SGI 2021 Friday Tutorial Exercise 1: Power Fields

Date: 23/Jul/2021

Errata

Introduction

The purpose of this exercise is to compute N-RoSy fields on triangle meshes in the power-field representation learned in class. You will set some (hard) constraints on a set of faces, and interpolate to the rest of them in the as-parallel-as-possible paradigm. Following that, you will devise the matching between every two neighboring faces, and from this derive the index of the dual cycles. For simplicity, we will only target simply-connected meshes without a boundary, which means each dual-cycle is just the 1-ring around a single vertex.

The exercise

The main practical directory comprises two subfolders: `code`, where there are the main codes files, including the ones you need to update, and `data`, containing the meshes. The files contain envelope and setup code that is (hopefully!) well-documented enough for you to work with; you need to complete to parts marked as "`TODO:`". In the end of the day, you will receive the reference solution, and can compare your approach and reflect on it.

The practical will automatically plot the results of the steps of your solution, properly titled. Currently the plots will be of trivial results, which have zeroes as stub values; these are the variables you need to fill in.

1 Computing Power Fields

The setup The envelope script which runs the algorithm and visualizes the result is `PowerFields.m`. The envelope script proceeds as follows (see Figure 1 for how the result looks when complete).

- Loading a triangle mesh with `readOBJ()` into variables `V` and `F`, and using `make_edge_list()` to compute edge-based quantities.
- Computing required geometric quantities: normals, face areas, face barycenter, and basis vectors B_1, and B_2. Some of them are left for you as a warm-up exercise (see below). The basis is subsequently visualized.
- Choosing the constraints on a small set of faces, and the order N of the field. The field is then interpolated to the rest of the faces. This is your main task for the first part of the exercise.
- Visualizing the field on the original mesh.

1.1 Computing smooth power fields

We consider the basis $B_1(f), B_2(f) \in \mathbb{R}^3$ per face $f \in F$, and use it to define a local complex plane: every "raw" vector $v(f) \in \mathbb{R}^3$ (in the xyz representation) is represented as a single complex number $\mathbf{v} \in \mathbb{C}$ where:

$$\mathbf{v}(f) = \langle v(f), B_1(f) \rangle + i \cdot \langle v(f), B_2(f) \rangle$$

.

The inverse conversion is straightforward to derive. To compare between two vectors in neighboring tangent planes that we denote as two faces $f, g \in F$, which are adjacent through a mutual edge e, we need to translate the bases: two vectors are perfectly parallel when:

$$\mathbf{v}(f)\overline{e_f} = \mathbf{v}(g)\overline{e_g},$$

where e_f and e_g are the complex representations of the *unit* edge vector e in the respective bases of f and g—this is akin to locally rotating both bases to agree on e as a mutual x axis. We next consider a field that has N vectors per face in a perfect rotational symmetric configuration, also called an "N-RoSy field". We represent it using the *power representation*, where we encode an entire symmetric set using a single complex vector $\mathbf{y}(f)$ per face, where:

$$\mathbf{y}(f) = \mathbf{v}(f)^N.$$

The parallel relation then generalizes to:

$$\mathbf{y}(f)\left(\overline{e_f}\right)^N = \mathbf{y}(g)\left(\overline{e_g}\right)^N.$$

We cannot have a perfectly parallel field in the presence of curvature, and we consequently solve this equation only in the least-squares sense. That is, we minimize the objective function:

$$E(\mathbf{y}) = \sum_e w_e \left| \mathbf{y}(f)\left(\overline{e_f}\right)^N - \mathbf{y}(g)\left(\overline{e_g}\right)^N \right|^2. \tag{1}$$

This can be described as a linear least-squares system of the following concise representation: $E(y) = W\left|A\mathbf{y}\right|^2$, where:

- $\mathbf{y}$ is a *complex* vector of size $|F|$, encoding the power field.
- A is a complex matrix of dimensions $|E| \times |F|$ encoding the covariant derivative (deviation from parallel transport). Essentially, every row contains $\overline{e_f}^N$ and $-\overline{e_g}^N$ in the resp. column indices f and g, and zero everywhere else; you need to use a sparse matrix!
- W is a *diagonal* real matrix of dimensions $|E| \times |E|$ containing the weights of each equation. We use the geometric weights learned in class: $\frac{3l_e}{A_f + A_g}$, where l_e is the length of the original edge e, and A_f, A_g are the respective areas of triangle f and g. This is again of course a sparse matrix with just diagonal values.

