Wani, Utkarsh
Estes, Tyler

Computational Methods in Additive Manufacturing

MECH-6024, MECH-5124

Group Assignment 6 (Group of 2)

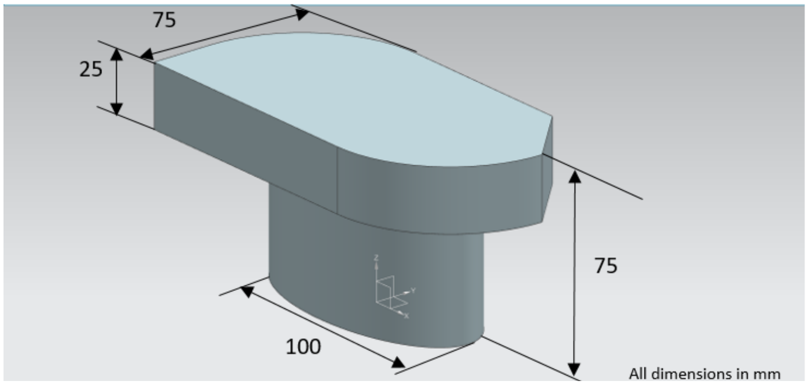
To be worked in groups of two, include both names on the final report. Do not consult other groups.

Please show all steps of the algorithm and calculations

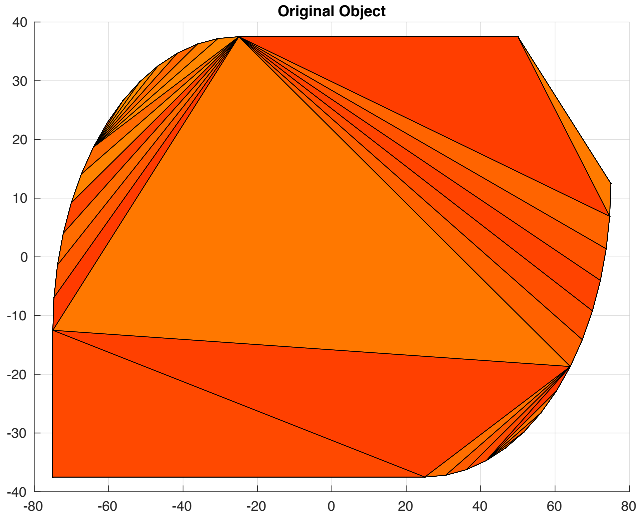
Due April 21, 2020 by 3:30 pm

The STL file of a CAD part is attached on Blackboard (part.stl) along the orientation shown in the figure below. Read the information from STL file. The part is manufactured with a uniform layer thickness of 1 mm. It was decided to change the final build orientation by rotating the part 30^0 with respect to the X-axis followed by a rotation of 45^0 with respect to the Y-axis.

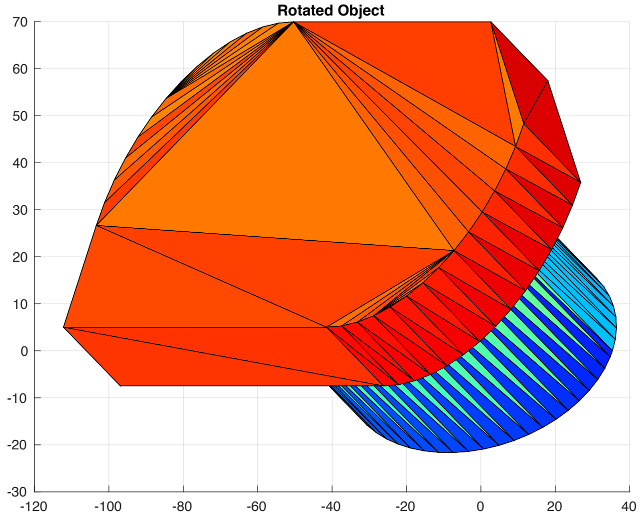
Plot the STL file of the rotated part. For the rotated part, calculate the intersection of a slicing planes with the rotated part at Z = 5mm and Z=30 mm. Provide the consecutive values of the vertices and plot the contour of each sliced layer (polygon). Also, calculate the sintering area (slice hatch area) for each slice and the volume of each slice. Please note that the build direction is always the positive z axis.



