

In [2]:

```
from openscad1 import *
# set_printoptions(suppress=True)
# from IPython.display import display, HTML
# display(HTML("<style>.container { width:100% !important; }</style>"))
```

Table of Contents (alphabetic ordered)

- **2W-wheel** : 2-wheeler wheel completely designed in openscad
- **3d-knots** : various interesting 3d knots
- **4W-wheel** : 4-wheeler wheel concept designed in openscad
- **align_sol_1(sol)**: same as align_sec_1 but for a solid with multiple slices
- **align_sol(sol,ang=10)** : same as align_sec but for solid with multiple slices
- **ang_2lineccw(p0,p1,p2)** : ccw angle of the line p0p2 from base line p0p1
- **ang_2linecw(p0,p1,p2)** : cw angle of the line p0p2 from base line p0p1
- **ang(x,y)**: calculates the angle with base(x) and perpendicular(y) distances. if x<0 and y>0 the angle will be 90<x<180
- **arc_2p_3d_cp(n1,p0,p1,r,cw=1)** : finds center point of the arc with 2 points and a normal
- **arc_2p_3d(n1,p0,p1,r,cw=1,s=20)** : draws an arc with 2 points and normal in 3d space
- **arc_2p_cp(p1,p2,r,cw=-1)** : calculates the center point of an arc based on 2 given points, cw or ccw directions are important to calculate the correct center point
- **arc_2p(p1,p2,r,cw=1,s=20)**: calculates arc with 2 given points.cw or ccw will change the arc, also this generates the shortest distance arc
- **arc_3d(v=[0,0,1],r=1,theta1=0,theta2=360,cw=-1,s=50)** : calculates the arc in 3d space w.r.t. plane defined by normal vector 'v1', radius 'r1', start and end angles, cw or ccw and number of segments 's'
- **arc_3p_3d(points,s=20)** : draws an arc through the 3 points list. 's' is the number of segments of the circle
- **arc_3p(p1,p2,p3,s=30)**: function to draw arc with 3 known points 'p1','p2','p3'. 's' is the number of segments of the arc
- **arc_long_2p_3d(n1,p0,p1,r,cw=1,s=20)** : draws a long arc with 2 points and normal
- **arc_long_2p(p1,p2,r,cw=1,s=20)**: long arc with 2 points 'p1,p2' with radius 'r' and with orientation clockwise (1) or counterclockwise(-1)
- **arc(radius=0,start_angle=0,end_angle=0,cp=[0,0],s=20)**: function for calculating 2d arc. 'cp': center point of the arc. 's': number of segments in the arc
- **artifact** :
- **axis_rot_1(sol,ax1,loc1,theta)**: rotate a solid on any pivot point 'loc1' with axis of rotation 'ax1' by an angle 'theta'
- **axis_rot_o(axis,solid,angle)**: rotate a solid around an axis considering the solid is centered at origin
- **axis_rot(axis,solid,angle)**: rotate a solid around an axis
- **back-camera-clamp** :
- **ball-bearing** :
- **bb(prism)**: function to find the bounding box dimensions of a prism
- **bezier(p,s=10)**: bezier curve defined by points 'p' and number of segments 's'
- **bottle-with-cut-design** :
- **bspline_cubic(px,s=10)** : draws a cubic bspline curve for the given control points. 's' defines the number of points in a bspline curve as (len(px)-2)*s
- **business-card-holder** :
- **c_hull(pnt)**: function to calculate convex hull for a list of points 'pnt'
- **c2ro(sol,s)** : change the orientation of points of a cylinder from circular to rectangular orientation. 'sol': is a cylindrical type 3d shape. 's': number of segments required between each straight line segments
- **c2t3(p)** : function to convert 2d to 3d, it just adds the z-coordinate to the points list
- **c3t2(a)**: function to convert 3d to 2d, it just removes the z-coordinate from the points list
- **cam-profile** :
- **chimney-panel-support** :

- **cir_2p(p1,p2,r,cw=1,s=20)** : circle with 2 points 'p1,p2' with radius 'r' and with orientation clockwise (1) or counterclock wise(-1)
- **cir_3p_3d(points,s=20)**: draws a circle through the 3 points list. 's' is the number of segments of the circle
- **cir_3p(p1,p2,p3,s=30)**: function to draw circle with 3 known points 'p1','p2','p3'. 's' is the number of segments of the circle
- **cir_p_t(cir,p)**: circle to point tangent line (point should be outside the circle)
- **circle(r,cp=[0,0],s=50)** : function for creating points in circle with radius "r", center point "cp" and number of segments "s"
- **coil-example :**
- **concave_hull(p_l,k=3)**:calculate the concave hull of a random list of points. larger number for 'k' will give smoother shape
- **cone-through-parameteric-equation :**
- **convert_3lines2fillet_closed(pnt1,pnt2,pnt3,f=1.9,s=10)** : Develops a fillet with 3 list of points (pnt1,pnt2,pnt3) in 3d space, f: is a factor which can be reduced to 1.5 in case of self intersection observed, s: number of segments in the fillet, increase the segments in case finer model is required
- **convert_3lines2fillet(pnt1,pnt2,pnt3,f=1.9,s=10)** : Develops a fillet with 3 list of points (pnt1,pnt2,pnt3) in 3d space, f: is a factor which can be reduced to 1.5 in case of self intersection observed, s: number of segments in the fillet, increase the segments in case finer model is required
- **convert_secv(sec)** : function removes all the radiiuses from the section 'sec' where points are ccw
- **convert_secv1(sec)** : function removes all the radiiuses from the section 'sec'
- **convert_secv2(sec,d)** : function removes all the radiiuses from the section 'sec' where points are cw. 'd' should be > the maximum radius at cw corner to give the right results
- **convex_hull(pnts)**: calculates convex hull for a list of points 'pnts'
- **convex(sec)** : function to check whether a section is convex or not
- **corner_radius(sec,s)** : function to create section with corner radiiuses)
- **cosinewave(l,n,a,p)** : creates a cosinewave with length 'l', number of cycles 'n' amplitude 'a' and number of points 'p'
- **cp_3p(p1,p2,p3)** : function to calculate center point of a circle created from 3 known points 'p1','p2','p3'
- **cp_arc(arc1)** : function returns the center point of a given circle or arc
- **cp_cir_3d(cir)** : center point of circle with atleast 3 known list of 'points' in 3d space
- **cpo(prism)** : function to change the orientation of the points of the prism
- **cr_3d(p,s=5)** : Corner radius 3d where 'p' are the list of points (turtle movement) and 's' is number of segments for each arc
- **cs1(sec,d)**: creates a cleaning section for removing excess points for offseting a section 'sec' with offset distance 'd'
- **cube(s,center=False)** : function to draw cube with size 's'
- **cut_plane(nv=[0,0,1],size=[5,5],thickness=10,trns1=0,trns2=0,trns3=0)** : function for defining a solid (cutting plane) oriented as per the defined normal vector, nv: normal vector for defining plane orientation of the section, thickness: thickness or height of the cutting plane, trns1: translate the solid in the direction of normal vector 'nv', trns2: translate the solid in the direction 'right' to the normal vector 'nv', trns3: translate the solid in the direction 'up' to the normal vector 'nv', '-ve' values given to the trns1,trns2,trns3 will translate the solid in the reverse direction
- **cw(sec)** : function to identify if an enclosed section is clockwise(cw) or counterclockwise(ccw). this returns 1 if section is clockwise and -1 if it is counterclockwise
- **cwv(sec)** : function to identify whether each point in a section is clockwise or counter clockwise.
- **cylinder(r1=1,r2=1,h=1,cp=[0,0],s=50,r=0,d=0,d1=0,d2=0,center=False)** : function for creating cylinder, cone
- **cylinder-with-rectangular-pocket :**
- **cylinder-with-star-pocket :**
- **cytz(path)**: converts 'y' points to 'z' points in a 2d list of points
- **drill-bit :**
- **drill-case :**
- **e_wave(l,a,w,t)** : create a graph of exponential function $a \cdot e^{-(wt)}$ where, w: omega, t: time steps, a: amplitude, l: length of time

- **end_cap(sol,r=1,s=10)** : function to draw radius at the ends of 'path_extrude_open' models. sol: path extruded solid, r: radius at the ends, s: segments of the radius
- **equidistant_path(path,s=10)** : divides a path in to equally spaced points
- **equidistant_pathc(path,s=10)** : divides a closed path in to equally spaced points
- **equivalent_rot_axis(r1=[])** : function returns an equivalent axis for rotation and angle of rotation for a sequence of rotations given by list 'r1'
- **example-of-rounding :**
- **exclude_points(list1,list_to_exclude)** : function to remove points from a list
- **f_prism(sec,path)** : creates a solid or prism with a 2d section and a 2d path. This is a much faster version of function prism, but may not work with few shapes correctly)
- **faces_1(l,m)** : calculate the faces for the vertices with shape l x m with first and the last end open
- **faces(l,m)** : calculate the faces for the vertices with shape l x m with first and the last end closed
- **fillet_2cir(r1,r2,c1,c2,r,s=50)** : fillet between 2 circles with radius 'r1' and 'r2' and center points 'c1' and 'c2' and 'r' is the radius of the fillet
- **fillet_3p_3d_cp(p0,p1,p2,r)** : center point 'cp' of the fillet with 3 known points 'p0,p1,p2' in 3d space. 'r' is the radius of fillet
- **fillet_3p_3d(p0,p1,p2,r,s)** : fillet with 3 known points 'p0,p1,p2' in 3d space
- **fillet_l_cir(line=[],cir1=[],fillet_radius=1,s=20)** : function to draw fillet between a line and a circle
- **filletto_2cir(r1,r2,c1,c2,r,s=50)** : fillet between 2 circles with radius 'r1' and 'r2' and center points 'c1' and 'c2' and 'r' is the radius of the fillet. This is an open fillet where first or the second fillet can be called based on requirement
- **flip(sec)** : flips the sequence of a list or list of points
- **gcd(a,b)** : calculates greatest common divisor for 2 numbers
- **glass-model :**
- **handling-trolley :**
- **helix(radius=10,pitch=10, number_of_coils=1, step_angle=1)** : creates helix
- **honeycomb(r,n1,n2)** : function to draw a honeycomb structure with radius 'r', n1: number of hexagons in 1 layer, n2: number of layers
- **i_line_fillet(sol1,sol2,ip,r1,r2,s=20,o=0)** : calculates a fillet at the intersection of 2 solids when the intersection points 'ip' are separately defined. r1 and r2 would be same in most of the cases, but the signs can be different depending on which side the fillet is required. r1 is the distance by which intersection line offsets on sol2 and similarly r2 is on sol1
- **i_line_planes(p1,p2)**: intersection line between 2 planes 'p1' and 'p2'
- **i_p_n(sol,i_p)** : calculates normal at the intersection points, sol: solid on which the normal is required, i_p: list of intersection points between 2 solids
- **i_p_p(sol,i_p,r)** : function to project the intersection point on the cutting lines based on the distance 'r'
- **i_p_t(path)** : function to calculate tangent vectors to a given path
- **intersections(segments)** : calculates the intersections of adjacent line segments only
- **ip_fillet(sol1,sol2,r1,r2,s=20,o=0)** : calculates a fillet at the intersection of 2 solids. r1 and r2 would be same in most of the cases, but the signs can be different depending on which side the fillet is required. r1 is the distance by which intersection line offsets on sol2 and similarly r2 is on sol1
- **ip_sol2line(sol,line)** : function to calculate intersection point between a 3d solid and a line. "sol" is the 3d object which is intersected with a "line". in case there are more than 1 intersections of the line with the solid/ prism ip(sol,line)[0] will give the first intersection point and ip(sol,line)[-1] will give the last intersection point
- **ip_sol2sol(sol,sol1,i=0)** : function to find the intersection point between 2 solids. this function is to be used where the cutting lines of sol1 are intersecting sol at more than 1 times.sol: solid which is intersected. sol1: this intersects the solid 'sol'. i: if the first intersection points of all the cutting lines are to be considered, value of i should be '0'. if the last intersection points of all the cutting lines are to be considered, value of 'i' should be set to '-1'. if all the intersection points are required, value of 'i' should be set to 'all'
- **ip_triangle(sol1,p0)** : finds the triangle where the intersection point "p0" lies in a solid "sol1"
- **ip(prism,prism1,side=-1)**: function to calculate intersection point between two 3d prisms. "prism" is the 3d object which is intersected with "prism1". side: when a ray intersects a solid it can intersect at 2 locations, if the ray is travelling from outside, in that case if '0' is given meaning only the first intersection point is considered, and in case '-1' is given meaning the last intersection point will be considered.)

- **iterative-approach-towards-creating-fillets :**
- **l_cir_ip(line,cir)** : line circle intersection point
- **l_len(l)** : calculates length of a line 'l'
- **l_lenv_o(l)** : calculates sum of lengths of all the segments in a line 'l' considering the section is open
- **l_lenv(l)** : calculates sum of lengths of all the segments in a line 'l' considering the section is closed
- **l_sec_ip_3d(sec,line)** : finds the intersection points between a section in 3d space and a line in 3d space
- **l_sec_ip(line,sec)** : line and section intersection point
- **I2I_intersection(I1,I2)** : function to find the intersection point between 2 lines in 3d space
- **lamp :**
- **lcm** : calculates least common multiple for 2 numbers
- **lexicographic_sort_xy(p)** : function sorts the points list 'p' first with x and then with y smallest to largest
- **lexicographic_sort_yx(p)** : function sorts the points list 'p' first with y and then with x smallest to largest
- **linear_extrude(sec,h=1,a=0,steps=1)** : function to linear extrude a section where, se/users/sanjeevprabhakar section to extrude, h: height of the extrusion, a: angle of twist while extruding, steps: number of steps in each angular extrusion
- **list_r(sec)** : function list the corner radiiuses of a given section (only where the radius is specified)
- **ls(line,n)**: plots 'n' points in a straight line)
- **m10 :**
- **m35 :**
- **m39 :**
- **max_r(sec)** : finds the maximum radius of a given closed section)
- **mobile-phone-stand :**
- **multiple_sec_extrude(path_points=[],radiiuses_list=[],sections_list=[],option=0,s=10):**
path_points: are the points at which sections needs to be placed,radiiuses: radius required at each path_point. this can be '0' in case no radius required in the path, sections_list= list of sections required at each path_points. same section can be provided for various path_points as well, option: can be '0' in case the number of points in each section do not match or '1' in case number of points for each section are same, s: in case value of radiiuses is provided 's' is the number of segments in that path curve
- **normals_along_path :**
- **nv(p)** : finds the normal vector for a given points in a 3d plane
- **o_3d(i_p,sol,r,o=0)** : function to offset the intersection points 'i_p' on a solid 'sol' by distance 'r'. option 'o' can have values '0' or '1' and changes the direction of offset
- **o_p_p(sol,i_p,d)**: calculates projected points on the surface of a solid , sol: solid on which the points to be projected, i_p: list of points in 3d space near the solid, d: approximate distance of the points from the surface, specifying too big distance , may create multiple projection of the same point on the solid
- **o_solid(nv=[0,0,1],sec=[],thickness=10,trns1=0,trns2=0,trns3=0, theta=[0,0,0])** : function for defining a solid with any defined section. solid gets oriented as per the defined normal vector, nv: normal vector for defining plane orientation of the section, se/users/sanjeevprabhakar cross section of the solid, thickness: thickness or height of the solid, trns1: translate the solid in the direction of normal vector 'nv', trns2: translate the solid in the direction 'right' to the normal vector 'nv', trns3: translate the solid in the direction 'up' to the normal vector 'nv', -ve values given to the trns1,trns2,trns3 will translate the solid in the reverse direction , theta: rotate the section around axis fox example if nv is [1,0,0] or x-axis, the sequence of rotation will be x, y ,z axis
- **offset_3d(sec,d)** : offsets an enclosed section in 3d space, in case the section is in 1 plane, se/users/sanjeevprabhakar section in 3d space, d: offset distance -ve sign means inner offset and +ve sign is outer offset
- **offset_points_ccw(sec,r)**: function to offset only those points which are counter clockwise
- **offset_points_cw(sec,r)** : function to offset only those points which are clockwise
- **offset_points(sec,r)** : function to calculate offset of a list of 2d points. in defining sections, providing corner radius is a must
- **offset_seg_cw(sec,r)** : function offsets the segment only when the point is clockwise
- **offset_segv(sec,d)** : function makes the segments of the original section and offset each segment by a distance 'd'
- **offset(sec,r)** : calculates offset for a section 'sec' by amount 'r'
- **orthos_along_path(path,scale=1) :**

- **p_cir_t(p,cir)** : point to circle tangent line (point should be outside the circle)
- **p2p_intersecion_line(pa,pb)** : function to calculate intersection line between 2 planes in 3d space
- **path_extrude_closed(sec,path,twist=0)** : function to extrude a closed section to a closed path. closed path means the path provided has it's first and the last point same example a circle
- **path_extrude_open(sec,path,twist=0)**: function to extrude a closed section to an open path. twist can be set either to '0' or '1' depending on the shape produced
- **path_offset(path,d)** : function to offset a 'path' by 'd' distance
- **pies1(sec,pnts)** : function to find points 'pnts' which are inside an enclosed section 'sec'
- **plane(nv, radius)** : plane defined by normal 'nv' and 'radius'
- **pntsnfaces(bead2)** : function returns points and faces of a prism
- **ppesec(p0,sec)**: point's projection on an enclosed 3d section
- **ppplane(p0,v1,loc)** : function to find projected points of a given list of points 'p0' on a plane defined by normal 'v1' and location 'loc'
- **prism(sec,path)** : function to make a prism with combination of 2d section and 2d path
- **pts(p)** : calculates the cumulative sum of 2d list of points 'p'
- **pts1(p)**: 'p' is a list of points. function calculates the cumulative sum of x,y values in the list while z value remains the same. this is mainly used in function corner_radius(pl,s).
- **q_rot(s,pl)** : function to rotate a group of points "pl" around a series of axis with defined angles
- **q_rot2d(theta,pl)** : function to rotate a 2d point or 2d points list by an angle theta around z-axis
- **q(vector=[1,0,0],point=[0,5,0],theta=0)** : function to rotate a point around a vector(axis) with angle theta
- **r_sec(r1,r2,cp1,cp2)** : creates a rounded section around a line defined by points 'cp1' and 'cp2'. radius around 'cp1' is 'r1' and radius around 'cp2' is 'r2'
- **rounding-various-rounded-cubes** :
- **rsz2d(sec,rsz)** : function to resize a 2d section to dimensions 'rsz'. resized section will be placed on bottom center of the original section
- **rsz2dc(sec,rsz)** : function to resize a 2d section to dimensions 'rsz'. resized section will be placed in center of the original section
- **rsz3d(prism,rsz)** : function to resize a 'prism' to dimensions 'rsz'. bottom left corner of both the prisms would be same
- **rsz3dc(prism,rsz)** : function to resize a 'prism' to dimensions 'rsz'. resized prism will be placed in the center of the original prism or center point of both the prisms will be same
- **s_int(s)** : calulates the self intersection points of a list of line segments 's'. it also picks the points in case the 2 lines are just connected at 1 point and are not crossing
- **s_int1(s)**: calcultes the self intersecting segments such that there is a complete intersection and not only end points touching
- **samsung-tab-s6-holder** :
- **scl2d_c(sec,sl)** : scale a 2d section such that the center of original and scaled section is same)
- **scl2d(sec,sl)**: scale a 2d section to a scaling factor 'sl')
- **scl3d(p,s)** : scale any 3d prism to a scaling factor 's')
- **sec2vector(v1=[1,0,0],sec=[])** : function to align a section 'sec' with a vector 'v1'
- **sinewave(l,n,a,p)** : creates a sinewave with length 'l', number of cycles 'n' amplitude 'a' and number of points 'p'
- **sinwave-box** :
- **sl_int(sec,line)** : function to find intersection between an enclosed section in 3d space and a line
- **slice_sol(sol,n=10)**: function to slice a solid with 'n' intermediate steps
- **sol2path(sol,path)**: function to extrude a solid along a path
- **sol2vector(v1=[],sol=[],loc=[0,0,0])**: orients a solid as per a given vector
- **sort_points(sec,list1)** : function picks the nearest point of a section from a reference section and matches the length of points for the 2 compared sections
- **sphere(r=0,cp=[0,0,0],s=50)** : function to draw sphere with radius 'r' , center point 'cp' and number of segments 's'
- **sphere-through-parameteric-equation** :
- **sunflower** :
- **surf_base(surf,h=0)** : creates a solid from any surface, 'h' is the height of the base of the surface

- **surf_extrude(sec,path)** : extrudes an open section 'sec' to a 3d 'path' to create surface
- **surf_extrude_surf(t=-.05)** : surface with a polyline 2d sketch and a 3d path. thickness of the surface can be set with parameter "t". positive and negative value creates thickness towards +z and -z directions respectively
- **surf_offset(sec,d)** : function to offset the surface 'sec' by a distance 'd'
- **surface_for_fillet(sol1=[],sol2=[],factor1=50,factor2=10,factor3=1,factor4=100,dia=40)**: sol1: Solid on which the surface needs to be created. sol2: Intersecting solid. factor1: number of segments in the circle. factor2: number of layers or slices of surface. factor3: decides the size of the surface lower value means bigger size. value can be set between 1 to any number. factor4: any high number should be ok like maybe 100 or greater, basically greater than the bounding box dimension of the "sol1". dia: diameter around the solid 2 where surfavce needs to be created
- **swp_prism_h(prism_big,prism_small)** : creates a hollow prism with 2 similar prisms (1 big and 1 smaller)
- **t_cir_tarc(r1,r2,cp1,cp2,r,side=0,s=50)** : function draws a arc which is tangent to 2 circles defined by radiiuses 'r1' and 'r2' and center points 'cp1' and 'cp2'. 's' is the number of segments of the tangent arc. 'r' is the radius of the tangent arc (it should be $\geq (r1+r2+\text{center distance of 2 circles})/2$). 'side' there are 2 sides of the circles where the arc could be created defined by '0' and '1'
- **tangents_along_path(path,scale=1)** :
- **tcct(r1,r2,cp1,cp2,cw=-1)** : two circle cross tangent
- **tctp(r1,r2,cp1,cp2)** : 2 circle tangent points (one side) r1 and r2 are the radius of 2 circles and cp1 and cp2 are the center points)
- **tcpf(r1,r2,cp1,cp2)** : 2 circle tangent point full (both the sides))
- **torus-through-parameteric-equation** :
- **translate(p,sec)** : translates a prism or section by [x,y,z] distance
- **translate_2d(p,sec)** : function to translate a group of points "sec" by "p" distance defined in [x,y]
- **v_sec_extrude(sec,path,o)** : extrude a section 'sec' through a path 'path' . section will vary from start to end such that at the end the section will be offset by 'o' distance
- **wrap_around** : complex function see the example to understand

surf_extrude

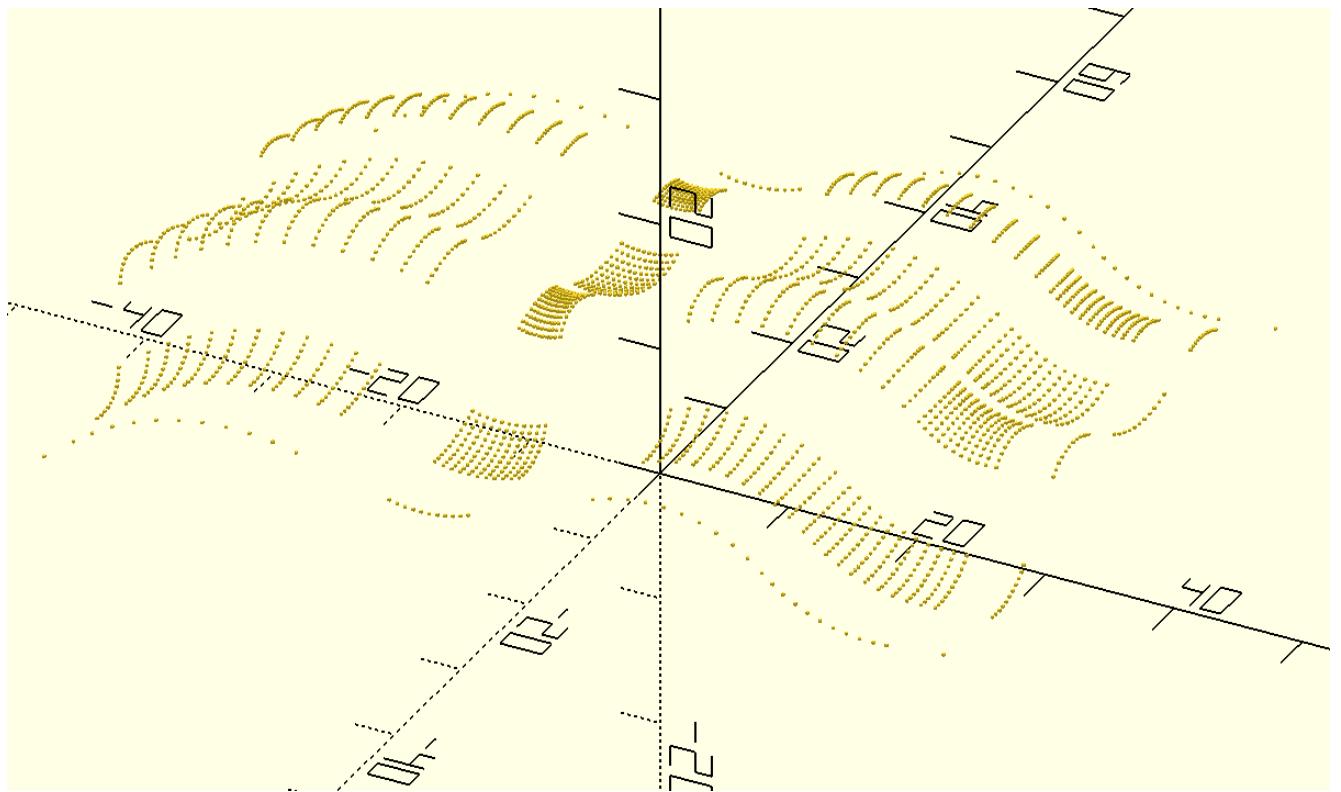
```
In [3]: # example of surf_extrude
t0=time.time()

sec2=corner_radius(pts1([[-25,0],[10,5,5],[10,-3,10],[10,5,5],[10,-8,7],[10,1]]),10)
path2=cytz(corner_radius(pts1([-35,5,0],[10,8,20],[20,-5,10],[20,8,20],[10,-9,20],[10,1,0])),10)

surf2=surf_extrude(sec2,path2)
with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>
for(p={surf2})points(p,.2);
color("blue")points({path2},.5);

'''')
t1=time.time()
total=t1-t0
total
```

Out [3]: 0.05582284927368164



surf_extrudef

```
In [4]: # example of surf_extrudef
t0=time.time()

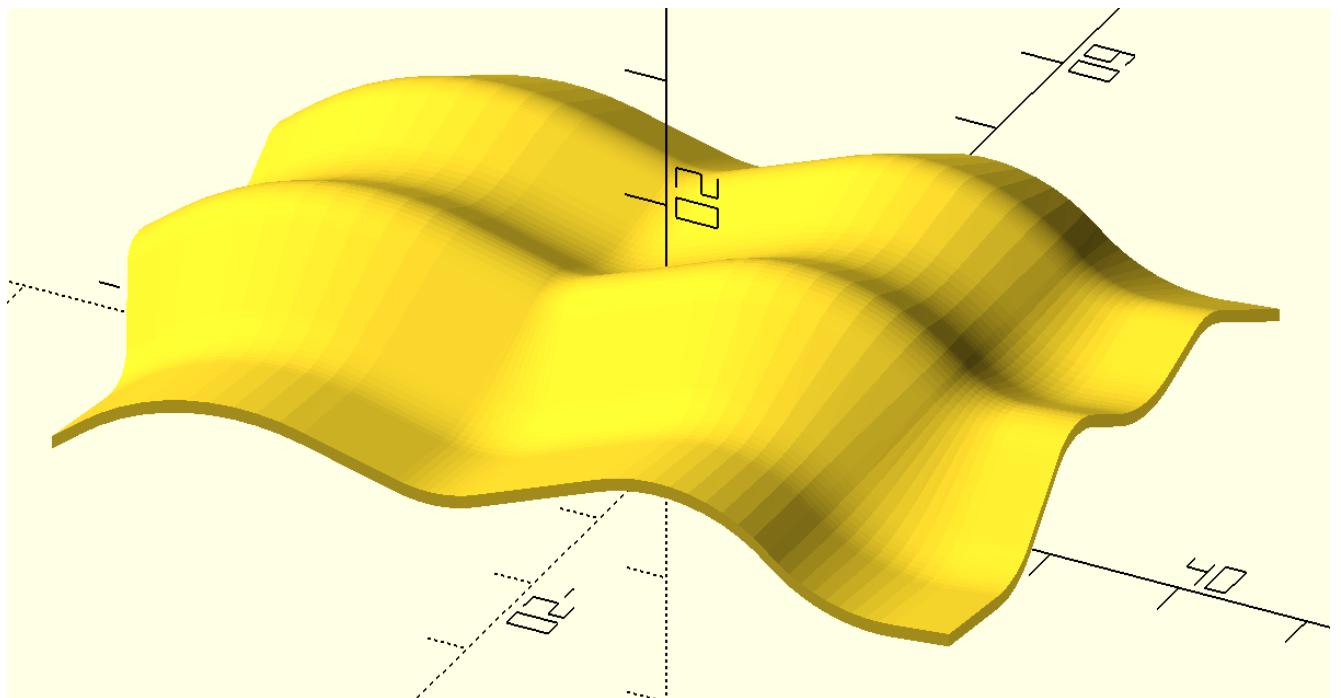
sec2=corner_radius(pts1([[-25,0],[10,5,5],[10,-3,10],[10,5,5],[10,-8,7],[10,1]]),10)
path2=cytz(corner_radius(pts1([[-35,5,0],[10,8,20],[20,-5,10],[20,8,20],[10,-9,20],[10,1,0]]),10))
surf2=surf_extrude(sec2,path2)
surf3=surf_extrudef(surf2,t=-1)

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>

{swp(surf3)}
//color("blue")for(p={surf3})p_line3dc(p,.1,rec=1);

...))
t1=time.time()
total=t1-t0
total
```

Out [4]: 0.08682537078857422

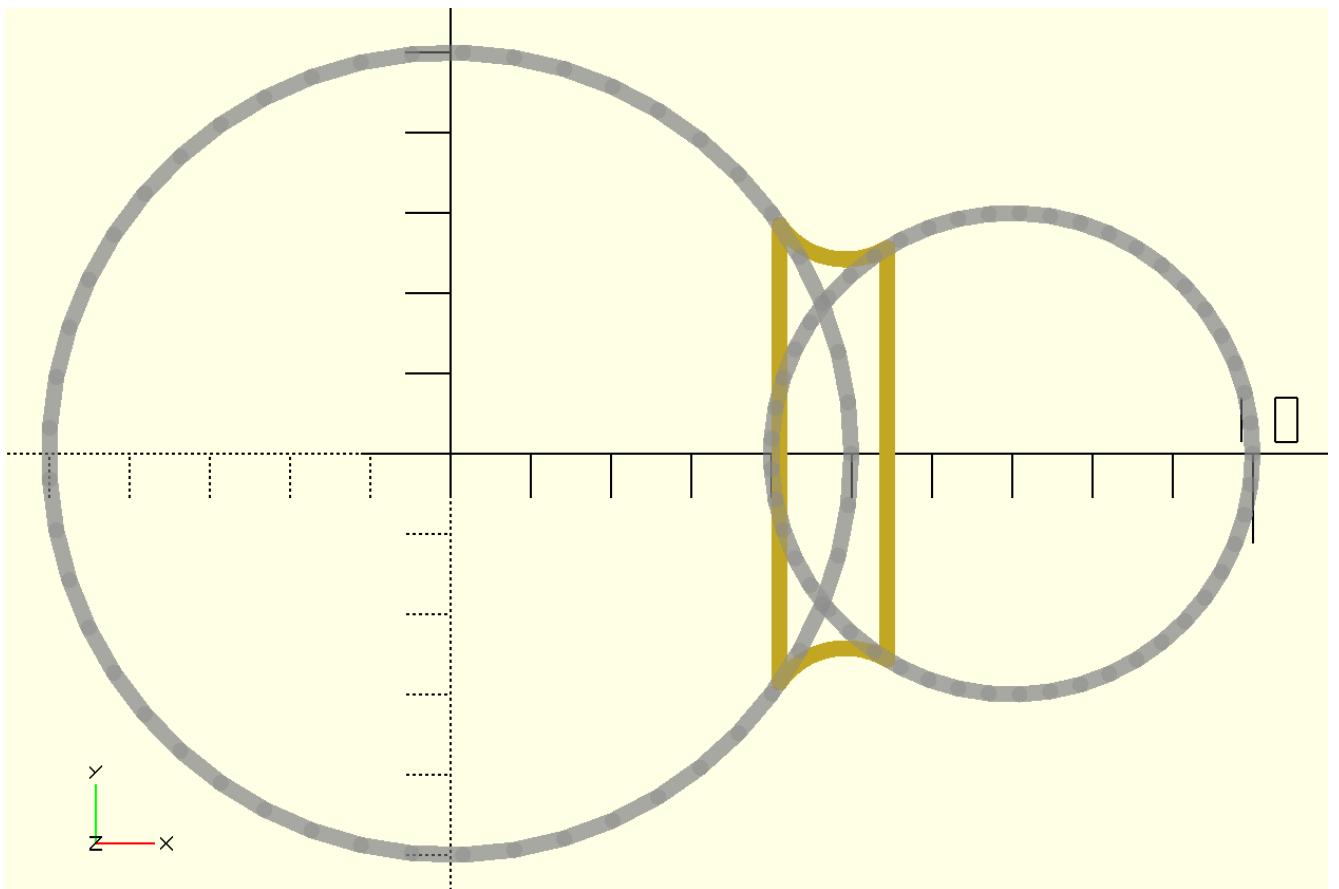


fillet_2cir

```
In [5]: # example of function fillet_2cir
t0=time.time()
fillet=fillet_2cir(r1=5,r2=3,c1=[0,0],c2=[7,0],r=1)
with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>
%p_line({{circle(5)}},.2);
    %p_line({{circle(3,[7,0])}},.2);
    p_line({{fillet}},.2);

    ''')
t1=time.time()
t1-t0
```

Out[5]: 0.005053997039794922



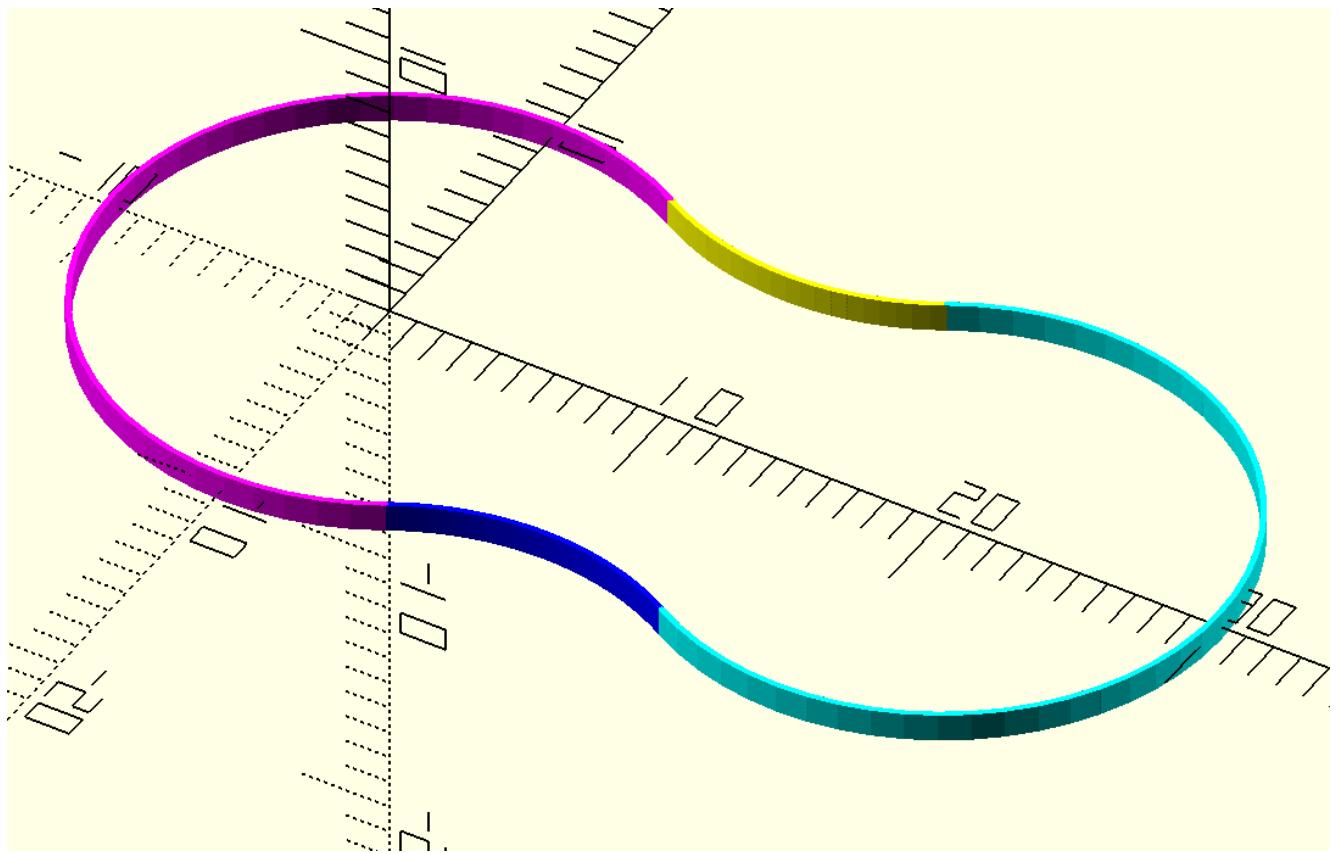
fillet_2cir

```
In [6]: # example of function fillet_2cir and arc_long_2p
t0=time.time()

fillet=fillet_2cir(r1=10,r2=10,c1=[0,0],c2=[20,0],r=10)
p0,p1,p2,p3=fillet[1][len(fillet[1])-1], fillet[0][0],fillet[0][len(fillet[0])-1],fillet[1][0]
cir1=arc_long_2p(p0,p1,10,-1,40)
cir2=arc_long_2p(p2,p3,10,-1,40)
with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>
//%p_line({circle(10)},.2);
//%p_line({circle(10,[20,0])},.2);
color("blue")p_lineo({fillet[0]},.2);
color("yellow")p_lineo({fillet[1]},.2);
color("magenta")p_lineo({cir1},.2);
color("cyan")p_lineo({cir2},.2);

''')
t1=time.time()
t1-t0
```

```
Out[6]: 0.00612187385559082
```



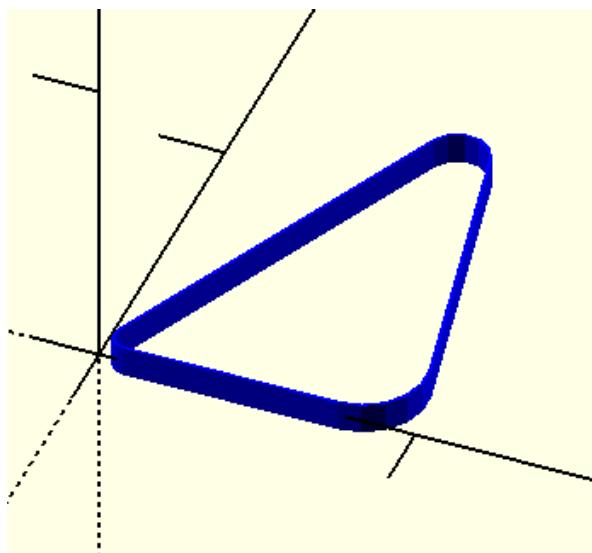
corner_radius(sec,s)

```
In [8]: # example of function corner_radius(pl,s)
t0=time.time()
sec=corner_radius(sec=[[0,0,.5],[10,0,2],[7,15,1]],s=5)

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>
color("blue")p_line({sec},.1);

...))
t1=time.time()
t1-t0
```

Out[8]: 0.0071659088134765625



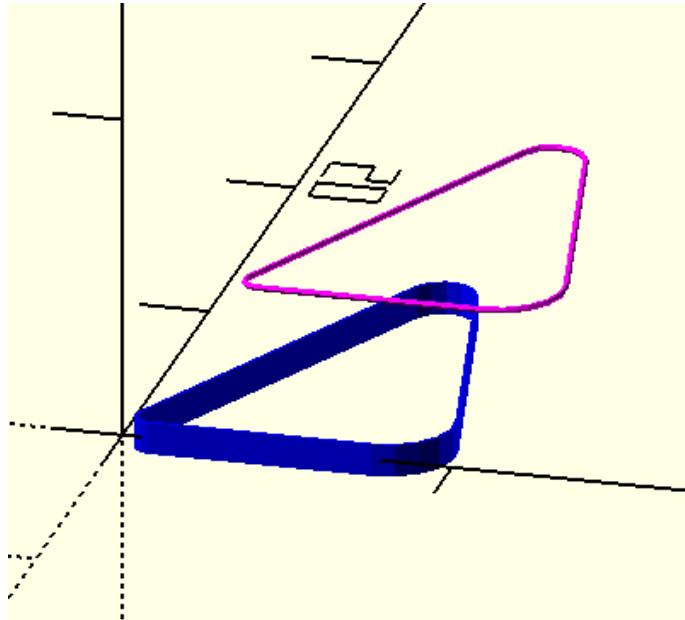
translate

```
In [9]: # example of function translate(p,sec)
t0=time.time()
sec=corner_radius([[0,0,.5],[10,0,2],[7,15,1]],5)
sec1=translate(p=[10,5,3],sec=sec)

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>
color("blue")p_line({sec},.1);
color("magenta")p_line3dc({sec1},.1);

...''')
t1=time.time()
t1-t0
```

Out[9]: 0.006701946258544922

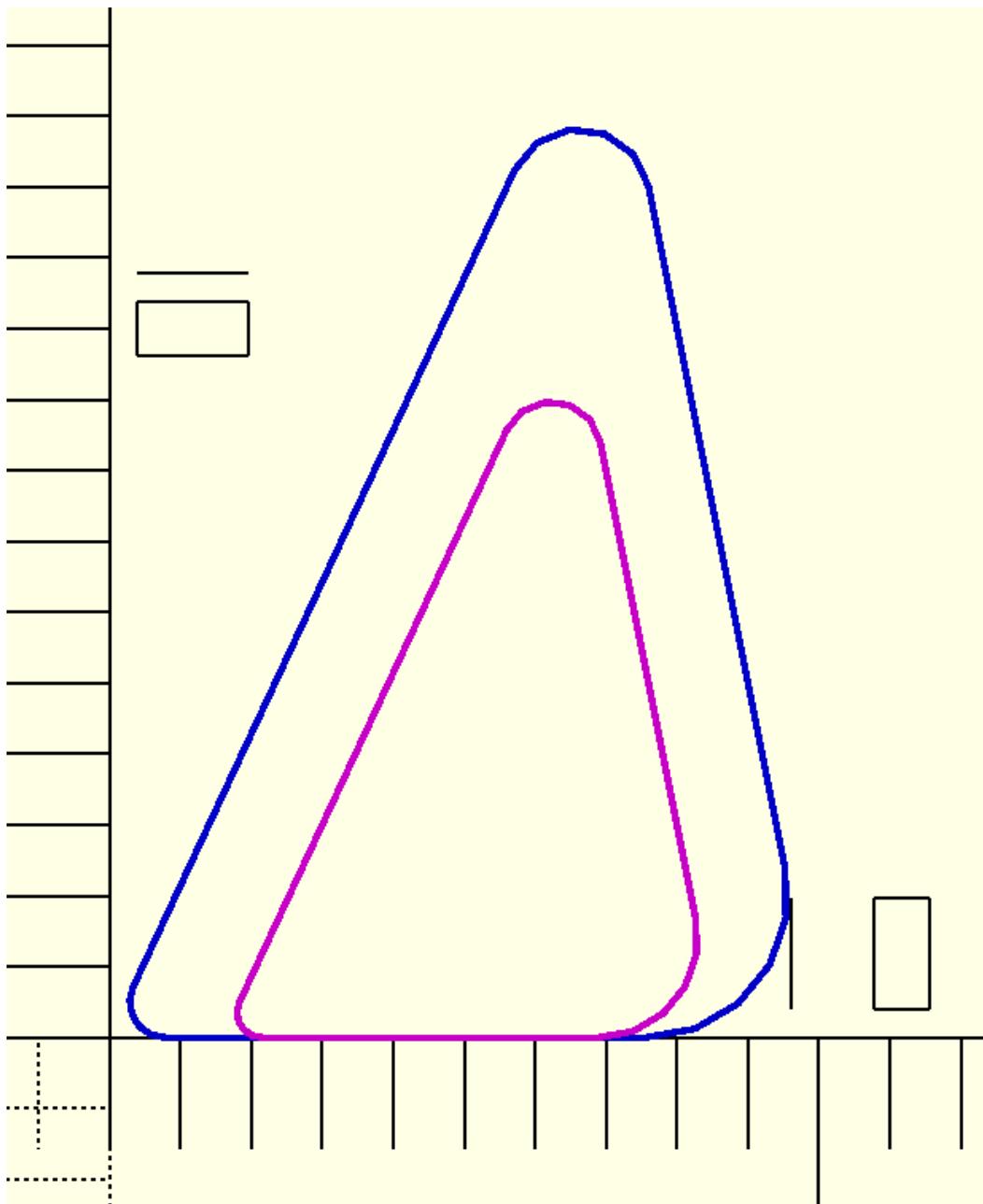


scl2d(sec,sl)

```
In [10]: # example of function scl2d(sec,sl)
t0=time.time()
sec=corner_radius([[0,0,.5],[10,0,2],[7,15,1]],5)
sec1=scl2d(sec,.7)
with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>
color("blue")p_line({sec},.1);
color("magenta")p_line({sec1},.1);

...''')
t1=time.time()
t1-t0
```

Out[10]: 0.006241798400878906



scl2d_c(sec,sl)

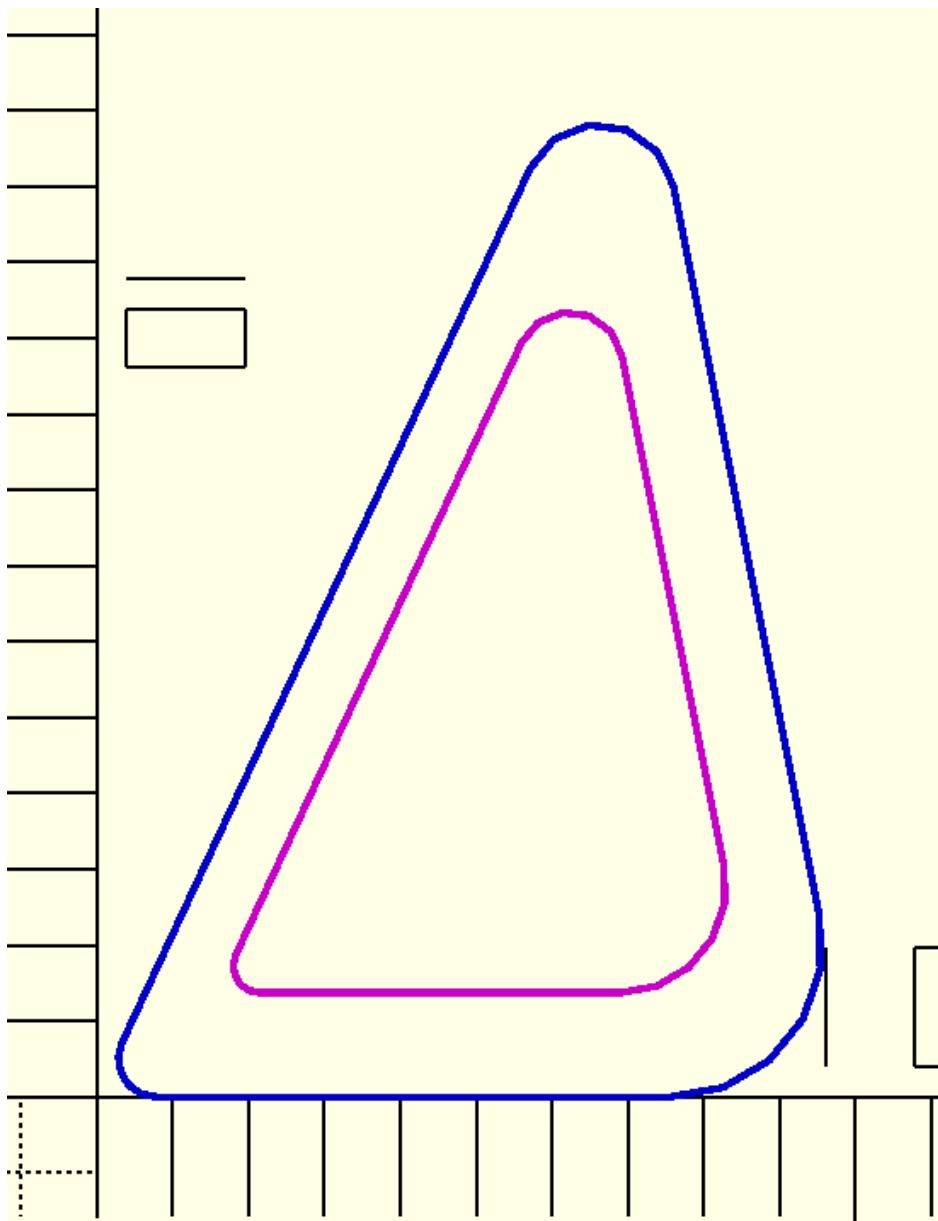
In [11]:

```
# example of function scl2d_c(sec,sl)
t0=time.time()
sec=corner_radius([[0,0,.5],[10,0,2],[7,15,1]],5)
sec1=scl2d_c(sec,.7)

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>
color("blue")p_line({sec},.1);
color("magenta")p_line({sec1},.1);

...))
t1=time.time()
t1-t0
```

Out[11]: 0.008507013320922852



scl3d(p,s)

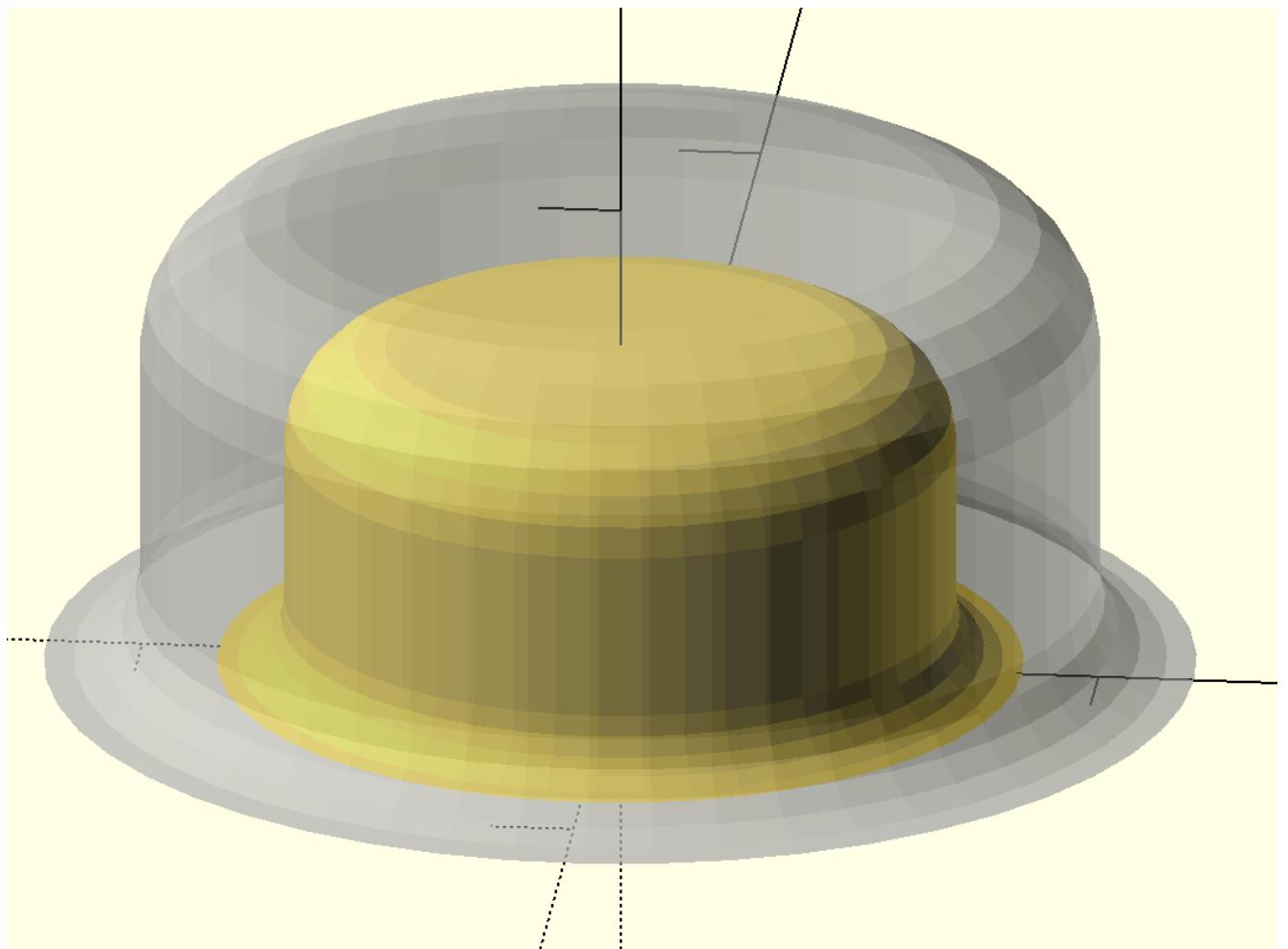
```
In [12]: # example of function scl3d(p,s)
t0=time.time()
sec=circle(10);
path=corner_radius(pts1([[2,0],[-2,0,2],[0,10,3],[-3,0]]),5)
sol=prism(sec,path)
sol1=scl3d(sol,.5)

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>
%{swp(sol)}
{swp(sol1)}

'''')

t1=time.time()
t1-t0
```

Out [12]: 0.05228996276855469



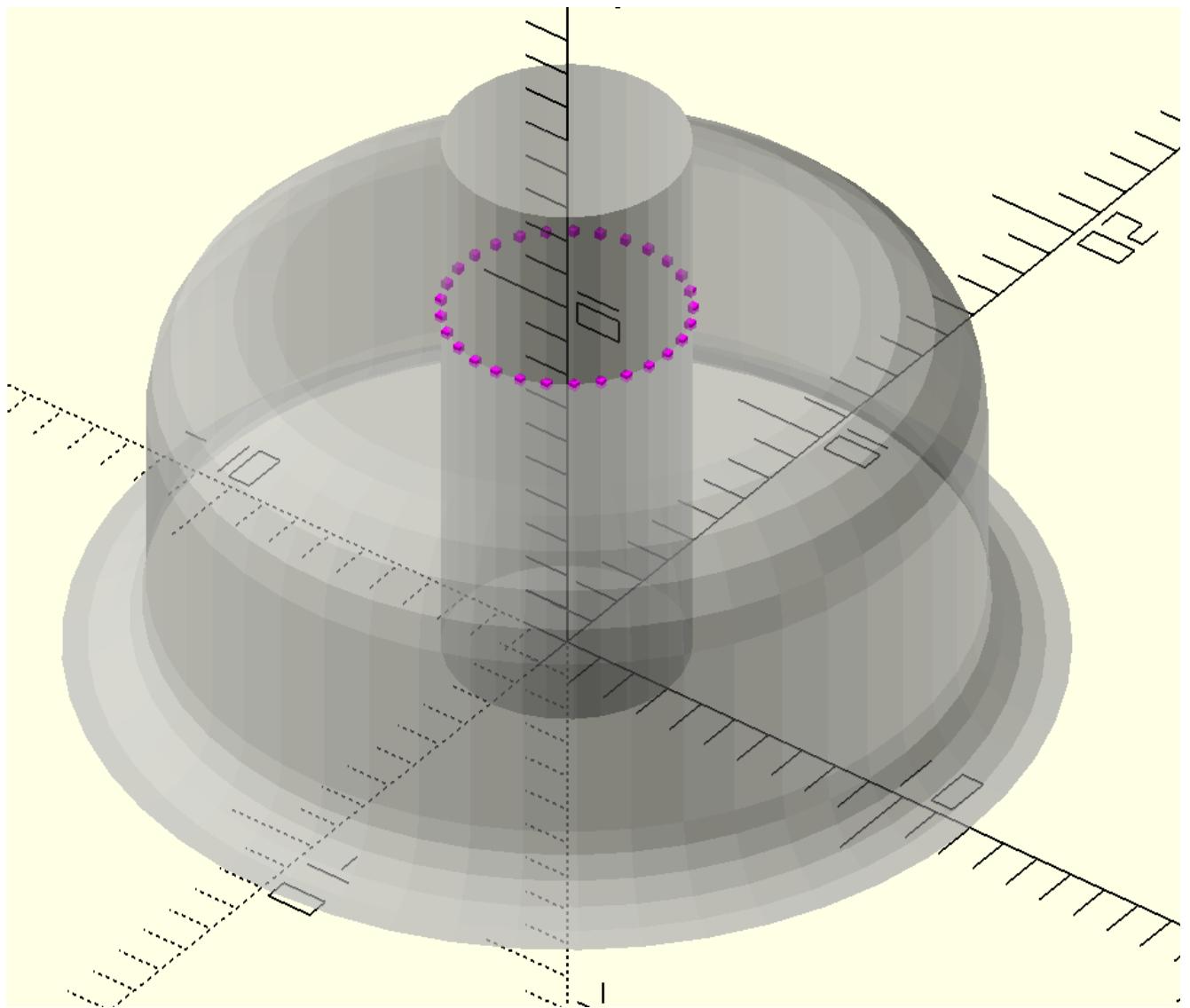
ip(prism,prism1)

```
In [13]: # example of function ip(prism,prism1)
t0=time()

sec=circle(10)
path=corner_radius(pts1([[2,0],[-2,0,2],[0,10,3],[-9.9,0]]),5)
p_0=prism(sec,path)
p_1=cylinder(r=3,h=15,s=30)
ip_1=ip(p_0,p_1)

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>
%{swp(p_0)}
%{swp(p_1)}
color("magenta")points({ip_1},.2);
//color("blue")for(p={cpo(p_0)})p_line3d(p,.05);
//color("blue")for(p={cpo(p_1)})p_line3d(p,.05);
''')
t1=time.time()
t1-t0
```

Out [13]: 0.06089186668395996

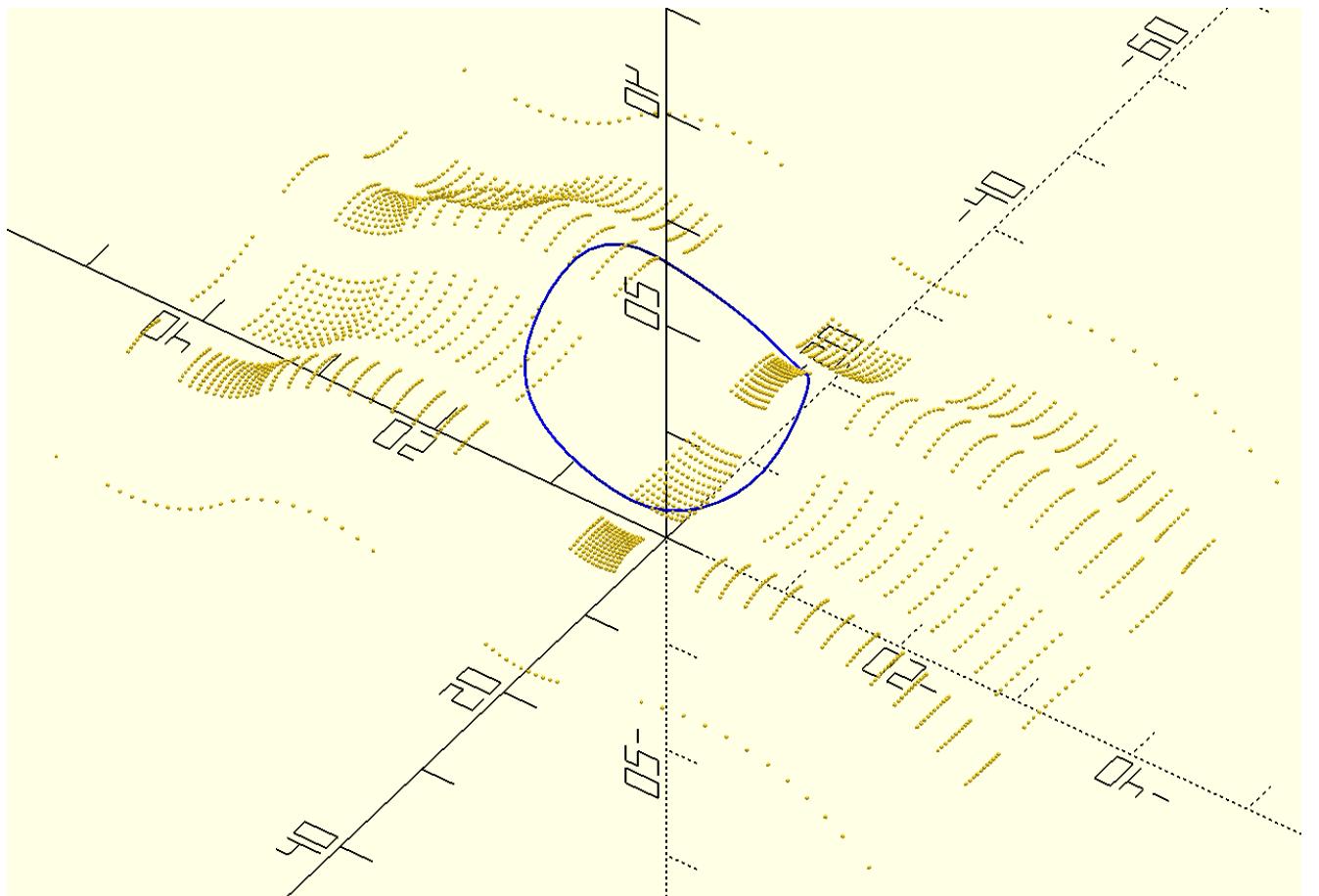


In [14]:

```
t0=time.time()

sec2=corner_radius(pts1([[-25,0],[10,5,5],[10,-3,10],[10,5,5],[10,-8,7],[10,1]]),10)
path2=cytz(corner_radius(pts1([[-35,5,0],[10,8,20],[20,-5,10],[20,8,20],[10,-9,20],[10,1,0]])),10)
surf2=surf_extrude(sec2,path2)
sol1=linear_extrude(circle(10),30)
# sol1=slice_sol(sol1,10)
i_p1=ip_surf(surf2,sol1)
surf3=surf_extrudef(surf2)
with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>
for(p={surf2})points(p,.2);
color("blue")p_line3dc({i_p1},.2);
//%{swp(sol1)}
{swp(surf3)}
''')
t1=time.time()
total=t1-t0
total
```