Prescribed constraints The system presented above is solved by $y \equiv 0$. To avoid the trivial solution, and obey user alignment constraints, we designate a subset $B \subset F$ of faces where we prescribe $\mathbf{y}(b \in B)$ in advance. Consider the remaining free faces $I = F \setminus B$, in which we want to obtain the interpolated values $\mathbf{y}_I$. We then separate A, by slicing columns, into A_I and A_B, and our objective function becomes:

$$E(\mathbf{y}_I) = W \left| A_I \mathbf{y}_I + A_B \mathbf{y}_B \right|^2 .$$

The solution is obtained by solving the linear system:

$$A_I^T W A_I \mathbf{y}_I = -A_I^T W A_B \mathbf{y}_B .$$

Having computed $\mathbf{y}_I$, we can recombine it with the prescribed $\mathbf{y}_B$ to get the full $\mathbf{y}$, and consequently retrieve a single vector in each face by $\mathbf{u} = \sqrt[N]{\mathbf{y}}$. The other vectors in the symmetric set are obtained by rotation (this is done in the visualization). See Figure 1 for how the result should look like.

Practical considerations

- The user parameters are the order of the field N, the constrained faces and their prescribed field values. You should play with these values to get the feel of the result. I suggest to start with $N = 2$ or $N = 4$ which produce intuitive results, and then venture to less conventional choices like $N = 3, 5, 6, 7$. Also start with smaller meshes until you are convinced it works, and then move on to the bigger ones.
- You will be doing a lot of indexing into V and other variables. For instance, to get all edge vectors you should do:

```
edgeVectors = V(EV(:,2),:) - V(EV(:,1),:);
```

- Other valuable functions are `cross(,2)`,`dot(,2)`, `spdiags()`, `normv()`, `setdiff()`, and `accumarray()`. A linear system of the type $Cx = d$ should be solved as `x=C\d;` (also known as `mldivide`).

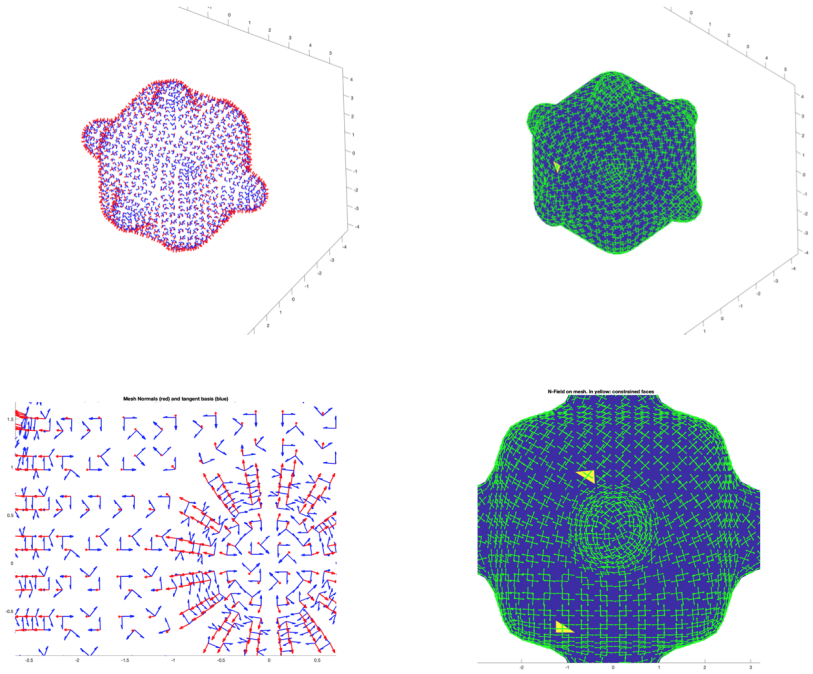

Figure 1: Power field computation. Left to right (bottom is zoom-in): original mesh with local bases, final N-RoSy field for $N = 4$. In yellow are the constrained faces.

