CMPUT307 Lab 3: 3D Geometric Transforms

Due on Feb **4, 2024**

In this lab we will learn about 3D geometric transforms. A skeleton code file is provided and you need to fill in the blanks. For documentation of NumPy, please refer to the NumPy user guide (included) or https://numpy.org/doc/stable/. You are NOT allowed to import any other package to finish this lab. Please give credit to any code you referenced.

Exercise 1: 3D Transformation Basics

In this exercise, we work on some basics about 3D transformation. First finish the implementation of the following functions using NumPy:

```
  generateTranslationMatrix(x, y, z): return the homogeneous
transformation matrix for the given translation (x, y, z).

  generateScalingMatrix(sx, sy, sz): return the homogeneous
transformation matrix for the given scaling parameters (sx, sy, sz).

  generateRotationMatrix(rad, axis): return the homogeneous
transformation matrix for the given rotation parameters (rad, axis).
rad is the radians for rotation, axis specifies the axis for rotation.
```

Solve the following questions using NumPy and print out your results. For each case, finish the corresponding function and print out the four results in that function.

a) Case 1. What is the homogenous transformation matrix for:
1) Translation of (2, 3, -2)
2) Followed by Scaling with (0.5 , 2 , 2)
3) Followed by Rotation of 45 degrees around Z-axis
4) Apply the above final transformation matrix to a 3D point at (2, 3, 4).

b) **Case 2.** What is the homogenous transformation matrix for:
1) Scaling with (3, 1, 3)
2) Followed by Translation of (4, -2 , 3)
3) Followed by Rotation of -30 degrees around Y-axis
4) Apply the above final transformation matrix to a 3D point at (6, 5, 2).

c) Case 3. What is the homogenous transformation matrix for:
1) Rotation of 15 degrees around X-axis
2) Followed by Scaling with (2 , 2 , -2)
3) Followed by Translation of (5, 2, -3)
4) Apply the above final transformation matrix to a 3D point at (3, 2, 5).

Exercise 2: 3D Transformation with Toy Point Cloud Sphere

In this exercise, we create a point cloud 3D sphere and use it to play around with 3D transformations. Figure 1 displays the expected visualized results for this exercise.

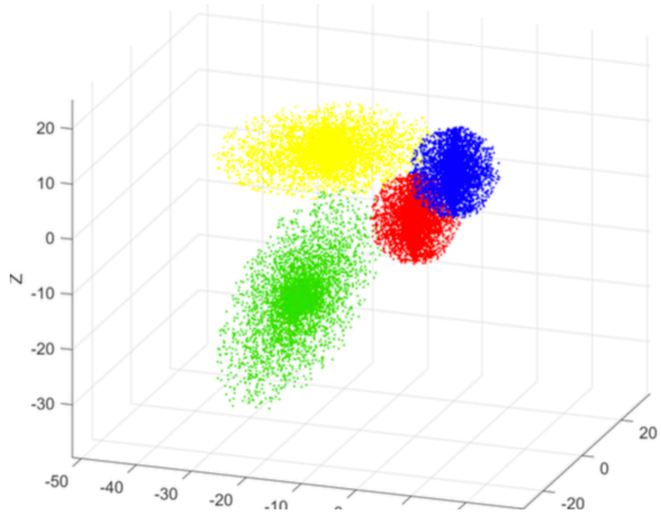

Fig. 1 Visualization of spheres after applying different 3D transformations.

Part 1: Generating a Point Cloud Sphere

First we generate a random sphere of n points. This can be done by sampling n random points from a uniform distribution. For convenience, we generate those points in spherical coordinates (radial distance, polar angle, azimuthal angle) first, and then convert them to cartesian coordinates (x, y, z).

Finish the implementation of:

```
  generateRandomSphere(r, n): generate a point cloud of n points in
spherical coordinates (radial distance, polar angle, azimuthal angle).

  sphericalToCatesian(coors): convert n points in spherical
coordinates to cartesian coordinates, then add a row of 1s to them to
convert them to homogeneous coordinates.
```

Part 2: Decomposition of 3D Transformation

Suppose we generated a random sphere point cloud, and the coordinates of the points are "sphere1".

a) First we generate a random composite transformation. Finish the implementation of:

```
  applyRandomTransformation(sphere1): generate two random
transformations, one of each (scaling, rotation), apply them to the
input sphere in random order, then apply a random translation. Return
the transformed coordinates of the sphere, the composite
transformation matrix,and the three random transformation matrices you
generated.
```

b) Next, let's calculate the composite translation matrix with only "sphere1" and "sphere2". Let's denote the transformation equation as:

$$\boldsymbol{M'} = \boldsymbol{HM},$$

Where $\boldsymbol{M}$ and $\boldsymbol{M'}$ (4 x n) are the original and transformed homogeneous coordinates, and $\boldsymbol{H}$ is the transformation matrix. Now if we factorize $\boldsymbol{M}$ using SVD (singular value decomposition):

$$\boldsymbol{M} = \boldsymbol{U\Sigma V^T},$$

where $\boldsymbol{U}$ is a 4 x 4 orthogonal matrix, $\boldsymbol{V}$ is an n x n orthogonal matrix, and $\boldsymbol{\Sigma}$ is a 4 x n matrix with singular values of $\boldsymbol{M}$ on the diagonal, we can define the pseudo inverse (generalized inverse) of $\boldsymbol{M}$ as:

$$\boldsymbol{M^+ = V\Sigma^+U^T}.$$

In a comment block in your code, use the formulas given above to derive how you can calculate $\boldsymbol{H}$ using pseudo inverse (you can use latex notation if you prefer, **you must show detailed steps to get full marks**). You can assume $\boldsymbol{\Sigma\Sigma^+} = \boldsymbol{I_{4\times4}}$ in this case. Finish implementation of `calculateTransformation(sphere1, sphere2)` using SVD and/or pseudo inverse.

```
  calculateTransformation(sphere1, sphere2): calculate the composite
transformation matrix from sphere1 to sphere2.
```

Verify the transformation matrix you calculated is the same as the one returned from `applyRandomTransformation`. Those are float matrices so you might not want to use "==" operator to compare them. NumPy has some functions to help with this problem.

c) From the transformation matrix we obtained in the last step, we can extract the translation, scaling, and rotation components.

Take a close look at the general form of the translation, scaling, and rotation matrix, and analyze how they affect the composite matrix. For example, consider the reason that a specific element in the composite matrix is non-zero.

Finish the implementation of decomposeTransformation(m).

```
decomposeTransformation(m): decompose the transformation and return
the translation, scaling, and rotation matrices.
```

Verify the three components you get from decomposition is the same from the ones you obtained from `applyRandomTransformation`.

Grading:

- Code Quality: 10%
- Comments: 5%
- Exercise 1: 35%
    - Matrix generation: 10% * 3
    - 3 test cases: 5%
- Exercise 2: 50%
    - Random sphere generation: 5%
    - Random transformation generation: 5%
    - Composite transformation matrix calculation: 15%
    - Transformation decomposition: 25%

Some NumPy stuff you might find helpful (you can find documentation and examples at https://numpy.org/doc/stable/reference/index.html):

np.pi
np.eye
np.sin
np.cos
np.transpose
np.random.rand
np.concatenate
np.stack
np.linalg.svd
np.linalg.pinv
np.isclose
np.all
np.allclose
np.arcsin
np.arctan