Out [14]: 0.21703720092773438



tctp(r1,r2,cp1,cp2)

```
In [15]: # example of function tctp(r1,r2,cp1,cp2)
t0=time.time()
cir1=circle(10)
cir2=circle(5,[15,6])
sec=tctp(r1=10,r2=5,cp1=[0,0],cp2=[15,6]);

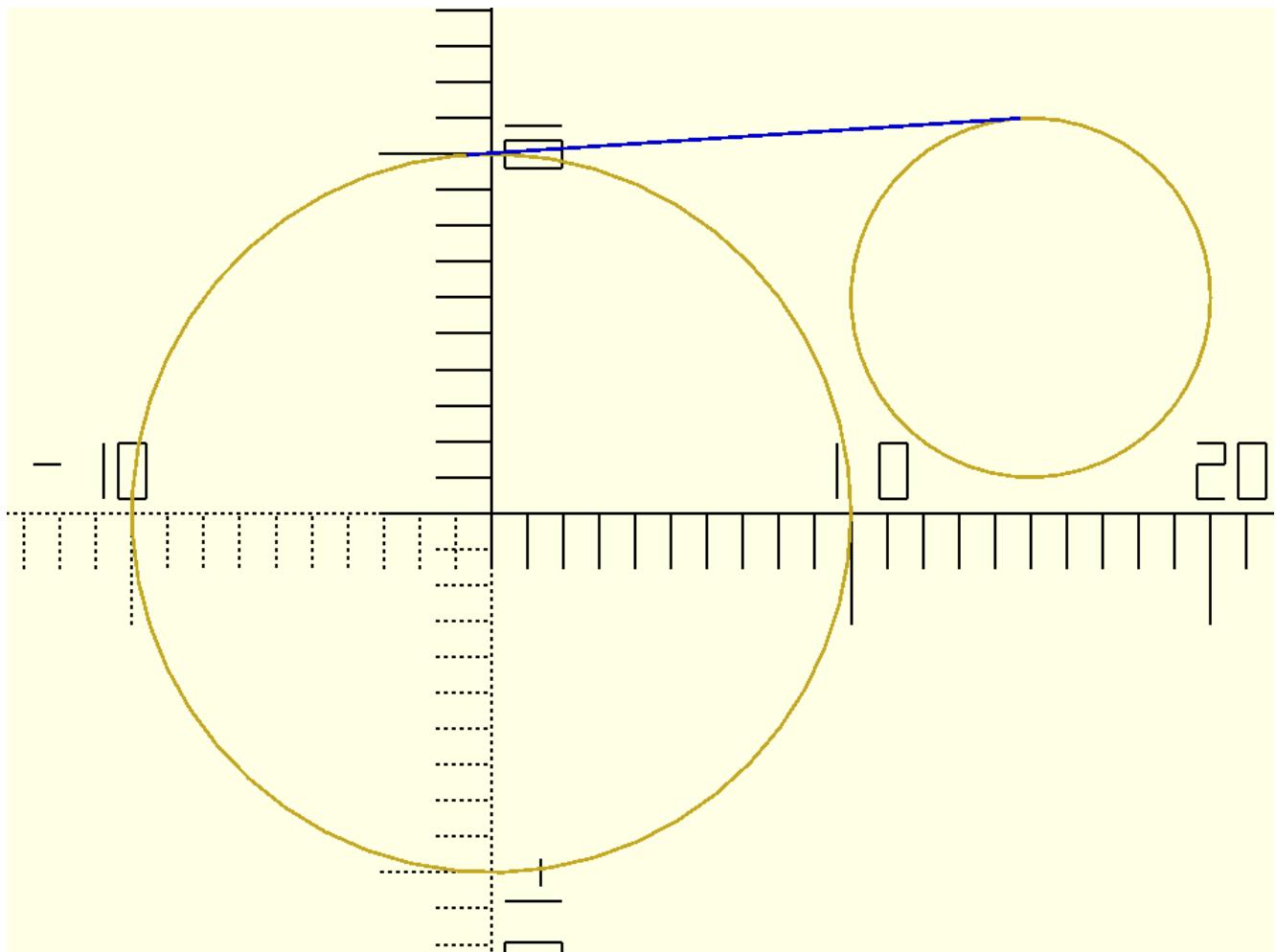
with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>
p_line({{cir1}},.1);
p_line({{cir2}},.1);

color("blue")p_line({{sec}},.1);
color("magenta")points({{[sec[1]]}}},.5);

...)

t1=time.time()
t1-t0
```

```
Out[15]: 0.004054069519042969
```



tctpf(r1,r2,cp1,cp2)

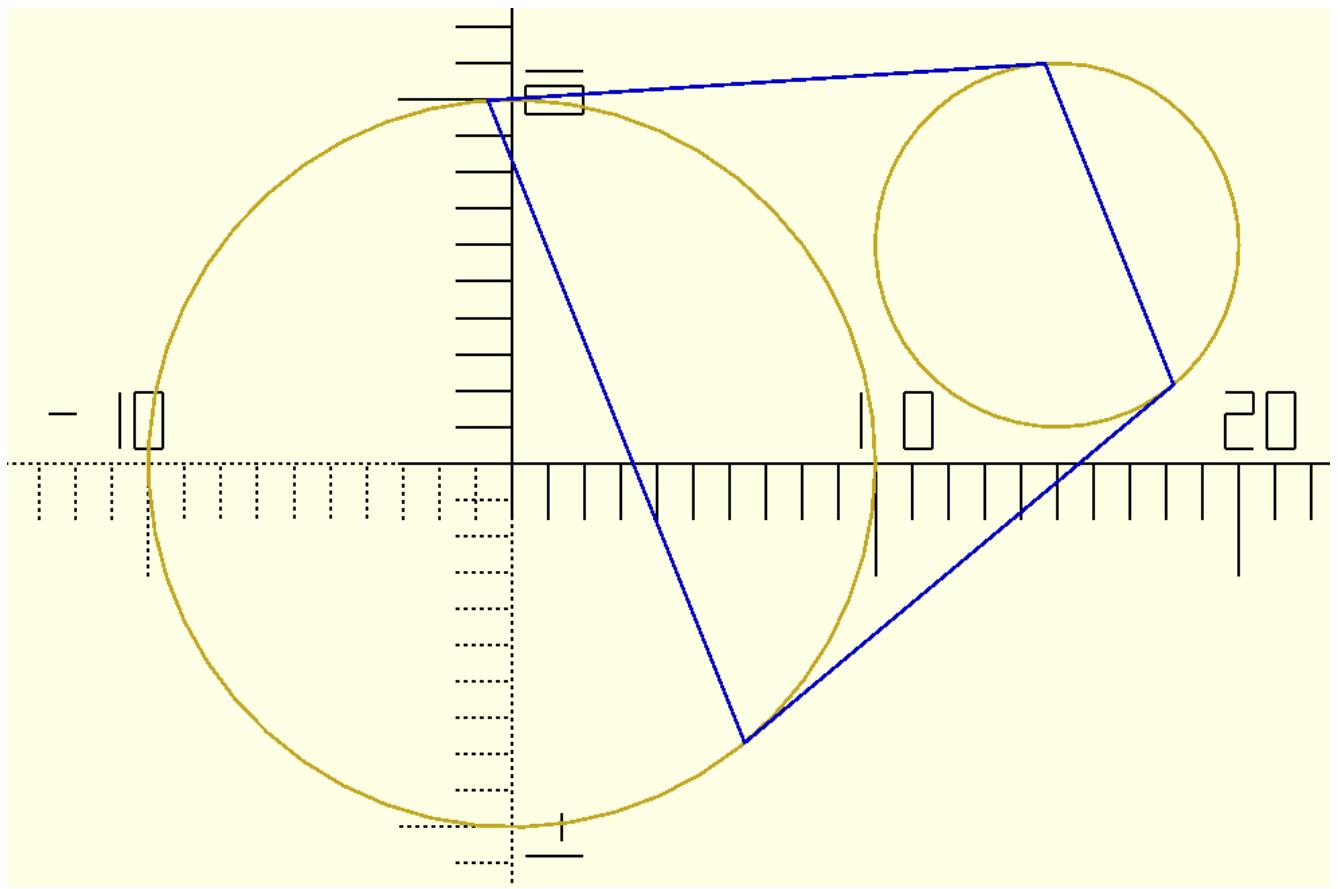
```
In [16]: # example of function tctpf(r1,r2,cp1,cp2)
t0=time.time()

cir1=circle(10)
cir2=circle(5,[15,6])
sec=tctpf(r1=5,r2=10,cp1=[15,6],cp2=[0,0])

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>
p_line({{cir1}},.1);
p_line({{cir2}},.1);
color("blue")p_line({{sec}},.1);

''')
t1=time.time()
t1-t0
```

Out[16]: 0.004689931869506836



multiple_sec_extrude

```
In [17]: # example of function multiple_sec_extrude(path_points=[],radiuses_list=[],sections_list=[],option=0,s
t0=time()

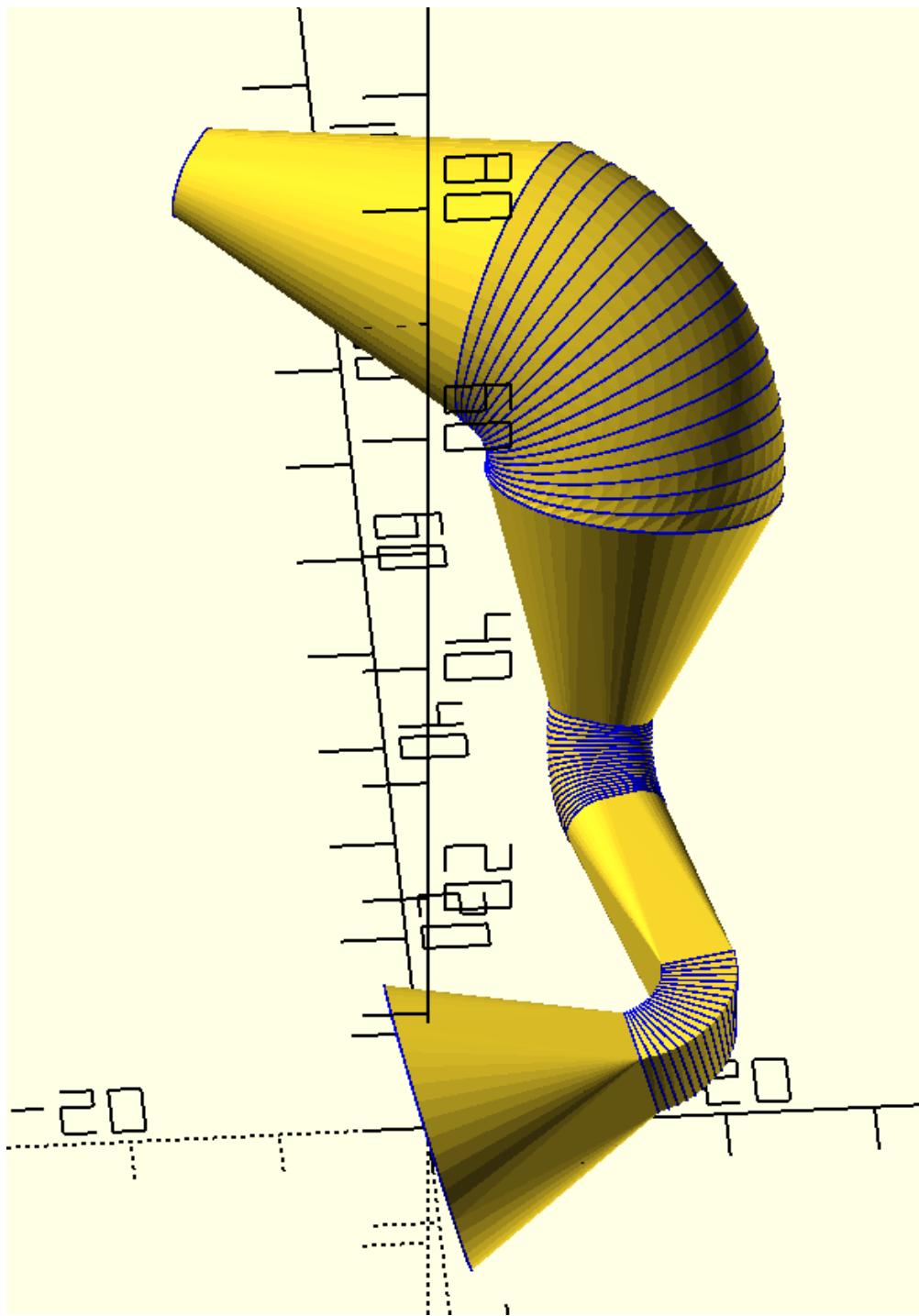
sec1=circle(10)
sec2=square(5,True)
sec3=corner_radius(pts1([[-3.5,-2.5,2.49],[7,0,2.49],[0,5,2.49],[-7,0,2.49]]))
sections=[sec1,sec2,sec3,sec1,sec3]
path=array([[0,0,0],[20,2,5],[-7,25,3],[5,10,30],[-30,15,3]]).cumsum(0)
r=[0,5,7,12,0]

with open('trial.scad','w+')as f:
    f.write(f'''  

include<dependencies2.scad>
//color("blue")for(p={multiple_sec_extrude(path,r,sections,0,20)})p_line3dc(p,.1);
{swp(multiple_sec_extrude(path,r,sections,0,20))}  

''')
t1=time.time()
t1-t0
```

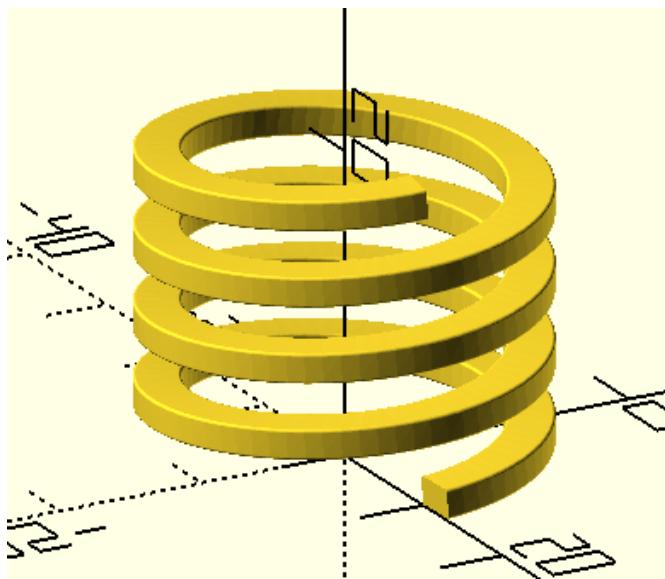
Out [17]: 0.3879821300506592



path_extrude_open(sec,path)

```
In [18]: # example of function path_extrude_open(sec,path)
t0=time.time()
sec=corner_radius(pts1([[0,0,.2],[3,0,.2],[0,2,1],[-3,0,1]]),10)
path=helix(10,2.5,4,5.01)
sol=path_extrude_open(sec,path)
with open('trial.scad','w+')as f:
    f.write(f'''
include<dependencies2.scad>
{swp(sol)}
color("blue")p_line3d({path},.1);
    ''')
t1=time.time()
t1-t0
```

Out[18]: 0.12933993339538574



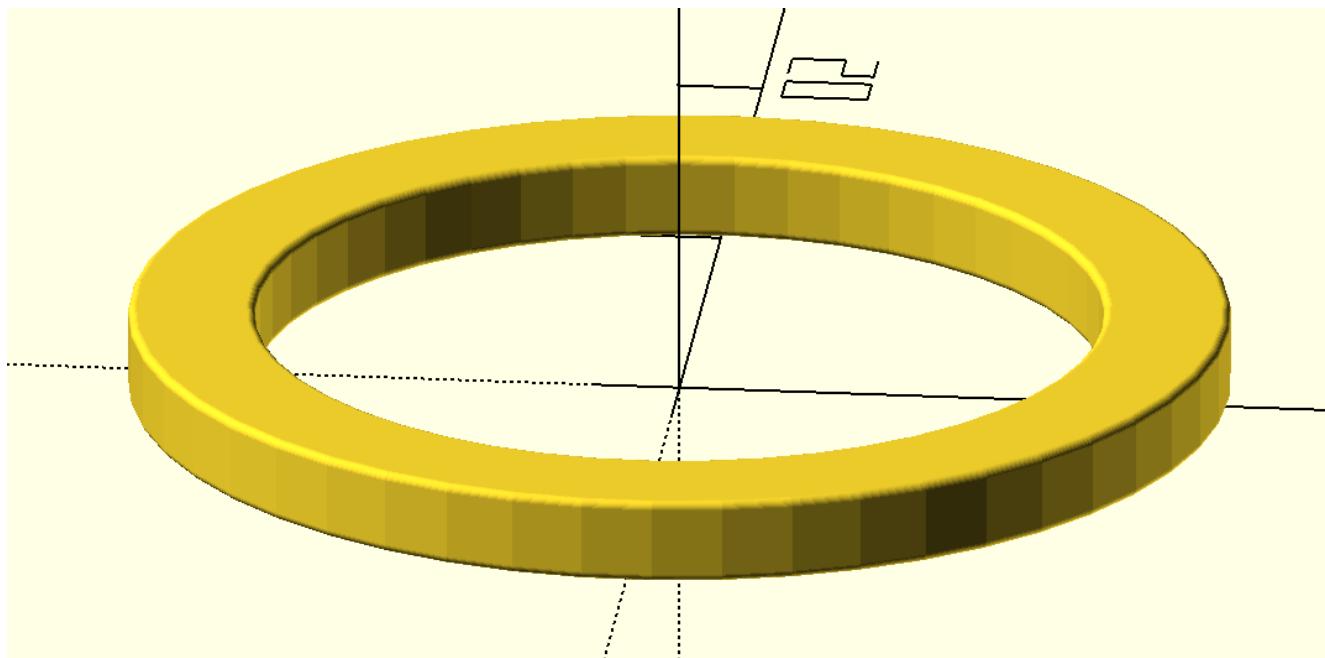
path_extrude_closed(sec,path)

In [19]:

```
# example of function path_extrude_closed(sec,path)
t0=time.time()
sec=corner_radius(pts1([[0,0,.2],[3,0,.2],[0,2,1.49],[-3,0,1.49]]),10)
path=c2t3(circle(10))
sol=path_extrude_closed(sec,path)
with open('trial.scad','w+')as f:
    f.write(f'''
include<dependencies2.scad>
{swp_c(sol)}
color("blue")p_line3d({path},.1);
    ''')

t1=time.time()
t1-t0
```

Out [19]: 0.1380298137664795



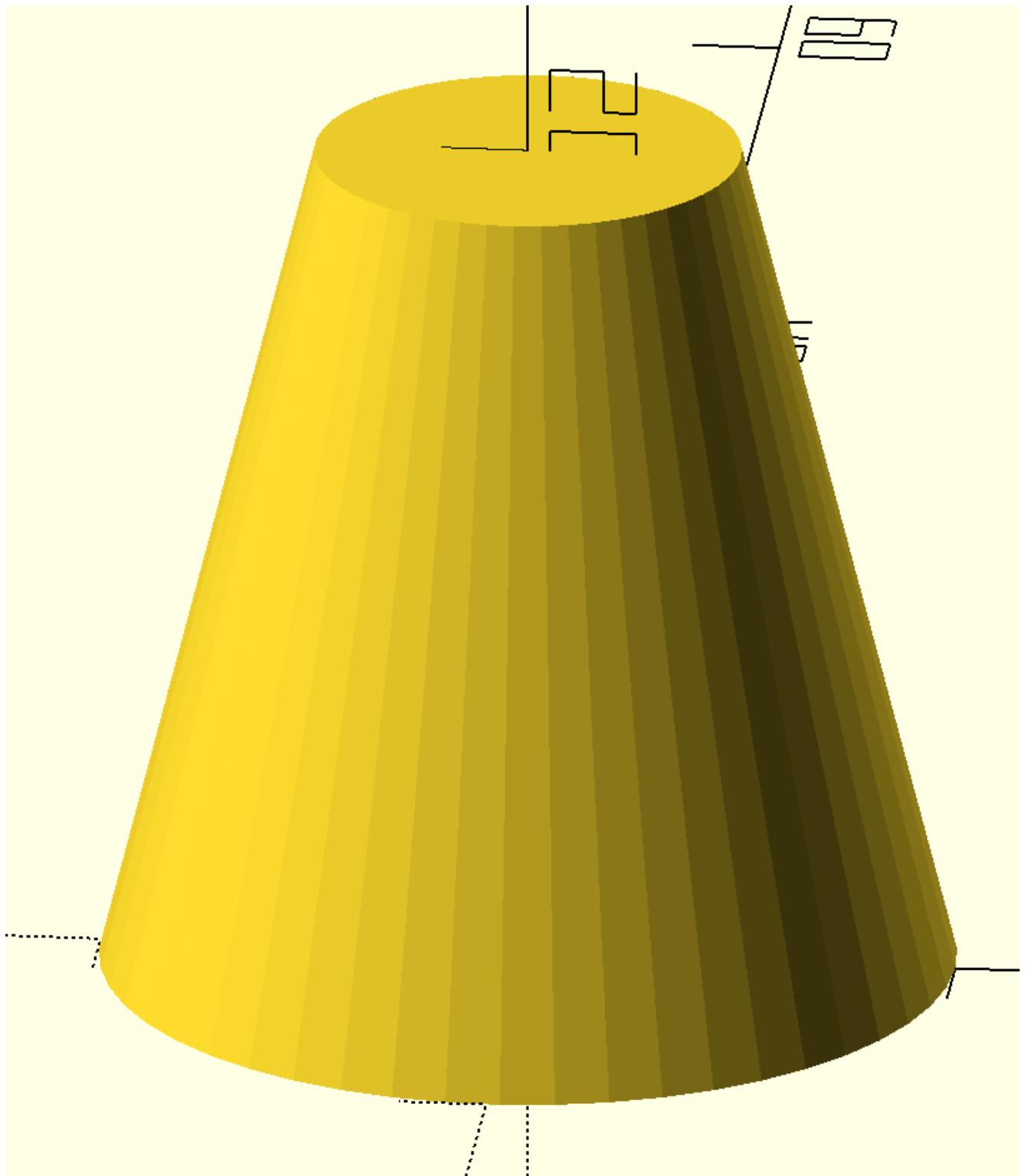
cylinder

In [20]:

```
# example of function cylinder(r1=1, r2=1, h=1, s=50, r=0, d=0, d1=0, d2=0, center=False) and function
t0=time.time()
sol=cylinder(r1=10,r2=5,h=20)
with open('trial.scad','w+')as f:
    f.write(f'''
include<dependencies2.scad>
```

```
{swp(sol)}  
'''')  
t1=time.time()  
t1-t0
```

Out [20]: 0.0055429935455322266

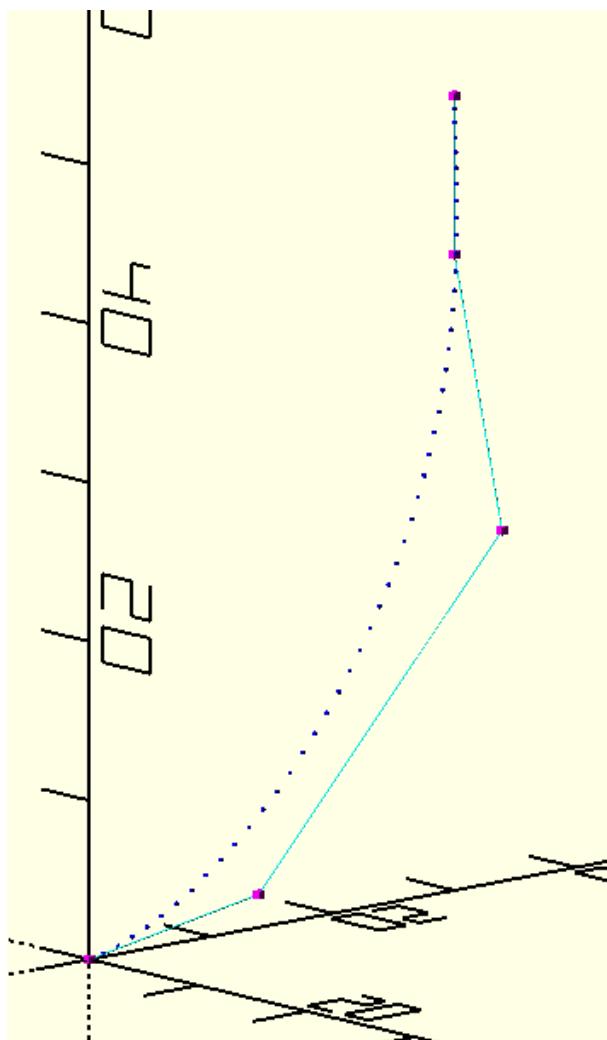


bezier

```
# example of function bezier(p,s=10)  
t0=time.time()  
control_points=array([[0,0,0],[10,5,5],[-10,10,20],[-10,5,15],[10,0,10]]).cumsum(0).tolist()  
curve=bezier(control_points,50)  
  
with open('trial.scad','w+')as f:  
    f.write(f'''  
include<dependencies2.scad>  
color("blue")p_line3d({curve},.1);  
// control points  
color("magenta")points({control_points},.5);  
color("cyan")p_line3d({control_points},.05);
```

```
'''')
t1=time.time()
t1-t0
```

Out[21]: 0.0071256160736083984



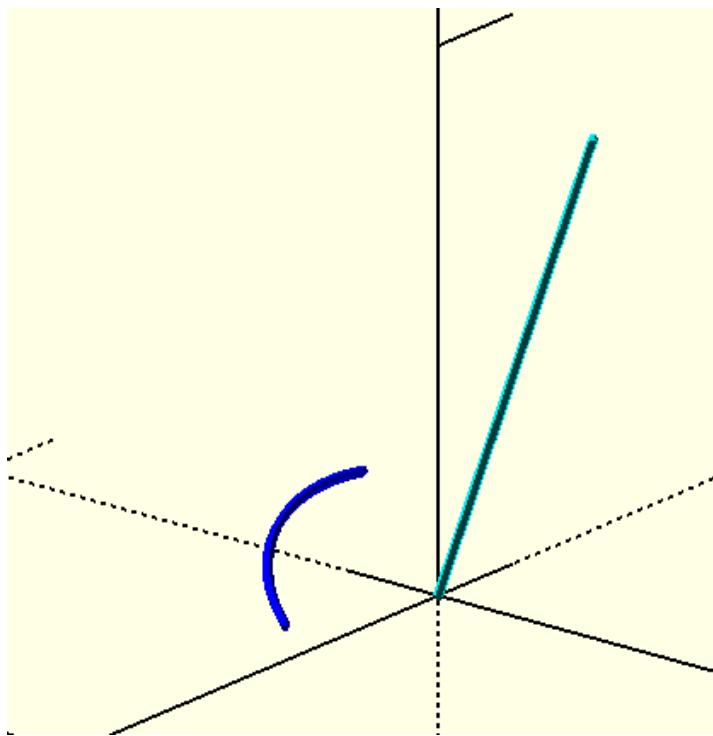
arc_3d

```
# example of function arc_3d(v=[0,0,1],r=1,theta1=0,theta2=360,cw=-1,s=50)
t0=time.time()
vector=[[0,0,0],[1,0,1]]
arc1=arc_3d(v=[1,0,1],r=3,theta1=0,theta2=270,cw=-1,s=50)

with open('trial.scad','w+')as f:
    f.write(f'''
include<dependencies2.scad>
//arc
color("blue")p_line3d({arc1},.1);
// vector
color("cyan")p_line3d({vector},.1);

''')
t1=time.time()
t1-t0
```

Out[22]: 0.014594078063964844



plane

```
In [23]: # example of function plane nv, radius)
t0=time.time()
vector=[[0,0,0],[2,5,10]]
plane1=plane(nv=[2,5,10],radius=10)

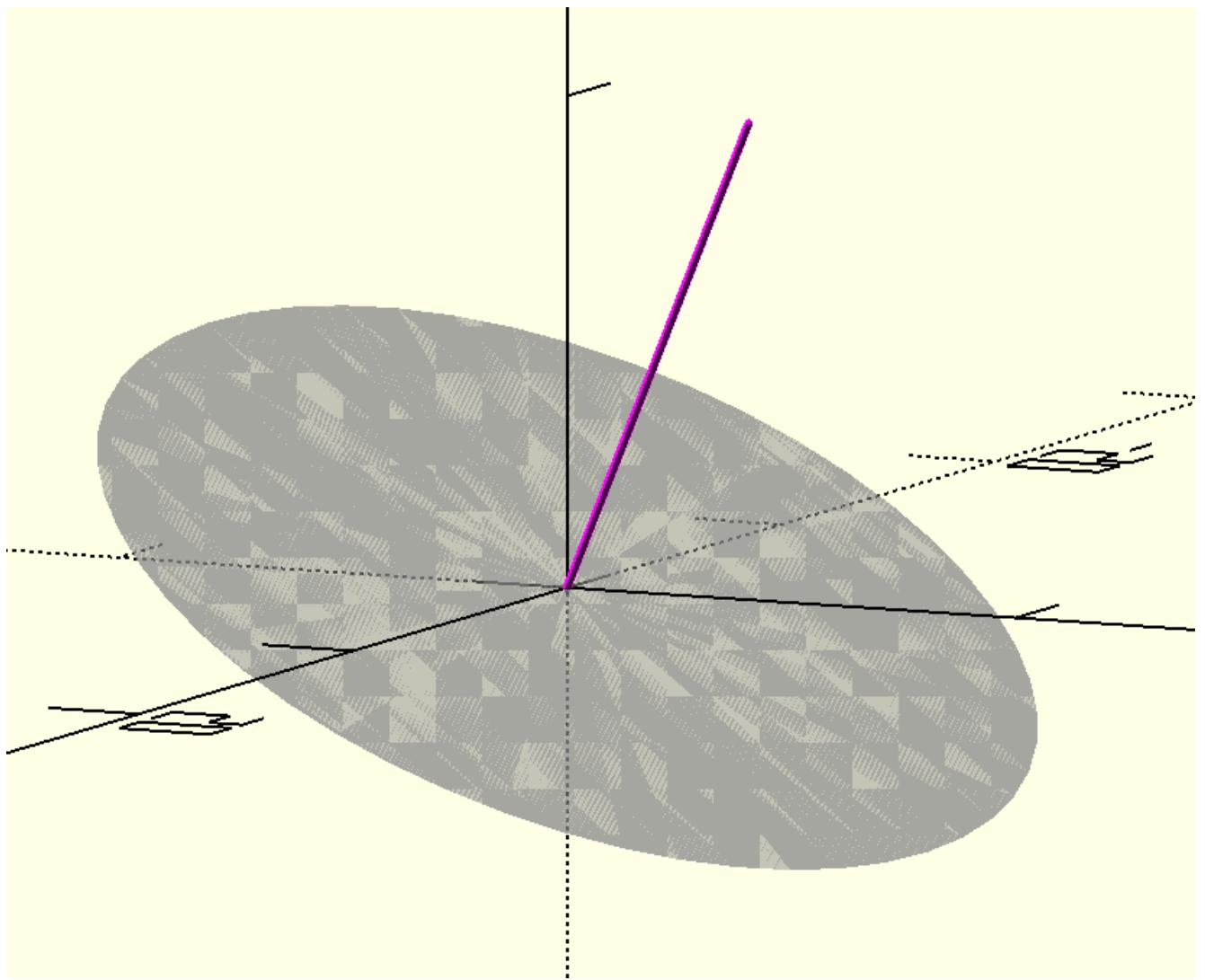
with open('trial.scad','w+')as f:
    f.write(f''''
include<dependencies2.scad>

//plane
%{swp(plane1)}

// vector
color("magenta")p_line3d({vector},.1);

''')
t1=time.time()
t1-t0
```

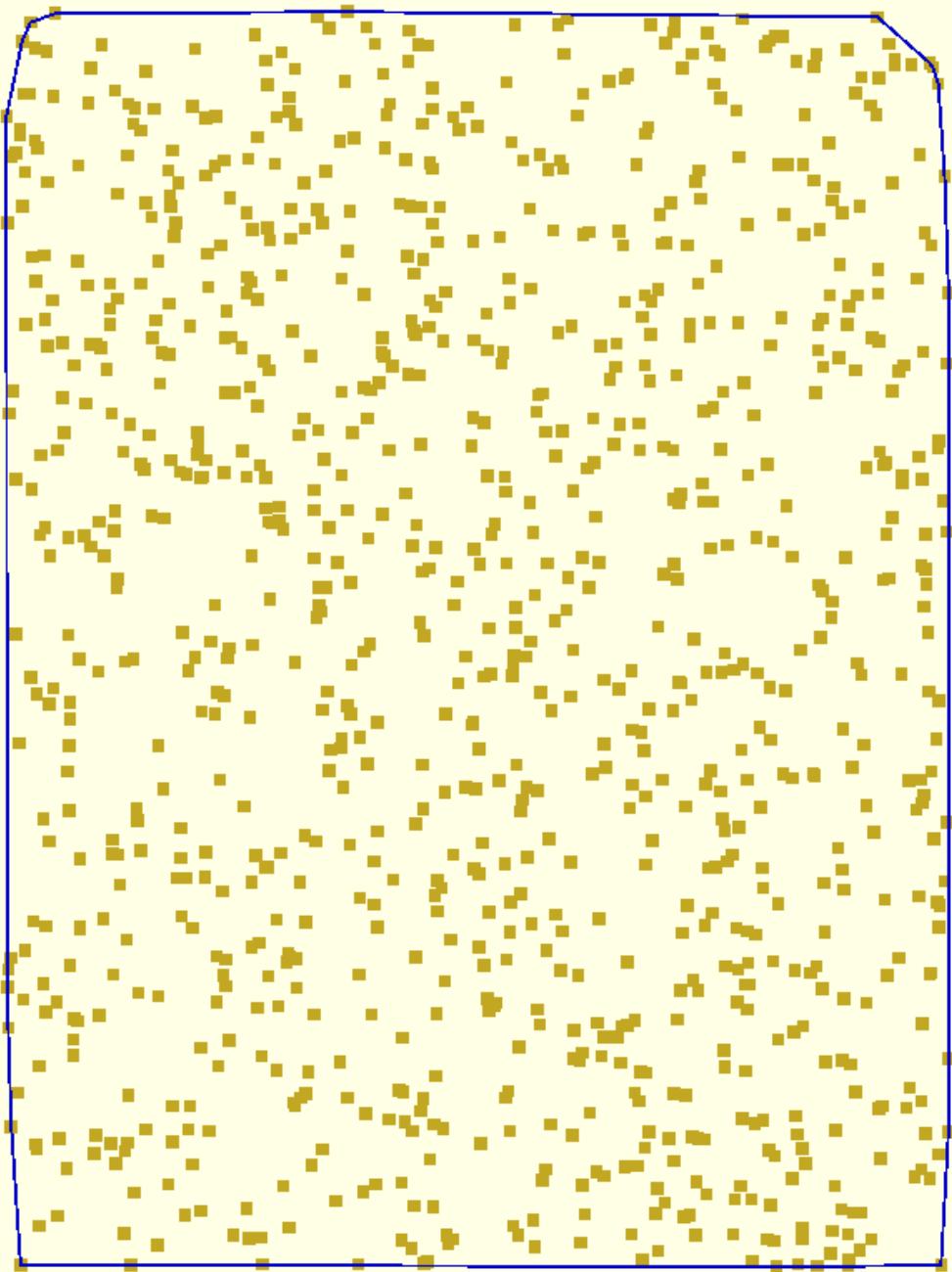
```
Out[23]: 0.01542806625366211
```



c_hull

```
In [24]: # example of function c_hull(pnt)
t0=time.time()
a=random.random(1000)*(20-5)+5
b=random.random(1000)*(30-10)+10
points=array([a,b]).transpose(1,0).tolist()
sec=c_hull(points)
with open('trial.scad','w+')as f:
    f.write(f'''  
include<dependencies2.scad>  
  
points({points},.2);  
color("blue")p_line({sec},.05);  
  
''')
t1=time.time()
t1-t0
```

Out[24]: 0.04248809814453125



convex

```
In [25]: # example of function convex(sec)
t0=time.time()
sec1=corner_radius(pts1([[0,0,.2],[8,3,3],[5,7,1],[-8,0,2],[-5,20,1]]),20)
sec2=corner_radius(pts1([[0,0,1],[7,5,2],[5,7,3],[-5,7,2],[-7,5,3]]),20)

with open('trial.scad','w+')as f:
    f.write(f'''
include<dependencies2.scad>

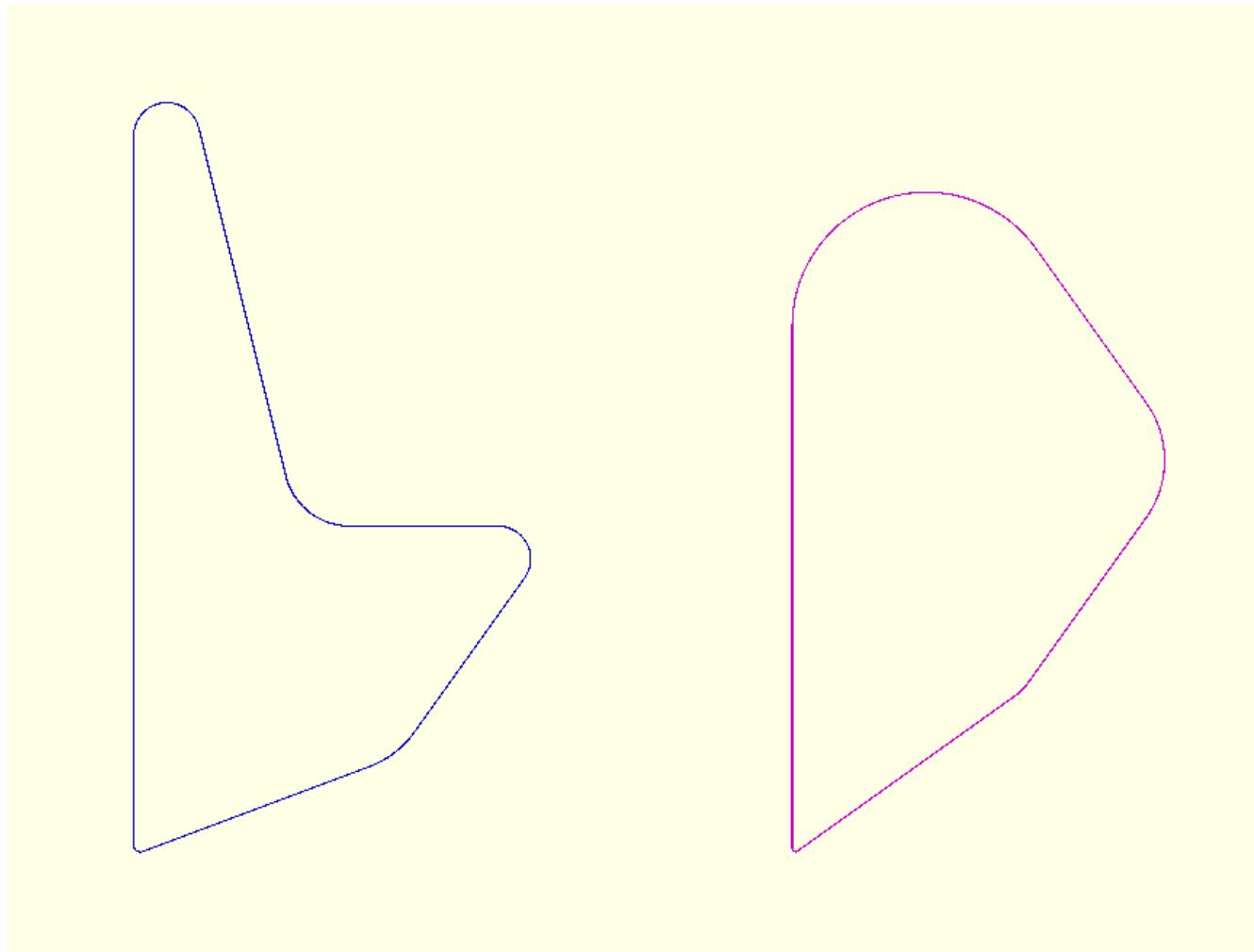
// not a convex section
color("blue")p_line({sec1},.05);

//convex section
color("magenta")translate([20,0,0])p_line({sec2},.05);

    ''')
```

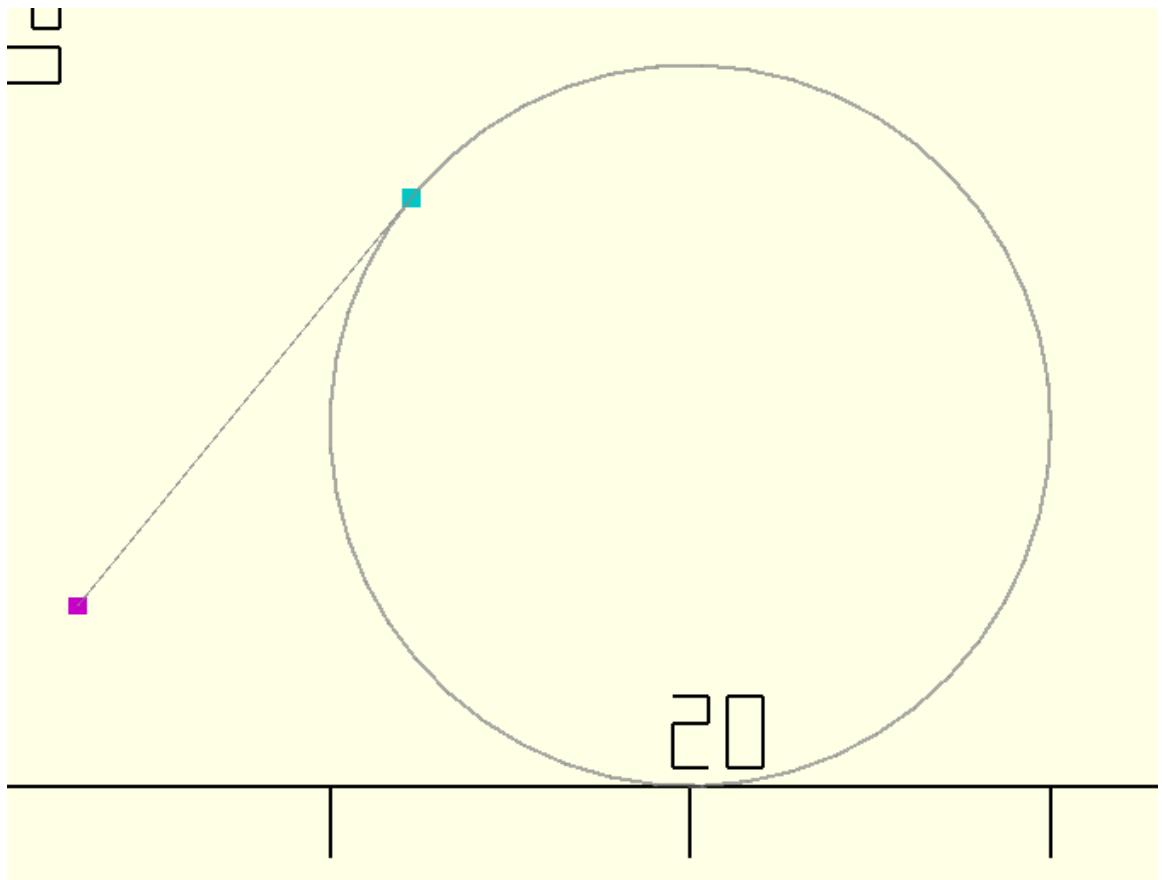
```
t1=time.time()
convex(sec1),convex(sec2),t1-t0
```

Out[25]: (False, True, 0.014436960220336914)



cir_p_t

```
In [26]: # example of function cir_p_t(cir,pnt)
cir=c3t2(translate([20,10,0],circle(10)))
point=[1,30]
tangent_point=cir_p_t(cir,point)
with open('trial.scad','w+')as f:
    f.write(f'''
include<dependencies2.scad>
// "cyan" is the tangent point from circle to some external point ("magenta color")
%p_line({cir},.1);
color("magenta")points({[point]},.5);
color("cyan")points({[tangent_point]},.5);
//color("blue")
%p_line({[point,tangent_point]},.05);
    ''')
```



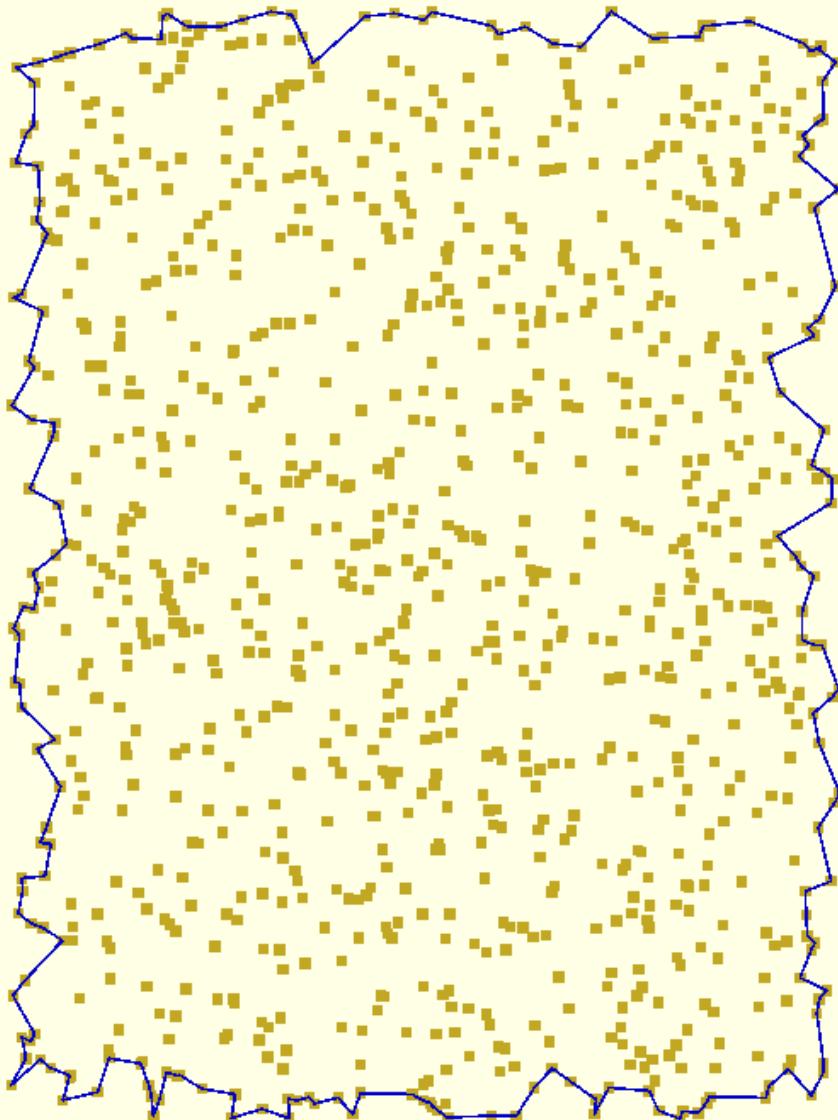
concave_hull

```
In [27]: # example of function concave_hull(pnts,x=1,loops=10)
t0=time.time()
a=random.random(1000)*(20-5)+5
b=random.random(1000)*(30-10)+10
points=remove_extra_points(array([a,b]).transpose(1,0))
conc_hull=concave_hull(points,3)
with open('trial.scad','w+')as f:
    f.write(f'''
include<dependencies2.scad>

points({points},.2);
color("blue")p_line({conc_hull},.05);

    ''')
t1=time.time()
t1-t0
```

Out [27]: 0.34740781784057617



```
In [28]: # example of pies1(sec,pnts) and concave_hull
t0=time.time()

a=random.random(10000)*(10-0)+0
b=random.random(10000)*(20-0)+0
c=array([a,b]).transpose(1,0).tolist()

sec=corner_radius(pts1([[1.5,8,.9],[2,0,.9],[0,-3,1.4],[3,0,1.4],[0,4,1.9],[-5,0,2.5],[0,6,2.5],
[7,0,2.5],[0,-4,.9],[-2,0,.9],[0,3,1.4],[-3,0,1.4],[0,-4,1.9],
[5,0,2.5],[0,-6,2.5],[-7,0,2.5]]),20)

pnts=pies1(sec,c)

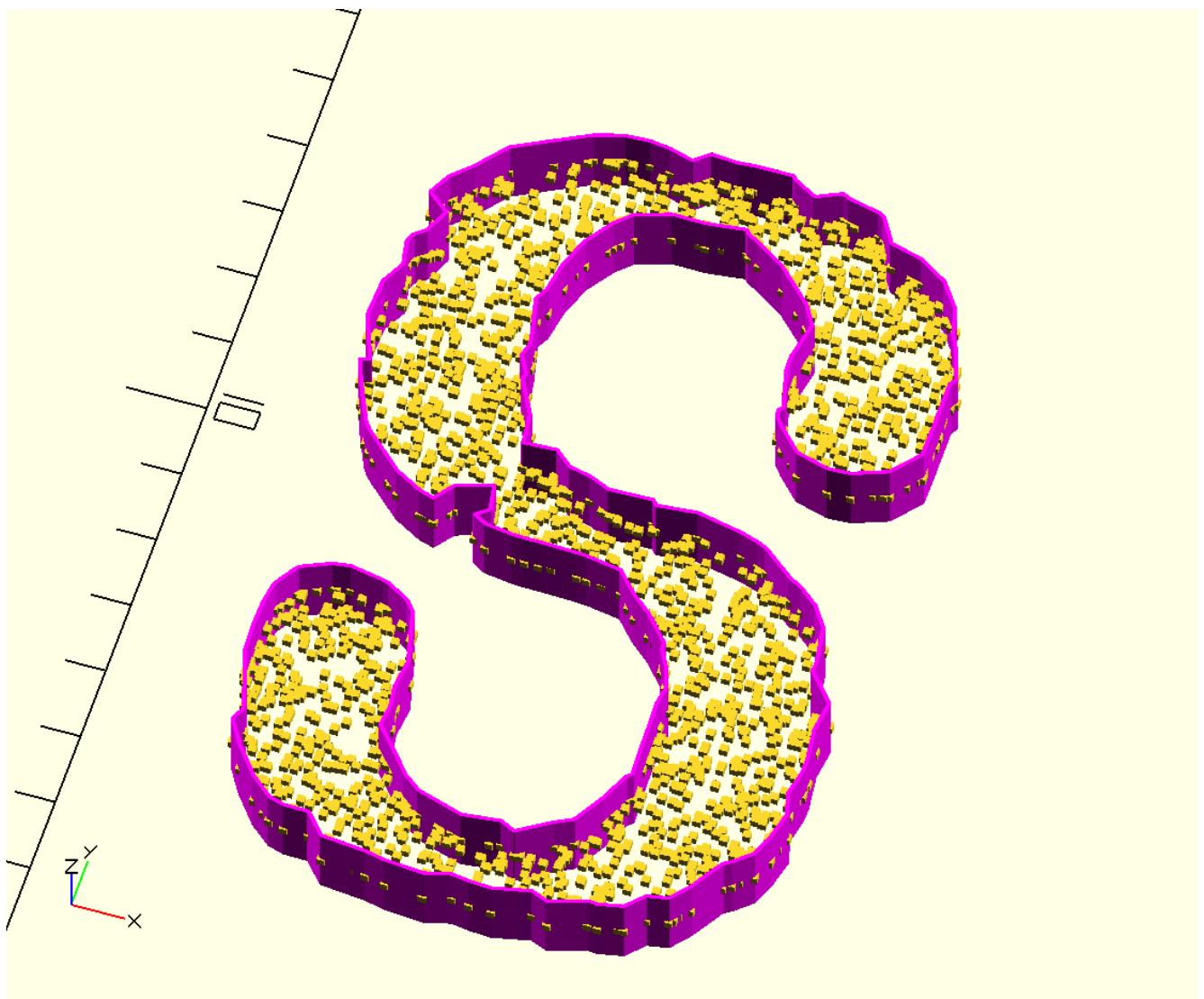
s1=concave_hull(pnts,20)

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>

points({pnts},.1);
color("magenta")p_line({s1},.05);

    ''')
t1=time.time()
t1-t0
```

Out [28]: 1.5716662406921387

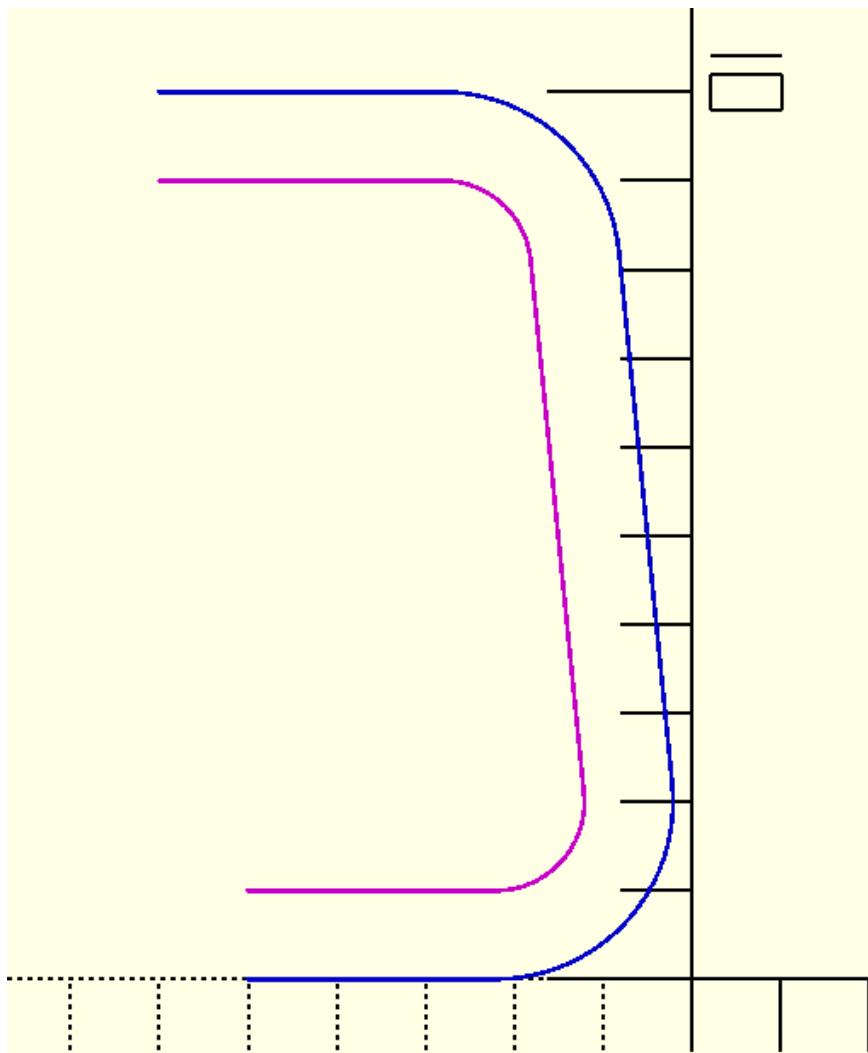


path_offset

In [29]: `# example of function path_offset(path,d)`

```
path=corner_radius(pts1([[-5,0],[5,0,2],[-1,10,2],[-5,0]]))
path1=path_offset(path,-1)
with open('trial.scad','w+')as f:
    f.write(f'''
include<dependencies2.scad>
// orginal path
color("blue")p_lineo({path},.05);
//offset path
color("magenta")p_lineo({path1},.05);

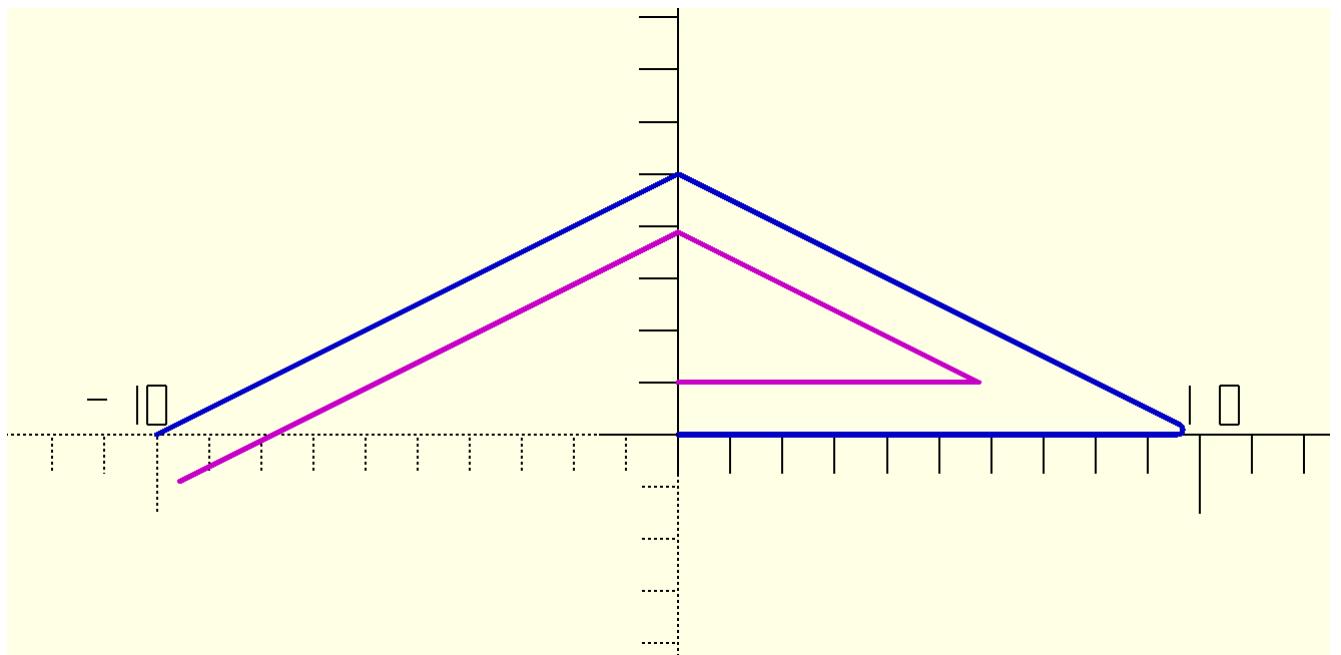
    ''')
```



```
In [30]: # example of function path_offset(path,d)
r=1
path=corner_radius(pts1([[-5,0],[5,0,2],[-1,10,2],[5,0]]),20)
# path=circle(10)
# path=corner_radius(pts1([[0,0],[10,0,.1],[-10,5,.1],[-10,-5]]),20)

path1=path_offset(path,r)

with open('trial.scad','w+')as f:
    f.write(f'''
include<dependencies2.scad>
// orginal path
color("blue")p_lineo({path},.1);
//offset path
color("magenta")p_lineo({path1},.1);
    ''')
```



pntsfaces

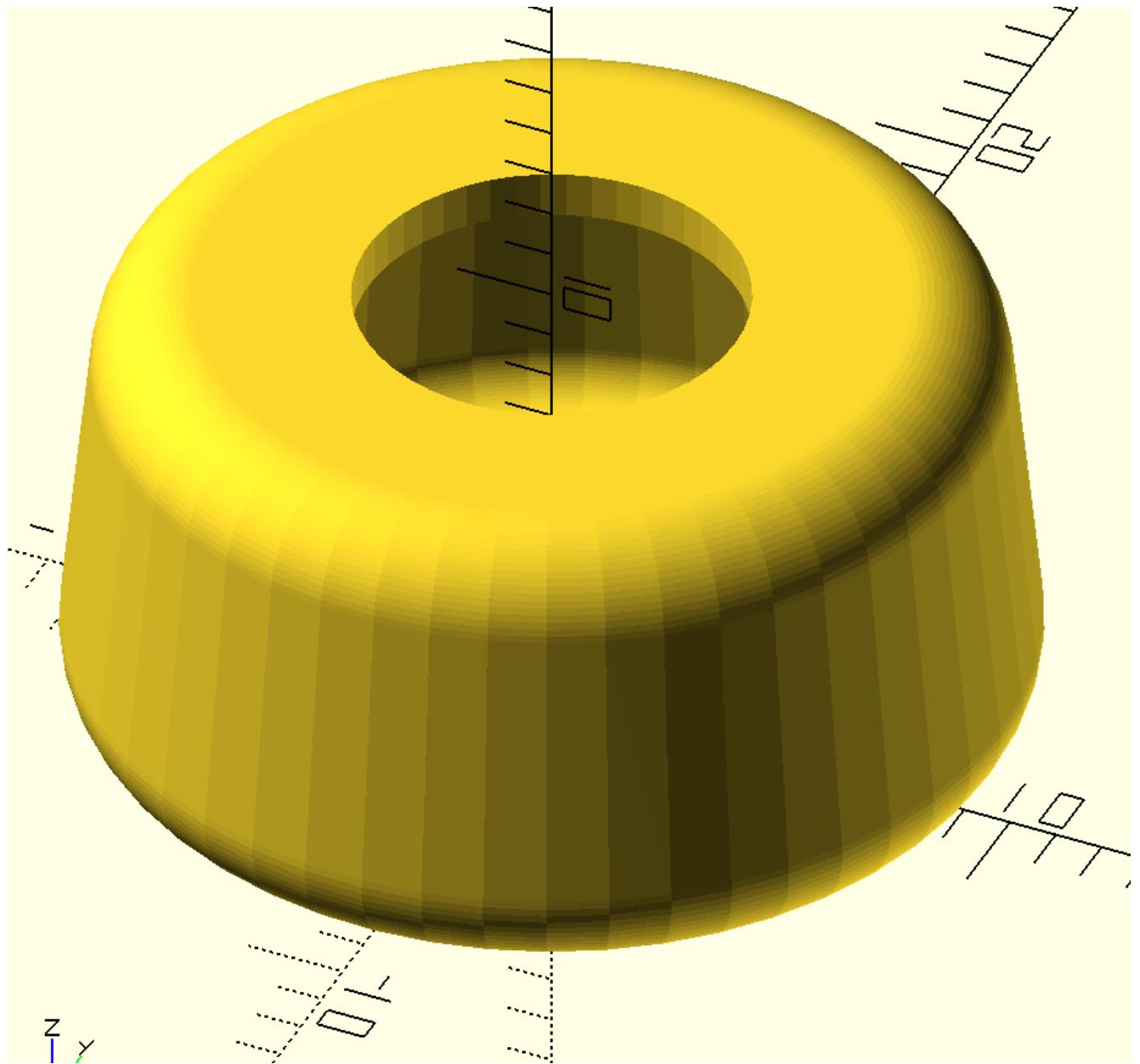
```
In [31]: # example of function pntsfaces(bead2)

path=corner_radius(pts1([[-5,0],[5,0,2],[-1,10,2],[-5,0]]))
path1=path_offset(path,-1)
final_path=path+flip(path1)
sec=circle(10)
sol=prism(sec,final_path)

points, faces=pntsfaces(sol)[0],pntsfaces(sol)[1]

with open('trial.scad','w+')as f:
    f.write(f'''
include<dependencies2.scad>

polyhedron({points},{faces},convexity=10);
    ''')
```



offset_points

```
In [32]: # example of function offset_points(sec,r)
t0=time.time()
sec=corner_radius(pts1([[0,0,.1],[7,5,2],[5,7,3],[-5,7,2],[-7,5,3]]),20)
sec1=offset_points(sec,2)

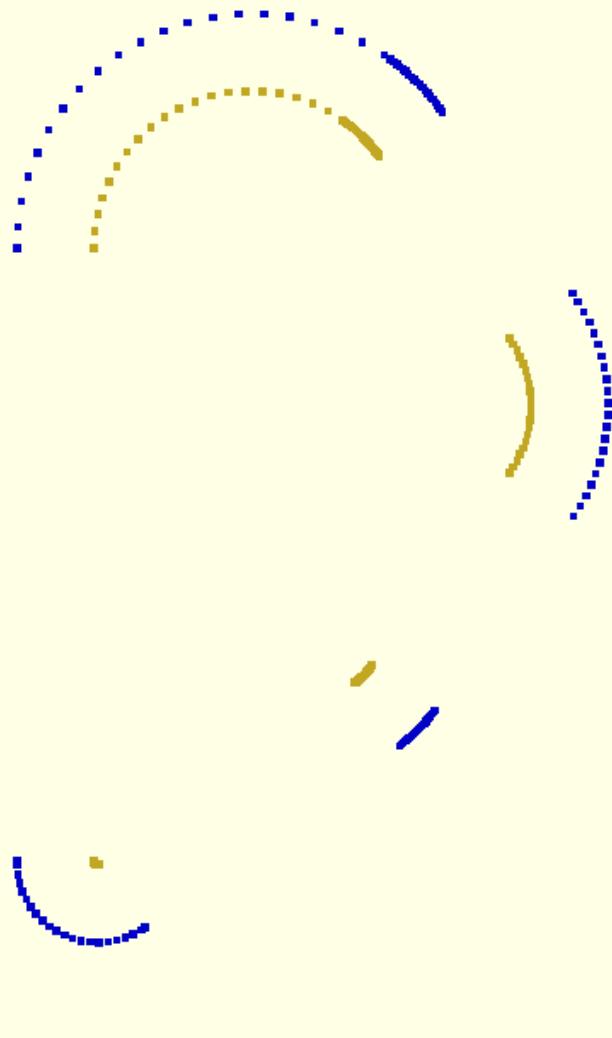
with open('trial.scad','w+')as f:
    f.write(f''''
include<dependencies2.scad>

//original points
points({sec},.2);

// offset points
color("blue") points({sec1},.2);

    ''')
t1=time.time()
t1-t0
```

