Homework 1 - Geometry Representations

Handout date:	January 12, 2021
Submission deadline:	January 21, 2021, 11:59pm

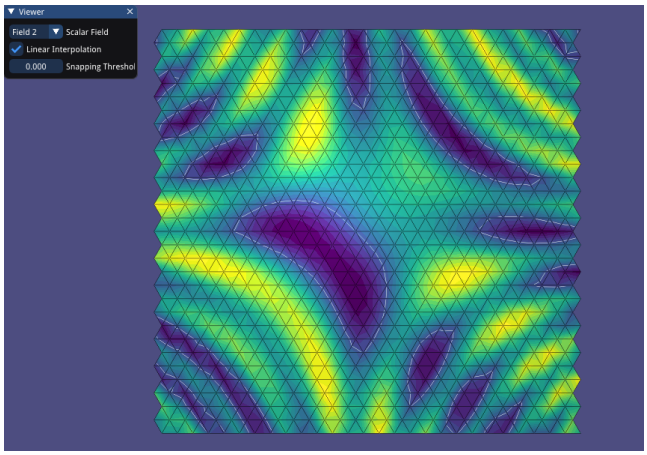

Figure 1: Expected view once you've completed the assignment (displaying scalar field F_{iii}).

1 Theory Part (20pts)

Please submit your solutions to the following problems as a PDF. This can be a legible scan of a handwritten solution if necessary, but a typeset solution is preferred.

1.1 Implicit Functions

i) *(3 pts)* Derive an expression for the signed distance to the half-space defined by a plane passing through point $\mathbf{p}$ with outward-pointing normal vector $\mathbf{n}$. Your expression should be positive *outside* the half-space (i.e., in the region $\mathbf{n}$ points towards) and negative *inside*.

ii) *(3 pts)* Derive a signed distance function for an annulus centered around the origin, with inner radius r and outer radius R. (The function should be negative inside the annulus.)

iii) *(3 pts)* Combine the two previous expressions to obtain an implicit function defining the portion of the annulus in the positive half-space.

iv) *(3 pts)* Calculate the **unit** normal for the ellipsoid defined by the level set function

$$F(x,y,z) = \left(\frac{x}{a}\right)^2 + \left(\frac{y}{b}\right)^2 + \left(\frac{z}{c}\right)^2$$

as a function of the evaluation point and parameters a, b, and c.

1.2 Parametric Curves

i) *(4 pts)* Define a planar curve possessing a sharp corner (discontinuous normal vector) using only polynomials as coordinate functions. Please give a parametric representation.
Hint: we saw a related example in class using trigonometric functions.

ii) *(4 pts)* Suppose a 2D curve $[x(t), y(t)], t \in [0,1]$ is discretized into a polyline using N uniformly spaced sample points $\mathbf{p}_i = \left[x\left(\frac{i}{N}\right), y\left(\frac{i}{N}\right)\right]$ for $i = 0, \ldots, N$. As we increase the number of sample points N, does the discrete polyline's length monotonically increase (never decrease)? If yes, give a proof; if not, give a counterexample.

2 Coding Part: Contour Extraction (80pts)

You will implement a simplified version of the marching squares contour extraction algorithm introduced in class. The algorithm will operate on grids of triangles instead of squares, which turns out to greatly simplify the cases that must be handled:

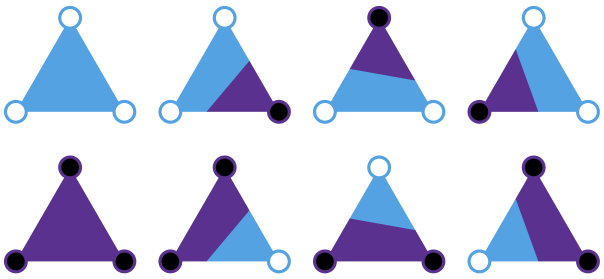

Figure 2: The 8 possible cases for contouring using a triangle grid (Wikipedia). The white/black colors indicate values above or below the zero contour.

Notice in particular the absence of ambiguous cases: there can only be 0 or 2 contour points in a triangle, so it is obvious which points need to be connected by edges.

2.1 Getting Started

If you haven't already, follow the instructions from `hw0` to prepare your computer to build the assignment code. Then download and unzip `hw1.zip` into `DGP_HW` and make sure it builds/runs:

```
cd DGP_HW/hw1
mkdir build && cd build && cmake .. -GNinja
ninja && ./hw1 ../meshes/mesh3.obj
```

Notice `hw1` expects a triangle grid mesh; several are provided in the `meshes` directory for you to try.

