Cylinder-based approximation of 3D objects Documentation

Ahmad M. Belbeisi, Benjamin Sundqvist, Cristian Betancourt, Chaudhry Taimoor Niaz, and the BMW development team

Jan 15, 2022

2D APPROXIMATION

1 Introduction **3**
1.1 Task description 3
1.2 Running the code 3

Index **17**

Students:

- Ahmad M. Belbeisi
- Cristian Saenz Betancourt
- Chaudhry Taimoor Niaz
- Benjamin Sundqvist

Supervised by:

- Giovanni Filomeno, M.Sc.
- Moustafa Alsayed Ahmad, M.Sc.

CHAPTER

ONE

INTRODUCTION

The automotive industry faces increasingly strict regulations concerning fleet CO₂ and pollutant emissions, which require innovative drive systems. To meet these requirements, a fleet containing a mix of conventional, hybrid and purely electric vehicles seems to be one likely answer. To handle the space challenges that the adding of electrical components produce, transmission synthesis tools have been developed. The industry successfully applies them to synthesize transmissions for purely electric, conventional and hybrid powertrains. A full evaluation of the transmission concept, however, requires design drafts. Therefore, automatizing the transmission design process is the content of current research, focusing on a fast generation of design drafts for multiple transmission topologies found by the transmission synthesis.

For the computer-aided optimization of engineering designs, 3D-objects may be approximated for later computational efficiency reasons. For instance, it may be beneficial to inner-approximate objects by coaxial cylinders. This task's goal is a model that uses few cylinders while approximating the main features of the object

1.1 Task description

The first task is to import a 3D geometry from an STL file. This volume should then be approximated by cylinders. All these cylinders need to be parallel. They are defined to be parallel to the y-axis of the given geometry. Furthermore, the shape can be defined as an addition and a subtraction of cylinders. In the following, the added cylinders will be called green, and the subtracted cylinders will be called red. The approximation needs to lie entirely inside the original volume, as this volume should model a construction space for a transmission system. Therefore, it needs to be guaranteed that a point is inside the original volume if it is inside the approximation. The aim is to approximate the shape with as few cylinders as possible while approximating the main features of the geometry well. To evaluate the quality of the approximation, also the volume should be computed and compared to the original volume. The code is tested using multiple different STL-files.

For a further details, please refer to `the project report`.

1.2 Running the code

The simplest way to use *Cylinder-based approximation of 3D objects* is through running the main file This can be achieved by executing the following command:

```
cylinder_approximation_3D.m
```

1.2.1 check_between

src.**check_between**(*point1*, *point2*, *point*)

check_between determines, wether a point (which is assumed to lie on a line defined by points 1 and 2) is located between these 2 points

Inputs:

point1,point2 defines the line

point point on that line, that is checked

Outputs:

is_between boolean value, if the point is in between

1.2.2 check_intersection

src.**check_intersection**(*end1*, *end2*, *center*, *radius*)

check_intersection determines, wether a line defined by the points end1 and end2 overlaps with a circle defined by the center point and the radius

Inputs:

end1,end2 defines the line

center,radius defines the circle

Outputs:

intersect boolean value, if there is an intersection

1.2.3 compute_area_MC

src.**compute_area_MC**(*X*, *Y*, *radii*, *X_red*, *Y_red*, *radii_red*, *bounds_x*, *bounds_y*)

compute_area computes the area of a shape, which is defined by the addition of a set of (green) circles and subtraction of a set of (red) circles. It computes the area using a Monte Carlo algorithm. It creates some random points that follow a uniform distribution inside a bounding-box. Then, it tests for each point, if it is part of the geometry. Then, it computes the area by computing the fracition of points, which lie inside the geometry and multiplying it with the area of the bounding box.

Inputs:

X,Y,radii vectors of center-coordinates and radii of circles, which are combined to a shape

X_red,Y_red, radii_red vectors of center-coordinates and radii of circles, which are subtracted from the shape

bounds_x,bounds_y bounding box, which lies around the given shape.

