ULSAN NATIONAL INSTITUTE OF
SCIENCE AND TECHNOLOGY

Assignment #3

Figuera Michal
Xiyana Veroska
20225398

2023-05-07

Task 1-1

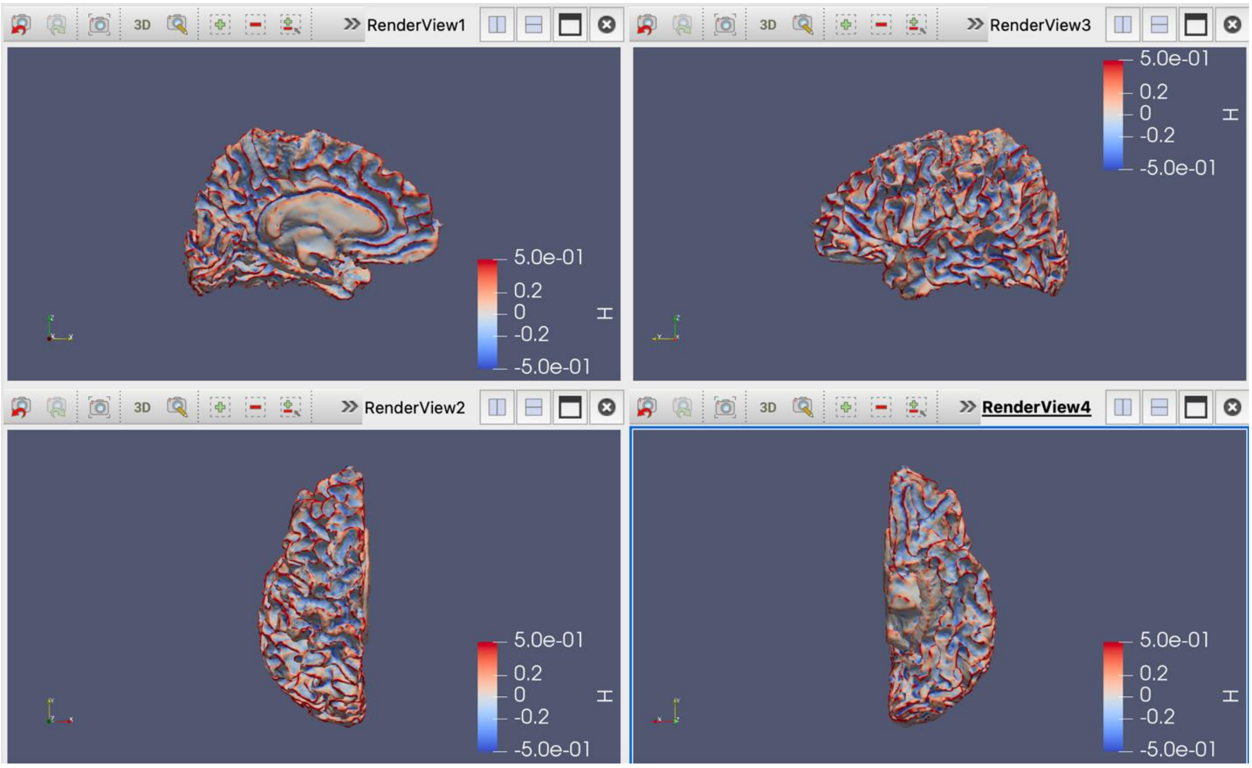
Explanation

The input mesh lh.white is loaded and also its mean curvature H. Then we overlay H onto the lh.white using the write_property function.

Code

```
clc;
clear;

[v, f] = read_vtk('/Users/xiyana/Downloads/med-course/homeworks/hw3/input_data/lh.white.vtk');
H = load('/Users/xiyana/Downloads/med-course/homeworks/hw3/input_data/lh.white.H.txt');

write_property('/Users/xiyana/Downloads/med-course/homeworks/hw3/created/lh.white.H.vtk', v, f, struct('H', H));
```

Result



Task 1-2

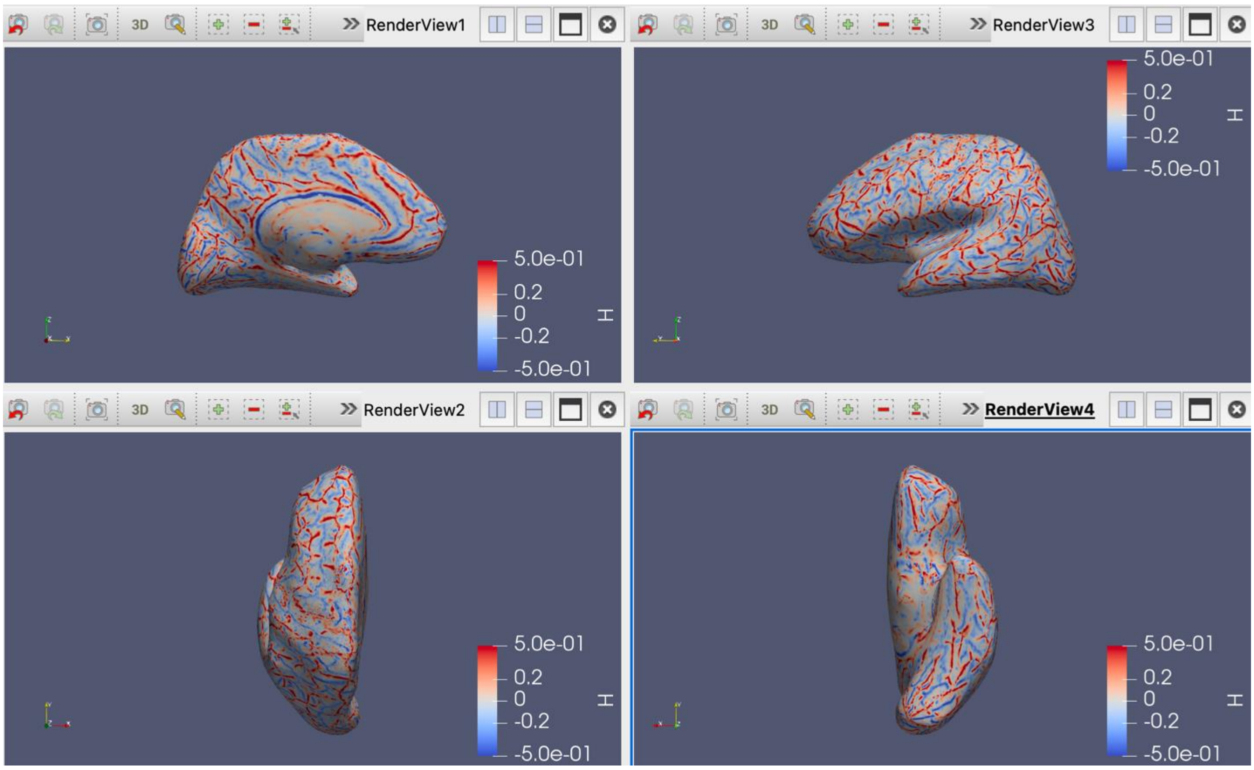
Explanation

First, we inflate lh.white using mris_inflate of freeview to obtain lf.inflate. And then lf.inflate is loaded and also the mean curvature H of the original lh.white. Then we overlay H onto the lf.inflate using the write_property function.