Out [32]: 0.013966798782348633



In []: `set_printoptions(suppress=True)`

s_int

```
# example of function s_int(s)
t0=time.time()
sec=corner_radius(pts1([[0,0,.1],[7,5,2],[-7,10,2]]),3)
sec1=offset_segv(sec,-1)
self_intersections=s_int(sec1)

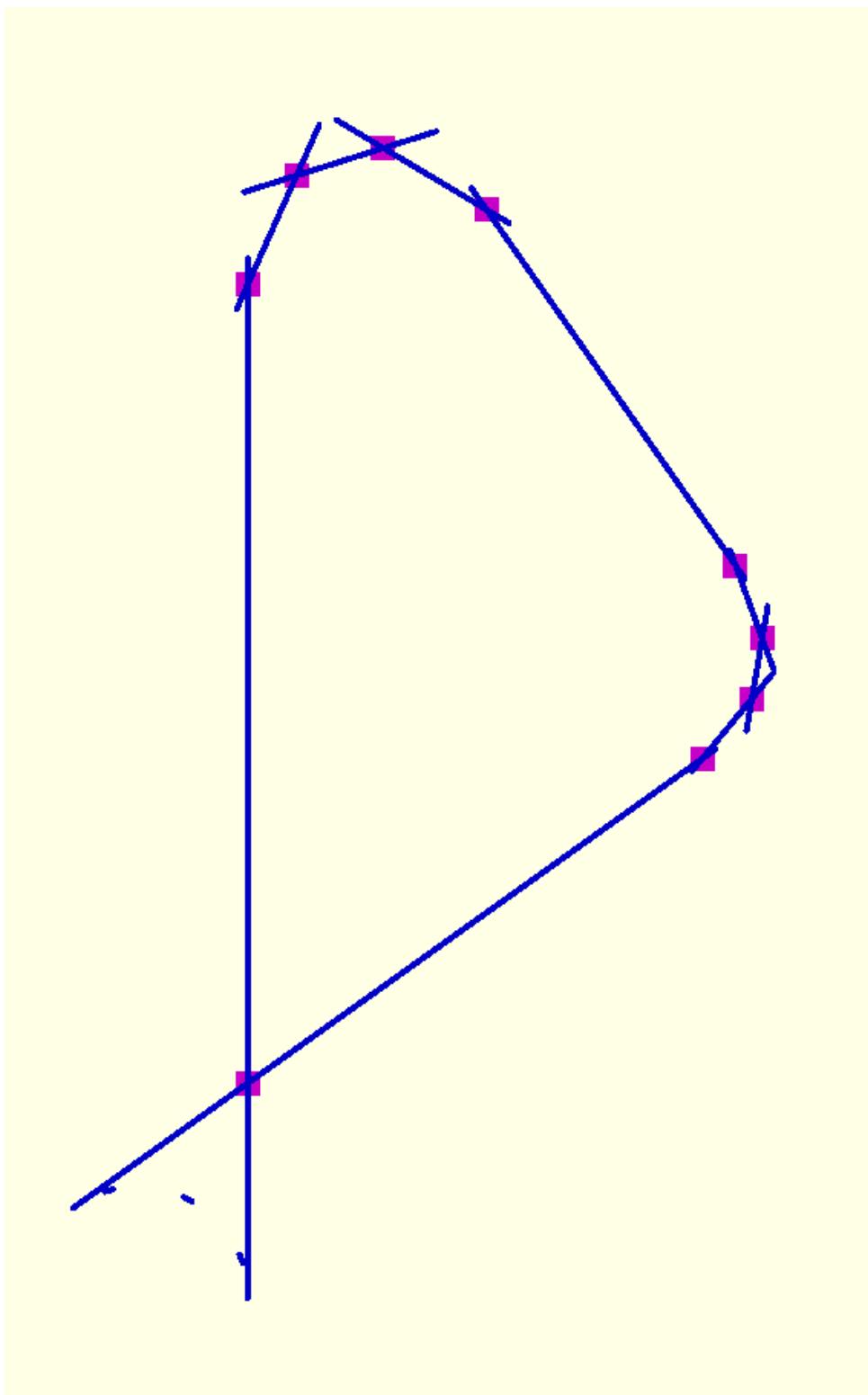
with open('trial.scad','w+')as f:
    f.write(f'''
include<dependencies2.scad>

// line segments
color("blue") for (p={sec1}) p_line(p,.05);

// intersection points
color("magenta")points({self_intersections},.2);

    ''')
t1=time.time()
t1-t0
```

Out[33]: 0.015115022659301758



```
In [34]: # example of function s_int(s)
t0=time.time()
sec=corner_radius(pts1([[0,0,2],[8,3,3],[5,7,1],[-8,0,2],[-5,20,1]]),5)
sec=seg(sec)
s=s_int(sec)

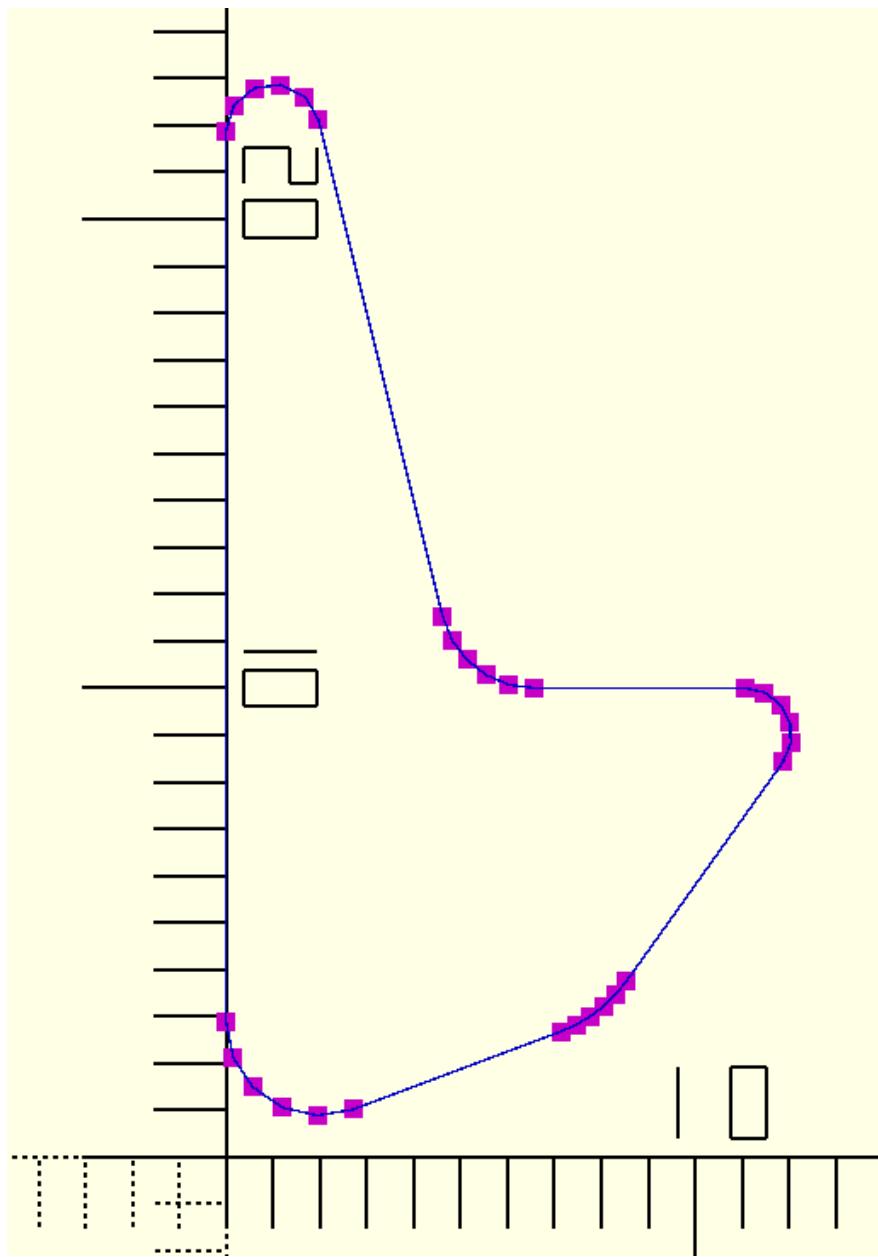
with open('trial.scad','w+')as f:
    f.write(f'''
include<dependencies2.scad>

// line segments
color("blue") for (p={sec}) p_line(p,.05);

// intersection points
color("magenta")points({s},.4);

    ''')
t1=time.time()
t1-t0
```

Out [34]: 0.023023128509521484



s_int1

```
In [35]: # example of function s_int1(s)
t0=time.time()
sec=offset_segv([[0,0],[10,0],[15,7]],-1)
self_intersections=s_int1(sec)

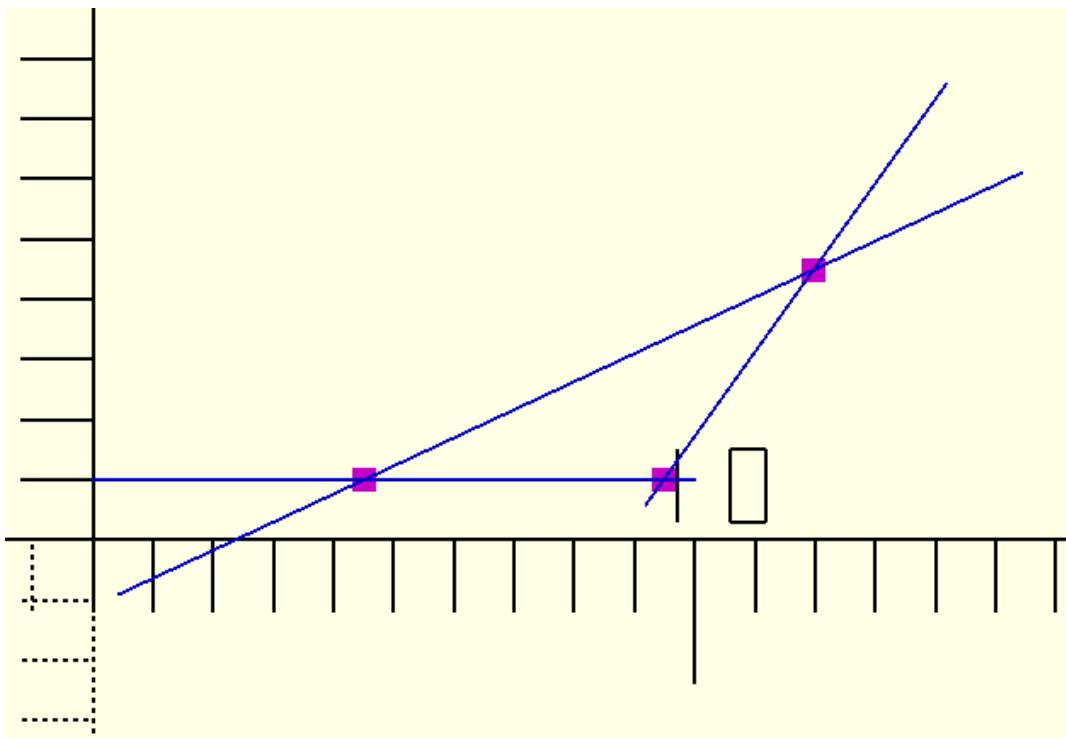
with open('trial.scad','w+')as f:
    f.write(f'''
include<dependencies2.scad>

// line segments
color("blue") for (p={sec}) p_line(p,.05);

// intersection points
color("magenta")points({self_intersections},.4);

    ''')
t1=time.time()
t1-t0
```

Out[35]: 0.004305124282836914



offset_seg_cw

```
In [36]: # example of function offset_seg_cw(sec,r)
t0=time.time()
sec=sec=corner_radius(pts1([[0,0,2],[8,3,3],[5,7,1],[-8,0,2],[-5,20,1]]),5)
sec1=offset_seg_cw(sec,-2)

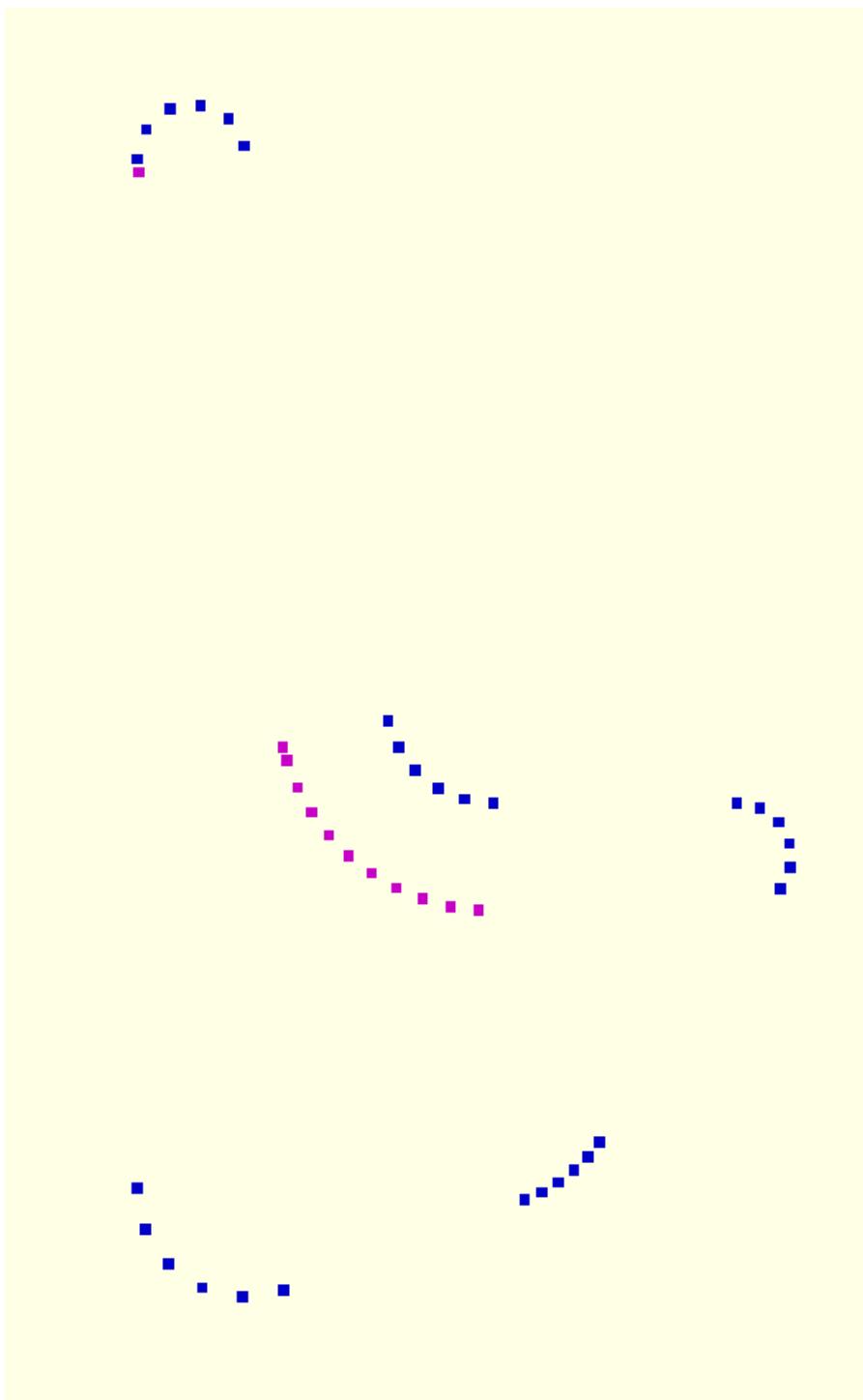
with open('trial.scad','w+')as f:
    f.write(f'''
include<dependencies2.scad>

// original section
color("blue") points({sec},.2);

// offset segments
color("magenta")points({sec1},.2);

    ''')
t1=time.time()
t1-t0
```

Out [36]: 0.013273000717163086



offset_segv

```
In [37]: # example of function offset_segv(sec,r)
t0=time.time()
sec=sec=corner_radius(pts1([[0,0,2],[8,3,3],[5,7,1],[-8,0,2],[-5,20,1]]),5)
sec1=offset_segv(sec,-1)

with open('trial.scad','w+')as f:
    f.write(f'''
include<dependencies2.scad>

// original section
color("blue") points({sec},.2);

// offset segments
color("magenta")for(p={sec1})points(p,.2);

    ''')
t1=time.time()
t1-t0
```

Out [37]: 0.009231805801391602

```
In [38]: # example of another method to offset segment where points are clockwise and example function cwv(sec
t0=time.time()
sec=corner_radius(pts1([[0,0,2],[8,3,3],[5,7,1],[-8,0,2],[-5,20,1]]),5)
sec0=seg(sec)
decision=cwv(sec)
sec1=[path_offset(sec0[i],-2) for i in range(len(sec0)) if decision[i]==1]

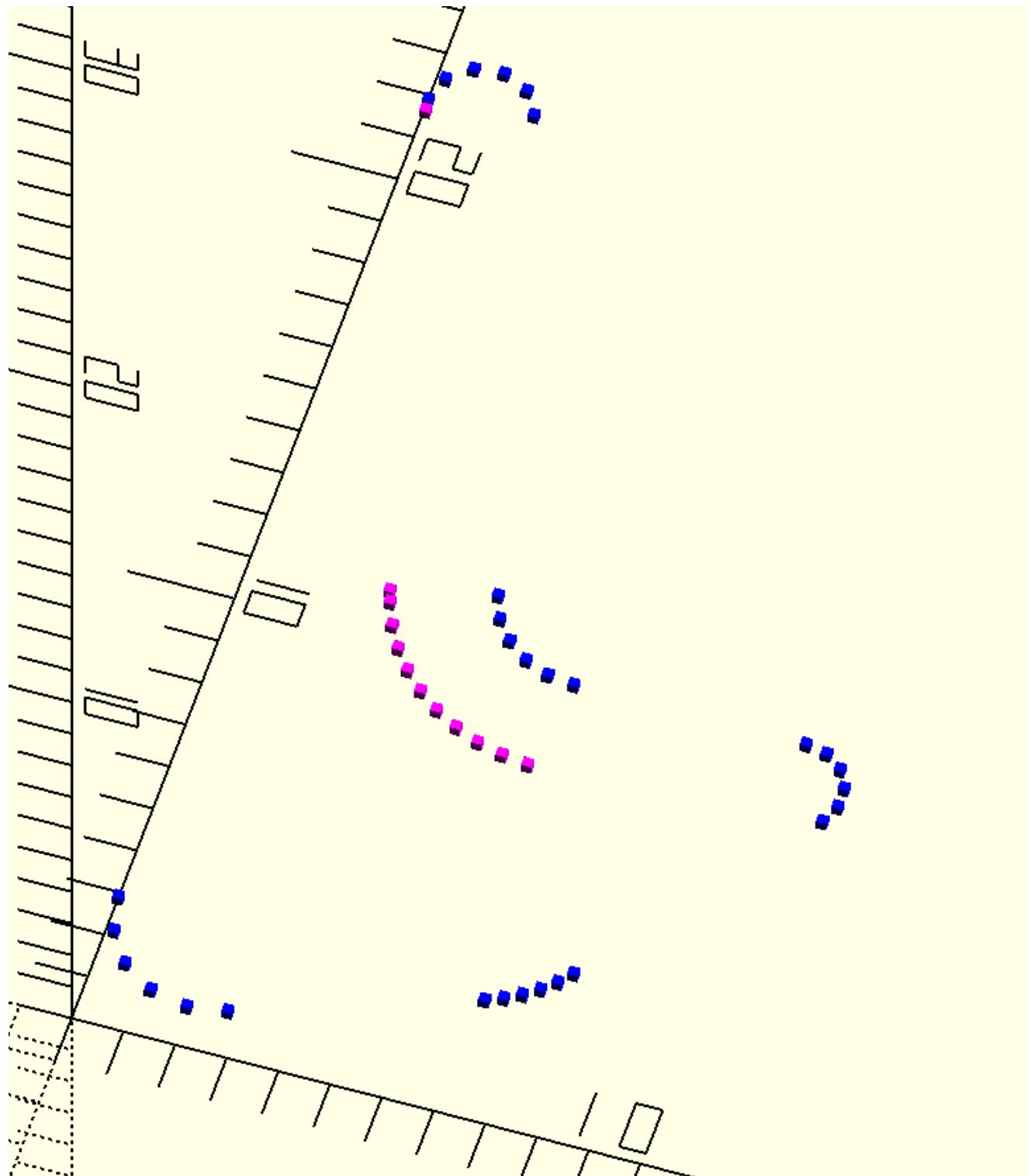
with open('trial.scad','w+')as f:
    f.write(f''''
include<dependencies2.scad>

// original section
color("blue") points({sec},.2);

// offset segments
color("magenta")for(p={sec1})points(p,.2);

''')
t1=time.time()
t1-t0
```

Out[38]: 0.01769280433654785



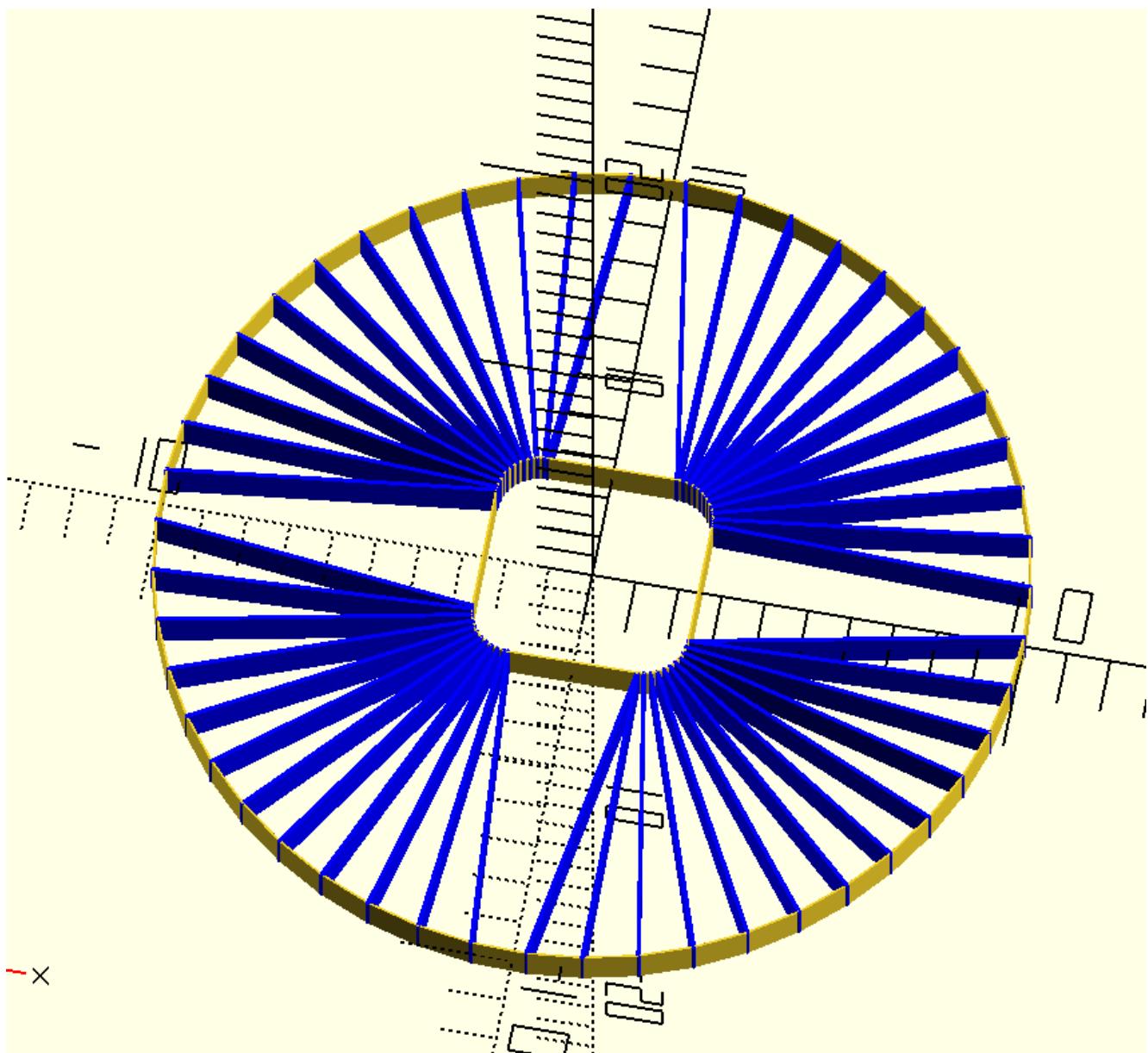
sort_points

In [39]:

```
# example of function sort_points(sec,sec1)
t0=time.time()
sec=circle(10)
sec1=corner_radius(pts1([[-2.5,-2.5,1],[5,0,1],[0,5,1],[-5,0,1]]))
sec2=sort_points(sec,sec1)
sec3=cpo([sec2,sec])
with open('trial.scad','w+')as f:
    f.write(f'''  
include<dependencies2.scad>  
  
p_line({{sec}},.05);
p_line({{sec2}},.05);
color("blue") for(p={{sec3}})p_line(p,.1);  

''')
t1=time.time()
t1-t0
```

Out[39]: 0.009630918502807617



surf_offset

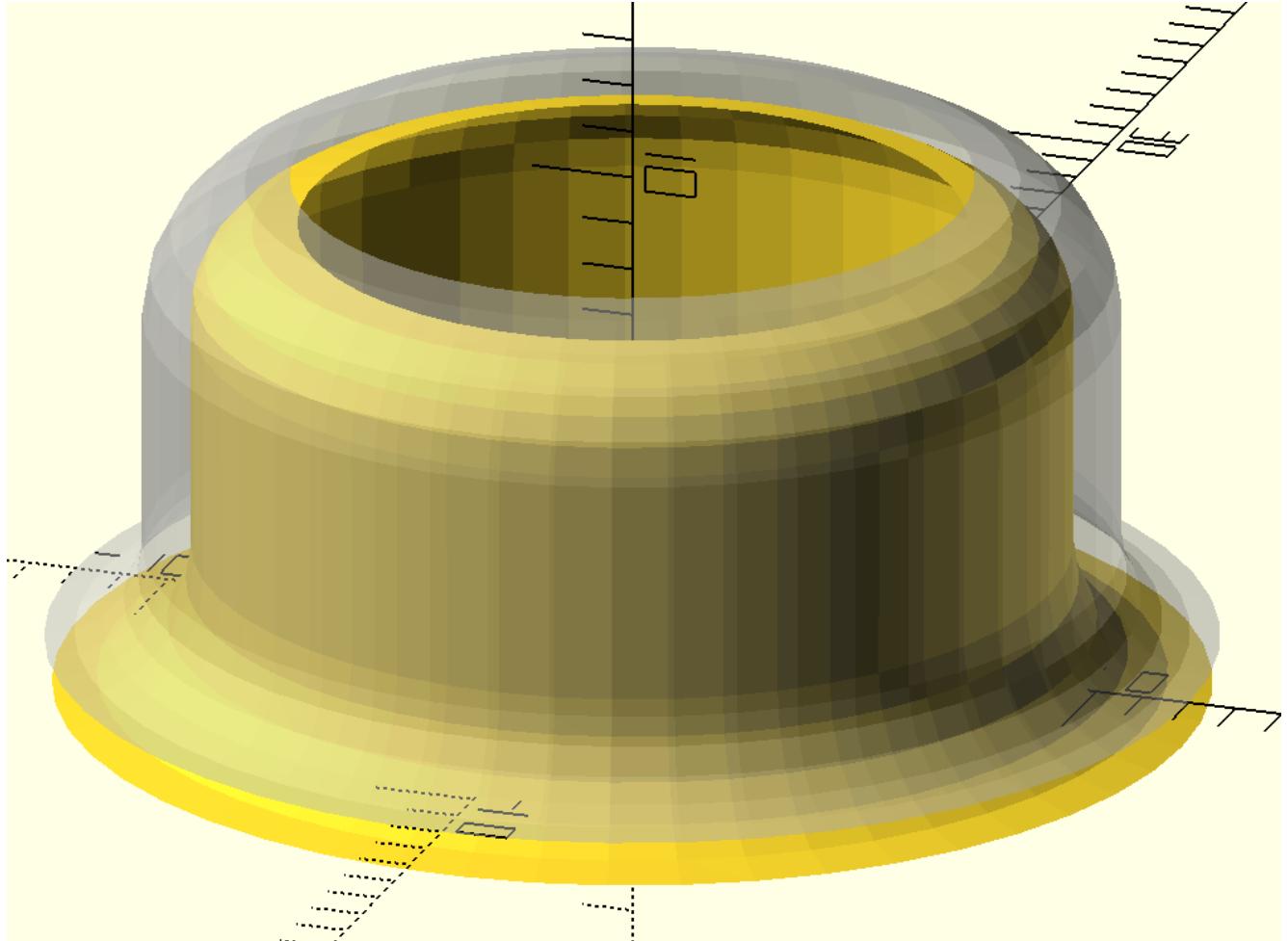
In [40]:

```
# example of function surf_offset(surf,o)
sec=circle(10);
path=corner_radius(pts1([[2,0],[-2,0,2],[0,10,3],[-3,0]]),5)
sol=prism(sec,path)
```

```
sol1=surf_offset(sol,-1)

with open('trial.scad','w+') as f:
    f.write(f'''  
include<dependencies2.scad>
%{swp_c(sol)}
{swp_c(sol1)}
```

```
...)
```



pts1

```
In [41]: # example of function pts1(p)

sec=corner_radius(pts1([[0,0,1],[10,0,1],[0,5,1],[-10,0,1]]),5)

with open('trial.scad','w+') as f:
    f.write(f'''  
include<dependencies2.scad>
color("blue")p_line({sec},.05);

...)
```



arc_2p

```
In [42]: # example of function arc_2p(p1,p2,r,cw=1,s=20)

p1=[2,0]
p2=[0,2]
arc1=arc_2p(p1,p2,2,1,20)

arc2=arc_2p(p1,p2,2,-1,20)

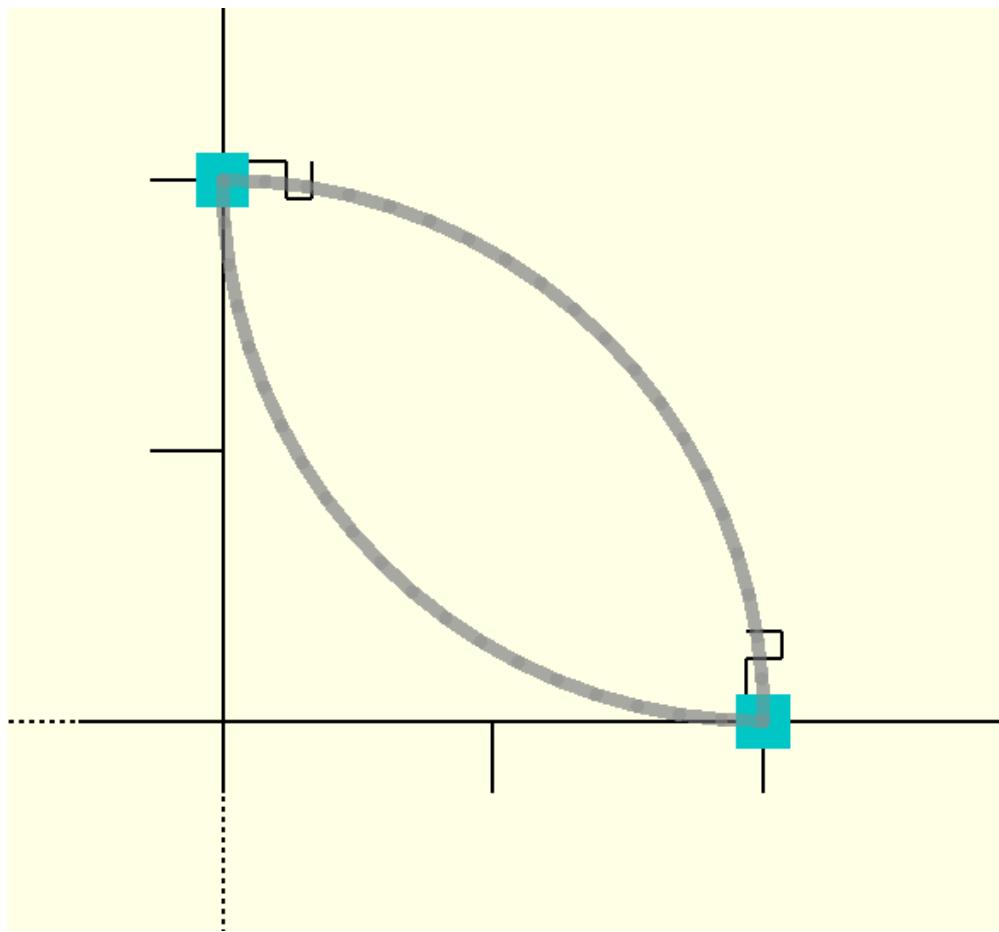
with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>

// arc clockwise
//color("blue")
%p_lineo({arc1},.05);

// arc counter clockwise
//color("magenta")
%p_lineo({arc2},.05);

color("cyan")points({[p1,p2]},.2);

''' )
```



arc_long_2p

```
In [43]: # example of function arc_long_2p(p1,p2,r,cw=1,s=20)

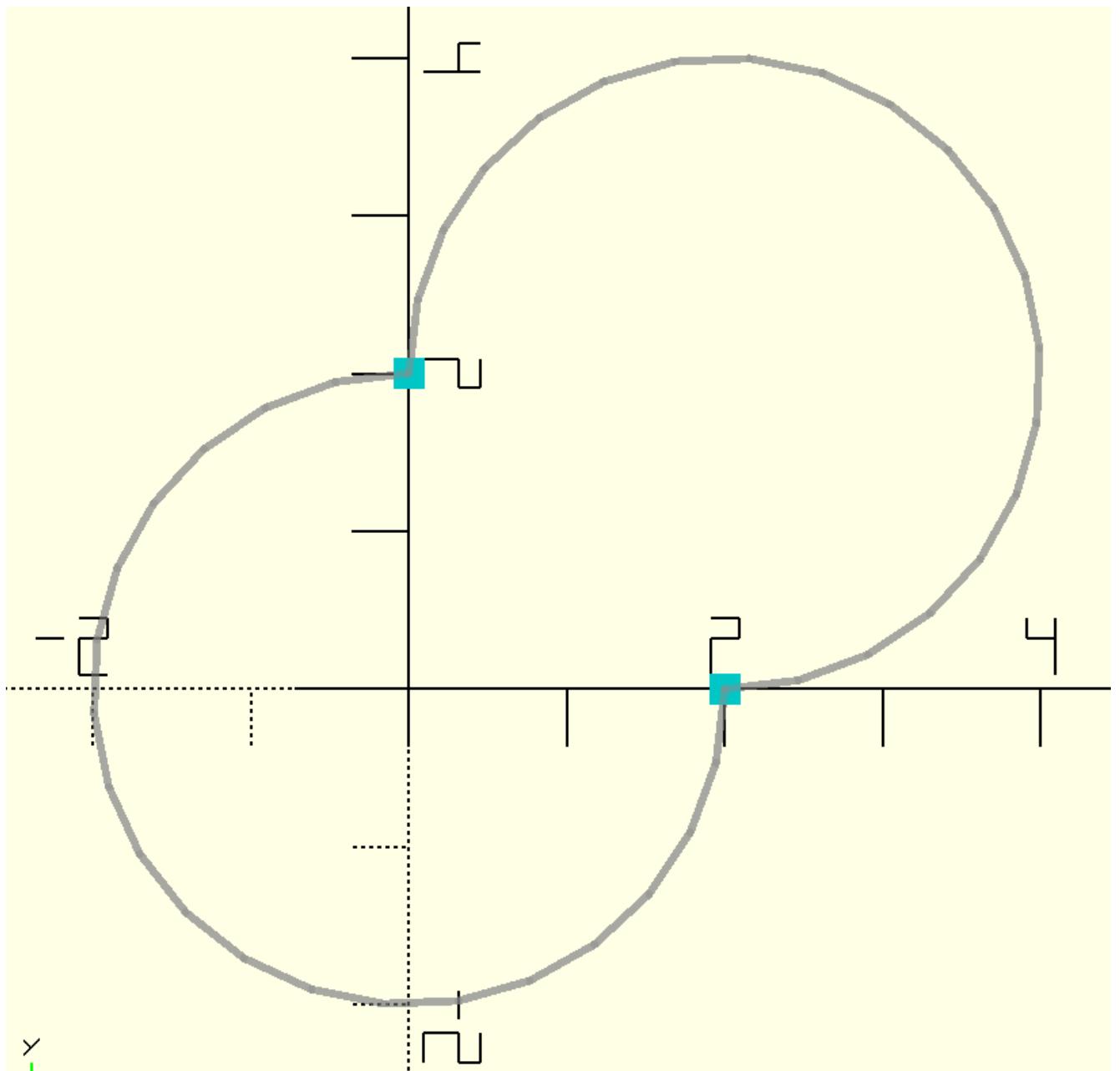
p1=[2,0]
p2=[0,2]
arc1=arc_long_2p(p1,p2,2,1,20)
arc2=arc_long_2p(p1,p2,2,-1,20)

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>

// arc clockwise
//color("blue")
%p_lineo({arc1},.05);

// arc counter clockwise
//color("magenta")
%p_lineo({arc2},.05);

color("cyan")points({[p1,p2]},.2);
''' )
```



arc_2p_cp

In [44]:

```
# example of function arc_2p_cp(p1,p2,r,cw=-1)

p1=[2,0]
p2=[0,2]
arc1=arc_2p(p1,p2,2,1,5)
cp1=arc_2p_cp(p1,p2,2,1)

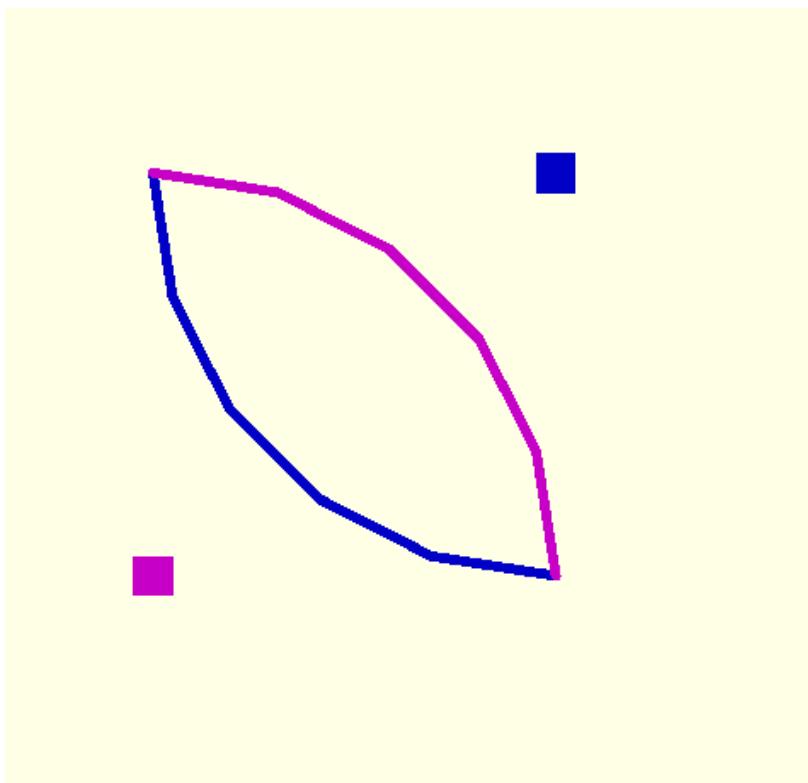
arc2=arc_2p(p1,p2,2,-1,5)
cp2=arc_2p_cp(p1,p2,2,-1)

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>

// arc clockwise with center point of the arc
color("blue")
{{p_lineo({arc1},.05);
points({[cp1]},.2);}}

// arc counter clockwise with center point of the arc
//color("magenta")
//{{p_lineo({arc2},.05);
//points({[cp2]},.2);}}
```

...)



offset

```
In [50]: # example of function offset(sec,r)

t0=time.time()

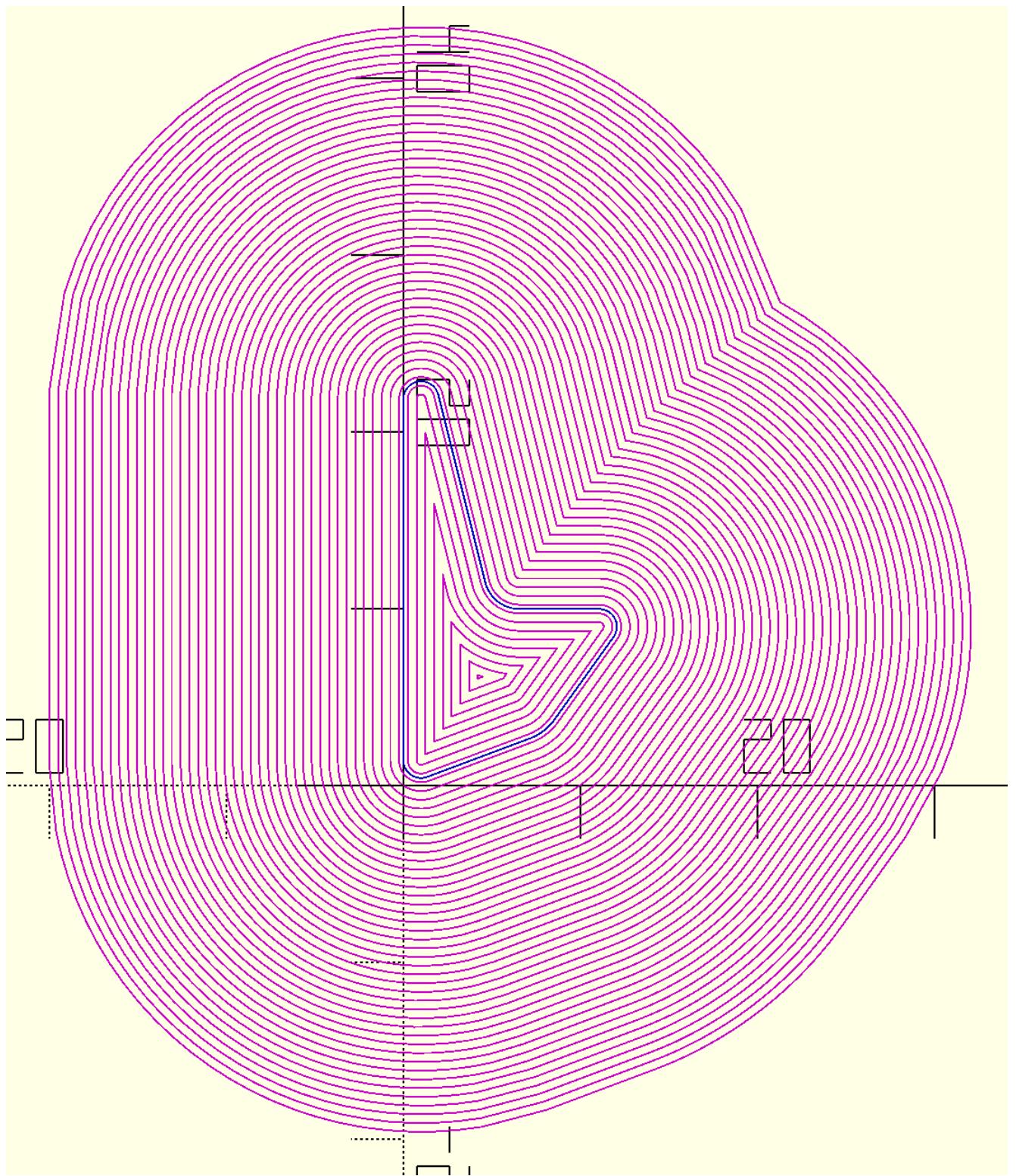
sec=corner_radius(pts1([[0,0,1],[8,3,3],[5,7,1],[-8,0,2],[-5,20,1]]),30)
# sec=corner_radius(pts1([[0,0,1],[7,5,2],[5,7,3],[-5,7,2],[-7,5,3]]),10)
# sec=corner_radius(pts1([[-15,0,2.4],[0,15,3],[30,0,3],[0,-15,2.4],[5,0,2.4],[0,20,7],[-40,0,7],[0,-2
# sec=corner_radius(pts1([[0,0,1],[10,0,1],[0,5,1],[-10,0,1]]),10)
# sec=circle(10)
# sec=pts([[0,0],[10,0],[0,5],[-10,0]])

# path=corner_radius(pts1([[2,0],[-2,0,2],[0,7,4.2],[-4.2,0]]),40)
# path=corner_radius(pts1([[2,0],[-2,0,2],[0,7,5.3],[-5.3,0]]),40)
# path=corner_radius(pts1([[2,0],[-2,0,2],[0,7,2],[-2,0]]),20)
os=linspace(-4.2,20,50)
sec1=[offset(sec,i) for i in os]
with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>

color("magenta")for(p={sec1})p_line(p,.1);
color("blue")p_line({sec},.1);

''')
t1=time.time()
t1-t0
```

Out [50]: 2.0691728591918945



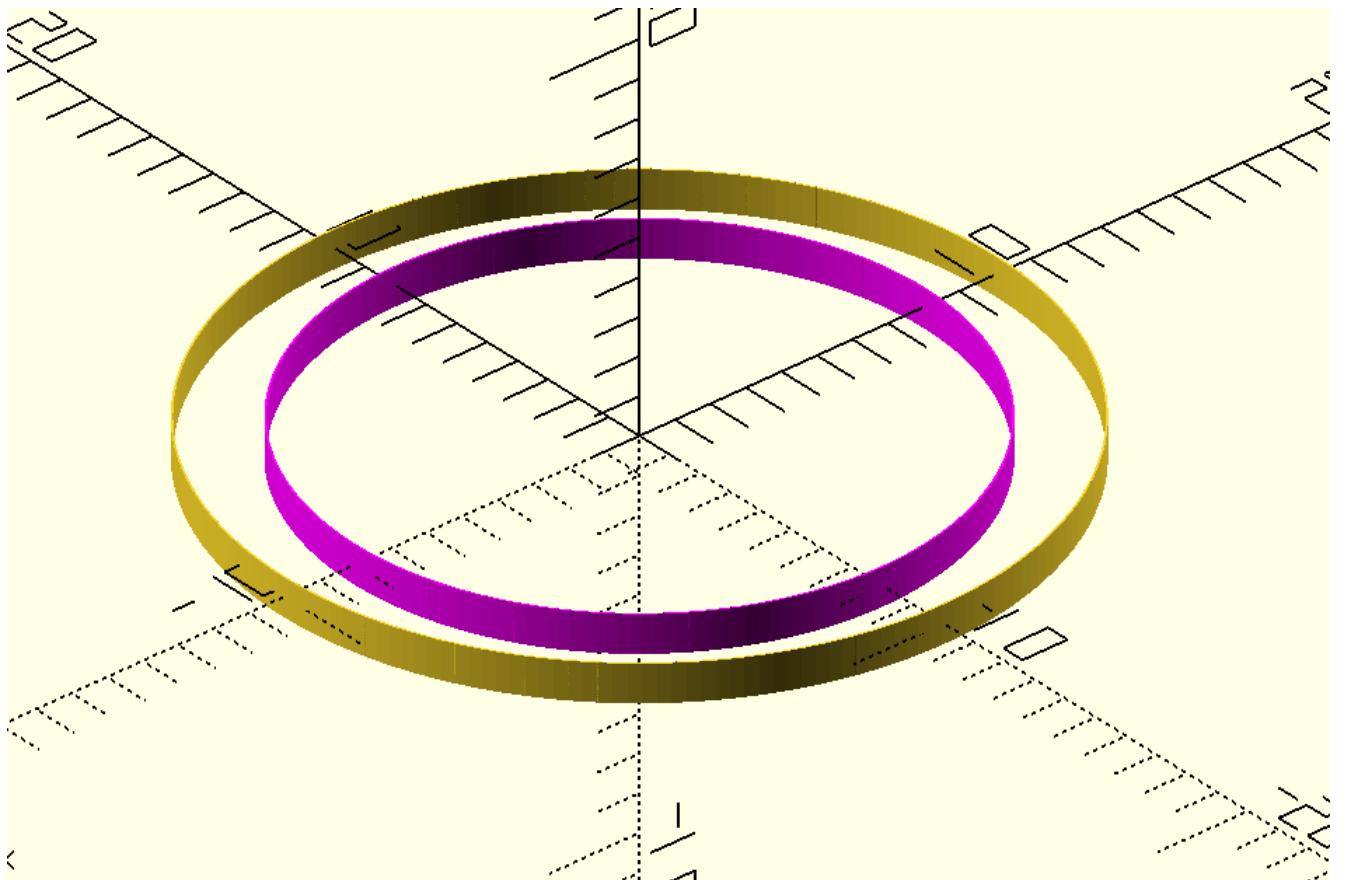
```
In [51]: t0=time.time()
r==2
# sec=corner_radius(pts1([[0,0,1],[7,5,2],[5,7,3],[-5,7,5],[-7,5,5]]),10)
sec=circle(10,s=200)
sec1=oset(sec,r)

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>
p_line({sec},.05);

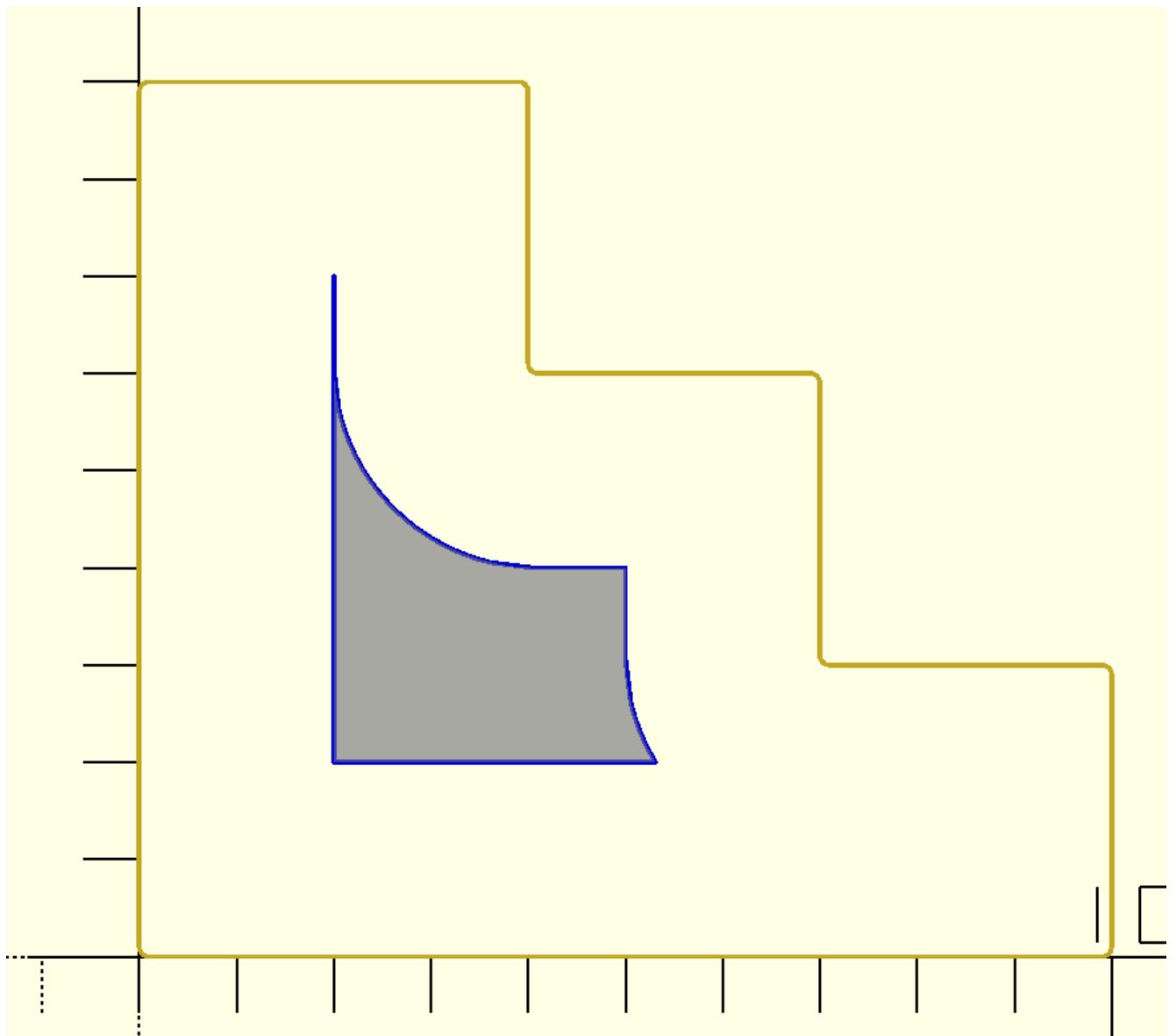
color("magenta")p_line({sec1},.05);
''')

t1=time.time()
t1-t0
```

Out[51]: 0.04294109344482422



```
In [53]: sec=corner_radius(pts1([[0,0,.1],[10,0,.1],[0,3,.1],[-3,0,.1],[0,3,.1],[-3,0,.1],[0,3,.1],[-4,0,.1]]),  
  
# sec=corner_radius(pts1([[0,0,1],[8,3,3],[5,7,1],[-8,0,2],[-5,20,1]]),40)  
# sec=sec=corner_radius(pts1([[0,0,1],[7,5,2],[5,7,3],[-5,7,5],[-7,5,5]]),20)  
# sec=corner_radius(pts1([-15,0,2.49],[0,15,3],[30,0,3],[0,-15,2.49],[5,0,2.49],[0,20,7],[-40,0,7],[0  
# sec=circle(10)  
d=-2  
  
with open('trial.scad','w+') as f:  
    f.write(f'''  
include<dependencies2.scad>  
p_line({sec},.05);  
color("blue")p_line({offset(sec,d)},.05);  
%offset({d})polygon({sec});  
'''')
```



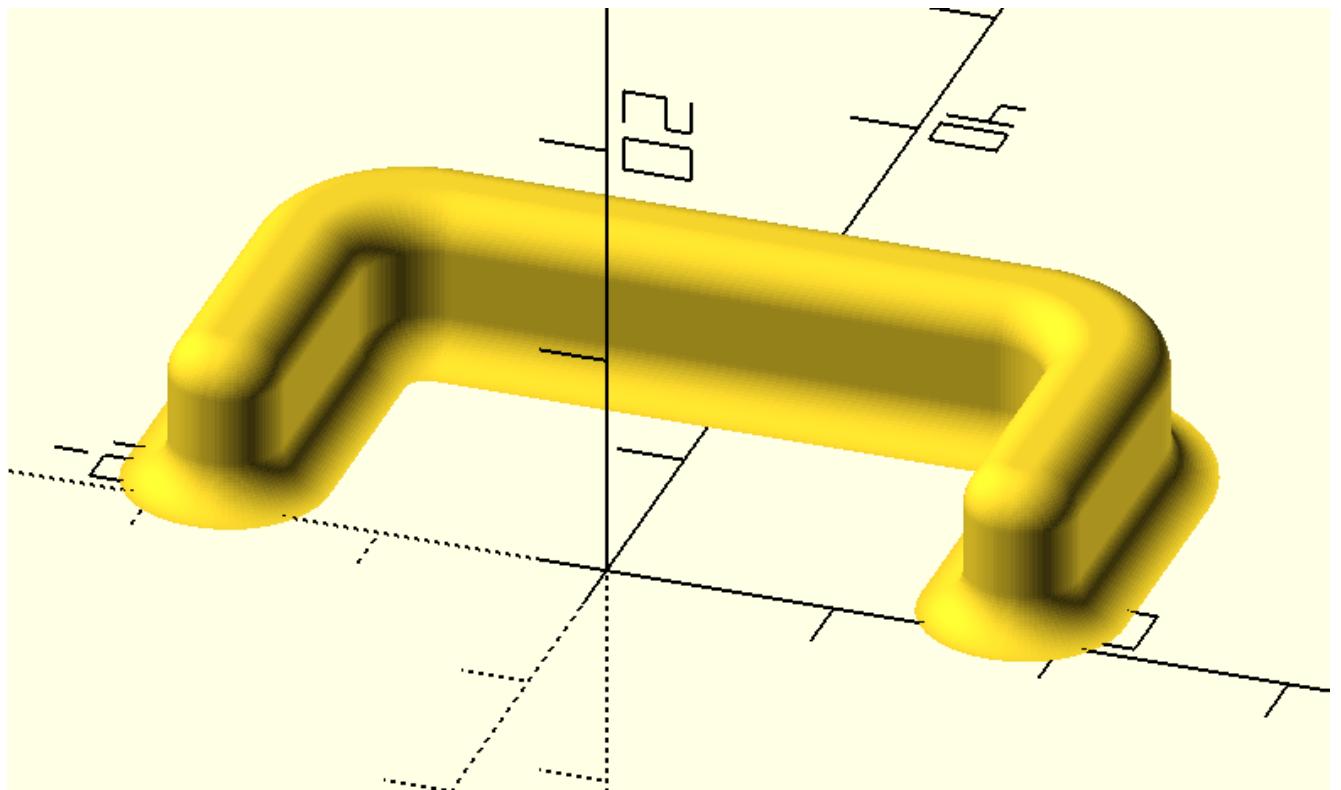
prism

```
In [54]: # example of function prism(sec,path) and function swp_prism_h(prism_big, prism_small)
t0=time.time()
# sec=corner_radius(pts1([[0,0,.2],[8,3,3],[5,7,1],[-8,0,2],[-5,20,1]]),20)
# sec=corner_radius(pts1([[0,0,1],[7,5,2],[5,7,3],[-5,7,2],[-7,5,3]]),20)
sec=corner_radius(pts1([[[-15,0,2.4],[0,15,3],[30,0,3],[0,-15,2.4],[5,0,2.4],[0,20,7],[-40,0,7],[0,-20,
# sec=pts([[0,0],[10,0],[0,5],[-10,0]])
# path=corner_radius(pts1([[2,0],[-2,0,2],[0,7,4.2],[-4.2,0]]),20)
# path=corner_radius(pts1([[2,0],[-2,0,2],[0,7,5.2],[-5.2,0]]),20)
path=corner_radius(pts1([[2,0],[-2,0,2],[0,7,2],[-2,0]]),20)

sol=prism(sec,path)
# sol1=prism(sec,path_offset(path,-.5))
# sol2=swp_prism_h(sol,sol1)

with open('trial.scad','w+') as f:
    f.write('''
{swp(sol)}
'''')
t1=time.time()
t1-t0
```