Outputs:

area approximated area of the given shape

1.2.4 compute_area3

src.**compute_area3**(*X*, *Y*, *radii*, *X_red*, *Y_red*, *radii_red*)

compute_area computes the area of a shape, which is defined by the addition of a set of (green) circles and subtraction of a set of (red) circles. It computes the area by approximating the given circles by polygons and computing the area of that polygon. By default, it returnes total_area = 0 (If the polygon-approximation cannot be created)

Inputs:

X,Y,radii vectors of center-coordinates and radii of circles, which are combined to a shape

X_red,Y_red, radii_red vectors of center-coordinates and radii of circles, which are subtracted from the shape

Outputs:

area approximated area of the given shape

1.2.5 convert_polyshape

src.**convert_polyshape**(*polygon*)

convert_polyshape converts polyshape objects into another format for polygons. In polyshape objects, the points at the boundaries of interior regions are always sorted clockwise and boundaries of holes always sorted counterclockwise The polygons will be transformed into another format: Two lists of points are used to define the start and endpoints of each edge of the polygon.

Inputs:

polygon a polyshape-object, that should be transformed

Outputs:

P array of all points of the polygon

P_end array of points, that has the same length as P. Together with P, this defines all edges of the polygon in the new data-structure.

1.2.6 create_circles

src.**create_circles**(*polygon*, *max_number_circles*, *red_radius_factor*)

create_circles approximates a 2D-polygon by adding (green) and subtracting (red) circles. The resulting shape needs to lie completely inside the given polygon. The green circles, which are added, are distributed uniformly around the perimeter of the polygon. Red circles are subtracted from edges, that lie on the convex-hull of the polygon. By that, straight edges are approximated with high accuracy.

Inputs:

polygon The polygon, that should be approximated

max_number_circles The number of green circles, that is distributed uniformly around the perimeter of the polygon. It is the maximum number, as the creation of some circles might fail.

red_radius_factor The largest possible radius of red circles is determined by the larges possible radius of green circles multiplied by this factor.

Outputs:

X,Y,radii vectors of center-coordinates and radii of circles, which are combined to form a shape

X_red,Y_red, radii_red vectors of center-coordinates and radii of circles, which are subtracted from the resulting shape

1.2.7 create_polyshape

src.**create_polyshape**(*X*, *Y*, *radii*, *X_red*, *Y_red*, *radii_red*, *n_sides*)

create_polyshape defines a polygon, that approximates a 2D combination of circles. All circles are approximated by regular n-sided polygons. Then, some of the circles are added and the others are subtracted. That forms the final 2D-polygon.

Inputs:

X,Y,radii vectors of center-coordinates and radii of circles, which are combined to a single shape

X_red,Y_red, radii_red vectors of center-coordinates and radii of circles, which are subtracted from the shape

n_sides All circles are approximated by polygons with this number of sides.

Outputs:

polygon the final shape approximated as a polygon given as a polyshape-object

polygon_red The combination of all circles, which are subtracted. This is used to plot the subtracted region.

1.2.8 find_lines_on_hull

src.**find_lines_on_hull**(*P*, *P_end*)

find_lines_on_hull returns lines of a polygon, which are also part of the

sconvex hull of that polygon.

Inputs:

P array of all points of the polygon

P_end array of points, that has the same length as P. Together with P, this defines all edges of the polygon.

Outputs:

lines_on_hull logical array, which is one, if the corresponding line lies on the convex hull

inner_point average point of the convex hull, which is guaranteed to lie inside the convex hull

1.2.9 polygon_approximation_2D_combined

src.**polygon_approximation_2D_combined**()

Approximate 2D-polygon by circles (red and green) Test before 3D-code worked Used to produce illustrations for the presentations

1.2.10 remove_circles_proximity

src.**remove_circles_proximity**(*radii*, *X*, *Y*, *radii_stay*, *X_stay*, *Y_stay*, *accuracy_factor*)

remove_circles_proximity deletes circles, which have centers very close to each other. Some given circles may be removed. Some other given circles always remain. The circles are removed, if the center is very close to another circle. Than, only the larger one remains.

Inputs:

X,Y,radii vectors of center-coordinates and radii of circles. Some of these circles will be removed.

X_stay,Y_stay,radii_stay vectors of center-coordinates and radii of green circles. None of these circles will be removed.

accuracy_factor Circles are concidered to be very close to each other, if the distance is less than this factor multiplied with the radius.

Outputs:

X,Y,radii vectors of center-coordinates and radii of circles. These are all circles, that remain after some others have been removed.

1.2.11 remove_circles

src.**remove_circles**(*radii*, *X*, *Y*, *radii_red*, *X_red*, *Y_red*, *radii_stay*, *X_stay*, *Y_stay*, *min_area_remain*, *max_area_removed*)

remove_circles deletes circles, which do not contribute much to the final area of a given shape. The shape consists of the addition of (green) circles and the subtraction of (red) circles. Some green circles may be removed. Some other green circles always remain. The circles are only removed, if the area after the removal compared to the initial area retains a minimum value and if the difference of the areas before and after the removal of a circle is sufficiently small.

