Contents

- Step 1: Clear variables/figures and define global variables and constants:
- Step 2: Read input data:
- Step 3: Parse EXCEL sheet:
- Step 4: Visualize the constructed input 3D point cloud:
- Step 5: Construct high-resolution 2D images from 3D point cloud:
- Step 6: Project the constructed high-resolution 3D surface to high-resolution 2D image:
- Step 7: Enhance the projected high-resolution 2D image:


```
function status = construct_high_resolution_2D_image_from_3D_point_cloud_data()
```

```
%=========================================================================
% Project: Construct high-resolution 2D images from 3D point cloud
%=========================================================================
% File: construct_high_resolution_2D_image_from_3D_point_cloud_data.m
% Author: Mohsen Ghazel
% Date: March 10, 2021
%=========================================================================
% Objectives:
%=========================================================================
%  - To read and parse 3D point cloud
%  - To fit high resolution surface models to the 3D point cloud
%  - To project 3D high resolution surface to a high resolution 2D image
%  - To enhance and visualize the constructed high resolution 2D image.
%=========================================================================
% Input:
%=========================================================================
% - None
%=========================================================================
% Output:
%=========================================================================
%  - status = 1 for success and 0 for failure
%=========================================================================
% Execution:
%=========================================================================
%
% >> status = construct_high_resolution_2D_image_from_3D_point_cloud_data(  )
%
%=========================================================================
% History
%=========================================================================
% Date                      Changes
%=========================================================================
% March 10th, 2021        Initial definition
%=========================================================================
% License
%=========================================================================
% MIT License: Free to copy, use, modify, share and redistribute.
% Copyright (c) 2021 mghazel2020
%=========================================================================
% display a message
fprintf(1, '===============================================================\n');
fprintf(1, 'Project: Construct high-resolution 2D images from 3D point cloud:\n');
fprintf(1, '===============================================================\n');
fprintf(1, 'Date: %s\n', datestr(now) );
fprintf(1, '===============================================================\n\n');
%=========================================================================
```

```
===============================================================
Project: Construct high-resolution 2D images from 3D point cloud:
===============================================================
Date: 11-Mar-2021 10:20:47
===============================================================
```

Step 1: Clear variables/figures and define global variables and constants:

```
%=========================================================================
% clear screen and close any open figures
close all;
clear all;
clc;
% supress warnings
warning('off');

fprintf(1, '================================================================\n');
fprintf(1, 'Step 1: Clear variables/figures and define global variables and constants:\n');
fprintf(1, '================================================================\n');
```

```
% execution status
status = 1;

% Missing data-flag = "-9999"
MISSING_DATA_FLAG = -9999;

% Number of distinct scan-lines per full circle
NUM_SCAN_LINES_PER_FULL_CIRCLE = 80000;

% Tire Outer and Inner Radui
R_outer = 12.25; % outer-radius
R_inner = 7.8125; % inner-radius

% estimatd tire thickness
true_tire_thickness = abs(R_outer - R_inner);

% Final concatenated vectors for visualization
XX = []; % x-coordinates of the points
YY = []; % y-coordinates of the points
% Z-coordinates
ZZ = []; % z-coordinates of the points

% start of execution
start_time = tic();

%==========================================================================
```

```
================================================================
Step 1: Clear variables/figures and define global variables and constants:
================================================================
```

Step 2: Read input data:

```
%==========================================================================
fprintf(1, '================================================================\n');
fprintf(1, 'Step 2: Read input data:\n');
fprintf(1, '================================================================\n');
%--------------------------------------------------------------------------
% input data file
%--------------------------------------------------------------------------
% input file name
input_excel_file_name  = '..\data\input-point-cloud-tire.xlsx';
% sheet name
sheet_name = 'Test Data - 25Apr2017';

% check if input file exists
if ~( exist(input_excel_file_name, 'file') == 2 )
    fprintf(1, 'Input data file: %s does not exist!\n', input_excel_file_name );
    fprintf(1, 'Please set the correct file name of the input data file and try again!)\n');
    status = 0;
    return;
end

% Check if sheet name is defined
if ~( exist(sheet_name, 'var') == 0 )
    fprintf(1, 'Input Excel file sheet name: %s is not defined!...\n', sheet_name );
    fprintf(1, 'Please set the correct sheet-name of the input EXCEL sheet and try again...)\n');
    status = 0;
    return;
end

% Read the input data from the EXCEL sheet
[ndata, text, alldata] = xlsread(input_excel_file_name, sheet_name);

% number of scanned lines
num_lines = size(ndata, 1);

%--------------------------------------------------------------------------
% set the output directory
%--------------------------------------------------------------------------
output_directory = '..\results\';

%==========================================================================
```

```
================================================================
Step 2: Read input data:
================================================================
```

