Replicating 3D surfaces of randomly distributed patterns

E.E. Gilberto Galvis

In a previous work we developed a function capable of generating a M-by-N matrix of Zsurf values that, seen as a surface, contain certain patterns or shapes whose size (width and height) vary according to a normal distribution and whose distance between center-to-center (pitch) along of the X and Y axes also follows a normal distribution. Now, similar surfaces are generated from certain sources and stored in particular files with an .sdf extension, which continue a certain header configuration and in the writing of the z values. Then, this document explains how these surfaces can be replicated starting from an .sdf file and using the previously developed function. We present four new functions that we have developed, with which we managed to read those files and compute the random distributions associated with the patterns to finally use the previous function and thus replicate the surface under study.

1. Developed functions

In this first section we present the four new functions that we have developed to extract the statistical distributions associated with the patterns/shapes distributed on the surface. We will be presenting in a systematic order according to the procedure used to achieve our goal. We then will start with the function that performs the reading of the .sdf files until we reach the main function that returns the parameters of the desired distributions.

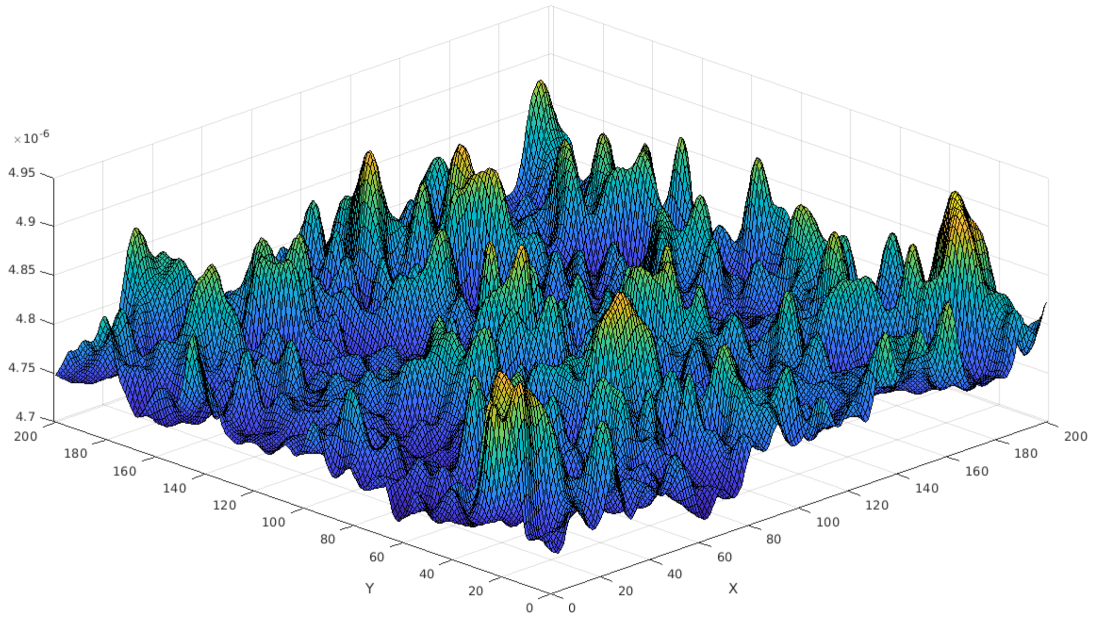
1.1. `Read SDF function`

We were looking for some function already developed that was able to read this type of files but we did not find it. Then we decided to develop it. We will let the function explain itself with its header comment.

```
%-----------------------------------------------------------------------------%
%---------------------------   read sdf function    --------------------------%
%                                                                             %
% [X,Y,Z] = read_sdf(filename) is a special function that reads the z values  %
% of files with extension .sdf. These files must have a specific configurati- %
% on, with a certain header and composition of their data. Then the function, %
% taking into account that, is responsible for properly extracting the z va-  %
% lues.                                                                       %
%                                                                             %