Out[54]: 0.8446097373962402



f_prism(sec,path)

```
In [55]: # example of function prism(sec,path) and function swp_prism_h(prism_big, prism_small)
t0=time.time()
# sec=corner_radius(pts1([[0,0,.2],[8,3,3],[5,7,1],[-8,0,2],[-5,20,1]]),20)
# sec=corner_radius(pts1([[0,0,1],[7,5,2],[5,7,3],[-5,7,2],[-7,5,3]]),20)
sec=corner_radius(pts1([[[-15,0,2.4],[0,15,3],[30,0,3],[0,-15,2.4],[5,0,2.4],[0,20,7],[-40,0,7],[0,-20,
# sec=pts([[0,0],[10,0],[0,5],[-10,0]])
# path=corner_radius(pts1([[2,0],[-2,0,2],[0,7,4.2],[-4.2,0]]),20)
# path=corner_radius(pts1([[2,0],[-2,0,2],[0,7,5.2],[-5.2,0]]),20)
path=corner_radius(pts1([[2,0],[-2,0,2],[0,7,2],[-2,0]]),20)

sol=f_prism(sec,path)
# sol1=f_prism(sec,path_offset(path,-.5))
# sol2=f_swp_prism_h(sol,sol1)

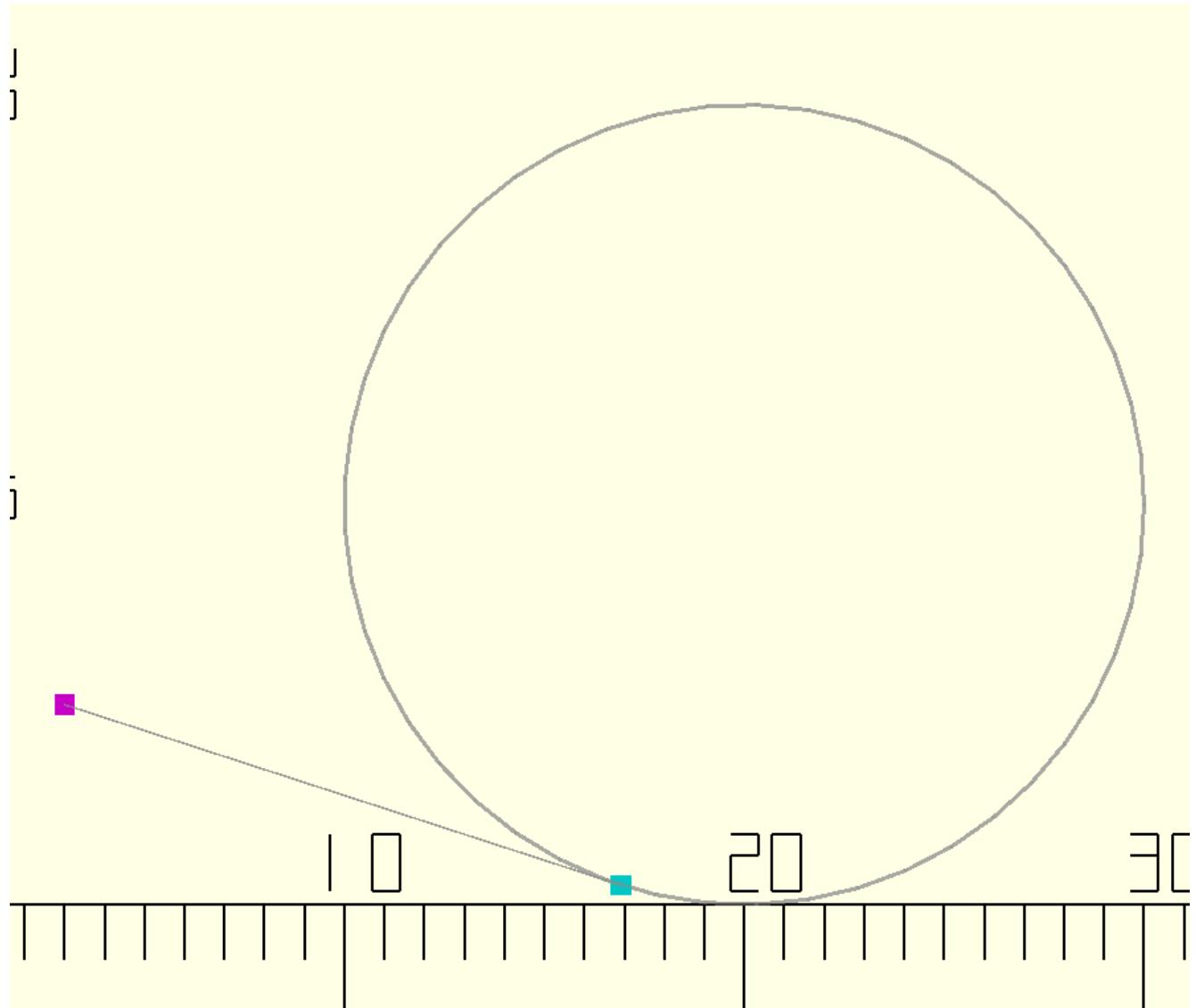
with open('trial.scad','w+') as f:
    f.write(f'''
{swp(sol)}
...))

t1=time.time()
t1-t0
```

Out [55]: 0.8629240989685059

p_cir_t

```
In [56]: # example of function p_cir_t(pnt,cir)
cir=c3t2(translate([20,10,0],circle(10)))
point=[3,5]
tangent_point=p_cir_t(point,cir)
with open('trial.scad','w+')as f:
    f.write(f'''  
include<dependencies2.scad>  
  
%p_line({{cir}},.1);
color("magenta")points({{point}},.5);
color("cyan")points({{tangent_point}},.5);
%p_line({{point,tangent_point}},.05);
'''')
```

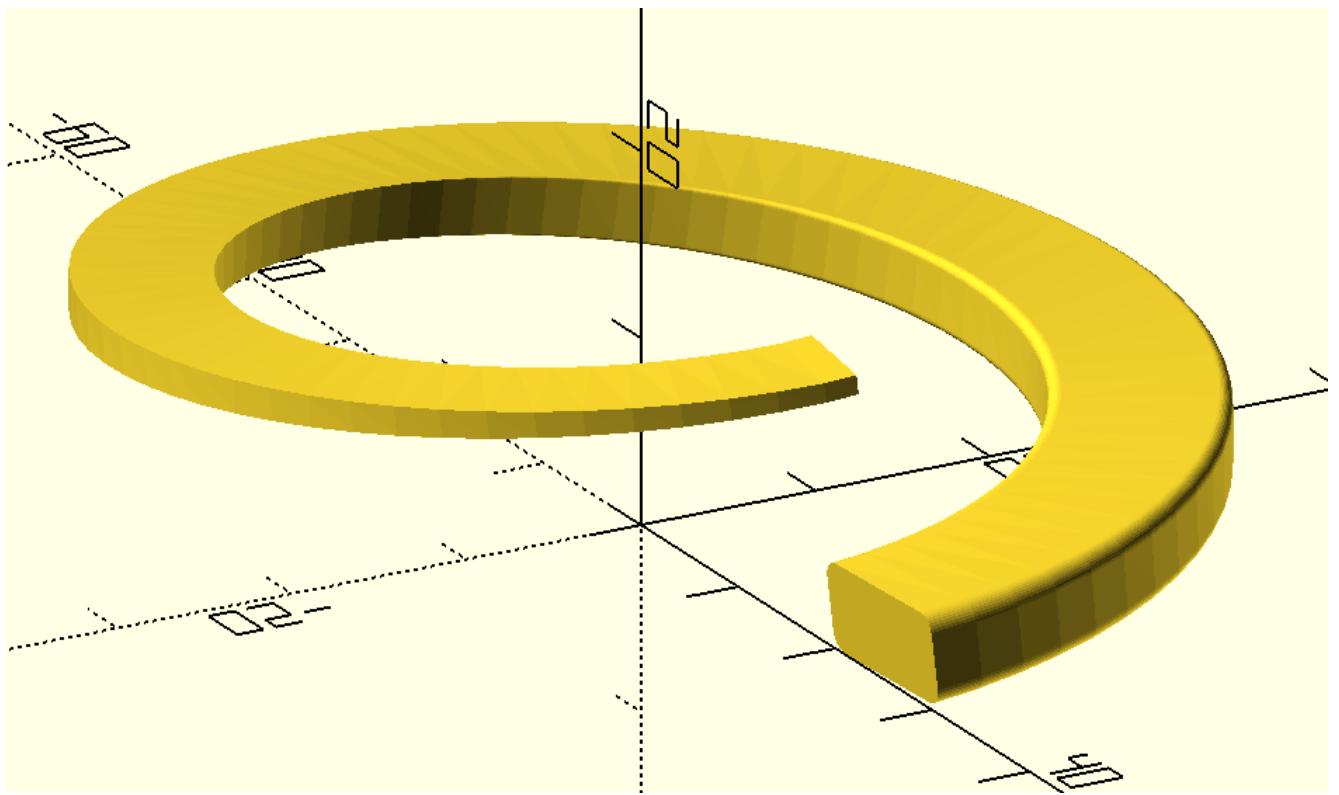


v_sec_extrude

```
In [57]: # example of function v_sec_extrude(sec,path,o)
sec=corner_radius(pts1([[0,0,1],[10,0,1],[0,5,1],[-10,0,1]]),10)
path=helix(20,15,1,5)

sol=v_sec_extrude(sec,path,-2)

with open('trial.scad','w+')as f:
    f.write(f'''  
include<dependencies2.scad>  
  
{swp(sol)}  
'''')
```



t_cir_tarc

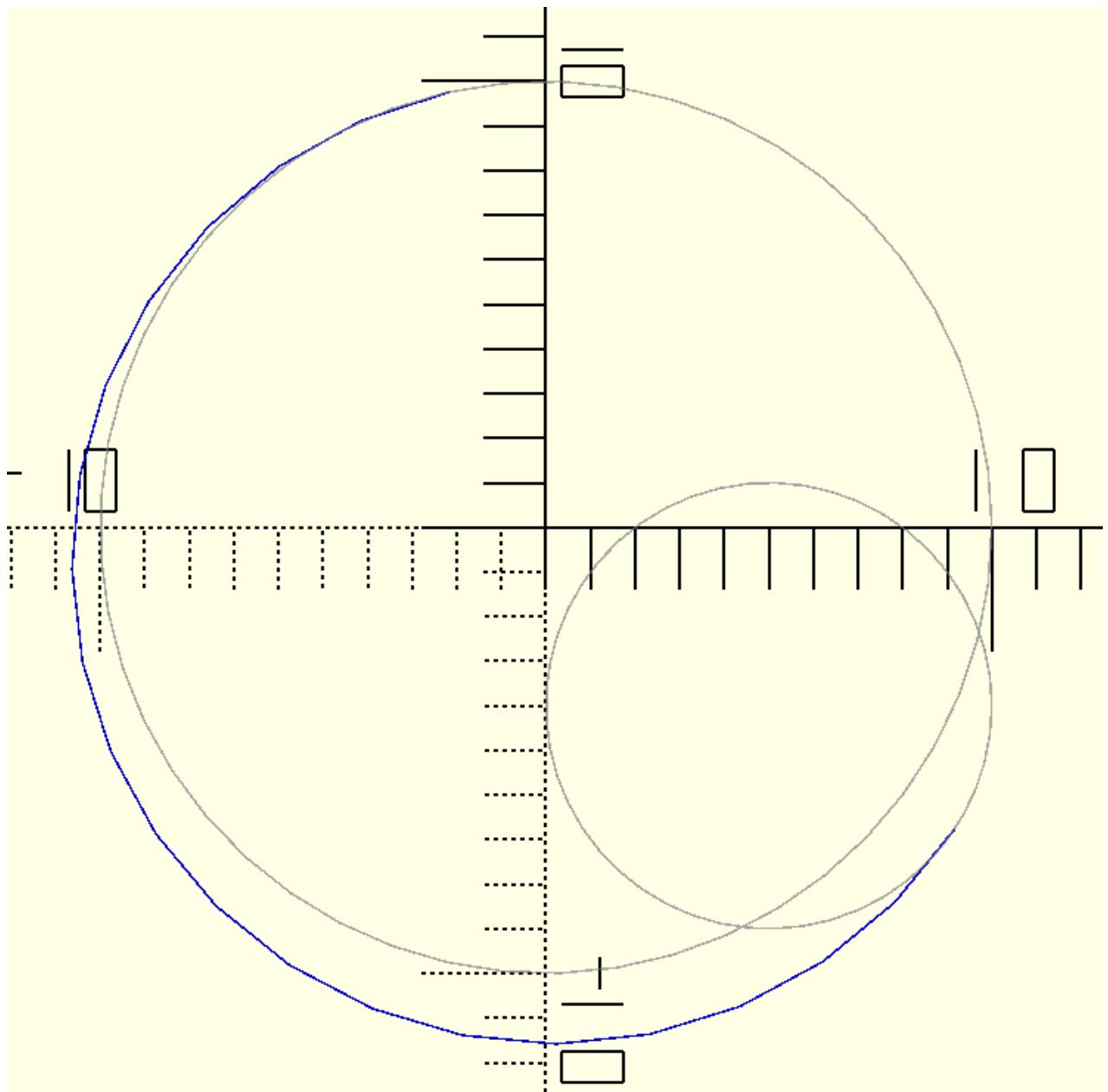
```
In [58]: # example of function t_cir_tarc(r1,r2,cp1,cp2,r,side=0,s=50)

r=17
arc1=t_cir_tarc(10,5,[0,0],[-15,10],r,0,20)

c1=circle(10)
c2=circle(5,[-15,10])
with open('trial.scad','w+')as f:
    f.write(f'''
include<dependencies2.scad>

%p_line({{c1}},.05);
%p_line({{c2}},.05);
arc1={arc1};
color("blue") p_line3d(arc1,.1);
color("magenta")points({{[arc1[0],arc1[-1]]}},.2);

    ''')
```



```
In [59]: # example of function t_cir_tarc

cir1=circle(20)
cir2=circle(10,[52.5*cos(d2r(30)),52.5*sin(d2r(30))])
cp1,cp2=[52.5*cos(d2r(30)),52.5*sin(d2r(30))],[0,0]

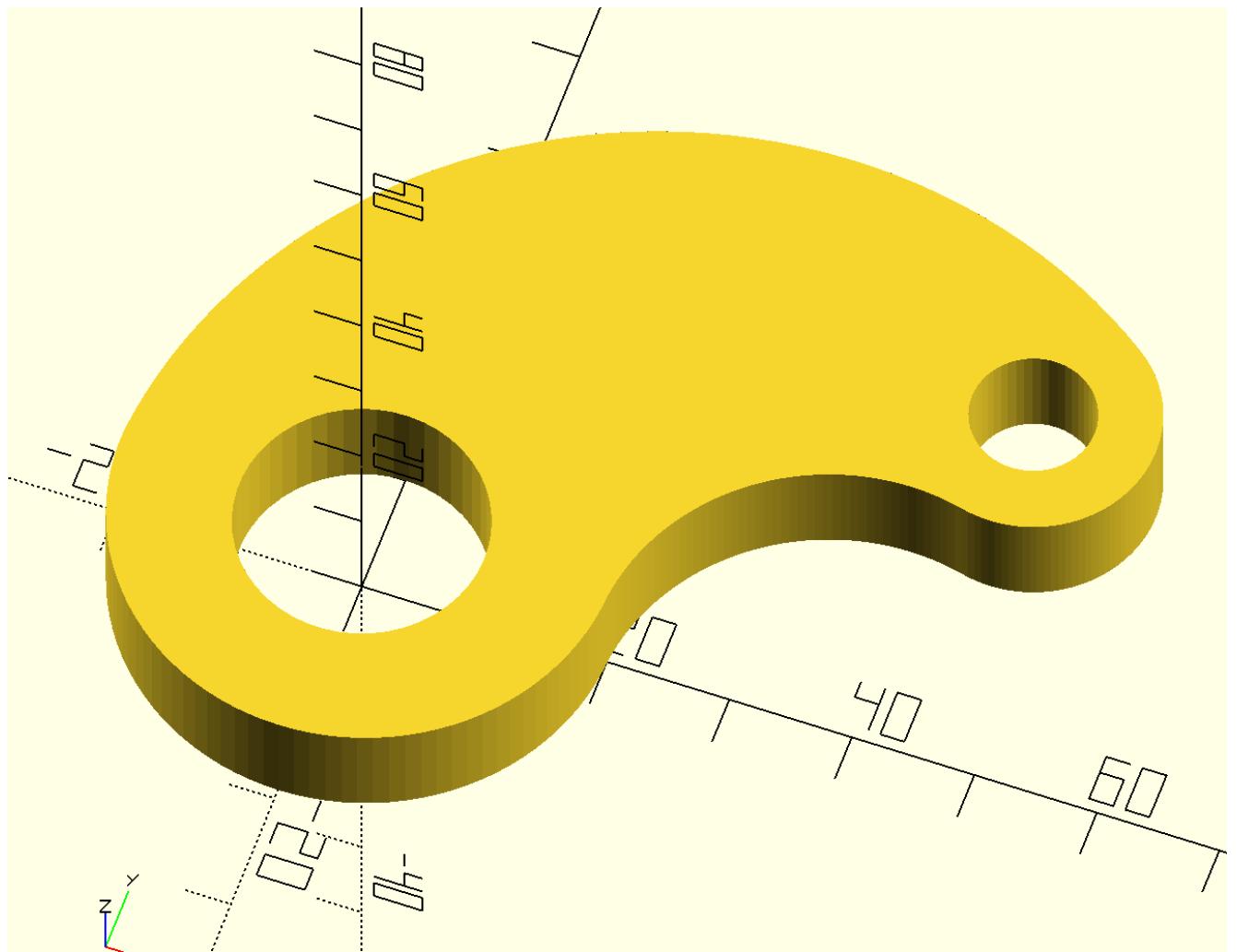
arc1=t_cir_tarc(10,20,cp1,cp2,45,0)
arc3=t_cir_tarc(10,20,cp1,cp2,18.75,1)
arc2=arc_long_2p(arc1[-1],arc3[0],20,-1)[1:-1]
arc4=arc_2p(arc3[-1],arc1[0],10,-1)[1:-1]

sec=arc1+arc2+arc3+arc4
sol=linear_extrude(sec,10)
sol1=translate([0,0,-.5],translate(c2t3(cp1),cylinder(r=5,h=11)))
sol2=translate([0,0,-.5],translate(c2t3(cp2),cylinder(r=10,h=11)))

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>
//p_line3d({cir1},.1);
//p_line3d({cir2},.1);
//sec={sec};
//color("magenta")p_line3d(loop(sec,0,len(sec)*$t),.5);
difference(){{
{swp(sol)}
{swp(sol1)}
{swp(sol2)}\n}}\n'''')
```

```
arc1[-1]
```

```
Out[59]: [-19.93067428455553, 1.6637976328138104]
```



tcct

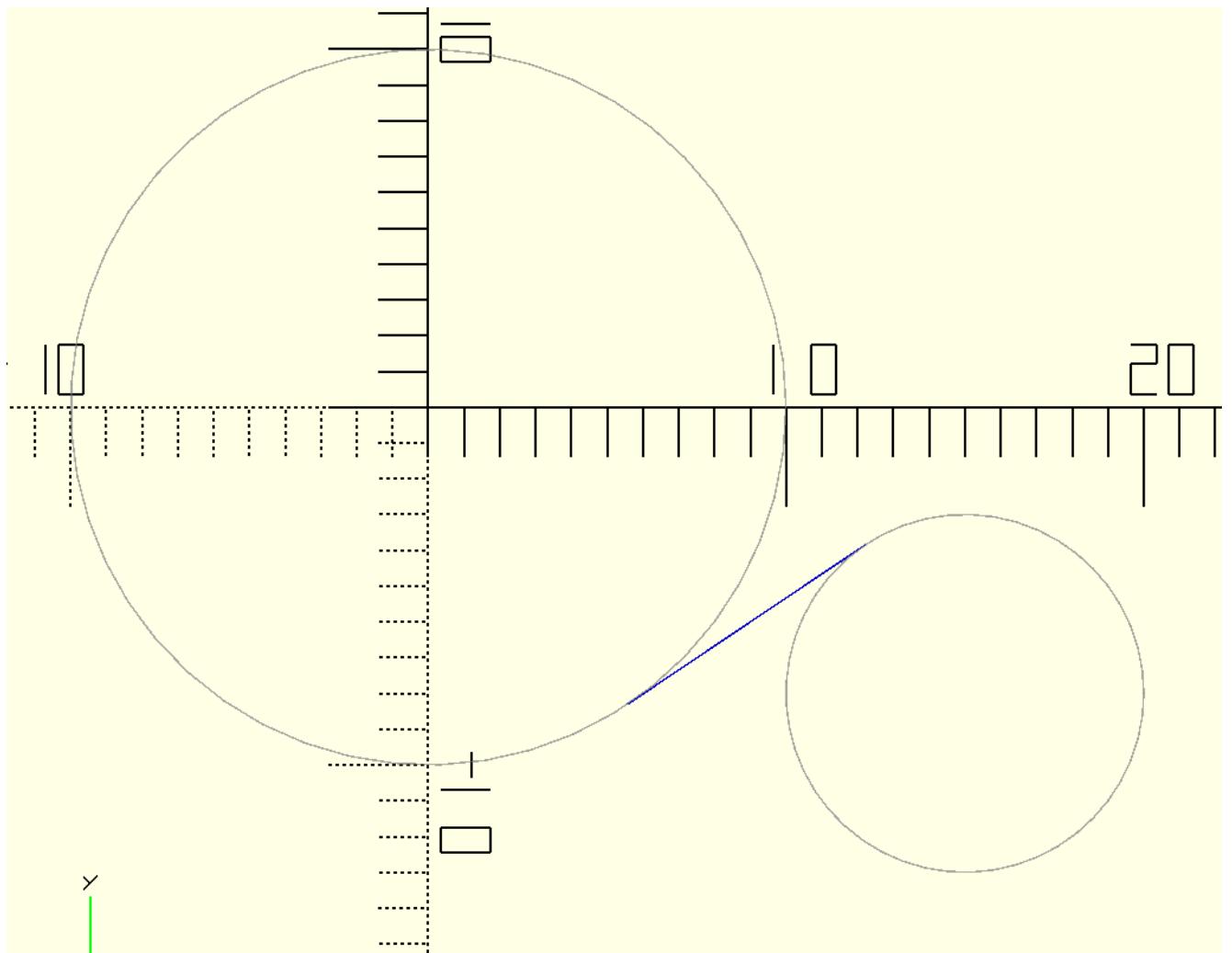
```
In [60]: # example of function tcct(r1,r2,cp1,cp2,cw=-1)
c1=circle(10)
c2=circle(5,[15,-8])

line= tcct(10,5,[0,0],[15,-8],cw=-1)

with open('trial.scad','w+')as f:
    f.write(f''''
include<dependencies2.scad>

%p_line({c1},.05);
%p_line({c2},.05);
color("blue") p_lineo({line},.05);

    ''')
```

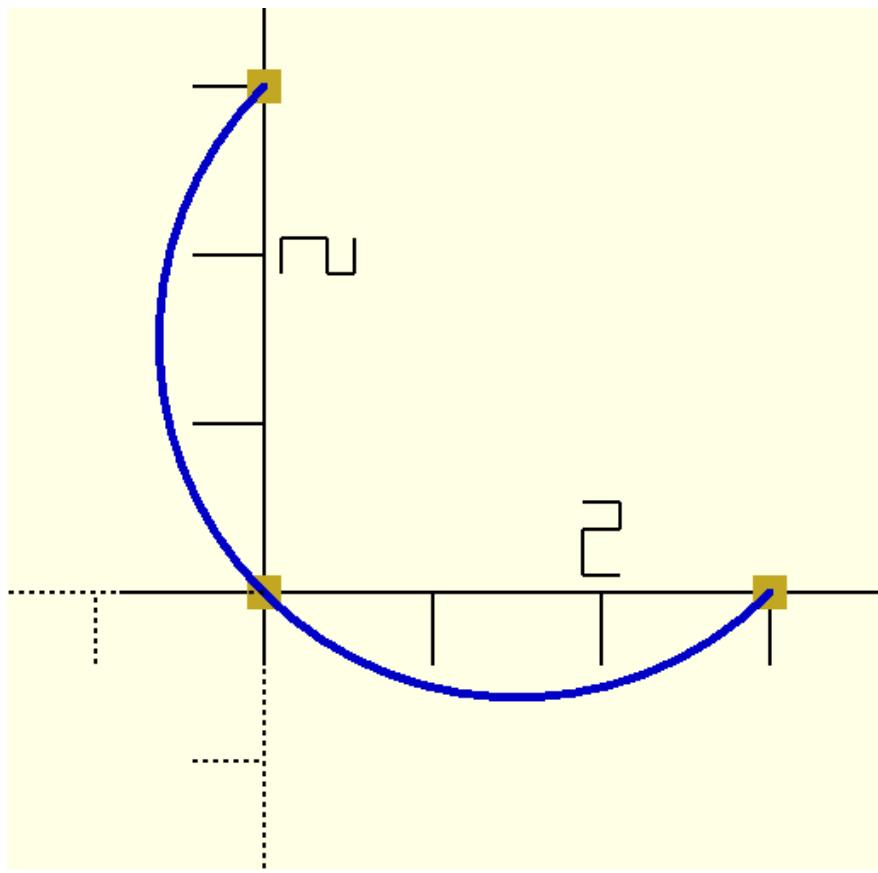


arc_3p

```
In [61]: # example of function arc_3p(p1,p2,p3,s=30)

p1,p2,p3=[3,0],[0,0],[0,3]
arc1=arc_3p(p1,p2,p3)
with open('trial.scad','w+')as f:
    f.write(f''''
include<dependencies2.scad>

color("blue") p_lineo({arc1},.05);
points({[p1,p2,p3]},.2);
'''')
```

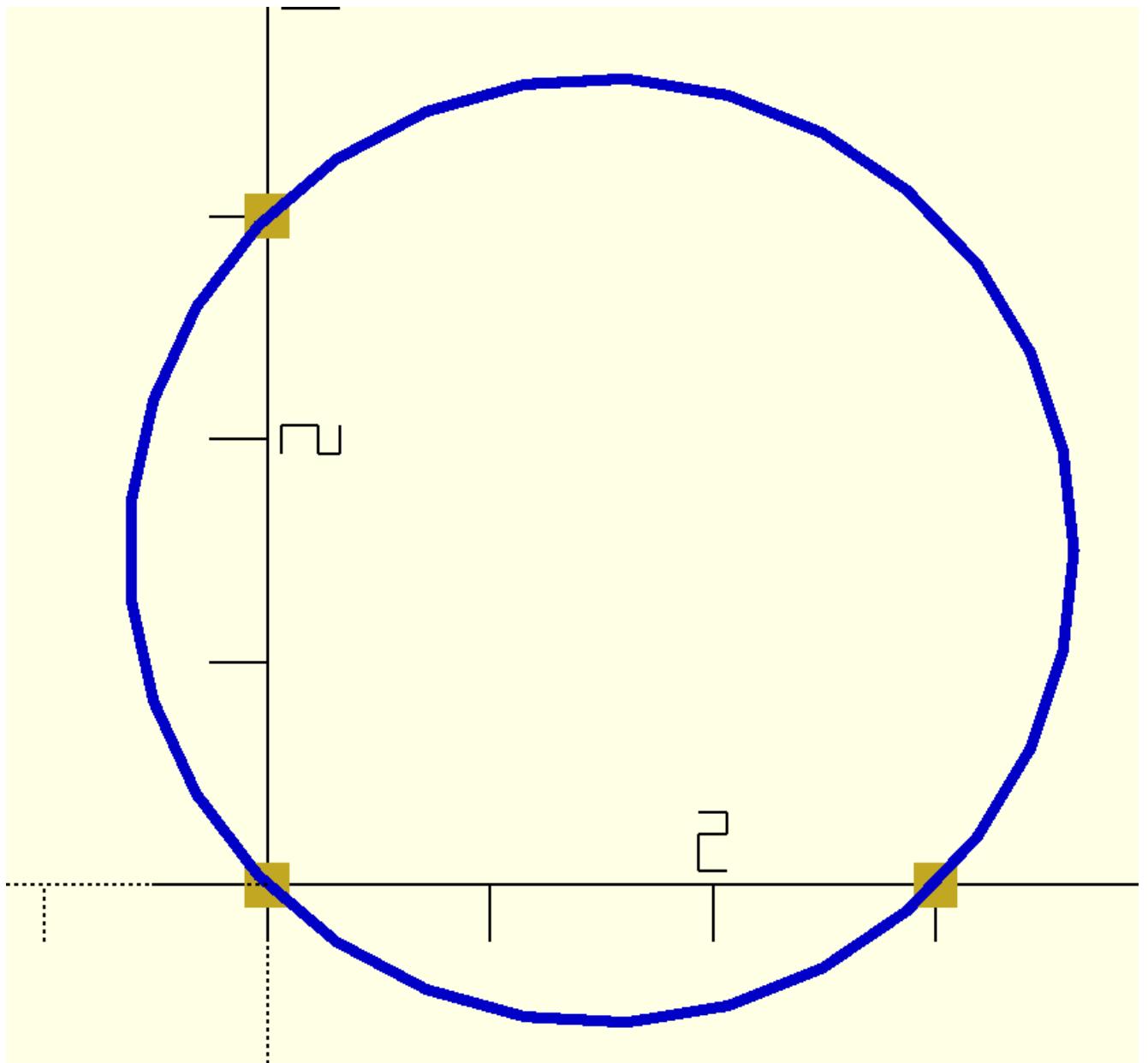


cir_3p

```
In [62]: # example of function cir_3p(p1,p2,p3,s=30)
```

```
p1,p2,p3=[3,0],[0,0],[0,3]
cir=cir_3p(p1,p2,p3,30)
with open('trial.scad','w+')as f:
    f.write(f'''
include<dependencies2.scad>

color("blue") p_line({cir},.05);
points({[p1,p2,p3]},.2);
    ''')
```

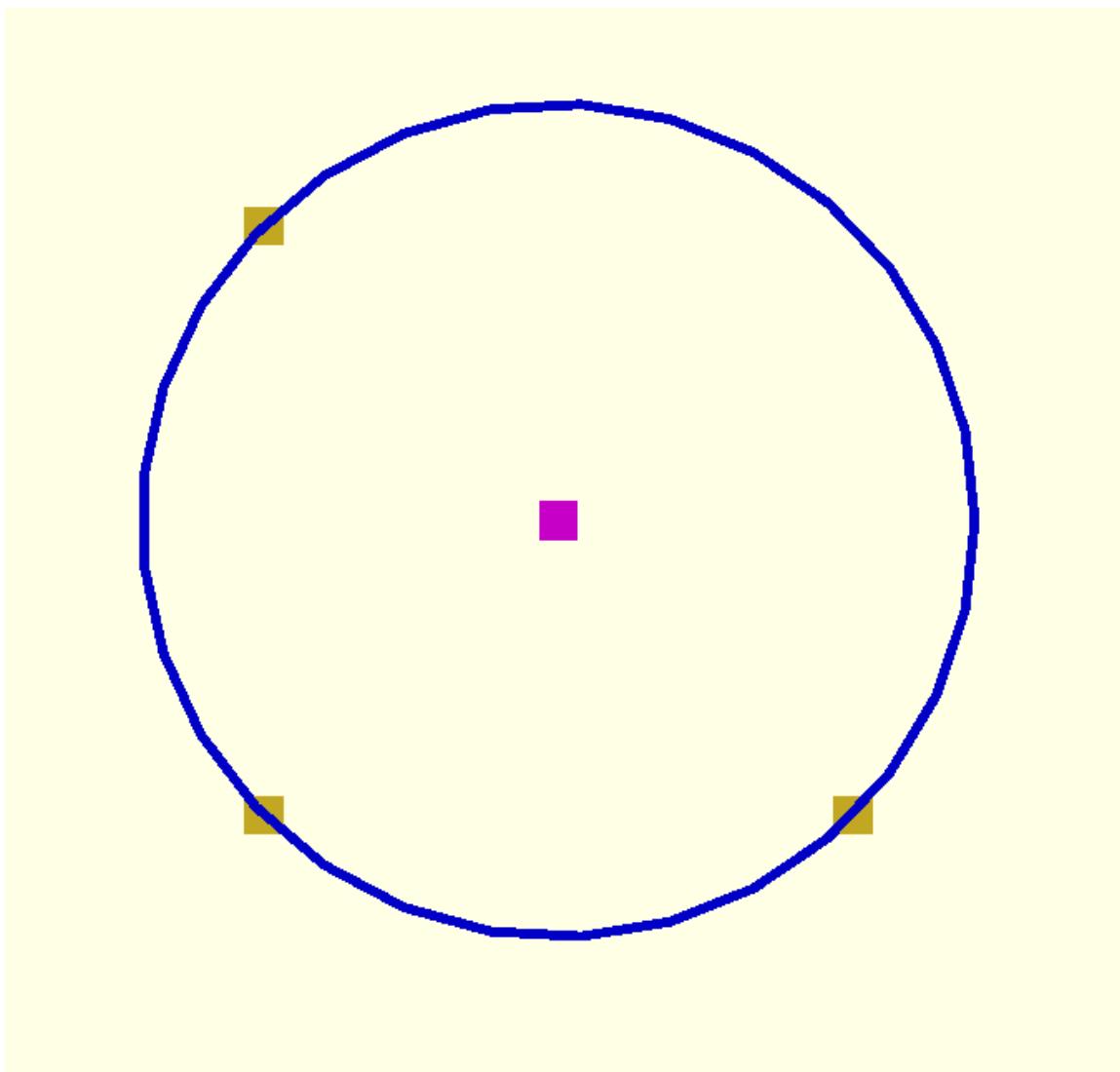


cp_3p

```
In [63]: # example of function cp_3p(p1,p2,p3)

p1,p2,p3=[3,0],[0,0],[0,3]
cir=cir_3p(p1,p2,p3,30)
center= cp_3p(p1,p2,p3)
with open('trial.scad','w+')as f:
    f.write(f'''
include<dependencies2.scad>

color("blue") p_line({cir},.05);
points({[p1,p2,p3]},.2);
color("magenta")points({[center]},.2);
    ''')
```



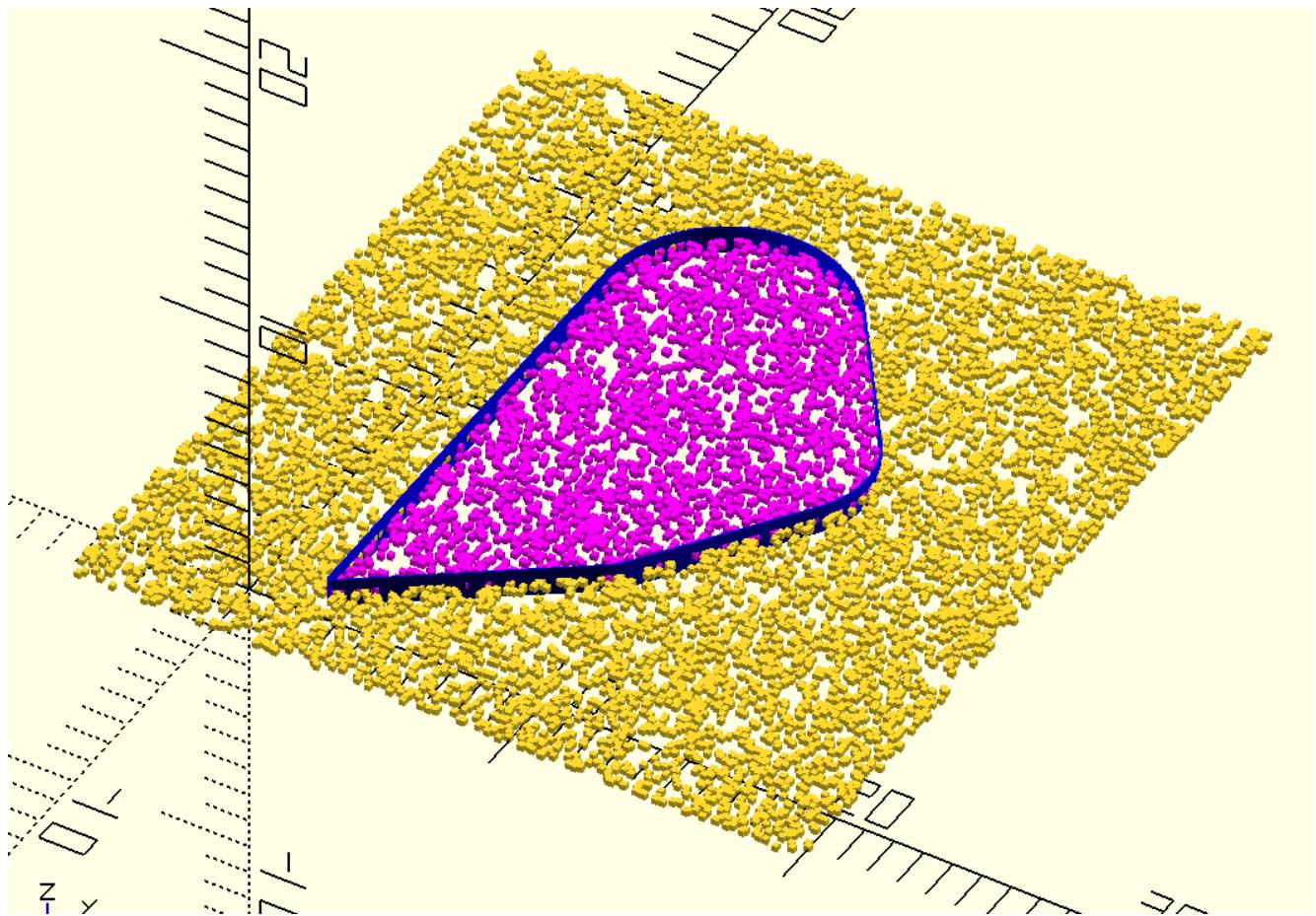
pies1

```
In [64]: # example of function pies1(sec,pnts)
# function points inside enclosed section
t0=time.time()
a=random.random(10000)*(20-(-5))+(-5)
b=random.random(10000)*(25-(-2))+(-2)
points=array([a,b]).transpose(1,0).tolist()
# sec=corner_radius(pts1([[2,1,.1],[7,5,2],[5,7,3],[-5,7,2],[-7,5,3]]),20)
sec=corner_radius(pts1([[8,0],[11,10,10],[0,10,5],[-10,0,5],[-1,-6,0.3],[-1,6,5],[-10,0,5],[0,-10,10]]))

with open('trial.scad','w+')as f:
    f.write(f'''
include<dependencies2.scad>

color("blue") p_line({sec},.05);
points({points},.2);
color("magenta")points({pies1(sec,points)},.2);
    ''')
t1=time.time()
t1-t0
```

```
Out[64]: 0.20709800720214844
```



```
In [65]: # example of pies1(sec,pnts)
t0=time.time()

a=random.random(10000)*(10-0)+0
b=random.random(10000)*(20-0)+0
c=array([a,b]).transpose(1,0).tolist()

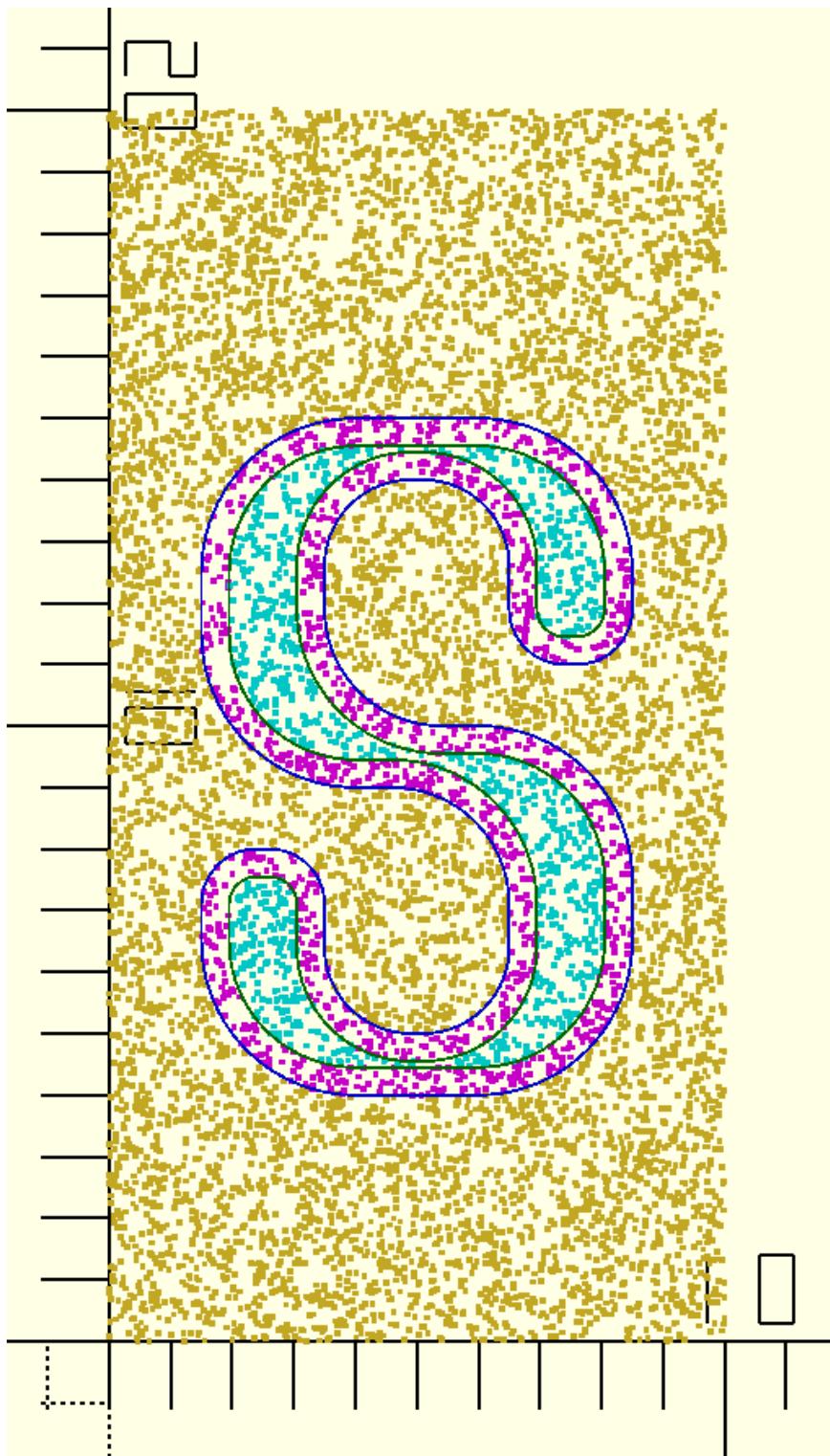
sec=corner_radius(pts1([[1.5,8,.9],[2,0,.9],[0,-3,1.4],[3,0,1.4],[0,4,1.9],[-5,0,2.5],[0,6,2.5],
[7,0,2.5],[0,-4,.9],[-2,0,.9],[0,3,1.4],[-3,0,1.4],[0,-4,1.9],
[5,0,2.5],[0,-6,2.5],[-7,0,2.5]]),20)

pnts=pies1(sec,c)
pnts1=pies1(offset(sec,-.45),pnts)

with open('trial.scad','w+')as f:
    f.write(f'''
    include<dependencies.scad>
    //points({c},.1);
    color("magenta")points({pnts},.1);
    color("cyan")points({pnts1},.1);
    color("blue")p_line({sec},.05);
    color("green")p_line({offset(sec,-.45)},.05);

    ''')
t1=time.time()
t1-t0
```

Out[65]: 0.4017791748046875



swp_prism_h

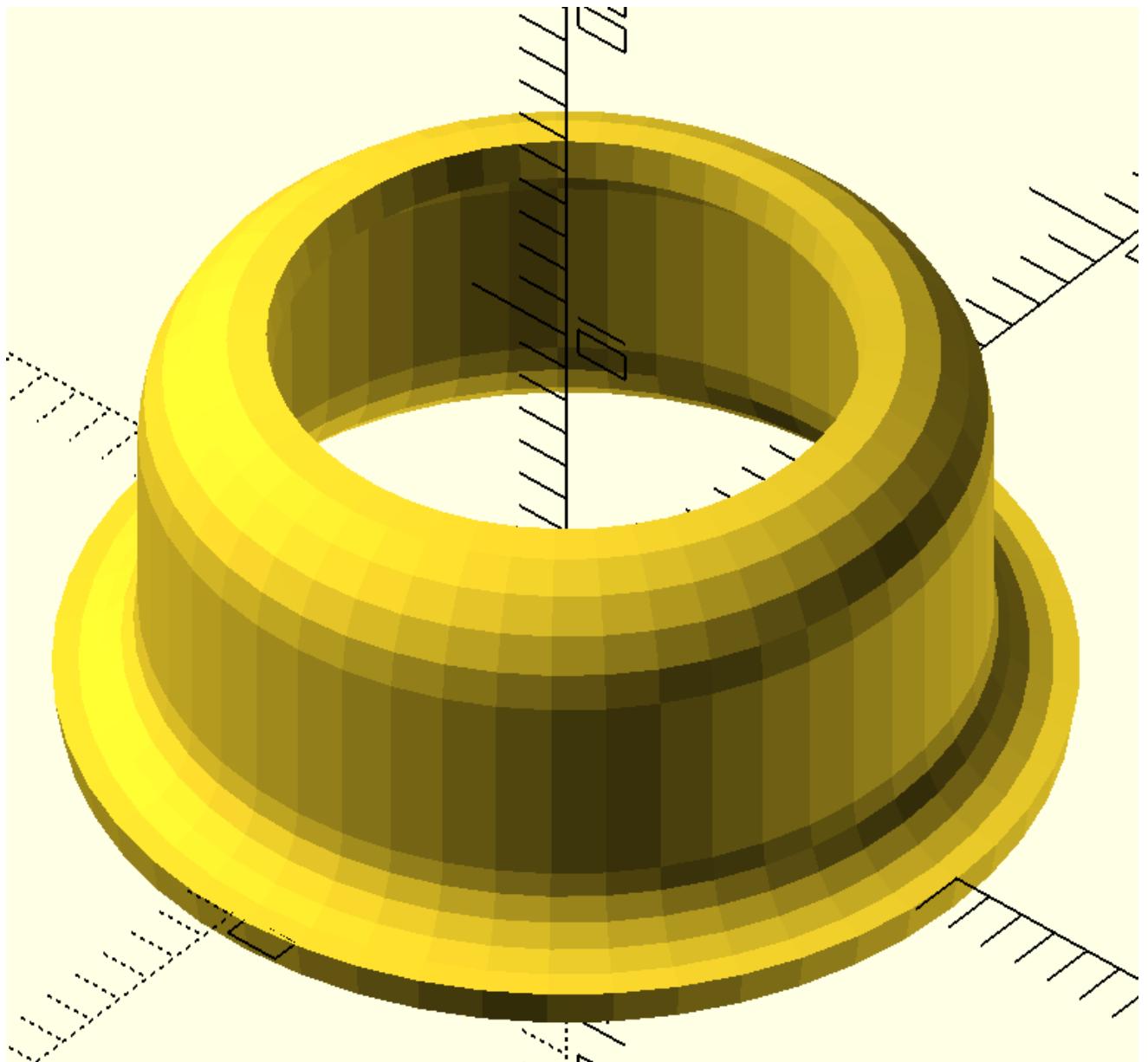
In [66]: # example of function swp_prism_h(prism_big,prism_small)

```
sec1=circle(10)
path=corner_radius(pts1([[2,0],[-2,0,2],[0,10,3],[-3,0]]),5)
sol=prism(sec1,path)
sol1=surf_offset(sol,-1)
sol2=swp_prism_h(sol,sol1)

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>

{swp_c(sol2)}

...)
```



```
In [67]: # example of function prism(sec,path) and function swp_prism_h(prism_big, prism_small)
t0=time.time()
# sec=corner_radius(pts1([[0,0,.2],[8,3,3],[5,7,1],[-8,0,2],[-5,20,1]]),20)
# sec=corner_radius(pts1([[0,0,1],[7,5,2],[5,7,3],[-5,7,5],[-7,5,5]]),20)
sec=corner_radius(pts1([[-15,0,2.4],[0,15,3],[30,0,3],[0,-15,2.4],[5,0,2.4],[0,20,7],[-40,0,7],[0,-20,
# sec=pts([[0,0],[10,0],[0,5],[-10,0]])
# path=corner_radius(pts1([[2,0],[-2,0,2],[0,7,4.2],[-4.2,0]]),20)
# path=corner_radius(pts1([[2,0],[-2,0,2],[0,7,5.2],[-5.2,0]]),20)
# path=corner_radius(pts1([[2,0],[-2,0,2],[0,7,2],[-2,0]]),20)

# sec=equidistant_pathc(sec,300)
sec1=offset(sec,.5)

sol=linear_extrude(sec,50)
sol1=linear_extrude(sec1,50)
sol2=swp_prism_h(sol1,sol)

with open('trial.scad','w+') as f:
    f.write(f'''  

include<dependencies2.scad>
{swp_c(sol2)}  

'''')
t1=time.time()
t1-t0
```

Out[67]: 0.048768043518066406

```
In [68]: sec=corner_radius(pts1([[0,0,1],[25,0,1],[0,15,1],[-25,0,1]]),10)
sec1=offset(sec,2)
sol=linear_extrude(sec,10)
```

```

sol1=linear_extrude(sec1,10)
sol2=swp_prism_h(sol1,sol)

with open('trial.scad','w+') as f:
    f.write(f'''

include<dependencies2.scad>
{swp_c(sol2)}

''')

```

In [69]:

```

sec=corner_radius(pts1([[0,0,1],[15,0,1],[0,10,1],[-15,0,1]]),10)
sec1=offset(sec,2)
sol=linear_extrude(sec,10)
sol1=linear_extrude(sec1,10)
sol2=swp_prism_h(sol1,sol)

with open('trial.scad','w+') as f:
    f.write(f'''

include<dependencies2.scad>
{swp_c(sol2)}

''')

```

surf_base

In [70]:

```

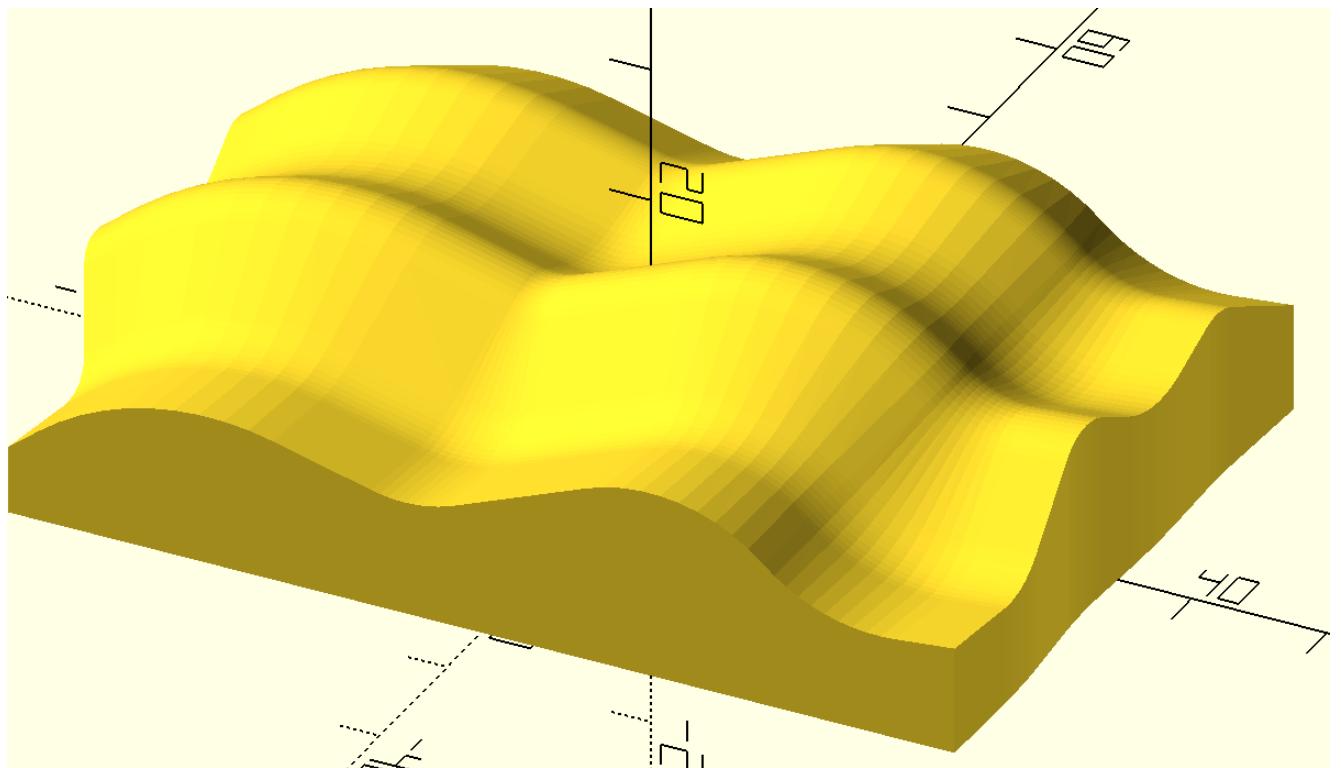
# example of function surf_base(surf,h)
t0=time.time()

sec2=corner_radius(pts1([[-25,0],[10,5,5],[10,-3,10],[10,5,5],[10,-8,7],[10,1]]),10)
path2=cytz(corner_radius(pts1([[-35,5,0],[10,8,20],[20,-5,10],[20,8,20],[10,-9,20],[10,1,0]]),10))
surf2=surf_extrude(sec2,path2)
sol=surf_base(surf2,0)
with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>
difference() {{
    {swp(sol)}
//{swp(cut_plane([0,0,1],[15,10],20,10))}

}}
''')
t1=time.time()
total=t1-t0
total

