Programming Project 05

This assignment is worth 45 points (4.5% of the course grade) and must be **completed and turned in before 11:59 PM on Monday, October 14, 2019.**

Assignment Overview

1. Functions
2. File input and output
3. try-except

Computational facial recognition is growing. Starting with the iPhone X it has even available on smartphones. In this project we examine some of the mathematics behind it. Some of the initial research in facial recognition was by Dr. Anil Jain here at MSU.

Use of advanced data structures such as lists, sets, and dictionaries are prohibited.

Assignment Background

In the field of Computer Graphics, creating and handling shapes is fundamental. Any shape can be represented by what is called a polygon mesh—a collection of **vertices**, **edges** and **faces** that define the shape. A triangle mesh is a common type of polygon mesh where each face is a **triangle**. The vertices are the points of the shape in 3d space. Each vertex has x,y and z values representing its location in Euclidean space. The vertices are connected by edges that form the triangles. Figure 1 shows an example of triangle mesh representing a dolphin.

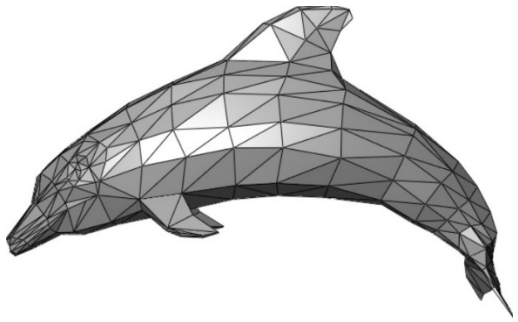

Figure 1 Triangle mesh (Wikipedia)

Figure 2 shows the vertices, edges and faces of a cube represented by a triangle mesh. The vertices are the points of the mesh, the edges are the lines connecting two vertices, and the faces are the triangles that are parts of the surface connecting 3 vertices.

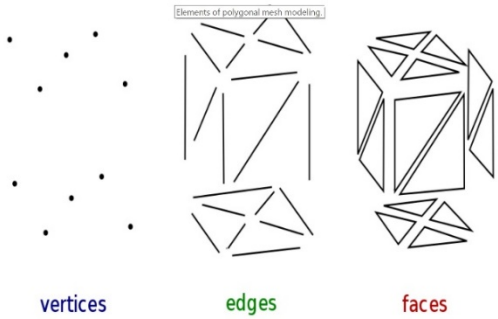

Figure 2 vertices, edges and faces of a cube (Wikipedia)

Each face of the triangle mesh is a triangle, that contains 3 vertices and 3 edges. Any two faces of the mesh are connected by only one edge. There are many of geometrical properties we can compute for mesh triangles. The face normal is the perpendicular vector of the plane that this triangle is part of. Figure 3 shows the meaning of face normal. In the figure, the triangle has 3 vertices a, b and c.

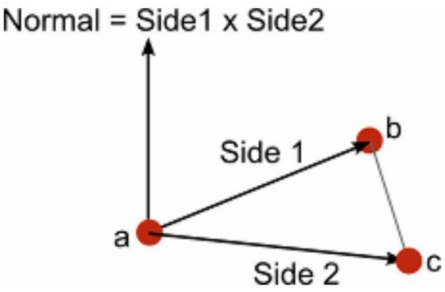

Figure 3 Face normal

To compute the normal of this face (triangle), we need to specify the coordinates of the two sides, side 1 that connects the vertices a and b, and side 2 that connects the vertices a and c. For example, if point a has coordinates $(a1, a2, a3)$ and point b has coordinates $(b1, b2, b3)$. The coordinates of the vector $\overrightarrow{ab}$ representing Side 1 are $(b1-a1, b2-a2, b3-a3)$.

The normal is the cross product of the two sides. The cross product is a mathematical operation that finds the perpendicular vector of two vectors (note that the × symbol stands for *cross product*, not multiplication). The cross product is defined as follows:

Let's assume that we have two vector **v** and **w**: $\begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix}$ and $\begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix}$. The cross product between **v** and **w** is:

$$\mathbf{v} \times \mathbf{w} = \begin{pmatrix} v_2 w_3 - v_3 w_2 \\ v_3 w_1 - v_1 w_3 \\ v_1 w_2 - v_2 w_1 \end{pmatrix}$$