% INPUTS:                                                                     %
%   -filename:    direct path to the sdf file that you want to read.          %
%                                                                             %
%   >> filename = 'files/test_JP2.sdf';                                       %
%                                                                             %
% OUTPUTS:                                                                    %
%   -X: x coordinates corresponding to each z value in the file. These coordi- %
%       nates consider the number of elements and the scale that the file in- %
%       dicates in its header.                                                %
%   -Y: y coordinates corresponding to each z value in the file. These coordi- %
%       nates consider the number of elements and the scale that the file in- %
%       dicates in its header.                                                %
%   -Z: z values read from the file. Consider the z scaling specified in the  %
%       header.                                                             %
%                                                                           %
%  NOTE: Now all the size parameters of the polygonal shapes (w and h)      %
%        will be defined by the distribution parameters specified in polyshape %
%                                                                           %
%  Example of usage:                                                        %
%      [Xg,Yg,Zg] = read_sdf(filename); read the file                       %
%      figure(), surf(Xg, Yg, Zg)       make the surface                    %
%                                                                           %
%-------------------------------------------------------------------------%
```

Next we show that the function works well. For this (and henceforth) we will use the files called 'test_JP2.sdf'

```
% --- parameters
  filename = 'files/test_JP1.sdf';

% --- load the dataset
  %[~,~,Zg] = read_sdf(filename);   % read the file
  no = 450; nx = 650; ny = 650;    % define a work domain
  Zsurf = Zg(no:ny, no:nx);

% --- show the original 3D surf
  figure(), surf(Zsurf)
  xlabel('X'), ylabel('Y');
  view(-45, 45)
  xlim([0 size(Zsurf, 2)])
  ylim([0 size(Zsurf, 1)])
  set(gcf, 'units','normalized', 'position', [0 0 1 1]);
```



NOTE: We have selected a reduced work domain of 200-by-200 to be able to visualize the surfaces well. Then we select the domain within the total surface to avoid the edges