Step 3: Parse EXCEL sheet:

```
%==========================================================================
```

```
fprintf(1, '=================================================================\n');
fprintf(1, 'Step 3: Parse EXCEL sheet:\n');
fprintf(1, '=================================================================\n');
% 1) Parse EXCEL sheet
%-------------------------------------------------------------------------
% Get all valid data (not equal to the FLAG = "-9999"
%-------------------------------------------------------------------------
% Read the encoders column (1st column)
encoder_values = ndata(:, 1);

% Remove repeated lines
encoders_values_no_duplicates = unique(encoder_values);

% Fixed but a function of the angle
THETA = (max(encoders_values_no_duplicates) - min(encoders_values_no_duplicates)) / NUM_SCAN_LINES_PER_FULL_CIRCLE * 2 * pi;

% number of distinct encoder values
num_distinct_lines = size(encoders_values_no_duplicates, 1);

% Determine the angles between the scanned lines
% Declare a vector to store the angles
theta = zeros(num_distinct_lines, 1);

% Iterate over all the values and determine the individual angles between
% the scanned lines
for counter = 2:num_distinct_lines
    % define the angle between the 2 adthetajacent scan lines
    theta(counter) = THETA * ( encoders_values_no_duplicates(counter) - encoders_values_no_duplicates (counter - 1) ) / (max(encoders_values_no_dup
licates) - min(encoders_values_no_duplicates));
end

% define the first element
theta(1,1) = theta(2, 1);

% (x,y) coordinates data (skip first 7 columns containg the encoder values and new data)
data0 = ndata(:, 8:end);

% data structure to store data
scanned_lines = repmat( struct('encoder', 0,'size', 0, 'X',[], 'Y',[], 'Z',[]), 1, num_distinct_lines);

% counter of distinct lines
distinct_lines_number = 0;

% distinct encoder values
distinct_encoder_values = [];

% tire thickness profile
tire_thickness = zeros(num_distinct_lines, 3);

% iterate over the read data
for line_number = 1: num_lines
    %---------------------------------------------------------------------
    % Check if the line is a duplicate of one of the previous lines:
    %---------------------------------------------------------------------
    %   - Check if the encoder-value of this scanned line is the same as
    %     teh encode value of one of the previously-encoutered encoder
    %     values add the new-line encoder-value
    %---------------------------------------------------------------------
    new_encoder_values = [distinct_encoder_values encoder_values(line_number, 1)];

    % Remove duplicates
    temp = unique(new_encoder_values);

    % check for duplicates, if so skip this scanned-line
    if ( size(temp, 2) < size(new_encoder_values, 2)) % there are duplicates
        continue;
    end

    % increment the number of distinct lines counter
    distinct_lines_number = distinct_lines_number + 1;

    % assign the encoder value
    scanned_lines(distinct_lines_number).encoder = encoder_values(line_number, 1);

    % update the list of distinct encoder values
    distinct_encoder_values = [distinct_encoder_values encoder_values(line_number, 1)];

    % consider the current scanned line of 9x,y) coordinates
    line_scan = data0(line_number, 1:end);

    % find entries of valid data points (column-array)
    valid_data_indices = find(line_scan ~= MISSING_DATA_FLAG);

    % number of new valid points
    num_valid_entries = size(valid_data_indices, 2);

    % number of valid points
    num_valid_points = num_valid_entries / 2;
```