Another property that we can compute is the area of a face (triangle). Figure 4 shows the meaning of face area. In the figure, the triangle has 3 vertices a, b and c. We need to specify 3 sides, side 1 that connects the vertices a and b, side 2 that connects the vertices a and c, and side 3 that connects the vertices b and c. Then, we need to calculate the distances (ab, ac, and bc) between the vertices. The distance between two points in a three dimensional - 3D - coordinate system can be calculated as:

$$d = \sqrt{(w_1 - v_1)^2 + (w_2 - v_2)^2 + (w_3 - v_3)^2}$$

After that we calculate the area using Heron's formula :

$$Area = \sqrt{p(p-ab)(p-ac)(p-bc)}$$

$$p = \frac{ab + ac + bc}{2}$$

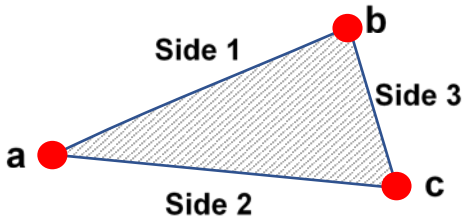

Figure 4 Face area

There are many ways to store the mesh data, the coordinates of all vertices and the vertices of each face should be all listed in the file. In this project, we will use `.off` data files with the following format:

```
OFF
  500 1000 0
     -12.621730      17.675560     -21.116800
     -14.944020      10.601960     -20.006480
     -19.904680      20.878550     -20.212760
     -11.535790       7.209767     -19.530910
      -8.708838      14.810500     -21.003320
…
…
…
     -18.477250      29.710110      -5.084175
     -16.631550      29.141630      -0.938385
 3    0    4    1
```

```
 3     1     2     0
 3     1     4     3
 3     0     8     4
…
…
 3   496   493    88
```

The first line in the file is the word "off" which is the file format. The second line contains three fields where each is 5 characters wide (right-justified, digits only): the first field is the number of vertices, the second field is the number of faces, and the third field is number of edges. We only need the first two numbers. In this example, we have 500 vertices and 1000 faces. The following lines contains 3 fields where each is 15 characters wide (right-justified): these are the coordinates of all vertices (3 floating numbers represent x,y and z of each vertex). For example, the first vertex has the coordinates x= -12.621730, y= 17.675560, z= -21.116800. Each vertex should have an index starting from 0. So, the first line is the coordinates of vertex 0 , then next line has is the coordinates of vertex 1 and so on.

Similarly, each line representing a face contains the number 3 then the indices of the vertices of that face. So, the first face has the vertices 0, 4 and 1. The second face has the vertices 1, 2 and 0 and so on. Each line in the face data contains 4 fields with length 2, 5, 5, 5 characters wide (right-justified), respectively.

Project Specification

In this project, we open the `off` file for reading the mesh data. Then we compute some geometrical features of the mesh. The field width specification defined above allows you to use slicing to extract values from each line.

Your code should include at least the following functions:

1. **`open_file() --------→ fp :`**
   This function takes no parameters, and then uses the try-except format to prompt for a filename, open the data file and return the file pointer if successful. Your function should be able to catch the error and print the error message if it fails to open; and then reprompt. It will reprompt until successful.
2. **`read_face_data(fp, index) ---→ int, int, int:`**
   This function takes as an input:
   a. `fp:` the file pointer for the mesh file,
   b. `index:` this variable stores the index of the face in consideration of type `int`.

   This function should return the indices of the 3 vertices that makes the face which are of type `int`. Note that sometimes you will be in the middle of the file and you want to access lines on the top. In this case the method `fp.seek(0)` could be used (see note below).

You should skip the first word of the file using `readline()` function. The second line has the number of vertices and the number of faces of this mesh. The faces indices starts from 0. In the previous example, we have 500 vertices and 1000 faces. That mean the next 500 lines contain the vertices coordinates, and the following 1000 lines are for all faces. The field width specification defined in the background allows you to use slicing to extract values from each line. Hint: loop until you find a line starting with the number 3 that indicates the start of the face data; then start counting indices.

3. **`read_vertex_data(fp, index) --->  float, float, float:`**

This function takes as an input:

a. `fp:` the file pointer for the mesh file,

b. `index:` this variable stores the index of the vertex in consideration of type `int`.

This function should return the coordinates of the vertex which are of type `float`. The field width specification defined in the background allows you to use slicing to extract values from each line. The indices of the vertices starts from 0. Hint: use multiple `fp.readline()` to skip the first two lines and then `for i in range()` to skip to your desired index.

4. **`compute_cross(v1,v2,v3,w1,w2,w3) ---->  float, float, float:`**

This function computes the cross product between two sides (2 vectors) of a face, using the formula above. It takes as inputs:

a. `v1,v2,v3:` coordinates of the first side each as `float`,

b. `w1,w2,w3:` coordinates of the second side each as `float`

It should return the 3 coordinates of cross-product as `floats` rounded to 5 digits.

