask that you use `MATLAB` for this assignment. You can represent meshes in `MATLAB` via the indexed-face-set representation: an $\#V \times 3$ float-valued array for the vertex positions and a $\#F \times 3$ integer-valued array for the faces - the latter one indexes into the vertex position array. You can use functions like `trimesh`, `triplot`, `trisurf` to visualize the triangulation.

Goals of this assignment

In this exercise you will

- Compute unoriented normals from a point cloud using PCA.

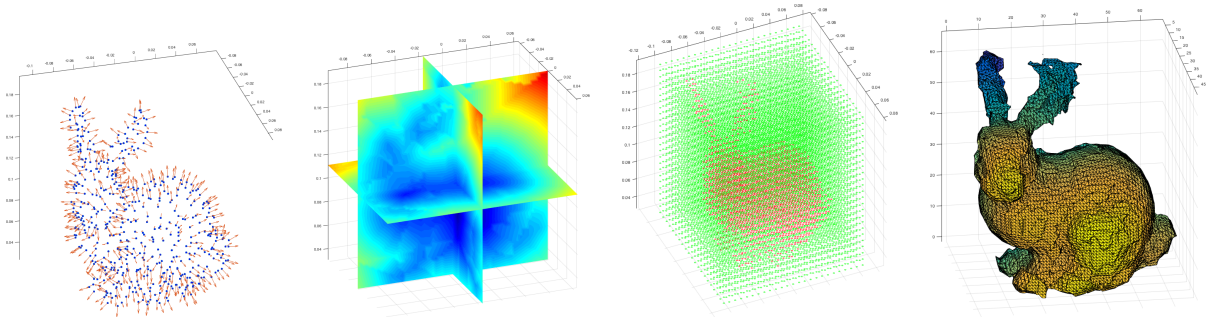

Figure 1: Stages of surface reconstruction from point cloud with normals. Left to right: input point cloud, implicit function slices, same function evaluated on a grid, Marching Cubes result.

- Compute an implicit function that approximates a given 3D point cloud with normals, such that the input points lie at the zero level set of the function.
- Sample the implicit function on a three dimensional volumetric grid.
- Reconstruct a triangle mesh of the zero level set of the implicit function, using the Marching Cubes algorithm.
- Sample the computed mesh to extract a point cloud, using the basic Farthest Point Sampling pipeline.
- Construct the discrete Laplace-Beltrami operator on a mesh, and visualize its eigenfunctions.

This assignment has been structured as a pipeline that can start from a single input (a point-cloud with normals) and proceed through the intemediate steps of implicit surface reconstruction, meshing, sampling and analysis. However, we recommend that you implement each step as an individual `MATLAB` function, so you can test it with different inputs and compare the results. For example, the Marching-Cubes-reconstructed mesh might be too irregular for subsequent Laplace-Beltrami calculations, so you can test your implementation one on of the provided input meshes instead. You should provide a main script that runs those functions and produces the required figures.

Problem 1. [25 points]

For this part of the assignment, you will compute approximate normals for the points in a point cloud, based on local tangent-plane fitting around each point. The procedure is based on PCA, and outlined in the attached document pca.pdf. Note that this procedure produces unoriented normals - consistently orienting those normals is typically needed to reconstruct a surface but will not be required in this assignment.

- Use the provided `readOFF` function to load a point cloud from the provided `.off` files. If any faces are present in the `.off` file, ignore them. Visualize the input point cloud and normals, which will serve as ground truth for comparisons.

- For each point in the cloud, find its K nearest neighbors in the point cloud.
- Fit a plane to this neighborhood to compute the normal direction at the point.
- Visualize those normals along with the ground truth normals as found inside the .off file.

Possibly relevant `MATLAB` *functions:* `quiver3`, `line`, `plot3`, `pdist2` , `eig`.

Problem 2. [20 points]

For this part of the assignment, you will compute an implicit function $f(\vec{p})$, $\vec{p} = (x, y, z)$ defined on all of 3D space, such that the input point cloud lies at the zero level set of this function, i.e. for any point $\vec{p}_i$ in the input point cloud, we have $f(\vec{p}_i) = 0$. The normals of the implicit function evaluated at the point cloud locations should agree with the normals of the point cloud. For this part you will use the normals provided as ground truth in the `.off` files, which have the correct orientation.

2(a). Create a grid sampling the 3D space. Create a regular volumetric grid around your point cloud: compute the axis-aligned bounding box of the point cloud, enlarge it slightly and divide it into uniform cells (cubes). Visualize the grid points.

Possibly relevant `MATLAB` *functions:* `meshgrid`, `linspace`.

2(b). Evaluate the implicit function on the grid points. For each node of the grid, compute the value of the implicit function $f(x, y, z)$ using the simple distance-to-tangent-plane approach that we learned in class ([3]). An example is shown in Fig. 1.

- For each node of your regular grid, find the closest sample from the point cloud and associated normal.
- Use the normal to compute a signed distance from the grid point to the tangent plane associated with the point cloud sample.
- Visualize the 3-D implicit function by slicing..
- Color-code the computed values of the implicit function on the grid points (by positive/negative values) and display them as a colored grid.

Possibly relevant `MATLAB` *functions:* `pdist2`, `slice`, `scatter3`.

Problem 3. [5 points]

For this part of the assignment, you will perform the implicit-function-to-mesh conversion. You will operate on the implicit function that you have computed in the previous part. For an example result see Fig. 1.

- Use the `MATLAB` implementation of Marching Cubes [4] to extract the zero isosurface from your grid.

