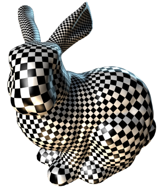
Optimal Transportation Theory and Computation

Euclidean Geometry



David Gu

Computer Science Department

Stony Brook University

Reference Book

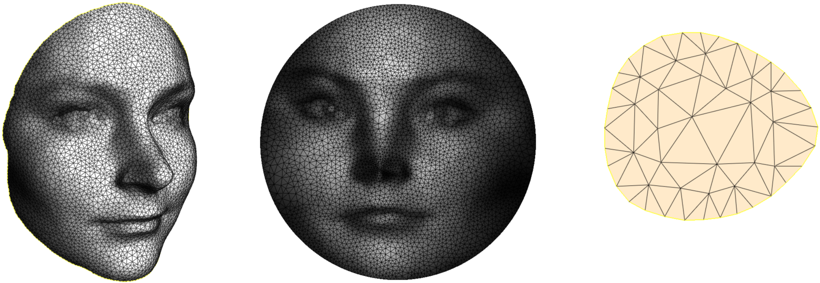
图: 教课书《最优传输理论和计算》.

Computational Geometric Algorithms

File Format

- The target measure (Ω^*, ν) is represented as a triangle mesh (obj format), each vertex has both (x, y, z) coordinates and (u, v) parameters. Each vertex v_i represents a sample $y_i = (u_i, v_i)$, (u_i, v_i) specify the planar position in Ω^*. The summation of the areas of all triangular faces adjacent to v_i is treated as ν_i, (after normalization).
- The source measure (Ω, μ) is represented as another triangle mesh (obj format), its boundary gives the boundary of Ω. For current version, μ is the uniform distribution.

File Format



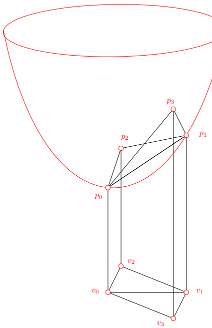
(a) Y and ν (b) planar positions $\{y_i\}$ (c) convex Ω

图: Input files.

1. The combinatorial data structure to represent the weighted Delaunay triangulation and the power diagram is either half-edge or Dart data structure;
2. The linear numerical solver is Eigen library;
3. The geometric predicate is based on adaptive (or exact) arithmetic method.
4. The weighted Delaunay is based on Lawson's edge flip algorithm.
5. The polygon clipping is based on Sutherland–Hodgman algorithm.
6. The optimization of Alexandrov energy is based on damping algorithm.

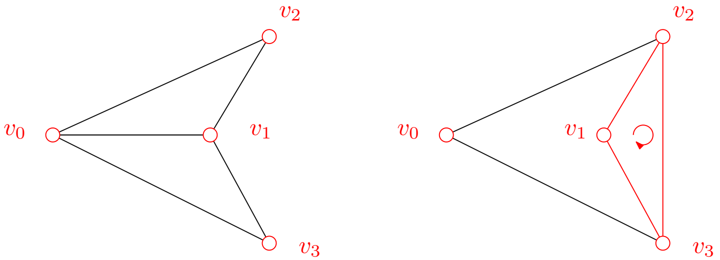
Edge Local Power Delaunay

Given an edge e in a planar triangulation $\mathcal{T}$, find the two neighboring faces, lift the four vertices to the convex hull φ, suppose vertex v_i is represented as $p_i(u_i, v_i, \varphi(u_i, v_i))$, compute the volume of the tetrahedron $[p_0, p_1, p_2, p_3]$. If the volume is positive, then e is locally powerd Delaunay, if the volume is negative, then e is non-locally-power-Delaunay.



Edge Flippable

Given an edge $e = [v_0, v_1]$ in a planar triangulation $\mathcal{T}$, if $[v_0, v_3, v_2]$ or $[v_1, v_2, v_3]$ is clockwise, then the edge is not flippable.

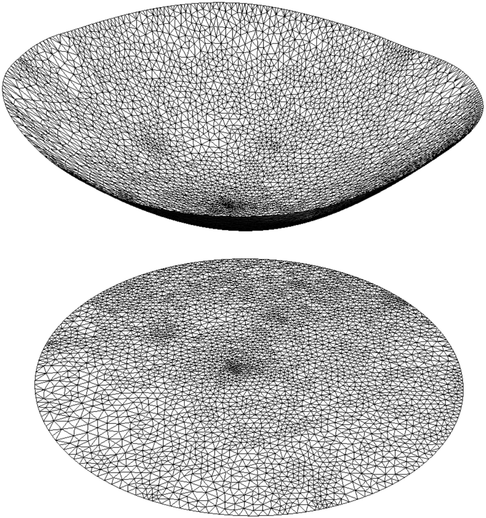


Lawson Edge Flip Algorithm

Input is a set of points S on the plane with the powers, the output is the power Delaunay triangulation.

