```matlab
%Analysis of Capillary Bridges by Lion Rainer, 06 2024
%Paris Lodron University Salzburg

%This program is part of my Bachelor Thesis work. Its purpose is to make
%visualizations of curvature data of a surface. The surface stems from
%preosteoblast tissue (MC3T3-E1 cell line) grown on PDMS capillary bridges
%of a size in the order of a 2x2x2mm cube. The imported data are that of
%the surface as an .stl file and that of the curvature as an excel file.
%There is a similar program that viszualizes Surface Evolver data which is
%compared to the sample data in the Thesis. This is because the units and
%the file types of the Surface Evolver data are different.

clear all
close all

%This prompt asks to define a directory to save the results of visualization
savePath = uigetdir(pwd, 'Select a directory to save your figures');

%This prompt asks to select the surface .stl file and stores its contents
%as a cell named "TR" (triangulation). Its contents include TR.Points, a
%nx3 Matrix of the Point coordinates. In the second entry TR.Connectivity-
% list of the cell TR, the triangles are defined in a nx3 matrix. Each row
% contains three numbers, each number referring to a point - That is why
% the point number and the order of the points matters.
[stlFileName, stlPathName] = uigetfile('*.stl', 'Select the STL file');
if isequal(stlFileName, 0)
    disp('User selected Cancel');
else
    stlFullPath = fullfile(stlPathName, stlFileName);

    TR = stlread(stlFullPath);
end

%This prompt asks to select the Gaussian Curvature (GC) data. Its contents
%are stored as "GC0" which is a nx1 matrix. The curvature value in row i
%corresponds to the triangle i in TR.Connectivitylist. The triangle i
%refers to three points by their numbers in the TR.Points file, the
%coordinates of the row with the point number belong to the same point. The
%number of triangles and the number of points are roughly the same, but
%they do not match. In a later step, the curvatures are assigned to
%coordinates.
[gcFileName, gcPathName] = uigetfile('*.xlsx', 'Select the Gaussian Curvature (GC) .xlsx file');
if isequal(gcFileName, 0)
    disp('User selected Cancel');
else
    gcFullPath = fullfile(gcPathName, gcFileName);
    GC0 = readmatrix(gcFullPath);
end

% The same goes for Mean Curvature (MC)
[mcFileName, mcPathName] = uigetfile('*.xlsx', 'Select the Mean Curvature (MC) .xlsx file');
if isequal(mcFileName, 0)
    disp('User selected Cancel');
else
    mcFullPath = fullfile(mcPathName, mcFileName);
    MC0 = readmatrix(mcFullPath);
```

```matlab
end

%Triangles and corresponding curvatures are stored together:
Connectivitylist = [TR.ConnectivityList MC0 GC0];

%The Mean and Gaussian Curvatures and Point coordinates will be assigned later on in this
%matrix in column 4 and 5:
PointsCurvatures = [TR.Points,zeros(size(TR.Points,1),2)];

%This loop assigns Curvatures to Points, which is needed for visualization:
for i = 1:size(PointsCurvatures,1)
    %search for all indices of Connectivitylist where Point i is in the
    %Connectivitylist
    Pointsinlist = find(Connectivitylist(:,1) == i | Connectivitylist(:,2) == i | Connectivitylist(:,3) == i );
    %Get first three of the indices. This means, Point i is part of these
    %three triangles, all of which have a MC and GC value. Taking only
    %three and not all was a decision which speeds up computation without
    %compromising the results.
    Selectedpoints = Pointsinlist(1:3);
    %At these Points, get the Curvature Values which are in the 4th and 5th
    %column
    Curvvalues = Connectivitylist(Selectedpoints,4:5);
    %Calculate the mean Value of each Curvature (MC and GC). This means,
    %around one Point P_i exist several triangles, of which P_i is a part
    %of. These triangles have Curvature values assigned to them, which are
    %averaged and then assigned to P_i.
    Meancurvvalues = [mean(Curvvalues(:,1)) mean(Curvvalues(:,2))];
    %Enter those values to the corresponding Point Coordinates
    PointsCurvatures(i,4:5) = Meancurvvalues;
end

    %Previous attempts included calculating the Vertex curvatures and assigning
    %the values directly since there are almost equally many points and
    %vertices, but the order of the Points and vertices are not the same...
    % PointsCurvatures = [double(TR.Points), MC0, GC0];

%The next step uses the function "reposition" to place the Origin [0 0 0]
%in the center of the CB (z being the axis of rotational "symmetry" and z=0
%being where the diameter of the CB is smallest, which is called the neck.
%This proved difficult because rotational symmetry is not perfect, circular
%circumference is not perfect and often times the CB is tilted in respect
%to z. The origin is generally placed too high, the neck is almost always
%at z < -100. However, in the figures made from the "perfect" Surface
%Evolver these mistakes did not occur, the center was always at the neck...
%The result of "reposition" is the Pointcloud (nx3 matrix containing x y z
%coordinates) and diameters, which is a nx4 matrix with center coordinates
%of the circular sections and the radius value in the 4th column
n = 50;
[Pointcloud,diameters] = reposition(PointsCurvatures(:,1:3),n);
x = Pointcloud(:,1);
y = Pointcloud(:,2);
z = Pointcloud(:,3);

%The next step uses the function "Preview_centers" to display the
%Pointcloud and the diameters coordinates, and a prompt asks to define the
%region of interest. This manual task is needed because every sample is
```

```
%different and might have faulty data, especially towards the endcaps where
%holes or curvature spikes occur. With this, the two parameters botval and
%topval are defined, which are used a lot later on.