1.2. `FastPeakFind function`

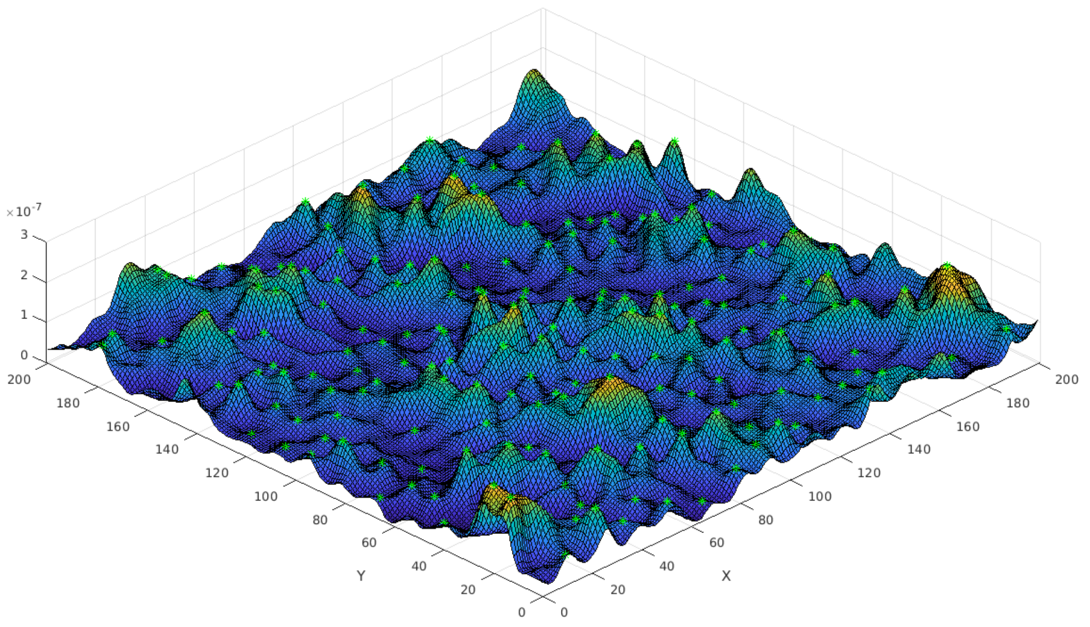
This function is used to find the centroids of the patterns/shapes on the surface. Initially we thought about thresholding Zsurf and using the `matlab regiongrous funtion`. However, by testing we saw that `regiongrous` had a low success rate, leaving many centroids undetected. Then we decided to look for a more robust way and we got this function. We will let the function explain itself with its header comment.

```
% Analyze noisy 2D images and find peaks using local maxima (1 pixel
% resolution) or weighted centroids (sub-pixel resolution).
% The code is designed to be as fast as possible, so I kept it pretty basic.
% The code assumes that the peaks are relatively sparse, test whether there
% is too much pile up and set threshold or user defined filter accordingly.
%
% How the code works:
% In theory, each peak is a smooth point spread function (SPF), like a
% Gaussian of some size, etc. In reality, there is always noise, such as
%"salt and pepper" noise, which typically has a 1 pixel variation.
% Because the peak's PSF is assumed to be larger than 1 pixel, the "true"
% local maximum of that PSF can be obtained if we can get rid of these
% single pixel noise variations. There comes medfilt2, which is a 2D median
% filter that gets rid of "salt and pepper" noise. Next we "smooth" the
% image using conv2, so that with high probability there will be only one
% pixel in each peak that will correspond to the "true" PSF local maximum.
% The weighted centroid approach uses the same image processing, with the
% difference that it just calculated the weighted centroid of each
% connected object that was obtained following the image processing.  While
% this gives sub-pixel resolution, it can miss peaks that are very close to
% each other, and runs slightly slower. Read more about how to treat these
% cases in the relevant code commentes.
%
% Inputs:
% d     The 2D data raw image - assumes a Double\Single-precision
%       floating-point, uint8 or unit16 array. Please note that the code
%       casts the raw image to uint16 if needed.  If the image dynamic range
%       is between 0 and 1, I multiplied to fit uint16. This might not be
%       optimal for generic use, so modify according to your needs.
% thres A number between 0 and max(raw_image(:)) to remove  background
% filt  A filter matrix used to smooth the image. The filter size
%       should correspond the characteristic size of the peaks
% edg   A number>1 for skipping the first few and the last few 'edge' pixels
% res   A handle that switches between two peak finding methods:
%       1 - the local maxima method (default).
%       2 - the weighted centroid sub-pixel resolution method.
%       Note that the latter method takes ~20% more time on average.
% fid   In case the user would like to save the peak positions to a file,
%       the code assumes a "fid = fopen([filename], 'w+');" line in the
%       script that uses this function.
```

```
%
%Optional Outputs:
% cent        a 1xN vector of coordinates of peaks (x1,y1,x2,y2,...
% [cent cm]   in addition to cent, cm is a binary matrix  of size(d)
%             with 1's for peak positions. (not supported in the
%             the weighted centroid sub-pixel resolution method)
%
%Example:
%
%   p=FastPeakFind(image);
%   imagesc(image); hold on
%   plot(p(1:2:end),p(2:2:end),'r+')
%
%   Adi Natan (natan@stanford.edu)
%   Ver 1.7 , Date: Oct 10th 2013
```

