Digital 3D Geometry Processing
Exercise 5 – Delaunay Triangulation

February 22, 2024

Note

Hand in a .zip compressed file renamed to `Exercise`*n*`-GroupMemberNames.zip` where *n* is the number of the current exercise sheet. It should contain:

- **Only** the files you changed (headers and source). It is up to you to make sure that all files that you have changed are in the zip.
- A `readme.txt` file containing a description on how you solved each exercise (use the same numbers and titles) and the encountered problems.
- Other files that are required by your `readme.txt` file. For example, if you mention some screenshot images in `readme.txt`, these images need to be submitted too.
- Submit your solutions to Gradescope before the submission deadline.

Coding Exercise (6 pts)

The goal of this exercise is to implement the Delaunay Triangulation algorithm which maximizes the minimum angle of all the angles of the triangles in the triangulation. After correct implementation, the demo will help you to build a connection between the Voronoi diagram and the triangulation in an interactive way.

- By clicking the button `Create Initial Mesh` in the `Plugin-DGPExercises`, the demo starts with an initial triangle mesh containing two triangles and four vertices. The corresponding Voronoi cells are visualized as projections of cones on the base plane in different colors. The Voronoi edges are the projections of the cone intersections on the base plane (See Figure 1). In case the view is changed, you can restore the perspective by pressing the `Set 2D View` button.

- In the demo, you need to incrementally add points to the triangle mesh. The picking mode should be activated by clicking on the arrow icon in the OpenFlipper toolbox. Then you can left click with your mouse inside the initial mesh to add a point. If the newly added point is on a mesh edge, it will perform an edge split; otherwise it will do a face split. The Voronoi diagram will be updated simultaneously.

- First, implement the `is_delaunay(...)` function in the file `DelaunayTriangulation2D.cc`. Fill in the missing code which tests whether an edge is Delaunay or not. It will serve as a basic operator for the next function.

- Then, add your code to the `insert_point(...)` function in the file `DelaunayTriangulation2D.cc`. The first part of the function inserts the new vertex (either on an edge or inside a triangle) and is already implemented. You thus need to implement the algorithm that checks if the new triangles meet the Delaunay condition and flips edges if not. Recall that multiple edges might have to be flipped at each insertion. Overall, your task is to make sure the Delaunay property is valid over the whole mesh.

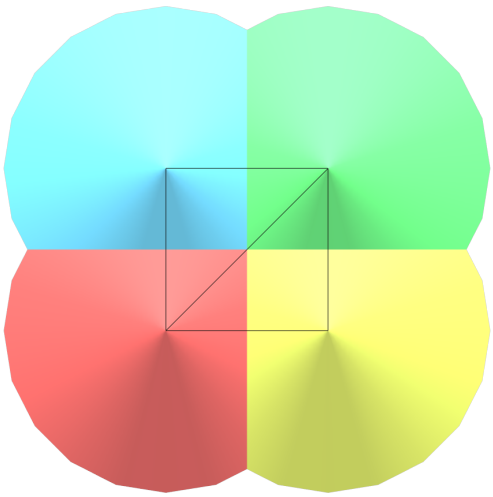

Figure 1: The initial mesh and Voronoi diagram.

Extra Credit Coding (+6 pts)

This quite open-ended extra credit assignment may be turned in anytime before Spring Break (March 8). It will build upon the framework for this assignment, but will require you to modify the interface, via `DGPExerciseToolbarBase.ui`, and to write some new functions.

What I'd like people to do is code up a button that performs *approximate* Lloyd iterations, which will converge to centroidal Voronoi diagrams. For a reminder of these terms, see the end of Lecture 4 on Delaunay triangulations.

The iterations will be approximate because we do not have an explicit representation of the Voronoi diagram (only a visualization via intersecting cones). Without this, it will be challenging to calculate exact centers-of-mass (centroids) for Voronoi cells. Thus, I would like people to use *Monte Carlo integration* to estimate the centroids. In this instance, this boils down to using a **large** random sample of points $\{\mathbf{q}_1, \ldots, \mathbf{q}_M\}$ from the square domain, and then averaging the positions of sampled points within each cell V_i to find an approximate centroid $\mathbf{c}_i$ for each cell.

$$\mathbf{c}_i = \frac{1}{|\{\mathbf{q}_j \in V_i\}|} \sum_{\mathbf{q}_j \in V_i} \mathbf{q}_j$$

Upon each button press:

- your seed points $\mathbf{p}_i$ (minus the corner points, which should remain fixed), should move to the approximate centroid $\mathbf{c}_i$ of their corresponding Voronoi cells V_i,
- the cones and thus the visualized Voronoi diagram should shift appropriately,
- and the Delaunay triangulation should update (this can be done via your incremental algorithm implemented above).