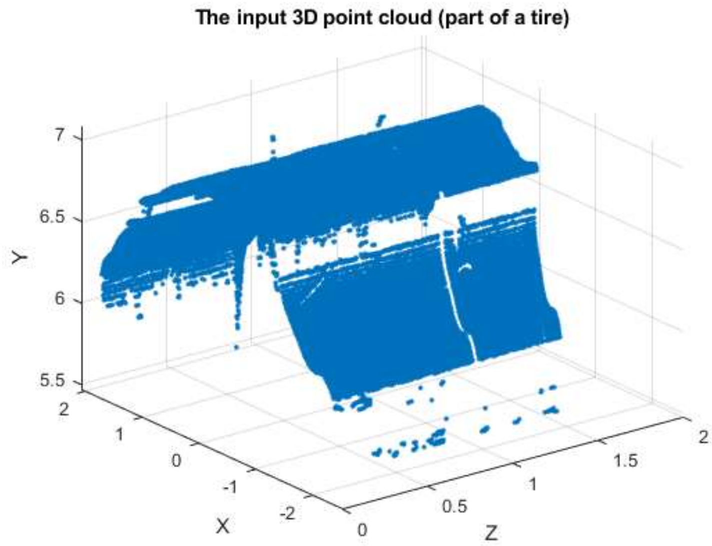
```
    % Assign the size = number of valid data points: each pair (x,y) represents one point
    scanned_lines(distinct_lines_number).size = num_valid_points;
    % X-ccordinates
    scanned_lines(distinct_lines_number).X = [scanned_lines(distinct_lines_number).X data0(line_number, valid_data_indices(1:2:end))];
    % Y-coordinates
    scanned_lines(distinct_lines_number).Y = [scanned_lines(distinct_lines_number).Y data0(line_number, valid_data_indices(2:2:end))];

    %----------------------------------------------------------------------
    % set Z-value: Use different radius value for different points
    %----------------------------------------------------------------------
    % copy (x,y) coordinates of points
    xx = scanned_lines(distinct_lines_number).X;
    yy = scanned_lines(distinct_lines_number).Y;

    % subtract the first point
    xx = xx - xx(1);
    yy = yy - yy(1);

    % tire thickness
    tire_thickness(distinct_lines_number, 1) = distinct_lines_number;
    tire_thickness(distinct_lines_number, 2) = true_tire_thickness;
    tire_thickness(distinct_lines_number, 3) = sqrt(xx(num_valid_points)*xx(num_valid_points) + yy(num_valid_points)*yy(num_valid_points));

    % compute the distance of each point from the centre of tire
    radius = zeros(num_valid_points, 1);

    % compute
    temp = xx.*xx + yy.*yy;

    % iterate and compute distance
    for point = 1: num_valid_points
        radius(point) = R_inner + sqrt(temp(point));
        radius(point) = R_inner + sqrt(xx(point)*xx(point) + yy(point)*yy(point));
    end

    % Z-value
    scanned_lines(distinct_lines_number).Z = [scanned_lines(distinct_lines_number).Z sum(theta(1:distinct_lines_number)) * radius]';

    %----------------------------------------------------------------------
    % Append coordinates to new vectors for visualization
    %----------------------------------------------------------------------
    % X-corrdinates
    XX = [ XX scanned_lines(distinct_lines_number).X ];

    % Y-coordinates
    YY = [ YY scanned_lines(distinct_lines_number).Y ];

    % Z-coordinates
    ZZ = [ ZZ scanned_lines(distinct_lines_number).Z];

end
%===========================================================================
```

```
================================================================
Step 3: Parse EXCEL sheet:
================================================================
```

Step 4: Visualize the constructed input 3D point cloud:

```
%===========================================================================
fprintf(1, '================================================================\n');
fprintf(1, 'Step 4: Visualize the constructed input 3D point cloud:\n');
fprintf(1, '================================================================\n');
% create a figure
h10 = figure(10);
plot3(ZZ, XX,YY,'.');
xlabel('Z');
ylabel('X');
zlabel('Y');
title(strcat(['The input 3D point cloud (part of a tire)']));
ylim([min(min(XX)) max(max(XX))]);
zlim([min(min(YY)) max(max(YY))]);
grid on
orient landscape
% save figure
saveas(h10,strcat([output_directory, 'input_3D_point_cloud_tire.jpg']));

%===========================================================================
```

```
================================================================
Step 4: Visualize the constructed input 3D point cloud:
================================================================
```