```

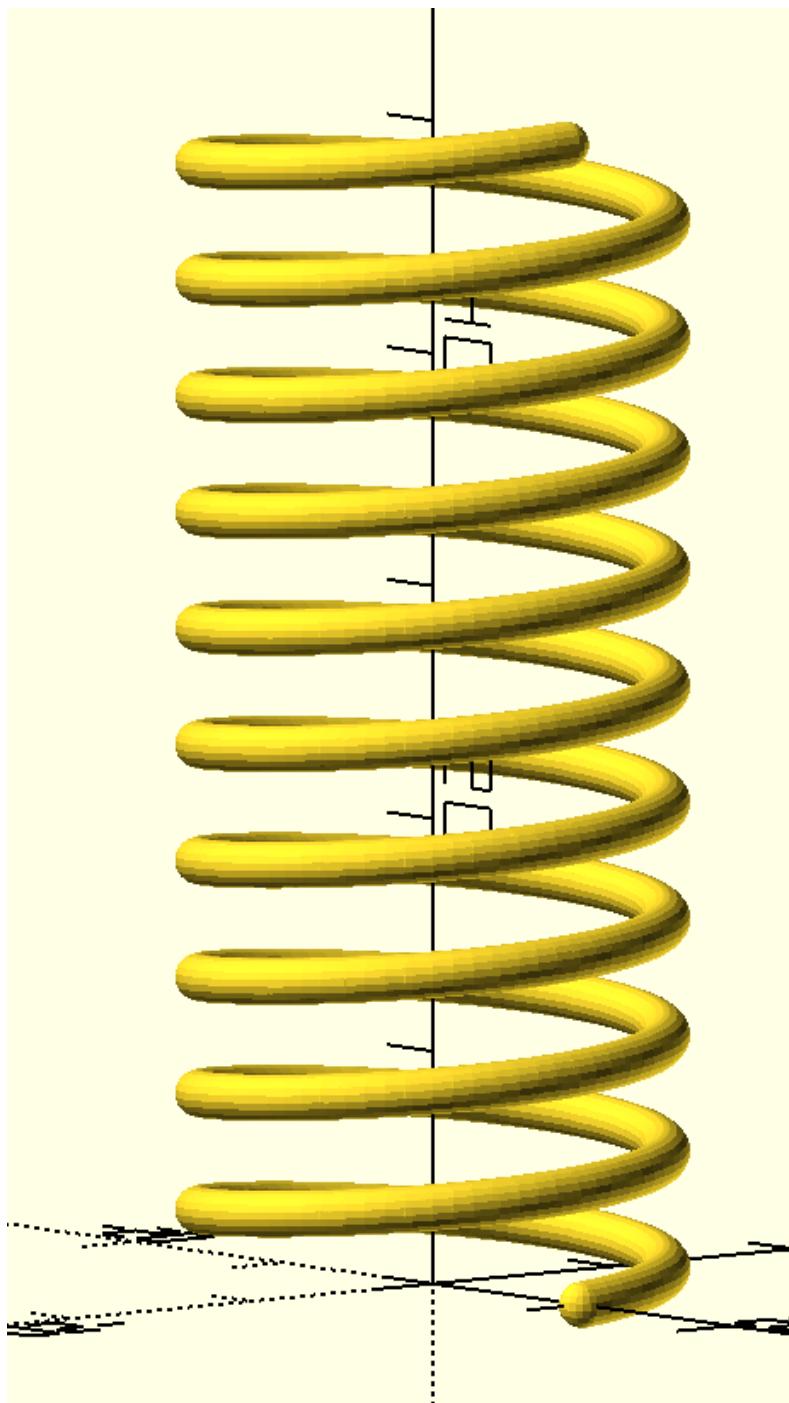
Out[70]: 0.0809319019317627



helix

```
In [71]: # example of function helix(radius=10,pitch=10, number_of_coils=1, step_angle=1)

path=helix(10,5,10,5)
sec=circle(1)
sol=path_extrude_open(sec,path)
with open('trial.scad','w+')as f:
    f.write(f'''
include<dependencies2.scad>
//p_line3d({{path}},1,$fn=20);
{swp(sol)}
    ''')
```



offset_points_cw

```
In [72]: # example of function offset_points_cw(sec,r)

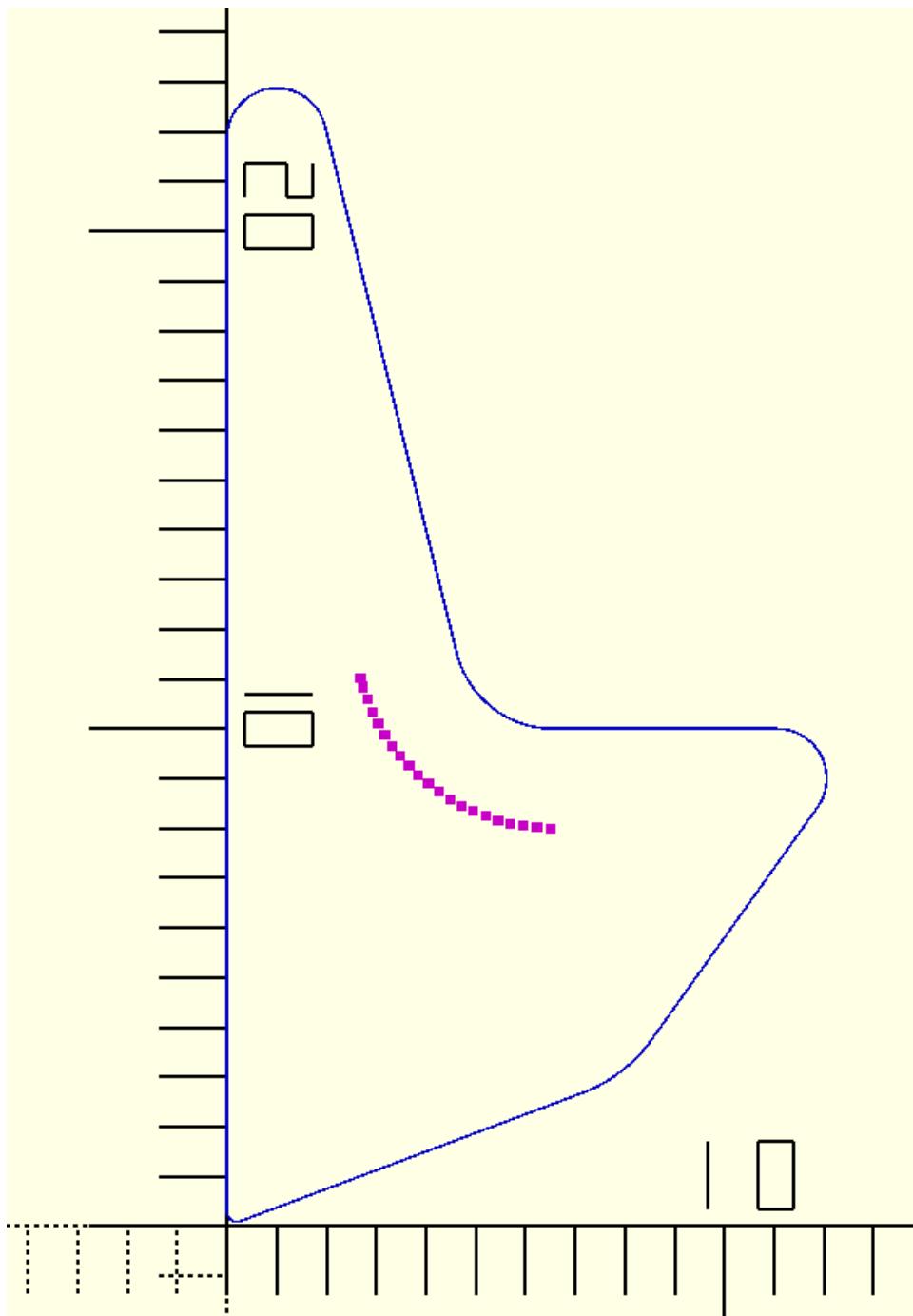
sec=corner_radius(pts1([[0,0,.2],[8,3,3],[5,7,1],[-8,0,2],[-5,20,1]]),20)
# sec=corner_radius(pts1([[0,0,.1],[7,5,2],[5,7,3],[-5,7,2],[-7,5,3]]),20)
# sec=corner_radius(pts1([[-15,0,2.4],[0,15,3],[30,0,3],[0,-15,2.4],[5,0,2.4],[0,20,7],[-40,0,7],[0,-2

# path=corner_radius(pts1([[2,0],[-2,0,2],[0,7,4.2],[-4.2,0]]),40)
# path=corner_radius(pts1([[2,0],[-2,0,2],[0,7,5.3],[-5.3,0]]),40)
# path=corner_radius(pts1([[2,0],[-2,0,2],[0,7,2],[-2,0]]),20)

offsetPoints=offset_points_cw(sec,-2)

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>
color("blue")p_line({sec},.05);
color("magenta")points({offsetPoints},.2);

'''')
```



offset_points_ccw

```
In [68]: # example of function offset_points_ccw(sec,r)

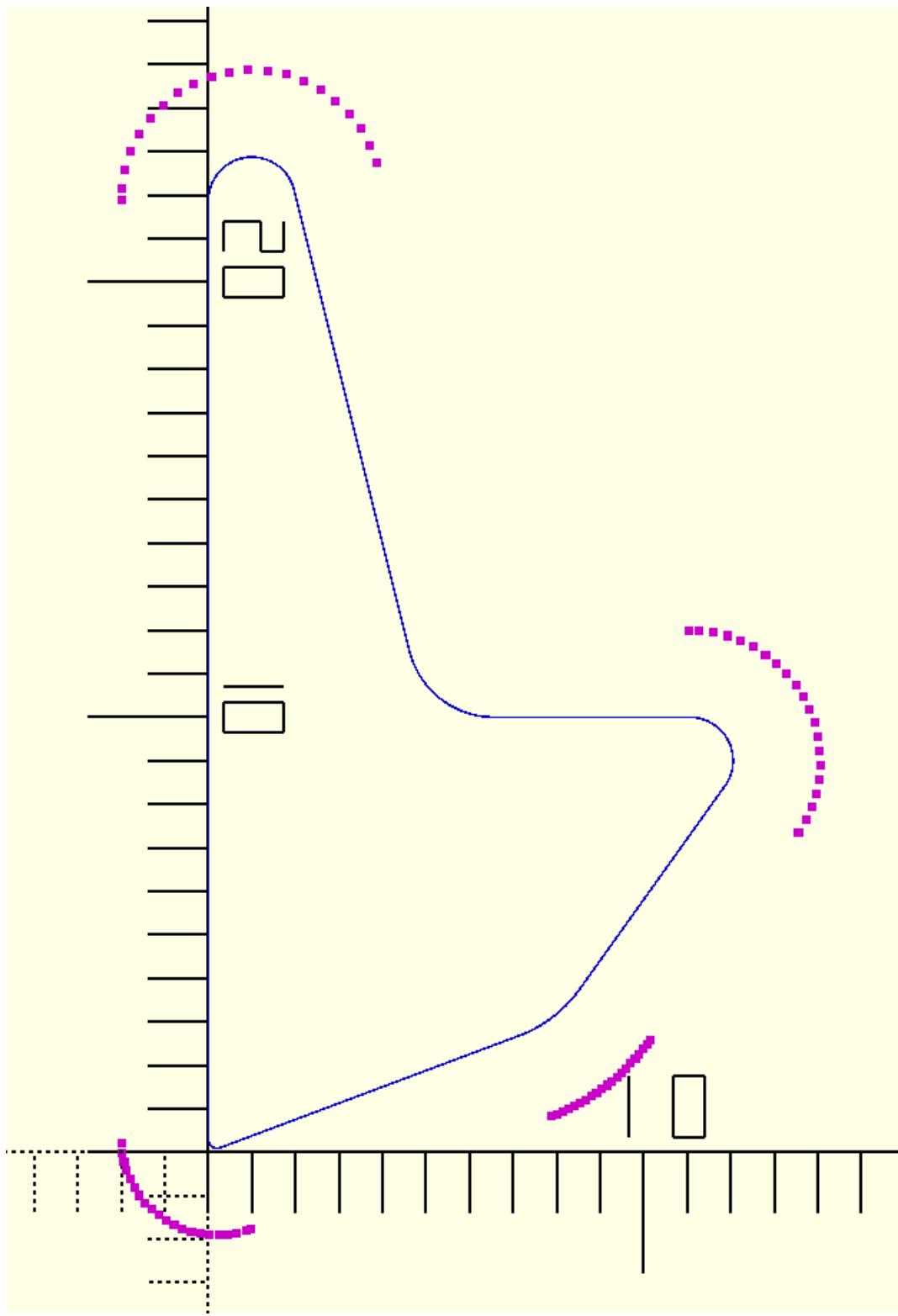
sec=corner_radius(pts1([[0,0,.2],[8,3,3],[5,7,1],[-8,0,2],[-5,20,1]]),20)
# sec=corner_radius(pts1([[0,0,.1],[7,5,2],[5,7,3],[-5,7,5],[-7,5,5]]),20)
# sec=corner_radius(pts1([-15,0,2.4],[0,15,3],[30,0,3],[0,-15,2.4],[5,0,2.4],[0,20,7],[-40,0,7],[0,-2

# path=corner_radius(pts1([[2,0],[-2,0,2],[0,7,4.2],[-4.2,0]]),40)
# path=corner_radius(pts1([[2,0],[-2,0,2],[0,7,5.3],[-5.3,0]]),40)
# path=corner_radius(pts1([[2,0],[-2,0,2],[0,7,2],[-2,0]]),20)

offsetPoints=offset_points_ccw(sec,2)

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>
color("blue")p_line({sec},.05);
color("magenta")points({offsetPoints},.2);

'''')
```



cpo

```
In [73]: # example of function cpo(prism)

sec=corner_radius(pts1([[0,0,.2],[8,3,3],[5,7,1],[-8,0,2],[-5,20,1]]),10)
# sec=corner_radius(pts1([[0,0,.1],[7,5,2],[5,7,3],[-5,7,5],[-7,5,5]]),20)
# sec=corner_radius(pts1([[-15,0,2.4],[0,15,3],[30,0,3],[0,-15,2.4],[5,0,2.4],[0,20,7],[-40,0,7],[0,-2

path=corner_radius(pts1([[2,0],[-2,0,2],[0,7,4.2],[-4.2,0]]),10)
# path=corner_radius(pts1([[2,0],[-2,0,2],[0,7,5.3],[-5.3,0]]),40)
# path=corner_radius(pts1([[2,0],[-2,0,2],[0,7,2],[-2,0]]),20)

sp1=translate([0,50,0],sphere(10,s=20))
sp2=translate([30,0,0],cpo(sp1))

sol=prism(sec,path)
sol1=translate([20,0,0],cpo(sol))
```

```

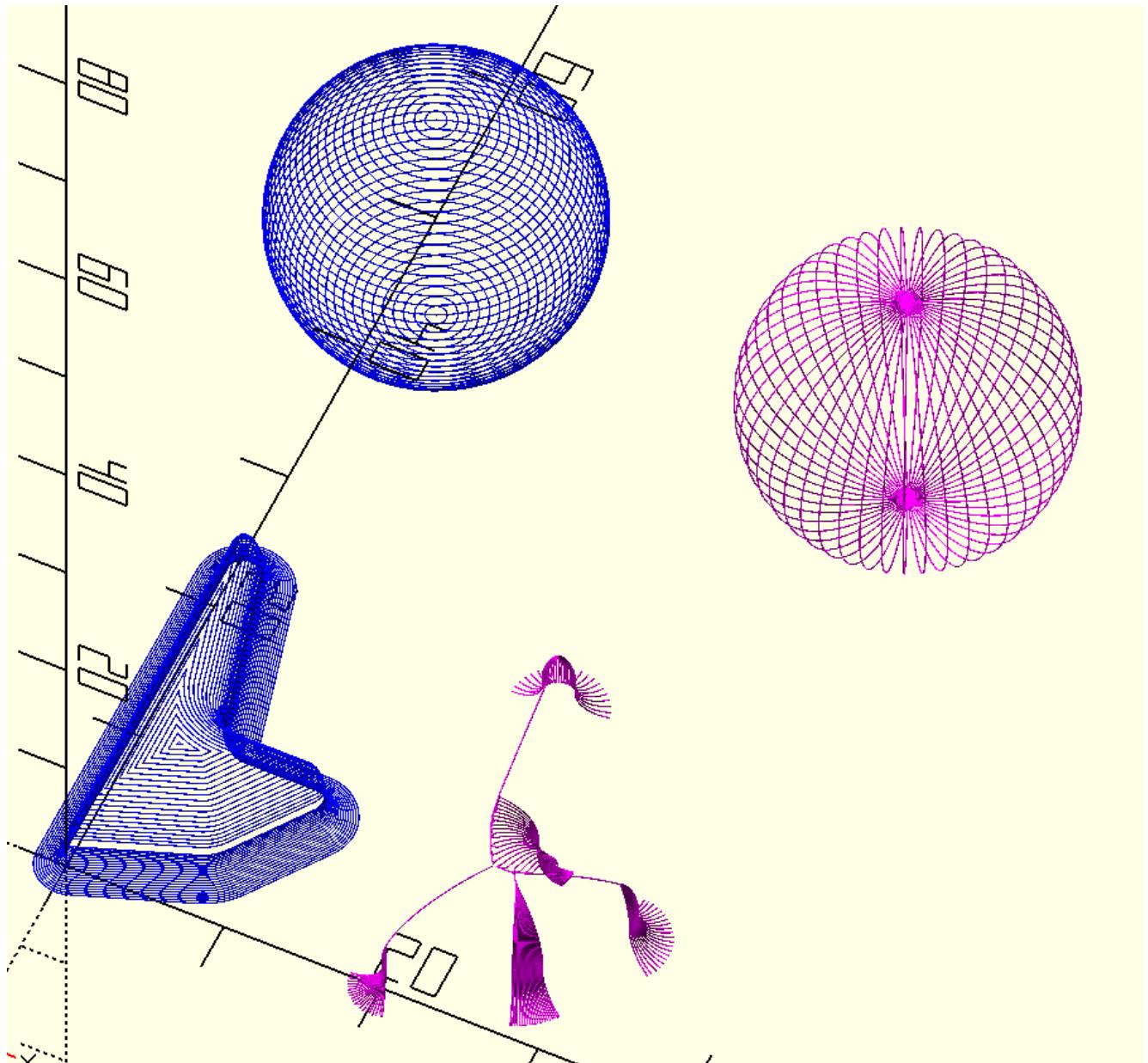
with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>

color("blue")for(p={sol})p_line3dc(p,.1,1);
color("magenta")for(p={sol1})p_line3d(p,.1,1);

color("blue")for(p={sp1})p_line3dc(p,.1,1);
color("magenta")for(p={sp2})p_line3d(p,.1,1);

...''')

```



fillet_3p_3d

```

In [74]: # example of function fillet_3p_3d(p0,p1,p2,r,s)

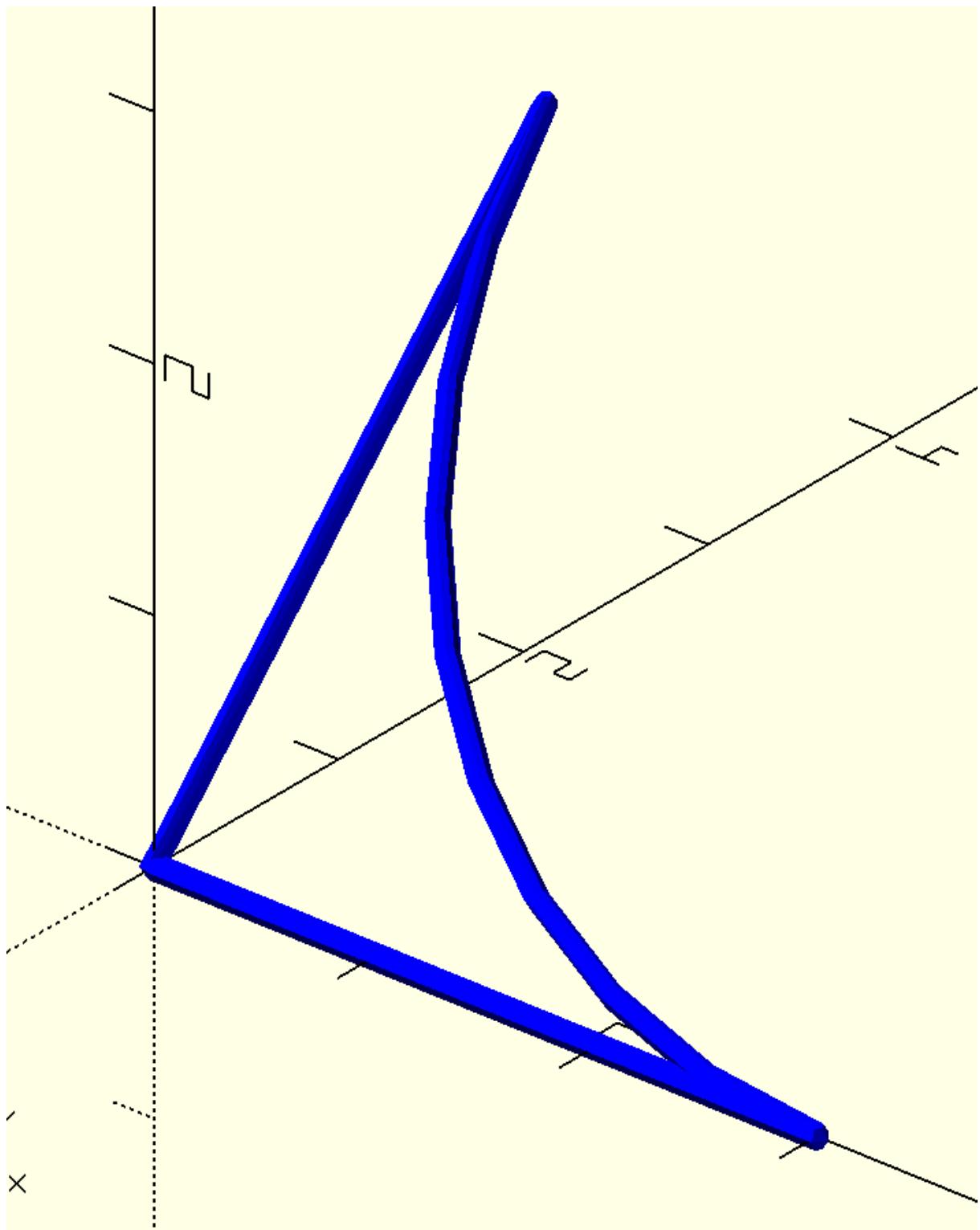
p1,p2,p3=[[3,0,0],[0,0,0],[0,3,3]]
fillet=fillet_3p_3d(p1,p2,p3,3,10)

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>

color("blue")p_line3dc({fillet},.05);
color("cyan")points({[p1,p2,p3]},.2);

...''')

```



fillet_3p_3d_cp

```
In [75]: # example of function fillet_3p_3d_cp(p0,p1,p2,r)

p1,p2,p3=[[3,0,0],[0,0,0],[0,3,3]]
fillet=fillet_3p_3d(p1,p2,p3,3,10)
centerPoint=fillet_3p_3d_cp(p1,p2,p3,3)

with open('trial.scad','w+') as f:
    f.write(f''''
include<dependencies2.scad>

color("blue")p_line3dc({fillet},.05);
color("magenta")points({[centerPoint]}),.2;

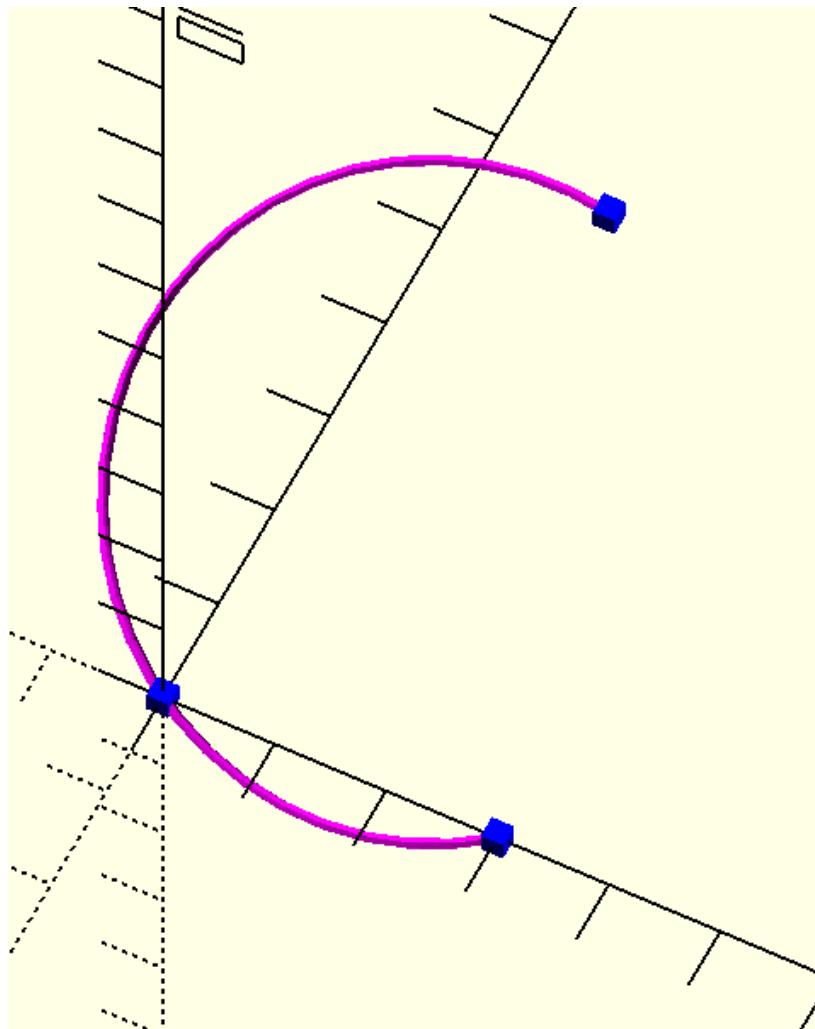
...)
```

arc_3p_3d

In [76]: # example of function arc_3p_3d(points,s)

```
p1,p2,p3=[[3,0,0],[0,1,0],[5,3,2]]  
arc1=arc_3p_3d([p1,p2,p3],30)
```

```
with open('trial.scad','w+') as f:  
    f.write(f'''  
include<dependencies2.scad>  
  
color("magenta")p_line3d({arc1},.05);  
color("blue")points({[p1,p2,p3]},.2);  
  
'''')
```



cir_3p_3d

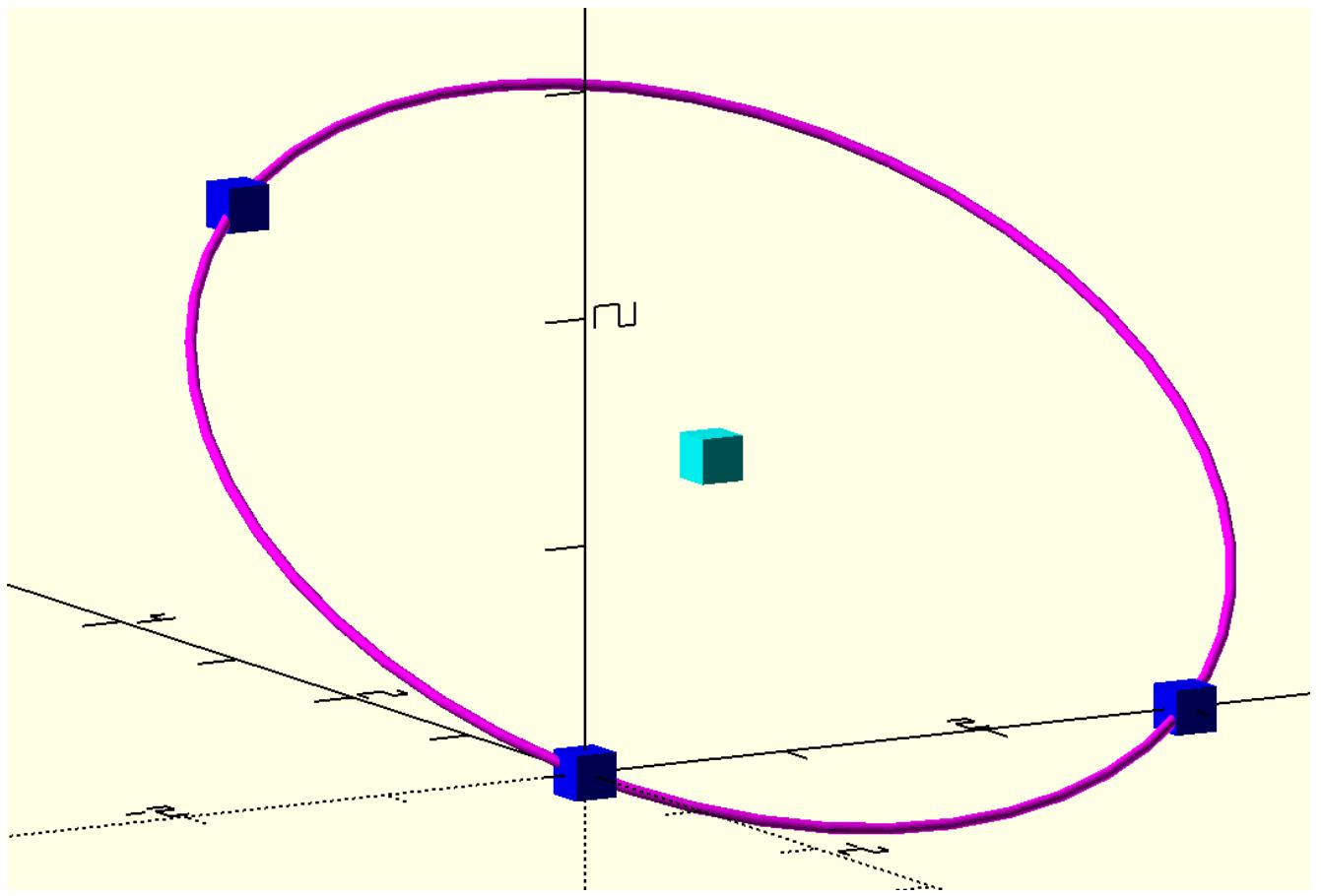
cp_cir_3d

In [77]: # example of function cir_3p_3d(points,s)

```
p1,p2,p3=[[3,0,0],[0,0,0],[0,3,2]]  
cir=cir_3p_3d([p1,p2,p3],50)  
  
cp1=cp_cir_3d(cir)  
  
with open('trial.scad','w+') as f:  
    f.write(f'''  
include<dependencies2.scad>
```

```
color("magenta")p_line3dc({cir},.05);
color("blue")points({{p1,p2,p3}},.2);
color("cyan")points({{cp1}},.2);

...)
```



linear_extrude

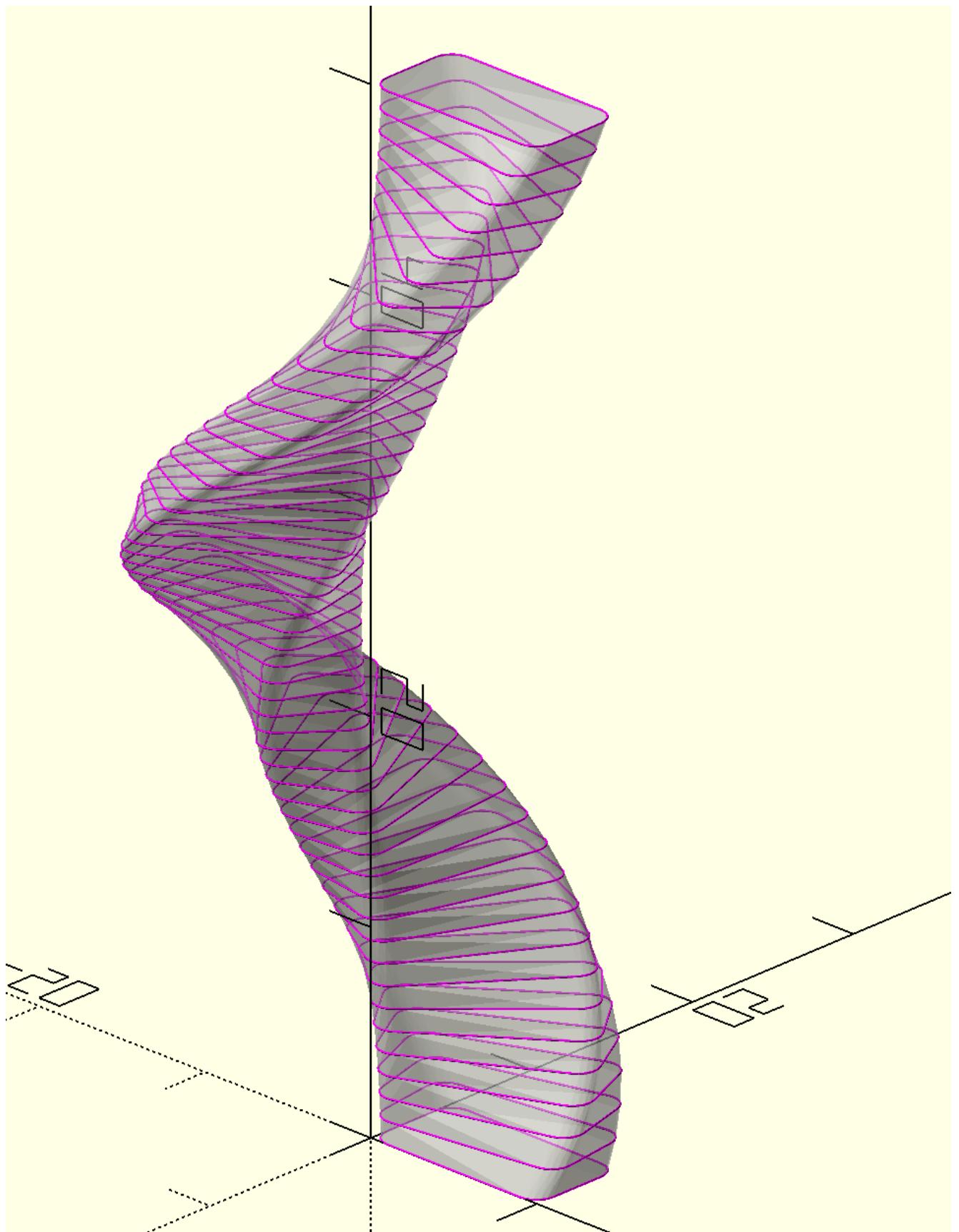
```
In [78]: # example of function linear_extrude(sec,h=1,a=0,steps=1)

sec=corner_radius(pts1([[0,0,1],[10,0,1],[0,5,1],[-10,0,1]]),5)
sol=linear_extrude(sec,50,360,50)

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>

//color("magenta") for(p={sol})p_line3dc(p,.05);
{swp(sol)}

...)
```



rsz3d

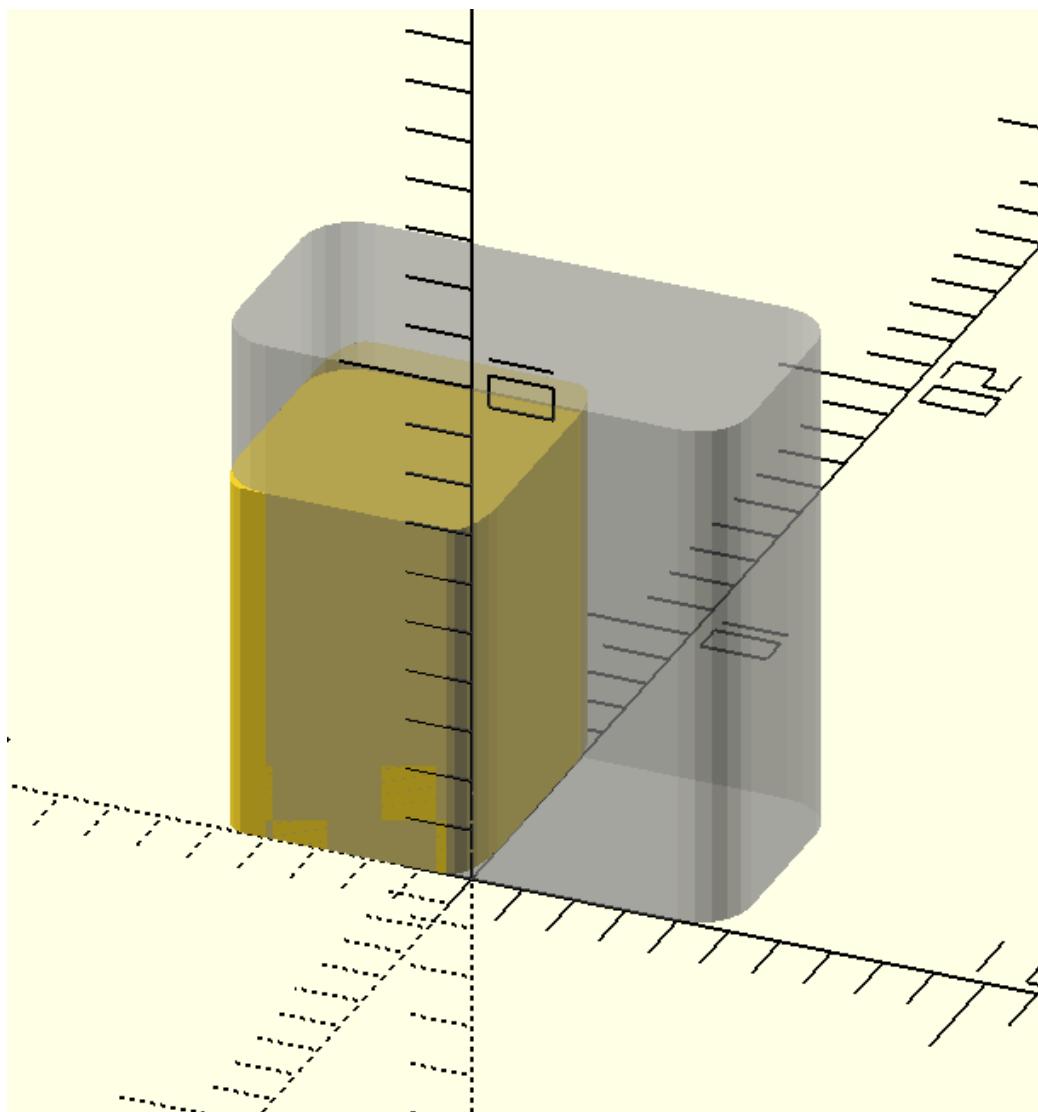
In [79]: `# example of function rsz3d(prism,rsz)`

```
sec=corner_radius(pts1([[0,0,1],[10,0,1],[0,5,1],[-10,0,1]]),5)
sol=translate([-5,0,0],linear_extrude(sec,10))
sol1=rsz3d(sol,[5,6,7])

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>

{swp(sol)}
{swp(sol1)}
```

...)



rsz3dc

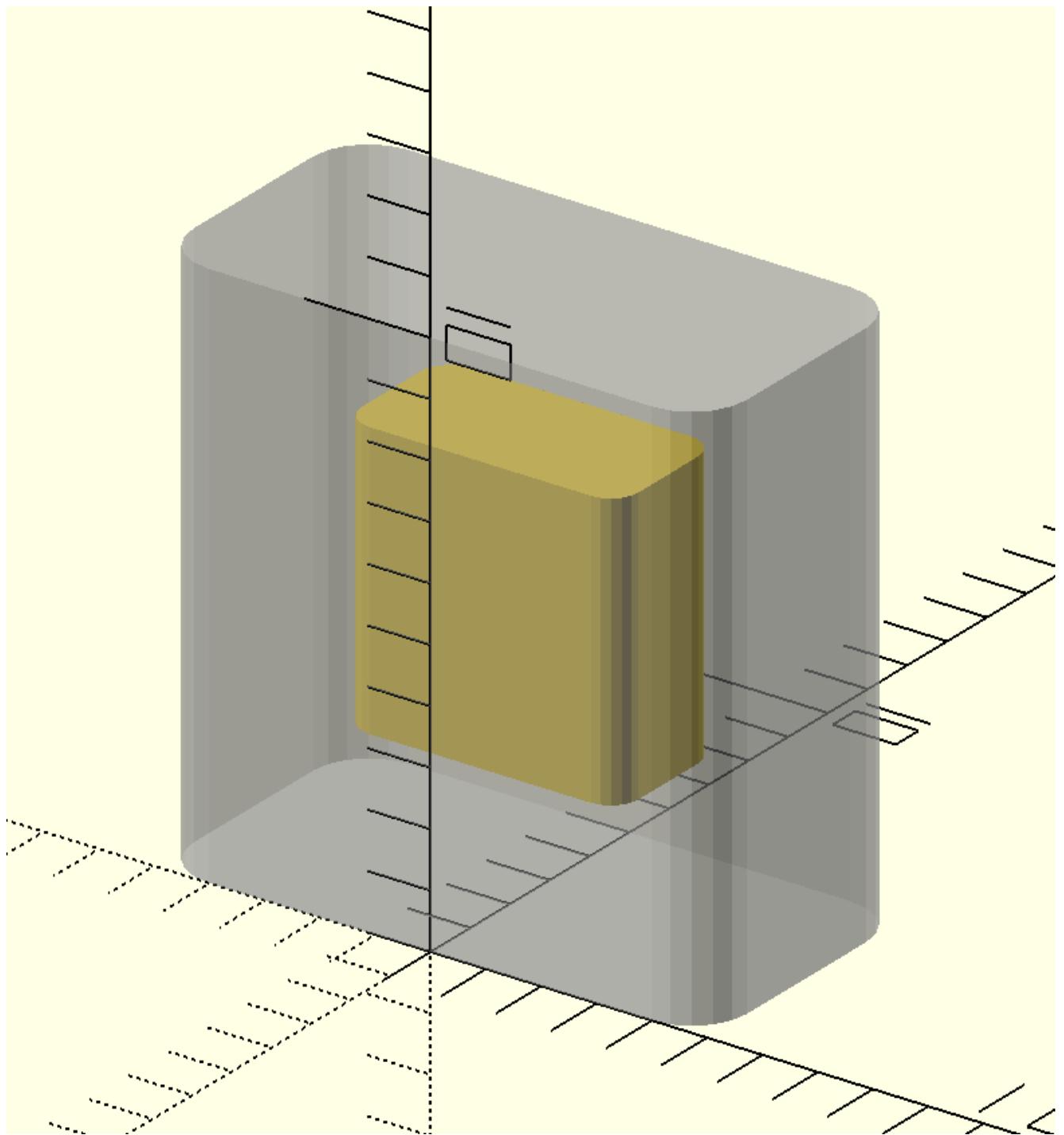
In [80]:

```
# example of function rsz3dc(prism,rsz)

sec=corner_radius(pts1([[0,0,1],[10,0,1],[0,5,1],[-10,0,1]]),5)
sol=translate([-5,0,0],linear_extrude(sec,10))
sol1=rsz3dc(sol,[5,2.5,5])

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>

%{swp(sol)}
{swp(sol1)}
...)
```



bb

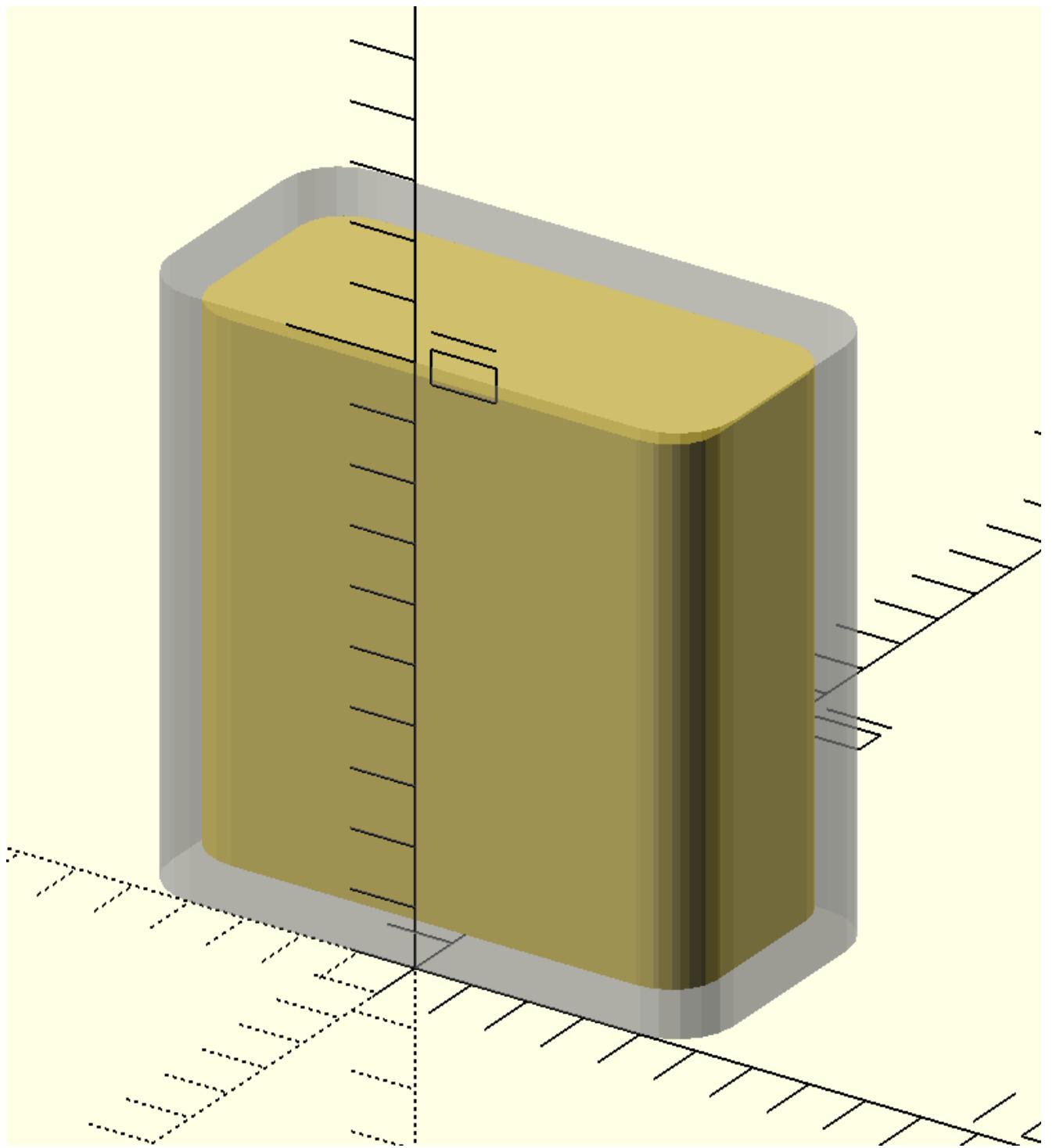
In [81]: `# example of function bb(prism)`

```
sec=corner_radius(pts1([[0,0,1],[10,0,1],[0,5,1],[-10,0,1]]),5)
sol=translate([-5,0,0],linear_extrude(sec,10))
dim=bb(sol)
sol1=rsz3dc(sol,array(dim)-[2,2,2])

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>

%{swp(sol)}
{swp(sol1)}

'''')
```



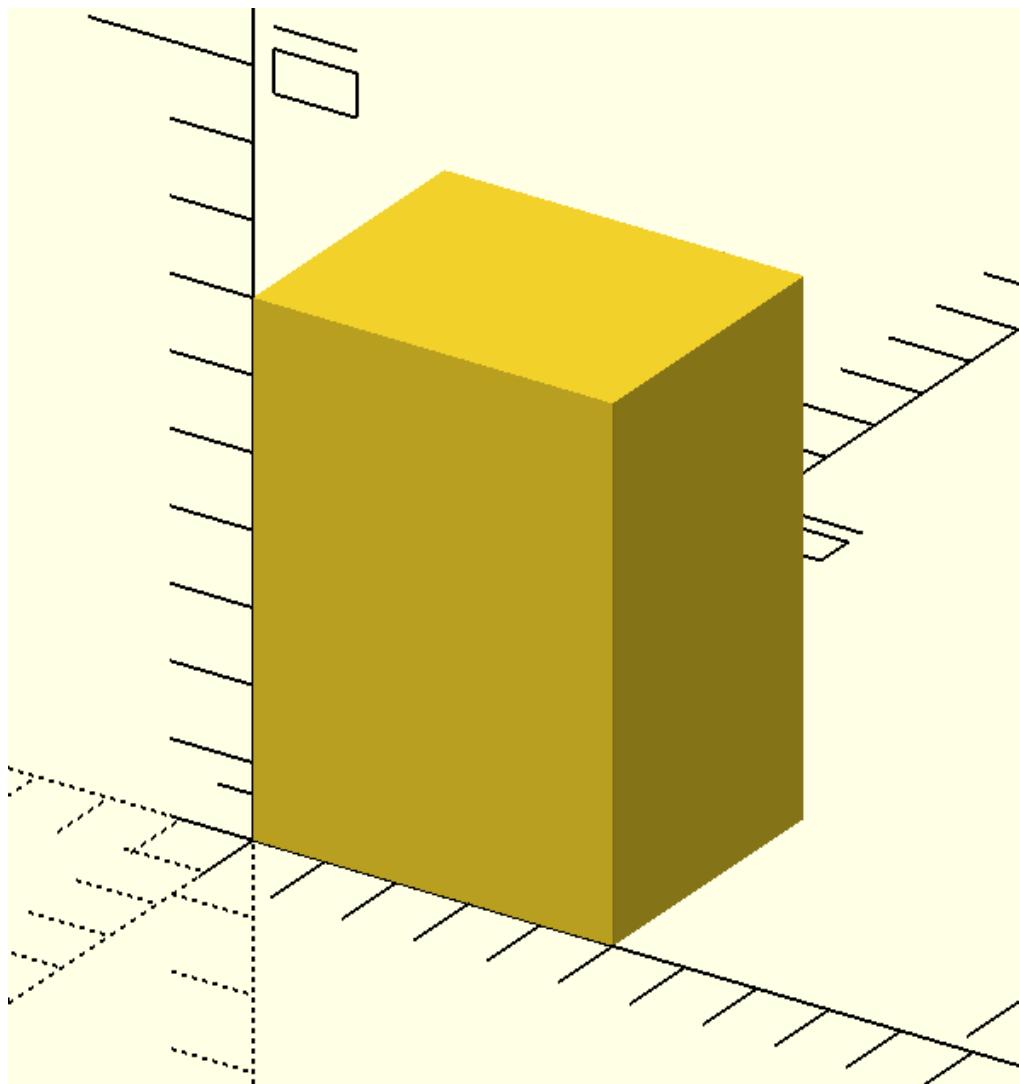
cube

```
In [82]: # example of function cube(s,center=False)

sol=cube([5,4,7],False)

with open('trial.scad','w+') as f:
    f.write(f'''include<dependencies2.scad>

{swp(sol)}
...''')
```

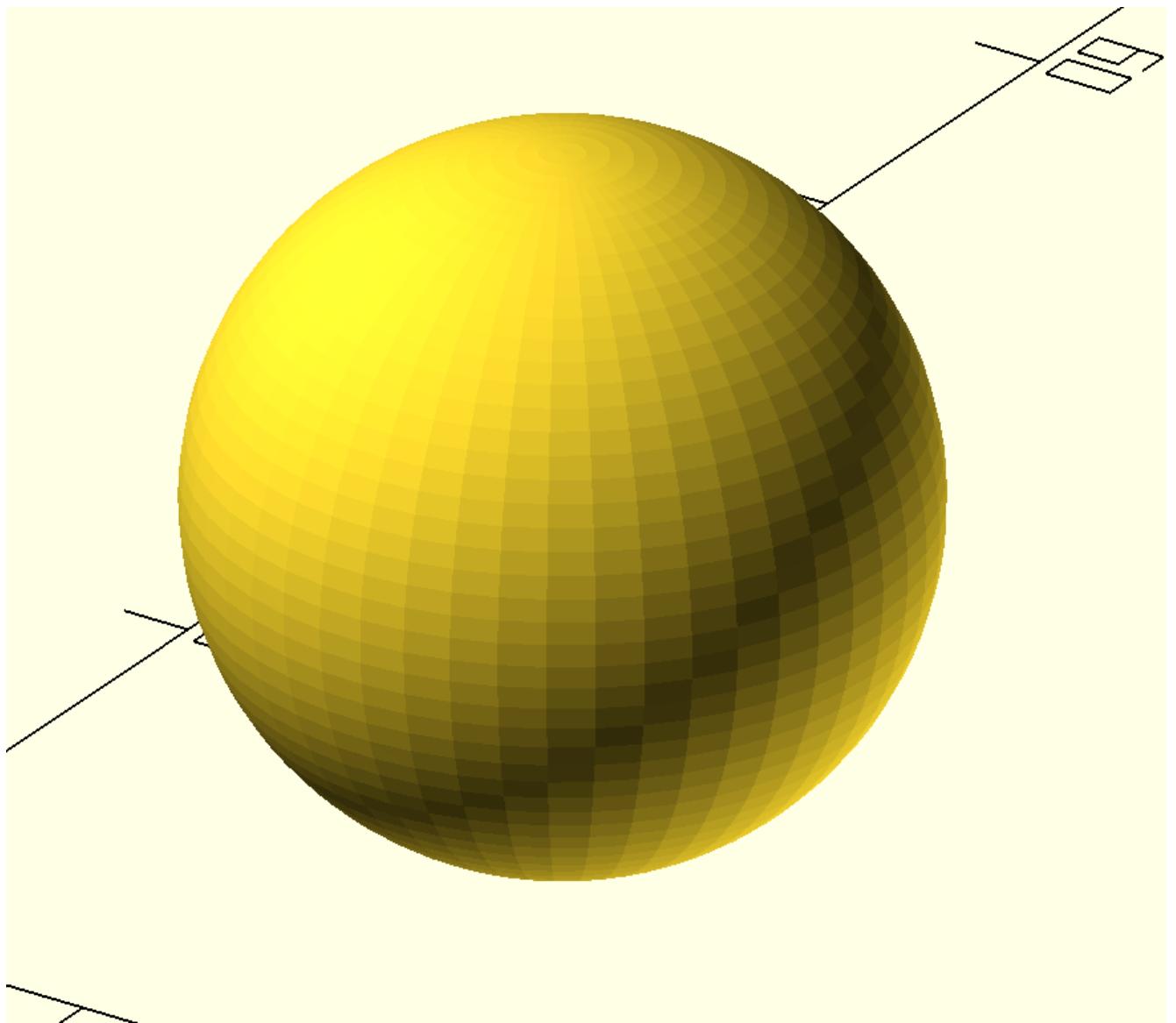


sphere

```
In [83]: # example of function sphere(r=0,c=[0,0,0],s=50)

sol=sphere(10,[15,15,10],30)

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>
difference() {{
    {swp(sol)}
    //cube(10);
}}
'''')
```



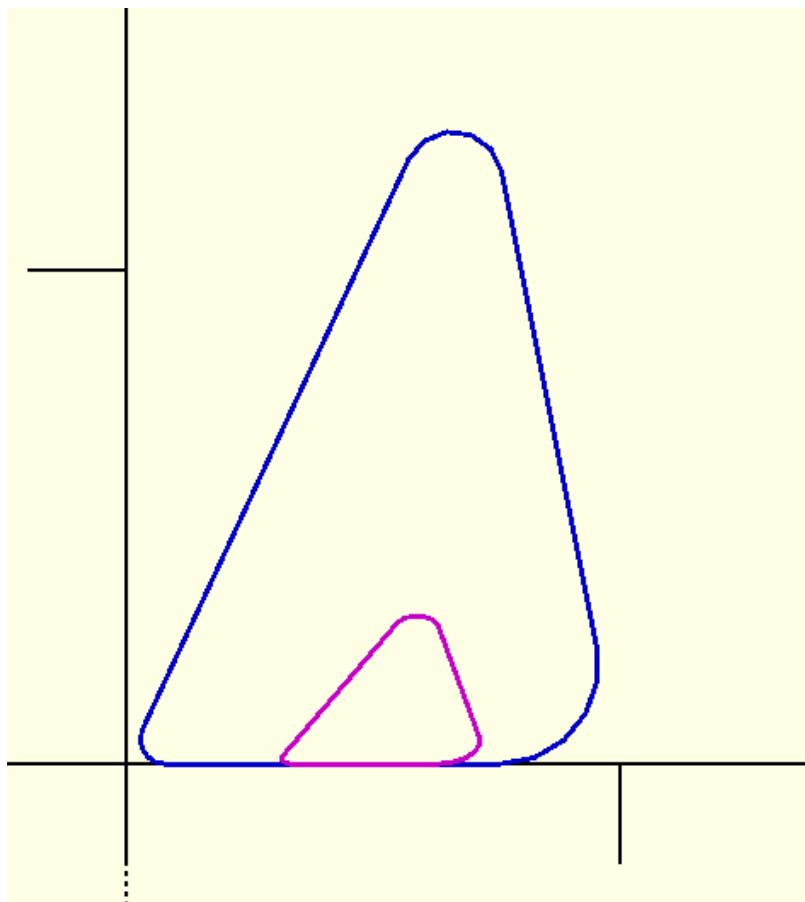
rsz2d

In [84]:

```
# example of function rsz2d(sec,rsz)

sec=corner_radius([[0,0,.5],[10,0,2],[7,15,1]],5)
sec1=rsz2d(sec,[4,3])
with open('trial.scad','w+') as f:
    f.write(f'''  
include<dependencies2.scad>
color("blue")p_line({sec},.1);
color("magenta")p_line({sec1},.1);

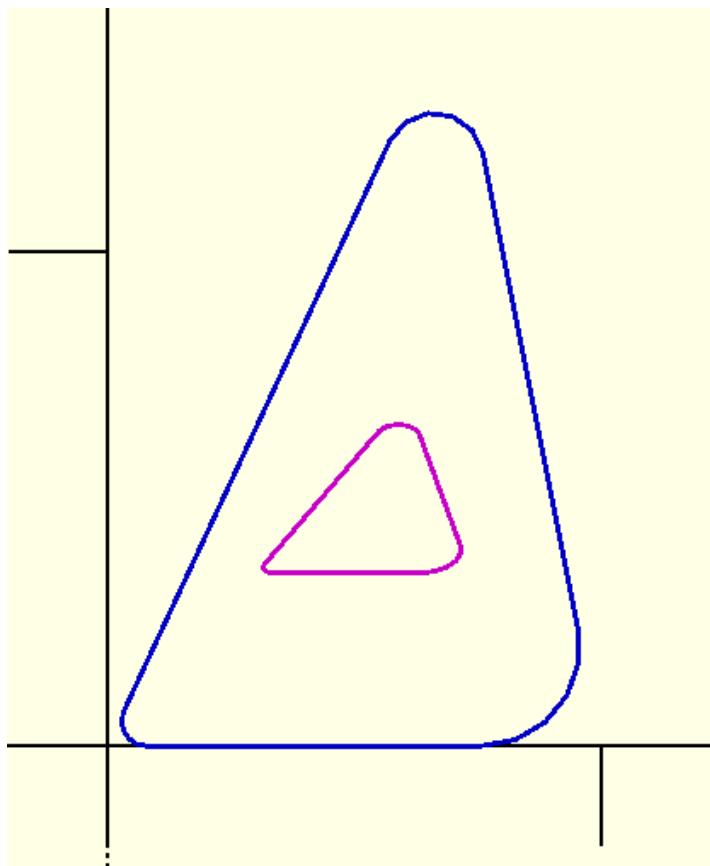
''')
```



rsz2dc

```
In [85]: # example of function rsz2dc(sec,rsz)
sec=corner_radius([[0,0,.5],[10,0,2],[7,15,1]],5)
sec1=rsz2dc(sec,[4,3])
with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>
color("blue")p_line({sec},.1);
color("magenta")p_line({sec1},.1);

...)
```

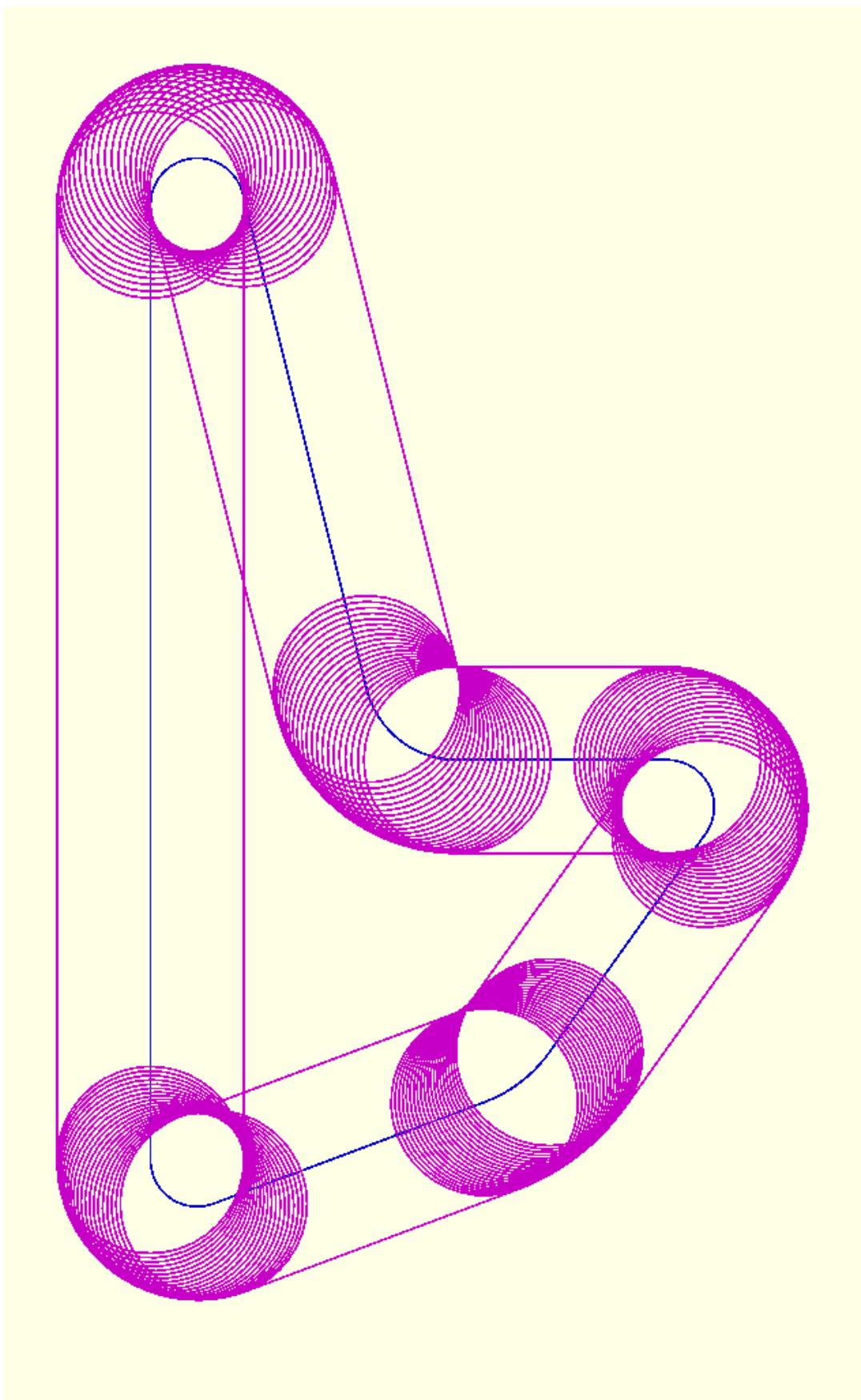


cs1

```
In [86]: # example of function cs1(sec,d) cs1 meaning cleaning section. This function is used in offset function
sec=corner_radius(pts1([[0,0,1],[8,3,3],[5,7,1],[-8,0,2],[-5,20,1]]),10)
# sec=corner_radius(pts1([[0,0,.1],[7,5,2],[5,7,3],[-5,7,5],[-7,5,5]]),20)
# sec=corner_radius(pts1([[-15,0,2.49],[0,15,3],[30,0,3],[0,-15,2.49],[5,0,2.49],[0,20,7],[-40,0,7],[0,0,1]]),10)

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>
color("blue")p_line({{sec}},.05);
color("magenta",.2)for(p={{cs1(sec,-2)}})p_line(p,.05);

...''')
```



convert_secv1

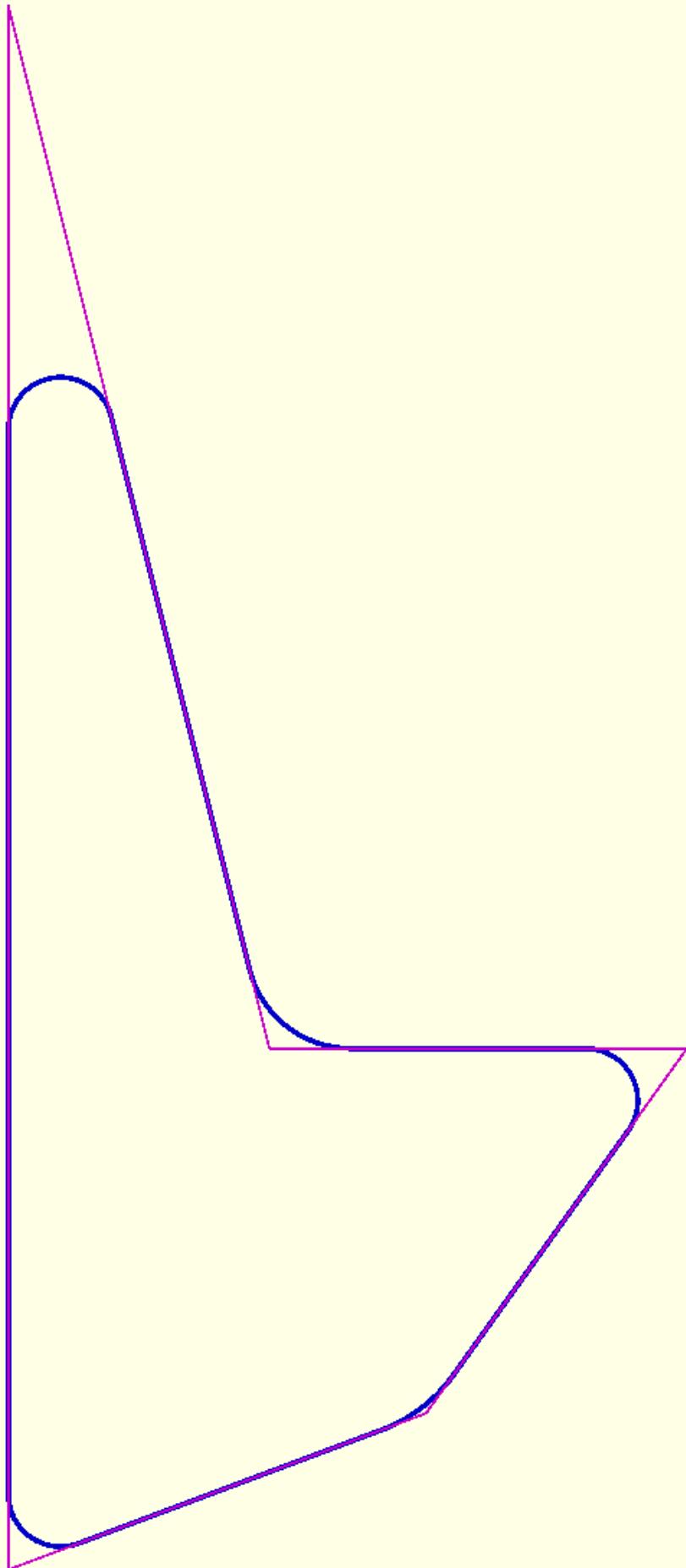
In [84]:

```
# example of function convert_secv1(sec,d)

sec=corner_radius(pts1([[0,0,1],[8,3,3],[5,7,1],[-8,0,2],[-5,20,1]]),20)
# sec=sec=corner_radius(pts1([0,0,.1],[7,5,2],[5,7,3],[-5,7,2],[-7,5,3])),20
# sec=corner_radius(pts1([-15,0,2.49],[0,15,3],[30,0,3],[0,-15,2.49],[5,0,2.49],[0,20,7],[-40,0,7],[0
# sec=circle(10)
# sec=pts([[0,0],[10,0],[0,10],[-10,0]])

sec1=convert_secv1(sec)
```

```
with open('trial.scad','w+') as f:  
    f.write(f'''  
include<dependencies2.scad>  
color("blue")p_line({sec},.1);  
  
color("magenta")p_line({sec1},.05);  
...)
```



convert_secv

In [85]: # example of function convert_secv(sec)

```
sec=corner_radius(pts1([[0,0,1],[8,3,3],[5,7,1],[-8,0,2],[-5,20,1]]),40)
# sec=corner_radius(pts1([[0,0,.1],[7,5,2],[5,7,3],[-5,7,2],[-7,5,3]]),20)
# sec=corner_radius(pts1([-15,0,2.49],[0,15,3],[30,0,3],[0,-15,2.49],[5,0,2.49],[0,20,7],[-40,0,7],[0,0,1]),40)
# sec=circle(10)
# sec=pts([[0,0],[10,0],[0,10],[-10,0]])
# path=corner_radius(pts1([[2,0],[-2,0,2],[0,7,4.2],[-4.2,0]]),40)
# path=corner_radius(pts1([[2,0],[-2,0,2.2],[0,7,5.3],[-5.3,0]]),40)
# path=corner_radius(pts1([[2,0],[-2,0,2],[0,7,2],[-2,0]]),20)

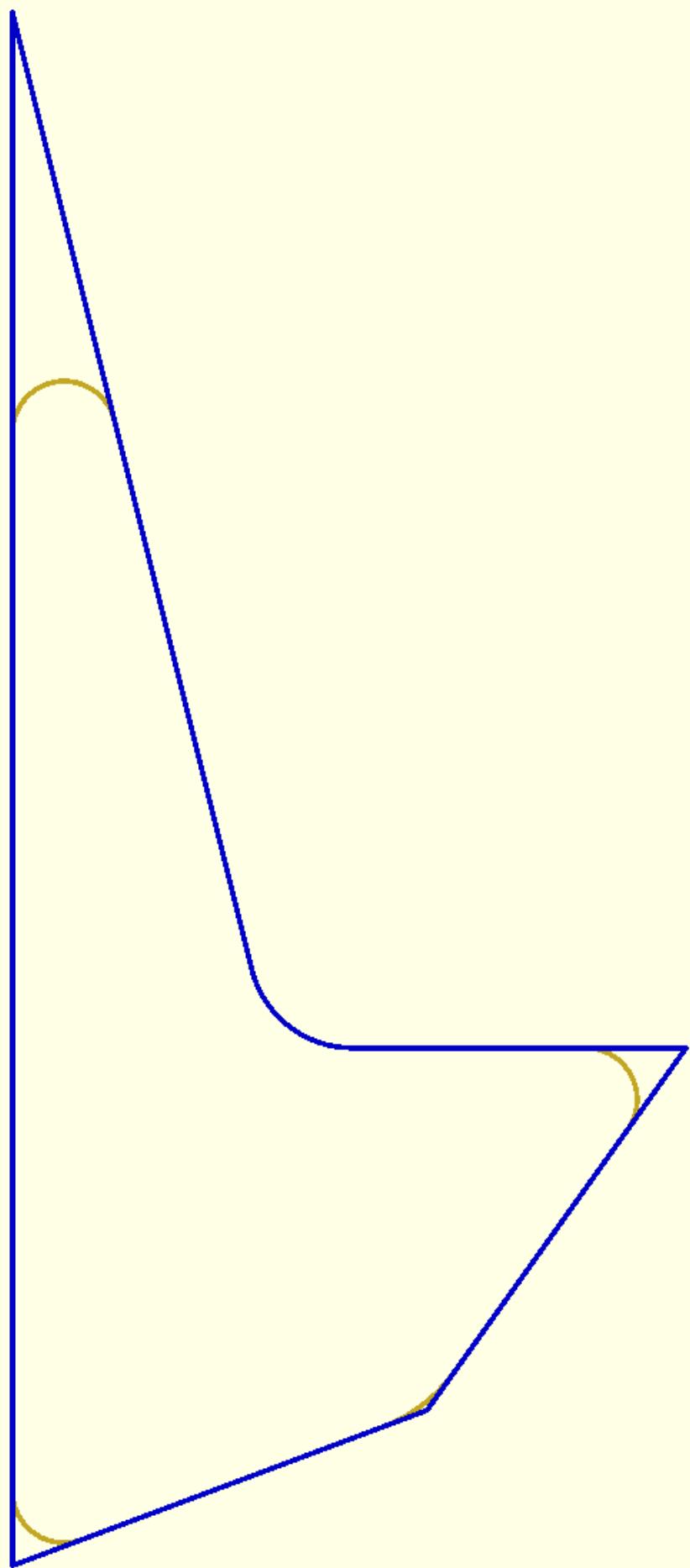
with open('trial.scad','w+') as f:
    f.write(f'''  

    include<dependencies2.scad>  

    p_line({sec},.1);
    color("blue")p_line({convert_secv(sec)},.1);  

    ''')
```



convert_secv2

In [86]:

