Computer Science 130B
Spring 2017
Programming Assignment #3

Due: **11:59pm**, Sunday, May 28th

In this programming assignment, we are going to consider a problem that arises in medical image visualization, or more specifically, that of visualizing the surface structure in between parallel slices of anatomical data. For example, computer tomography (CT) gathers anatomical data as a collection of parallel slices. The Visible Human Project of the National Library of Medicine collects transverse CT, MRI, and cryosection images of representative male and female cadavers at one millimeter intervals. Some sample images are shown in Fig. 1.

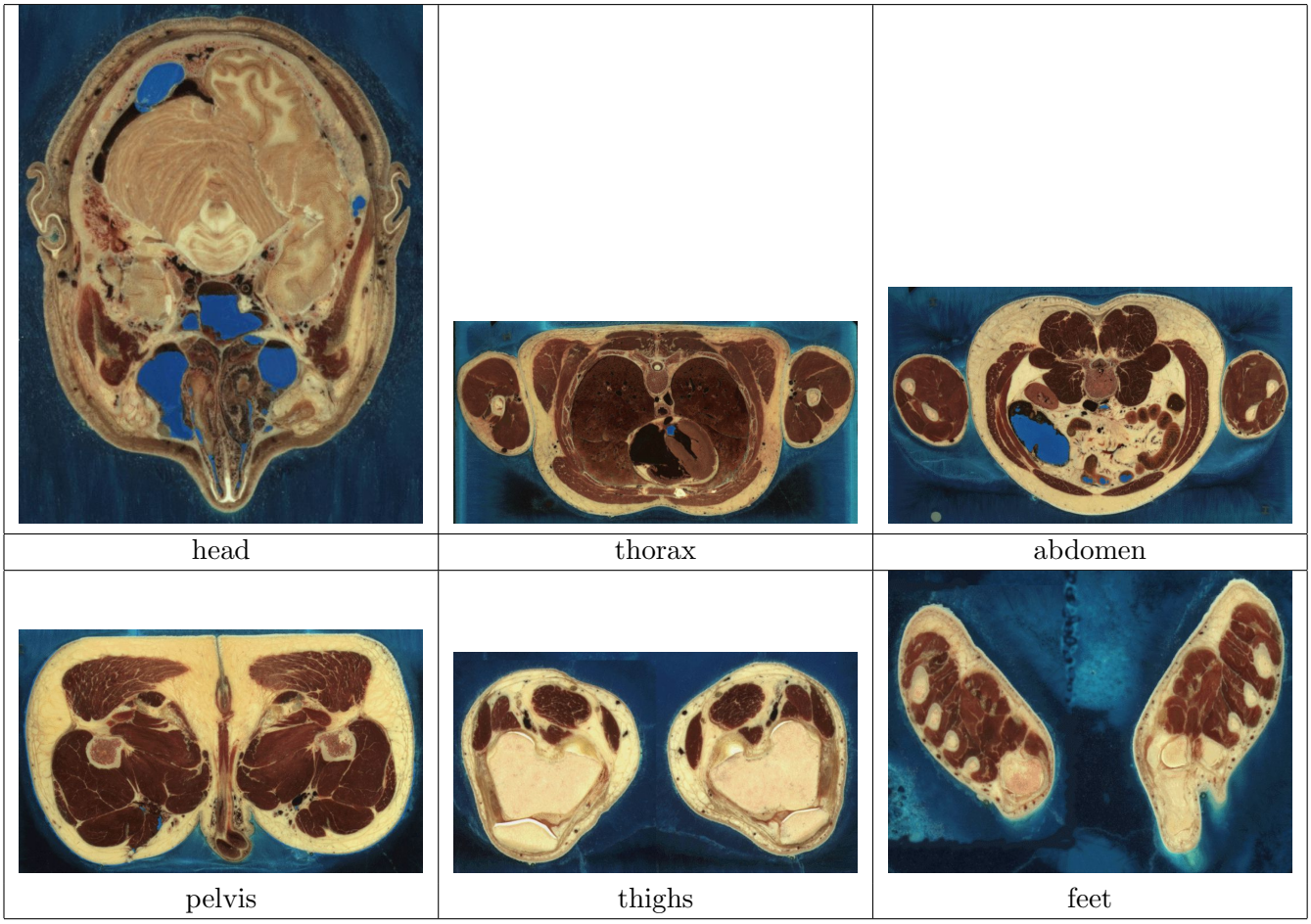


Figure 1: Sample cryosection images from the Visible Human Project

If a transverse scan (from head to toe) is performed on the human body, the arm and leg bone structures will be imaged in many parallel slices. It often of interest to generate 3D description of these anatomical structures from slices. Suppose that each slice is recorded in a digital format as a 2D image. Furthermore, suppose that certain image processing algorithm has been applied to locate the boundary, say, between a bone and the surrounding tissue. Then to obtain a 3D surface description of the bone in between two adjacent slices, bone-tissue boundaries in adjacent slices are interpolated to generate a surface description (e.g., see Figure 2).