Next we show that the function works well.

```
  % 2D signal from zero
  Z = Zsurf - min(Zsurf(:));

  % work size
  ly = size(Z, 1);
  lx = size(Z, 2);

% --- get the centroids

  % execute the function
  pks = FastPeakFind(Z, median(Z));

  % then the centroids are
  centers = [pks(1:2:end), pks(2:2:end)];

% --- show the surf and its centroids

  hcenters = zeros(size(centers, 1), 1);
  % height of each centroid
  for k = 1:size(centers, 1)
    % get the centroid coordenates
    hcenters(k) = Z(centers(k,2), centers(k,1));
  end

  figure(), surf(Z)
  hold on, plot3(centers(:, 1), centers(:, 2), hcenters, 'g*')
  xlabel('X'), ylabel('Y');
  view(-45, 70)
  xlim([0 size(Zsurf, 2)])
  ylim([0 size(Zsurf, 1)])
  set(gcf, 'units','normalized', 'position', [0 0 1 1]);
```



With simple visual inspection, we can see that a high percentage of real centroids were identified (almost all centroids)

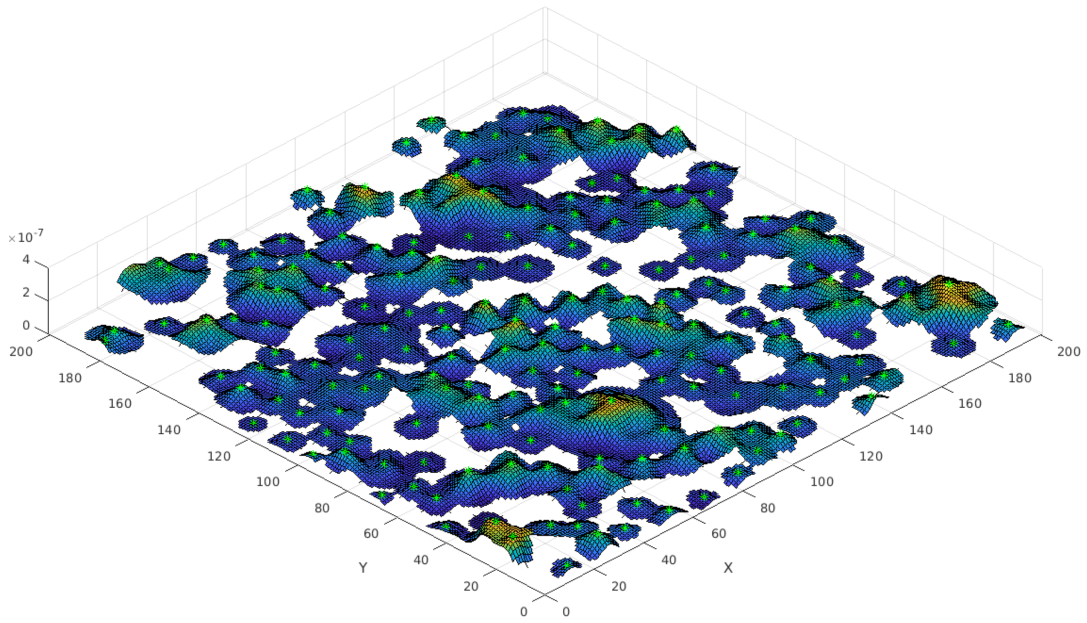
1.3. `get features function`

This function stems from the search for a clustering algorithm to extract patterns from previously found centroids. But it was difficult to find an already developed function that did it, because they all use a stop criterion such as the number of samples in the groups or the maximum distance between the points. That does not help us because precisely the patterns have variable size. Then we decided to develop our own clustering algorithm based on the height of the shapes (we only work for now with top hemispheres). In this way, the function takes advantage of the sweep for each centroid and extracts the features of the forms such as: width and height. It also returns coordinate matrices that will later help us find the distribution parameters for center-to-center distances between patterns. We will let the function explain itself with its header comment.

```
%-----------------------------------------------------------------------------%
%-------------------------   get features function    -------------------------%
%                                                                             %
% [wc, hc, xp, yp] = get_features(Zsurf, pts) It is a function that determines %
% the size of each of the shapes (widths and heights) distributed on the Zsurf %
% surface, starting from its centroids. The function internally runs an itera- %
% tive clustering algorithm to extract all the points of each sahpe starting   %
% from the centroid. The algorithm iteratively selects point rings from the    %
% centroid, computes the average value (corresponding to the average height of %
% the ring) and then computes the distance of the current ring from the pre-   %
% vious one. The reference or stop criterion is the distance of the first ring %
% with respect to the centroid. If the distance between the current ring and   %
% the previous one is less than that reference, then the grouping ends. Then   %
% the last ring found corresponds to the width of the current pattern under    %
% study. Finally, the width of the shapes is the diameter of that last ring   %
% and the heights are simply the evaluation of Zsurf in the centroid.         %
%                                                                             %
% INPUTS:                                                                     %
%   -Zsurf: M-by-N matrix of Zsurf values that, seen as a surface, contain    %
%           certain patterns or shapes whose size (width and height) vary ac- %
