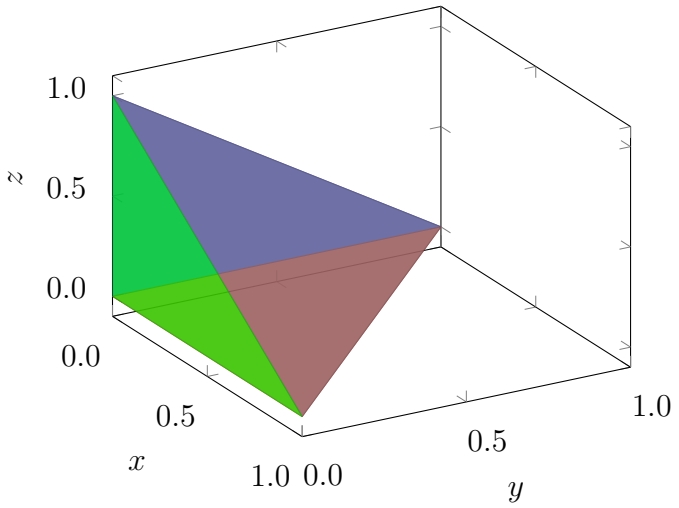
EA1 Homework Program 7: Triangle Meshes and Linear Transformations

In this assignment, you will write a MATLAB function which applies a linear transformation to a *triangle mesh*, a particular type of three-dimensional graphics model. In such a model, we represent a three-dimensional object as a collection of planar triangles sitting in the three-dimensional space $\mathbb{R}^3$. Each such triangle is defined by three corners, or *vertices*, with each vertex having its own x-, y-, and z-coordinates. In other words, we define such a triangle by providing three vectors in $\mathbb{R}^3$, with each vector representing a vertex (or corner) of the triangle. For example, we might define a pyramid as the collection of the following four colored triangles (made somewhat transparent here for better visibility):



	Vertex			Vertex			Vertex		
	x	y	z	x	y	z	x	y	z
Triangle 1	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0
Triangle 2	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0
Triangle 3	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0
Triangle 4	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0

However, there is redundancy in this description of the pyramid: we have listed 12 vertices (three for each triangle), but in fact there are only four distinct vertices in the pyramid. A more compact description of the pyramid would involve two separate tables as follows:

Matrix V	x	y	z
Vertex 1	0.0	0.0	0.0
Vertex 2	1.0	0.0	0.0
Vertex 3	0.0	1.0	0.0
Vertex 4	0.0	0.0	1.0

Matrix T	Vertices		
Triangle 1	1	2	3
Triangle 2	1	2	4
Triangle 3	1	3	4
Triangle 4	2	3	4

The table on the left, which we will call matrix V, is simply a list of vertices, labeled with an index ranging from 1 to 4. The table on the right, which we will call matrix T, describes each triangle in the pyramid as before, but now the vertices are listed by their index rather than by their coordinates: for example, the corners of the second triangle are Vertex 1, Vertex 2, and Vertex 4. MATLAB can display triangle meshes using built-in functions `trimesh` (which draws only the triangle edges)

and `trisurf` (which also fills in the triangles). Each of these functions takes four input arguments: the first input is the matrix T above, and the second through fourth inputs are the three columns of the matrix V above. Thus to draw our pyramid, we could enter the following in MATLAB:

```
V = [0 0 0; 1 0 0; 0 1 0; 0 0 1];
T = [1 2 3; 1 2 4; 1 3 4; 2 3 4];
trisurf(T, V(:,1), V(:,2), V(:,3));
```

You can interact with figures in MATLAB; in particular, if you click on the "Rotate 3D" tool button on the top of the figure, then you can click and drag the plot to examine it from multiple viewpoints.