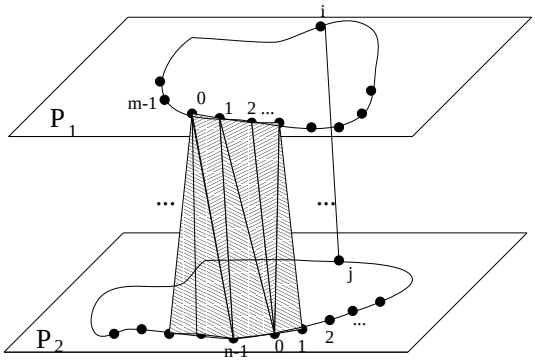


Figure 2: Triangulation in between two parallel planes.

Mathematically, the problem can be stated as follows: You are given two parallel planes, P_1 and P_2, with P_1 on top of P_2. On each plane, you have an *ordered* sequence of points which, when connected together, form a closed, simple (non-intersecting) contour. The contours may have different number of points though and there is no requirement that the points should "line up" (or on top of one another) in any manner. The number of points on the contour on P_1 is m and that on P_2 is n, where $m \geq 1$ and $n \geq 1$. Then the set of points on P_1 are:

$$(x_0^1, y_0^1), (x_1^1, y_1^1), \cdots, (x_{m-1}^1, y_{m-1}^1) \ ,$$

and those on P_2 are

$$(x_0^2, y_0^2), (x_1^2, y_1^2), \cdots, (x_{n-1}^2, y_{n-1}^2) \ .$$

Assume $z^1 = 1$ and $z^2 = 0$ for convenience. Your task is to design a triangulation algorithm **in between** the boundary points on these two planes (again, see Figure 2). The goal is to form a closed surface in between the two contours satisfying the following constraints:

- Only triangles are used as building elements.
- These triangles use points on P_1 and P_2 as their vertices.
- The triangulation result should be such that the collection of triangles form a closed (no holes) surface in between the two contours.

- The total area of these triangles should be as small as possible. [1]

For this homework, you can assume that only two slices are given and only one contour per slice. The input data will be of the following format:

$$
\begin{array}{ll}
m \;\; n & //number\ of\ points\ on\ the\ first\ and\ second\ contours \\
x_0^1 \;\; y_0^1 \;\; 1 & //zeroth\ point\ on\ the\ first\ contour \\
x_1^1 \;\; y_1^1 \;\; 1 & //first\ point\ on\ the\ first\ contour \\
\cdots & \\
x_{m-1}^1 \;\; y_{m-1}^1 \;\; 1 & //last\ point\ on\ the\ first\ contour \\
x_0^2 \;\; y_0^2 \;\; 0 & //zeroth\ point\ on\ the\ second\ contour \\
x_1^2 \;\; y_1^2 \;\; 0 & //first\ point\ on\ the\ second\ contour \\
\cdots & \\
x_{n-1}^2 \;\; y_{n-1}^2 \;\; 0 & //last\ point\ on\ the\ second\ contour
\end{array}
$$

Your output should be in the following format:

$$
\begin{array}{ll}
i_{11} \; i_{12} \; i_{13} & //first\ triangle \\
i_{21} \; i_{22} \; i_{23} & //second\ triangle \\
\cdots & \\
i_{t1} \; i_{t2} \; i_{t3} & //last\ triangle
\end{array}
$$

where i_{k1}, i_{k2}, i_{k3} are the index numbers of the three vertices of the k-th triangle. If a vertex is drawn from the first contour, then the index number is what it is shown above plus 1. If a vertex is drawn from the second contour, then the index number is the index number shown above plus $m+1$. I.e., the zeroth vertex on the first contour will be 1, the first vertex on the first contour will be 2, and the last vertex on the first contour will be m. The zeroth vertex on the second contour will be $m+1$, the first vertex on the second contour will be $m+2$, and the last vertex on the second contour will be $m+n$. That is, *the output will be in 1-index format instead of 0-index format.* The reason is that you can use Matlab for visualization (try `trimesh` or `trisurf` in Matlab).

