```
function pointCloud = farthest_point_sampling(obj_file)
% get data from parser,
% modified object file reader function implemented by someone else
obj = wf_load(obj_file);

% get vertices and faces of the object
vert = obj.vertex;
face = obj.faces;

% calculate area and weights of triangles in mesh
area_tri = tri_area_mesh(vert,face);
weight = area_tri/sum(area_tri);

% determine number of points to sample from each triangle
total_pts = 10000;
num_pts = round(weight*total_pts);

% sampling of about 10K points
P = zeros(sum(num_pts),3);
count = 1;

for i=1:size(face,1)
    vert1 = vert(face(i,1),:);
    vert2 = vert(face(i,2),:);
    vert3 = vert(face(i,3),:);
    D = sample_pts(vert1, vert2, vert3, num_pts(i));

    P(count:(count+num_pts(i)-1),:) = D;
    count = count+num_pts(i);
end

% calculation of distance matrix
dist_mat = pdist2(P,P);

% farthest point sampling pipeline
S = zeros(1000,3);
S_ind = zeros(1000,1); % index of point from set P

init = randi([1,size(P,1)]);
S(1,:) = P(init,:); % initial pt
S_ind(1) = init;
dist = dist_mat(init,:);

for i=2:1000
    [max_val, ind] = max(dist);
    S(i,:) = P(ind,:);
    S_ind(i) = ind;
    dist_new = dist_mat(ind,:);
    dist = min(dist, dist_new);
end
```

```
pointCloud = S;
end

function area = tri_area_mesh(vert, face)
    s12 = vert(face(:,2),:) - vert(face(:,1),:);
    s13 = vert(face(:,3),:) - vert(face(:,1),:);
    val = cross(s12,s13);
    area = 0.5 * sqrt(val(:,1).^2 + val(:,2).^2 + val(:,3).^2);
end

function D = sample_pts(A, B, C, numPts)
    r = rand(numPts,2);
    r1 = sqrt(r(:,1));
    r2 = r(:,2);
    D = ((1-r1)*A) + ((r1.*(1-r2))*B) + ((r1.*r2)*C);
end

%%
function bundle = wf_load(filename)
% WF_LOAD  Loads a wavefront object from a file.
%    OBJ = WF_LOAD(FILENAME) Loads a 3D model contained in the file with
%       the given FILENAME. The file must be in the format specified by the
%       Wavefront specification. The full specification can be found in the
%       following URL:
%
%       http://netghost.narod.ru/gff/vendspec/waveobj/obj_spec.txt
%
%    Generally speaking, a wavefront object is a structure which contains
%    the XYZ coordinates of points in a 3D space. These points are used as
%    reference points to define polygons, lines, curves and surfaces which
%    define the object. The structure contains the required information to
%    construct the 3D object. This information may specify a set of
%    vertices and the way in which such a set should be connected to form a
%    polygon.
%
%    Example 1
%    ---------
%    This is an example of how a simple cube would be represented in a .obj
%    file:
%
%       # Vertices
%       v  -0.5  -0.5  -0.5
%       v  -0.5  -0.5   0.5
%       v  -0.5   0.5  -0.5
%       v  -0.5   0.5   0.5
%       v   0.5  -0.5  -0.5
%       v   0.5  -0.5   0.5
%       v   0.5   0.5  -0.5
%       v   0.5   0.5   0.5
```

```
%
%       # Triangle facets
%       f  1  7  5
%       f  1  3  7
%       f  2  6  8
%       f  2  8  4
%       f  1  4  3
%       f  1  2  4
%       f  3  8  7
%       f  3  4  8
%       f  5  7  8
%       f  5  8  6
%       f  1  5  6
%       f  1  6  2
%
%    In the example above the vertices of the cube are defined first,
%    and then the vertices are joined to form triangular facets, which form
%    the cube. In this case, the 1st ( -0.5  -0.5  -0.5 ), 7th (  0.5   0.5
%    -0.5 ) and 5th ( 0.5  -0.5  -0.5 ) vertices form the first face, and
%    so on.
%
%    NOTES:
%       - Parser is very lazy and will not generate errors on for badly
%       formated OBJ files.
%       - Parser does not implement some statements (free-form geometry
%       statements), however a warning is displayed when a non-implemented
%       method is encountered.