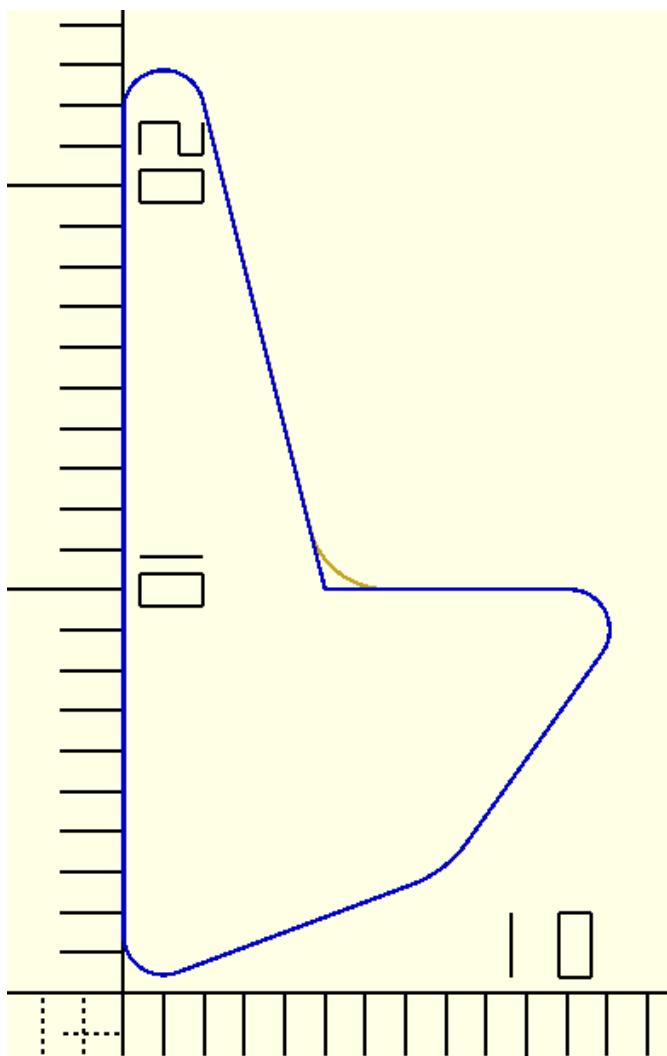
```
# example of function convert_secv2(sec,d)

sec=corner_radius(pts1([[0,0,1],[8,3,3],[5,7,1],[-8,0,2],[-5,20,1]]),40)
# sec=corner_radius(pts1([[0,0,.1],[7,5,2],[5,7,3],[-5,7,2],[-7,5,3]]),20)
# sec=corner_radius(pts1([[-15,0,2.49],[0,15,3],[30,0,3],[0,-15,2.49],[5,0,2.49],[0,20,7],[-40,0,7],[0,0,1]]),40)
# sec=circle(10)
# sec=pts([[0,0],[10,0],[0,10],[-10,0]])
# path=corner_radius(pts1([[2,0],[-2,0,2],[0,7,4.2],[-4.2,0]]),40)
# path=corner_radius(pts1([[2,0],[-2,0,2.2],[0,7,5.3],[-5.3,0]]),40)
# path=corner_radius(pts1([[2,0],[-2,0,2],[0,7,2],[-2,0]]),20)

with open('trial.scad','w+') as f:
    f.write(f'''')
include<dependencies2.scad>

p_line({sec},.1);
color("blue")p_line({convert_secv2(sec,5)},.1);

'''')
```



c2ro

In [2]:

```
# example of function c2ro(sol,s), 'c2ro' stands for circular to rectangular orientation

cyl=linear_extrude(circle(5,s=30),10)

cyl1=translate([15,0,0],c2ro(cyl,1))

cyl2=translate([30,0,0],cpo(c2ro(cyl,5)))

with open('trial.scad','w+')as f:
    f.write(f'''')
include<dependencies2.scad>
```

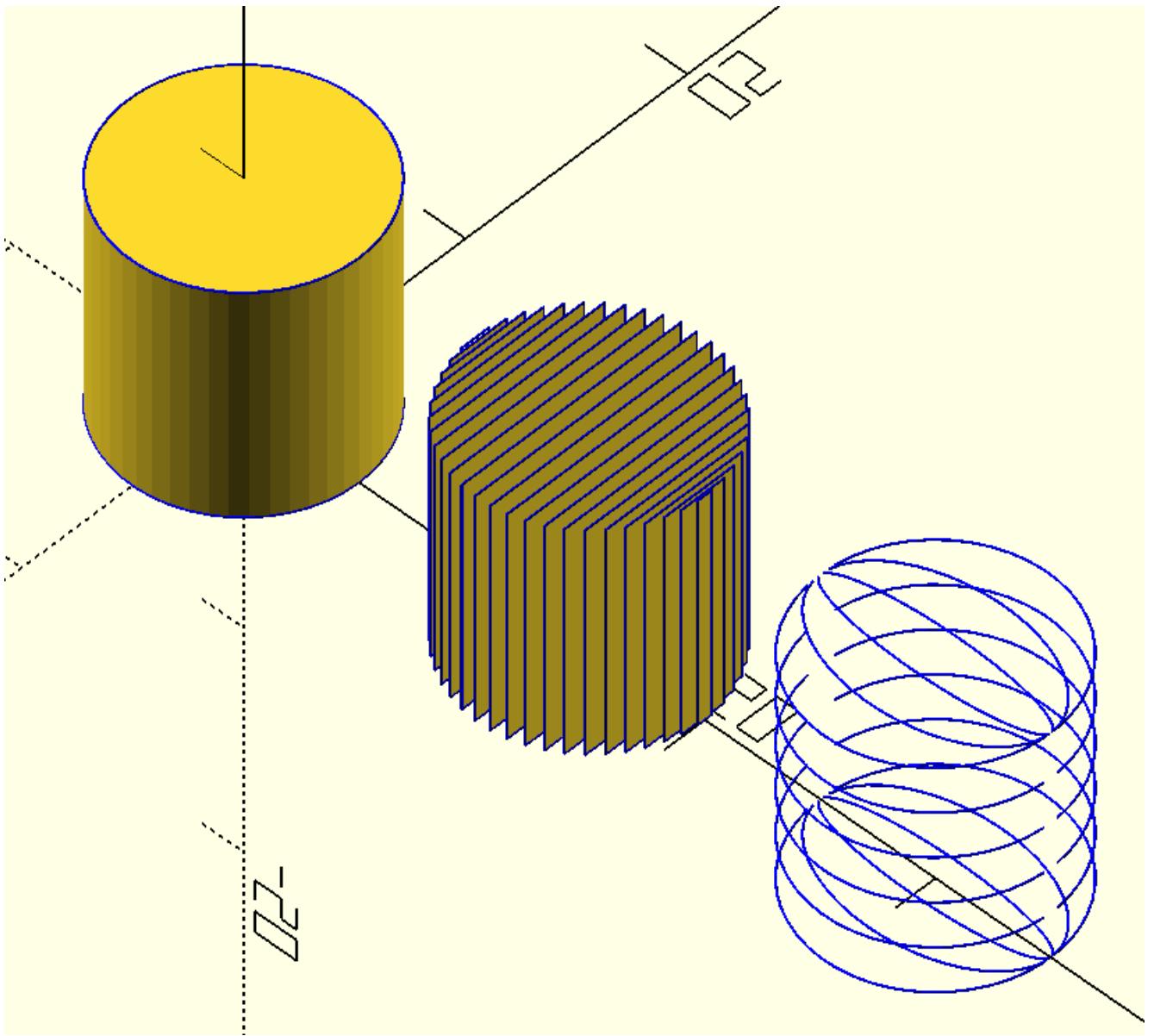
```

color("blue")for(p={cyl})p_line3dc(p,.05);
color("blue")for(p={cyl1})p_line3dc(p,.05);
color("blue")for(p={cyl2})p_line3d(p,.05);

    ''')
for i in range(14):
    with open('trial.scad','a')as f:
        f.write(f'''
{swo(cyl1[i:i+1])}

{swo(cyl)}
'''')

```



```

In [3]: # example of function c2ro(sol,s)

sp=sphere(5,s=21)

sp1=translate([15,0,0],c2ro(sp,3))

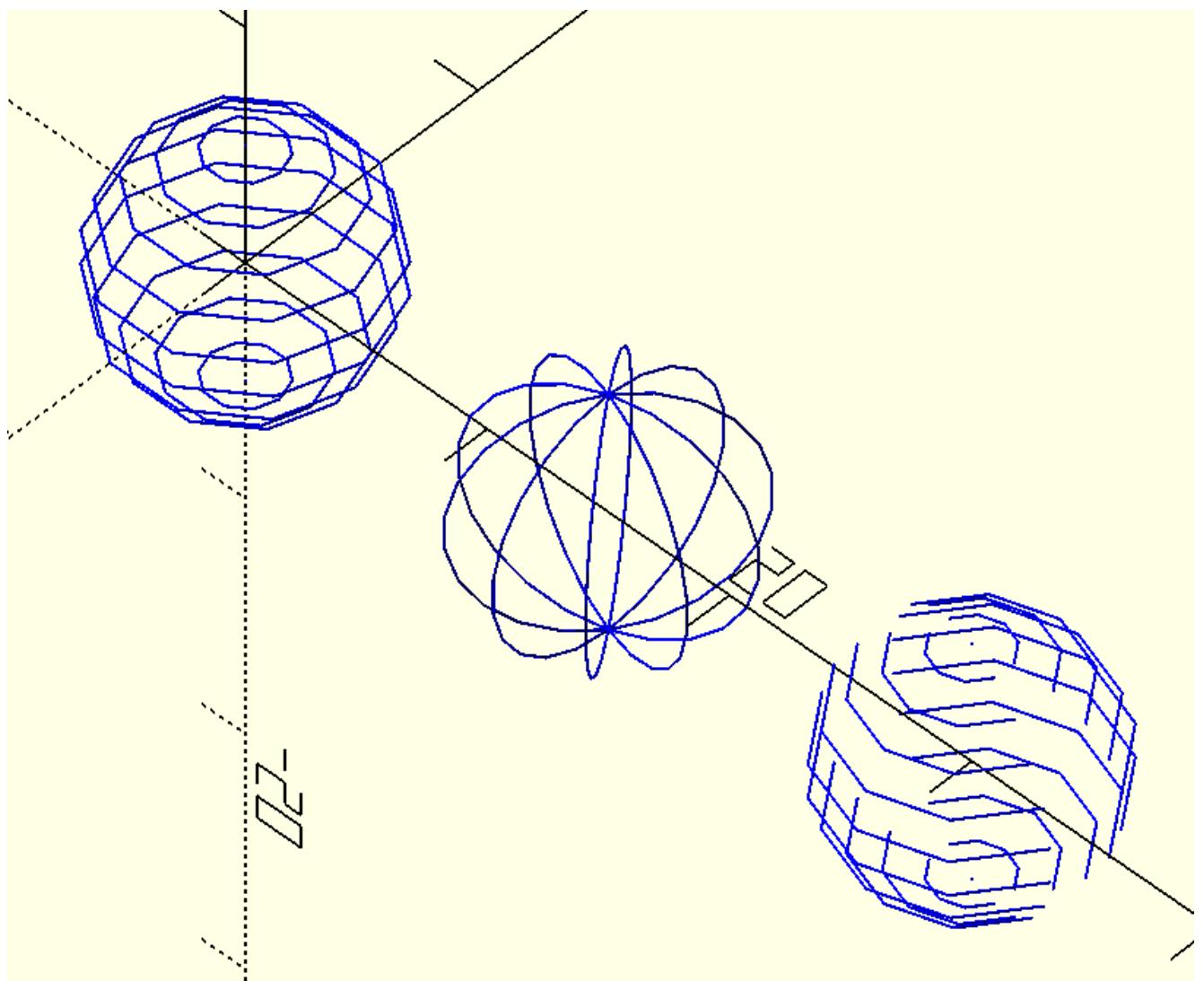
sp2=translate([30,0,0],cpo(c2ro(sp,3)))

with open('trial.scad','w+')as f:
    f.write(f'''
include<dependencies2.scad>

color("blue")for(p={sp})p_line3dc(p,.1);
color("blue")for(p={sp1})p_line3dc(p,.1);
color("blue")for(p={sp2})p_line3d(p,.1);

```

...)



In [4]: # application of function c2ro(sol, s) for creating fillet between 2 cylinders

```
sec=circle(5,s=50)
path=corner_radius(pts1([[-1,2],[1,0,1],[0,10,1],[-1,0]]),10)

sol1=linear_extrude(circle(10,s=100),14)

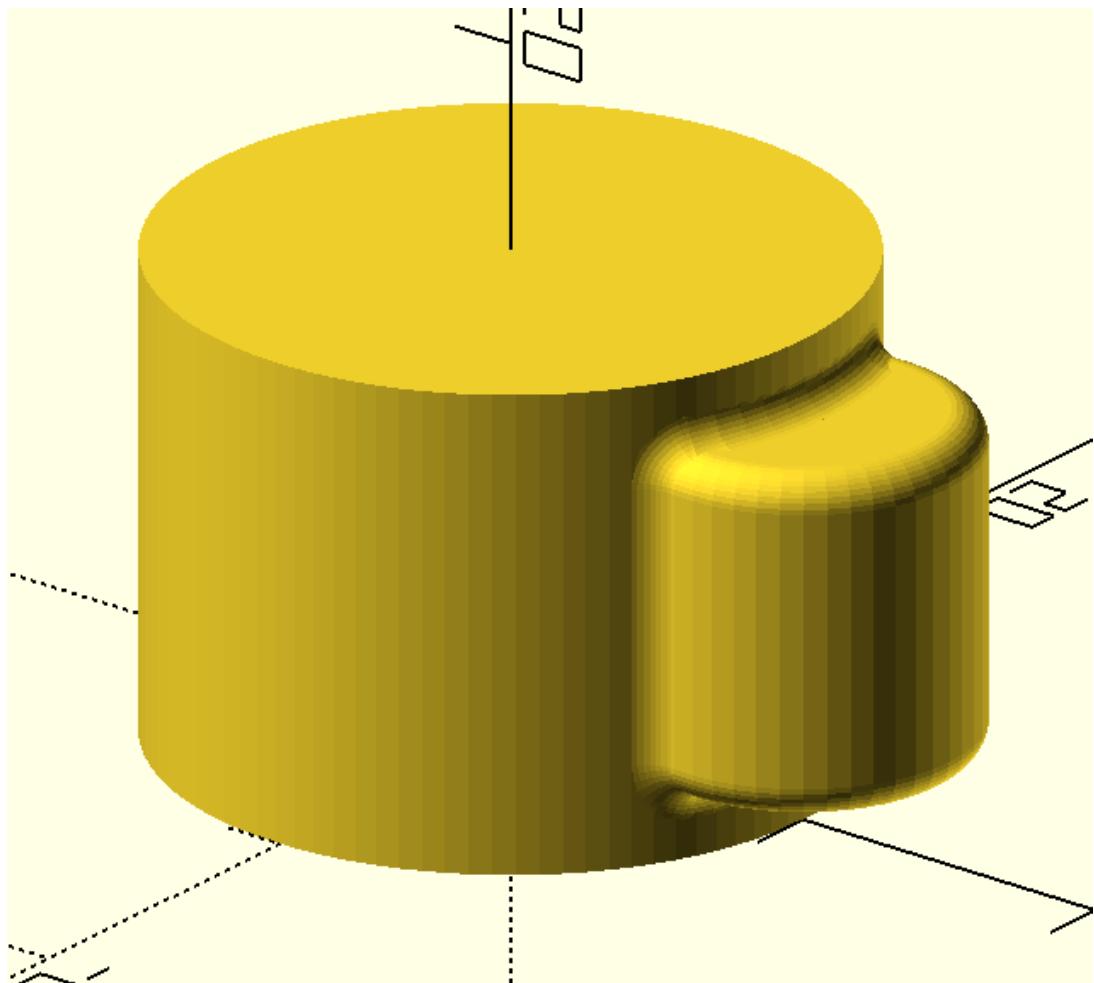
sol2=translate([10,0,0],c2ro(prism(sec,path),10))

sol3=translate([10,0,0],prism(sec,path))

sol5=ip_fillet(sol1,flip(sol2),1,-1)
sol5=cpo(sol5)[-1]

with open('trial.scad','w+')as f:
    f.write(f'''
include<dependencies2.scad>
%{swp(sol3)}
%{swp(sol1)}
{swp(sol5)}

//color("blue")for(p={cpo(sol2)})p_line3d(p,.02);
//color("blue")for(p={sol5})p_line3dc(p,.02);
    ''')
```



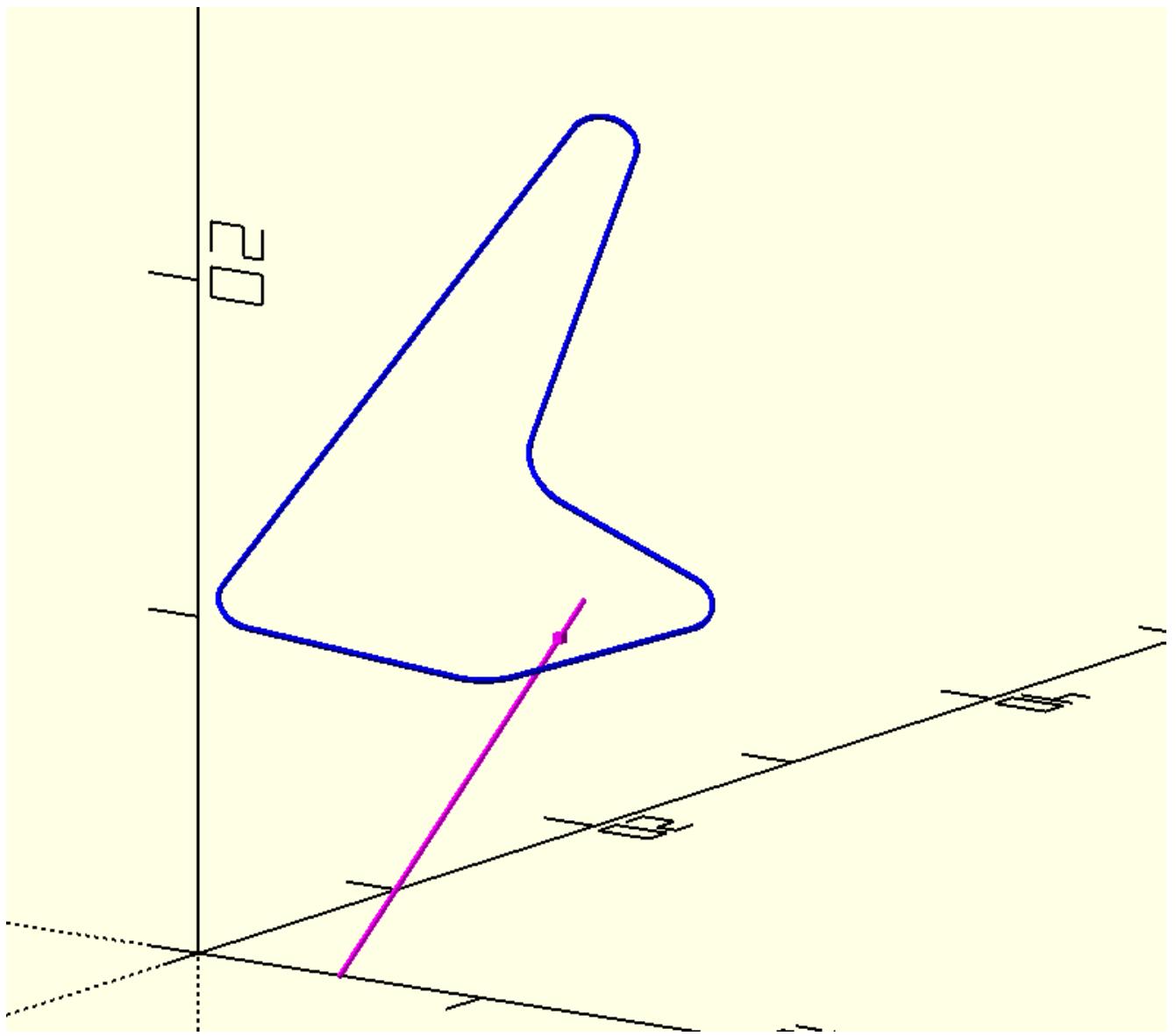
sl_int

```
In [90]: # example of function sl_int(sec,line)

sec=corner_radius(pts1([[0,0,1],[8,3,3],[5,7,1],[-8,0,2],[-5,20,1]]),20)
sec=translate([0,0,10],q_rot(['y30','x30'],sec))
line=[[5,0,0],[8,8,10]]
ip_1=sl_int(sec,line)

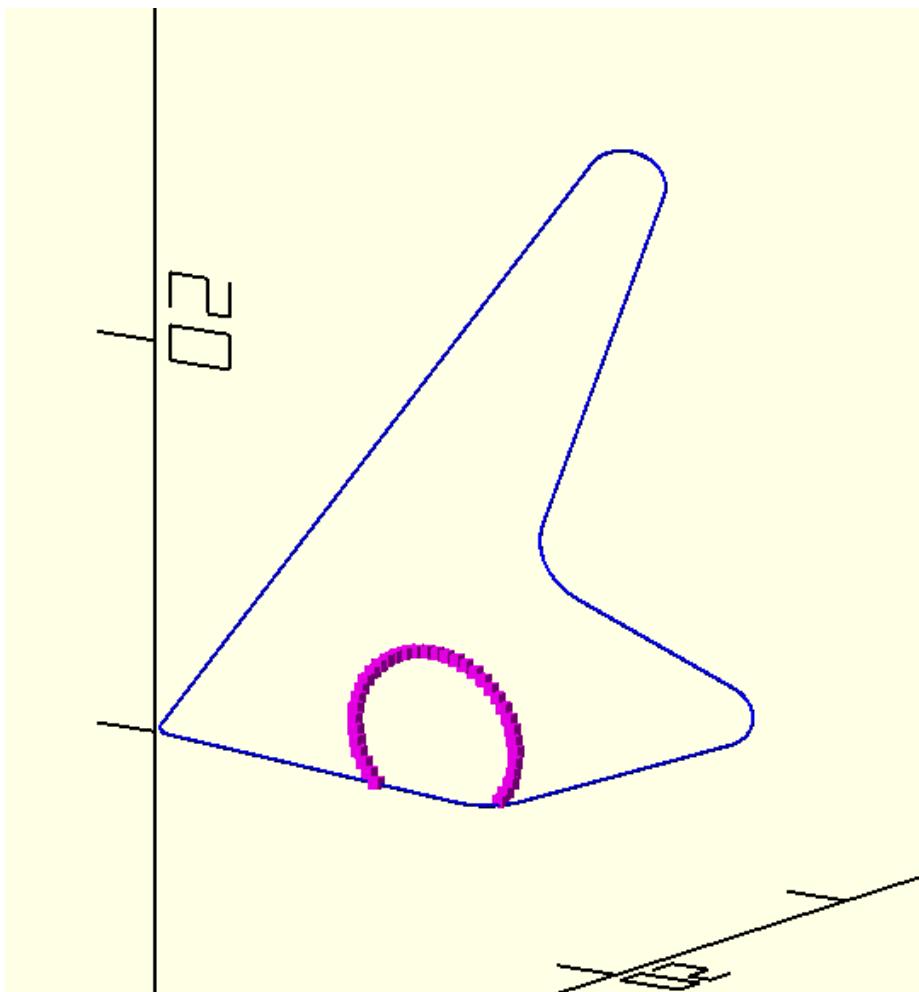
with open('trial.scad','w+')as f:
    f.write(f'''
include<dependencies2.scad>
color("blue")p_line3dc({sec},.1);
color("magenta")p_line3d({line},.1);
color("magenta")points({ip_1},.3);

    ''')
```



```
In [91]: sec=corner_radius(pts1([[0,0,1],[8,3,3],[5,7,1],[-8,0,2],[-5,20,1]]),20)
sec=translate([0,0,10],q_rot(['y30','x30'],sec))
sol=cpo(linear_extrude(circle(2,[5,9],s=50),20))
# lines=array(sol).reshape(-1,2,3)
ip_1=[sl_int(sec,line) for line in sol if sl_int(sec,line)!=[]]
ip_1=concatenate(ip_1).tolist()
with open('trial.scad','w+')as f:
    f.write(f'''
include<dependencies2.scad>
polyhedron({sec},{{arange(len(sec)).tolist()}});
color("blue")p_line3dc({sec},.05);
color("magenta")points({ip_1},.1);
%{swp(cpo(sol))}

    ''')
```



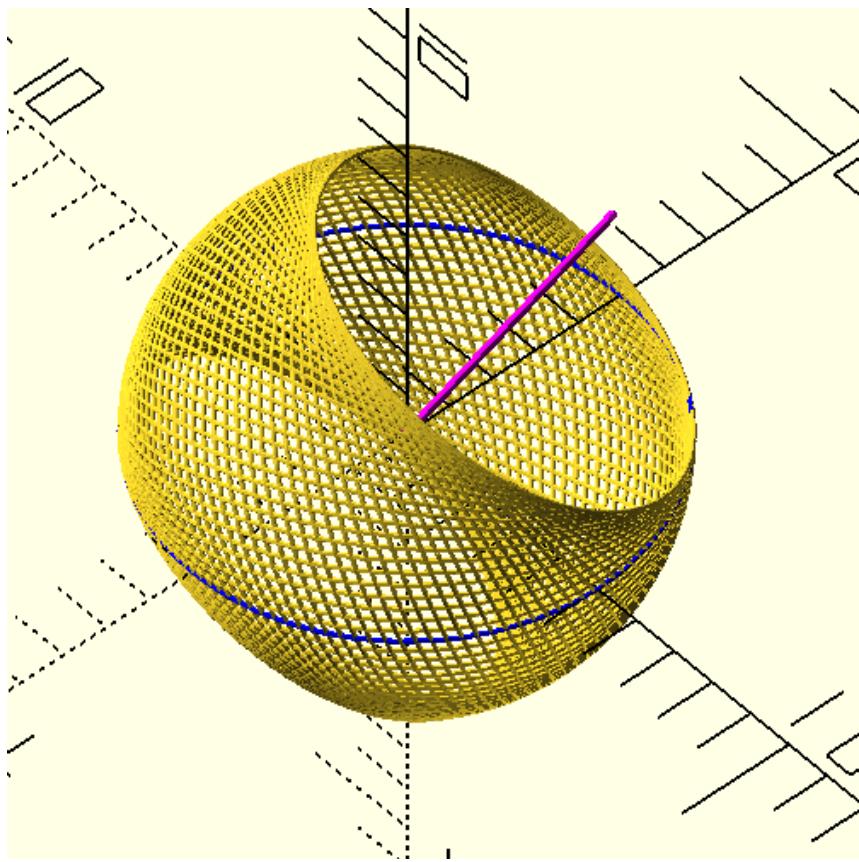
axis_rot

```
In [92]: # example of function axis_rot(axis,solid,angle)
vector=[2,3,5]
# sec=translate([-5,5,0],circle(5))
sec=path_extrude_closed(circle(.05),c2t3(circle(5)))
sec1=[path_extrude_closed(circle(.05),axis_rot(vector,circle(5),i)) for i in arange(0,360,20)]

with open('trial.scad','w+')as f:
    f.write(f'''
include<dependencies2.scad>

for(p={sec1})swp(p);
//original section
color("blue"){swp(sec)}
// axis of rotation
color("magenta")p_line3d({[[0,0,0]},vector],.1);

    ''')
```

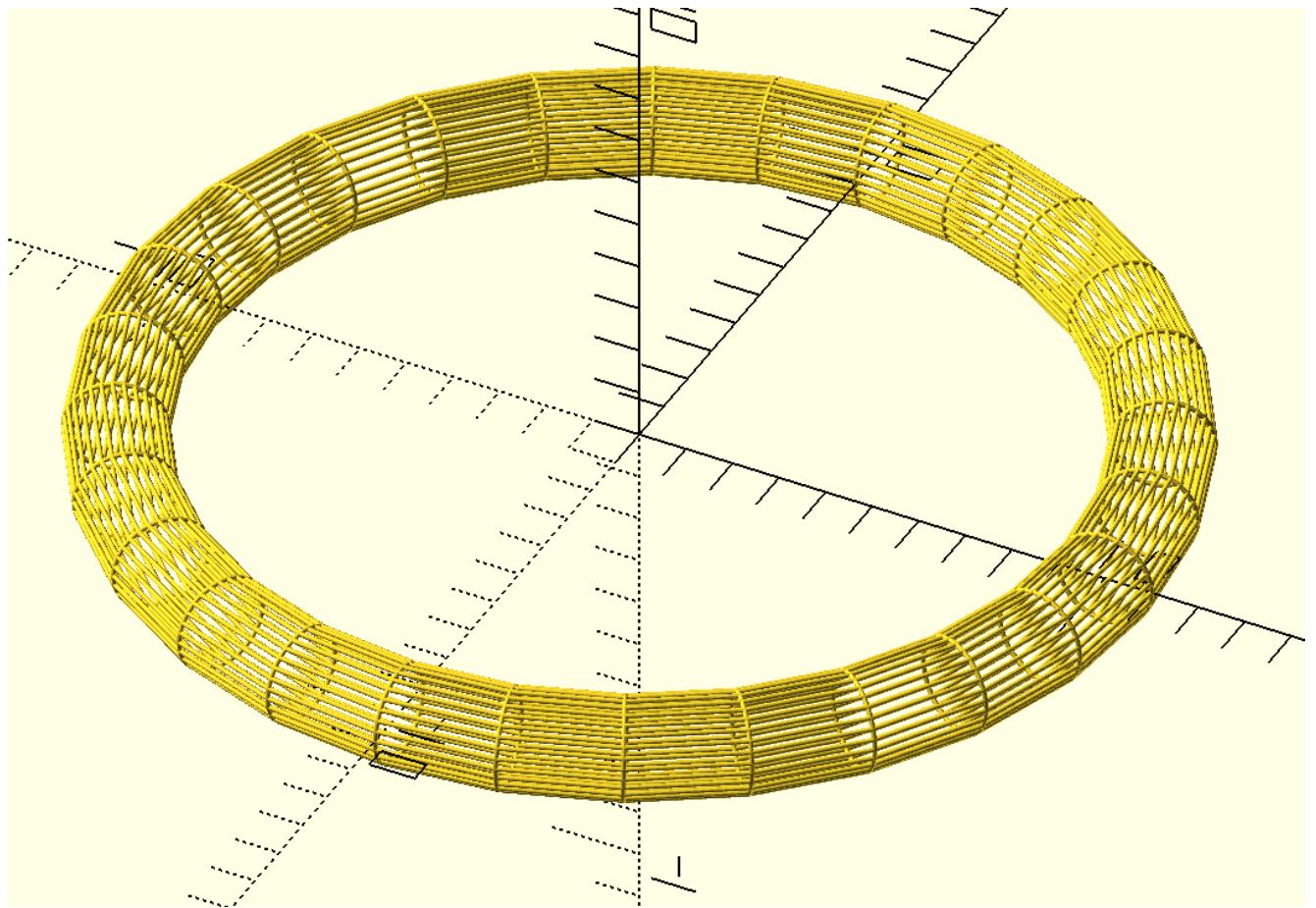


path_extrude_closed

```
In [97]: sec=circle(1,s=20)
path=c2t3(circle(10,s=20))
sol1=path_extrude_closed(sec,path)

with open('trial.scad','w+')as f:
    f.write(f'''
include<dependencies2.scad>
$fn=20;
for(p={sol1})p_line3dc(p,.05);
for(p={cpo(sol1)})p_line3dc(p,.05);

    ''')
```



cam-profile

```
In [93]: # example of cam profile

path1=m_points_o(corner_radius(pts1([[0,15],[12,0,5],[10,-5,.5],[22,0]]),5),.5)
path2=array(c2t3(arc(10,0,362,s=len(path1)-1)))

path_h=array([[0,0,y] for (x,y) in path1])
path3=(path2+path_h).tolist()
sketch1=[[-1.5,0],[1.5,0]]
surf1=surf_extrude(sketch1,path3)
surf2=c2t3(c3t2(surf1))
s1,s2=array(surf1),array(surf2)

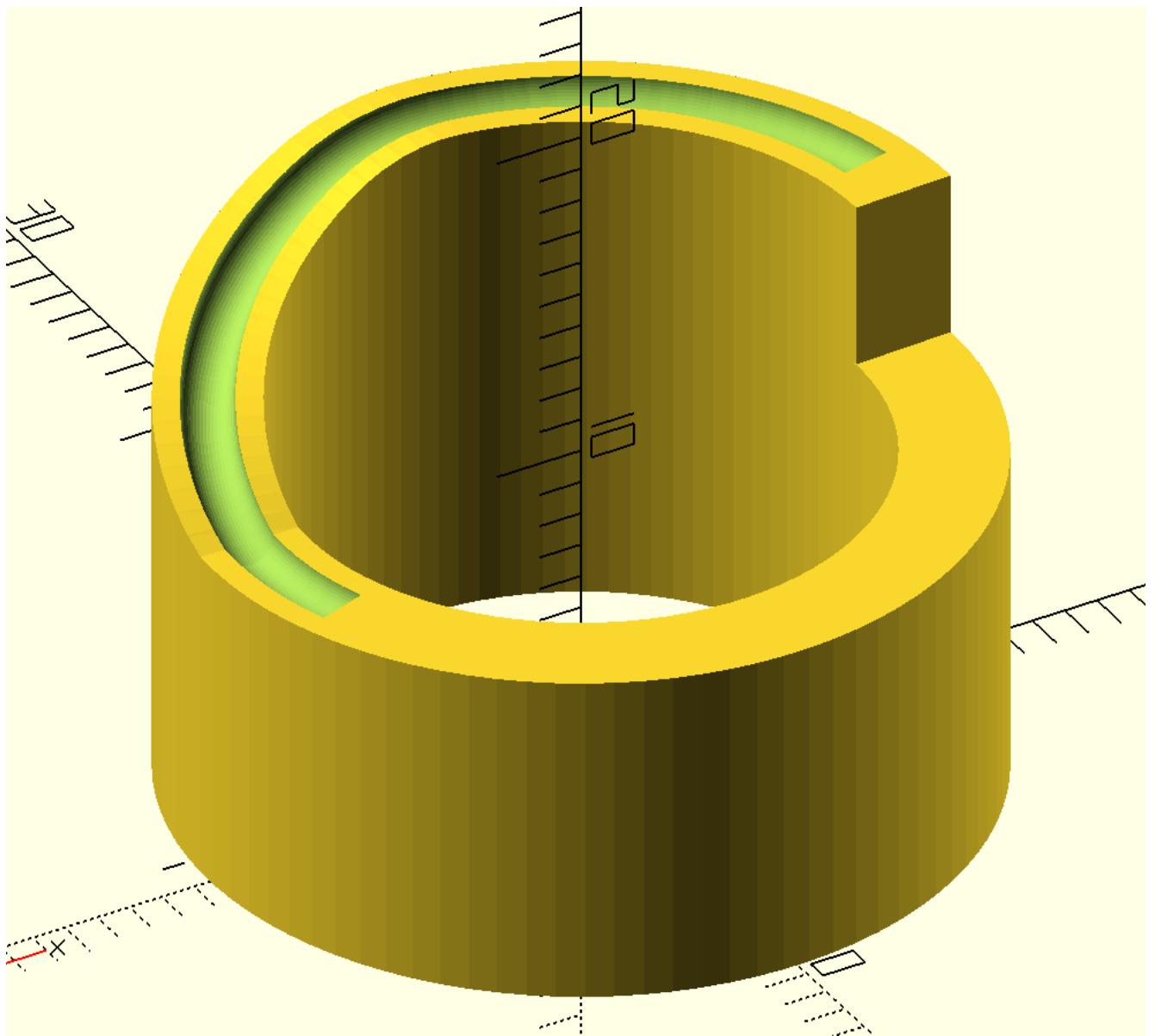
a,b=s2.transpose(1,0,2)
c,d=s1.transpose(1,0,2)
sol=array([a,c,d,b]).transpose(1,0,2).tolist()

arc1=translate([0,0,0],arc(10,10,200,s=50))
arc2=translate([0,0,50],arc1)
sol1=[arc1,arc2]

ip_1=ip_surf(sol,sol1)
ip_1=min_d_points(ip_1,.1)

sec=circle(.75)
sol2=path_extrude_open(sec,ip_1)

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>
//points({path1},.2);
//points({path3},.2);
//color("magenta")p_line3d({ip_1},.2);
difference(){
{s1}
{s2}
}
'''')
```



2W-wheel

In [94]: # 2w wheel completely designed in openscad, it takes 160 sec to compute

```
t1=time.time()

p0=[[-3,-5-41/2,3],[15,-5,2],[0,7,5],[-12,3,2],[-1,8,.5],[1,1,1.5],
    [-.25,2,1.5],[-8,2,3],[0,15,3],[8,2,1.5],[0.25,2,1.5],
    [-1,1,1],[1,8,2],[12,3,5],[0,7,2],[-15,-5,3],[-2,-13,2],
    [-8,-2,3],[-2,-10.5,3],[2,-10.5,3],[8,-2,3]]
sec=corner_radius(pts1(p0),10)
path=c2t3(circle(12*25.4/2,s=100))
sol=path_extrude_closed(sec,path)

p1=[[0,-15,.5],[6,0,.3],[0,-16,.1],[1,-1,.1],[4,0,.2],[2,10,4],[35,0,10],
    [5,22,5],[-5,22,10],[-35,0,4],[-2,10,.2],[-4,0,.1],[-1,-1,.1],[0,-16,.3],
    [-6,0,.5]]
sec1=corner_radius(pts1(p1),10)
sec2=offset(sec1,-4)
path1=c2t3(circle(10,s=72))
sol1=path_extrude_closed(sec1,path1)
sol2=path_extrude_closed(sec2,path1)

sec3=circle(7)
path2=corner_radius(pts1([[2,0],[-2,25+7]]),10)

# Disc mounting post top
sol3=translate([51,0,18-7],prism(sec3,path2))

arc1=flip(c2t3(arc(30,13,60,s=10)))
arc2=c2t3(arc(45,18,55,s=10))
```

```

p2=(array(arc2[0])+[0,0,3]).tolist()
p3=arc2[1:-1]
p4=(array(arc2[-1])+[0,0,3]).tolist()
p5=(array(arc1[0])+[0,0,3]).tolist()
p6=arc1[1:-1]
p7=(array(arc1[-1])+[0,0,3]).tolist()

pl=[p2]+p3+[p4]+[p5]+p6+[p7]
sec4=corner_radius(pl,10)
# sand extraction opening top
sol4=linear_extrude(sec4,50)

# Disc mounting post bottom
sol5=q_rot(['z45'],translate([51,0,-18+7],q_rot(['y180'],prism(sec3,path2)))) 

# sand extraction opening bottom
sol6=q_rot(['z-30'],translate([0,0,-50],sol4))

sec5=corner_radius(pts1([[ -20,-7.5,2.49],[5,0,2.49],[0,10,3],[15,2,70],[15,-2,3],[0,-10,2.49],
[5,0,2.49],[1,10,1],[-1,5,7],[-20,3,90],[-20,-3,7],[-1,-5,1]]),10)

sol7=translate([0,50,-2.5],q_rot(['x90','z200'],[c2t3(sec5),translate([0,0,93],scl2d_c(sec5,.7))]))

fillet1=ip_fillet_surf(sol1,sol3,4,-4)
fillet2=ip_fillet_surf(sol1,sol5,4,-4)

fillet3=ip_fillet(sol1,sol7,6,-6)
fillet4=ip_fillet(sol,flip(sol7),5,5)

end_round_1=end_cap(q_rot(['x.0001'],sol3),2)[1]
end_round_2=end_cap(q_rot(['x.0001'],sol5),2)[1]

with open('trial.scad','w+')as f:
    f.write(f''' 
include<dependencies2.scad>

{swp_c(sol)}
difference()
{{{
    difference()
    {{{
        union()
        {{{
            {swp_c(sol1)}
            for(i=[0:360/5:359])
            rotate([0,0,i])
            {{{
                difference()
                {{{
                    {swp(sol3)}
                    {swp_c(end_round_1)}
                }}}
                difference()
                {{{
                    {swp(sol5)}
                    {swp_c(end_round_2)}
                }}}
                {swp(sol7)}
            }}}
        }}}
    }}}
    {swp_c(sol2)}
}}}

for(i=[0:360/5:359])
rotate([0,0,i])
{{{
    {swp(sol4)}
    {swp(sol6)}
}}}
}}
```

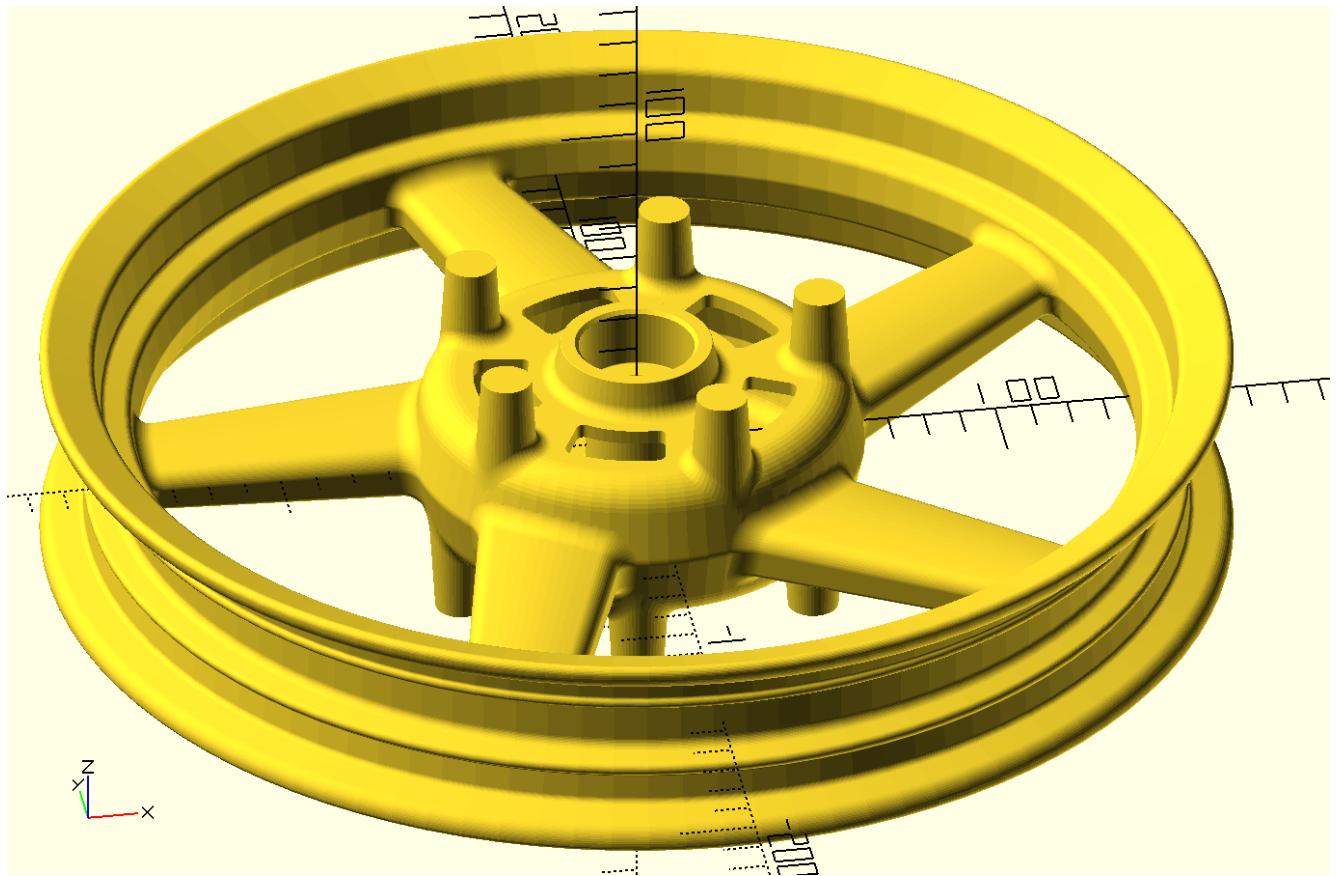
```

for(i=[0:360/5:359])
rotate([0,0,i])
{{
{swp_c(fillet1)}
{swp_c(fillet2)}
{swp_c(fillet3)}
{swp_c(fillet4)}

}}
...
t2=time.time()
t2-t1

```

Out[94]: 7.3932108879089355



drill-bit

```

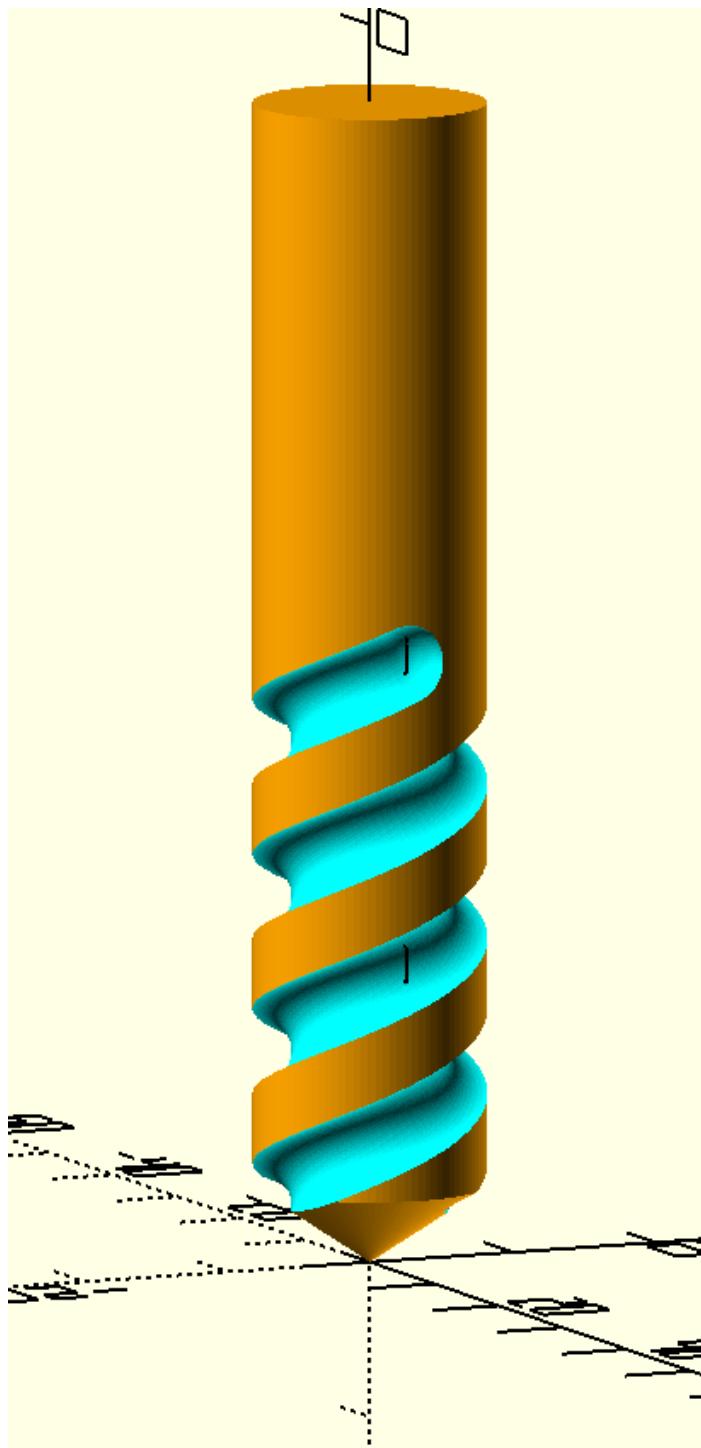
In [7]: # drill bit

sec=circle(7.5,s=101)
path=pts([[ -7.5*cos(360/200*pi/180),0],[7.5,5],[0,70]])
sol=prism(sec,path)
hx1=helix(7.5,20,2,5)
hx2=q_rot(['z180'],hx1)
with open('trial.scad','w+')as f:
    f.write(f'''
        include<dependencies2.scad>
//      render(){{

difference(){
    color("orange"){swp(sol)}
    color("cyan")p_line3d({hx1},2.5,$fn=70);
    color("cyan")p_line3d({hx2},2.5,$fn=70);

}
//  }}
    ''')

```



bottle-with-cut-design

```
In [95]: # bottle with cut design

t0=time.time()

sec=circle(10,s=100)
path=corner_radius(pts1([[-3,0,0],[3,0,3],[3,5,7],[-5,20,100],[8,30,20],[-11,10,5],[0,10,0]]),50)
path=equidistant_path(path,500)
sol=prism(sec,path)

sec1=corner_radius(pts1([[5,10,3],[10,0,3],[0,20,30],[-3,20,30],[0,10,1.9],[-4,0,1.9],[0,-10,30],[-3,0,0]]),10)
sec1=equidistant_pathc(sec1,200)
path1=corner_radius(pts1([[1,0],[-1,0,1],[-1,1,1],[-.5,0]]),10)
path1=equidistant_path(path1,20)
sol1=prism(sec1,path1)

# Solid above has some open space at the center which might create artifacts on bending
# so to avoid that the empty space needs to be filled with more points
l1=equidistant_path([[0,12,1],[0,58,1]],100)
l1=sort_points(sol1[-1],l1)
sol1_1=sol1[:-1]
sol1_2=slice_sol([sol1[-1]]+[l1],10)
```

```

sol1=sol1_1+sol1_2

path2=q_rot(['y90'],path)

sol2=[wrap_around(p,path2) for p in sol1]

sol2=flip(sol2)+translate([0,0,-3],sol2)

sol2=translate([0,10,0],q_rot(['z-90'],sol2))

path3=q_rot(['y90'],circle(10,s=1000))
sol2=[wrap_around(p,path3) for p in sol2]

sol3=flip(q_rot(['y-90'],sol2))

# offset the original bottle by -0.5mm
sol4=prism(sec, path_offset(path,-.5))

sec1=corner_radius(pts1([-5,10,3],[10,0,3],[0,20,30],[-3,20,30],[0,10,1.9],[-4,0,1.9],[0,-10,30],[-3,
sec1=equidistant_pathc(sec1,200)

# offset the cut-out path by 0.5mm
path1=path_offset(path1,.5)

sol1=prism(sec1,path1)
l1=equidistant_path([[0,12,1.5],[0,58,1.5]],100)
l1=sort_points(sol1[-1],l1)
sol1_1=sol1[:-1]
sol1_2=slice_sol([sol1[-1]]+[l1],10)
sol1=sol1_1+sol1_2
path2=q_rot(['y90'],path)

sol2=[wrap_around(p,path2) for p in sol1]

sol2=flip(sol2)+translate([0,0,-3],sol2)
sol2=translate([0,10,0],q_rot(['z-90'],sol2))
path3=q_rot(['y90'],circle(10,s=500))
sol2=[wrap_around(p,path3) for p in sol2]
sol5=flip(q_rot(['y-90'],sol2))

with open('trial.scad','w+')as f:
    f.write(f'''
include<dependencies2.scad>

difference(){
difference(){
{swp(sol)}
for(i=[0,90,180,270])
rotate([0,0,i])
{swp(sol3)}

}

difference(){
{swp(sol4)}
for(i=[0,90,180,270])
rotate([0,0,i])
{swp(sol5)}

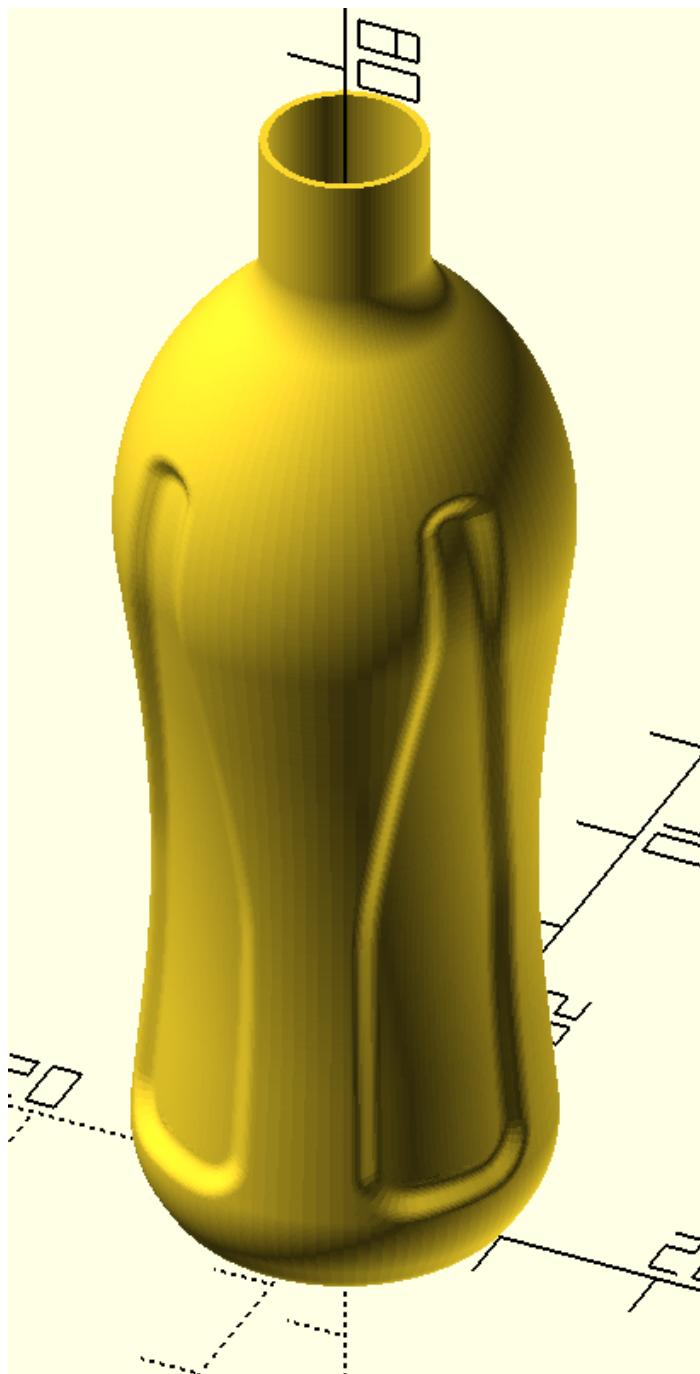
}

})
})

t1=time.time()
t1-t0

```

Out[95]: 10.957280158996582



example-of-rounding

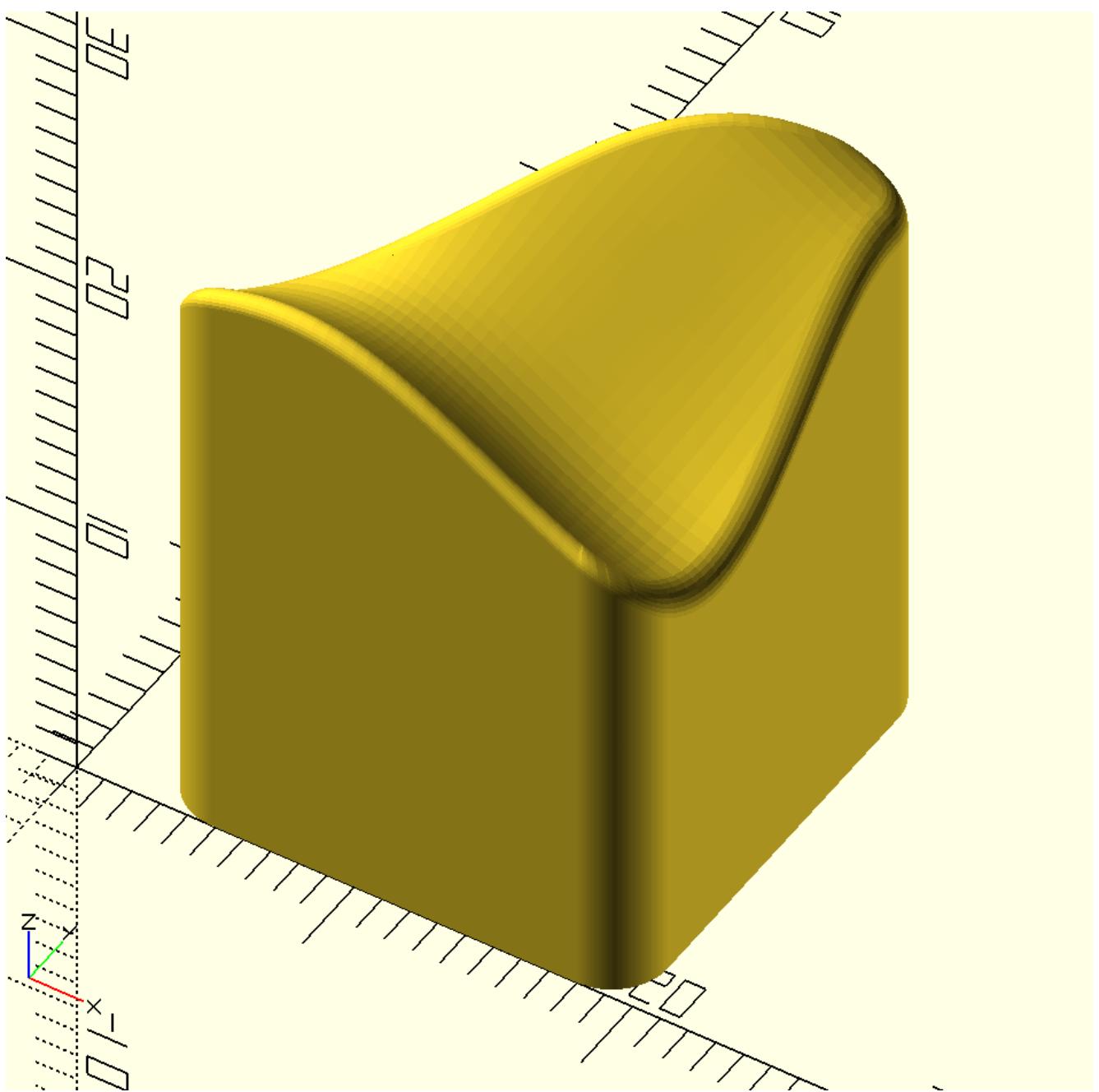
```
In [95]: # m37 rounded
sec=[[i,3*sin(i*18*pi/180)] for i in arange(-20,20,.5)]
path=[[i,0,20+3*sin(i*12*pi/180)] for i in arange(-30,30,.5)]
surf=surf_extrude(sec,path)
sec1=corner_radius(pts1([[0,0,2],[20,0,2],[0,20,2]]),40)
sol1=translate([0,0,-.5],q_rot(["z.001"],linear_extrude(m_points(sec1,.5),40)))
sol01=translate([0,0,-.5],q_rot(["z.001"],linear_extrude(m_points(offset(sec1,.5),.5),40)))

sol2=cpo(ipf(surf,flip(sol1),1,10,0))[1:]
ip=sort_points(sol2[0],ip_surf(surf,sol01))
sol2=[translate([0,0,10],ip)+[ip]+sol2+[translate([0,0,10],sol2[len(sol2)-1])]+[translate([0,0,10],ip
with open('trial.scad','w+')as f:
    f.write(f'''
    render(){}
    difference(){}
    intersection{}{}
```

```

{swp(surf_base(surf,0))
{swp(sol1)}
}
{swp_c(sol2)}
}
}
...
)