For example, if $m = 4$ and $n = 4$, and both contours are made of the following points:

$$(0,0), (0,1), (1,0), (1,1)$$

Then, the input file will look like:

$$
\begin{array}{c}
4 \;\; 4 \\
0 \;\; 0 \;\; 1 \\
0 \;\; 1 \;\; 1 \\
1 \;\; 1 \;\; 1 \\
1 \;\; 0 \;\; 1 \\
0 \;\; 0 \;\; 0 \\
0 \;\; 1 \;\; 0 \\
1 \;\; 1 \;\; 0 \\
1 \;\; 0 \;\; 0
\end{array}
$$

[1] The area of a triangle, expressed in terms of its three edge lengths a, b, and c is:

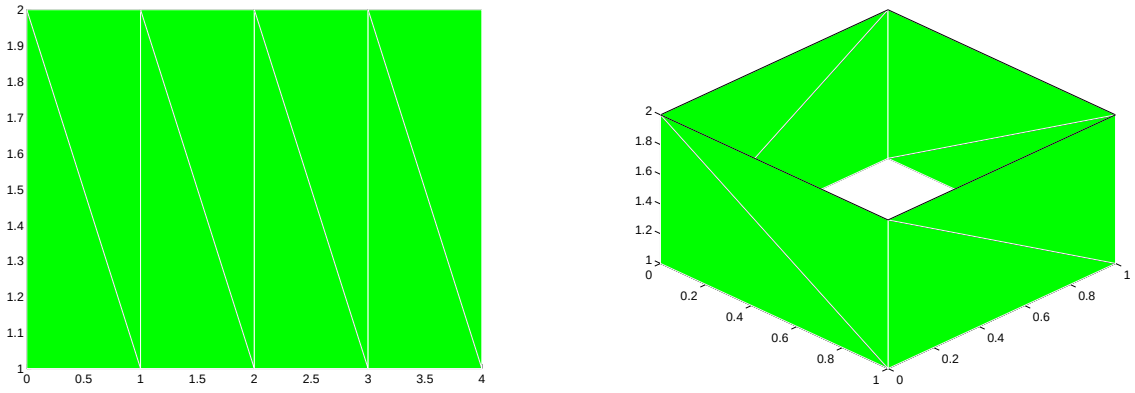
$$\sqrt{s(s-a)(s-b)(s-c)}\,,$$

where $s = \frac{a+b+c}{2}$.

and a valid algorithm might generate results as shown in Figure 3 with the following output:

1 2 6
6 5 1
2 3 7
7 6 2
3 4 8
8 7 3
4 1 5
5 8 4

A slightly more complicated example, where the contours on both P_1 and P_2 are unit circles, but the contour on P_1 is made of 15 points while that on P_2 is made of twice as many points, with contour points distributed evenly around the circle in both cases, is shown in Figure 4.

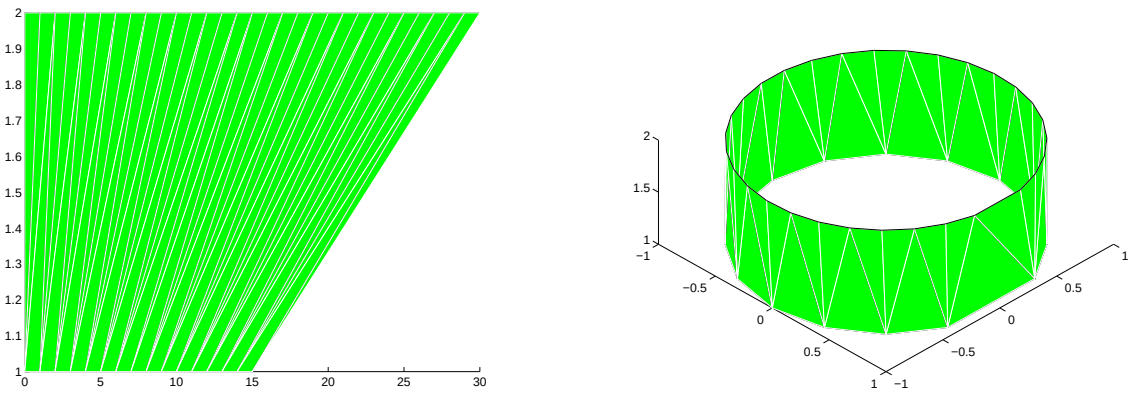


(a) (b)

Figure 3: Triangulation in between two parallel planes, and the contours on both planes are square: (a) Triangles displayed in between two index lists and (b) triangles displayed in between actual 3D points.

Solve the triangulation problem using dynamic programming. In addition to the program, you should turn in written answers (README.pdf) with your program to the following questions:

1. How does the principal of optimality apply?
2. What is the recurrence relation used in solving this problem?



(a) (b)

Figure 4: Triangulation in between two parallel planes, and the contours on both planes are circle: (a) Triangles displayed in between two index lists and (b) triangles displayed in between actual 3D points.

3. How does the table of partial solutions look like? How to construct such a table?