1. Construct an arbitrary triangulation of the point set S;
2. Push all non-locally intrior edges of $\mathcal{T}$ on stack and mark them;
3. While the stack is non-empty do
   - 3.1 $e \leftarrow \text{pop}()$;
   - 3.2 unmark e;
   - 3.3 if e is locally power Delaunay then continue;
   - 3.4 if e can't be flipped then continue;
   - 3.5 flip edge e;
   - 3.6 push other four edges of the two triangles adjacent to e into the stack if unmarked;
4. If there is an edge e, which is not local power Delaunay, then there is some point p_i that is not on the convex hull of all p_k's.

Lawson Edge Flip for Convex Hull



Given a convex hull, which is the graph of a convex function φ, we compute its Legendre dual φ^*. Each point $p_i = (a_i, b_i, c_i)$ on the convex hull represents a plane π_i,

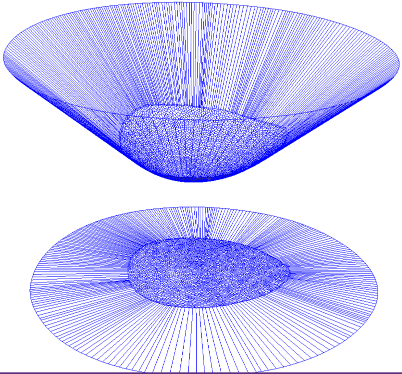
$$\pi(x, y) = a_i x + b_i y - c_i.$$

Each face $[p_i, p_j, p_k]$ is dual to a point (x, y, z) satisfying the linear equation group,

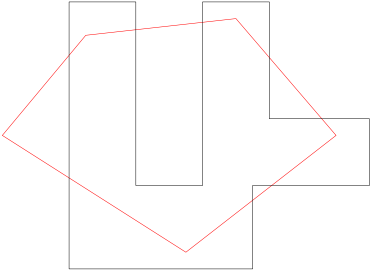
$$\begin{pmatrix} c_i \\ c_j \\ c_k \end{pmatrix} = \begin{pmatrix} a_i & b_i & -1 \\ a_j & b_j & -1 \\ a_k & b_k & -1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Upper Envelope-Brenier Potential

Given the convex hull $\{p_1, p_2, \cdots, p_k\}$, where $p_i(u_i, v_i, \varphi(u_i, v_i))$, add one more point as infinity point $(0, 0, -h)$, h is big enough to be above all other points. Each face f_α is dual to a point f_α^*; each vertex v_i is dual to a supporting plane v_i^*.

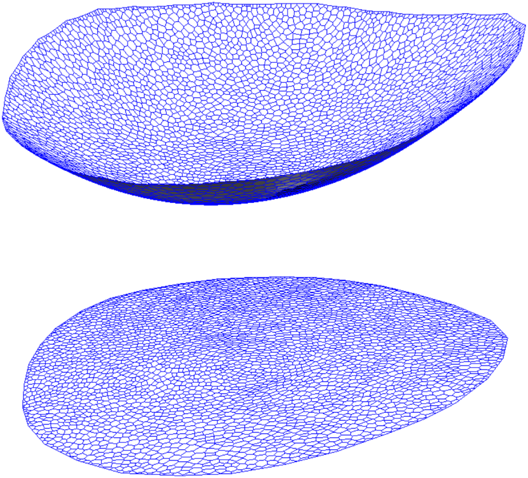


Given a subject polygon S and a convex clipping polygon C, we use C to clip S. Each time, we use one edge e of C to cut off a corner of S.



```
foreach Edge clipEdge in clipPolygon do
    List inputList ← outputList;
    outputList.clear();
    foreach Edge [p_{k−1}, p_k] in inputList do
        Point q ← ComputeIntersection(p_{k−1}, p_k, clipEdge);
        if p_k inside clipEdge then
            if p_{k−1} not inside clipEdge then
                outputList.add(q);
            end
            outputList.add(p_k);
        end
        else if p_{k−1} inside clipEdge then
            outputList.add(q)
        end
    end
```