```



m39

In [3]: # m39

```

p0=i_p2d(pts([[-45,0],[0,1]]),cir_theta_line(10,[-35,0],16.5,1))
p1=i_p2d(cir_theta_line(10,[-35,0],16.5,1),cir_theta_line(10,[35,0],-16.5,1))
p2=i_p2d(pts([[45,0],[0,1]]),cir_theta_line(10,[35,0],-16.5,1))
p3=[45,35]
p4=i_p2d([p3,p_cir_t(p3,circle(10,[0,44.06]))],[cir_p_t(circle(10,[0,44.06]),[-45,35]),[-45,35]])
p5=[-45,35]
p_l=array(c2t3([p0,p1,p2,p3,p4,p5]))
p_l=p_l+array([[0,0,i] for i in [10,32.5,10,10,10,10]])

sec1=corner_radius(p_l,20)

p0,p1,p2,p3,p4,p5=offset([p0,p1,p2,p3,p4,p5],-2.5)

c1,c2,c3=[circle(7.5,i) for i in [[-35,0],[35,0],[0,44.06]]]
a1=flip(fillet_l_cir([p0,p1],c1,2.5,10)[1])
a2=fillet_intersection_lines([p0,p1],[p2,p1],35,20)

```

```

a3=fillet_l_cir([p1,p2],c2,2.5,10)[0]
a4=flip(fillet_l_cir([p2,p3],c2,2.5,10)[1])
a3_1=arc_2p(a3[-1],a4[0],7.5,1,20)
a5=fillet_intersection_lines([p2,p3],[p4,p3],7.5,10)
a6=fillet_l_cir([p3,p4],c3,2.5,10)[0]
a7=flip(fillet_l_cir([p4,p5],c3,2.5,10)[1])
a6_1=arc_long_2p(a6[-1],a7[0],7.5,1,40)
a8=fillet_intersection_lines([p4,p5],[p0,p5],7.5,10)
a9=fillet_l_cir([p5,p0],c1,2.5,10)[0]
a10=arc_2p(a9[-1],a1[0],7.5,1,20)
sec2=a1+a2+a3+a3_1+a4+a5+a6+a6_1+a7+a8+a9+a10
sec2=remove_extra_points(array(sec2).round(4))

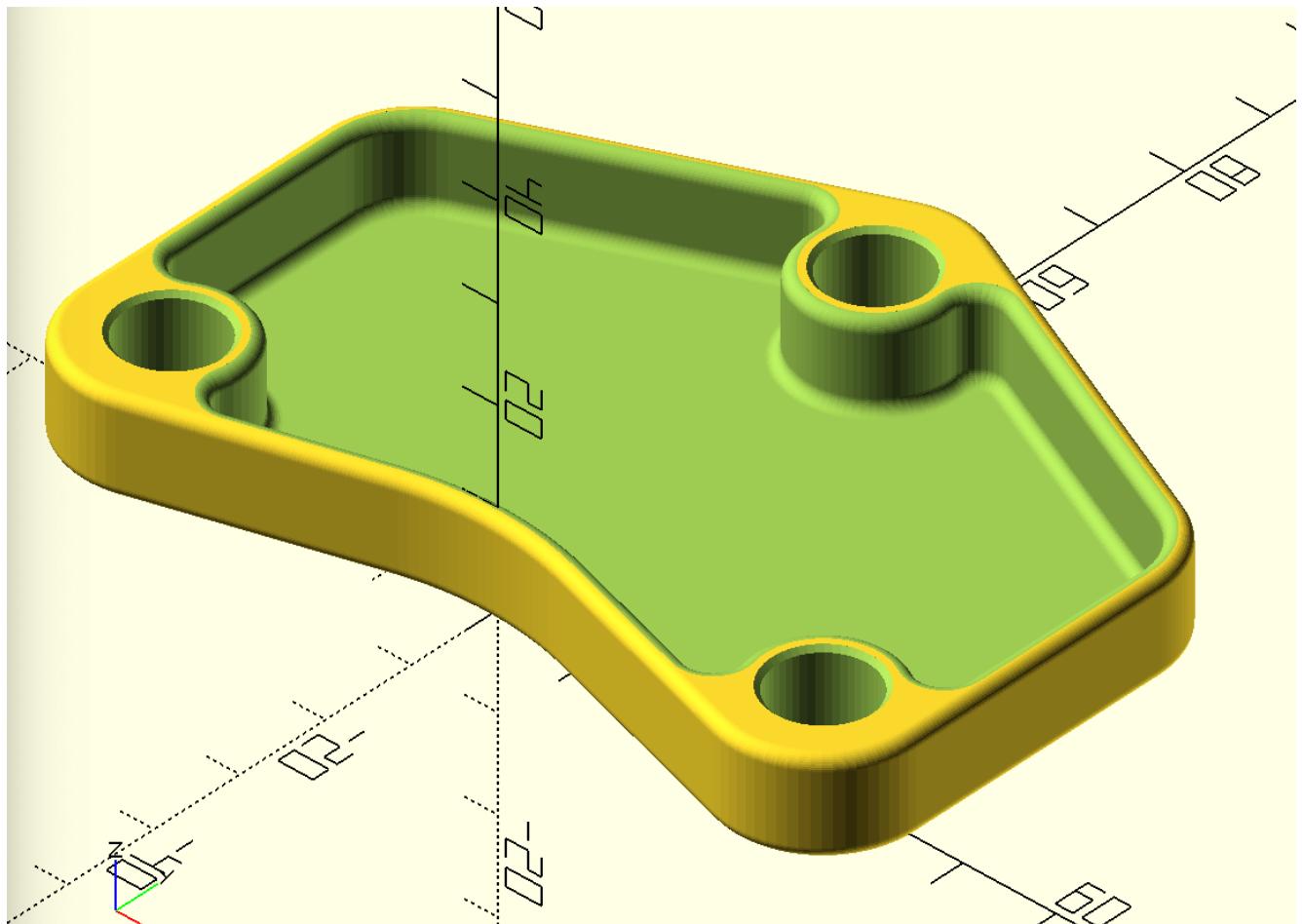
c4,c5,c6=[circle(5,i)  for i in [[-35,0],[35,0],[0,44.06]]]

path1=corner_radius(pts1([[ -1.25,0],[1.25,0,1.25],[0,10,1.25],[-1.25,0]]),10)
sol1=prism(sec1,path1)

path2=corner_radius(pts1([[ -1.25,1.25],[1.25,0,1.25],[0,10-1.25,1.25],[1.25,0],[0,.2]]),10)
sol2=prism(sec2,path2)

path3=corner_radius(pts1([[0.5,-.2],[0,0.2],[-.5,.5],[0,9],[.5,.5],[0,.2]]),5)
sol3=[prism(s,path3) for s in [c4,c5,c6]]
with open('trial.scad','w+') as f:
    f.write(f''''
include<dependencies2.scad>
difference() {{
    {swp(sol1)}
    {swp(sol2)}
    for(p={sol3}) swp(p);
}}
'''')

```



4W-wheel

```

In [4]: # 4W
i_t=time.time()

# spoke1
sec1=corner_radius([[300,0,7], [0,0,10] , [300*cos(-30*pi/180),300*sin(-30*pi/180),7]]+c2t3(arc(300,-3

```

```

sec4=corner_radius(pts1([[3,0,0],[0,3,1.49],[-3,0,1.49],[0,-3,0]]),10)
sec=corner_radius(pts1([[-100,0],[100,30,500],[100,-30]]),20)
path=translate([0,0,217],q_rot(["y16","z15"]),cytz(corner_radius(pts1([[0,0],[130,30,1000],[260,-10,0]])))
surf=surf_extrude(sec,path)
sec5=corner_radius([[300*cos(30*pi/180),300*sin(30*pi/180),7],[0,0,10],[300,0,7]]+c2t3(arc(300,0,30,[0,0,10]))+linear_extrude(sec5,300)
sec6=corner_radius([[300*cos(30*pi/180),300*sin(30*pi/180),10],[0,0,30],[300,0,10]]+c2t3(arc(300,0,30,[0,0,30])))
sol5=translate([50*cos(15*pi/180),50*sin(15*pi/180),0],linear_extrude(m_points(sec6,10),300))
ip2=ip_surf(surf,sol5)
ip2=remove_extra_points(array(ip2).round(4))
bead2=path_extrude_closed(sec4,ip2)
cyl1=translate([110*cos(15*pi/180),110*sin(15*pi/180)],cylinder(r=15,h=300,s=40))
ip3=ip_surf(surf,cyl1)
ip3=remove_extra_points(array(ip3).round(4))
bead3=path_extrude_closed(sec4,ip3)
sec9=circle(245,s=100)
path9=corner_radius(pts1([[2,0],[-2,0,2],[0,5,5],[-13,20,5],[0,64,5],[-12,20,5],[0,78,5],[12,20,5],[0,85,5]]),20)
sol7=prism(sec9,path9)
sec12=corner_radius([[50,0,4],[0,0,10],[50*cos(-30*pi/180),50*sin(-30*pi/180),4]]+c2t3(arc(50,-30,0,[0,0,10])))
sol12=linear_extrude(sec12,300)
sol14=q_rot(["z30"],sol12)
sol15=ip_fillet_surf(surf,sol14,3,-3)
sol16=linear_extrude(m_points(offset(sec1,-2.5),10),300)
sol18=q_rot(["z30"],sol16)
sol19=ip_fillet_surf(surf,sol18,3,3)
surf3=translate([0,0,-5],surf)
sol27=ip_fillet_surf(surf3,flip(sol18),3,3)
sol28=ip_fillet(surf3,flip(sol14),3,-3)

```

```

# spoke2
i_t=time.time()
sec10=corner_radius(pts1([[-100,0],[100,30,500],[100,-30]]),20)
path10=translate([0,0,190],q_rot(["y-4","z-15"]),cytz(corner_radius(pts1([[0,0],[130,30,1000],[260,-10,0]])))
sec1=corner_radius([[300,0,7],[0,0,10],[300*cos(-30*pi/180),300*sin(-30*pi/180),7]]+c2t3(arc(300,-30,0,[0,0,10])))
sol1=linear_extrude(sec1,300)

sec9=circle(245,s=100)
path9=corner_radius(pts1([[2,0],[-2,0,2],[0,5,5],[-13,20,5],[0,64,5],[-12,20,5],[0,78,5],[12,20,5],[0,85,5]]),20)
sol7=prism(sec9,path9)

surf1=surf_extrude(sec10,path10)
surf1t=surf_extrudef(surf1,-5)
# fillet_surf1t=ip_fillet(sol7,flip(surf1t),3,3)

sec2=corner_radius([[300,0,1],[0,0,15],[300*cos(-30*pi/180),300*sin(-30*pi/180),1]]+c2t3(arc(300,-30,0,[0,0,15])))
sol2=translate([50*cos(-15*pi/180),50*sin(-15*pi/180),0],linear_extrude(sec2,300))
sec3=corner_radius([[300,0,1],[0,0,15],[300*cos(-30*pi/180),300*sin(-30*pi/180),1]]+c2t3(arc(300,-30,0,[0,0,15])))
sol3=translate([50*cos(-15*pi/180),50*sin(-15*pi/180),0],linear_extrude(m_points(sec3,10),300))
# sec4=corner_radius(pts1([[3,0,0],[0,3,1.49],[-3,0,1.49],[0,-3,0]]),10)
sec4=corner_radius(pts1([[3,-5,1.49],[0,8,1.49],[-3,0,1.49],[0,-8,1.49]]),10)
ip1=ip_surf(surf1,sol3)
bead1=path_extrude_open(sec4,ip1)
b1_fillet_1=ip_fillet(sol7,bead1[28:33],3,3)
b1_fillet_2=ip_fillet(sol7,flip(bead1[-50:-45]),3,-3)

sec12=corner_radius([[50,0,4],[0,0,10],[50*cos(-30*pi/180),50*sin(-30*pi/180),4]]+c2t3(arc(50,-30,0,[0,0,10])))
sol12=linear_extrude(sec12,300)

sol13=ip_fillet_surf(surf1,sol12,3,-3)
sol16=linear_extrude(m_points(offset(sec1,-2.5),10),300)

sol17=ip_fillet_surf(surf1,sol16,3,3)
surf2=translate([0,0,-5],surf1)
sol25=ip_fillet_surf(surf2,flip(sol16),3,3)
sol26=ip_fillet_surf(surf2,flip(sol12),3,-3)
sol31=cpo(ip_fillet_surf(surf2,sol3,1.5,-1.5))[:-1]

```

```
# spoke3
```

```

sec8=corner_radius(pts1([[-2.5,0,2.49],[5,0,2.49],[0,70,2.49],[-5,0,2.49]]),20)
sol6=translate([22,0,178],q_rot(["x90","z90"]),linear_extrude(m_points(sec8,.25),300)))

```

```

sec9=circle(245,s=100)
path9=corner_radius(pts1([[2,0],[-2,0,2],[0,5,5],[-13,20,5],[0,64,5],[-12,20,5],[0,78,5],[12,20,5],[0,
sol7=prism(sec9,path9)

sol8=ip_fillet(sol7,flip(sol6),3,-3)

sec11=circle(45,s=100)
path11=corner_radius(pts1([[0,175],[5,0,5],[0,65,5],[-25,15,5],[-24.5,0]]),15)
sol9=prism(sec11,path11)
sol10=ip_fillet(sol9,sol6,3,-3)

# Hub

sec11=circle(45,s=100)
path11=corner_radius(pts1([[0,175],[5,0,5],[0,65,5],[-25,15,5],[-24.5,0]]),15)
sol9=prism(sec11,path11)

# Rim

sec9=circle(245,s=100)
path9=corner_radius(pts1([[2,0],[-2,0,2],[0,5,5],[-13,20,5],[0,64,5],[-12,20,5],[0,78,5],[12,20,5],[0,
sol7=prism(sec9,path9)

sec13=circle(250,s=100)
path12=corner_radius(pts1([[-2,0],[2,0,2],[0,5,5],[-13,20,5],[0,64,5],[-12,20,5],[0,78,5],[12,20,5],[0,
sol23=prism(sec13,path12)

with open('trial.scad','w+') as f:
    f.write(f'''
        include<dependencies2.scad>

        // spoke 1

        for(i=[0:60:300])rotate([0,0,i])
        difference() {{
            difference() {{
                intersection() {{
                    {swp(surf_extrude(surf_extrude(sec,path),-5))}

                    {swp(sol4)}
                    {swp(sol7)}
                }}
                {swp(sol5)}
                {swp(cyl1)}

            }}
            {swp(cpo(ip_fillet_surf(surf3,cyl1,2,-2))[:-1])}
            {swp(cpo(ip_fillet_surf(surf3,sol5,2,-2))[:-1])}
        }}
    ''')

    }

    for(i=[0:60:360-60])
        rotate([0,0,i])
    intersection() {{
        {swp_c(flip(sol19))}
        {swp(sol7)}
    }}

    for(i=[0:60:360-60])
        rotate([0,0,i])
    {swp_c(sol15)}

    for(i=[0:60:360-60])
        rotate([0,0,i])
    intersection() {{
        {swp_c(sol27)}
        {swp(sol7)}
    }}

    for(i=[0:60:360-60])
        rotate([0,0,i])

```

```

{swp_c(sol28)}

for(i=[0:60:300])rotate([0,0,i])
intersection(){{
{swp_c(bead2)}
{swp(sol7)}
} }

for(i=[0:60:300])rotate([0,0,i]){{{
{swp_c(bead3)}
} }

// spoke 2

{{for(i=[0:60:300])rotate([0,0,i])
difference(){{{
intersection(){{{
{swp(surf1t)}
{swp(sol1)}
{swp(sol7)}
} }
{swp(sol2)}
{swp(flip(sol31))}

}}}
}}}

for(i=[0:60:360-60])
rotate([0,0,i]){{{
intersection(){{{
{swp(bead1)}
{swp(sol7)}
} }

{swp_c(b1_fillet_1)}
{swp_c(b1_fillet_2)}
} }

for(i=[0:60:360-60])
rotate([0,0,i]){{{
intersection(){{{
{swp_c(sol25)}
{swp(sol7)}
} }
{swp_c(sol26)}
} }

for(i=[0:60:360-60])
rotate([0,0,i])
{swp_c(sol13)}

for(i=[0:60:360-60])
rotate([0,0,i])
intersection(){{{
{swp_c(flip(sol17))}
{swp(sol7)}
} }

// spoke 3
for(i=[0:30:360-30])
rotate([0,0,i])
intersection(){{{
{swp(sol6)}
{swp(sol7)}
} }

for(i=[0:30:360-30])
rotate([0,0,i]){{{
{swp_c(sol8)}
{swp_c(sol10)}
} }

```

```

// hub
difference(){{
{sweep(sol9)}
{sweep(cylinder(r=20,h=300,s=72))}}
}}
```

// rim

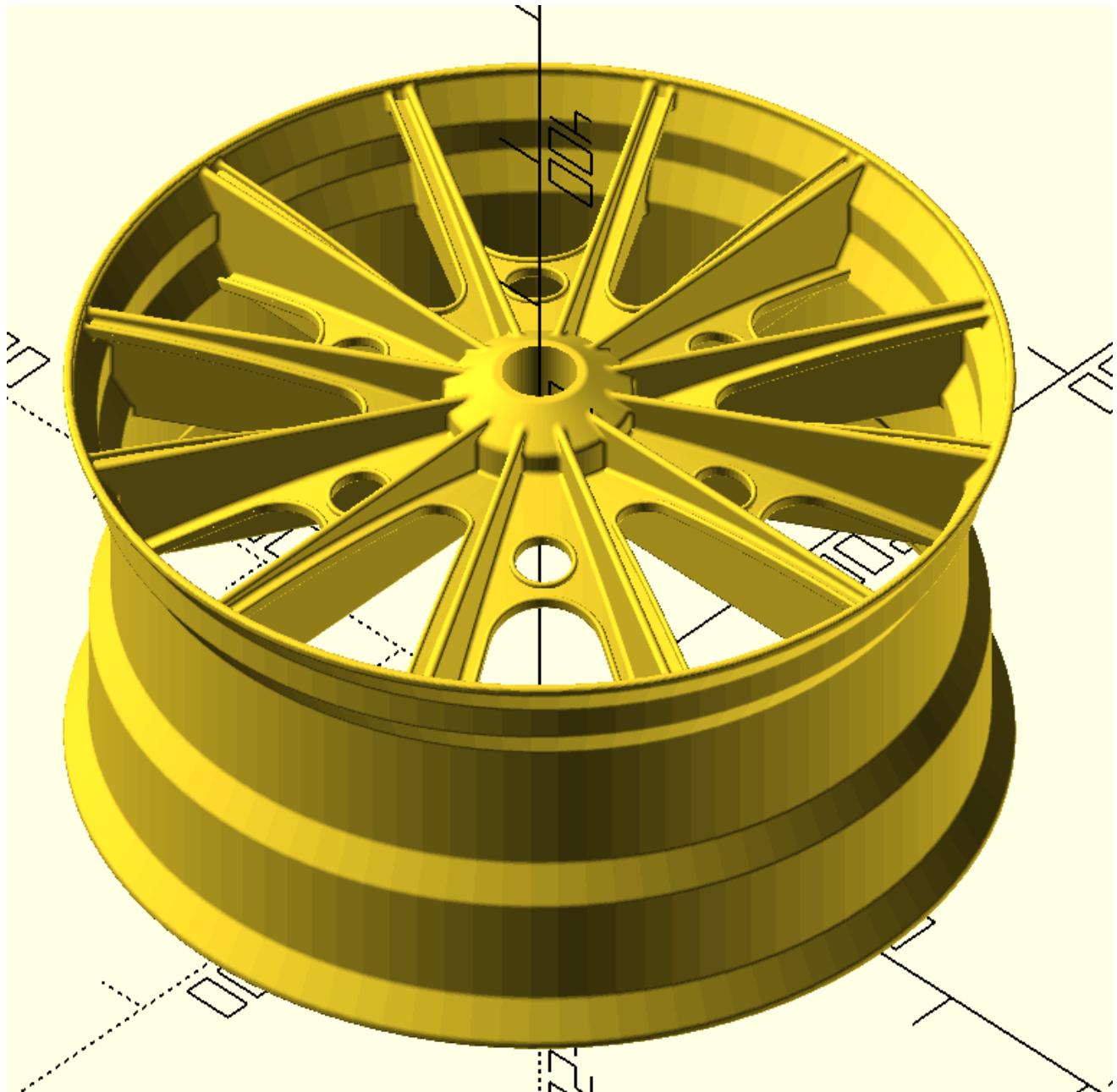
```
{sweep_c(swp_prism_h(sol23,sol7))}
```

'''

f_t=time.time()

f_t-i_t

Out[4]: 11.850472927093506



cylinder-with-rectangular-pocket

```

In [103...]: # cylinder with rectangular pocket
t=2 #thickness
n=100 # number of segments in the circle
s=[15,25] #size of the pocket
cir=circle(20,s=n)
path=corner_radius(pts1([[t/2,0,t/2],[0,80,t/2-.001],[-t,0,t/2-.001],[0,-80,t/2]]),20)
sol1=prism(cir,path)
sol1=sol1+[sol1[0]]
```

```

sec1=corner_radius(pts1([[-s[1]/2,-s[0]/2,t],[s[1],0,t],[0,s[0],t],[-s[1],0,t]]),10)
sol2=translate([0,0,40],q_rot(["y90"],linear_extrude(m_points(sec1,.51),40)))

fillet1=cpo(ip_fillet(sol1,flip(sol2),t/2,-t/2))[:-1]
fillet2=cpo(ip_fillet(flip(sol1),sol2,t/2,-t/2))[:-1]
sol3=flip([translate([5,0,0],fillet1[0])]+fillet1+flip(fillet2)+[translate([-5,0,0],fillet1[0])])

with open('trial.scad','w+') as f:
    f.write(f''''
include<dependencies2.scad>

difference() {{
    {swp_c(sol1)}
    {swp(sol3)}
}}

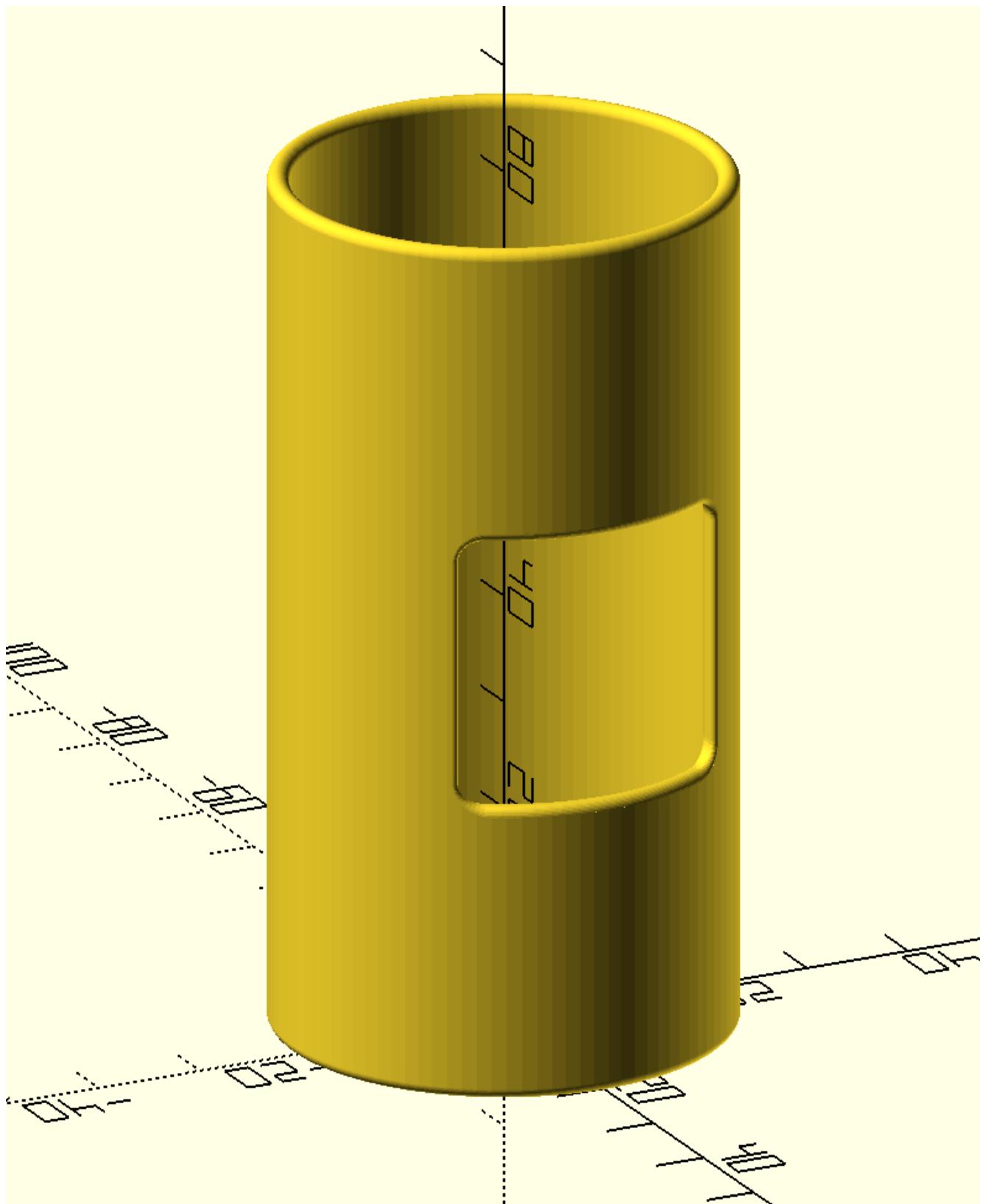

...''')

```

```

/Users/sanjeevprabhakar/iCloud Drive (Archive)/Documents/Download/python/openscad1.py:4629: RuntimeWarning: divide by zero encountered in divide
    t=einsum('kl,ijkl->ijk',cross(p01,p02),la[:, :,None]-p0)/(einsum('ijl,kl->ijk',(-lab),cross(p01,p02))+.00000)
/Users/sanjeevprabhakar/iCloud Drive (Archive)/Documents/Download/python/openscad1.py:4629: RuntimeWarning: invalid value encountered in divide
    t=einsum('kl,ijkl->ijk',cross(p01,p02),la[:, :,None]-p0)/(einsum('ijl,kl->ijk',(-lab),cross(p01,p02))+.00000)
/Users/sanjeevprabhakar/iCloud Drive (Archive)/Documents/Download/python/openscad1.py:4630: RuntimeWarning: divide by zero encountered in divide
    u=einsum('ijkl,ijkl->ijk',cross(p02[None,None,:,:],(-lab)[:, :,None,:]),(la[:, :,None,:]-p0[None,None,:,:]))/(einsum('ijl,kl->ijk',(-lab),cross(p01,p02))+.00000)
/Users/sanjeevprabhakar/iCloud Drive (Archive)/Documents/Download/python/openscad1.py:4630: RuntimeWarning: invalid value encountered in divide
    u=einsum('ijkl,ijkl->ijk',cross(p02[None,None,:,:],(-lab)[:, :,None,:]),(la[:, :,None,:]-p0[None,None,:,:]))/(einsum('ijl,kl->ijk',(-lab),cross(p01,p02))+.00000)
/Users/sanjeevprabhakar/iCloud Drive (Archive)/Documents/Download/python/openscad1.py:4631: RuntimeWarning: divide by zero encountered in divide
    v=einsum('ijkl,ijkl->ijk',cross((-lab)[:, :,None,:],p01[None,None,:,:]),(la[:, :,None,:]-p0[None,None,:,:]))/(einsum('ijl,kl->ijk',(-lab),cross(p01,p02))+.00000)
/Users/sanjeevprabhakar/iCloud Drive (Archive)/Documents/Download/python/openscad1.py:4632: RuntimeWarning: invalid value encountered in add
    condition=(t>=0)&(t<=1)&(u>=0)&(u<=1)&(v>=0)&(v<=1)&(u+v<1)
/Users/sanjeevprabhakar/iCloud Drive (Archive)/Documents/Download/python/openscad1.py:4634: RuntimeWarning: invalid value encountered in multiply
    a=(la[:,None,:,:]+lab[:,None,:,:]*t[:,None,:,:])

```



cylinder-with-star-pocket

In [104...]

```
# cylinder with star pocket

t=2 # thickness
n=100 # number of segments of circle
s=5 # number of sides of the star
d=20 # outer diameter of the star
h=50 # height of the star prism
cir1=circle(d,s=(s+1))
cir2=c3t2(q_rot([f"z{360/(s+1)/2}"],circle(d/4,s=(s+1))))
sec1=array(c2t3([cir1,cir2]))
sec1=sec1.transpose(1,0,2).reshape(-1,3)
sec1=sec1
sec1=[(sec1[i]+[0,0,t/1.5]).tolist() if i%2==0 else (sec1[i]+[0,0,t]).tolist() for i in range(len(sec1))]
sec1=corner_radius(sec1,20)
sol1=translate([0,0,40],q_rot(["y90"],linear_extrude(m_points(sec1,1.1),h)))
```

```

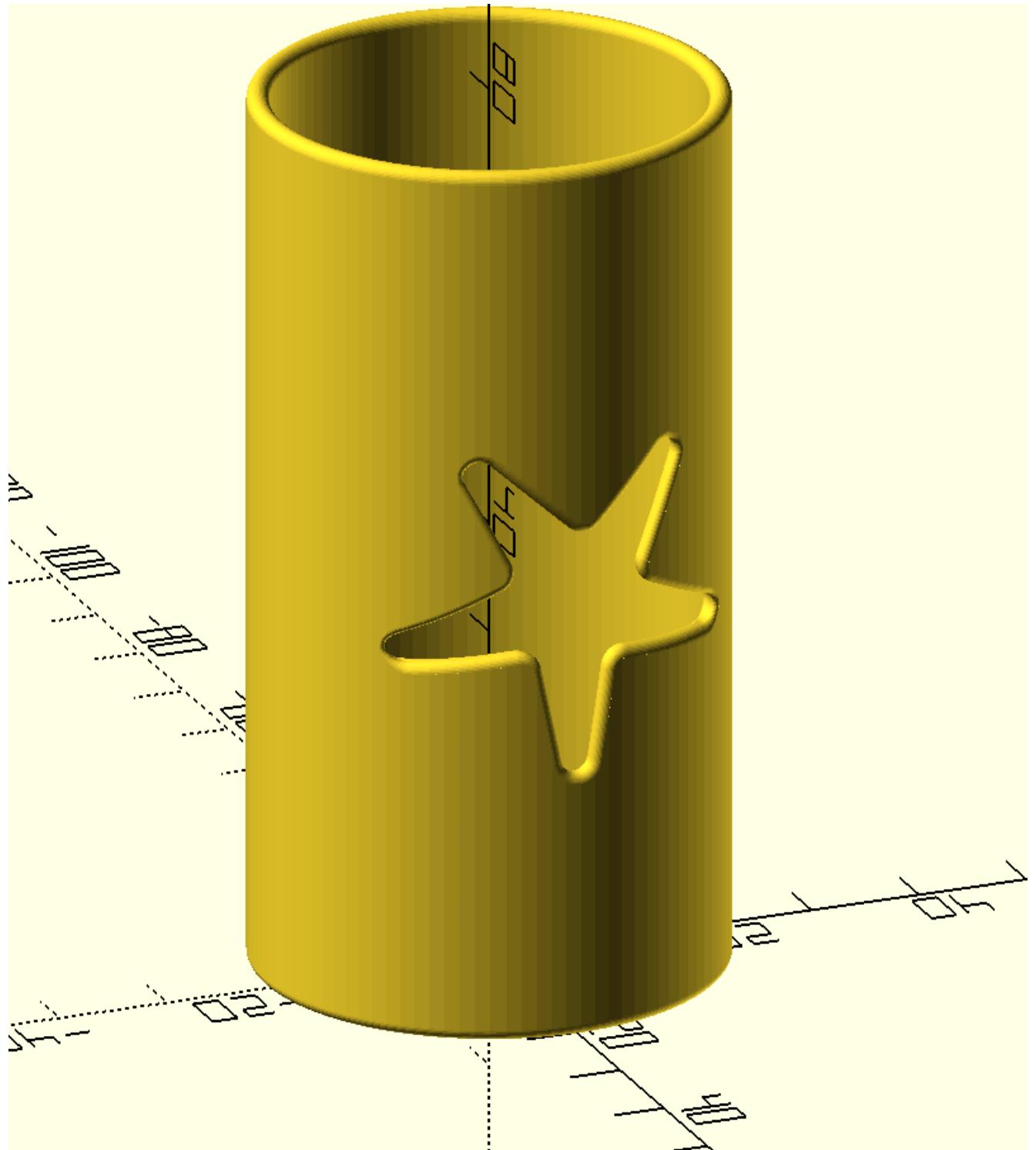
cir=circle(20,s=n)
path=corner_radius(pts1([[t/2,0,t/2],[0,80,t/2-.001],[-t,0,t/2-.001],[0,-80,t/2]]),10)
sol2=prism(cir,path)
sol2=sol2+[sol2[0]]

fillet1=cpo(ip_fillet(sol2,flip(sol1),t/2,-t/2))[:-1]
fillet2=cpo(ip_fillet(flip(sol2),sol1,t/2,-t/2))[:-1]
sol3=flip([translate([5,0,0],fillet1[0])]+fillet1+flip(fillet2)+[translate([-5,0,0],fillet1[0])])

with open('trial.scad','w+') as f:
    f.write(f'''  

    include<dependencies2.scad>  
  

difference(){{
{swp_c(sol2)}
{swp(sol3)}
}}
...)
```



lamp

In [105...]

```
# lamp

n=20 #number of strands in the lamp
t=5 # thickness of each strand
path=[[42,0],[62,50],[25,100],[25,180]]

path1=cytz(bezier(path,100))

path2=[q([0,0,1],path1[i],i/(len(path1)-1)*(180+1.8)) for i in range(len(path1)-1)]

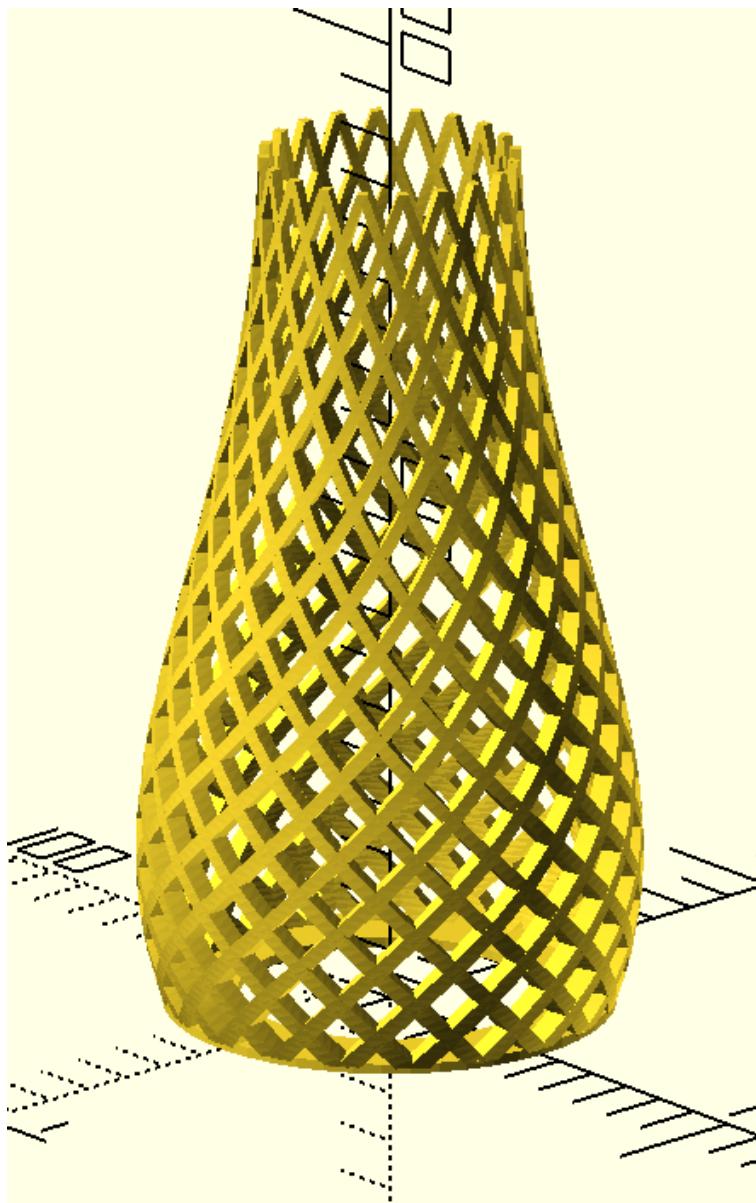
sol=[]
for i in range(len(path2)):
    theta=ang(path2[i][0],path2[i][1])
    sol.append(translate(path2[i],q_rot([f'z{theta}'], \
        offset(square(t,center=True),i/len(path2)*-1.5))))
sol=q_rot(['z0'],sol)

c1=circle(45)
c2=circle(30)

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>
for(i=[0:360/{n}:359])
rotate([0,0,i])
{{{
{swp(sol)}
//p_line3d({path2},.5);
mirror([0,1,0])
{swp(sol)}
//p_line3d({path2},.5);

}}}
linear_extrude(2)
difference()
{{{
polygon({c1});
polygon({c2});

}}}
''' )
```



samsung-tab-s6-holder

In [106...]

```
# samsung tab s6 holder to hang in car back seat

sec=corner_radius(pts1([[0,0,.1],[124,0,3],27*cos(45*pi/180),27*sin(45*pi/180),1],
[15*cos(135*pi/180),15*sin(135*pi/180),1],
[5*cos((180+45)*pi/180),5*sin((180+45)*pi/180),1],
[10*cos(-45*pi/180),10*sin(-45*pi/180),1],
[17*cos((180+45)*pi/180),17*sin((180+45)*pi/180),1],
[10*cos(135*pi/180),10*sin(135*pi/180),1],
[8*cos(225*pi/180),8*sin(225*pi/180),1],[20*cos(135*pi/180),20*sin(135*pi/180),1],
[8*cos(45*pi/180),8*sin(45*pi/180),1],
[105*cos(135*pi/180),105*sin(135*pi/180),1],
[8*cos(225*pi/180),8*sin(225*pi/180),1],
[20*cos(135*pi/180),20*sin(135*pi/180),1],
[8*cos(45*pi/180),8*sin(45*pi/180),1],[10*cos(135*pi/180),10*sin(135*pi/180),1],
[17*cos(45*pi/180),17*sin(45*pi/180),1],[10*cos(-45*pi/180),10*sin(-45*pi/180),1],
[5*cos(45*pi/180),5*sin(45*pi/180),1],[15*cos(135*pi/180),15*sin(135*pi/180),1],
[21*cos(225*pi/180),21*sin(225*pi/180),1],[0,30,2],[-4,0,1]]),10)

sec1=corner_radius(pts1([[0,0,2],[17,0,2],[0,85,2],[-17,17,2]]),5)
path1=[[0,0],[0,6]]
sol=translate([123.5,7.7,30],q_rot(["x90","z45"],prism(sec1,path1)))

sol1=translate([5,129,130],q_rot(["x90","z45"],prism(sec1,path1)))

sec2=corner_radius(pts1([[0,0,4],[10,0,4],[0,40,4],[-10,0,4]]),5)
path2=[[0,0],[0,8]]
sol2=translate([-0.25,140,40],q_rot(["x90","z90"],prism(sec2,path2)))
```

```

sol3=translate([-0.25,140,170],q_rot(["x90","z90"],prism(sec2,path2)))

sec4=corner_radius(pts1([[0,0,0],[15,0,5],[0,90,5],[-15,0,0]]),5)
path4=[[0,0],[0,3]]
sol4=translate([90,35,0],q_rot(["z45"],prism(sec4,path4)))

sec5=corner_radius(pts1([[0,0],[120,0,0],[32*cos(135*pi/180),32*sin(135*pi/180),1],
[8*cos(45*pi/180),8*sin(45*pi/180),1],
[105*cos(135*pi/180),105*sin(135*pi/180),1],[8*cos(225*pi/180),8*sin(225*pi/180),1],
[33*cos(135*pi/180),33*sin(135*pi/180),0]]),5)

sec6=corner_radius(pts1([[0,0,5],[90,0,5],[0,110,5],[-90,90,5]]),5)
path6=[[0,0],[0,7]]

sol6=translate([15,6,25],q_rot(["x90"],prism(sec6,path6)));
sol7=translate([-1,15,25],q_rot(["x90","z90"],prism(sec6,path6)));
sol8=translate([95,30,25],q_rot(["x90","z135"],prism(sec6,path6)));

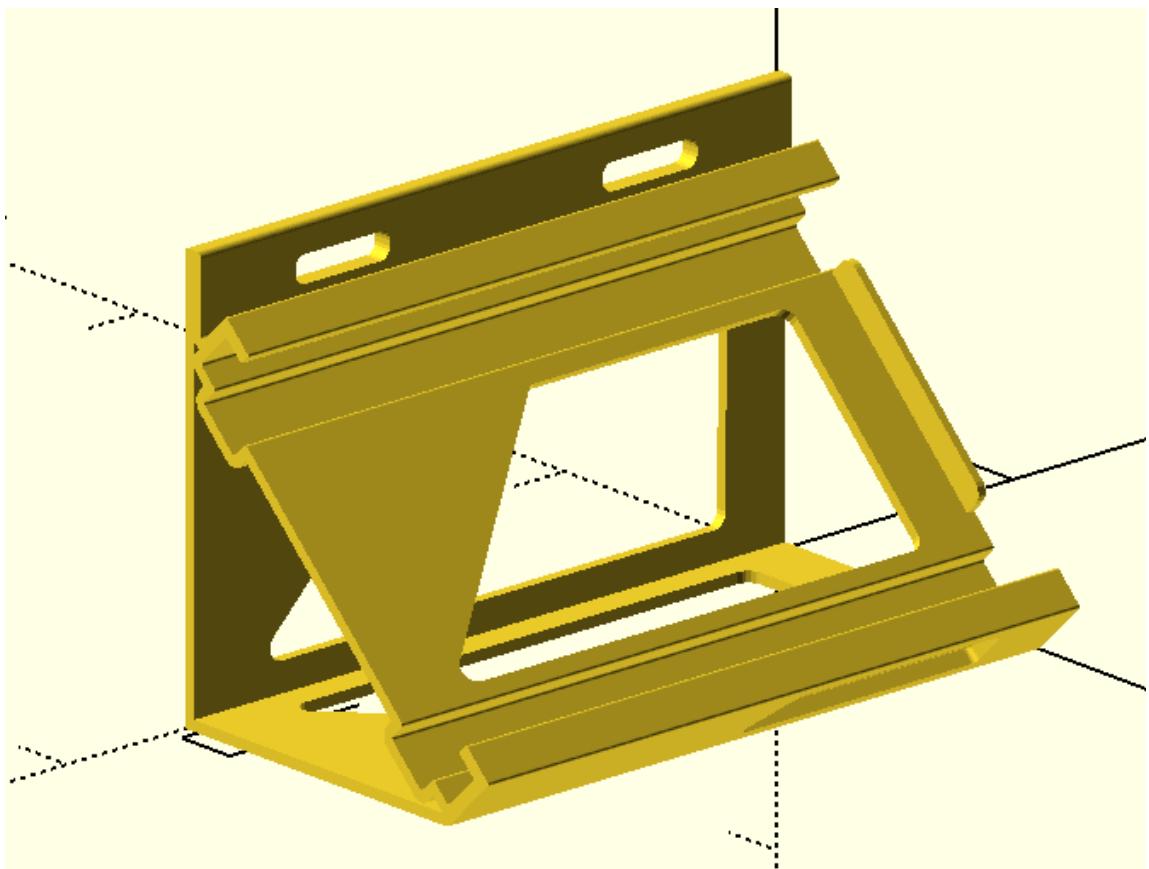
with open('trial.scad','w+') as f:
    f.write('''
include<dependencies2.scad>
rotate([90,0,0])
{{
difference(){{
difference(){{
linear_extrude(250
polygon({sec});

translate([0,0,-.05])
linear_extrude(250.1)
polygon({offset(sec,-3.95)}});
}}}

//color("blue")
{swp(sol)}
//color("blue")
{swp(sol1)}
//color("blue")
{swp(sol2)}
//color("blue")
{swp(sol3)}
//color("blue")
{swp(sol6)}
//color("blue")
{swp(sol7)}
//color("blue")
{swp(sol8)}
}}


//color("magenta")
{swp(sol4)}
}}
''' )

```



business-card-holder

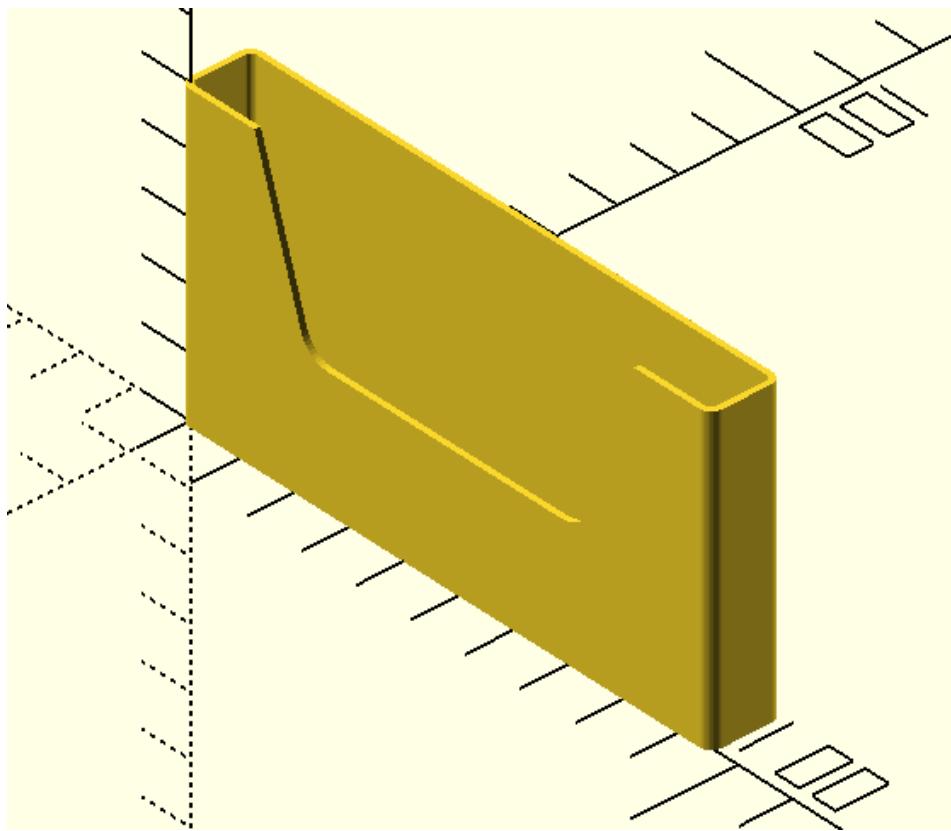
```
In [107]: # business card holder

sec=corner_radius(pts1([[0,0,1],[95,0,1],[0,10,1],[-95,0,1]]),10)
sol2=linear_extrude(offset(sec,1),1);
sec1=corner_radius(pts1([[10,0,5],[50,0,5],[10,30],[-70,0]]),10);
sol3=translate([12.5,2,21],q_rot(["x90"],linear_extrude(sec1,5)));

# difference(){
# union(){
# sol4=swp_prism_h(prism1,prism)
# swp(prism2);}
# swp(prism3);}

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>
render(){
difference(){
linear_extrude(50)
difference(){
polygon({offset(sec,1)});
polygon({sec});
}};

{swp(sol3)}
}
{swp(sol2)}
})
''' )
```



m10

```
In [108...]: p0,p1,p6,p7=tctpf(7.5,17.5,[14.5,0],[0,0])
p2,p3,p4,p5=tctpf(17.5,7.5,[0,0],[-14.5,0])
sec1=arc_2p(p1,p2,17.5,-1)+arc_2p(p3,p4,7.5,-1)+arc_2p(p5,p6,17.5,-1)+arc_2p(p7,p0,7.5,-1)

path1=corner_radius(pts1([[-17,0],[17,0],[0,5.5],[-17,0]]))
sol1=prism(sec1,path1)

sec2=circle(10)
path2=cr_3d([[0,0,0.001,0],[0,0,29-.001,17.5],[-44+.001,0,0,0]],20)

sol2=align_sol_1(path_extrude_open(sec2,path2))

sec3=circle(39/2)
path3=corner_radius(pts1([[-39/2+.5,0],[39/2-.5,0],[0,4],[-39/2+.5,0]]),10)
sol3=prism(sec3,path3)
sol3=sol2vector([1,0,0],sol3,[-44,0,29])

fillet1=ip_fillet(sol1,sol2,1.5,-1.5)
fillet2=ip_fillet(sol3,flip(sol2),1.5,-1.5)

sol4=align_sol_1(path_extrude_open(circle(13/2),path2))
sol4=swp_prism_h(sol2,sol4)

sol5=translate([0,0,-.25],linear_extrude(circle(10-.5),6))
sol6=sol2vector([1,0,0],sol5,[-44,0,29])
sol7=translate([0,0,-.25],linear_extrude(circle(2.5,[15,0]),5.5))
sol7=[q_rot([f'z{i}'],sol7) for i in [0,90,180,270]]
sol7=[sol2vector([1,0,0],p,[-44,0,29]) for p in sol7]

sol8=translate([0,0,-.25],linear_extrude(circle(3,[14.5,0]),6))
sol8=[q_rot([f'z{i}'],sol8) for i in [0,180]]

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>

//color("blue")p_line({sec1},.1);
sol8={sol8};
difference() {{
{swp(sol1)}
{swp(sol5)}
for(i=[0,1])swp(sol8[i]);
}}
```

```

    }
    {swp_c(sol4)}
sol7={sol7};
difference()|{
{swp(sol3)}
{swp(sol6)}
for(i=[0,1,2,3])swp(sol7[i]);
}}
{swp_c(fillet1)}
{swp_c(fillet2)}

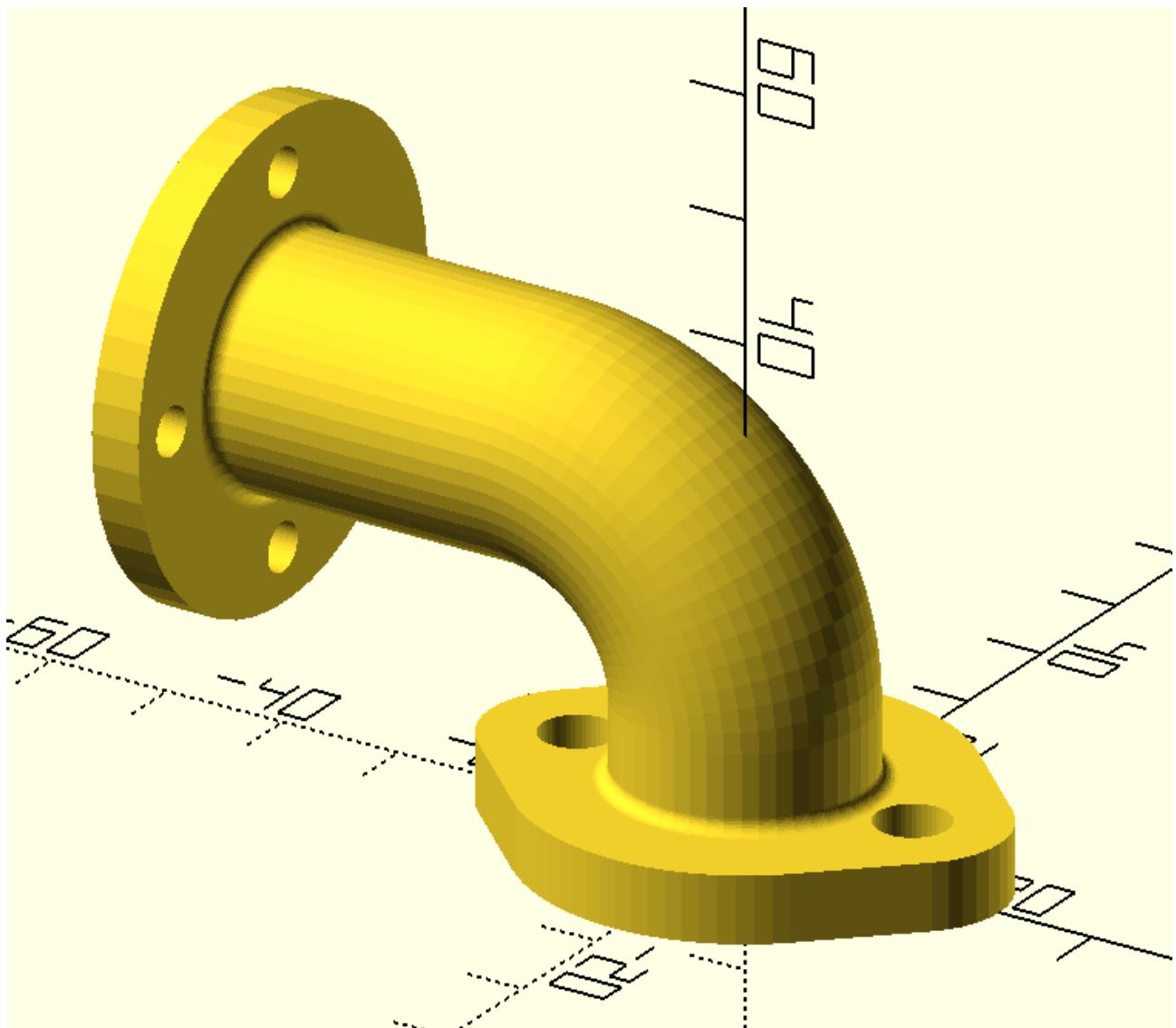
...