Code

```
clear
clc

[v, f] = read_vtk('/Users/xiyana/Downloads/med-course/homeworks/hw3/created/lh.inflate.vtk');
H = load('/Users/xiyana/Downloads/med-course/homeworks/hw3/input_data/lh.white.H.txt');

write_property('/Users/xiyana/Downloads/med-course/homeworks/hw3/created/lh.inflated.H.vtk', v, f, struct('H', H));
```

Result



Task 2-1

Explanation

In task 2-1, we first generate a sparse adjacency matrix based on the original mesh file (lh.white.vtk) by using the same function from last assignment but modifying it to convert input to matlab indexing by adding 1. Then we calculate the length of each edge in the mesh using Euclidean distance and calculate the Gaussian weight by using Gaussian distribution with a mean of 0 mm and standard deviation equal to the average of all the edge lengths. Then we update the weights in the sparse adjacency matrix according to the calculated Gaussian weights and set the diagonal elements of the matrix to 0. Finally, we normalize each row of the matrix so that the sum of the weights in each row equals 1 to normalize.

Code

```
clear
clc

[v, f] = read_vtk('/Users/xiyana/Downloads/med-course/homeworks/hw3/input_data/lh.white.vtk');

%%%%%%%%%%%%% Task 2-1 %%%%%%%%%%%%%%%%%%%%

% Create sparse matrix
mask_f_matrix = create_sparse_matrix(f);
[row_indices, col_indices, values] = find(mask_f_matrix);

% Compute edge lengths using Euclidean distances
edge_lengths = sqrt(sum((v(row_indices,:) - v(col_indices,:)).^2, 2));

% Compute average length of edge
avg_edge_length = mean(edge_lengths);

% Compute Gaussian weights with mean 0 and sd as average length of edge
gaussian_weights = exp(-(edge_lengths.^2)/(2 * avg_edge_length^2))/(avg_edge_length*sqrt(2*pi));


% Update weights of the sparse adjacency matrix
mask_f_matrix(sub2ind(size(mask_f_matrix), row_indices, col_indices)) = gaussian_weights;
mask_f_matrix(sub2ind(size(mask_f_matrix), col_indices, row_indices)) = gaussian_weights;

% Set diagonal to 0
W = mask_f_matrix - diag(diag(mask_f_matrix));

% normalize
W = bsxfun(@rdivide, W, sum(W, 2));

%%%% Function for creating the sparse matrix

function [mask_f_matrix] = create_sparse_matrix(f)

    len = size(f, 1);
    f_list = zeros(len * 6, 2);

    for i=1:len

        vtx = f(i, :);
        f_list((i-1)*6+1, :) = [vtx(1), vtx(2)];
        f_list((i-1)*6+2, :) = [vtx(2), vtx(3)];
        f_list((i-1)*6+3, :) = [vtx(3), vtx(1)];
        f_list((i-1)*6+4, :) = [vtx(2), vtx(1)];
        f_list((i-1)*6+5, :) = [vtx(3), vtx(2)];
        f_list((i-1)*6+6, :) = [vtx(1), vtx(3)];
    end

    n = max(f_list, [], 'all') + 1;  % convert to matlab one-based indexing
    mask_f_matrix = sparse(f_list(:,1) + 1, f_list(:,2) + 1, 1, n, n);  % convert to matlab one-based indexing

end
```

Task 2-2

Explanation

In task 2-2, First we will perform everything from task 2-1. Then with the updated sparse matrix we obtained which we are calling w, we will apply smoothing by multiplying this updated mesh with the mean curvature H, then we will update do this for 10, 20 and 40 iterations obtaining H_10, H_20 and H_40. Then we overlay the updated mean curvatures on lf.inflated.

Code

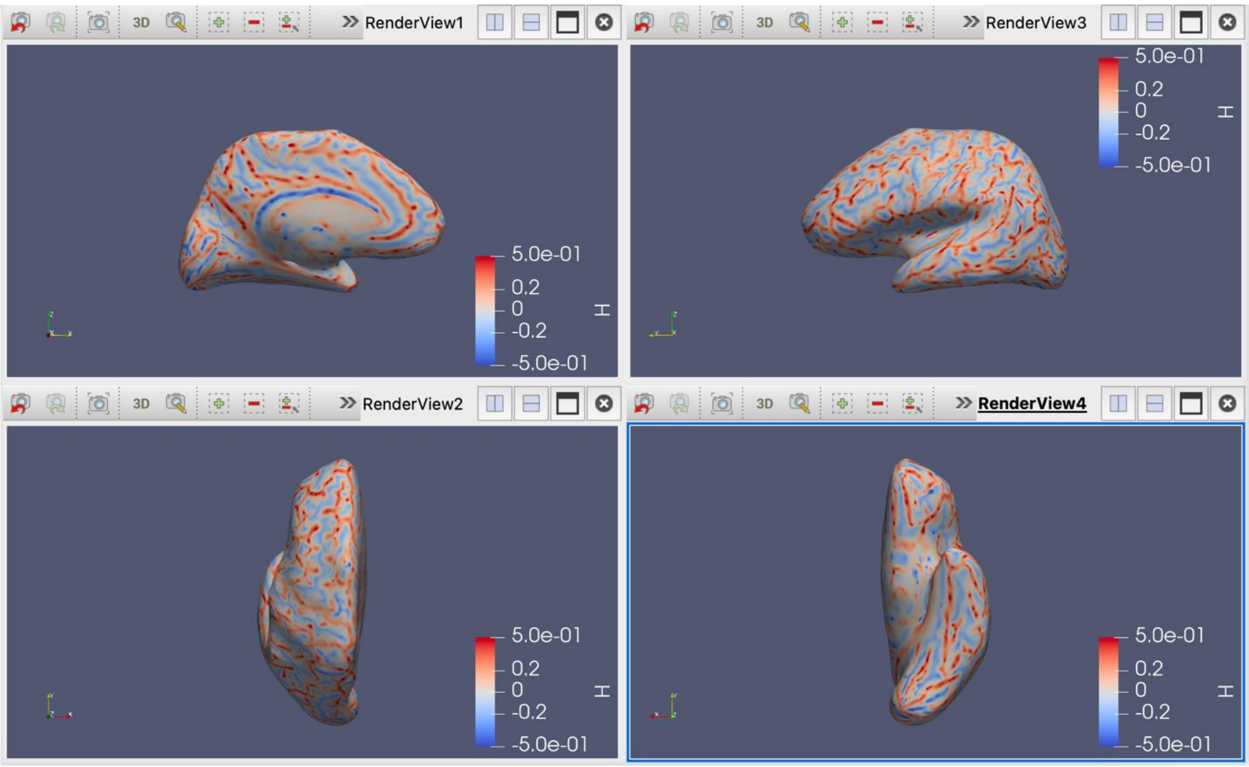
```
clear
clc

[v, f] = read_vtk('/Users/xiyana/Downloads/med-course/homeworks/hw3/input_data/lh.white.vtk');
[v_inflated, f_inflated] = read_vtk('/Users/xiyana/Downloads/med-course/homeworks/hw3/created/lh.inflate.vtk');
H = load('/Users/xiyana/Downloads/med-course/homeworks/hw3/input_data/lh.white.H.txt');


%%%%%%%%%%%%% Task 2-1 %%%%%%%%%%%%%%%%%%%%

% Create sparse matrix
mask_f_matrix = create_sparse_matrix(f);
[row_indices, col_indices, values] = find(mask_f_matrix);

% Compute edge lengths using Euclidean distances
edge_lengths = sqrt(sum((v(row_indices,:) - v(col_indices,:)).^2, 2));

% Compute average length of edge
avg_edge_length = mean(edge_lengths);

% Update weights of the sparse adjacency matrix
mask_f_matrix(sub2ind(size(mask_f_matrix), row_indices, col_indices)) = gaussian_weights;
mask_f_matrix(sub2ind(size(mask_f_matrix), col_indices, row_indices)) = gaussian_weights;

% Set diagonal to 0
W = mask_f_matrix - diag(diag(mask_f_matrix));

% normalize
W = bsxfun(@rdivide, W, sum(W, 2));

%%%%%%%%%%%%% Task 2-3 %%%%%%%%%%%%%%%%%%%%

% Smooth geometric data for 10, 20, and 40 iterations

v_original = v;
num_iterations = [10, 20, 40];

for j=1:num_iterations(1)
    v = W * v;
end
v_10 = v;
fprintf('Smoothing has been completed for %d iterations\n', j);
write_property('/Users/xiyana/Downloads/med-course/homeworks/hw3/created/lh.white.v10.vtk', v, f, struct('v', v_10));


for j=num_iterations(1):num_iterations(2)
    v = W * v;
end
v_20 = v;
fprintf('Smoothing has been completed for %d iterations\n', j);
write_property('/Users/xiyana/Downloads/med-course/homeworks/hw3/created/lh.white.v20.vtk', v, f, struct('v', v_20));
```

```
    struct('H', H_20));


for j=num_iterations(2):num_iterations(3)
    H = W * H;
end
H_40 = H;
fprintf('Smoothing has been completed for %d iterations\n', j);
write_property('/Users/xiyana/Downloads/med-course/homeworks/hw3/created/lh.inflated.H40.vtk', v_inflated, f_inflated, ...
    struct('H', H_40));
```

