Computer Vision
Lab Exercise 7
Surface rendering

In previous lab exercises, you have worked on estimating the 3D shape of an object from images. Once the shape is recovered in terms of a point cloud, the original texture of the object in image can be mapped to the point cloud to improve the visualization.

In this lab, you will create a function that takes as input the reconstructed point cloud (The merged cloud after aligning with procrustes), the entries of the motion matrix corresponding to the first point (so first two rows). The mean of the measurement matrix across rows and the image you consider as the main view.

Note : This lab will require some modification of the previously created *reconstruction_demo.m* file.
Please modify the script as suggested below. This modification is necessary because we require the variables 'M1' and 'MeanFrame1' for the surface rendering function.

After this line (calling sfm on the measurement matrix),

```
1 [M, S, p]=sfm(X);
```

Add the following lines:

```
1 % Save the M matrix and Meanvalues for the first frame. In ...
      this example,the
```

```
% first frame is the camera plane (view) where every point ...
    will be projected
% Please do check if M is non-zero before selection. ...
    Otherwise, you
% have to select another view
if i==1 && ¬p
    M1=M(1:2,:);
    MeanFrame1=sum(X,2)/numPoints;
end
```

1 Surface rendering

Once the necessary changes have been made to the *reconstruction_demo.m* file, proceed to fill out the surface render function. The steps are detailed below.

1. Assign columns of the computed point cloud to their respective coordinate axes (They should be in the order X, Y, Z)

```
    % (X,Y,Z) of the point cloud
    pointcloud = unique(pointcloud', 'rows')';
    X = ...
    Y = ...
    Z = ...
```

   The MATLAB fucntion `surf` can be used to display the texture surface. But, surf uses all points given as input. So, points that are occluded in the images would then need to be removed. To achieve this, you can compute the viewing direction, place a clipping plane at the centre of mass of the object and remove points that lie behind the plane.

2. Compute the cross product of the X and Y entries of M and normalize:

```
    viewdir = cross(M(1,:), M(2,:));
    viewdir = viewdir/sum(abs(viewdir));
```

3. Determine the means along each axis of the point cloud(X, Y and Z). 'X0' contains as elements the transposed columns of the point clouds

you assigned in step 1. Tile 'X1' and 'Xm' to be the same size as X1 with the computed viewing direction and mean respectively.

```
m  = [... ;...  ;... ];
X0 = [... ;...  ;... ];
X1 = repmat(viewdir,...  ,... );
Xm = repmat(m,...  ,... );
```

4. Find the indices where the dot product between the mean subtracted points (given by 'X0 - Xm') and the viewing direction is negative. These are the points that need to be removed.

```
indices = find(...);
X(indices) = [];
Y(indices) = [];
Z(indices) = [];
```

5. Define the grid that your surface can then be created on. The array 'ti' defines the range of values (for eg., -500:500 or -700:2.5:700)

```
ti = ...
[qx,qy] = meshgrid(... , ...);
```

6. Use the in-built MATLAB function `TriScatteredInterp` to determine the interpolant and apply it to the defined grid.

```
% Surface generation using TriScatteredInterp
% You can also use scatteredInterpolant instead.
% Please check the detailed usage of these functions
F  = TriScatteredInterp(...);
qz = F(qx,qy);
```

7. Reshape the grid values to row vectors of the number of elements.(**Hint:** Have a look at the `numel` function)

```
% Reshape (qx,qy,qz) to row vectors for next step
qxrow = ...
qyrow = ...
qzrow = ...
```

8. Transform the reshaped grid (with the reshaped rows each occupying a different row) to the reference view by multiplying with the motion/-transformation matrix: M. To compensate for the mean subtraction step with the measurement matrix – means of the frame (the 'Mean-frame' variable from the *reconstruction_demo.m*) along each axis need to be added to the corresponding grid axis.

```
% Transform to the main view using the ...
    corresponding transformation matrix
q_xy = ...

% All transformed points are normalized by mean ...
    values in advance, we have to move
% them to the correct positions by adding ...
    corresponding mean values of each dimension.
q_x = ...
q_y = ...
```

9. Assign image channels (R,G,B) to the appropriate variable('imgr', 'imgg', 'imgb'). For each colour channel, obtain the intensity values at linear indices given by 'y '(row subscript) and 'x'(column subscript) grid values.

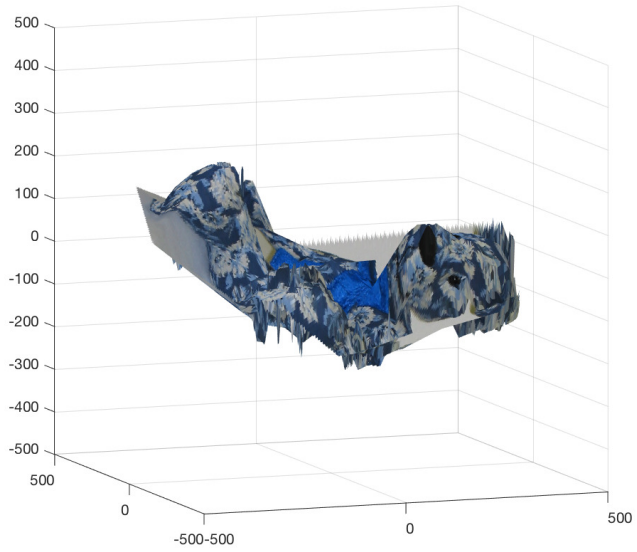
   [**Hint**: look at the `sub2ind` function and also note that the grid values need to be rounded to serve as subscripts]

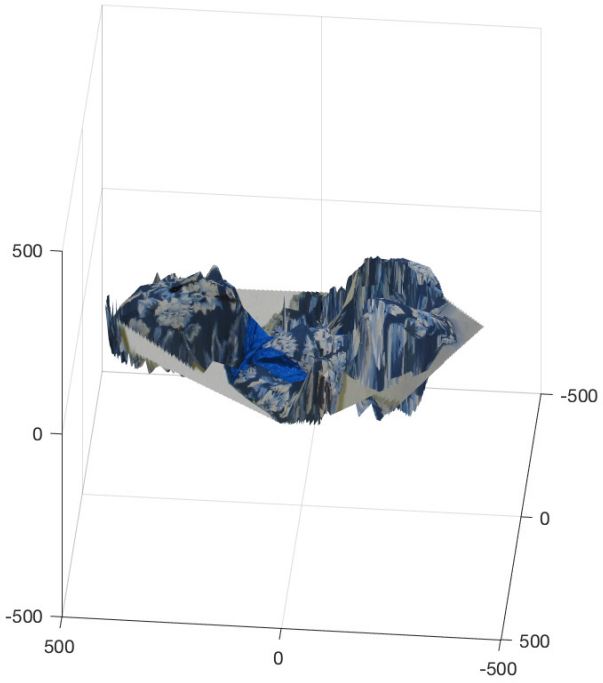
```
1       if(size(img,3)==3)
2           % Select the corresponding r,g,b image channels
3           imgr = ...
4           imgg = ...
5           imgb = ...
6
7           % Color selection from image according to ...
                (q_y, q_x) using sub2ind
8           Cr = ...
9           Cg = ...
10          Cb = ...
```

You will then display the surface given the rendering parameters and shading.

2 Output

Here are displayed the rendered version of the teddy bear object for reference. You can use it to compare against your own implementation.




500
0
-500
500
0
-500
-500
0
500