Original Object:

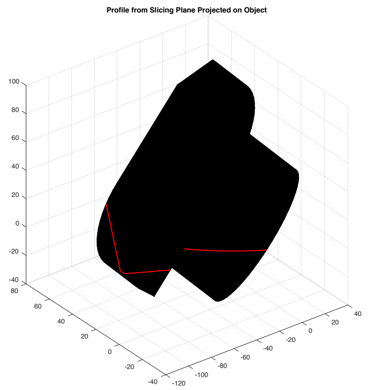


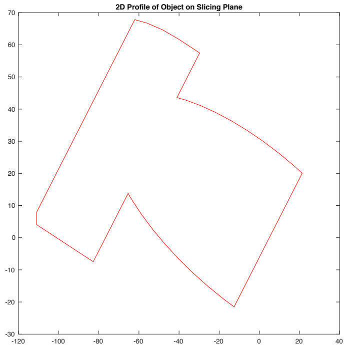
Rotated Object:



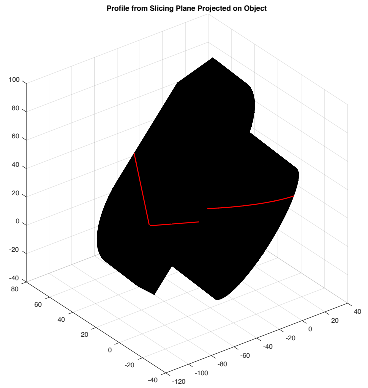
Sliced Objects:

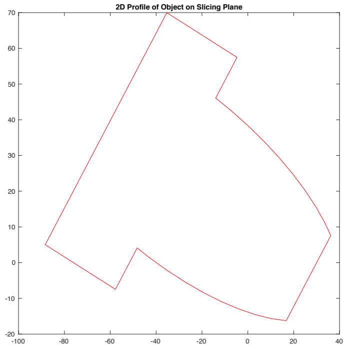
Z = 5 mm





Z = 30 mm





Surface Area:

Z = 5 mm → SA = 8.6557e+03 mm^2
Z = 30 mm → SA = 7.5221e+03 mm^2

Note: During the SA calculation, some of the vectors ($a_i = P_i – P_m$ & $a_{i+1} = P_{i+1} – P_m$) were in the same direction (part of a straight line with P_i, P_{i+1}, and P_m all being on the line) which resulted in a cross product of zero for the i, j, and k directions. However, since they were in the same direction with all the points being on the line, the contribution to the total surface area should be zero we believe. Our program reflects that.

STL slicing algorithm

Step-I

- Generated STL file from NX read in matlab
- changed the build orientation of the part.
- 30° rotation about x-axis and 45° rotation about y-axis.
- plotted the STL file of the rotated part.

Step-II

- To find the maximum and minimum value of z-co-ordinate amongst all the Vertices to know the z-limits for slicing the STL file
- In this case, slicing is performed at the $z = 5$mm and $z = 30$mm

Step-III

- To find all the facets amongst the generated STL file for which slicing height lies in between minimum (z_1, z_2, z_3) and max $(z_1\ z_2\ z_3)$
- If above condition is satisfied, consider this facet for the intersection.
- At this point, no. of facets compared to the generated STL file in the beginning will be lesser and from here on consider only this facets which has satisfied the condition.

·) step IV

for each facet, check for intersection of all the three edges at the slicing height (i.e at $z = 5mm$ and $z \sim 30mm$)

Equation of line has been considered for each edge in a single facet and same is repeated for all the facets.

$$z_{slicing\ plane} = z_1 + t_1 (z_2 - z_1)$$
$$z\ slicing\ plane = z_2 + t_2 (z_3 - z_2)$$
$$z\ slicing\ plane = z_3 + t_3 (z_1 - z_3)$$

where,
t_1, t_2, t_3 are the intersection points for all the three faces respectively.