%           cording to a normal distribution and whose distance between center %
%           to center (pitch) along of the X and Y axes also follows a normal  %
%           distribution                                                      %
%                                                                             %
%    >> [~,~,Zurf] = read_sdf('test_JP2.sdf');                                %
%                                                                             %
%   -pts: centroid points of each of the patterns in Zsurf                    %
%                                                                             %
% OUTPUTS:                                                                    %
%   -wc: Width of each of the shapes                                          %
%   -hc: Height of each of the shapes                                         %
%   -xp: X coordinate of the centroids of each shape geolocated in a matrix   %
%        of the same dimension as Zsurf (the matrices contain NaN values at   %
%        the other points where there is no centroid).                        %
%   -yp: Y coordinate of the centroids of each shape geolocated in a matrix   %
%        of the same dimension as Zsurf (the matrices contain NaN values at   %
%        the other points where there is no centroid).                        %
%                                                                             %
%  Example of usage:                                                          %
%      [~,~,Zurf] = read_sdf('test_JP2.sdf');                                 %
%                                                                             %
%      % -- 2D surf from zero                                                 %
%      Zsurf = Zsurf - min(Zsurf(:));                                         %
%                                                                             %
%      % -- function that find peaks over 2D signals (centroids)              %
%      pks = FastPeakFind(Z, median(Z));                                      %
%                                                                             %
%      % -- then the centroids are                                            %
%      centers = [pks(1:2:end), pks(2:2:end)];                                %
%                                                                             %
%      % -- execute our function                                              %
%     [wc, hc, xp, yp] = get_features(Z, centers);                %
%                                                                             %
%-----------------------------------------------------------------------------%
```

Next we show that the function works well.

```
% --- execute the function
  [wc, hc, xp, yp] = get_features(Z, centers);
  [Xs, Ys] = meshgrid(1:lx, 1:ly);

% --- Show only the patters extracted
  Zs = NaN(size(Z));
  ind = zeros(size(Z));
  for i = 1:size(wc, 1)
    xo = round(centers(i, 1));
    yo = round(centers(i, 2));
    r = wc(i) / 2;
    ind = ind + ( (Xs-xo).^2 + (Ys-yo).^2 <= r^2 );
  end
  ind = logical(ind);
  Zs(ind) = Z(ind);

  figure(), surf(Zs)
  hold on, plot3(centers(:, 1), centers(:, 2), hcenters, 'g*')
  xlabel('X'), ylabel('Y');
  view(-45, 80)
  xlim([0 size(Zs, 2)])
  ylim([0 size(Zs, 1)])
  set(gcf, 'units','normalized', 'position', [0 0 1 1]);
```