- Display your result as a triangulation.
- Experiment with different grid resolutions and compare the final result.

Possibly relevant `MATLAB` *functions:* `isosurface, trisurf, cameratoolbar`.

Problem 4. [20 points]

For this part of the assignment, you will convert a mesh into a point cloud. Namely, you will sample points from the mesh using the Farthest Point Sampling approach ([1], [2]). For simplicity, you can sample only from the vertex set of the mesh, and use Dijkstra's algorithm on the mesh graph to compute on-surface distances on the mesh. The general pipeline then is as follows:

- Create an initial sample point set S, that includes a single vertex from the mesh.
- Compute the on-mesh distance between the points in S and the remaining mesh vertices.
- Find the mesh vertex which is the farthest from all point in S according to this distance.
- Insert the point to S.
- While more points are needed, iterate.

Visualize the points on top of the mesh. Experiment with different numbers of samples.

Posssibly relevant `MATLAB` *functions:* `distances`. For computing Dijkstra distances you can also use Gabriel Peyre's Graph Toolbox, found at `https://goo.gl/4PAI2P`

Problem 5. [30 points]

For this final part of the assignment, you will build the Laplace Beltrami operator on the constructed mesh (or any of the given meshes). You will try out both the uniform and cotangent discretizations [6, 5]. When computing the cotangent weight Laplace Beltrami operator, use barycentric area computation, namely: area corresponding to a vertex is computed as a sum of the area of its adjacent triangles, divided by 3.

You will use this operator to visualize the "eigen-modes" of the shape, as computed by the eigenvalues of the Laplacian operator. These eigen-functions provide a basis for all piecewise-linear scalar functions on the mesh, constructed by linearly interpolating per-vertex values into the triangles. When computing the eigendecomposition of the cotangent weight Laplace Beltrami operator, note that the eigenvalue problem in the discrete case is given by (using the notations from the lecture slides)

$$\Delta\phi = \mathbf{A}^{-1}\mathbf{L}\phi = \lambda\phi \quad \Leftrightarrow \quad \mathbf{L}\phi = \lambda\mathbf{A}\phi, \tag{1}$$

where $\mathbf{L}$ is the matrix of cotangent weights, $\mathbf{A}$ is the diagonal matrix with vertex areas on the diagonal, and λ and ϕ are an eigenvalue and its corresponding eigenvector, respectively. Thus above is called a *generalized eigenvalue problem*, and can be solved using `MATLAB` function `eigs`.

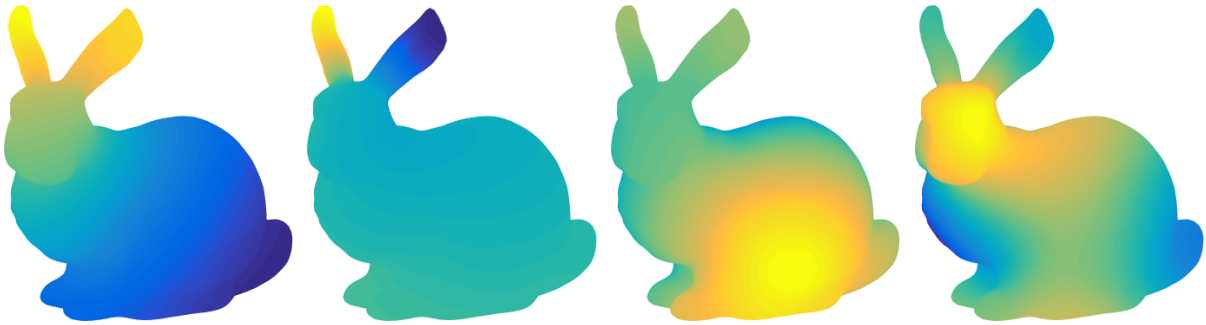

Figure 2: Eigenfunctions #2, #4, #6, #8 of the uniform Laplace Beltrami operator.

- Construct the discrete Laplace-Beltrami operator on a mesh, as a **sparse** matrix of size $\#V \times \#V$. You may need to build a vertex-to-vertex adjacency structure for this, or (more efficiently) compute it by iterating through the face index array directly. Use `MATLAB` vectorization whenever possible.
- Compute its smallest N eigenfunctions, by looking at the eigen-decomposition of that matrix. Each of them will be a piecewise-linear function on the mesh, essentially determined by its values at the mesh vertices.
- Visualize the eigenfunctions as colors over the mesh. Do not forget to switch on linear interpolation inside the faces (e.g., using `shading interp`).
- Experiment with different numbers of eigen-functions N.
- Experiment with the uniform- vs. cotangent-weight schemes, and note the differences, if any.

For an example, see Figure 2.
Posssibly relevant `MATLAB` *functions:* `eigs`, `jet` (or any other colormap).

References

[1] Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293 – 306, 1985.

[2] Dorit S. Hochbaum and David B. Shmoys. A best possible heuristic for the k-center problem. *Math. Oper. Res.*, 10(2):180–184, May 1985.

[3] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Surface reconstruction from unorganized points. *SIGGRAPH Comput. Graph.*, 26(2):71–78, July 1992.

[4] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '87, pages 163–169, New York, NY, USA, 1987. ACM.

[5] U. Pinkall and K. Polthier. Computing discrete minimal surfaces and their conjugates. *Experimental mathematics*, 2(1):15–36, 1993.

[6] Hao Zhang. Discrete combinatorial laplacian operators for digital geometry processing. In *in SIAM Conference on Geometric Design, 2004*, pages 575–592. Press, 2004.