In this case, for every face we will get a ~~3x3~~ vector of dimension $[1 \times 3]$ which stores t_1, t_2 and t_3 values.

·) step - V

Now, after finding t_1, t_2 and t_3, substitute this values in the equation of line.

Now, every edge will have intersection point r, which is a vector of dimension $[1 \times 3]$.

If $t_1 >= 0$ and $t_1 \leq 1$

$$X\ slicing\ plane\ edge\ 1 = x_1 + t_1 (x_2 - x_1) \quad - ①$$
$$Y\ slicing\ plane\ edge\ 1 = y_1 + t_1 (y_2 - y_1) \quad - ②$$
$$z\ slicing\ plane\ edge\ 1 = z_1 + t_1 (z_2 - z_1) \quad - ③$$

Intersection point r_edge 1 = [①, ②, ③]

This condition is simulated for the other two edges with t_2 and t_3 respectively.
If any interesection point from t_1, t_2 and t_3 is not in between the limit [0-1] then intersection point 'r' for that face is zero.
After executing this, we will get [3x3] matrix of the intersection points per face and amongst the entries in [3x3] matrix, one of the rows will be zero depending on t_1, t_2 and t_3.

Step-VI

In this case, no special case is reported. However, there are two special cases in general

i) Facets with repeated intersection points.

ii) STL facet with one intersection.

Step VII

) All the intersected points in the sequence are arranged to form a contours.

) since the given component does not have any discontinuity, multiple contours won't exist.

) slicing has been performed at the given slicing height

) slicing area is calculated for the slice with the formula,

$$SA_i = \left(\frac{1}{2}\right) \underbrace{\hat{n}}_{\text{unit normal}} (a_i \times a_{i+1}) \quad , \text{ where } a_i = P_i - P_m$$

$m \Rightarrow$ last point

$$\text{Total area} = \left| \sum_{i=1}^{m-2} SA_i \right|$$

Assignment_6.m

```
clc, clear all, close all
%% Step I
[F, V] = stlread('part.stl');

figure()
patch('vertices', V, 'faces', F, 'facevertexcdata', jet(length(F)),
'facecolor', 'flat')
grid on
title('Original Object')

%% Rotation
thetaX = 30;
thetaY = 45;
Rx = [1, 0, 0;
    0, cosd(thetaX), -sind(thetaX);
    0, sind(thetaX),cosd(thetaX)]; % rotation x
Ry = [cosd(thetaY), 0 , sind(thetaY);
    0, 1, 0;
    -sind(thetaY) 0, cosd(thetaY)]; % rotation y
R = Rx*Ry;

for i = 1:size(V, 1)
    V(i, :) = V(i, :)*R;
end

figure()
patch('vertices', V, 'faces', F, 'facevertexcdata', jet(length(F)),
'facecolor', 'flat')
grid on
title('Rotated Object')
%% Step II - Step VII
P5 = slicer(F, V, 5, 1);
P30 = slicer(F, V, 30, 1);
%% Slice Area
SA5 = 0;
m = length(P5);
for i = 1:m-2
    a = [P5(i, :) - P5(m, :);
        P5(i+1, :) - P5(m, :)];
    n = cross(a(1, :), a(2, :));
    if n == zeros(1, 3)                                     % some normal
vectors resulted in [0, 0, 0] because the vectors to P_m were in the same
direction -- they shouldn't contribute to SA
        continue
    end
    u = n / norm(n);
    SA5 = SA5 + (0.5 * dot(u, cross(a(1, :), a(2, :))));
end
SA5 = abs(SA5)

SA30 = 0;
m = length(P30);
for i = 1:m-2
```

```
    a = [P30(i, :) - P30(m, :);
        P30(i+1, :) - P30(m, :)];
    n = cross(a(1, :), a(2, :));
    if n == zeros(1, 3)                                     % some normal
vectors resulted in [0, 0, 0] because the vectors to P_m were in the same
direction -- they shouldn't contribute to SA
        continue
    end
    u = n / norm(n);
    SA30 = SA30 + (0.5 * dot(u, cross(a(1, :), a(2, :))));
end
SA30 = abs(SA30)
```