2 Principal Matching and Singularities

We next devise the topology of the field. This will be done through a continuing envelope script `PrincipalMatching.m`, most of which you will be implementing.

Principal matching we first have to consider which vector goes to which on neighboring faces, and measure the *effort*, which is the sum of deviations from parallel transport. In the case of an N-RoSy, the effort $\Theta(e)$ simplifies into:

$$\Theta(e) = \arg\left(\frac{y_g \overline{e_g}^N}{y_f \overline{e_f}^N}\right).$$

Note that this is 0 for a perfect parallel transport. Around a basis cycle, which is our case a set of edges going counterclockwise around a vertex, the (oriented!) effort has to sum up to $-NK(v) + 2\pi I(v)$, where the first part cancels out the Gaussian curvature (the field as a "trivial connection", as learned in class), and the second part is the amount of turns an N-directional set does when returning to the original face (multiplied by N). To get the index $I(v)$ of every vertex $v \in V$, we then compute:

$$I(v) = \frac{1}{2\pi N}\left((d_0)^T \Theta + NK(v)\right)$$

Let us understand the different terms of this computation:

- d_0 is the (sparse) *differential* matrix of dimensions $|E| \times |V|$ encoding the orientation of an edge against an adjacent vertex. That is, for every edge (i, j) (where i is source and j is target), there is a single row of d_0 with -1 in column i and 1 in column j, and zero otherwise. The effect of $(d_0)^T$ is to sum quantities from edges (in our case the effort) to vertices, where we factor in the orientation, so that we sum everything in the correct CCW order.

- $K(v)$ is the discrete *angle defect*. That is, for every vertex v with adjacent corner angles $\{\alpha_1, \ldots \alpha_d\}$ (that means d adjacent faces), we have: $K(v) = 2\pi - \sum_i \alpha_i$. In fact, most of your code will be about computing this quantity! This should be implemented in the `GaussCurvature()` function. Use `accumarray()` to accumulate quantities from faces (the angles) into vertices.

Having computed the indices $I(v)$, you will see that most of them are 0; a small set will be $\pm\frac{1}{N}$.

Confidence checks The script contains several confidence checks:

1. That the angles you computed sum to π in a face; this is very useful since the oriented dot products to compute them are confusing! a good result means `angleError` is almost 0 (expect $< 10^{-11}$).

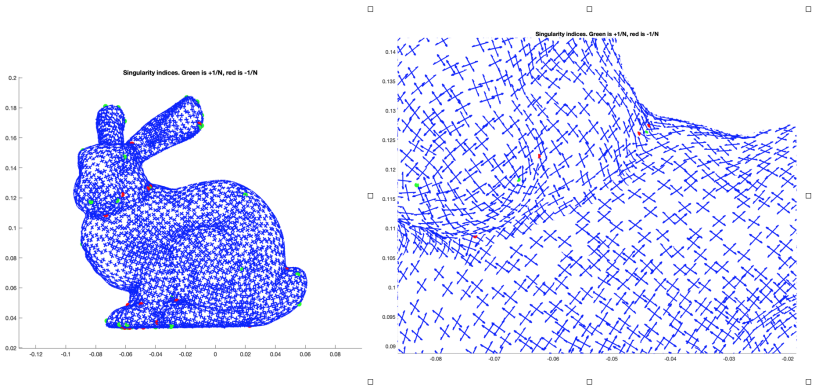

Figure 2: The singularities of the field (green are positive and red are negative).

2. That the Gaussian curvature conforms to the Gauss-Bonnet Thoerem; namely, that:

$$\sum_v K(v) = 2\pi \left(|V| - |E| + |F|\right) = 2\pi\chi$$

. The result is in `angleDefectError`.

3. That the indices you got are integer multiples of $\frac{1}{N}$. The result is in `indicesIntegerError`.

4. That the indices also sum up to the Gauss-Bonnet theorem (of the induced curvature of the field as a trivial connection). That is that:

$$\sum_v I(v) = \chi.$$

The singularities will be shown as spheres on the mesh with the field, as seen in Figure 2.