CS6491- Fall 2019 - Project 3: Vector Field

Teams of up to 3, due Nov 21, presentation at last 2 lectures

ABSTRACT

*The goal is to invent, implement, test, and improve algorithms for: (Phase 1) Designing and visualizing a continuous **vector-field** F over a (planar), **base triangle-mesh**, M, (Phase 2) Constructing a **field-aligned triangle mesh**, T, for which most triangles have one edge aligned with the flow lines of F, (Phase 3) explore **applications**: (a) to the design of housing subdivision based on terrain elevation, (b) to computing meshes over planar regions that align with stress lines, (c) to the artistic or informative hatching of curved surfaces represented by triangle meshes, (d) to other challenges in various fields, or **extensions:** (e) quad meshes in the plane or in 3D or (f) to tetrahedral meshes .*

1 Teamwork

Each team, no matter the size, should deliver Phase 1 and Phase 2.

But teams of 2 or 3 are expected to **also** deliver more advanced solutions for these phases (as discussed below), and to demonstrate their advantages.

Teams of 2 or 3 are expected to include an initial exploration of Phase 3, which includes a review of the most relevant prior art, the outline of a proposed approach, and a proof of concept or demo for a simplified version of the problem.

Teams of 3 are expected to include an honest attempt at a full solution, a demonstration of what it can do, and an analysis of its limitations.

2 Deliverables: Nov 21

2.1 Paper

Short paper with formal (journal/conference) layout (header, abstract, problem statement, outline, details, results, discussion, prior art).

Suggested length: 2 pages per person.

2.2 Sketch

Sketch with the data files and a readme explaining how to run it and how to get a self-guided tour of the main functionalities.

2.3 Presentation

A short presentation (3 mns per person) with a few slides (each having a few words and images or math formulae or pseudocode) and a short video showing a demo of (1) the design, editing, and visualization of the vector field, (2) the construction and visualization of the field-aligned triangle mesh, and, when applicable, (3) the application to the chosen domain.

3 Sources

3.1 Prior art

Please discuss briefly and cite properly all prior art that inspired or guided your solution or that may provide the reader with an overview of the state of the art.

3.2 Source code

You do not need to cite the source code that I provide. If you use or replicate code from other sources (other projects, online depositories, libraries), you should clearly explain what they do, how they do it, how you use them, how you integrated them with your applications and you should provide links and citations.

4 Phase 1: Base Triangle-mesh and its Vector-Field

4.1 Base triangle-mesh M

Start with code that lets the user load a point cloud in 2D, edit it (by adding, deleting, or moving points OR by perturbing all points by a small random vector). Use or write code that (1) builds a Delaunay triangulation of these points, (2) builds a Corner Table for the

corresponding **base triangle-mesh**, M, and (3) supports corner operators (c.n, c.s, c.v, c.t, c.l, c.r, c.o), the visualization of the edges, vertices, and selected corners, and the mouse-click selection of a corner.

4.2 Specifying vector-field constraints

Use an array F[v] of vectors, each associated with a vertex v and a Boolean array constrained[v], which identifies which of the vectors are constrained by the user or the application.

Let the user click near a vertex of M, drag the mouse, and release. Compute the vertex v closest to the click location and associate with it the vector F[v] between the click and the release locations. When 'c' is pressed, clicking on a vertex v should toggle its constrained[v] status. Display constrained vectors in red and other vectors in dark green.

4.3 Solve for the unconstrained vectors

Invent a simple, yet reasonably efficient solution for computing the unconstrained vectors. I suggest to start with a smoothing solution which performs one or two iteration of the {tuck, untuck} combination of each vector (including the constrained ones) and then snaps the constrained ones back to their constrained value. This may be slow to converge. So, initially, perform one or a few these smoothing steps at each frame so that the response to editing the vertices or vectors is still realtime (some 20 frames per second) and the smoothing evolves over dozens of frames towards a stable state.

An idea for the tuck is:

- For each triangle, compute the average of the vectors of its vertices
- For each vertex, compute the average of the vectors of its incident triangles

Allow the user to press 'A' to toggle between linear and log-spiral averaging of vectors. In log-spiral averaging, uses the arithmetic average of the angles (which is not trivial to do because the sum of angles may be more than PI or less than -PI) and geometric average of the magnitudes.

4.4 Extra credit for inventing other means for designing the field

For extra credit, explore strategies (for example Conjugate Gradient) for faster convergence or different approaches that let the user design smooth, yet complex fields interactively. For example, you may use a combination of a bilinear vector field controlled by 4 corner vectors of a unit square domain and warp that field using the COTS map (code provided). Then, you may use the COTS inversion to compute the vector F[v] at each vertex of M.

5 Phase 2: Field-aligned triangle-mesh T

5.1 Compute and visualize the vector in each triangle

Assume that the vector field is linear inside any given triangle t. Implement a query V(P,A,A',B,B',C,C'), which, given vertices (A,B,C) of t, given the associated vectors (A',B',C') and given a query point P (typically, but not necessarily, in t), returns the vector V(P…) at P. Explain why using here the log-spiral averaging would not be appropriate.

Visualize the field by showing V(P…) at the centroid of the perimeter (**CoP**) of each triangle (weighted average of the mid-edge points with weights proportional to the edge-lengths).