slicer.m

```
function [p] = slicer(f, v, z_slice, th, show)
%slicer
%   Inputs:
%       f (faces) [3xn array] - indexes into the vertex array (ex.
%       [4, 6, 7] -> 4th, 6th, and 7th values of vertexes `v`
%       v (vertexes) [3xn array] = [x, y, z] values used to compose
%       faces `f`
%       z_slice (z height) [uint] - the z height to slice at
%       th (thickness) [uint] - thickness of slice in mm
%       show [bool] - whether or not to plot
%   Outputs:
%       p (points) [3xn array] - the points on z slice plane

%% Initialize
if nargin < 4
    error('not enough arguments')
elseif nargin == 4
    show = true;
end

%% Step II
z_max = max(v(:, 3));
z_min = min(v(:, 3));
layer_cnt = (z_max - z_min) / th;                           % number of
layers

%% Step III
target_faces = [];
for i = 1:size(f, 1)                                        % for face
    v_i = f(i, :);                                          % get 3 points
(ex. [104, 106, 107])
    z = v(v_i, 3);                                          % get z-val for 3
pts
    if min(z) <= z_slice && max(z) >= z_slice               % if some
vertices of face lower than z_slice and some higher
        target_faces = [target_faces; f(i, :)];             % then that face
is part of slice
    end
end

%% Step IV
t = zeros(size(target_faces, 1), 3);                        % preallocate
```

```
for i = 1:size(target_faces, 1)                             % for target_face
    v_i = target_faces(i, :);                               % get 3 points
(ex. [104, 106, 107])
    z = v(v_i, 3);                                          % get z-val for 3
pts
    tt = zeros(1, 3);                                       % preallocate
    tt(1) = (z_slice - z(1)) / (z(2) - z(1));               % ratio along
edge 1
    tt(2) = (z_slice - z(2)) / (z(3) - z(2));               % ratio along
edge 2
    tt(3) = (z_slice - z(3)) / (z(1) - z(3));               % ratio along
edge 3
    t(i, :) = tt;                                           % store ratios
along 3 edges for each target_face
end

%% Step V
r = zeros(3, 3, size(target_faces, 1));                     % rows:vertexes,
cols:directions, aisles:faces
for i = 1:size(target_faces, 1)
    v_i = target_faces(i, :);                               % get 3 points
(ex. [104, 106, 107])
    x = v(v_i, 1);                                          % get x-val for 3
pts
    y = v(v_i, 2);                                          % get y-val for 3
pts
    z = v(v_i, 3);                                          % get z-val for 3
pts
    tt = t(i, :);

    if tt(1) >= 0 && tt(1) <= 1
        r(1, 1, i) = x(1) + tt(1) * (x(2) - x(1));
        r(1, 2, i) = y(1) + tt(1) * (y(2) - y(1));
        r(1, 3, i) = z(1) + tt(1) * (z(2) - z(1));
    else
        r(1, :, i) = zeros(1, 3);
    end
    if tt(2) >= 0 && tt(2) <= 1
        r(2, 1, i) = x(2) + tt(2) * (x(3) - x(2));
        r(2, 2, i) = y(2) + tt(2) * (y(3) - y(2));
        r(2, 3, i) = z(2) + tt(2) * (z(3) - z(2));
    else
        r(2, :, i) = zeros(1, 3);
    end
    if tt(3) >= 0 && tt(3) <= 1
        r(3, 1, i) = x(3) + tt(3) * (x(1) - x(3));
        r(3, 2, i) = y(3) + tt(3) * (y(1) - y(3));
        r(3, 3, i) = z(3) + tt(3) * (z(1) - z(3));
    else
        r(3, :, i) = zeros(1, 3);
    end
end

%% Step VI
for i = 1:size(r, 3)                                        % for face
indexes
    vv = r(:, :, i);                                        % rows:vertexes,
cols:directions
```

```
    if all(vv(1, :) == vv(2, :))                            % if two vertexes
(in the same face) are on the slicing plane and equal