%
%    See also ISWF

  file = fopen(filename,'r');
  if file < 0
    error([ 'Could not open file: ' filename ]);
  else
    disp(['Reading file: ' filename]);
  end

  % Initialize object fields
  bundle.vertex    = [];
  bundle.vtex      = [];
  bundle.vnorm     = [];
  bundle.vparam    = [];
  bundle.points    = {};
  bundle.lines     = {};
  bundle.faces     = [];
%   bundle.faces     = {};
  bundle.curvs     = [];
  bundle.curvs2    = [];
  bundle.surfs     = [];
  bundle.conn      = [];
  curv_surf        = [];

  while ~feof(file)
    elems = getline(file);

    if ~isempty(elems)
      command = lower(elems{1});

      if command(1)=='#' % It is a comment, do nothing
      elseif strcmp(command,'v')   % GEOMETRIC VERTICES
        x = str2double(elems{2});
        y = str2double(elems{3});
        z = str2double(elems{4});
%         if length(elems)>=5
%           w = str2double(elems{5});
%         else
%           w = 1;
%         end
%         bundle.vertex = [ bundle.vertex; x y z w ];
        bundle.vertex = [ bundle.vertex; x y z ];
      elseif strcmp(command,'vt')  % TEXTURE VERTICES
        u = str2double(elems{2});
        v = str2double(elems{3});
        if length(elems)>=4
          w = str2double(elems{4});
        else
          w = 0;
        end
        bundle.vtex = [ bundle.vtex; u v w ];
      elseif strcmp(command,'vn')  % VERTEX NORMALS
        i = str2double(elems{2});
        j = str2double(elems{3});
        k = str2double(elems{4});
        bundle.vnorm = [ bundle.vnorm; i j k ];
      elseif strcmp(command,'vp')  % PARAMETER SPACE VERTICES
        u = str2double(elems{2});
        v = str2double(elems{3});
        if length(elems)>=4
          w = str2double(elems{4});
        else
          w = 1;
        end
        bundle.vparam = [ bundle.vparam; u v w ];
      elseif strcmp(command,'cstype') % CURVE/SURFACE TYPE
        if length(elems)==2
          rat = false;
          type = lower(elems{2});
        elseif length(elems)==3
          if strcmpi(elems{2},'rat')
```

```
            rat = true;
          end
          type = lower(elems{3});
        end
        if sum(strcmp(type, {'bmatrix','bezier','bspline',...
                'cardinal','taylor'}))
          curv_surf = getFFCurvSurf(file,rat,type);
        end

      elseif strcmp(command,'curv')   % CURVE
        if isempty(curv_surf)
          error('  ERROR: CSTYPE not defined while defining CURV');
        else
          curv = curv_surf;  % Makes a copy with the same values
        end
        curv.u0 = str2double(elems{2});
        curv.u1 = str2double(elems{3});

        curv.v = zeros(length(elems)-3,1);
        curv.v(1) = getTriplet(elems{4}, bundle);
        curv.v(2) = getTriplet(elems{5}, bundle);
        for i=3:size(curv.v,1)
          curv.v(i) = getTriplet(elems{i+3}, bundle);
        end
        curv = getCSBodyStatements(file,curv);
        bundle.curvs = [ bundle.curvs ; curv ];
      elseif strcmp(command,'curv2')   % CURVE 2
        if isempty(curv_surf)
          error('  ERROR: CSTYPE not defined while defining CURV2');
        else
          curv2 = curv_surf;  % Makes a copy with the same values
        end
        curv2.vp = zeros(length(elems)-1,1);
        curv2.vp(1) = handleNegVert(str2double(elems{2}), ...
            size(bundle.vparam,1));
        curv2.vp(2) = handleNegVert(str2double(elems{3}), ...
            size(bundle.vparam,1));
        for i=3:size(curv2.vp,1)
          curv2.vp(i) = handleNegVert(str2double(elems{i+1}), ...
              size(bundle.vparam,1));
        end
        curv2 = getCSBodyStatements(file,curv2);
        bundle.curvs2 = [ bundle.curvs2 ; curv2 ];
      elseif strcmp(command,'surf')   % SURFACE
        if isempty(curv_surf)
          error('  ERROR: CSTYPE not defined while defining SURF');
        else
          surf = curv_surf;  % Makes a copy with the same values
        end
        surf.s0 = str2double(elems{2});
        surf.s1 = str2double(elems{3});
        surf.t0 = str2double(elems{4});
        surf.t1 = str2double(elems{5});

        surf.v = zeros(length(elems)-5,3);
        for i=1:size(surf.v,1)
          [ surf.v(i,1) surf.v(i,2) surf.v(i,3) ] = ...