Preview_centers(x,y,z,diameters,round((max(z)-min(z))/40));
    prompt = {'Z-Value of upper End of Neck Region:' , 'Z-Value of lower End of Neck Region:'};
    dlgtitle = 'Input';
    dims = [1 35];
    definput = {'', ''};
    %Input dialog
    answer = inputdlg(prompt, dlgtitle, dims, definput);
    % Check if the dialog is not empty
    if ~isempty(answer)
        topval = str2double(answer{1});
        botval = str2double(answer{2});
        if isnan(topval) || isnan(botval)
            disp('Invalid input. Please enter numeric values.');
        else
            disp(['Z-Value of upper End of Neck Region: ' , num2str(topval)]);
            disp(['Z-Value of lower End of Neck Region: ' , num2str(botval)]);
        end
    else
        disp('Dialog closed without input.');
    end

neckind = diameters(:,4) >= botval & diameters(:,4) <= topval;
neck = diameters(neckind,:);

    %Previously used script for rotating the Pointcloud so the rotational Axis
    %is aligned with the z-axis. This is best done in Amira beforehand, the
    %manual results are generally better. A better script might help here.
    % [SymPC,Symdiam] = symmetry(Pointcloud,neck);
    % xrot = SymPC(:,1);
    % yrot = SymPC(:,2);
    % zrot = SymPC(:,3);


%The function "makepolar" is used to get Polar Coordinates for
%Visualization. Mpolar = [z, r, theta]. For visualization, r is not used.
%MC in the 4th, GC in the 5th column.
Mpolar = makepolar(Pointcloud);
Mdatapolar = [Mpolar PointsCurvatures(:,4:5)];
Mdatacart = [x y z PointsCurvatures(:,4:5)];

%The limits for the visualization of the curvatures. These remain the same
%to be able to easily compare samples and plots with eachother. The
%Gaussian curvature is several orders of magnitude smaller than the mean
%curvature because it is squared (K1*K2), its unit is µm^-2 opposed to MC
%unit µm^-1.
MC_min = -2e-3;
MC_max = 2e-3;
GC_min = -3e-5;
GC_max = 3e-5;

%The following section uses the visualisation function "Visualizesection" which requires inputs(1: (nx5 Matrix with z,r,Theta,MC,GC), 2: z-range around
%center (+- tolerance), 3: Number of Slices, 4: Version). There are 4
%Version options which make different plots. The Number of slices are used
```

```
%in two of the plots to slice the data into a number of sections with
%similar z value. Each Version, when executed, makes two plots, one with MC
%and one with GC.

% 4 displays a 3D surface made of [z Theta Curvature]. This is essentially
% a "Central Cylindrical Projection" which ignores r to map 3D geometric
% properties to 2D and then adds the parameter of curvature as the new z
% component. To view this new surface in 2D, it is colored according to MC
% or GC and viewed in -z direction (from above). The result is a rectangle
% colored according to curvature, which gives a good overview of curvature
% distribution and trends, while curvature itself is not really
% quantifiable and only displayed through color.
Visualizesection(Mdatapolar,botval,topval,0,4);

% Version 2 makes 3D Pointclouds with GC and MC Coloring, giving the most
% direct representation of the data. This figure can be compared to the
% visualization in Amira to see if the sample is represented accurately.
% The third entry refers to the point size rather than the number of
% slices. Cartesian data is used insted of polar.
Visualizesection(Mdatacart,botval,topval,70,2);

% Version 1 makes n slices normal to z to get n+1 sections. Of this, n+1
% graphs are made in one plot, displaying MC(Theta) or GC(Theta). The
% graphs are of different color to differentiate height sections. Tis plot
% shows all datapoints, no averages. It is used to see deviation along the
% circumference angle Theta. In future work, a better representation will
% be needed, and one conclusion of seeing the results of this visualization
% was to make less plots and focus on a smaller area around the neck
% because deviation becomes higher towards the upper and lower end. Maybe
% limits of 0.3*botval and 0.3* topval and only 3 or 4 plots would help...
Visualizesection(Mdatapolar,botval,topval,10,1);

% Version 3 takes the polar data and separates it into n sections of equal
% deltaz. In these sections, the Curvature values are averaged and a 2D
% graph is plotted: MC(z) or GC(z). For each section, the min, mean and max
% value are plotted.
Visualizesection(Mdatapolar,botval,topval,50,3);

%Visualization script, shows gray Pointcloud and colored centers of
%sections, which represent the "middle axis", colored by diameter size.
%This figure is not used in the Results, but with better code might be
%useful to see asymmetry in the center data.
Visualizecenters(x,y,z,diameters,botval,topval);


%In this last step, all figures are saved to the directory chosen in the
%beginning. The names have been predefined in an excel namelist. This is
%possible because they are generated in the order they appear on the
%Namelist.
NameList = readcell('NameList.xlsx');
% Get a list of all open figures
figHandles = findall(groot, 'Type', 'figure');

% Loop through each figure and save it
for i = 1:length(figHandles)
    fig = figHandles(i);

    % Define the file name
```

```matlab
    fileName = NameList{i};

    % Full file path
    filePath = fullfile(savePath, fileName);

    % Save the figure
    saveas(fig, filePath, 'fig');
end

disp('All figures have been saved successfully.');

```