        r(1, :, i) = zeros(1, 3);                           % then drop one
of the vertexes (zeros will drop out later)
    elseif all(vv(1, :) == vv(3, :))
        r(1, :, i) = zeros(1, 3);
    elseif all(vv(2, :) == vv(3, :))
        r(2, :, i) = zeros(1, 3);
    end
    if sum(~all(vv, 2)) > 1                                 % if the face
only has one intersection with the slicing plane
        r(:, :, i) = zeros(3, 3);                           % then drop the
face (zeros will drop out later)
    end
end

%% Step VII
rr = zeros(2, size(r, 2), size(r, 3));                      % 3-by-3-by-
points -> 2-by-3-by-points; drop zero-filled row
for i = 1:size(r, 3)
    if r(1, :, i) == zeros(1, 3)
        rr(:, :, i) = r(2:3, :, i);
    elseif r(2, :, i) == zeros(1, 3)
        rr(:, :, i) = r([1, 3], :, i);
    else
        rr(:, :, i) = r(1:2, :, i);
    end
end
r = rr;                                                     % this is an
overwrite; original no longer needed

% contour_groups = {};
contour_faces = [];
face_indx = 1;
while 1
    contour_faces = [contour_faces, face_indx];
    ptToFind = r(end, :, contour_faces(end));
    for i = 1:size(r, 3)
        if length(find(contour_faces == i)) ~= 0            % if this face
has already been used
            continue                                        % then skip it
        end
        if norm(r(1, :, i) - ptToFind) < 1e-5              % if a face's
first value is a match
            face_indx = i;                                  % then set that
face to be the next face index
            break
        elseif norm(r(end, :, i) - ptToFind) < 1e-5        % if a face's
second value is a match
            r_temp = r(1, :, i);                            % then swap the
first and second value
            r(1, :, i) = r(end, :, i);
            r(end, :, i) = r_temp;
            face_indx = i;                                  % and set that
face to be the next face index
            break
        end
```

```
%         if i == size(r, 3)                                  % if there are
no matches
%             contour_groups = {contour_groups{:}, contour_faces};  % then it
must be a new contour
%             contour_faces = [];
%             face_indx = 0; % not sure what to put here
%         end
    end
    if contour_faces(end) == face_indx                      % if face_indx is
never set
        break                                               % then no more
matches -> exit
    end
end

p = zeros(size(r, 1)*size(r, 3), size(r, 2));              % 3D array -> 2D
array (couldn't get reshape to work how I needed)
% for i = 1:size(r, 3)
%     j = (i - 1) * size(r, 1) + 1;                           % i = 1, j = 1
-> 2; i = 2, j = 3 -> 4; etc.
%     p(j:j+size(r, 1)-1, :) = r(:, :, i);
% end
for i = 1:size(contour_faces, 2)
    j = (i - 1) * size(r, 1) + 1;                           % i = 1, j = 1 ->
2; i = 2, j = 3 -> 4; etc.
    p(j:j+size(r, 1)-1, :) = r(:, :, contour_faces(i));
end
for i = 2:size(p, 1)-1
    if norm( p(i, :) - p(i-1, :) ) < 1e-5 || norm( p(i, :) - p(i+1, :) ) <
1e-5
        p(i, :) = zeros(1, 3);
    end
end
p = p(any(p, 2), :);                                        % drop zeros
%% Post
if show == true
    figure()

    subplot(1, 2, 1)
    plot3(p(:, 1), p(:, 2), p(:, 3), 'r', 'linewidth', 1.5), hold on
    patch('vertices', v, 'faces', f, 'facecolor', 'k'), hold off
    title('Profile from Slicing Plane Projected on Object')
    grid on, axis square

    subplot(1, 2, 2)
    plot(p(:, 1), p(:, 2), 'r')
    title('2D Profile of Object on Slicing Plane')
    axis square
end
```