Supporting Information to: Capturing Crystal Shape Evolution from Molecular Simulations

Ekaterina Elts* and Heiko Briesen

Chair of Process Systems Engineering, TUM School of Life Sciences, Technical University of Munich, Freising, Germany

E-mail: ekaterinaelts@mytum.de

DBSCAN clustering

A Matlab function to implement DBSCAN clustering of triangles based on the direction of their normal vectors (addressed by epsilon1) as well as on the spatial arrangement of triangles (addressed by epsilon2) is presented in Listing S1.

Listing S1: Matlab code

```
% Density-Based Spatial Clustering of Applications with Noise | DBSCAN
% Clustering algorithm based on the angle between triangle normals and
% the distance between triangle centroids
% Syntax: [c, idx]=DBSCAN_angle_distance(vects, points, epsilon1, epsilon2, minPts)
% Input:    vects    - array with triangle normals
%           points   - array with triangle centroids
%           epsilon1 - max. angle (in degrees) between normal vectors in a cluster
%           epsilon2 - max. distance between the centroids in a cluster
%           minPts   - min. number of triangles in a cluster
% Output:   c        - number of clusters
%           idx      - index of cluster for each triangle; if idx=-1, then noise

function [c, idx]=DBSCAN_angle_distance(vects, points, epsilon1, epsilon2, minPts)

n = length(vects(:,1)); % number of triangles
idx=zeros(n,1);
check=zeros(n,1);
A_max = epsilon1; % max angle between normal vectors in a cluster
D_max = epsilon2; % max distance between centroids in a cluster
A = zeros(n,n);
D = zeros(n,n);
S = zeros(n,n);
ResNum = zeros(n,1);
for i=1:n
    % find triangles with normal vectors deviating <= A_max from the current one
    for j=1:n
      A(:,i)=(acosd(dot(vects(j,:), vects(i,:))))<=A_max;
    end
    % find triangles with distance <= D_max from the current one
    D(:,i)=sqrt((points(:,1)-points(i,1)).^2 + (points(:,2)-points(i,2)).^2 + ...
                (points(:,3)-points(i,3)).^2)<=D_max;
    % find triangles satisfying both conditions
    S(:,i)=times(D(:,i),A(:,i));
    % calculate the number of triangles satisfying both conditions
    ResNum(i) = sum(S(:,i))-1;
    % mark numbers of triangles satisfying both conditions
    S(:,i)=S(:,i).*(1:n)';
end
% initialize cluster calculator
c=1;
for i=1:n
    if check(i) == 0 % if the triangle is not yet checked
        if ResNum(i)<minPts % if number of triangles satisfying conditions < minPts
```

```
            % mark triangle as "checked"
            check(i)=1;
            % mark triangle as "noise"
            idx(i)=-1;
        else % otherwise check further
            % save the current state of cluster distribution vector
            curState = idx(:);
            % mark the current triangle as belonging to the current cluster
            idx(i)=c;
            while sum(curState-idx(:))~=0 % while cluster distribution is changing
                % save the current state of cluster distribution vector
                curState = idx(:);
                for j=1:n
                    if idx(j)==c && check(j)==0 % triangle in cluster & not marked
                        if ResNum(j)<minPts % and has less than minPts neighbours
                            % mark it as "checked"
                            check(j)=1;
                            % mark it as "noise"
                            idx(j)=-1;
                        else % otherwise
                            % mark it as "checked"
                            check(j)=1;
                            % mark all triangles satisfying conditions
                            % as belonging to cluster
                            idx(S(j,:)>0)=c;
                        end
                    end;
                end;
            end;
            % increase the number of clusters
            c=c+1;
        end;
    else
       continue
    end;
end;
end
```