What about color? If you try the above commands, you will notice that MATLAB uses some default colors for the triangles. We can assign our own colors to the triangles by adding a column to the matrix T as in the following example:

Matrix T	Vertices			Color
Triangle 1	1	2	3	2
Triangle 2	1	2	4	4
Triangle 3	1	3	4	1
Triangle 4	2	3	4	3

The entries in the fourth column of this new matrix T define a color for each triangle; for example, the second triangle should have color 4. To provide such triangle colors to `trisurf`, we give this fourth column of the matrix T as an optional fifth input argument:

```
V = [0 0 0; 1 0 0; 0 1 0; 0 0 1];            % same V as before
T = [1 2 3 2; 1 2 4 4; 1 3 4 1; 2 3 4 3];    % new T
trisurf(T(:,1:3), V(:,1), V(:,2), V(:,3), T(:,4));
```

Now you will see triangle colors different from the original ones, but we still have not specified our own colors. To do this, we must assign specific colors to "color 1," "color 2," and so forth. We accomplish this using the `colormap` function. The input to this `colormap` function is yet another matrix, such as the following matrix C:

Matrix C	R	G	B
Color 1	0.0	0.0	1.0
Color 2	1.0	0.0	0.0
Color 3	0.5	0.5	0.5
Color 4	0.0	1.0	0.0

Each row of this matrix C defines a color by assigning a specific value between 0 and 1 to three standard color channels: red (R), green (G), and blue (B), in that order. In our example, color 1 represents blue and color 3 represents a shade of gray. We can now assign these specific colors to our triangles as follows:

```
V = [0 0 0; 1 0 0; 0 1 0; 0 0 1];
T = [1 2 3 2; 1 2 4 4; 1 3 4 1; 2 3 4 3];
C = [0 0 1; 1 0 0; 0.5 0.5 0.5; 0 1 0];
trisurf(T(:,1:3), V(:,1), V(:,2), V(:,3), T(:,4));
colormap(C);
```

We can apply exactly the same ideas to more complicated shapes—the only differences will be the numbers of rows in the matrices V, T, and C. This assignment comes with a second file called `mesh_examples.mat` which contains data for ten different objects. To display one of these objects using `trisurf`, try the following commands:

```
load mesh_examples.mat
V = flower.vertices;
T = flower.triangles;
C = flower.colormap;
trisurf(T(:,1:3), V(:,1), V(:,2), V(:,3), T(:,4));
colormap(C);
axis equal;
```

To display a different object, just substitute `flower` with one of the other nine shapes which get loaded from the file `mesh_examples.mat`.

Now recall that every 3-by-3 matrix A represents a linear transformation from $\mathbb{R}^3$ to $\mathbb{R}^3$. Given A, we can apply this transformation to each of the vertices in the matrix V to get a new, transformed vertex matrix V_{new}. We can then display the transformed object using the same `trisurf` command as above, but replacing the old V with the new one, V_{new}. Note that the triangle matrix T and colormap matrix C do not change at all in this transformation! Your task in this assignment is to write a MATLAB function called `transform_mesh` which takes inputs A, V, T, and C, computes the transformed vertex matrix V_{new}, and uses `trisurf` to plot both the original and transformed objects on the same set of axes. The function declaration should be:

```
function Vnew = transform_mesh(A,V,T,C)
```

Your function should perform the following steps:

1. Include an appropriate H1 comment line.

2. Include appropriate help comments.

3. Include a comment section like

```
% Homework Program 7
%
% Name:      Granger, Hermione
% Section:   22
% Date:      11/10/2023
```

4. Use features of the arguments block to check for bad inputs, as you did in the previous two homeworks. For example, to check that `A` is a 3-by-3 numeric matrix and `T` is a matrix with 4 columns with numeric integer values ≥ 1, use the following lines of code in the arguments block:

```
A (3,3) {mustBeNumeric}
T (:,4) {mustBeNumeric,mustBeInteger,
   mustBeGreaterThanOrEqual(T,1)}
```

In your function, include similar lines of code for the other inputs `V` and `C`. Type `help arguments` or `doc arguments` in the command window for more information about how to use these MATLAB features.