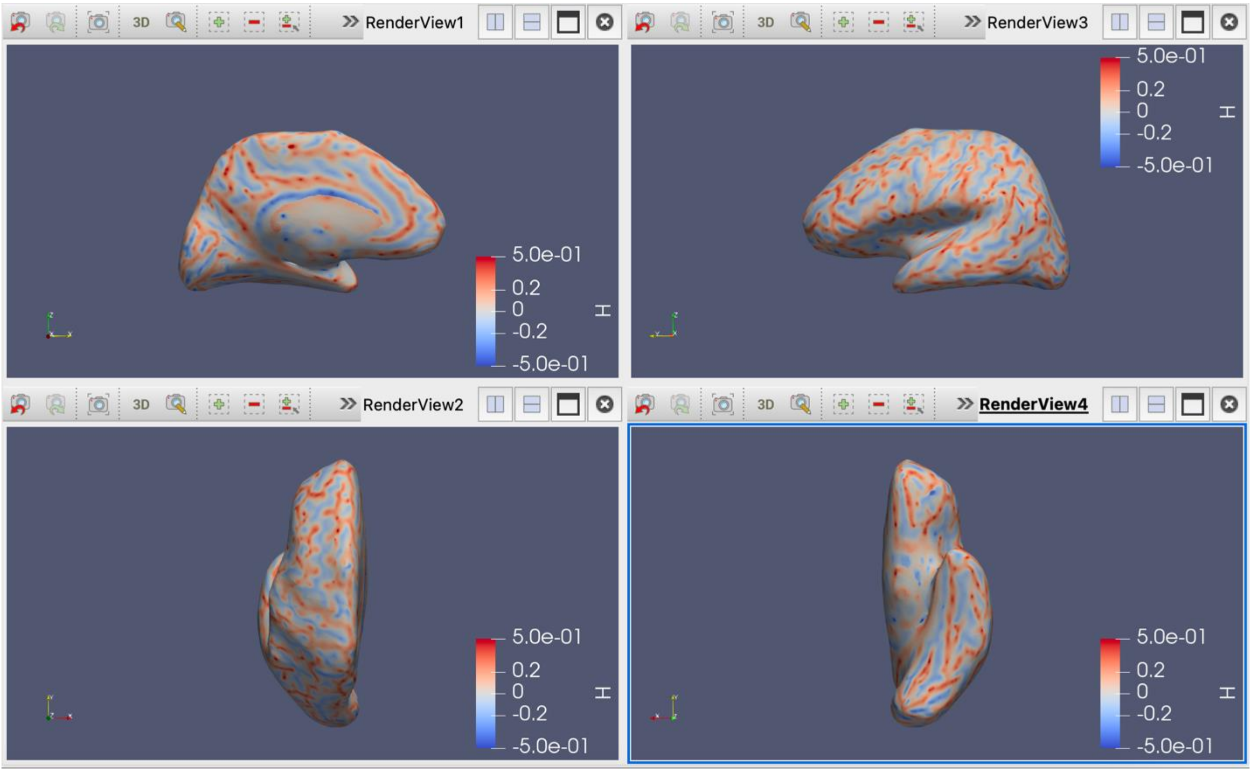
```
%%%% Function for creating the sparse matrix

function [mask_f_matrix] = create_sparse_matrix(f)

    len = size(f, 1);
    f_list = zeros(len * 6, 2);

    for i=1:len

        vtx = f(i, :);
        f_list((i-1)*6+1, :) = [vtx(1), vtx(2)];
        f_list((i-1)*6+2, :) = [vtx(2), vtx(3)];
        f_list((i-1)*6+3, :) = [vtx(3), vtx(1)];
        f_list((i-1)*6+4, :) = [vtx(2), vtx(1)];
        f_list((i-1)*6+5, :) = [vtx(3), vtx(2)];
        f_list((i-1)*6+6, :) = [vtx(1), vtx(3)];
    end

    n = max(f_list, [], 'all') + 1;  % convert to matlab one-based indexing
    mask_f_matrix = sparse(f_list(:,1) + 1, f_list(:,2) + 1, 1, n, n);  % convert to matlab one-based indexing

end
```