5. **`compute_distance(x1,y1,z1,x2,y2,z2) ---->  float:`**

This function computes the Euclidian distance between two points. It takes as inputs:

a. `x1,y1,z1:` coordinates of the first point each as `float`,

b. `x2,y2,z2:` coordinates of the second point each as `float`

It should return the distance as a `float` rounded to 2 digits.

The Euclidean distance D between two points `A` and `B` with coordinates `(x1,y1,z1)` and `(x2,y2,z2)`, respectively is computed as follows:

$$\boldsymbol{D = \sqrt{(x1 - x2)^2 + (y1 - y2)^2 + (z1 - z2)^2}}$$

6. **`compute_face_normal(fp,face_index)--->  float, float, float:`**

This function computes the normal vector of a face. Figure 3 explains how the face normal is computed. The `compute_cross()` function is useful here.

This function takes as inputs:

a. `fp:` the file pointer for the mesh file,

b. `face_index:` the index of the face in consideration as `int`.

It should return the 3 coordinates of the face normal rounded to 5 digits.

You should first find the vertices of the face from the mesh file using the `face_index` and the `read_face_data` function. That function returns 3 vertices. You can capture the three vertices returned from the functoion like this:

```
first, second, third = read_face_data(fp, face_index)
```

You will find the normal at the first vertex. For each vertex you need their three coordinates; `read_vertex_data` can be used. Note that to calculate the normal vector, you need to calculate the coordinates of the sides first.

The vector components (vector coordinates) through two points (initial and terminal points) in three-dimensional space is calculated as the difference between the coordinates of each point. For example, if A and B are the initial and terminal points with coordinates `(x1,y1,z1)` and `(x2,y2,z2)` respectively, the coordinates of vector $\overrightarrow{AB}$ are:

```
(x2-x1,y2-y1,z2-z1)
```

Find the vectors of the two sides and then take their cross product using your `compute_cross` function.

7. **`compute_face_area(fp, face_index)`----→ `float`:**

This function calculate the face area. Figure 4 explains how the face normal is computed. It takes as an input:

a. `fp`: the file pointer for the mesh file,
b. `face_index`: the index of the face in consideration as `int`.

It should return the area of the face which is of type `float` rounded to 2 digits. The function `compute_distance()` is useful in this function.

8. **`is_connected_faces(fp, f1_ind, f2_ind)` -------→ Boolean:**

This function checks if two faces are connected. Two faces are connected if they share 2 vertices. So you can expect that each face might be connected with three other faces, one for each side. The function takes the file pointer and the faces' indexes and checks if they have two common vertices. The function returns `True` if the two faces are connected, or `False` otherwise.
This function takes as an input:

a. `fp`: the file pointer for the mesh file,
b. `f1_ind`: the index of the first face as `int`,
c. `f2_ind`: the index of the second face as `int`.

9. `check_valid(fp,index,shape)`-----→ Boolean:

This function checks if the face or vertex index is valid. It takes as input:

a. `fp`: the file pointer for the mesh file,
b. `index`: the index of the shape in question as `str`,
c. `shape`: the type of shape as a `string` which is either `'face'` or `'vertex'`.

The function returns `True` if valid, or `False` otherwise. An index is valid if it is an integer greater than or equal to zero and less than the number of vertices if shape is `'vertex'` or less than the number of faces if shape is `'face'`. If shape is neither `'face'` nor `'vertex'`, return False.

10. `main()`

In the main function, use the nine (9) functions above to complete the mission!
When the program starts, a welcome message is displayed. Then call the `open_file()` function to open the mesh file. After that, display the menu and ask to choose one option from it:

1- display the information of the first 5 faces
2- compute face normal
3- compute face area
4- check two faces connectivity
5- open another file.
6- exit

If the choice is not valid, display an error message and reprompt until a valid choice is entered. A choice is valid if it is an integer and between 1 and 7.