5.2 Trace the Steady Affine Motion (SAM) from a CoP

Implement a naïve tracing, where you start at a P = CoP of a user-selected triangle, compute and make small steps (P=P+sV(P)) until you exit the mesh. Show the **path** as a continuous line; we shall call it a trace. Show, on top of the trace, in a different color, samples. The spacing between the samples should be roughly proportional to the local magnitude of V(P). Color code the triangles visited by the path.

5.3 Traces and samples

Invent an algorithm that produces traces that are in general long and as uniformly spaced as possible. There are many approaches. Before you read about them, consider the one proposed below and try your own. Then, improve one of these if you find good inspirations.

Let the user pick a triangle t and seed the first path with its CoP. Trace it in both directions, marking the triangles **visited** by the path (and painting them in a different color).

In each direction, stope when the path exits the mesh M or tries to enter an already visited triangle.

Repeat this process, where the user can only pick a not-yet-visited triangle to seed a new path. See, whether you can make this work and you like the results. Some paths may be confined to a single triangle, because they would exit the starting triangle onto an already visited one. Reduce their path to a single point: the CoP of that triangle.

Decide whether picking triangles randomly will work. If so, provide that option. If not, try to figure out rules for picking good triangles.

5.4 Constrained Delaunay Triangulation

Adjust the sampling density so that, on average, you get about one sample per triangle of M. Of course, some triangles traversed by a path will not get any sample, because the path traverses them close to a corner and has a short segment in that triangle. Other triangles may have two or more segments of the path that crosses them.

Write an algorithm that uses the samples along these paths as vertices of the Field-Aligned mesh T. Furthermore, use **constrained-edges** between consecutive samples of the path as edges that must be edges of T (these must be edges of T and cannot be crossed or split). Devise an algorithm for computing the triangles (and hence the other edges) of T, so that T is a valid planar triangulation. Note that about a third of its edges will be constrained-edges, which are aligned with the field F.

If this proves too challenging, consider splitting, when needed, some of the constrained edges by inserting new vertices.

Note that you may try to use the "disk-bulging" paradigm as you progress along the corridors between paths. The Disk may reach a bifurcation and may have to split or merge. So, to avoid these complications, it may be easier to use local decisions.

Here's an idea. For each constrained-edge, create two triangles, one on each side. Use "constrained-bulging" (to avoid making a triangle that crosses a neighboring path) to locate the best tip for each triangle. I expect that, if the field turns quickly, the tip may be a sample along the same path, so be careful. This, I think, will produce a mesh T with holes. Build its Corner Table and try to fill the holes.

5.5 Plausible

Show the edges of (the original or the refined) T that are aligned with the vector-field F using a stronger color and greater thickness. There are the stress supporting ones.

Make a version of your program that extends to not convex base-meshes (for example by deleting triangles of M that have a large associated Delaunay circle).

Create an example in the form of a bridge or dome or some support structure, or a plate with 3 circular holes. Constrain the field F so that the stress lines follow plausible paths for each example.

5.6 Extra-credit: Refinement

For extra credit, provide a key and an algorithm for subdividing each triangle of T into 4 triangles, for building the corresponding Corner Table.

Then, consider the set of new interior-edges (3 per triangle). Adjust their vertices so as to align some of them to form long path along the vector field.

6 Phase 3: Other applications

This is up to you, but I will gladly discuss in class and give advice about whether they are sufficiently cool and/or about ideas for approaching the associated challenges or simplifying the goals to a manageable set.

7 Ethics

You can ask other students or colleagues for help about Processing or about where to look for prior art, but you cannot ask for help with the technical parts of the project. You should solve them yourself. If you cannot, you may (without penalty) look for solutions or **inspirations** online or in prior art, but, if you find any, you MUST report exactly what you found and where AND you also MUST include a clear and detailed **tutorial** that could be used to help someone else derive these results without looking at the solution. (The idea here is to have you study and understand that inspiration so clearly that you could teach others how to derive it. You may use an additional page in the report or an additional minute of the video for each one of these.)

8 Video and PDF Reports

Each team should also submit a short video showing user interactions and animations produced by your program and your extension. The video is mandatory. It may be up to 3 mns long (MAX!). But shorter is better.

After the title screen, which should not last more than 1 sec, show a 6 seconds preview of editing M, editing some constrained vectors, of doing that as the other vectors are computed via smoothing, and finally the resulting mesh T.

The video file should be as small as possible (much less than 100 MB). You may use the '~' key in your program to start/stop recording video-frames and use "Tools > Movie Maker" to convert these video-frames to a movie.

Both the PDF and the video must include (on front page of PDF or initial screen of the video):

9 Tips for good writing: The 7 C's

Your report should, as much as possible, satisfy the 7 Cs:

- **Concise**: Avoid unimportant details and redundant words
- **Concrete**: Use simple and intuitive concepts instead of (or to illustrate) abstractions, which may be ambiguous to the reader
- **Clear**: Strive to make the structure and explanations clear to the intended reader (students, TA, instructor, future boss)
- **Correct**: Make sure that your math, statements, derivations are correct and all terms defined unambiguously
- **Convincing**: Include convincing arguments for statements that you believe to be true (or label them as "conjectures")
- **Complete**: Mention all cases. Say that you omit some cases for lack of space, but say briefly how they could be addressed.
- **Clever**: Try to simplify the solution or derivation by a clever choice of parameters, approaches or concepts

Start each paragraph with words that announce what the paragraph conveys. Place the key result (the one that the reader should remember) at the end of the paragraph.

Include very short "glue" sentences between paragraphs that clarify the flow and logical relation between consecutive paragraphs.