Results

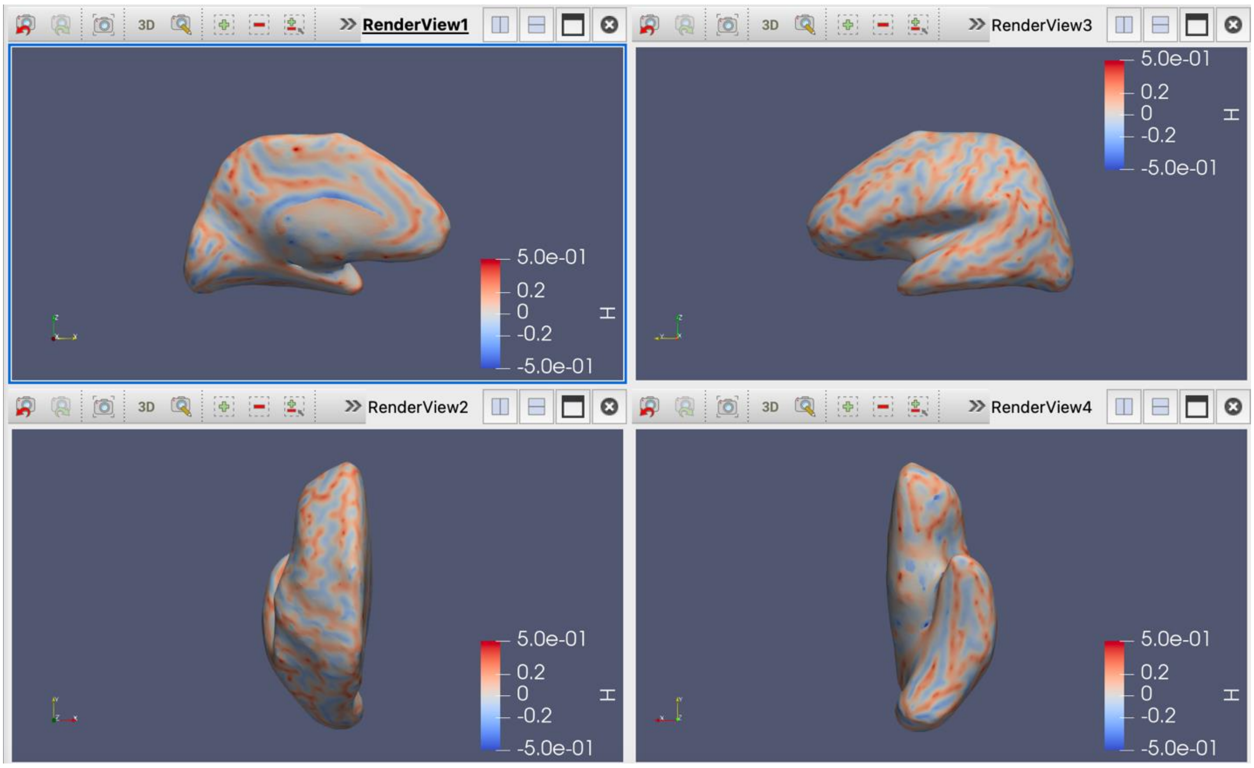
10 Iterations



20 iterations



40 iterations



Task 2-3

Explanation

In task 2-3, First we will perform everything from task 1-1. Then with the updated sparse matrix we obtained which we are calling w, we will apply smoothing by multiplying this updated mesh with the original mesh vertices, then we will update do this for 10, 20 and 40 iterations obtaining v_10, v_20 and v_40. Then we overlay the updated mesh vertices on lf.white.

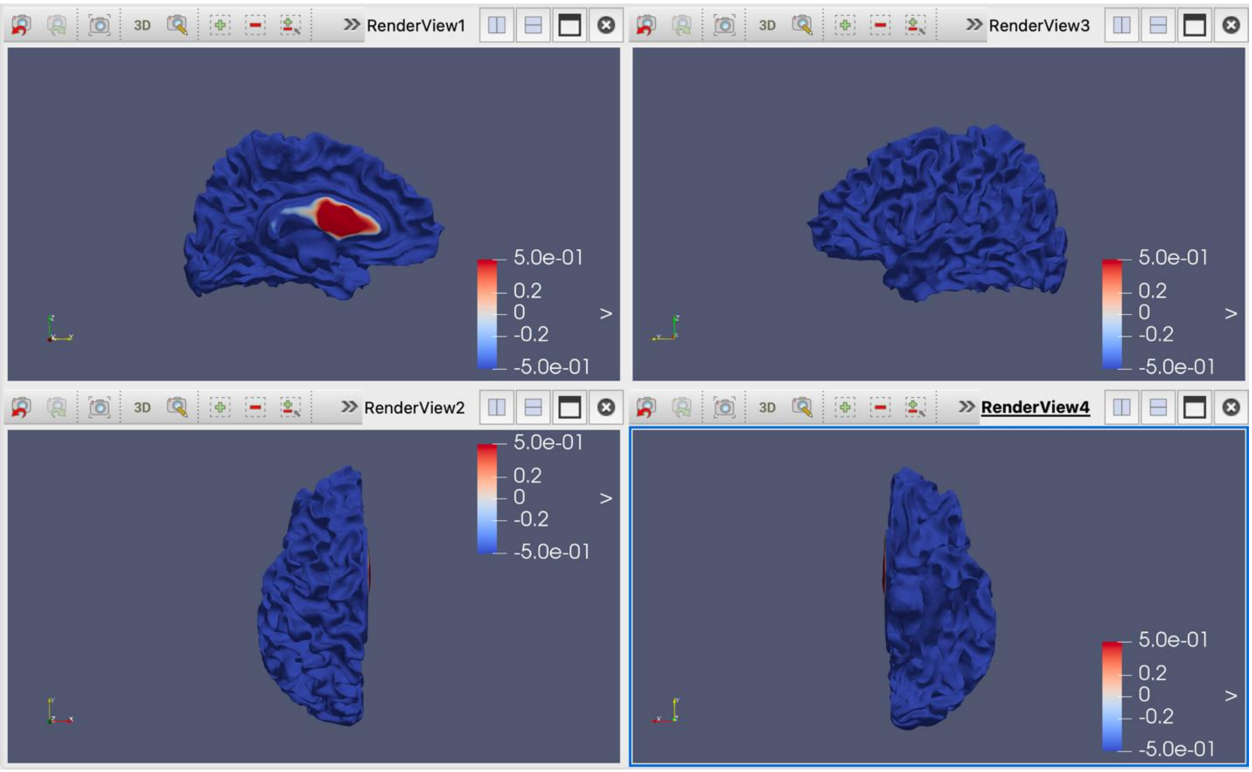
Code

```
clear
clc

[v, f] = read_vtk('/Users/xiyana/Downloads/med-course/homeworks/hw3/input_data/lh.white.vtk');

%%%%%%%%%%%%% Task 2-1 %%%%%%%%%%%%%%%%%%%%

% Create sparse matrix
mask_f_matrix = create_sparse_matrix(f);
[row_indices, col_indices, values] = find(mask_f_matrix);

% Compute edge lengths using Euclidean distances
edge_lengths = sqrt(sum((v(row_indices,:) - v(col_indices,:)).^2, 2));

% Compute average length of edge
avg_edge_length = mean(edge_lengths);

% Compute Gaussian weights with mean 0 and sd as average length of edge
gaussian_weights = exp(-(edge_lengths.^2)/(2 * avg_edge_length^2))/(avg_edge_length*sqrt(2*pi));


% Update weights of the sparse adjacency matrix
mask_f_matrix(sub2ind(size(mask_f_matrix), row_indices, col_indices)) = gaussian_weights;
mask_f_matrix(sub2ind(size(mask_f_matrix), col_indices, row_indices)) = gaussian_weights;

% Set diagonal to 0
W = mask_f_matrix - diag(diag(mask_f_matrix));

% normalize
W = bsxfun(@rdivide, W, sum(W, 2));

%%%%%%%%%%%%% Task 2-3 %%%%%%%%%%%%%%%%%%%%

% Apply smoothing for 10, 20, and 40 iterations

v_original = v;
num_iterations = [10, 20, 40];

for j=1:num_iterations(1)
    v = W * v;
end
v_10 = v;
fprintf('Smoothing has been completed for %d iterations\n', j);
write_property('/Users/xiyana/Downloads/med-course/homeworks/hw3/created/lh.white.v10.vtk', v, f, struct('v', v_10));


for j=num_iterations(1):num_iterations(2)
    v = W * v;
end
v_20 = v;
fprintf('Smoothing has been completed for %d iterations\n', j);
write_property('/Users/xiyana/Downloads/med-course/homeworks/hw3/created/lh.white.v20.vtk', v, f, struct('v', v_20));

for j=num_iterations(2):num_iterations(3)
    v = W * v;
end
v_40 = v;
fprintf('Smoothing has been completed for %d iterations\n', j);
write_property('/Users/xiyana/Downloads/med-course/homeworks/hw3/created/lh.white.v40.vtk', v, f, struct('v', v_40));
```