              getTriplet(elems{i+5}, bundle);
        end
        surf = getCSBodyStatements(file,surf);
        bundle.surfs = [ bundle.surfs ; surf ];
      elseif strcmp(command,'p')      % POINT
        p = zeros(length(elems)-1,1);
        for i=1:size(p,1)
          p(i) = getTriplet(elems{i+1}, bundle);
        end
        bundle.points = [ bundle.points ; p ];
      elseif strcmp(command,'l')      % LINE
        l = zeros(length(elems)-1,2);
        for i=1:size(l,1)
          [ l(i,1) l(i,2) ] = getTriplet(elems{i+1}, bundle);
        end
        bundle.lines = [ bundle.lines ; l ];
      elseif strcmp(command,'f')     % FACE
          f1 = str2num(elems{2});
          f2 = str2num(elems{3});
          f3 = str2num(elems{4});
          bundle.faces = [ bundle.faces; f1 f2 f3 ];
%       elseif strcmp(command,'f')      % FACE
%         f = zeros(length(elems)-1,3);
%         for i=1:size(f,1)
%           [ f(i,1) f(i,2) f(i,3) ] = getTriplet(elems{i+1}, bundle);
%         end
%         bundle.faces = [ bundle.faces ; f ];
%

%         elseif strcmp(command,'v')   % GEOMETRIC VERTICES
%         x = str2double(elems{2});
%         y = str2double(elems{3});
%         z = str2double(elems{4});
%         if length(elems)>=5
%           w = str2double(elems{5});
%         else
%           w = 1;
%         end
%         bundle.vertex = [ bundle.vertex; x y z w ];
      elseif strcmp(command,'con')    % CONNECTIVITY
        conn.surf_1 = str2double(elems{2});
        conn.q0_1   = str2double(elems{3});
        conn.q1_1   = str2double(elems{4});
        conn.curv2d_1=str2double(elems{5});
        conn.surf_2 = str2double(elems{6});
        conn.q0_2   = str2double(elems{7});
        conn.q1_2   = str2double(elems{8});
        conn.curv2d_2=str2double(elems{9});

        bundle.conn = [ bundle.conn ; conn ];
      elseif strcmp(command,'g')      % GROUP NAME
        unimplemented(command,filename);
      elseif strcmp(command,'s')      % SMOOTHING GROUP
        unimplemented(command,filename);
      elseif strcmp(command,'mg')     % MERGING GROUP
        unimplemented(command,filename);
      elseif strcmp(command,'o')      % OBJECT NAME
        unimplemented(command,filename);
      elseif strcmp(command,'bevel')  % BEVEL INTERPOLATION
        unimplemented(command,filename);
      elseif strcmp(command,'c_interp')% COLOUR INTERPOLATION
        unimplemented(command,filename);
      elseif strcmp(command,'d_interp')% DISSOLVE INTERPOLATION
        unimplemented(command,filename);
      elseif strcmp(command,'lod')    % LEVEL OF DETAIL
        unimplemented(command,filename);
      elseif strcmp(command,'maplib') % LIBRARY MAP
        unimplemented(command,filename);
      elseif strcmp(command,'usemap') % TEXTURE MAP
        unimplemented(command,filename);
      elseif strcmp(command,'usemtl') % MATERIAL
        unimplemented(command,filename);
      elseif strcmp(command,'mtllib') % MATERIAL LIBRARY
        unimplemented(command,filename);
      elseif strcmp(command,'shadow_obj')% SHADOW
        unimplemented(command,filename);
      elseif strcmp(command,'trace_obj')% RAY TRACING
        unimplemented(command,filename);
      else
        disp(['Unknown element: ' command ]);
      end
    end
  end

  fclose(file);

  function curv_surf = getFFCurvSurf(fid,rat,type)
  % Create Curvature/Surface object
    curv_surf.type = type;
    curv_surf.israt = rat;

    doit = 1;
    while doit && ~feof(fid)
```

```
pos = ftell(fid);
elems = getline(fid);
if ~isempty(elems)
  command = lower(elems{1});

  if strcmp(command,'deg')      % DEGREE
    if strcmp(curv_surf.type,'cardinal')
      curv_surf.degu = 3;
    else
      curv_surf.degu = str2double(elems{2});
    end

    if length(elems)>=3
      if strcmp(curv_surf.type,'cardinal')
        curv_surf.degv = 3;
      else
        curv_surf.degv = str2double(elems{3});
      end
    end
  elseif strcmp(command,'bmat') % BASIS MATRIX
    if strcmpi(elems{2},'u')
      deg = curv_surf.degu;
    elseif strcmpi(elems{2},'v')
      deg = curv_surf.degv;
    end

    mat = zeros(deg+1);
    idx = 3;
    for i=1:deg+1;
      for j=1:deg+1;
        mat(i,j) = str2double(elems{idx});
        idx = idx + 1;
      end
    end

    if strcmpi(elems{2},'u')
      curv_surf.matu = mat;
    elseif strcmpi(elems{2},'v')
      curv_surf.matv = mat;
    end
  elseif strcmp(command,'step') % STEP
    curv_surf.stepu = str2double(elems{2});
    if length(elems)>=3
      curv_surf.stepv = str2double(elems{3});
    end
  elseif strcmp(command,'ctech')  % CURVE APPROX TECHNIQUE
    unimplemented(command,' ');
  elseif strcmp(command,'stech')  % SURFACE APPROX TECHNIQUE
    unimplemented(command,' ');
  else  % NON CURV/SURF ELEMENT - UNREAD
```

```
        doit = 0;
        fseek(fid,pos,'bof');
      end
    end
  end

function strs = getline(fid)
% Lines can be logically joined with the line continuation character ( \ )
% at the end of a line.