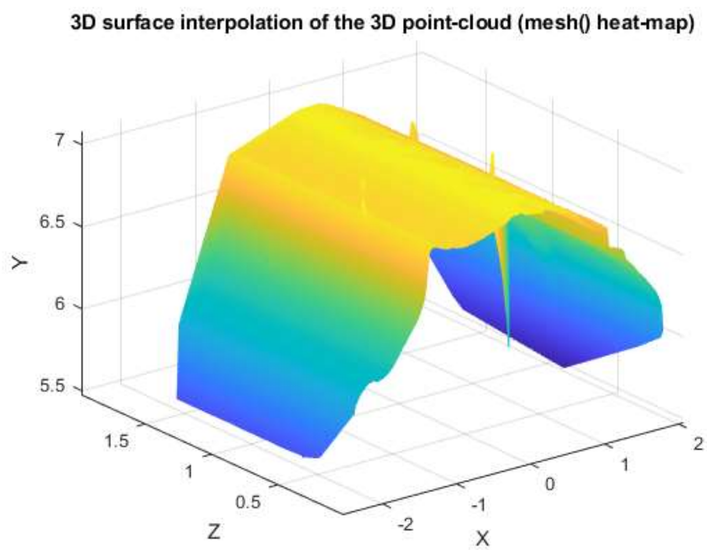


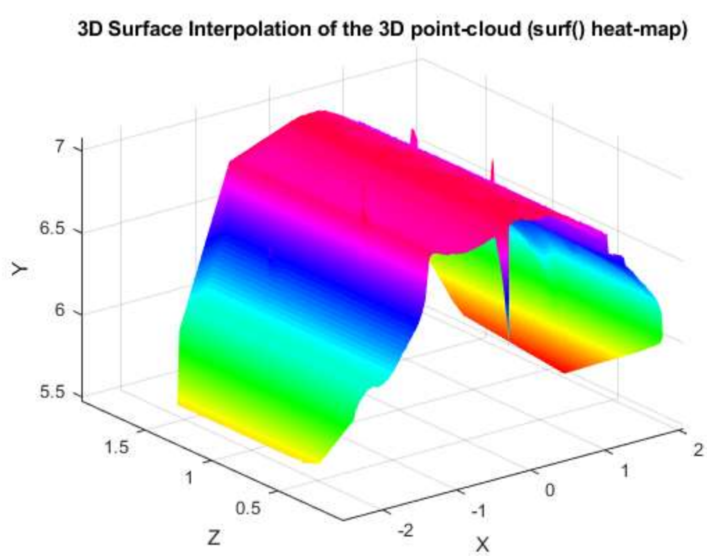
Step 5: Construct high-resolution 2D images from 3D point cloud:

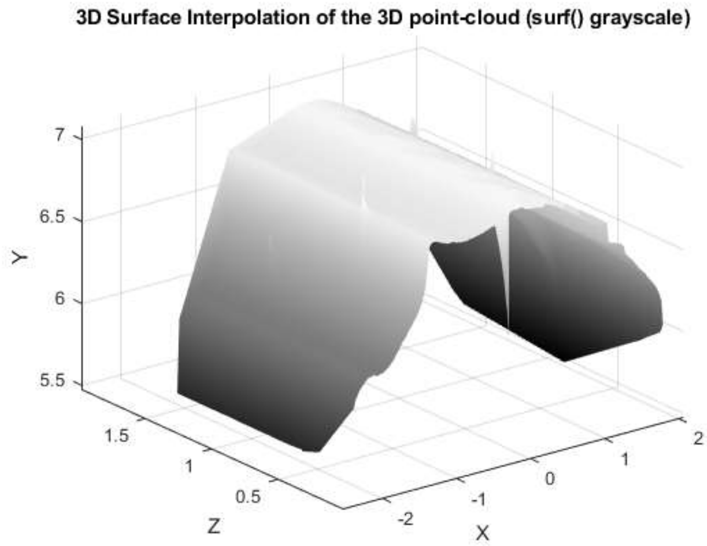
```
%==========================================================================
% vq = griddata(x,y,v,xq,yq) fits a surface of the form v = f(x,y) to the
% scattered data in the vectors (x,y,v). The griddata function
% interpolates the surface at the query points specified by (xq,yq) and
% returns the interpolated values, vq.
% The surface always passes through the data points defined by x and y.
%--------------------------------------------------------------------------
fprintf(1, '===============================================================\n');
fprintf(1, 'Step 5: Construct high-resolution 2D images from 3D point cloud:\n');
fprintf(1, '===============================================================\n');
% Define a regular grid and interpolate the scattered data over the grid.
[xq,zq] = meshgrid(min(min(XX)):0.001:max(max(XX)), min(min(ZZ)):0.001:max(max(ZZ)));
% interpolate the y-coordinates
yq = griddata(XX,ZZ,YY, xq,zq);
%--------------------------------------------------------------------------
% Heat-map visualization using mesh() functionality:
%--------------------------------------------------------------------------
h20 = figure(20);
mesh(xq,zq,yq);
xlabel('X');
ylabel('Z');
zlabel('Y');
title('3D surface interpolation of the 3D point-cloud (mesh() heat-map)');
xlim([min(min(XX)) max(max(XX))]);
ylim([min(min(ZZ)) max(max(ZZ))]);
grid on
orient landscape
% save figure
saveas(h20,strcat([output_directory, 'surface_3D_reconstruction_using_mesh_heat_map.jpg']));
%--------------------------------------------------------------------------
% Heat-map visualization using surf() functionality:
%--------------------------------------------------------------------------
h30 = figure(30);
colormap hsv
surf(xq,zq,yq,'FaceColor','interp',...
   'EdgeColor','none',...
   'FaceLighting','gouraud')
% set axes to tight
axis tight
xlabel('X');
ylabel('Z');
zlabel('Y');
title('3D Surface Interpolation of the 3D point-cloud (surf() heat-map)');
% orient landscape
orient landscape
% save figure
saveas(h30,strcat([output_directory, 'surface_3D_reconstruction_using_surf_heat_map.jpg']));
%--------------------------------------------------------------------------
% Grayscale visualization using surf() functionality:
%--------------------------------------------------------------------------
h40 = figure(40);
colormap(gray(256))
surf(xq,zq,yq,'FaceColor','interp',...
   'EdgeColor','none',...
   'FaceLighting','gouraud')
% daspect([5 5 1])
axis tight
xlabel('X');
```

```
ylabel('Z');
zlabel('Y');
title('3D Surface Interpolation of the 3D point-cloud (surf() grayscale)');
% orient landscape
orient landscape
% save figure
saveas(h40,strcat([output_directory, 'surface_3D_reconstruction_using_surf_grayscale.jpg']));

%===========================================================================
```

```
================================================================
Step 5: Construct high-resolution 2D images from 3D point cloud:
================================================================
```