Upper Envelope - Brenier Potential



Cell Clipping

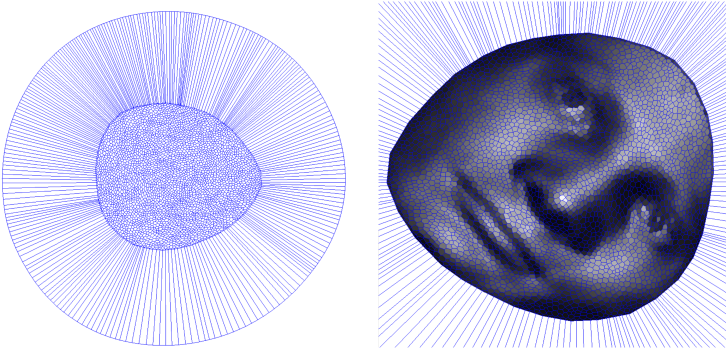


图: Boundary cell clipping.

Power Diagram Algorithm

1. Compute the convex hull using Lawson edge flipping, add the infinity vertex $(0, 0, -h)$; project the convex hull to power Delaunay triangulation $\mathcal{T}$;
2. Compute the upper envelope using Legendre dual algorithm and project to the power diagram $\mathcal{D}$;
3. Clip the power cells using Sutherland-Hodgman algorithm;

Damping Algorithm

1. Initialize the step length λ;
2. $\varphi \leftarrow \varphi + \lambda d$;
3. Compute the convex hull using Lawson edge flipping, add the infinity vertex $(0, 0, -h)$; project the convex hull to power Delaunay triangulation $\mathcal{T}$;
4. If the convex hull misses any vertex, then $\lambda \leftarrow \frac{1}{2}\lambda$, repeat step 2 and step 3;
5. Compute the upper envelope using Legendre dual algorithm, project to the power diagram $\mathcal{D}$;
6. Clip the power cells using Sutherland-Hodgman algorithm;
7. If any power cell is empty, then $\lambda \leftarrow \frac{1}{2}\lambda$, repeat step 5 and step 6;

Newton's Method

1. Initialize ϕ as $\phi(u,v) = \frac{1}{2}(u^2 + v^2)$;
2. Call the power diagram algorithm;
3. Compute the gradient ∇E, the target area minus the current power cell area;
4. Compute the Hessian matrix H, using the power diagram edge length;
5. Compute the update direction $Hd = \nabla E$;
6. Call the damping algorithm, set $\phi \leftarrow \phi + \lambda d$, such that ϕ is admissible;
7. Repeat step 2 through step 6, until the gradient is close to 0.

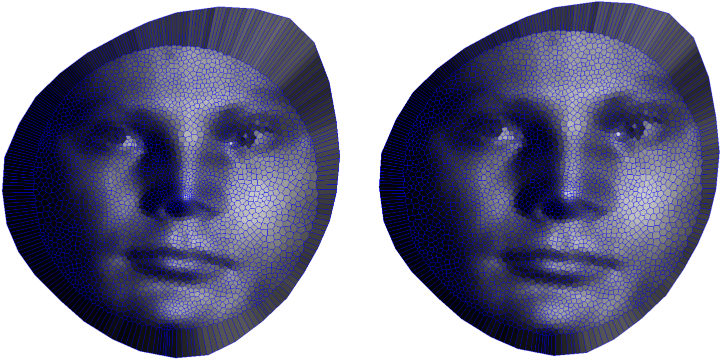


图: Optimal transportation map.

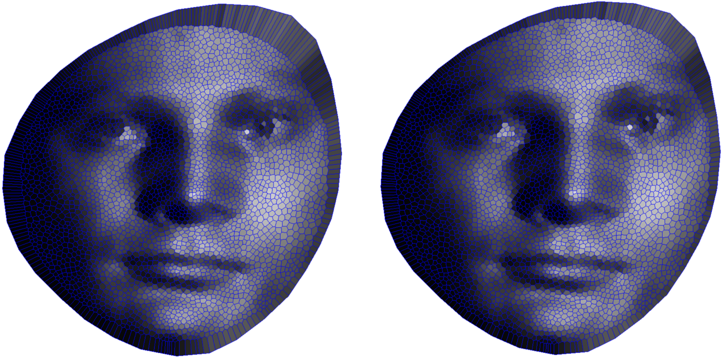


图: Optimal transportation map.