1.4. get 3D pattern statistics function

We have developed this function to provide a main function that encapsulates the two previous functions and returns the parameters of the random distributions associated with the patterns/shapes. Specifically, this function locates the centroids of the shapes by executing the `FastPeakFind function` and then runs the `get_features function` to obtain the values of widths, heights and coordenate matrices. Finally, on the last values returned, calculate the distribution parameters. We will let the function explain itself with its header comment.

```
%---------------------------------------------------------------------------%
%-----------------------   get 3D pattern statistics   ---------------------%
%                                                                           %
% [polyshape, distparams, imsize, features] = ...                           %
%     get_3D_pattern_statistics(Zsurf, pattype)                             %
% receives an M-by-N matrix of Zsurf values that, seen as a surface, contain %
% certain patterns or shapes whose size (width and height) vary according to %
% a normal distribution and whose distance between center to center (pitch)   %
% along of the X and Y axes also follows a normal distribution. Then the func- %
% tion, also knowing as input the type of patterns (pattype, top hemispheres   %
% for example) contained in Zsurf, locate these patterns and extract their     %
% main statistics. That is, it determines the normal distributions associated  %
% with its size and pitch.                                                     %
%                                                                              %
% INPUTS:                                                                      %
%   -Zsurf: M-by-N matrix Z values read from an .sdf file with the special     %
%           function read_sdf().                                               %
%                                                                              %
%  >> [~,~,Zurf] = read_sdf('test_JP2.sdf');                                   %
%                                                                              %
%  -pattype: string indicating the type of patterns contained in Zsurf (See    %
%            table below)                                                      %
%  >> pattype = 'htop' the patterns are top hemispheres (top curvature)        %
%                                                                              %
% OUTPUTS:                                                                     %
%  -polyshape:    Array of 3 cells defining the pattern of the 3D shapes:      %
%                 - First cell is a string specifying the type of 3D shape de- %
%                   sired (see the table below to know the types of shapes     %
%                   available).                                                %
%                 - Second cell is a vector that can be two or four elements.  %
%                   The two first elements are the mean & standar deviation of %
%                   a pseudo-normal distribution used to define the width of   %
%                   each polygon. The 3rd & 4rd elements (optionals) are the   %
%                   min & max values, respectibily, to truncate the pseudo-    %
%                   normal distribution                                        %
%                 - Third cell is similar to the previous one but to define    %
%                   the height of the shapes                                   %
%  >> polyshape = {'htop', [meanwidth, stdwidth], [meanheight, stdheight]}     %
%                              - hemispheres (top curvature) with              %
%                                width following ~ N(meanwidth, stdwidth)      %
%                                height following ~ N(meanheight, stdheight)  %
%                                                                              %
%  -distparams:   Vector of two or four elements where the two first of them   %
%                 specify the mean & standar deviation error of a pseudo-      %
%                 normal distribution used to define the center-to-center dis- %
%                 tance (pitch). The 3rd & 4rd elements (optionals) are the    %
%                 min & max values, respectibily, to truncate the pseudo-      %
%                 normal distribution                                          %
%                                                                              %
%  -imsize:       Vector of two elements specifying the size of the 3D out-    %
%                 put surface, 'img'.                                          %
%                                                                              %
%  -features:    Array of four cells. The first cell contains the widths of    %
%                each extracted shape. The second cell contains the height va- %
%                lues. The last cell is an Nc-by-2 matrix that contains the X  %
%                and Y coordinates of the centroids of the extracted shapes.   %
%                Where Nc is the number of centroids                           %
%                                                                              %
%  Table: pattype options                                                      %
%  +------------+------------+-------------+--------+----------------+         %
%  | shape       |  long form  |  short form  |  size   |  description    |    %
%  +-------------+-------------+--------------+---------+-----------------+    %
%  +-------------+-------------+--------------+---------+-----------------+    %
%  | hemisphere | 'htop'      |  'ht'        | [w, h]  | w: width        |    %
%  | top        |             |              |         | h: height       |    %
%  +-------------+-------------+--------------+---------+-----------------+    %
%  | hemisphere | 'hbottom'   |  'hb'        | [w, h]  | w: width        |    %
%  | bottom     |             |              |         | h: height       |    %
%  +-------------+-------------+--------------+---------+-----------------+    %
%                                                                              %
%  Example of usage:                                                           %
%      [~,~,Zurf] = read_sdf('test_JP2.sdf');                                  %
%      [polyshape, distparams, imsize, features] = ...                         %
%          get_3D_pattern_statistics(Zsurf, pattype);                          %
%                                                                              %
%      % execute the previus function to replicate the original Zsurf          %
%      [imgp, pitchp, wsp, hsp] = ...                                          %
%          shape_placement_3D_V2(imsize, polyshape, distparams);               %
%      Zp = imgp(:,:,3);                                                       %
%                                                                              %
%      % original 3D surf                                                      %
%      figure(), surf(Z)                                                       %
%      xlabel('X'), ylabel('Y');                                               %
%      xlim([0 imsize(1)])                                                     %
%      ylim([0 imsize(2)])                                                     %
%                                                                              %
%      % replicated 3D surf                                                    %
%      figure(), surf(Zp);                                                     %
%      xlabel('X'), ylabel('Y');                                               %
%      xlim([0 imsize(1)])                                                     %
%      ylim([0 imsize(2)])                                                     %
%                                                                              %
%------------------------------------------------------------------------------%
```