Inputs:

X,Y,radii vectors of center-coordinates and radii of circles, which are combined to a single shape. Some of these circles will be removed.

X_red,Y_red, radii_red vectors of center-coordinates and radii of circles, which are subtracted from the shape

X_stay,Y_stay,radii_stay vectors of center-coordinates and radii of green circles, which are added to the shape. None of these circles will be removed.

min_area_remain At least this fraction of the initial area has to remain after the circles are removed.

max_area_removed Each circle, that is removed, may only decrease the area by this fraction of the initial area.

Outputs:

X,Y,radii vectors of center-coordinates and radii of circles, which are combined to a single shape. These are all circles, that remain after some others have been removed.

1.2.12 test_inside

src.**test_inside**(*X*, *Y*, *radii*, *X_red*, *Y_red*, *radii_red*, *x*, *y*)

is_inside tests, if a given point lies inside a shape, that is defined by circles. The shape is defined by the addition of (green) circles and the subtraction of (red) circles.

Inputs:

X,Y,radii vectors of center-coordinates and radii of circles, which are combined to a single shape

X_red,Y_red, radii_red vectors of center-coordinates and radii of circles, which are subtracted from the shape

x,y coordinates of a point, which is tested

Outputs:

is_inside logical value, that is 1, if the point lies inside the given shape

1.2.13 create_cylinders

src.**create_cylinders**(*polygon_list*, *y_values*, *number_circles_per_section*, *red_radius_factor*, *remove_circle_parameters*)

create_cylinders approximates a geometry by cylinders. The original geometry is not given, instead, it is divided into several sections between certain y-values. Between any 2 of these y-values, the geometry is defined by 2D-polygons. If all 2D-polygons are extruded between the 2 corresponding y-values, that gives the geometry. So at the end, that geometry is approximated by cylinders. This function uses the 2D-algorithm (approximation of all 2D-polygons with circles) in order to approximate each section of the geometry with cylinders. If possible, it reuses cylinders from previous sections (from left to right) In the resulting approximation, one set of cylinders is added and another set is subtracted, in order to form the geometry.

Inputs:

polygon_list list of 2D-polygons, which fit into a certain geometry that should be approximated

y_values array of length polygon_list + 1, stores all y_values between which the 2D-polygons span the original geometry

number_circles_per_section,red_radius_factor Used in the 2D-code, which approximates polygons by circles. (See create_circles)

remove_circle_parameters Used for the functions remove_circles_proximity and remove_circles

Outputs:

cylinders list of all cylinders, which are added to the geometry. It consists of the coordinates for both endpoints of the cylinders and the radius

cylinders_red list of all cylinders, which are subtracted from the geometry. Same structure, as cylinders

1.2.14 create_sections_initial

src.**create_sections_initial**(*F*, *V*, *N*, *number_of_sections*, *area_percentage_parallel*, *ends_offset_fraction*)

create_sections_initial generates y-values, and cuts the geometry into according sections. The cuts are performed at x-z-planes at the corresponding y-values. It chooses the y-values at both ends of the geometry. Furthermore, it places cuts at y-values, where many triangles are parallel to the y-plane. At the end, it defines the y-values, such that the thickness of each section is at most delta = (ymax-ymin)/number_of_sections

Inputs:

F,V,N faces, vertices and normal-vectors of the given geometry

number_of_sections minimum number of sections

area_percentage_parallel if some triangles at a certain y-value are parallel to the x-z-plane and their area is at least this fraction of the total crossectional area, then this y-value is chosen as a cutting-plane

ends_offset_fraction if it is not possible to define a polygon at one end of the geometry, the end of the geometry is cut from the rest. This parameter influences, how much of the geometry is cut away

Outputs:

mesh_list The geometry of each section, stl-like-datastructure. It consists of faces, vertices and normal vectors

y_values array of length mesh_list + 1, stores all y_values between the sections and at the ends of the complete geometry

F_return,V_return,N_return The faces, vertices and normal vectors of the complete geometry. It only changes from the input, if one of the ends has been cut away.

1.2.15 create_sections

src.**create_sections**(*F*, *V*, *N*, *number_of_sections_or_vector*)

create_sections cuts the given geometry into several sections. If a number is given, it splits the geometry into that number of sections with uniform thickness. If a vector of y-values is given, it cuts the geometry at these y-values. The cuts are performed at x-z-planes at the corresponding y-values.