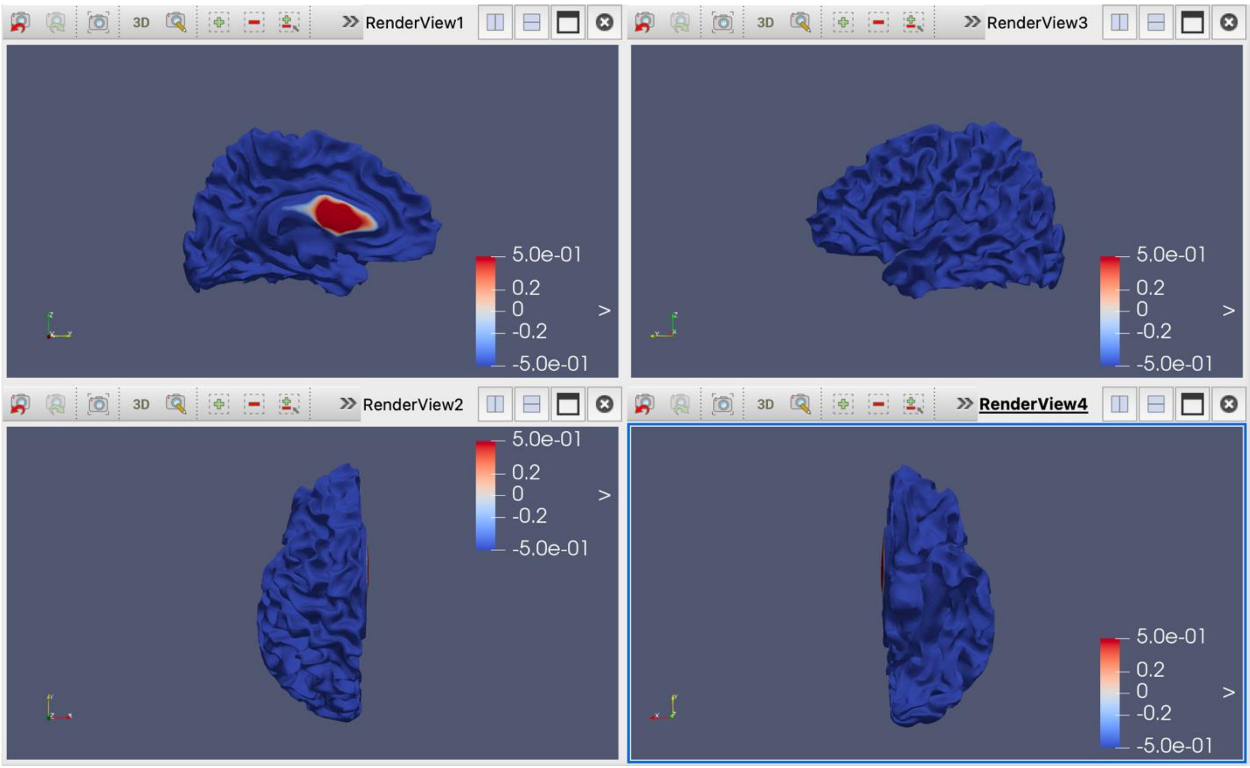
```matlab
%%%% Function for creating the sparse matrix

function [mask_f_matrix] = create_sparse_matrix(f)

    len = size(f, 1);
    f_list = zeros(len * 6, 2);

    for i=1:len

        vtx = f(i, :);
        f_list((i-1)*6+1, :) = [vtx(1), vtx(2)];
        f_list((i-1)*6+2, :) = [vtx(2), vtx(3)];
        f_list((i-1)*6+3, :) = [vtx(3), vtx(1)];
        f_list((i-1)*6+4, :) = [vtx(2), vtx(1)];
        f_list((i-1)*6+5, :) = [vtx(3), vtx(2)];
        f_list((i-1)*6+6, :) = [vtx(1), vtx(3)];
    end

    n = max(f_list, [], 'all') + 1;  % convert to matlab one-based indexing
    mask_f_matrix = sparse(f_list(:,1) + 1, f_list(:,2) + 1, 1, n, n);  % convert to matlab one-based indexing

end
```