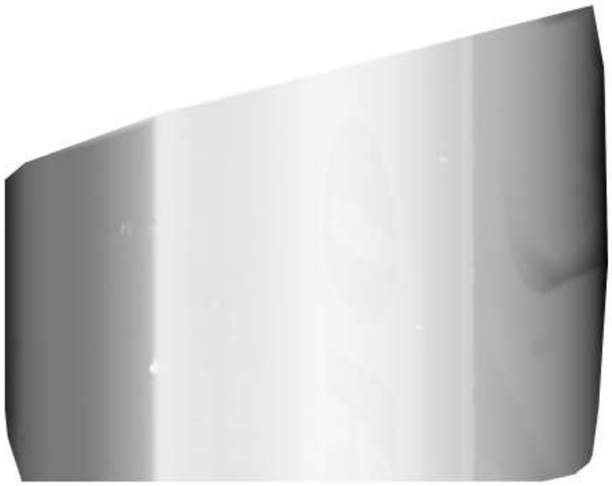




Step 6: Project the constructed high-resolution 3D surface to high-resolution 2D image:

```
%==========================================================================
fprintf(1, '================================================================\n');
fprintf(1, 'Step 6: Project the constructed high-resolution 3D surface to high-resolution 2D image:\n');
fprintf(1, '================================================================\n');
%--------------------------------------------------------------------------
% 2D top-down view grayscale visualization using surf() functionality:
%--------------------------------------------------------------------------
% create a new figure
h50 = figure(50);
% set the grayscale colormap
colormap(gray(256))
% constrcu the surface
surf(xq,zq,yq,'FaceColor','interp',...
   'EdgeColor','none',...
   'FaceLighting','gouraud')
% set axes properties
axis tight
xlabel('X');
ylabel('Z');
zlabel('Y');
%--------------------------------------------------------------------------
% display and save the plot in a 2-D view:
%--------------------------------------------------------------------------
% set the figure view to top-down (xx-plane in our case)
view(2)
% set figure in landscape orientation
% orient landscape
% remove the axes
axis off
% title('Constructed High-Resolution 2D Image');
% save figure
saveas(h50,strcat([output_directory, 'projected_high_resolution_2D_image.png']));

%==========================================================================
```

```
================================================================
Step 6: Project the constructed high-resolution 3D surface to high-resolution 2D image:
================================================================
```