Inputs:

F,V,N faces, vertices and normal-vectors of the given geometry

number_of_sections_or_vector if it's a number, the geometry is cut into this number of uniformly thick sections. If it's a vector, these are the y-values, at which the geometry is cut.

Outputs:

mesh_list The geometry of each section, stl-like-datastructure. It consists of faces, vertices and normal vectors

y_values array of length mesh_list + 1, stores all y_values between the sections and at the ends of the complete geometry

1.2.16 cut_the_geometry

src.**cut_the_geometry**(*Finput*, *Vinput*, *Ninput*, *cutvalue*)

cut_the_geometry divides a geometry from STL format into two parts at a value in the Y axis

Inputs:

[Finput],[Vinput],[Ninput] 3 matrices from a STL file

Outputs: 2 geometries in STL-like matrix format:

[FGR],[VGR],[NGR] The green part "GR" is the geometry before the cutvalue

[FRD],[VRD],[NRD] Red part "RD" is the geometry after the cutvalue

STL MATRICES [F...]= Faces matrix [V...]= Vertices matrix [N...]= Normal vectors matrix

1.2.17 cylinder_approximation_3D

src.**cylinder_approximation_3D**()

An stl_file is approximated by parallel cylinders (parallel to the y-axis) the cylinders must not lie outside of the initial geometry

Step 0 import an stl-geometry

Step 1 y_values are chosen, where geometry is cut.

Step 2 the geometry is cut into sections with stl-like datastructure.

Step 3 maximum allowable 2D-polygon is defined for each section

Step 4 some cuts are removed again, if they are not necessary

Step 5 cylinders are created, using a 2D-algorithm with circles and cylinders are reused, whenever possible

1.2.18 define_2D_polygons

src.**define_2D_polygons**(*mesh_list*, *y_values*)

define_2D_polygons creates the maximum possible 2D-polygon for each section. That means, the maximum polygon, such that an extrusion of this polygon in the section lies completely within the geometry

Inputs:

mesh_list The geometry of each section, stl-like-datastructure. It consists of faces, vertices and normal vectors

y_values array of length mesh_list + 1, stores all y_values between the sections and at the ends of the complete geometry

Outputs:

polygon_list list of length mesh_list including all computed 2D-polygons used as input for the function create_circles

y_values the same as the input

1.2.19 define_cut_polygon

src.**define_cut_polygon**(*F*, *V*, *N*, *y_value*, *tol_on_plane*, *tol_uniquetol*, *tol*)

define_cut_polygon defines a 2D-polygon from a geometry and a given cutting-plane. It is assumed, that the given geometry was already cut at that given plane, so it has edges that lie on that plane. These edges are detected as a first step. Then, the edges are ordered into a polygon using graph-functionality.

Inputs:

F,V,N faces, vertices and normal-vectors of the given geometry

y_value the x-z-plane at this y-value is the cutting-plane tol_on_plane,tol_uniquetol,tol: some tolerances

Outputs:

polygon the polygon as a polyshape-object

1.2.20 define_section_polygon

src.**define_section_polygon**(*F*, *V*, *N*, *polygon_left*, *polygon_right*, *y_min_section*, *y_max_section*, *tol_on_plane*)

define_section_polygon creates the maximum possible polygon inside a given geometry. The constraint is, that an extrusion of the polygon along the y-axis through the geometry needs to lie completely inside the geometry. The polygons at the right and left cutting-planes of the geometry are given.

Inputs:

F,V,N faces, vertices and normal-vectors of the given geometry

y_min_section, y_max_section the x-z-plane at these y-values are the cutting-planes at both ends of the given section

polygon_left,polygon_right the polygons at the ends of the section

tol_on_plane a tolerance

Outputs:

polygon the polygon as a polyshape-object

1.2.21 define_triangle_region

src.**define_triangle_region**(*F*, *V*)

define_triangle_region creates a polyshape as the union of several triangles (all in 2D)

Inputs:

F,V faces, and vertices of all triangles

Outputs:

triangle_region final polygon given as polyshape-object

1.2.22 stlVolume

src.**stlVolume**(*V*, *F*, *N*)

stlVolume computes the volume of a geometry. Given a surface triangulation, compute the volume enclosed using divergence theorem. Assumption: Triangle nodes are ordered correctly, i.e.,computed normal is outwards Input: p: (3xnPoints), t: (3xnTriangles) Output: total volume enclosed, and total area of surface Author: K. Suresh; suresh@engr.wisc.edu Adjusted by: Benjamin Sundqvist

1.2.23 stlread

src.**stlread**(*file*)

STLREAD imports geometry from an STL file into MATLAB. FV = STLREAD(FILENAME) imports triangular faces from the ASCII or binary STL file idicated by FILENAME, and returns the patch struct FV, with fields 'faces' and 'vertices'.

