Assignment 1 - 3D Triangle Mesh Rigid Body Mass

In this assignment you will use symbolic computation to create functions to compute the integrals necessary to compute volume, center of mass, and rotational inertia of a tetrahedron, and then use these functions to compute the mass matrix of triangle meshes.

Getting Started

These steps will be similar for any operating system, but are described from the point of view of a windows user. Should you encounter any problems specific to linux or mac then please start a discussion online.

Install Anaconda3, or make sure you're using a relatively recent version (e.g., I'm using python version 3.9.7).

Start anaconda prompt with administrator privileges to be able to install modules. To have a clean environment specific to this course, I would suggest creating and activating an environment specific to this course, but this is not necessary:

```
conda create --name comp559w24
conda activate comp559w24
```

The first assignment needs sympy for symbolic computation and libigl for loading meshes.

```
python -m pip install sympy
python -m pip install libigl
```

For this assignment, the sympy docs are the most important, but here I also provide you links to documentation for libigl, though expect the c++ docs for this library are more complete.

- https://docs.sympy.org/latest/index.html
- https://libigl.github.io/libigl-python-bindings/

You should ultimately choose to use an IDE with an interactive debugger, e.g., vscode or pycharm or others. You may have similar steps to create an isolated environment within the IDE, or have a mechanism to tell the IDE about the one you create on the command line.

Objectives

This assignment has several steps where the marks associated with different steps are below. Use the following code snippet to import modules and set up command line parameters to allow you to test individual objectives and final results. Your assignment file must be named `comp559a1.py` and must include your name and student number in comments at the top.

```
from sympy import *
import numpy as np
import igl
import argparse

parser = argparse.ArgumentParser()
parser.add_argument("--file", type=str, default="cube.obj")
parser.add_argument("--density", type-float, default=1000, help="density in kg/m^3")
parser.add_argument("--scale", nargs=3, type=float, default=(1,1,1),help="scale the mesh")
parser.add_argument("--translate", nargs=3, type=float, default=(0,0,0),help="translate (after scale)")
parser.add_argument("--test", type=int, help="run a numbered unit test")
args = parser.parse_args()

V, _, _, F, _, _ = igl.read_obj(args.file)
V = V * args.scale
V = V + args.translate
```

1. (1 mark) Compute the volume of a tetrahedron. You will need to define symbolic variables that you need and note that python and sympy offers several ways to define matrices and vectors (e.g., `CoordSys3D`, `Point3D`, `MatrixSymbol`). Your easiest option is to simply use `sympy.Matrix` objects, e.g., `x = Matrix(symbols('x0 x1 x2'))`. Use `sympy.lambdaify` to let you efficiently evaluate your volume function. Write code to permit testing by using the argument `--test 1` which should print the volume and mass (using the density argument) of the tetrahedron defined by first face of the loaded mesh combined with the origin. For example, for the cube this test should print the following to the console:

   ```
   vol =  0.666666666666668
   mass =  666.666666666668
   ```

2. (1 mark) Compute the mass weighted center of mass of a tetrahedron. Note that the barycenter of a tetrahedron is simply the average of its vertices, and this could be useful to keep in mind when testing. Because we ultimately want to compute a weighted average across many tetrahedra, the mass weighted quantity is more useful as it can handle zero volume and inverted tetrahedra. Use `sympy.integrate` along with the change of coordinates integration seen in class to integrate position times density across the tetrahedron. Providing the argument `--test 2` to your program should print the mass weighted center of mass of the tetrahedron defined by first face of the loaded mesh combined with the origin. Print the transpose to display the result as a row vector, for example, for the sphere this test should print the following to the console:

   ```
   weighted com =  [[ 0.96596142  0.31386479 -7.33810853]]
   ```

3. (2 marks) Compute the 3x3 rotational inertia tensor of a tetrahedron. Providing the argument `--test 3` to your program should print the matrix for the tetrahedron defined by the first triangle of the mesh combined with the origin. For example, for the cube, the following should print to the console:

   ```
   J =
    [[ 533.33333333 -133.33333333    0.        ]
    [-133.33333333  266.66666667  133.33333333]
    [   0.          133.33333333  533.33333333]]
   ```

4. (1 mark) Compute quantities for the full mesh. Sum up the signed quantities (i.e., accounting for inverted tetrahedra) for the volume, mass weighted center of mass, and 3x3 rotational inertia. Compute the total mass from the density and volume, and compute the position of the center of mass by dividing by the mass. You should print out these quantities along with the full 6x6 mass matrix, using `np.round` to round all quantities, except the center of mass, to 3 decimal places. Your output for `bunnyLowRes.obj` should look like the following:

```
volume =  0.366
mass =  366.26
com =  [[-0.03211898 -0.17747379  0.07533627]]
J =
 [[43.692  6.963  0.919]
 [ 6.963 40.369  6.091]
 [ 0.919  6.091 60.91 ]]
M =
 [[366.26    0.      0.      0.     27.593  65.002]
 [  0.    366.26    0.    -27.593   0.    -11.764]
 [  0.      0.    366.26  -65.002  11.764   0.   ]
 [  0.    -27.593 -65.002  43.692   6.963   0.919]
 [ 27.593   0.     11.764   6.963  40.369   6.091]
 [ 65.002 -11.764   0.      0.919   6.091  60.91 ]]
```

See moment of inertia formulas to further check your results with values computed by hand for solid rectangular cuboid and solid ellipsoid by using the `--scale` parameter with non-uniform scale values. You can also use `--translate` to shift the bunny vertices such that its center of mass is the origin (within floating point precision).

Finished?

Great! Submit only a file called `comp559a1.py` using the comments at the top of the file to list your name, student number, and any other information you would put in a readme file (e.g., request to use your late penalty waiver). Do not submit an archive file, that is, no zip files for this assignment!

Note that you are encouraged to discuss assignments with your classmates, but not to the point of sharing code and answers. All code and written answers must be your own. Please see the course outline and the fine print in the slides of the first lecture.