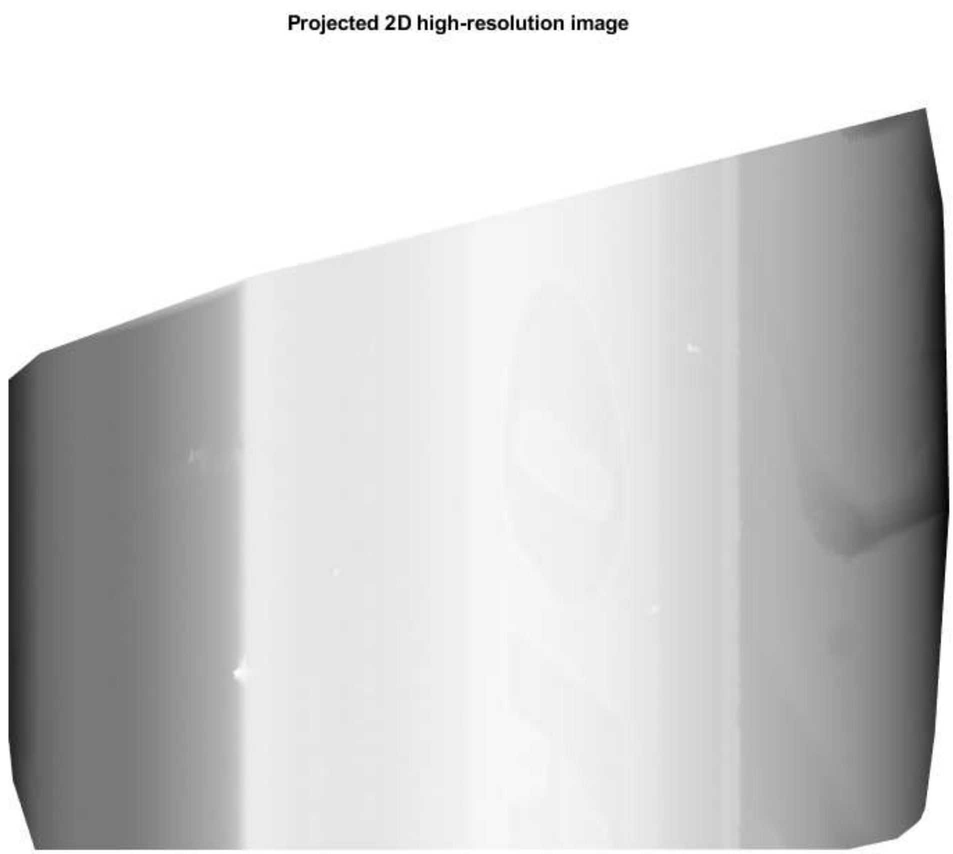


Step 7: Enhance the projected high-resolution 2D image:

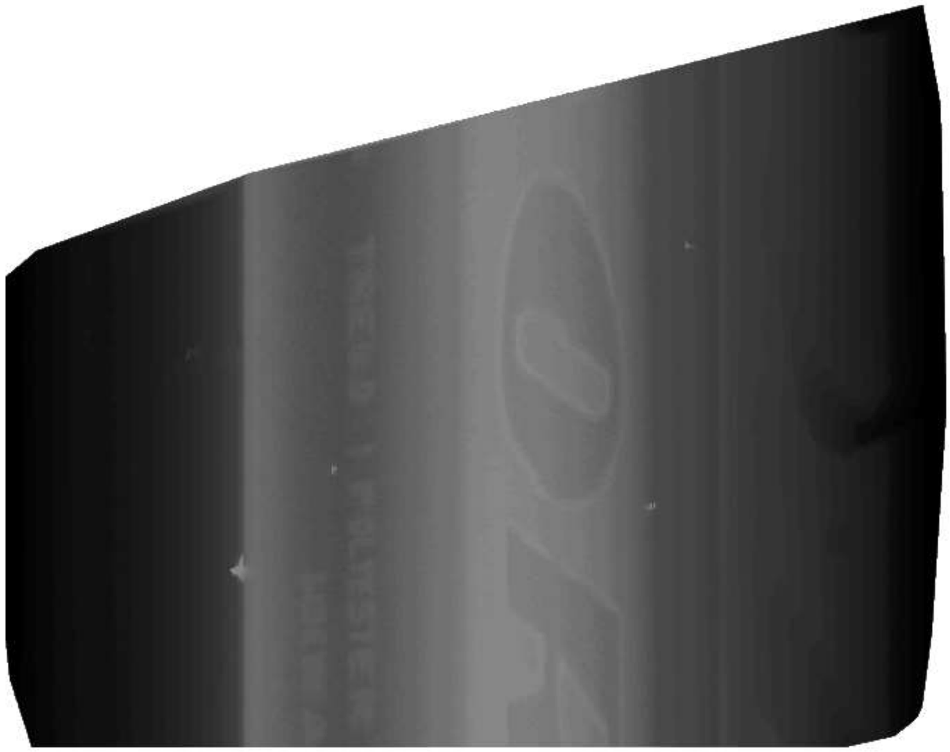
```
%==========================================================================
fprintf(1, '==================================================================\n');
fprintf(1, 'Step 7: Enhance the projected high-resolution 2D image:\n');
fprintf(1, '==================================================================\n');
%--------------------------------------------------------------------------
% - Apply Histogram Equalization
% - APply image sharpening
%--------------------------------------------------------------------------
% read the 2D projection high-resolution image:
%--------------------------------------------------------------------------
% read the image
img = imread(strcat([output_directory, 'projected_high_resolution_2D_image.png']));
% display the original image
h55 = figure(55);
% display the image
imshow(img);
% set the figure title
title('Projected 2D high-resolution image');
% save the figure
saveas(h55, strcat([output_directory, 'projected_high_resolution_2D_image.png']));
%--------------------------------------------------------------------------
% 7.1) Apply simple image enhancement in the form of histogram equalization:
%--------------------------------------------------------------------------
% >> help histeq
%--------------------------------------------------------------------------
%  histeq Enhance contrast using histogram equalization.
%     histeq enhances the contrast of images by transforming the values in an
%     intensity image, or the values in the colormap of an indexed image, so
%     that the histogram of the output image approximately matches a specified
%     histogram.
%--------------------------------------------------------------------------
% apply histogram equalization
img_histeq = histeq(img);
% display the original and enhanced images
h60 = figure(60);
% display the enhanced image
imshow(img_histeq);
% set the figure title
title('After histogram equalization');
% save the figure
saveas(h60,strcat([output_directory, 'projected_high_resolution_2D_image_histeq.png']));

%--------------------------------------------------------------------------
% 7.2) Apply simple image enhancement in the form of image sharpening:
%--------------------------------------------------------------------------
% >> help imsharpen
%--------------------------------------------------------------------------
%  imsharpen Sharpen image using unsharp masking.
%     B = imsharpen(A) returns an enhanced version of the grayscale or
%     truecolor input image A where the image features, such as edges, have
%     been sharpened using the unsharp masking method.
%--------------------------------------------------------------------------
% apply image sharpening
img_histeq_sharp = imsharpen(img_histeq,'Radius',3,'Amount',2);
% display the original and enhanced images
h70 = figure(70);
% display the enhanced image
imshow(img_histeq_sharp);
```

```
% set the figure title
title('After image sharpening');
% save the figure
saveas(h70,strcat([output_directory, 'projected_high_resolution_2D_image_histeq_imsharpen.png']));
% display a message confirming successful execution
fclose('all');
fprintf(1, '=============================================================\n');
fprintf(1, 'The program execution completed successfully...\n');
finish_time = toc(start_time);
fprintf(1, 'Execution time = %s\n', format_time(finish_time));
fprintf(1, '=============================================================\n');
```