[F,V] = STLREAD(FILENAME) returns the faces F and vertices V separately.

[F,V,N] = STLREAD(FILENAME) also returns the face normal vectors.

The faces and vertices are arranged in the format used by the PATCH plot object.

1.2.24 stlReadAscii

src.**stlReadAscii**(*fileName*)

STLREADASCII reads a STL file written in ASCII format V are the vertices F are the faces N are the normals NAME is the name of the STL object (NOT the name of the STL file)

1.2.25 stlReadBinary

src.**stlReadBinary**(*fileName*)

STLREADBINARY reads a STL file written in BINARY format V are the vertices F are the faces N are the normals NAME is the name of the STL object (NOT the name of the STL file)

1.2.26 stlReadFirst

src.**stlReadFirst**(*fileName*)

STLREADFIRST reads any STL file not depending on its format V are the vertices F are the faces N are the normals NAME is the name of the STL object (NOT the name of the STL file)

1.2.27 stlSlimVerts

src.**stlSlimVerts**(*v*, *f*)

STLSLIMVERTS removes duplicate vertices in surface meshes.

This function finds and removes duplicate vertices.

USAGE: [v, f]=patchslim(v, f)

Where v is the vertex list and f is the face list specifying vertex connectivity.

v contains the vertices for all triangles [3*n x 3]. f contains the vertex lists defining each triangle face [n x 3].

This will reduce the size of typical v matrix by about a factor of 6.

For more information see: http://www.esmonde-white.com/home/diversions/matlab-program-for-loading-stl-files

Francis Esmonde-White, May 2010

1.2.28 stlWrite

src.**stlWrite**(*filename*, *varargin*)

STLWRITE writes STL file from patch or surface data.

STLWRITE(FILE, FV) writes a stereolithography (STL) file to FILE for a triangulated patch defined by FV (a structure with fields 'vertices' and 'faces').

STLWRITE(FILE, FACES, VERTICES) takes faces and vertices separately, rather than in an FV struct

STLWRITE(FILE, X, Y, Z) creates an STL file from surface data in X, Y, and Z. STLWRITE triangulates this gridded data into a triangulated surface using triangulation options specified below. X, Y and Z can be two-dimensional arrays with the same size. If X and Y are vectors with length equal to SIZE(Z,2) and SIZE(Z,1), respectively, they are passed through MESHGRID to create gridded data. If X or Y are scalar values, they are used to specify the X and Y spacing between grid points.

STLWRITE(...,'PropertyName',VALUE,'PropertyName',VALUE,...) writes an STL file using the following property values:

MODE - File is written using 'binary' (default) or 'ascii'.

TITLE - Header text (max 80 chars) written to the STL file.

TRIANGULATION - When used with gridded data, TRIANGULATION is either:

'delaunay' - (default) Delaunay triangulation of X, Y 'f' - Forward slash division of grid quads 'b' - Back slash division of quadrilaterals 'x' - Cross division of quadrilaterals

Note that 'f', 'b', or 't' triangulations now use an inbuilt version of FEX entry 28327, "mesh2tri".

FACECOLOR - Single colour (1-by-3) or one-colour-per-face (N-by-3) vector of RGB colours, for face/vertex input. RGB range is 5 bits (0:31), stored in VisCAM/SolidView format (http://en.wikipedia.org/wiki/STL_(file_format)#Color_in_binary_STL)

Example 1: % Write binary STL from face/vertex data tmpvol = false(20,20,20); % Empty voxel volume tmpvol(8:12,8:12,5:15) = 1; % Turn some voxels on fv = isosurface(~tmpvol, 0.5); % Make patch w. faces "out" stlwrite('test.stl',fv) % Save to binary .stl

Example 2: % Write ascii STL from gridded data [X,Y] = deal(1:40); % Create grid reference Z = peaks(40); % Create grid height stlwrite('test.stl',X,Y,Z,'mode','ascii')