- Choice 1: collect and display the information of first 5 faces.
  - Print the header first using the string format `"{:^7s}{:^15s}".format('face', 'vertices')`
  - Next 5 lines will contain 4 fields (index,vertex1,vertex2, and vertex3) where each is 5 characters wide (right-justified).
- Choice 2: compute the face normal. Prompt for the face index. Check validity of the index and display an error message if the index is not valid. Keep reprompting until the index is valid. Once valid, compute the face normal and display the results with the string formatting where the index is and each vertex is 9 characters wide, right-justified, and 5 digits of precision. The coordinates should be displayed with only 5 digits after the decimal point.
- Choice 3: compute the face area. Prompt for the face index. Check validity of the index and display an error message if the index is not valid. Keep reprompting until the index is valid. Once valid, compute the face area and display the results with the correct string formatting. The area should be displayed with only 2 digits after the decimal point.
- Choice 4: check if two faces are connected. Prompt for face index twice. Each time, check validity of the index and display an error message if the index is not valid. Keep reprompting until the index is valid. Then you should display the results using strings from the strings.txt file.
- Choice 5: Prompt to open a file and proceed as described above.
- Choice 6: exit the program

Assignment Notes

1. Coding Standard 1-9 will be enforced.
2. Use of `fp.seek(0)` is inefficient so it should rarely (as in never) be used. In fact, we will prohibit its use in future projects. Why? Reading from files on disk is slow compared to getting information from memory (that is thousands to millions of times slower). The proper way to do it is to read information from a file ONCE into a data structure in memory and thereafter always refer to the data structure. However, we haven't learned the appropriate data structures yet so we are using `fp.seek(0)`.

Assignment Deliverable

The deliverable for this assignment is the following file:

`proj05.py` – the source code for your Python program

Be sure to use the specified file name and to submit it for grading via the **Mimir** before the project deadline.

Test Cases

Function Test `check_valid`:

```
Filename: kitten.off

##########face#########
```

index	output
5	TRUE
999	TRUE
0	TRUE
1500	FALSE
1000	FALSE
-2	FALSE
1.5	FALSE
1e4	FALSE

```
##########vertex#########
```

index	output
10	TRUE
0	TRUE
499	TRUE
800	FALSE
500	FALSE
-5	FALSE
2.3	FALSE
2e5	FALSE

Function Test `read_face_data`:

```
Filename: cone.off
face 0 info:  2 3 0
face 20 info:  12 13 0
face 39 info:  2 21 1
```

Function Test `read_vertex_data`:

```
Filename: cone.off
student vertex 0 info:    0.000000  1.000000   0.000000
Student vertex 10 info:  -0.404508  0.000000   0.293893
Student vertex 21 info:   0.475528  0.000000  -0.154508
```

Function Test `compute_cross`:

```
v=[0.000000,1.000000,0.000000]
w=[-0.404508,0.000000,0.293893]
output: 0.29389, -0.0, 0.40451

v=[-0.404508,0.000000,0.293893]
w=[0.000000,1.000000,0.000000]
output: -0.29389, 0.0, -0.40451

v=[3.912892,-2.865060,0.113480]
w=[-2.322290,-7.073600,1.110320]
output: -2.37842, -4.6081, -34.33173
```

Function Test `compute_distance`:

```
v=[0.000000,1.000000,0.000000]
w=[-0.404508,0.000000,0.293893]
output:  1.12

v=[-0.404508,0.000000,0.293893]
w=[0.000000,1.000000,0.000000]
output:  1.12

v=[3.912892,-2.865060,0.113480]
w=[-2.322290,-7.073600,1.110320]
output:  7.59
```

Function Test compute`_face_normal`:

```
Filename: cone.off
Index 0 : -0.15451, -0.07725, -0.02447

Index 10 : 0.02447, -0.07725, -0.15451

Index 20 : 0.15451, -0.07725, 0.02447
```

Function Test compute_face_area:

```
Filename: cone.off
Index 0 : 0.09
Filename: kitten.off
Index 10 : 8.48
```

Function Test is_connected_faces:

```
 Filename:
 cone.offIndex1  Index2  output
              0       1   TRUE
              0      38   TRUE
              0       2   TRUE
              0       3  FALSE
              0       4  FALSE
             22      23   TRUE
             22      20   TRUE
             22      24   TRUE
             22      35  FALSE
             22      39  FALSE
```

Test 1 :

```
Input: input1.txt

Output: output1.txt
```

Test 2 :

```
Input: input2.txt

Output: output2.txt
```

Grading Rubric

```
Computer Project #05

General Requirements:

Scoring Summary

  ( 4 pts) Coding Standard 1-9

             (descriptive comments, function headers, etc...)

Implementation:
```

(4 pts) open_file function (no Mimir test)

-3 Did not use try/except

(4 pts) read_face_data function

(4 pts) read_vertex_data function

(2 pts) compute_cross function

(2 pts) compute_distance function

(4 pts) compute_face_normal function

(4 pts) compute_face_area function

(4 pts) is_connected_faces function

(4 pts) check_valid function

(4 pts) Test1

(5 pts) Test2

Note: hard coding an answer earns zero points for the whole project - 10 points for not using main()