Computing Science CMPT 464/764 **Spring 2016**
Instructor: Richard (Hao) Zhang **Simon Fraser University**

Assignment #3
Written part due Thursday, March 31, in class
Programming part due Thursday, March 31, 11:45 p.m. via electronic submission

---

On the cover page of your homework, write and sign the following statement: "I have read and understood the policy concerning collaboration on homework and lab assignments". Without such a signed statement, your work will not be marked. A sample cover page can be found on the course webpage.

Problem 1 (4 marks): Tutte embedding and convexity

Is it true that in a 2D Tutte embedding, the edges connecting successive vertices along the one-ring of any interior vertex is a convex polygon? If so, give a proof. Otherwise, provide a counterexample.

Problem 2 (16 marks): Clustering and the *K*-means algorithm

The clustering problem can be simply stated as one that attempts to group a given set of data points into some number of clusters according to some distance/proximity measure between data points. It is hard to define precisely what a cluster is, but various measures for the "goodness" of a clustering result have been proposed. For example, the so-called **Fisher criterion** has been suggested for two clusters

$$\gamma(A,B) = \frac{(\mu_A - \mu_B)^2}{\sigma_A^2 + \sigma_B^2}$$

where μ and σ denote the mean and variance of the clusters. A great deal of research has been conducted to solve the clustering problem in a variety of fields, including artificial intelligence (machine learning and computer vision), biology, psychology, and statistics. In digital geometry processing, clustering algorithm would be quite useful in 3D mesh segmentation and shape recognition.

One of the most well known algorithms is the so-called *K*-means clustering algorithm. Given a set of N data points x_i, $i = 1, \ldots, N$, the *K*-means algorithm partitions the data set into K clusters, in an attempt to minimize the total squared distance from data points to their respective cluster centroids. This objective function J is given by,

$$J = \sum_{j=1}^{K} \sum_{x \in S_j} \left\| x - \mu_j \right\|^2 ,$$

where μ_j is the center of cluster S_j. Such an optimization problem has been shown to be NP-hard. The *K*-means algorithm can thus result in any one of the numerous local minima. An outline of the algorithm is given below:

1. Find an initial clustering of the data set into K groups, e.g., a random assignment to K groups, and compute the group centroids (centers).

2. Each point x_i is reassigned to a group S_j whose center is closest to x_i.
3. The group centers are recomputed and then step 2 is repeated. The algorithm terminates when no data point needs to be re-assigned.

You are to work in MATLAB with *K*-means algorithms applied to 2D points.

1. **[7 marks]** Write a MATLAB function kmeans(x, k, initg) that performs *k*-means clustering on the set of 2D points given by x (an $n \times 2$ array). The number of clusters k is given and the array initg supplies the initial clustering. If this argument is ignored, then a random initial clustering is assumed. The function should also produce a scatter point plot showing points in different clusters in different colors. Check out the MATLAB function scatter.
2. **[3 marks]** Is the *k*-means algorithm guaranteed to converge? Why or why not?
3. **[3 marks]** Provide an example in which the best result produced by *k*-means differs definitively from our intuitive notion of a good clustering. How does the *k*-means algorithm perform according to the Fisher criterion (stick with $k = 2$ for this)?
4. **[3 marks]** Suggest a good way to choose the number of clusters k. Of course, random selection of k or a solution that relies on an examination of the data set by a human does not fall into this category. Your method will most likely be some heuristic that may work most of the time, i.e., it is able to choose the right number of clusters. But try to provide an example for which your heuristic fails.

Finally, think about how to choose a good initial clustering (you need not submit an answer to this). Obviously, you would want this procedure to be fairly efficient.

You are encouraged not to consult outside help for this. But if you do feel the need to do so, make sure you provide sufficient references.

What to submit: Your MATLAB function kmeans, as an M-file kmeans.m submitted electronically, and required plots and write-ups on paper. In the assignment directory, you can find some test data under kmeans_data.

Problem 3 (12 marks) Spectral mesh compression using "Fourier descriptors"

A triangle mesh can be specified by a graph $G = (V, E)$, characterizing the connectivity among its vertices, and a *coordinate signal* X: $V \rightarrow \mathbf{R}^3$, where $X(v) = (v_x, v_y, v_z)$ provides the 3D coordinates of vertex v in the mesh. For convenience, we shall use $X \in \mathbf{R}^{n \times 3}$, to denote the 3D coordinates of the n vertices in the mesh. The "Fourier transform" of this mesh signal can be defined as follows:

$$F(X) = X' = Q^T X,$$

where the matrix Q is such that its columns are the eigenvectors $\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_n$ of a suitably defined *mesh Laplacian* operator. The transform coefficients are stored in X' and if we let its rows be $X'_1, X'_2, \ldots, X'_n$, then they form the "Fourier transform" of the mesh signal X. Correspondingly, the inverse Fourier transform is given by

$$X = QX' = \mathbf{e}_1 X'_1 + \mathbf{e}_2 X'_2 + \ldots + \mathbf{e}_n X'_n,$$

where we write the mesh coordinate signal X as a linear combination of the eigenvectors $\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_n$. For our purpose, we use the Laplacian operator $K \in \mathbf{R}^{n \times n}$ defined as follows: K_{ii} = the degree of vertex i and $K_{ij} = -1$ if $(i, j) \in E$, i.e., (i, j) is an edge in the mesh, and $K_{ij} = 0$ otherwise. Clearly, K is a symmetric matrix and from linear algebra, we know that the eigenvalues of K are all real and its eigenvectors can be made orthonormal (unit length and mutually orthogonal).

Assume that the eigenvectors above are ordered in increasing order of eigenvalues of K, one can roughly think of the eigenvalues as frequencies. Thus, by truncating the linear sum given above, e.g., using only the leading $m < n$ eigenvectors and their associated transform coefficients X'_i's, which are called the "*Fourier descriptors*" in our context, one can obtain a *spectral compression* of the original mesh. The following shows a snake-shaped mesh with 530 vertices (far left) along with spectral compressions of the mesh with 5, 10, 20, 30, 50 Fourier descriptors (from left to right):

Your goal in this problem is to compute such spectral compressions of a given (small) triangle mesh. Since you need to compute the eigenvalues and eigenvectors of a matrix, you need not handle meshes with more than 1,000 vertices (note that computing eigenvalues and eigenvectors of a matrix, even sparse, is computationally expensive).

What to submit electronically: Submit a single MATLAB file called fds.m. The function fds() should take in two parameters, the first being a string specifying the name of an input SMF file and the second parameter is a row vectors specifying a list of numbers. For example, by executing

```
fds('small_horse.smf', [5 10 20]);
```

within MATLAB, the program will display four meshes in four figure windows. In the first figure window, the original mesh (small_horse.smf) is shown. In the next three figure windows, meshes reconstructed using 5, 10, and 20 leading Fourier descriptors are shown. A sample MATLAB program, plot_smf.m, which displays a mesh given as an SMF is provided in the assignment directory, along with some sample SMF files.

Total marks: 32