Example 3: % Write binary STL with coloured faces cVals = fv.vertices(fv.faces(:,1),3); % Colour by Z height. cLims = [min(cVals) max(cVals)]; % Transform height values nCols = 255; cMap = jet(nCols); % onto an 8-bit colour map fColsDbl = interp1(linspace(cLims(1),cLims(2),nCols),cMap,cVals); fCols8bit = fColsDbl*255; % Pass cols in 8bit (0-255) RGB triplets stlwrite('testCol.stl',fv,'FaceColor',fCols8bit)

1.2.29 stlGetFormat

src.**stlGetFormat**(*fileName*)

STLGETFORMAT identifies the format of the STL file and returns 'binary' or 'ascii'

1.2.30 plot_circles

src.**plot_circles**(*radii*, *X*, *Y*, *radii_red*, *X_red*, *Y_red*, *y_values*)

plot_circles plots 2D-circles in a 3D-space by approximating them as polygons It includes circles, which are added (green) and subtracted (red). The circles are plotted at every y-plane of the given y-values. If exactly 2 y-values are given, also the also side-faces are plotted, such that the resulting plot gives a closed 3D-geometry. As the area can be computed quickly from these polygon-approximations, also the area of the shape is computed and returned.

Inputs:

X,Y,radii vectors of center-coordinates and radii of circles, which are combined to a single shape

X_red,Y_red, radii_red vectors of center-coordinates and radii of circles, which are subtracted from the shape

y_values at these y_values, the 2D-geometries are plotted in the according x-z-planes

Outputs:

area_section area of the given shape

1.2.31 plot_cylinders

src.**plot_cylinders**(*cylinders*, *cylinders_red*, *y_values*)

plot_cylinders plots some parallel cylinders in a 3D-space by approximating the corresponding circles as polygons. It includes cylinders, which are added (green) and subtracted (red). Between 2 y-values (in one section), there are always the same cylinders. As the area of each crossection is given in the process also the volume of the cylinder-geometry is computed and returned.

Inputs:

cylinders list of all cylinders, which are added to the geometry. It consists of the coordinates for both endpoints of the cylinders and the radius

cylinders_red list of all cylinders, which are subtracted from the geometry. Same structure, as cylinders

y_values between these y-values, the crossections are constant

Outputs:

volume_approximated volume of the given cylinder-geometry

1.2.32 plot_STL

src.**plot_STL**(*V*, *F*, *color*)

plot_STL plots a geometry, which is given in an stl-like datastructure. It either plots a mesh, if the color is "none", or it plots filled faces, if the color is "yes".

Inputs:

V,F vertices and faces of the given geometry

color if "none", only the edges are plotted, if "yes", the faces are filled with color

Outputs: :no outputs

1.2.33 Random circle packing:

1.2.34 Collins and Stephenson Circle Packing:

1.2.35 Books on optimization problems:

1.2.36 Codes to read/write STL:

1.2.37 Code to define-2D-polygon:

1.2.38 Code for stl-volume:

INDEX

C

check_between() (*in module src*), 4
check_intersection() (*in module src*), 4
compute_area3() (*in module src*), 5
compute_area_MC() (*in module src*), 4
convert_polyshape() (*in module src*), 5
create_circles() (*in module src*), 5
create_cylinders() (*in module src*), 8
create_polyshape() (*in module src*), 6
create_sections() (*in module src*), 9
create_sections_initial() (*in module src*), 9
cut_the_geometry() (*in module src*), 10
cylinder_approximation_3D() (*in module src*), 10

D

define_2D_polygons() (*in module src*), 10
define_cut_polygon() (*in module src*), 11
define_section_polygon() (*in module src*), 11
define_triangle_region() (*in module src*), 12

F

find_lines_on_hull() (*in module src*), 6

P

plot_circles() (*in module src*), 14
plot_cylinders() (*in module src*), 14
plot_STL() (*in module src*), 15
polygon_approximation_2D_combined() (*in module src*), 7

R

remove_circles() (*in module src*), 7
remove_circles_proximity() (*in module src*), 7

S

stlGetFormat() (*in module src*), 14
stlread() (*in module src*), 12
stlReadAscii() (*in module src*), 12
stlReadBinary() (*in module src*), 12
stlReadFirst() (*in module src*), 13
stlSlimVerts() (*in module src*), 13
stlVolume() (*in module src*), 12
stlWrite() (*in module src*), 13

T

test_inside() (*in module src*), 8