Results

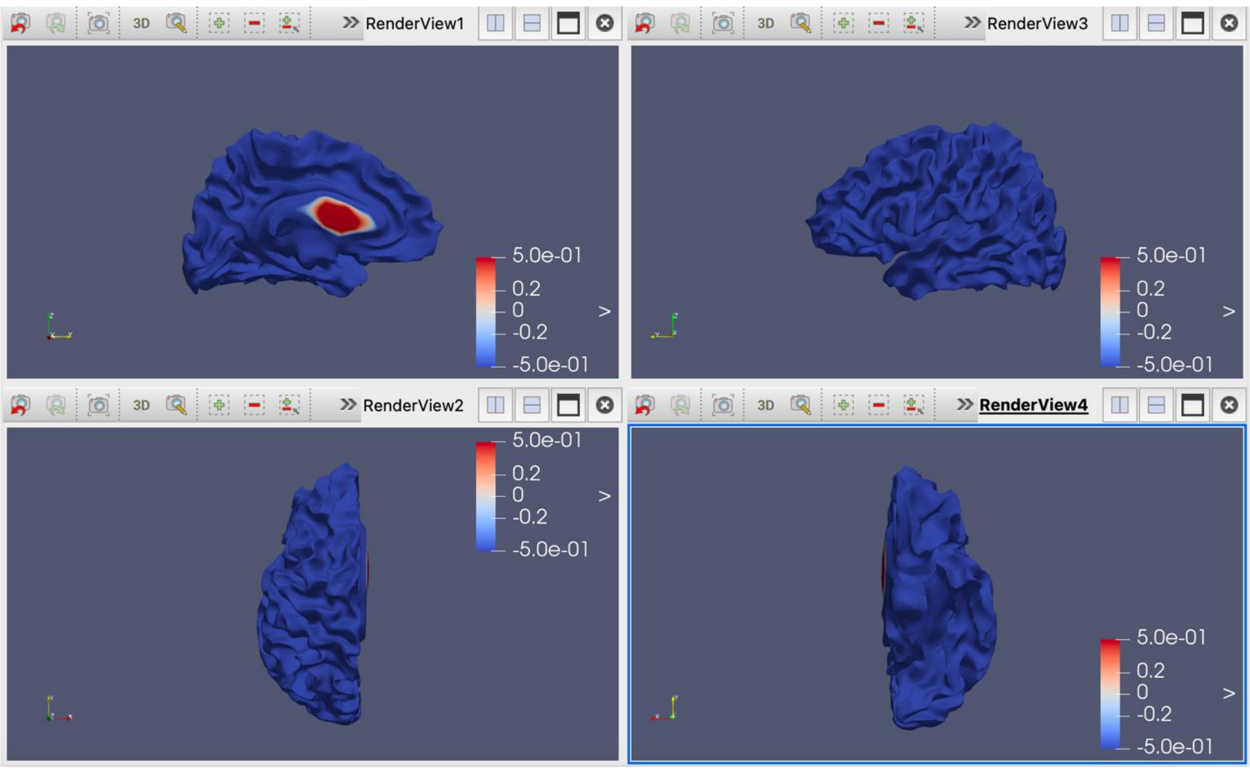
10 Iterations



20 iterations



40 iterations



Discussion

Shrinkage issues are common in data smoothing. There are several techniques to overcome this issue, among them is it possible to use nonparametric regression methods such as kernel regression, as well as weighted mechanisms with a window which size is not fixed but that gives more weight to points that are closer to the point being smoothed. These methods allow to mitigate the effect of extreme values which cause the shrinkage issue.

Task 3-1

Discussion

Suppose that the plane of a triangle ABC, and a point q are inside a sphere, to verify if the point q is inside the triangle ABC, that is, if this point is part of the enclosing triangle ABC or not. We must determine if the barycentric coordinates of q are all non-negative since this would indicate that the point q is inside the triangle ABC in the plane, and therefore would not require any additional rescaling, and there would be no need to project point Q orthogonally onto the plane of the triangle. If any coordinate is negative, this would indicate that the point q is not inside the triangle ABC.

An example, would be the case where we have a triangle with vertices A, B, and C and a point q. Which even though q is in the same plane as triangle ABC, the closest point to q can be outside the triangle, given that the point q is closest to an edge or vertex of the triangle, in which case the closest point is outside the area of the triangle. Therefore, it is not always guaranteed that the closest point to q is in the triangle ABC.

Task 3-2

Explanation

n 3-2, we rescale Q onto the plane of triangle ABC to ensure the correct calculation of barycentric coefficients to determine whether the query point is enclosed by triangle ABC. The rescaling is computed using the normal vector of the triangle plane and projecting the vector from A to Q onto the normal vector. Finally, we rescale q by subtracting the distance.

```
%%%%%%% Task 3-2

function Q_prime = rescalePoint(Q, A, B, C)

    % Compute the normal vector
    N = cross(B-A, C-A);

    % Compute vector from vertex A to point Q and get scale factor
    scale_factor = dot(N, A) / dot(N, Q);

    % Compute projected distance from A to Q using the normal vector N
    Q_prime = Q * scale_factor;
end
```

Task 3-3

```
clear
clc

% Load icosphere for query and sphere for triangles
[v_o, f_o] = read_vtk('/Users/xiyana/Downloads/med-course/homeworks/hw3/input_data/lh.sphere.vtk');
[v, f] = read_vtk('/Users/xiyana/Downloads/med-course/homeworks/hw3/icosphere_mesh/icosphere_4.vtk');

%Initialize tree for knn search
MD = KDTreeSearcher(v_o);

%Initialize triangulation
tr = triangulation(f_o+1, v_o);

% Set Q to be the vertices of icosphere
Q = v;

% Initiliaze matrices and variable to pass to next query
reachy = 1;
triang_id = [];
triang_bary = [];
p_closest = [];
```

```
% while Q is not empty but with a for loop
for i = 1:length(Q)

    % Per query
    q = Q(i,:);

    % Starting from one closest neighbor
    k= 1;

    % While there are still faces to visit keep same k
    while reachy==i

        % Temporal k
        k_temp = k;

        % Knn search, find faces that share the closest vertex p using vertex attachment
        % and find vertices using connectivity list
        p = MD.knnsearch(q, 'k', k, 'distance', 'euclidean');
        p = p.';
        T = tr.vertexAttachments(p);
        face = T(k);
        faces = face{1};

        vertices = tr.ConnectivityList(face{1},:);
```

```
        % For every triangle
        for triangle=1:length(vertices)

            % Rescale query point
            A = tr.Points(vertices(triangle,1),:);
            B = tr.Points(vertices(triangle,2),:);
            C = tr.Points(vertices(triangle,3),:);

            q_scaled = rescalePoint(q,A,B,C);

            % Obtain barycentric coefficients
            bc_coords = cartesianToBarycentric(tr, faces(triangle),q_scaled);

            % When all faces are checked go to next closest neighbor
            if triangle==length(vertices)

                   k=k+1;

            end

            % Do inside test using barycentric coefficients
            if all(bc_coords >= 0)

               % Pass to the next query
               reachy = reachy+1;

               % store this triangle as closest one to q
               % store barycentric coefficients for re-tessellation
               % store p
```

```
                triang_id = [triang_id;faces(triangle)];
                triang_bary =[triang_bary;bc_coords];
                p_closest = [p_closest;p(k_temp)];

                break


            end


        end


    end

end
```

Task 3-4

```
%%%%%%%%%%%%%%%%%  3-4
H = load('/Users/xiyana/Downloads/med-course/homeworks/hw3/input_data/lh.white.H.txt');
[v_original, f_original] = read_vtk('/Users/xiyana/Downloads/med-course/homeworks/hw3/input_data/lh.white.vtk');
tr_original = triangulation(f_original+1, v_original);

% Initialize variables for the new mesh
newVertices = zeros(size(triang_bary, 1), size(v_original, 2));
new_H = zeros(size(triang_bary, 1), size(H, 2));

% Iterate over each vertex
for i = 1:size(triang_bary, 1)

    % Get the barycentric coefficients and associated faces for the vertex
    baryCoeffs = triang_bary(i, :);
    faces_id = triang_id(i, :);

    %Find vertices connected to the triangles and their ABC points
    vertices_original = tr_original.ConnectivityList(faces_id,:);
    ABC= tr_original.Points(vertices_original(1,:),:);

    % Do interpolation with the barycentric coefficients and triangles of
    % original mesh
    newVertex_v = ABC*baryCoeffs';
    newVertices(i,1) = newVertex_v(1);
    newVertices(i,2) = newVertex_v(2);
    newVertices(i,3) = newVertex_v(3);


end

% Find new H
for i=1:length(Q)

    new_H(i,:) = H(i);

end

write_property('/Users/xiyana/Downloads/med-course/homeworks/hw3/created/search_icosphere_4.vtk',newVertices, ...
    f, struct('H', new_H));
```

Task 4-1

Explanation

In 4-1 we generate harmonic basis functions at some specific degree. By using the spharm_real() function and as input the vertices of the sphere and the specific degree, the we save the harmonic bases in matrixVector for future tasks. Notice that for later use in 4-3 this code generates them at 10, 20 and 40 degree.