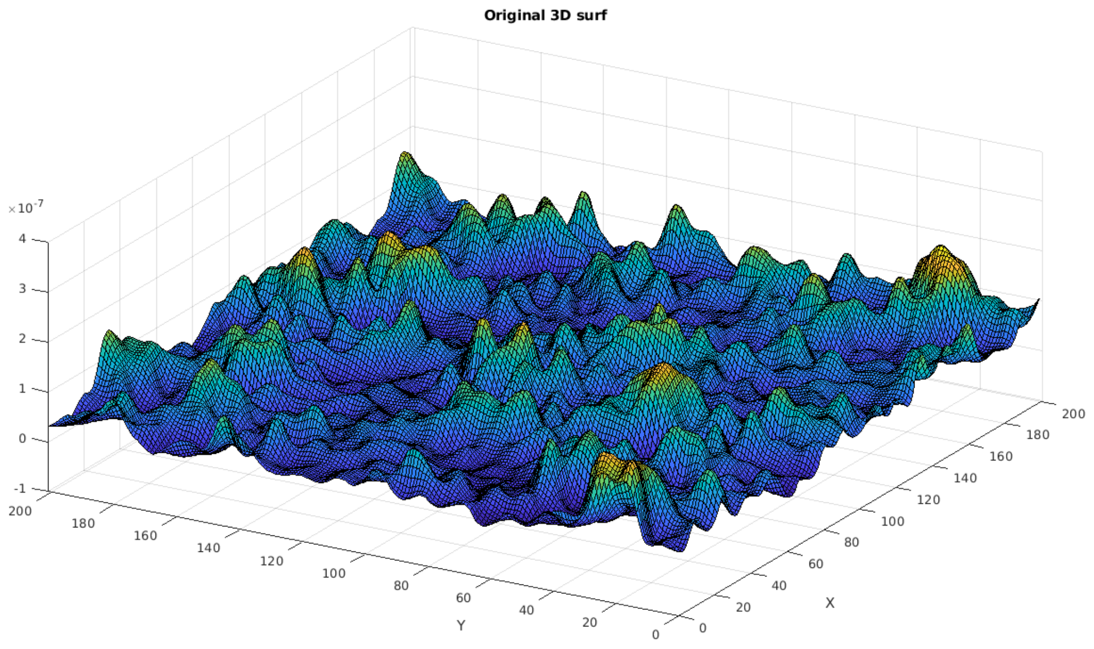
In the next section, we will test this function by trying to replicate the original 3D surface.