5. Apply the linear transformation represented by the matrix `A` to each vertex in `V`, and store the results in the matrix `Vnew` (which should be the same size and shape as `V`). Note that each *row* of `V` represents a vertex in $\mathbb{R}^3$, not each column, so the transpose operator may be useful.

6. Plot the original and transformed objects using the following code:

```
% Plot the new and original meshes
trisurf(T(:,1:3),Vnew(:,1),Vnew(:,2),Vnew(:,3),T(:,4));
hold on
trisurf(T(:,1:3),V(:,1),V(:,2),V(:,3),T(:,4), ...
        'EdgeColor','none','FaceAlpha',0.2);
hold off
colormap(C);
axis equal;
```

   The `EdgeColor` and `FaceAlpha` parts of the `trisurf` command simply make the original object somewhat transparent, so that it is easier for you to tell which is the original object and which is the transformed one.

The final part of the assignment is to calculate the matrices of some linear transformations from $\mathbb{R}^3$ to $\mathbb{R}^3$, and use your function to see how these transformations change the sample objects. In particular, you should calculate the following 3-by-3 matrices:

1. Find the matrix representing scaling by a factor of 0.67 in the x-direction and 1.5 in both the y- and z-directions, and store it in the variable `A_scale`.

2. Find the matrix representing reflection across the xz-plane, and store it in the variable `A_refl`.

3. Find the matrix representing rotation by $45°$ about the axis defined by the line going through the origin and the vector `[0; 1; 1]`, and store it in the variable `A_rot`. Hint: there is a MATLAB function called `vrrotvec2mat` which will perform this calculation for you, but you have to give it an angle in radians, not degrees.

4. Find the matrix representing the combined transformation of first going through the rotation `A_rot`, then going through the reflection `A_refl`, and finally going through the scaling `A_scale`. Store the matrix in the variable `A`.

5. Find the matrix representing the combined transformation of first going through the reflection `A_refl`, then going through the scaling `A_scale`, and finally going through the rotation `A_rot`. Store the matrix in the variable `B`.

6. Type the names of the five matrices `A_scale`, `A_refl`, `A_rot`, `A`, and `B` in the command window, one by one, so that MATLAB displays their contents. Copy and paste the results to the end of your function, and comment out these lines.

Use your function `transform_mesh` to apply these five transformations (or others) to the example objects you loaded from the file `mesh_examples.mat`. You do not need to turn in the results of doing so. Instead, simply submit your function file `transform_mesh.m` and make sure it contains the commented-out values of the five matrices listed above. A few examples of using the transformations are shown below.