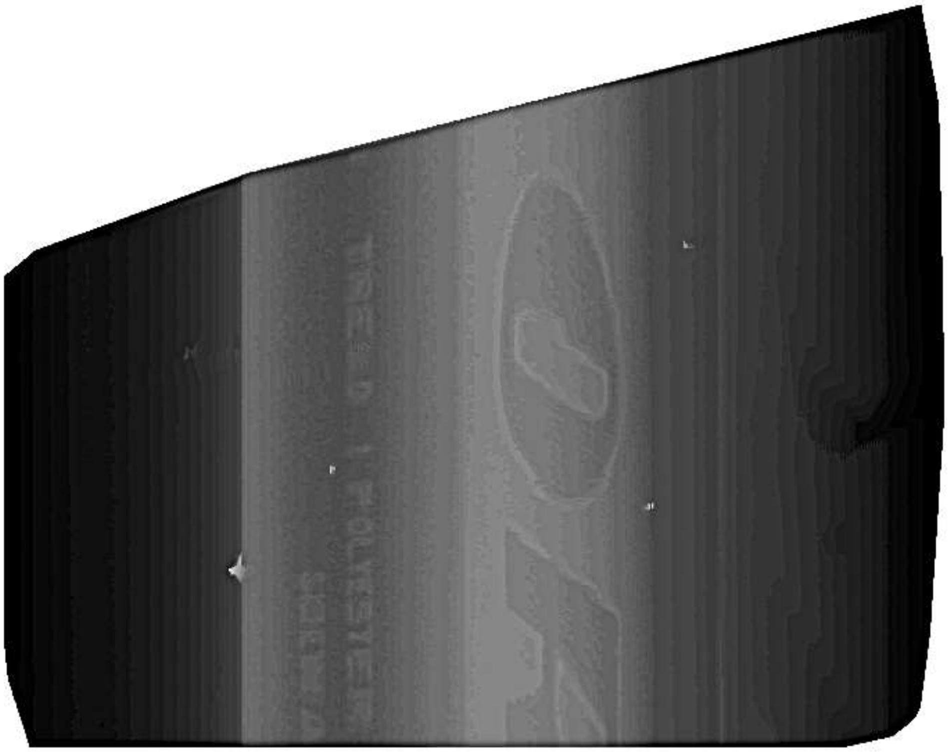
```
================================================================
Step 7: Enhance the projected high-resolution 2D image:
================================================================
===============================================================
The program execution completed successfully...
Execution time = 58.5 secs
===============================================================
```



After histogram equalization



After image sharpening



```
end

function [timeString] = format_time(timeInSecs)
%==========================================================================
% File: format_time.m
% Author: Mohsen Ghazel
% Date: January 21st, 2021
%==========================================================================
% Specifications:
%==========================================================================
% - This function converts and formats the input time from seconds to
%   hours, minutes, seconds
% - This utility function used to format the execution time of the program
%==========================================================================
% Intput:
%==========================================================================
% - timeInSecs: time in seconds
%==========================================================================
% Output:
%==========================================================================
%  - timeString: time formatted in hours, minutes, seconds
%--------------------------------------------------------------------------
% Execution:
%
% >> [timeString] = format_time(timeInSecs)
%
%==========================================================================
% History
%==========================================================================
% Date                      Changes
%--------------------------------------------------------------------------
% March 10th, 2021        Initial definition
%==========================================================================
% License
%==========================================================================
% MIT License: Free to copy, use, modify, share and redistribute.
% Copyright (c) 2021 mghazel2020
%==========================================================================
%--------------------------------------------------------------------------
```

```
% Step 1: Initialize the ouput variables
%-------------------------------------------------------------------------
timeString = '';

%-------------------------------------------------------------------------
% Step 2: Initialize local variables
%-------------------------------------------------------------------------
% number of hours
numHours = 0;

% number of minutes
numMins = 0;

%-------------------------------------------------------------------------
% Step 3: Format the time from seconds to  hours, minutes, seconds
%-------------------------------------------------------------------------
% 3.1) check if input time is longer than 3600 seconds (1 hours)
%-------------------------------------------------------------------------
if ( timeInSecs >= 3600 ) % if time is more than one hour
    % compute the number of hours (integer division)
    numHours = floor(timeInSecs/3600);
    % if more then 1 hour, then plural (hours)
    if ( numHours > 1 )
        hourString = ' hours, ';
    else % otherwise, then singular (hour)
        hourString = ' hour, ';
    end
    % the time string
    timeString = [num2str(numHours ) hourString];
end
%-------------------------------------------------------------------------
% 3.2) check if input time is longer than 60 seconds (1 minute)
%-------------------------------------------------------------------------
if ( timeInSecs >= 60 ) % if time is more than one minute
    % number of minutes
    numMins = floor((timeInSecs - 3600*numHours)/60);
    if numMins > 1
        minuteString = ' mins, ';
    else
        minuteString = ' min, ';
    end
    timeString = [timeString num2str(numMins) minuteString];
end

%-------------------------------------------------------------------------
% 3.3) the remaining number of seconds
%-------------------------------------------------------------------------
numSecs = timeInSecs - 3600*numHours - 60*numMins;

%-------------------------------------------------------------------------
% 3.4) the final formatted time string
%-------------------------------------------------------------------------
timeString = [timeString sprintf('%2.1f', numSecs) ' secs'];

% return
return;

end
```

```
ans =

     1
```