  doit = 1;
  strs = {};

  while doit && ~feof(fid)
    line = fgetl(fid);
    line = strtrim(line);
    if ~isempty(line)
      elems = textscan(line,'%s');
      elems = elems{1};

      if strcmp('\', elems{length(elems)})
        elems(length(elems)) = [];
      else
        doit = 0;
      end
      strs = vertcat(strs,elems);
    end
  end

function [ v vt vn ] = getTriplet(str, bundle)
  x = textscan(str,'%n%n%n','delimiter', '/');
  v  = handleNegVert(x{1}, size(bundle.vertex,1));
  if isempty(x{2})
    vt = NaN;
  else
    vt  = handleNegVert(x{2}, size(bundle.vtex,1));
  end
  if isempty(x{3})
    vn = NaN;
  else
    vn  = handleNegVert(x{3}, size(bundle.vnorm,1));
  end

function cs = getCSBodyStatements(fid,cs)
% Body statements are valid only when they appear between the free-form
% element statement (curv, curv2, surf) and the end statement. If they
% are anywhere else in the .obj file, they do not have any effect.
%
% You can use body statements to specify the following values:
%       parameter
%       knot vector
%       trimming loop
%       hole
%       special curve
%       special point
  cs.body = {};

  doit = true;
  while doit && ~feof(fid)
    pos = ftell(fid);
    elems = getline(fid);
    if ~isempty(elems)
      command = lower(elems{1});

      if strcmp(command,'parm')     % PARAMETER
        x = struct;
        x.type = command;
        x.p = zeros(length(elems)-2,1);
        x.p(1) = str2double(elems{3});
        x.p(2) = str2double(elems{4});
        for i=3:size(x.p,1)
          x.p(i) = str2double(elems{i+2});
        end
        if strcmpi(elems{2},'u')
          x.dir = 'u';
        elseif strcmpi(elems{2},'v')
          x.dir = 'v';
        end
        cs.body = [ cs.body ; x ];
      elseif strcmp(command,'trim') || ... % TRIM or
             strcmp(command,'hole') || ... % HOLE or
             strcmp(command,'scrv')        % SPECIAL CURVE
        x = struct;
        x.type = command;
        x.curv = zeros((length(elems)-1)/3,1);
        x.u    = zeros((length(elems)-1)/3,2);
        for i=1:size(x.curv,1)
          x.u(i,1) = str2double(elems{i*3-1});
          x.u(i,2) = str2double(elems{i*3});
          x.curv(i)= str2double(elems{i*3+1});
        end
        cs.body = [ cs.body ; x ];
      elseif strcmp(command,'sp')   % SPECIAL POINT
        x = struct;
        x.type = command;
        x.vp = zeros(length(elems)-1,1);
        for i=1:size(x.vp,1)
          x.vp(i)= str2double(elems{i+1});
        end
        cs.body = [ cs.body ; x ];
      elseif strcmp(command,'end') % END
```

```
          doit = false;
        else  % NON CURV/SURF ELEMENT - UNREAD
          doit = false;
          fseek(fid,pos,'bof');
        end
      end
    end

  function unimplemented(command, file)
    disp(['WARNING: ' command ' was found in file ' file ...
          ' but it was omitted by the parser! ']);

  function v = handleNegVert(v1, v2)
    if v1<0
      v = 1 + v2 + v1;
    else
      v = v1;
    end
```