Optimal Transportation Map

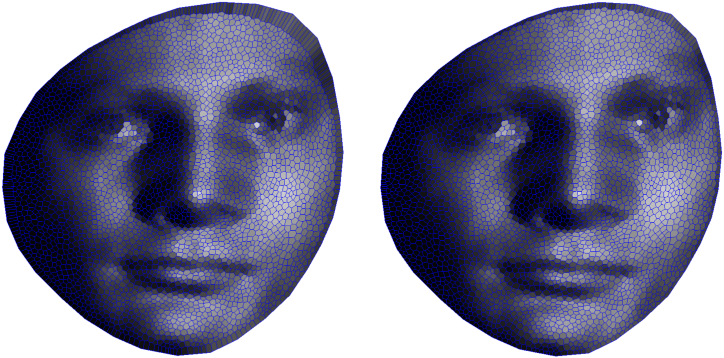


图: Optimal transportation map.

Optimal Transportation Map

图: Optimal transportation map.

Transportation Map

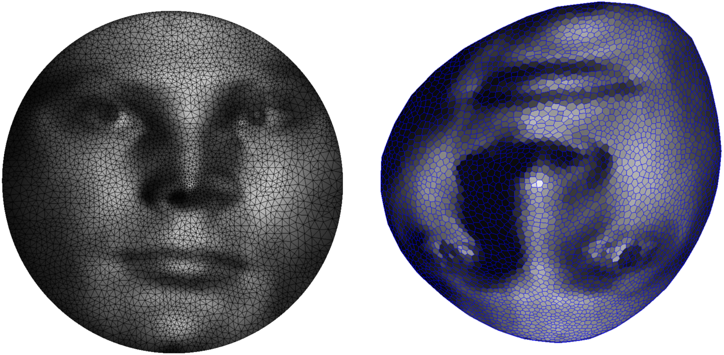


图: The worst transportation map.

Instruction

Dependencies

1. ‘DartLib‘ or ’MeshLib’, a general purpose mesh library based on Dart data structure.
2. ’Eigen’, numerical solver.
3. ‘freeglut‘, a free-software/open-source alternative to the OpenGL Utility Toolkit (GLUT) library.

Commands and Hot keys

- Command: -target target_mesh -source source_mesh
- '!': Newton's method
- 'm': Compute the mass center of power cells
- 'W': output the Legendre dual mesh and the optimal transportation map mesh
- 'L': Edit the lighting
- 'd': Show convex hull or upper envelope; power Delaunay or diagram
- 'g': Show 3D view or 2D view
- 'e': Show edges
- 'c': Show cell centers
- 'o': Take a snapshot

Compute the Power Delaunay and Power Diagram.

1. $CPDMesh :: \_Lawson\_edge\_swap$ Lawson edge swap algorithm to compute convex hull u_h^*, Power Delaunay triangulation;
2. $CPDMesh :: \_Legendre\_transform$ Legendre dual transformation compute upper envelope u_h, Power voronoi diagram;
3. $CPDMesh :: \_power\_cell\_clip$ Clip power cells, based on Sutherland-Hodgman algorithm;

Compute the Optimal Mass Transportation Map.

1. *COMTMesh* :: *_update_direction* compute the update direction, based on Newton's method;
2. *COMTMesh* :: *_calculate_gradient* calculate the gradient of the Alexandrov energy;
3. *COMTMesh* :: *_calculate_hessian* calculate the Hessian matrix of the Alexandrov energy;
4. *COMTMesh* :: *_edge_weight* calculate the edge weight

Compute the Optimal Mass Transportation Map.

1. Implement Lawson's edge flipping algorithm to compute weighted Delaunay triangulation, $CPDMesh :: \_Lawson\_edge\_swap$;
2. Implement Sutherland-Hodgman algorithm for convex polygon clipping, $Polygon2D :: Sutherland\_Hodgman$;
3. Implement Computing the Wasserstein distance.

Directory Structure

- 3rdparty/DartLib or 3rdparty/MeshLib, header files for mesh;
- MeshLib/algorithms/OMT, the header files for Power Diagram Mesh and Optimal Mass Transportation Map Mesh;
- OT/src, the source files for optimal transportation map;
- CMakeLists.txt, CMake configuration file;

Before you start, read README.md carefully, then go three the following procedures, step by step.

1. Install [CMake](https://cmake.org/download/).
2. Download the source code of the C++ framework.
3. Configure and generate the project for Visual Studio.
4. Open the .sln using Visual Studio, and complie the solution.
5. Finish your code in your IDE.
6. Run the executable program.

Configure and generate the project

1. open a command window
2. cd ot-homework3_skeleton
3. mkdir build
4. cd build
5. cmake ..
6. open OTHomework.sln inside the build directory.

For more information, please contact gu@cs.stonybrook.edu

Thank You!