Code

```
clc;
clear;

%%%%%%%%%%%% Task 4-1 %%%%%%%%%%%%%%%%%%%%

% Load sphere
[v, f] = read_vtk('/Users/xiyana/Downloads/med-course/homeworks/hw3/input_data/lh.sphere.vtk');

% Define range for later use in task 4-3 obtaining harmonic for 10,20,40
max_lenghts = [10,20,40];

% Define matrix vector to save harmonic bases matrices
matrixVector = cell(1, 3);

%Define loop to generate harmonic base at degree 10, 20 and 40
for i=1:length(max_lenghts)

    % Define degree of base
    max_l = max_lenghts(i);

    % Define number of bases
    number_bs = (max_l+1)^2;

    % Define matrix to store harmonic bases
    harmonic_bs = zeros(size(v,1), number_bs);

    % Define loop to iterate from 0 to degree max_l
    for l = 0:max_l

        % Generate harmonic basis at degree l
        h = spharm_real(v, l);

        % Calculate the base index
        base_idx = l^2 + (1:2*l+1);

        % Save the harmonic base
        harmonic_bs(:, base_idx) = h;
    end

    % Save the harmonic bases at degree 10, 20 and 40
    matrixVector{i} = harmonic_bs;
end
```

Task 4-2

Explanation

In 4-2 we use the harmonic basis from task 4-1 that are in the matrixVector to obtain the coefficients that best approximate the input mesh by solving a linear system. We perform this process at degree 10,20 and 40 (Notice they are all save in matrixVector)

Code

```
%%%%%%% task 4-2
clear v;
clear f;

% Load mean curvate
H = load('/Users/xiyana/Downloads/med-course/homeworks/hw3/input_data/lh.white.H.txt');

% Load the overlayed input mesh
[v,f] = read_vtk('/Users/xiyana/Downloads/med-course/homeworks/hw3/created/lh.white.H.vtk');

% Define a vector to store coefficient matrices
coefficientsVector = cell(1, 3);
```

```
% Define loop to obtain coefficients at degree 10, 20 and 40
for i=1:length(matrixVector)

    % Get the stored harmonic bases at degree 10, 20 and 40
    harmonic_bs = matrixVector{i};

    % Solve linear system
    coefficient = harmonic_bs \ [v,H];

    % Save coefficients
    coefficientsVector{i} = coefficient;

    % Clear the coefficient variable just in case
    clear coefficient;

end
```

Task 4-3

Explanation

In 4-3 we use the coefficients from 4-2 to compute the new bases using the icosahedral surfaces icosphere 4, 5 and 6. First we compute the harmonic bases for the icosphere and then we use this together with the coefficients to obtain the reconstructed signals. Then we extract the reconstructed vertices and mean curvatures and then use write_property to write these to the icospheres.

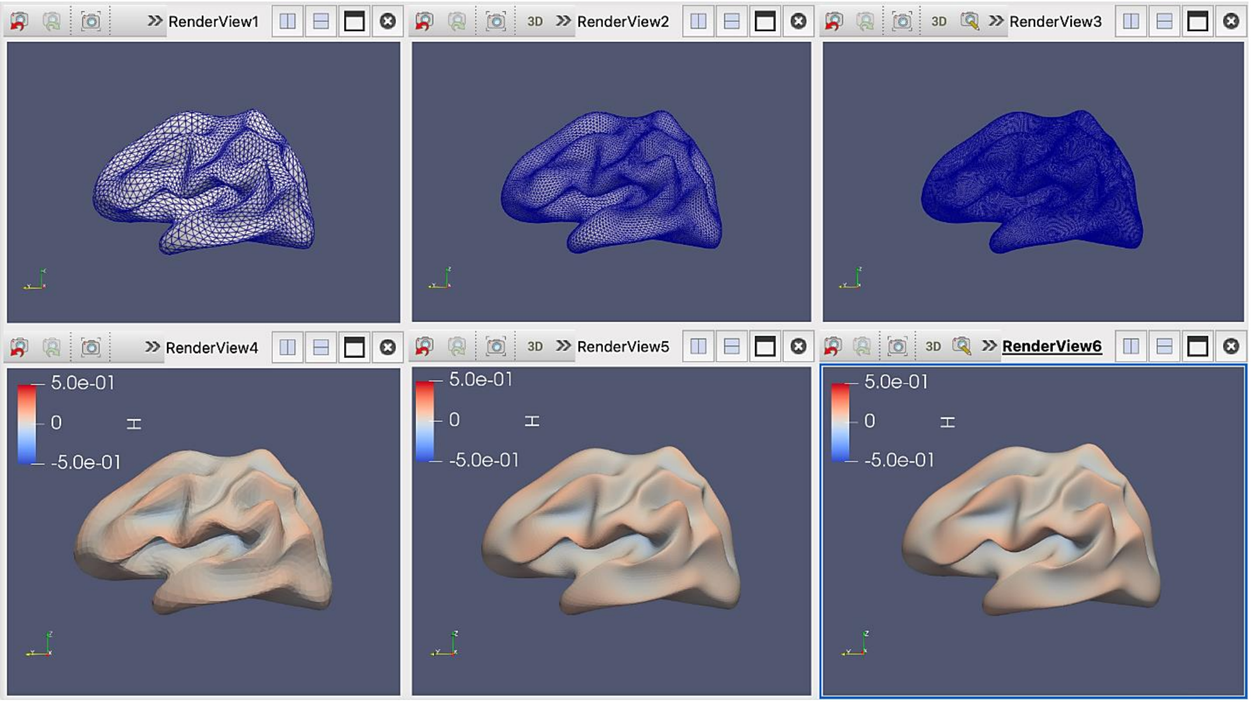
Code

```
%%%%%% Task 4-3 %%%%%%%%%%
clear H;
clear v;
clear f;
% 4-3 must be repeated for icosphere 4, 5 and 6

% Define range of degree for obtaining harmonic
max_lenghts = [10,20,40];

% Load icosphere
[v_ico,f_ico] = read_vtk('/Users/xiyana/Downloads/med-course/homeworks/hw3/icosphere_mesh/icosphere_6.vtk');

% Define a reconstruction vector for vertices
rcnst_v_Vector = cell(1, 3);

% Define a reconstruction vector for mean curvatures
rcnst_H_Vector = cell(1, 3);

% Define loop to reconstruct signals at degree 10, 20 and 40
for i=1:length(max_lenghts)

    % Define degree of base
    max_l = max_lenghts(i);

    % Define matrix for base of icosphere
    base = [];

    % Define loop to iterate from 0 to degree max_l
    for l = 0:max_l

        % Generate harmonic basis at degree l
        base = [base,spharm_real(v_ico,l)];

    end


    % Reconstruct signal using harmonic base of icosphere and coefficients
    % of the sphere ar the same degree from 4-2
    rcnst_signal = base * coefficientsVector{i};


     % Reconstruc vertices
     v = rcnst_signal(:,1:3);

     % Reconstruc mean curvatures
     H = rcnst_signal(:,4);

     % Save reconstructed vertices
     rcnst_v_Vector{i} = v;

     % Save reconstructed mean curvatures
     rcnst_H_Vector{i} = H;

     % Delete v and H
     clear v;
     clear H;

 end
```