2. Replicating surfaces

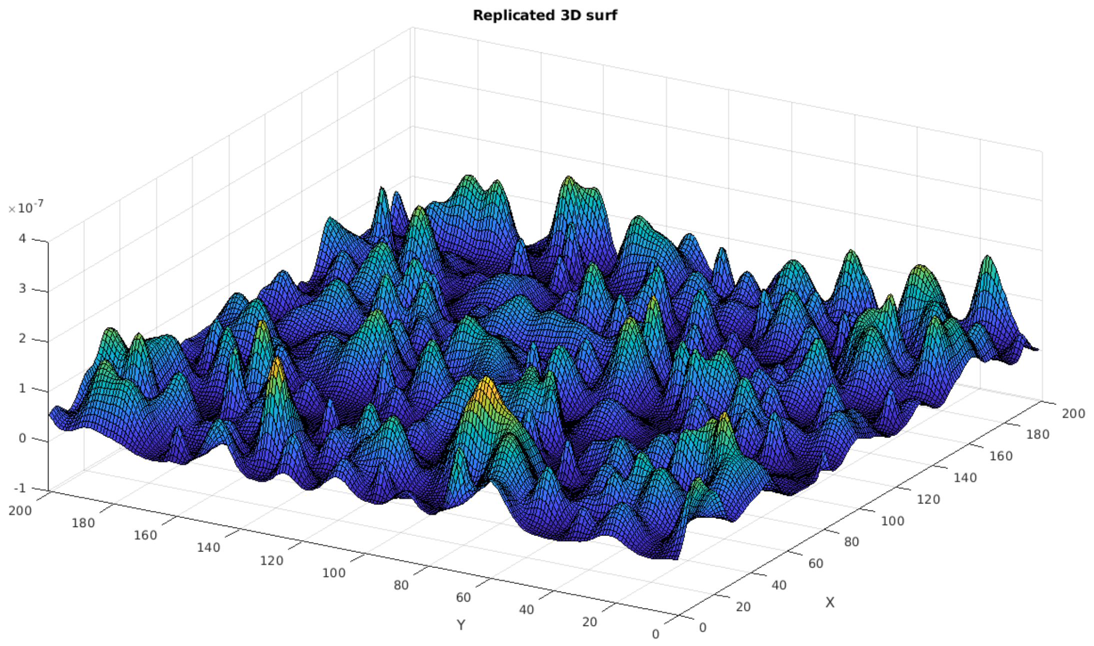
Here we will test the main function when trying to replicate the original 3D surface, generating a synthetic surface from the returned distribution parameters and executing the function of the previous work.

```
% --- get the 3D pattern statistics

  % parameters
  pattype = 'htop'; % We knows that the pattern type is hemisphere top

  % execute the main function
  [polyshape, distparams, imsize, features] = get_3D_pattern_statistics(Zsurf, pattype

% --- replicate the original 3D surface
```

```
  % execute the function of previus work
  [imgp, pitchp, wsp, hsp] = shape_placement_3D_V2(imsize, polyshape, distparams);

  % get only the z values
  Zp = imgp(:,:,3);

% --- plot the surfaces

  % original 3D surf
  figure(), surf(Z)
  xlabel('X'), ylabel('Y');
  title('Original 3D surf')
  xlim([0 imsize(1)])
  ylim([0 imsize(2)])
  zlim([-1e-7 4e-7])
  view(-60, 45)
  set(gcf, 'units','normalized', 'position', [0 0 1 1]);
```



```
  % replicated 3D surf
  figure(), surf(Zp);
  xlabel('X'), ylabel('Y');
  title('Replicated 3D surf')
  xlim([0 imsize(1)])
  ylim([0 imsize(2)])
  zlim([-1e-7 4e-7])
  view(-60, 45)
  set(gcf, 'units','normalized', 'position', [0 0 1 1]);
```


Replicated 3D surf
×10-7
4
3
2
1
0
-1
200
180
160
140
120
100
80
60
40
20
0
Y
0
20
40
60
80
100
120
140
160
180
200
X