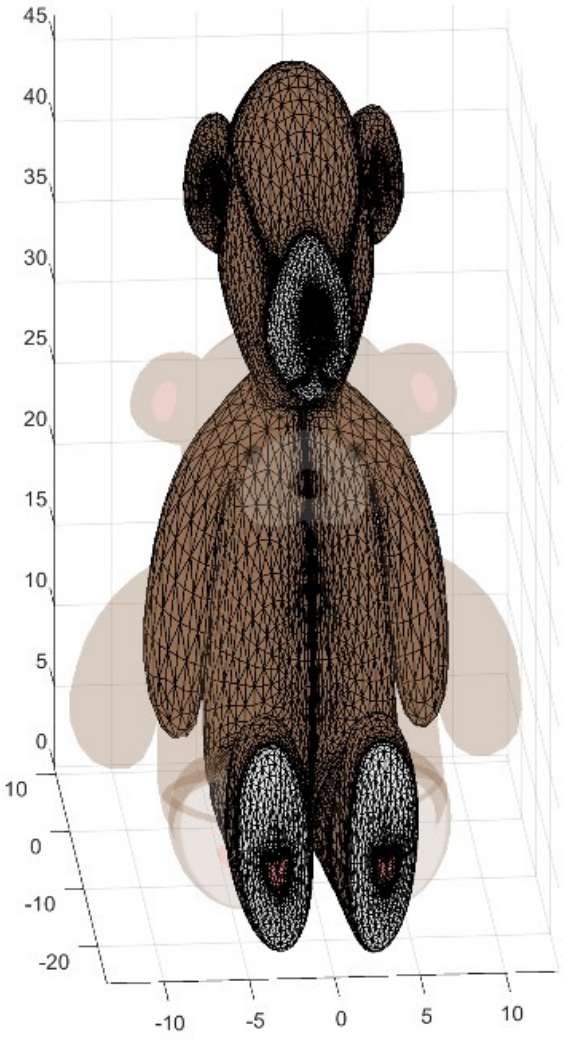


Figure 1: Scaling corresponding to transformation 1

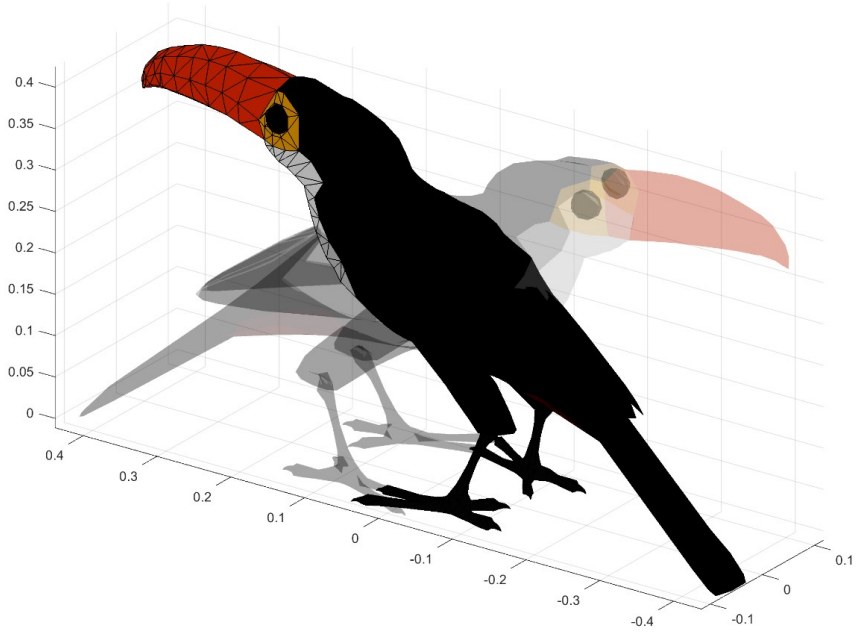


Figure 2: Reflection corresponding to transformation 2

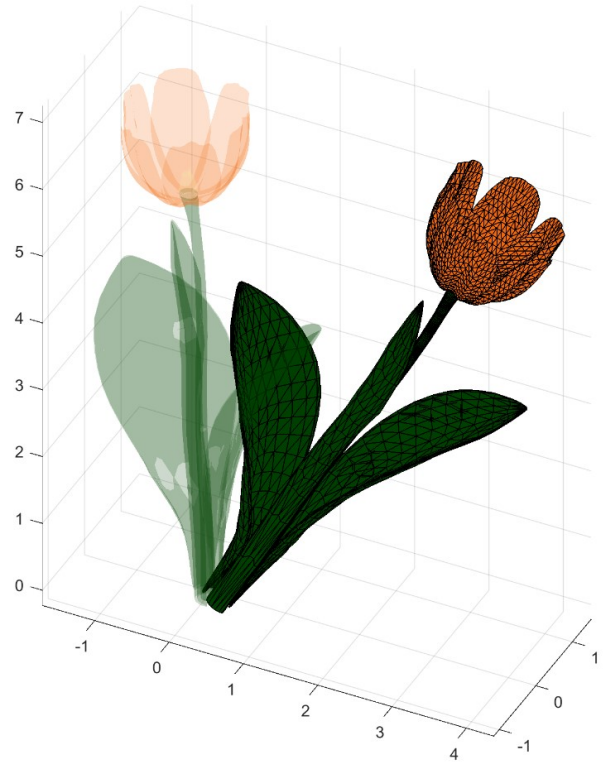


Figure 3: Rotation corresponding to transformation 3