2.2 Basic Contouring Algorithm (40pts)

Start adding the missing code described by the `TODO` comment in `extract_contours.cc`. For each triangle, sample the passed implicit function `sf` at the corners (passing the corner vertex positions to `sf`). Determine the signs of the sampled values (you can define 0 as "positive"), and use the signs to determine which edges need contour points.

If contour points exist for a triangle, add them at the midpoints of the corresponding edges (add these midpoints to `contourPts`), and add a connecting edge to `contourEdges`.

2.3 Nicer Contours (20pts)

Placing contour points always at the edge midpoints will produce jagged contours. We can obtain a much higher quality contour using piecewise linear interpolation. Specifically, to place the contour point for an edge e, we solve for the exact zero crossing of the linear interpolation along e of the function values sampled at the endpoints. Derive and implement expression for this point. Chose between inserting this point and the original edge midpoint based on the passed `linearInterpolation` parameter.

Hint: suppose the edge has endpoints $\mathbf{p}_1, \mathbf{p}_2$ and corresponding implicit function samples s_1 and s_2. If you parametrize the edge using using $\mathbf{p}(t) = (1-t)\mathbf{p}_1 + t\mathbf{p}_2$, then the linearly interpolated field values along the edge are given by $s(t) = (1-t)s_1 + ts_2$.

2.4 Avoid Duplicated Contour Points (15pts)

The basic contouring algorithm described in Section 2.2 will create two copies of each contour point on interior edges of the triangle grid mesh. This is wasteful and causes problems if we want to traverse the contour as a connected polyline (see Section 3.2): we cannot determine edges that connect with each other just by looking for edges sharing a contour point index.

Use the simple `pointOnTriEdge` data structure to ensure that at most one contour point is generated for every (undirected) edge of the triangle grid. (Before deciding to add a point to an edge, check that one does not already exist. If it does, just reuse that point.)

You do not need to keep the original duplicate-point-generating code path in your implementation. If your implementation is correct, you should get a smooth approximation to the circle with the provided demo implicit function, and the contour should evolve smoothly when you press the `WASD` keys to translate the implicit function relative to the grid.

2.5 Implement Additional Functions (5pts)

Edit `main.cpp` to include the following additional implicit functions (F_i is already implemented):

- $F_i(x, y) = \sqrt{x^2 + y^2} - 1$
- $F_{ii}(x, y) = y^2 - \sin(x^2)$
- $F_{iii}(x, y) = \sin(2x + 2y) - \cos(4xy) + 1$

You can also use your code to test the expressions you derived in the theory part (1.1 i-iii).

3 Bonus Opportunities

Implementing the following tasks will earn you bonus points that can make up for incorrect solutions on this or future assignments.

3.1 Snapping (10pts)

If a contour slices very close to a vertex of the extraction grid, arbitrarily short contours can be generated in `linearInterpolation` mode. This can cause problems for downstream algorithms operating on the extracted contour, and the issue is particularly serious for the higher-dimensional case of extracting triangle surface meshes with marching cubes (which can generate arbitrarily bad sliver triangles). To avoid this, snap the interpolated crossing point to the nearest edge endpoint whenever its relative distance to this point—relative to the full edge length—is closer than the user-specified `snapEpsilon`.

For full credit, you must ensure duplicate points are not generated on the snapped-to vertex (you can use the `pointOnTriVertex` map for this) and that contour edges that shrink fully to zero with this snapping are omitted from the output. You may find the `WASD` keys helpful in debugging your snapping implementation.

3.2 Polyline Extraction (10pts)

The `extract_contours` function obtains the contour in the form of two arrays: contour point array `P` and edge endpoint index array `E`. Also, the pairs of indices in `E` are not in any particular order (e.g., counter-clockwise). Suppose you now want the contour expressed as a list of polylines, where each polyline is just a sequential list of points making up one of the contour's connected components. Devise and implement an algorithm to obtain this result as a `std::vector<Eigen::MatrixX3d>`.

Hint: you can use `E` to construct an adjacency list representation of the "contour graph" and then use a breadth-first search to traverse each connected component, building up its polyline. You can avoid needing to insert points at "both ends" of the polyline by starting your search from one of the two endpoints of an open contour (or an arbitrary point of a closed contour); then you can traverse the entire component in just a single direction. You can detect endpoints of an open contour by looking at the size of the adjacency lists.

4 What to Submit

Please submit a `.zip` or `.tgz` archive containing the following files:

- A `theory.pdf` file containing your **legible** solution to the theory exercises.
- The source files `main.cpp` and `extract_contours.cpp`, along with any other files you modified or added. Note, though, that additional modifications should not be necessary.
- A `screenshot.png` file showing a screenshot of the program displaying the scalar field F_{ii} using the triangle grid `meshes/mesh3.obj`.
- Optionally, a `readme.txt` describing problems you encountered when trying to solve the assignment and any comments you might have.