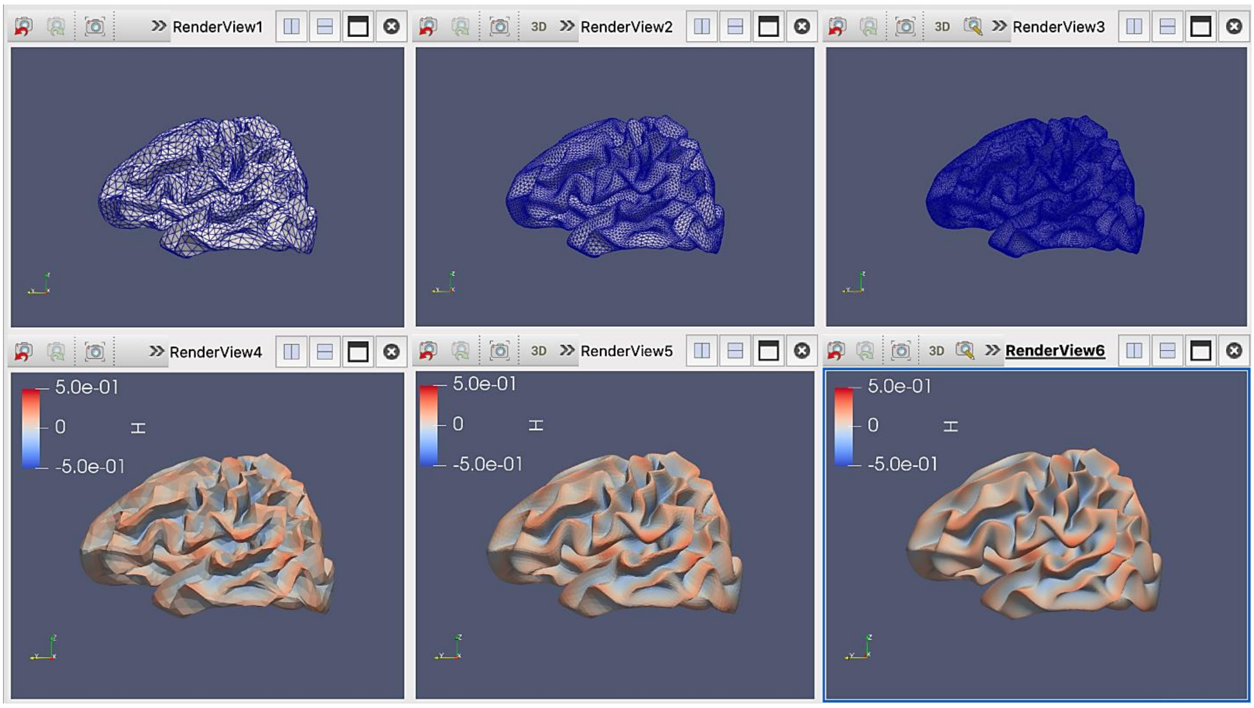
```
% Write the reconstructed v and H to the icoshpere at degree 10, 20 and 40

write_property('/Users/xiyana/Downloads/med-course/homeworks/hw3/created/icosphere_6_degree_10.vtk', rcnst_v_Vector{1}, ...
    f_ico, struct('H', rcnst_H_Vector{1}));

write_property('/Users/xiyana/Downloads/med-course/homeworks/hw3/created/icosphere_6_degree_20.vtk', rcnst_v_Vector{2}, ...
    f_ico, struct('H', rcnst_H_Vector{2}));

write_property('/Users/xiyana/Downloads/med-course/homeworks/hw3/created/icosphere_6_degree_40.vtk', rcnst_v_Vector{3}, ...
    f_ico, struct('H', rcnst_H_Vector{3}));
```

Results

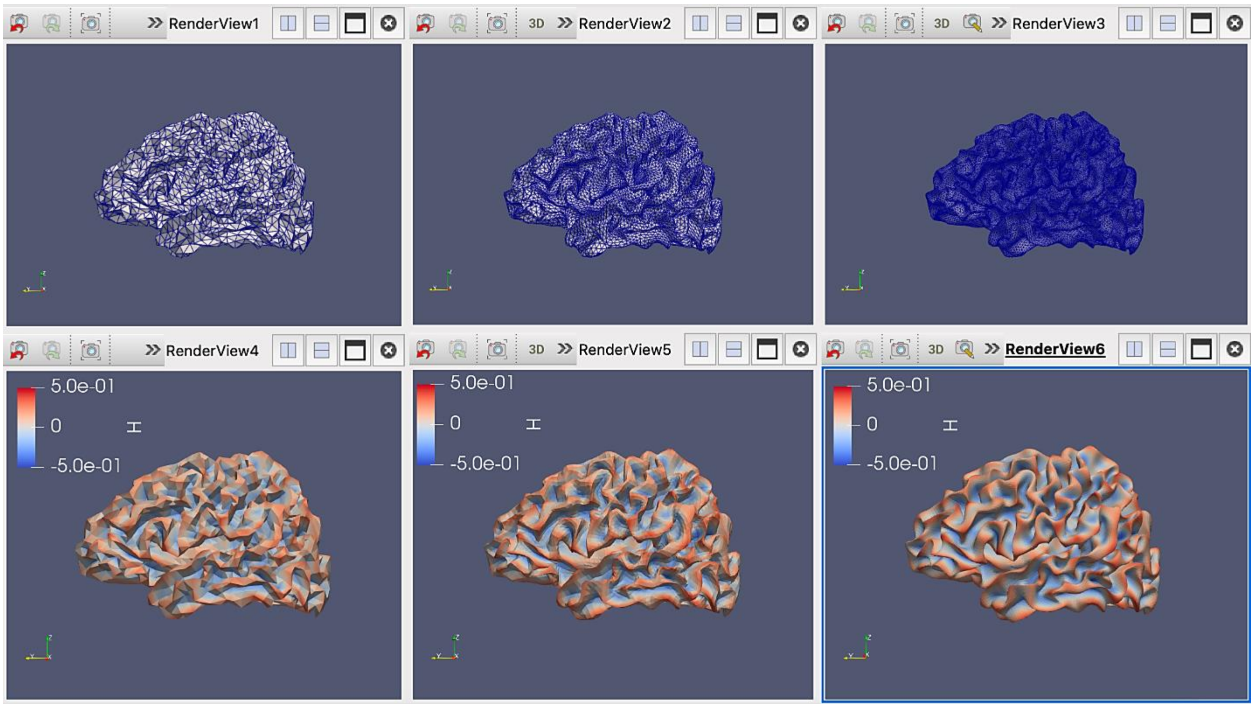
Icosphere 4, 5 and 6 at degree 10



Icosphere 4, 5 and 6 at degree 20



Icosphere 4, 5 and 6 at degree 40



Task 4-4

Harmonic-based re-tessellation techniques and search-based approaches are two commonly used methods in mesh processing. Harmonic-based methods focus on achieving smoothness while search-based approaches provide more flexibility and control by incorporating search algorithms and optimization techniques.

Harmonic based re-tessellation preserves the original features of the mesh while achieving a smooth mesh. It produces visually appealing meshes and ensures that important geometric details are preserved. But the control of the desired level of refinement is limited while also being sensitive to the quality of the original mesh which can lead to distortion.

Search-based approaches allow a better control during re-tessellation being more flexible and adaptable, often allowing localized refinement for better details in desired areas of the mesh. But these methods usually require optimization and fine-tunning making them of a higher computational complexity than harmonic-based re-tessellation.

Finally, the choice between harmonic-based re-tessellation and search-based approaches depends on the specific requirements for the mesh re-tessellation task. Harmonic-based methods offer simplicity, smoothness, and feature preservation but with limited control, while search-based approaches offer more control, adaptability, and localized refinement with a higher computational complexity .