```

```

/Users/sanjeevprabhakar/iCloud Drive (Archive)/Documents/Download/python/openscad1.py:4631: RuntimeWarning: invalid value encountered in divide
    v=einsum('ijkl,ijkl->ijk',cross((-lab)[:, :, None, :],p01[None,None,:,:]),(la[:, :, None, :]-p0[None,None,:,:]))/(einsum('ijl,kl->ijk',(-lab),cross(p01,p02))+.00000)

```



m35

```

In [109...]: # m35
t0=time.time()

p0=[15,0]
p1=[15,45]
cir1=circle(7.5,[0,75-7.5])
p2=p_cir_t(p1,cir1)
p4=[-15,45]
p3=cir_p_t(cir1,p4)
arc1=arc_2p(p2,p3,7.5,-1,45)
p5=[-15,0]
p0,p1,arc1,p4,p5=[[15,0,0]],[[15,45,1]],c2t3(arc1),[[ -15,45,1]],[[-15,0,0]]
sec=corner_radius(p0+p1+arc1+p4+p5,10)
sec=equidistant_pathc(sec,1000)

```

```

l1=equidistant_path([[0,5],[0,70]],200)
l1=sort_points(sec,l1)
sec1=translate([0,.1,0],slice_sol([l1,sec],10))

path=corner_radius(pts1([[0,0],[45,0,1],[45*cos(d2r(30)),45*sin(d2r(30))]]),30)
path=q_rot(['x90','z90'],path)

surf1=flip([wrap_around(p,path) for p in sec1])
surf2=surf_offset(surf1,-4)
sol=translate([0,0,9+4],q_rot(['z-90'],flip(surf2)+surf1))

sec2=[circle(.5,[0,75-7.5]),circle(3.75,[0,75-7.5])]
surf3=flip([wrap_around(p,translate([0,0,.001],path)) for p in sec2])
surf4=surf_offset(surf3,-4.1)
sol1=translate([0,0,9+4],q_rot(['z-90'],flip(surf3)+surf4))

path=cytz(pts([[0,9],[25.5+19.5,0],[(30-7.5)*cos(45*pi/180),(30-7.5)*sin(45*pi/180)]]))
# sec1=c2t3(m_points_o(pts([-15,0],[30,0],[0,5],[-30,0])),1)

sec3=[[-30,-15]]+arc_2p([0,-15],[0,15],15,-1,20)+[[-30,15]]
p6=l_cir_ip([-30,-15+10.5],[0,-15+10.5],circle(7.5))
p7=l_cir_ip([-30,15-10.5],[0,15-10.5],circle(7.5))
sec4=[[-30,-15+10.5]]+arc_long_2p(p6[0],p7[0],7.5,-1,30)+[[-30,15-10.5]]

sec5=circle(6)
path1=corner_radius(pts1([[0,-15.1],[0,4.6],[-3,0],[0,21],[3,0],[0,4.6]]),5)
sol2=translate([-30+12,0,11.25],q_rot(['x-90'],prism(sec5,path1)))

sec6=[[25.5+9,-15.1]]+arc_2p([25.5+9,0],[25.5,0],4.5,-1)+[[25.5,-15.1]]
with open('trial.scad','w+') as f:
    f.write(f'''')
include<dependencies2.scad>

difference(){
{swp(sol)}
linear_extrude(22.5)polygon({sec4});
{swp(flip(sol1))}

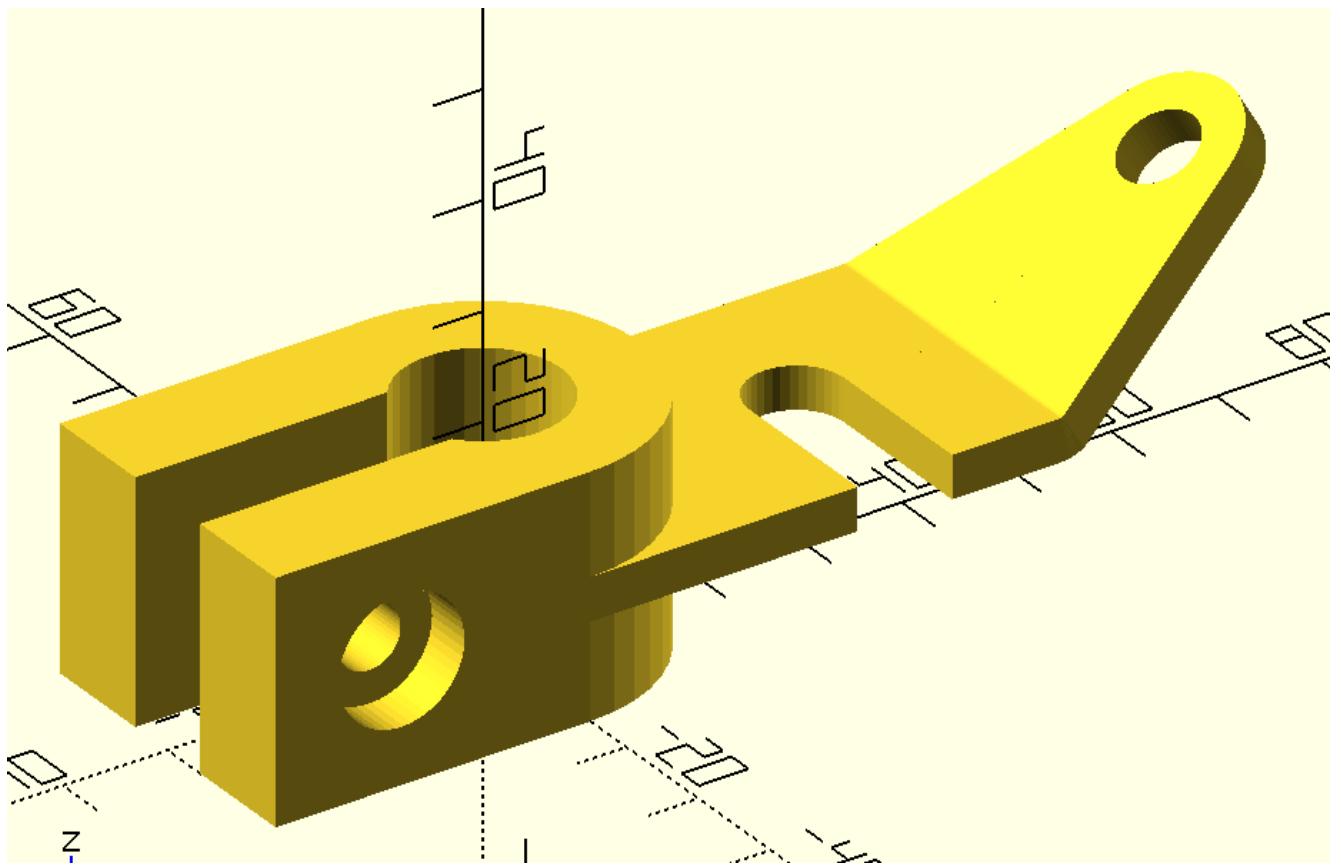
linear_extrude(22.5)polygon({sec6});

}
difference(){
linear_extrude(22.5)
difference(){
polygon({sec3});
polygon({sec4});
}}
{swp(sol2)}
}

...)
t1=time.time()
t1-t0

```

Out[109... 2.0842528343200684



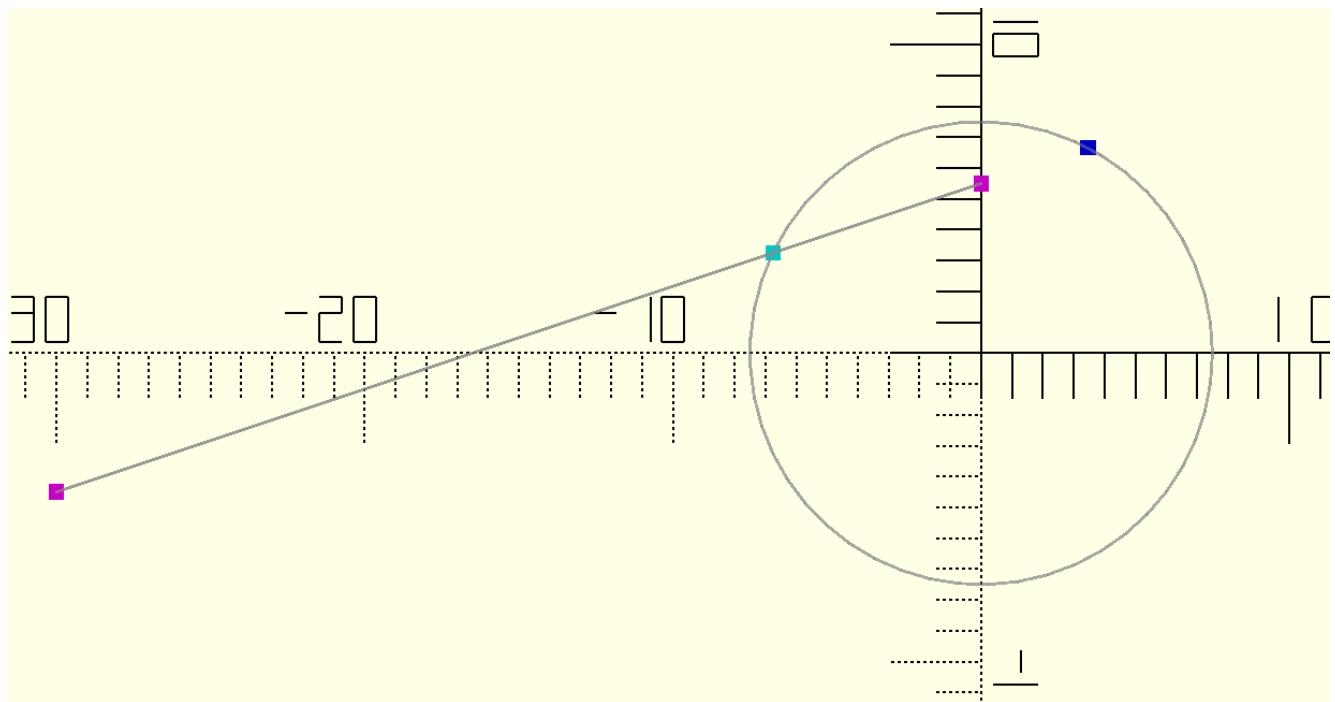
l_cir_ip

```
In [110]: # example of function l_cir_ip(line, cir)
p0=[-30,-15+10.5]
p1=[0,-15+10.5+10]
line=[p0,p1]
cir=circle(7.5)
p2=l_cir_ip(line,cir)

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>

%p_line({{cir}},.1);
%p_line({{[p0,p1]}},.1);
color("magenta")points({{[p0,p1]}},.5);
color("cyan")points({{[p2[0]]}}},.5);
color("blue")points({{[p2[1]]}}},.5);

...)
```



fillet_l_cir

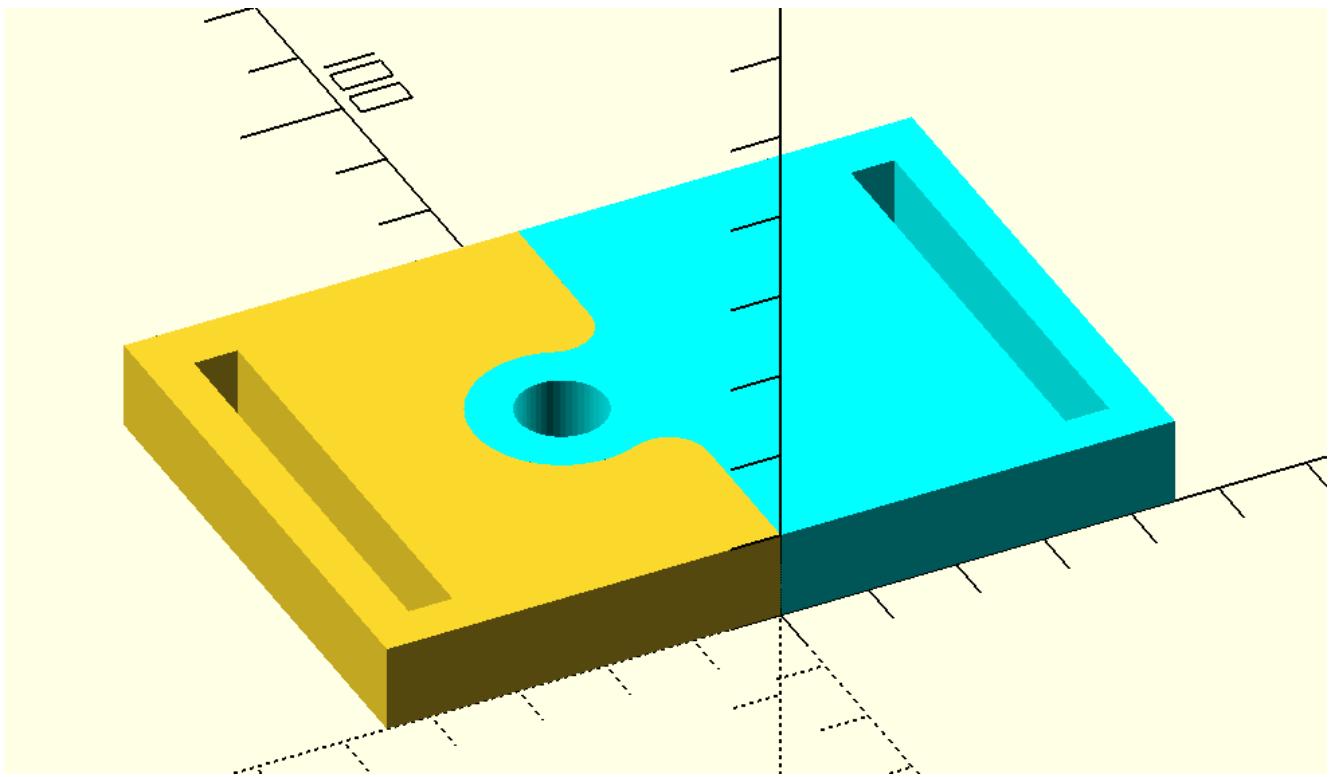
```
In [111]: # example of function fillet_l_cir(line,cir1,fillet_radius,s)

line=[[0,0],[0,60]]
cir1=circle(10,[-10,30])
cir2=circle(5,[-10,30])
fillet1=fillet_l_cir(line,cir1,4.5,10)[0]
fillet2=flip(fillet_l_cir(line,cir1,4.5,10)[1])
p0,p1,p2,p3,p4,p5,p6,p7,p8,p9=line[0],fillet1[0],fillet1[-1],fillet2[0],fillet2[-1],line[1],[-45,60],[
sec1=[p0]+fillet1[:-1]+arc_long_2p(p2,p3,10,1,50)+fillet2[1:]+[p5]+[p6]+[p7]
sec2=[p0]+fillet1[:-1]+arc_long_2p(p2,p3,10,1,50)+fillet2[1:]+[p5]+[p8]+[p9]
sec3=c3t2(translate([-45+5,5,0],square([5,49])))
sec4=c3t2(translate([45-10,5,0],square([5,49])))

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>

linear_extrude(10)
difference(){{{
polygon({sec1});
polygon({sec3});
}}}

color("cyan")
linear_extrude(10)
difference(){{{
polygon({sec2});
polygon({cir2});
polygon({sec4});
}}}
''' )
```



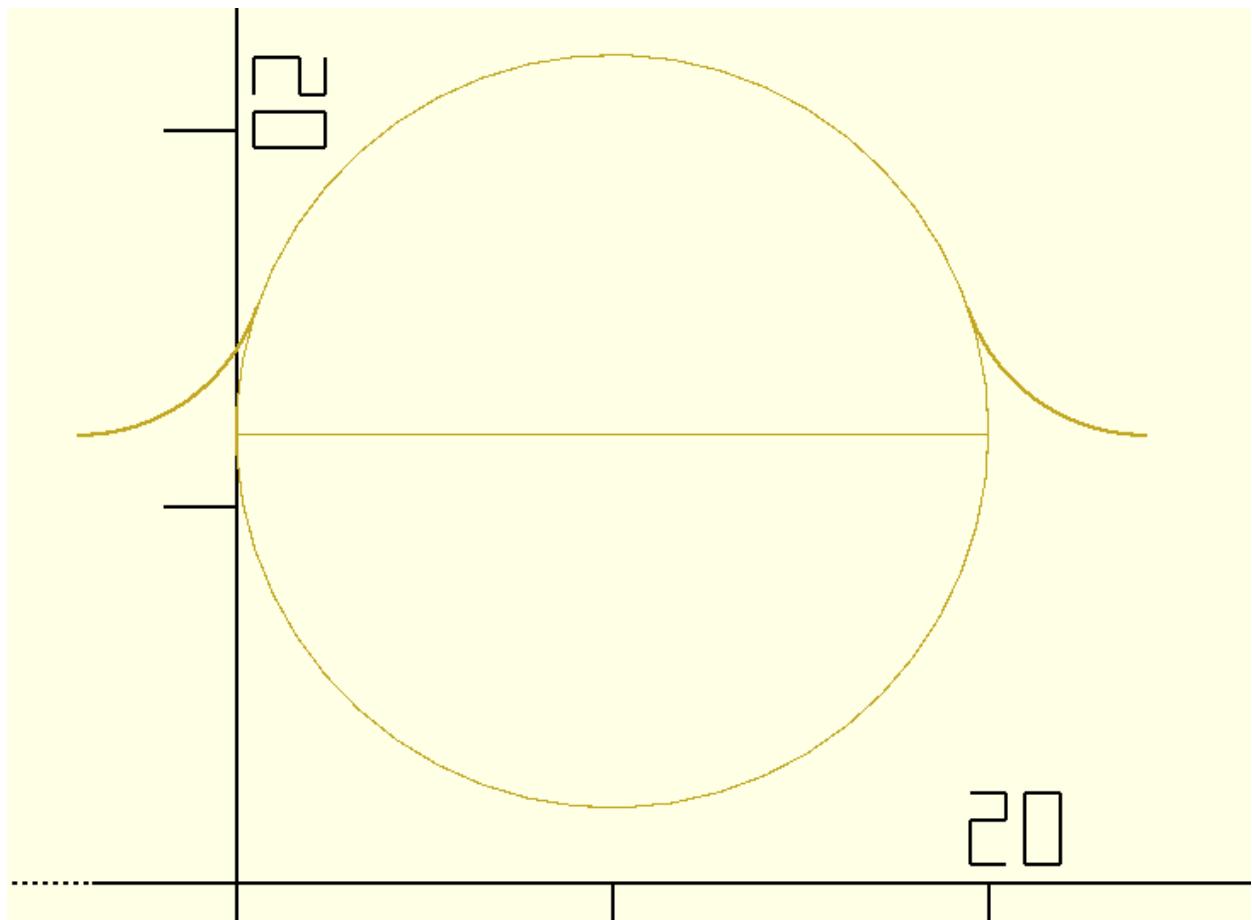
```
In [112]: # example of function fillet_l_cir(line,circle,fillet_radius,s)
```

```
line=[[-10,13],[30,13]]
cir1=circle(10,[10,12])
r2=5
s=20

with open('trial.scad','w+') as f:
    f.write(f'''  
include<dependencies2.scad>

color("blue")p_line({line},.05);
p_line({cir1},.05);
color("cyan")p_lineo({fillet_l_cir(line,cir1,r2,s)[0]},.1);
color("cyan")p_lineo({fillet_l_cir(line,cir1,r2,s)[1]},.1);

...)
```

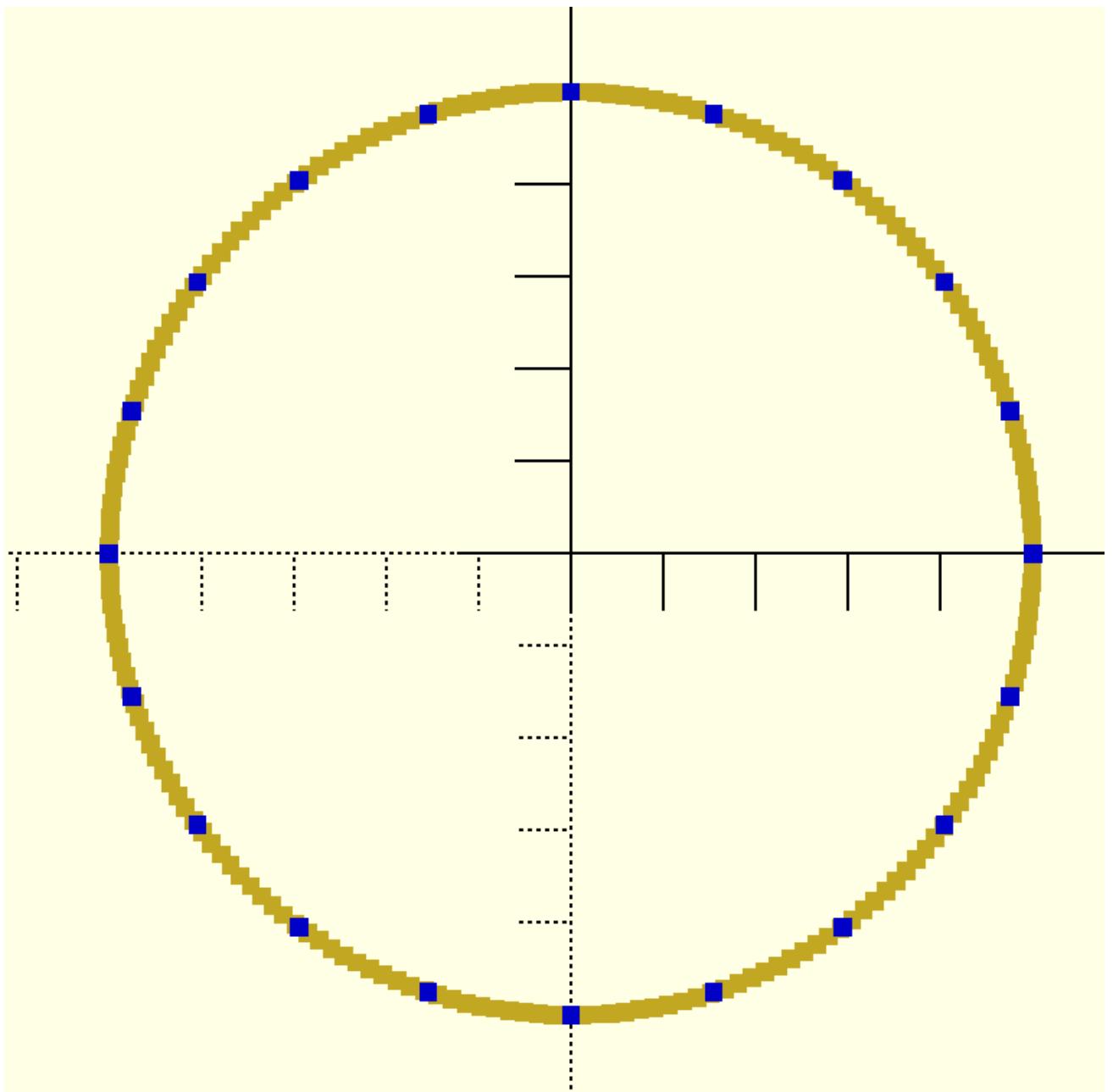


equidistant_pathc

```
In [113]: a=circle(5,s=200)

with open('trial.scad','w+') as f:
    f.write(f''''
include<dependencies2.scad>
points({a},.2);
color("blue")points({equidistant_pathc(a,20)},.2);

...)
```



artifact

In [5]: *# it takes around 60 sec for calculations*

```
t0=time.time()  
stages =200  
stage_height = 1.25  
rad = 50  
f1 = 25  
f2 = 25  
phase1 = 0  
phase2 = 180  
height_depth=5  
depth1 = 20  
depth2 = 20  
thickness = 2
```

```

bottom_thickness = 3

myslices = 5

angle_step=.5

var=1

def pauw(x,p):
    return sign(x)*abs(x)**p

def f(i,stages):
    return sin(d2r(i/stages * 120))**2 * 7 + 1

def a(i,stages,var,height_depth):
    return (sin(d2r((i/stages*360*f(i,stages))%360)) * 0.5 + 0.5) * (var * height_depth)

# generate outer points

# points_base = [for (i = [0:1:stages])
#                 let(f = sin(i/stages * 120)^2 * 7 + 1, var = 1 , a=((sin((i/stages*360*f)%360) * 0.5
#                 [for(j = [0:angle_step:360-angle_step])
#                     [sin(j) * (rad+a+(pauw(sin(j *f1+phase1),0.5)*0.5+0.5)*depth1*i/stages+(pauw(sin(j *
#                     cos(j) *(rad+a+(pauw(sin(j *f1+phase1),0.5)*0.5+0.5)*depth1*i/stages+(pauw(sin(j *
#                     i*stage_height))

points_base=[[[
    [sin(d2r(j)) * (rad+a(i,stages,var,height_depth)+(pauw(sin(d2r(j *f1+phase1)),0.5)*0.5+0.5)*depth1*i*stage_height
    cos(d2r(j)) * (rad+a(i,stages,var,height_depth)+(pauw(sin(d2r(j *f1+phase1)),0.5)*0.5+0.5)*depth1*i*stage_height
] for j in arange(0,360,angle_step)] for i in range(stages+1)]

p1,p2=[[],[]
for i in range(stages):
    points_base1= flip(c3t2(points_base[i]))#flip(c3t2(points_base[i])) if cw(c3t2(points_base[i]))==1
    points_base2=oerset(points_base1,-thickness)
    p1.append(translate([0,0,i*stage_height],points_base1))
    p2.append(translate([0,0,i*stage_height],points_base2))

sol=swp_prism_h(p1,p2)

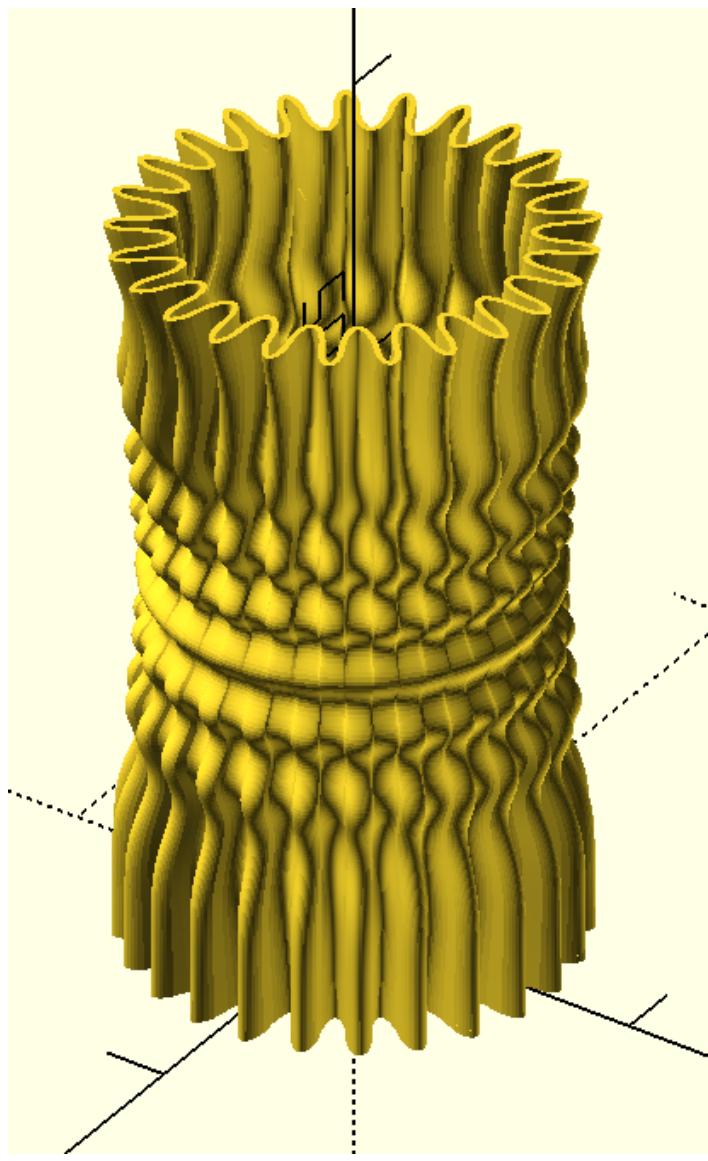
sol1=[p1[0],p1[2]]

with open('trial.scad','w+') as f:
    f.write(f'''
{swp_c(sol)}
{swp(sol1)}
'''')

t1=time.time()
t1-t0

```

Out[5]: 71.87775683403015



wrap_around

```
In [6]: # example of function wrap_around(sec,path)
t0=time.time()

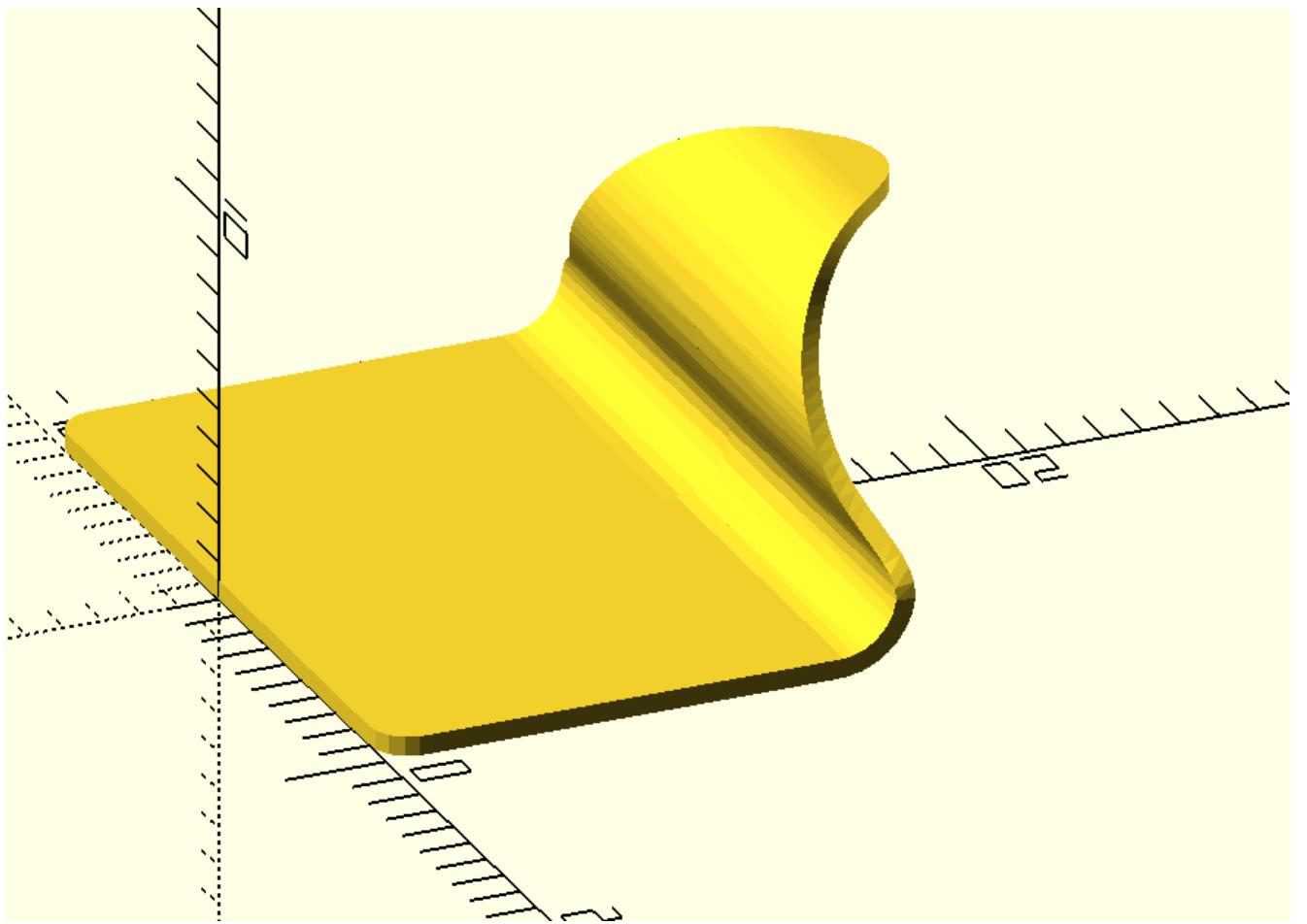
sec=equidistant_pathc(corner_radius(pts1([[-10,0,1],[20,0,1],[0,15,1],[-10,10,2],[-10,-10,1]]),20),200
l1=equidistant_path([[0,2],[0,22]],40)
l1=sort_points(sec,l1)
sol1=translate([0,.1,0],slice_sol([l1]+[sec],20))
path=cr_3d([[0,0,0,0],[0,15,0,2],[0,-4,8,3],[0,10,0,0]],20)

sol2=flip([wrap_around(p,path) for p in sol1])
sol3=surf_offset(sol2,.5)
sol4=flip(sol2)+sol3

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>
{swp_c(sol4)}

...))
t1=time.time()
t1-t0
```

Out[6]: 0.3208591938018799



```
In [110]: # example of function wrap_around(sec,path) work flow

sec=equidistant_pathc(corner_radius(pts1([[ -10,0.1,1],[20,0,1],[0,15,1],[-10,10,2],[-10,-10,1]]),20),20)

path1=corner_radius(pts1([[0,0],[15,0,1],[-16,15]]),50)
path2=circle(5,s=200)
path3=corner_radius(pts1([[0,0],[5,0,1],[0,5,1],[3,0,1],[0,-5,1],[10,0]]),10)
path1=q_rot(['x90','z90'],path1)
path2=q_rot(['x90','z90'],path2)
path3=q_rot(['x90','z90'],path3)

sec1=wrap_around(sec,path1)
sec2=wrap_around(sec,path2)
sec3=wrap_around(sec,path3)

path4=corner_radius(pts1([[6,2],[-6,0,1],[0,-2.001,1],[25,0]]),10)
path4=q_rot(['x90','z90'],path4)

sec4=translate([0,9,0],q_rot(['z-45'],sec1))
sec4=wrap_around(sec4,path4)

sec5=translate([0,9,0],q_rot(['z90'],translate([0,-9,0],sec4)))
sec5=wrap_around(sec5,path4)

sec6=q_rot(['z-45'],translate([0,-9,0],sec5))

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>

translate([30,40,0]){
p_line3d({path2},.1);
color("magenta")p_line3dc({sec2},.1);}

translate([30,60,0]){
p_line3d({path3},.1);
color("magenta")p_line3dc({sec3},.1);}

color("blue")p_line3dc({sec},.1);
translate([0,30,0]){
p_line3d({path4},.1);
color("red")p_line3dc({sec4},.1);
color("green")p_line3dc({sec5},.1);
color("cyan")p_line3dc({sec6},.1);}}})
```

```

color("blue")p_line3d({path1},.1);
color("magenta")p_line3dc({sec1},.1);}

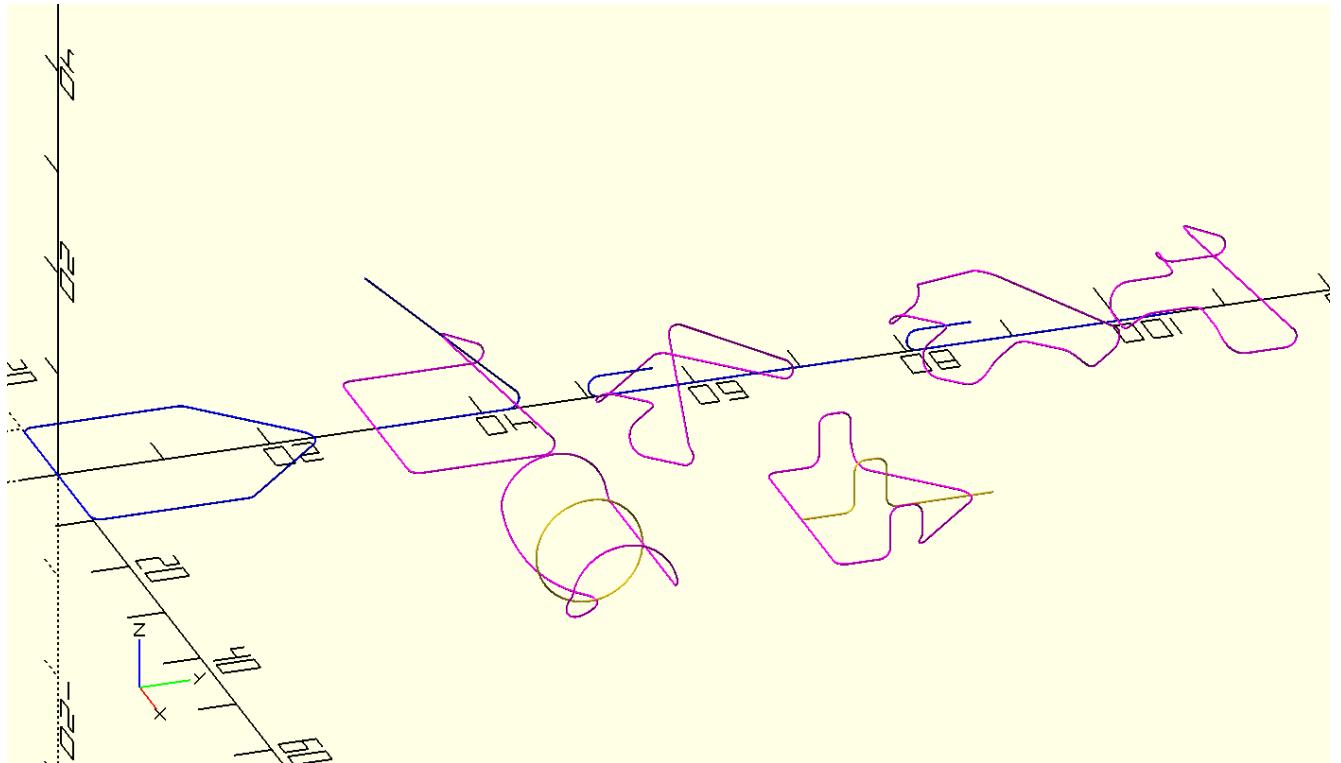
translate([0,50,0]){{}
color("blue")p_line3d({path4},.1);
color("magenta")p_line3dc({sec4},.1);}

translate([0,80,0]){{}
color("blue")p_line3d({path4},.1);
color("magenta")p_line3dc({sec5},.1);}

translate([0,110,0]){{}
//p_line3d({path4},.1);
color("magenta")p_line3dc({sec6},.1);}

...

```



```

In [7]: # wrap_around example 4
t0=time.time()

sec0=corner_radius(pts1([[ -10,0.1,1],[ 20,0,1],[ 0,15,1],[-10,10,2],[-10,-10,1]]),20)
cp=array(offset(sec0,-10)).mean(0).tolist()
sec=equidistant_pathc(corner_radius(pts1([[ -10,0.1,1],[ 20,0,1],[ 0,15,1],[-10,10,2],[-10,-10,1]]),20),2
sec1=cpo([sort_points(sec,[cp]),sec])
sec1=cpo([ls(p,50) for p in sec1])+[sec]

path1=corner_radius(pts1([[ 0,0],[15,0,1],[-16,15]]),50)
path1=q_rot(['x90','z90'],path1)

sec1=[wrap_around(p,path1) for p in sec1]

path4=corner_radius(pts1([[ 7,5],[-7,-5,1],[45,0]]),10)
path4=q_rot(['x90','z90'],path4)

sec2=translate([0,9,0],q_rot(['z-45'],sec1))
sec2=[wrap_around(p,path4) for p in sec2]

sec3=translate([0,9,0],q_rot(['z90'],translate([0,-9,0],sec2)))
sec3=[wrap_around(p,path4) for p in sec3]

sec3=q_rot(['z-45'],translate([0,-9,0],sec3))
sec4=surf_offset(flip(sec3),.5)
sol=sec3+sec4+[sec3[0]]
sol=translate([0,3,0],sol)

with open('trial.scad','w+') as f:

```

```

f.write(f'''
include<dependencies2.scad>

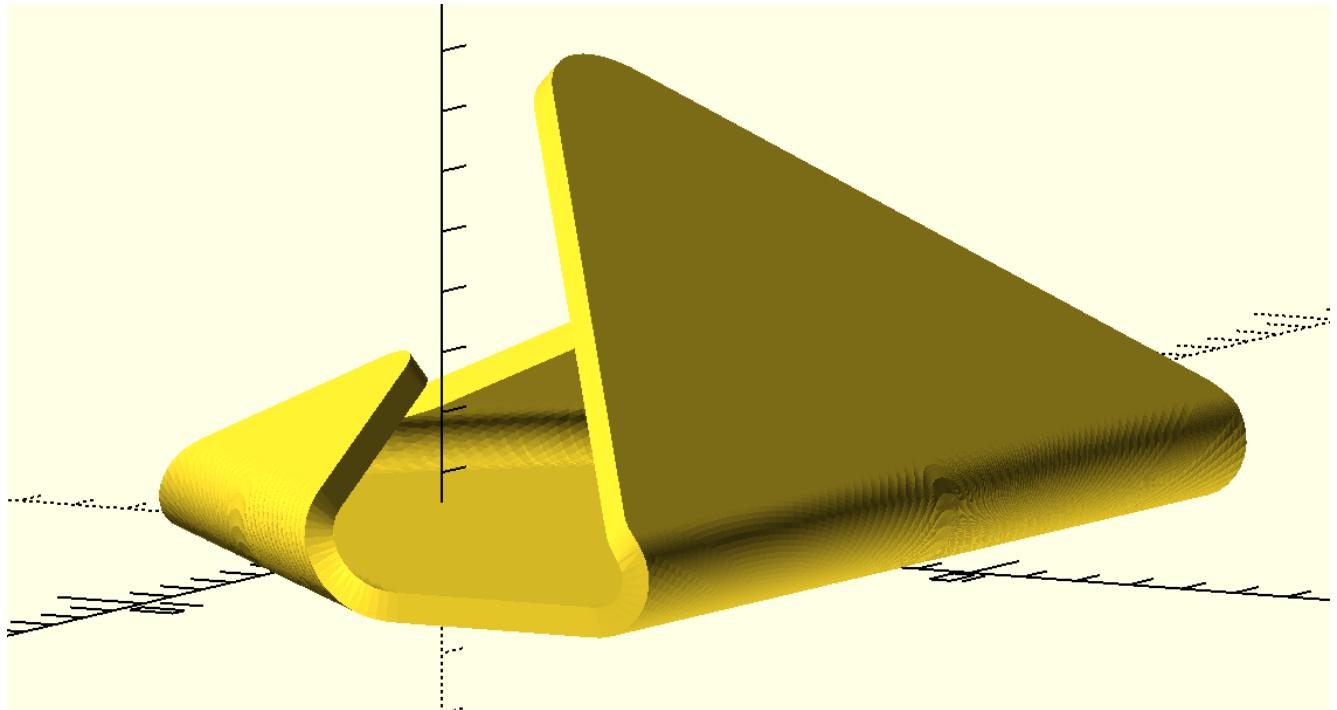
{swp(sol)}

...)

t1=time.time()
t1-t0

```

Out[7]: 1.5276551246643066



ball-bearing

In [112...]

```

# ball bearing

sec=corner_radius(pts1([[31,.5,0],[.5,-.5,0],[9.5,0,1],[0,15,1],[-9.5,0,0],[-.5,-.5,0]]),10)
sec1=corner_radius(pts1([[16,.5,0],[.5,-.5],[9,0,0],[0.5,.5,0],[0,14,0],[-.5,.5,0],[-9,0,0],[-.5,-.5,0]]),10)
sec2=circle(7,[28.5,7.5])

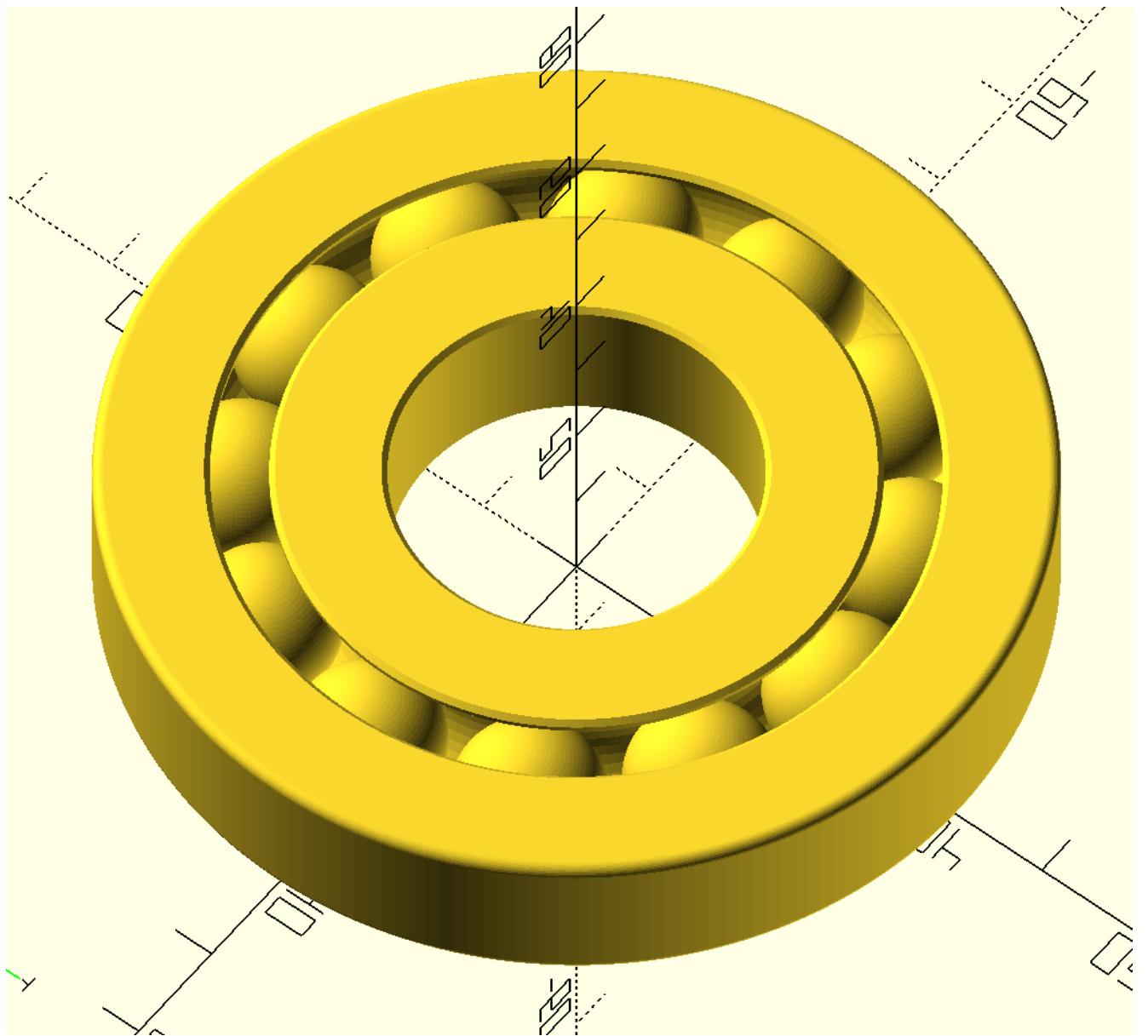
with open('trial.scad','w+') as f:
    f.write(f'''
    include<dependencies2.scad>

    rotate_extrude($fn=200)
    difference(){
        union(){
            polygon({sec});
            polygon({sec1});
        }
        polygon({sec2});
    }

    for(i=[0:360/12:359])
        rotate([0,0,i])
        translate([28.5,0,7.5])
        sphere(6.8,$fn=100);

    ...)

```



mobile-phone-stand

In [113...]

```
# mobile phone stand with application of functions wrap_around and surf_offset

t0=time.time()
sec1=corner_radius(pts1([[-75,20,2],[75,-20,280],[75,20,2],[0,70,2],[-75,7,280],[-75,-7,2]]),30)
cp1=array(offset(sec1,-35)).mean(0)
sec1=m_points(sec1,.5)
sec2=cpo([sort_points(sec1,[cp1]),sec1])
sec2=cpo([ls(p,60)for p in sec2])+[sec1]
path1=corner_radius(pts1([[0,60*sin(d2r(60)),0],[60*cos(d2r(-60)),60*sin(d2r(-60)),5+1],[15+10,0,4+1]],30))
path1=path1
path1=q_rot(['x90','z90'],path1)

surf1=[wrap_around(p,path1) for p in sec2]
surf2=surf_offset(flip(surf1),4)
sol=surf1+surf2

sec1=corner_radius(pts1([[-70,20,2],[70,-20,200],[70,20,2],[0,80,2],[-140,0,2]]),30)
cp1=array(offset(sec1,-25)).mean(0)
sec1=m_points(sec1,.5)
sec2=cpo([sort_points(sec1,[cp1]),sec1])
sec2=cpo([ls(p,60)for p in sec2])+[sec1]

path2=corner_radius(pts1([[0,0],[95,0,4+1],[100*cos(d2r(120)),100*sin(d2r(120))]]),50)
path2=q_rot(['x90','z90'],path2)
surf1=[wrap_around(p,path2) for p in sec2]
surf2=surf_offset(flip(surf1),4)
sol=surf1+surf2

sol2=[sol[57]]+slice_sol(sol[60:62],3)+[sol[63]]
```

```

p0,p1,p2,p3,p4,p5=sol2
fillet1=cpo(convert_3lines2fillet(p2,p0,p1))[:-1]
fillet2=cpo(convert_3lines2fillet(p5,p3,p4))[:-1]

sol3=flip(fillet1)+flip(fillet2)
sol4=sol[:57]+sol3+sol[64:]

sol2=[sol1[57]]+slice_sol(sol1[60:62],3)+[sol1[63]]

p0,p1,p2,p3,p4,p5=sol2
fillet1=cpo(convert_3lines2fillet(p2,p0,p1))[:-1]
fillet2=cpo(convert_3lines2fillet(p5,p3,p4))[:-1]

sol3=flip(fillet1)+flip(fillet2)
sol5=sol1[:57]+sol3+sol1[64:]

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>

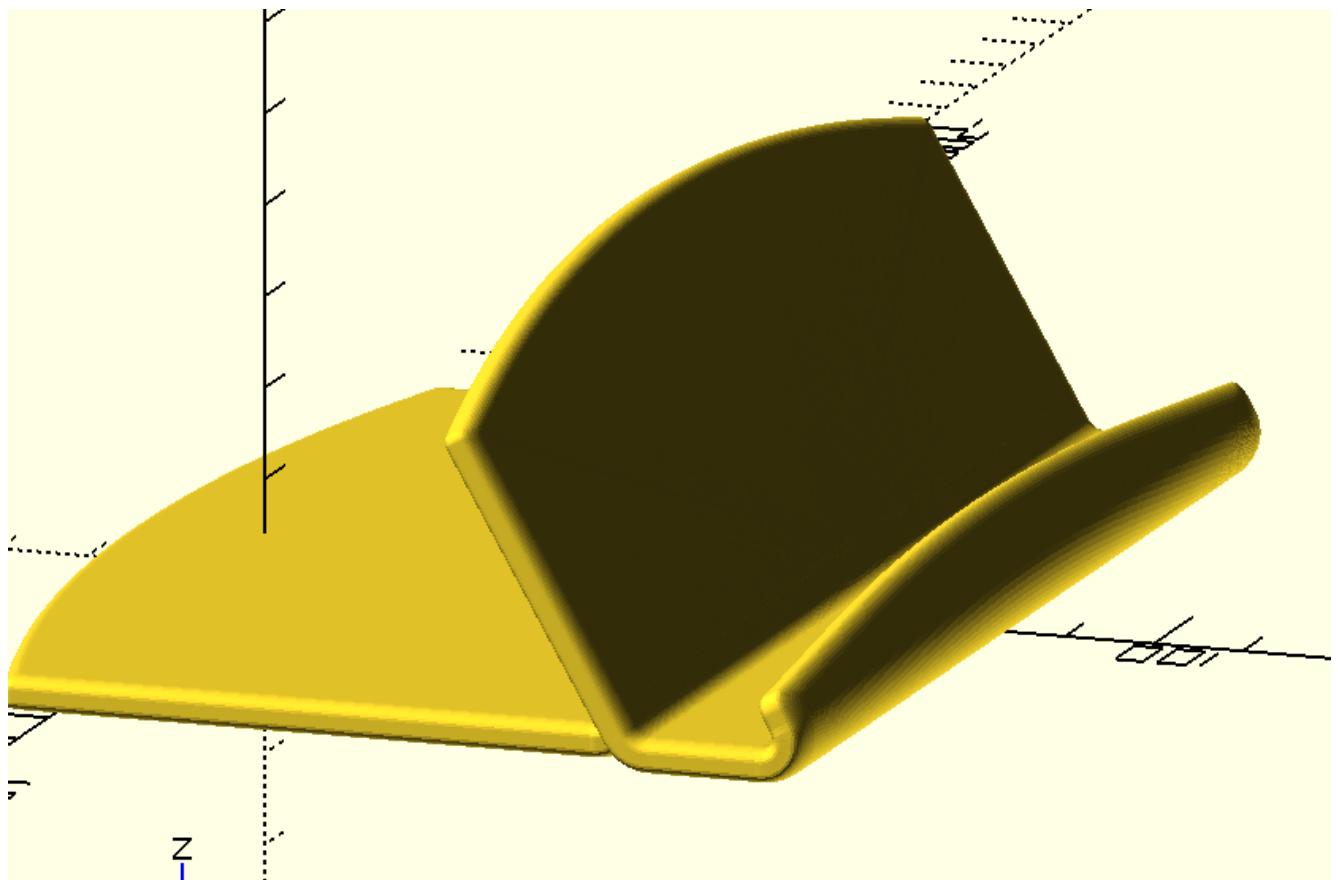
translate([0,40-4.625,0])
{swp(sol4)}
translate([0,-25.006125,0])
{swp(sol5)}

''')

t1=time.time()
t1-t0

```

Out[113... 20.074058055877686



In [111]: # mobile phone stand with application of functions wrap_around and surf_offset

```

t0=time.time()
sec1=corner_radius(pts1([[-75,20,2],[75,-20,280],[75,20,2],[0,70,2],[-75,7,280],[-75,-7,2]]),30)
cp1=array(offset(sec1,-35)).mean(0)
sec1=m_points(sec1,.5)
sec2=cpo([sort_points(sec1,[cp1]),sec1])
sec2=cpo([ls(p,60)for p in sec2])+[sec1]
path1=corner_radius(pts1([[0,60*sin(d2r(60)),0],[60*cos(d2r(-60)),60*sin(d2r(-60)),5+1],[15+10,0,4+1]],30))
path1=path1
path1=q_rot(['x90','z90'],path1)

```

```

surf1=[wrap_around(p,path1) for p in sec2]
surf2=surf_offset(flip(surf1),3)
sol=surf1+surf2

sec1=corner_radius(pts1([[-70,20,2],[70,-20,200],[70,20,2],[0,80,2],[-140,0,2]]),30)
cp1=array(offset(sec1,-25)).mean(0)
sec1=m_points(sec1,.5)
sec2=cpo([sort_points(sec1,[cp1]),sec1])
sec2=cpo([ls(p,60) for p in sec2])+[sec1]

path2=corner_radius(pts1([[0,0],[95,0,4+1],[100*cos(d2r(120)),100*sin(d2r(120))]]),50)
path2=q_rot(['x90','z90'],path2)
surf1=[wrap_around(p,path2) for p in sec2]
surf2=surf_offset(flip(surf1),3)
sol1=surf1+surf2

sol2=[sol[57]]+slice_sol(sol[60:62],3)+[sol[63]]

p0,p1,p2,p3,p4,p5=sol2
fillet1=cpo(convert_3lines2fillet(p2,p0,p1))[:-1]
fillet2=cpo(convert_3lines2fillet(p5,p3,p4))[:-1]

sol3=flip(fillet1)+flip(fillet2)
sol4=sol[:57]+sol3+sol[64:]

sol2=[sol1[57]]+slice_sol(sol1[60:62],3)+[sol1[63]]

p0,p1,p2,p3,p4,p5=sol2
fillet1=cpo(convert_3lines2fillet(p2,p0,p1))[:-1]
fillet2=cpo(convert_3lines2fillet(p5,p3,p4))[:-1]

sol3=flip(fillet1)+flip(fillet2)
sol5=sol1[:57]+sol3+sol1[64:]

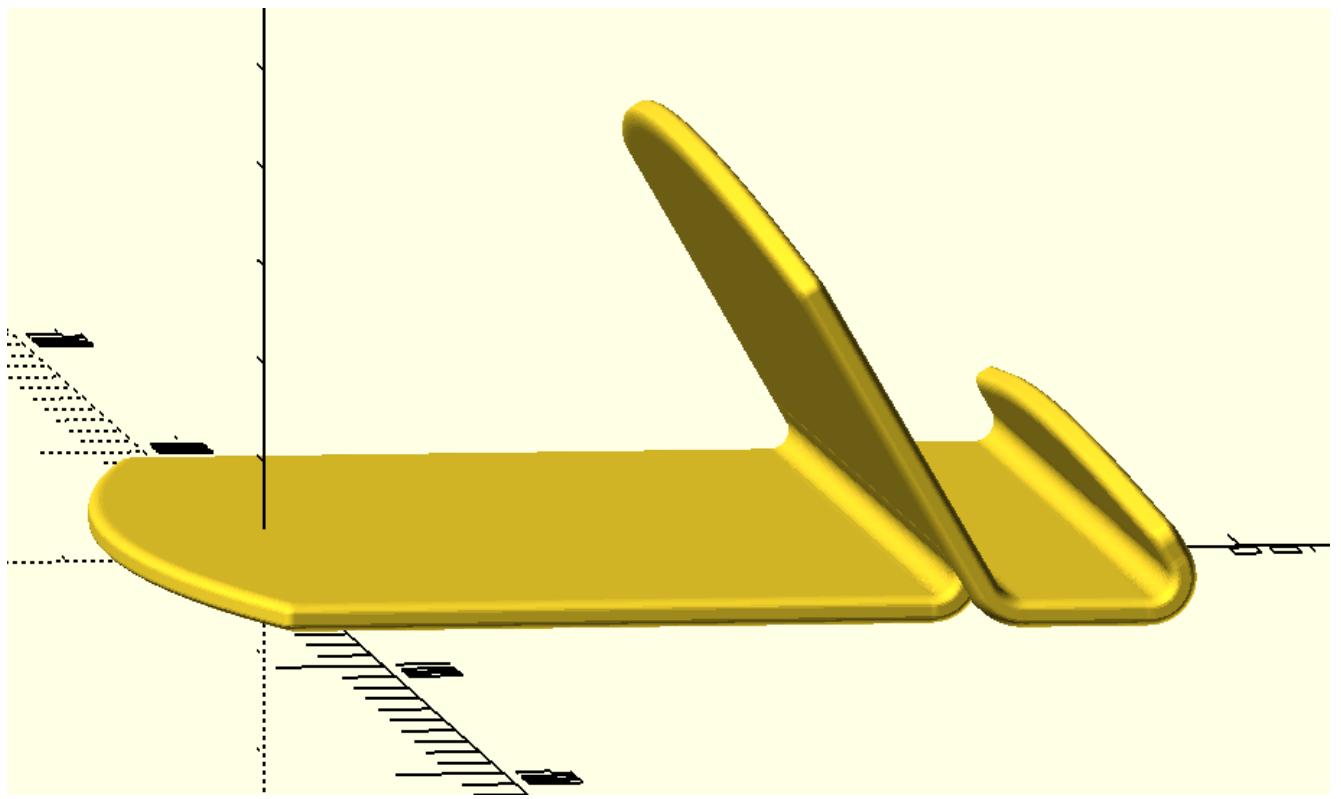
with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>

translate([0,40-4.625,0])
{swp(sol4)}
translate([0,-26.2,0])
{swp(sol5)}

''')
t1=time.time()
t1-t0

```

Out[111]: 20.579778909683228

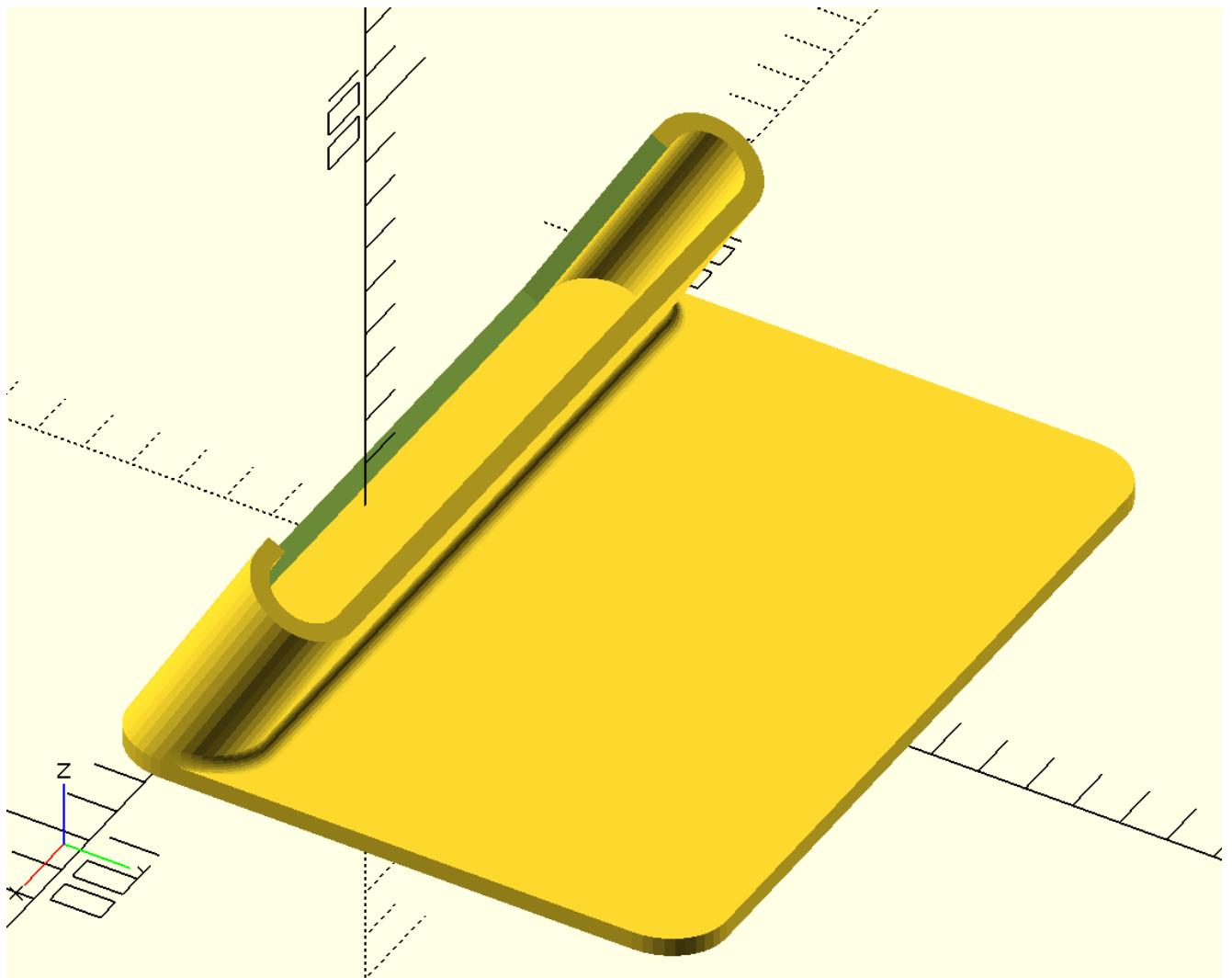


```
In [116]: # car mobile stand
sec=corner_radius(pts1([[-160/2-4,-15/2-4,23/2-.1],[160+2*4,0,23/2-.1],[0,15+2*4,23/2-.1],[-160-2*4,0,23/2-.1],[0,-15-2*4,23/2-.1],[160-2*4,0,23/2-.1],[0,15-2*4,23/2-.1],[-160/2-4,-15/2-4,23/2-.1]]),10)
sol=o_solid([0,1,sqrt(3)],sec,60)
sol1=surf_offset(sol,-4)
sol2=swp_prism_h(sol,sol1)
surf1=translate([0,0,10],plane([0,0,1],200))
ip1=ip_surf(surf1,sol)
surf2=ip1[:22]+translate([0,100,0],ip1[22:44])
surf3=translate([0,0,-4],surf2)
sol3=[surf3]+[surf2]
sol4=o_solid([0,0,1],square(200,True),20,-10)

sec1=corner_radius(pts1([[-130/2,0],[130,0],[5,60],[-130-2*5,0]]),10)
sol5=o_solid([0,-sqrt(3),1],sec1,10,5,0,10)

sol6=[ip1[22:]]+[translate([0,2,0],ip1[22:])] + [translate([0,1,sqrt(3)],ip1[22:])]
fillet1=convert_3lines2fillet(translate([0,5,0],ip1[22:]),translate([0,2.4,4.34],ip1[22:]),ip1[22:])
with open('trial.scad','w+') as f:
    f.write(f'''')
include<dependencies2.scad>

difference(){{{
{swp_c(sol2)}
{swp(sol4)}
{swp(sol5)}
}}
//%{swp(surf1)}
//color("magenta")points({ip1[:22]},.5);
{swp(sol3)}
{swp(fillet1)}
...
})
```



back-camera-clamp

```
In [117]: # back camera clamp

res=.5
sec=corner_radius(pts1([[-37/2,0.1,.5],[37,0,.5],[0,15,.5],[-5,24,13.5],[-27,0,13.5],[-5,-24,.5]]),20)
cp1=array(offset(sec,-17)).mean(0)
sec1=[sort_points(m_points(sec,res),[cp1])]+[m_points(sec,res)]
sec2=cpo([ls(p,20) for p in cpo(sec1)])+[m_points(sec,res)]

path=corner_radius(pts1([[0,0],[15,0,3.5],[15,41.21]]),10)
path=q_rot(['x90','z90'],path)

sec3=[wrap_around(p,path) for p in sec2]
sec4=surf_offset(flip(sec3),2)

sol=sec3+sec4

sec5=translate([0,0,-1],c2t3([circle(.1,[0,28])]+[circle(9,[0,28])]))
sec5=[wrap_around(p,path) for p in sec5]
sec6=surf_offset(flip(sec5),4)

sol1=sec5+sec6

cyl1=cylinder(d=4,h=5)

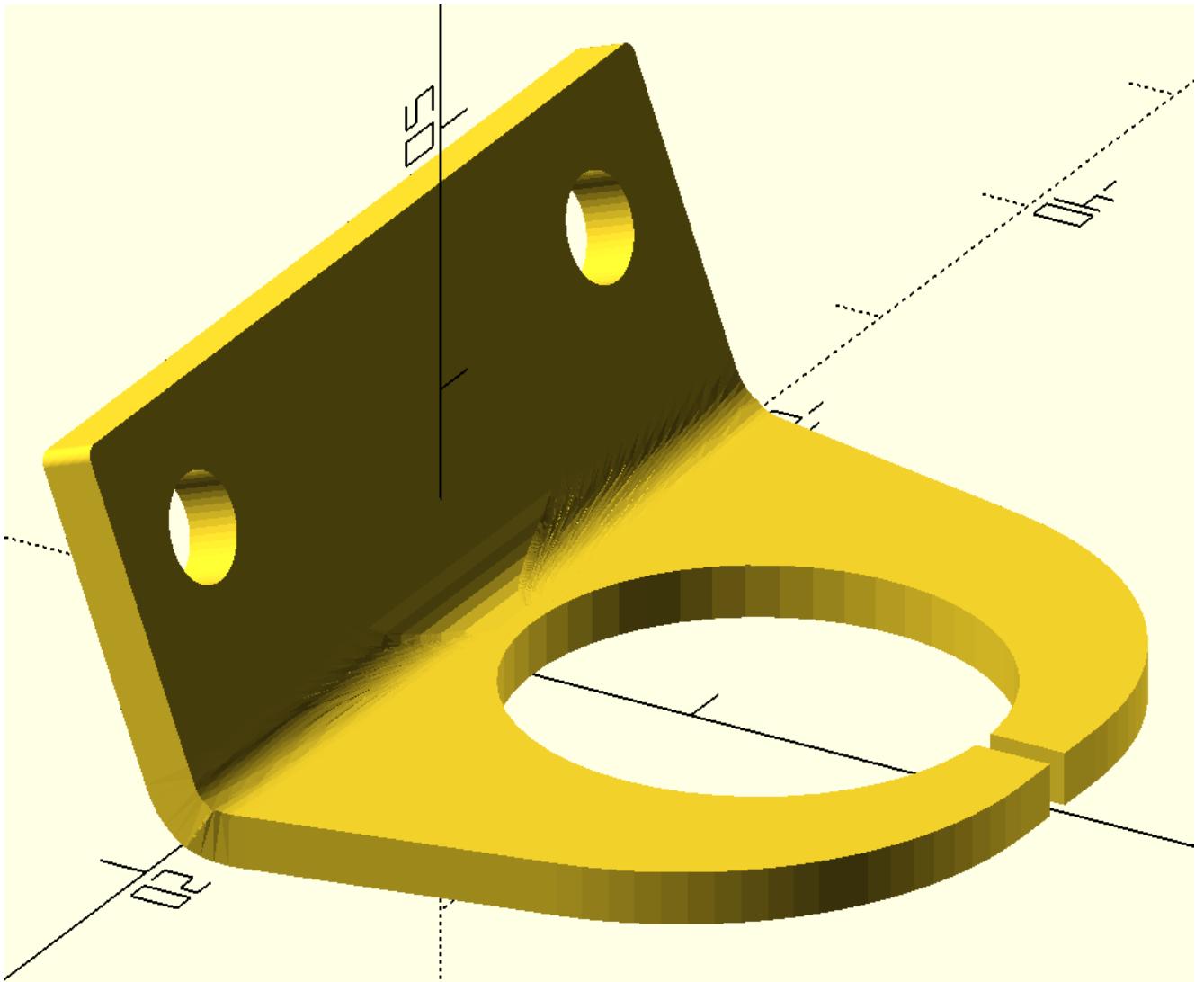
slit=q_rot(['x70'],translate([-5,15,-17],linear_extrude(square([1,20]),10)))

with open('trial.scad','w+') as f:
    f.write(f'''  
include<dependencies2.scad>
rotate([-70,0,0])
translate([0,-15,0])
difference(){{  
{swp(sol)}  
{swp(sol1)}}
```

```

for(i=[-27/2,27/2])
translate([i,5,-2])
{swp(cyl1)}

{swp(slit)}
}}
'''
```



align_sol

In [3]: # example of function align_sol(sol,angl)

```

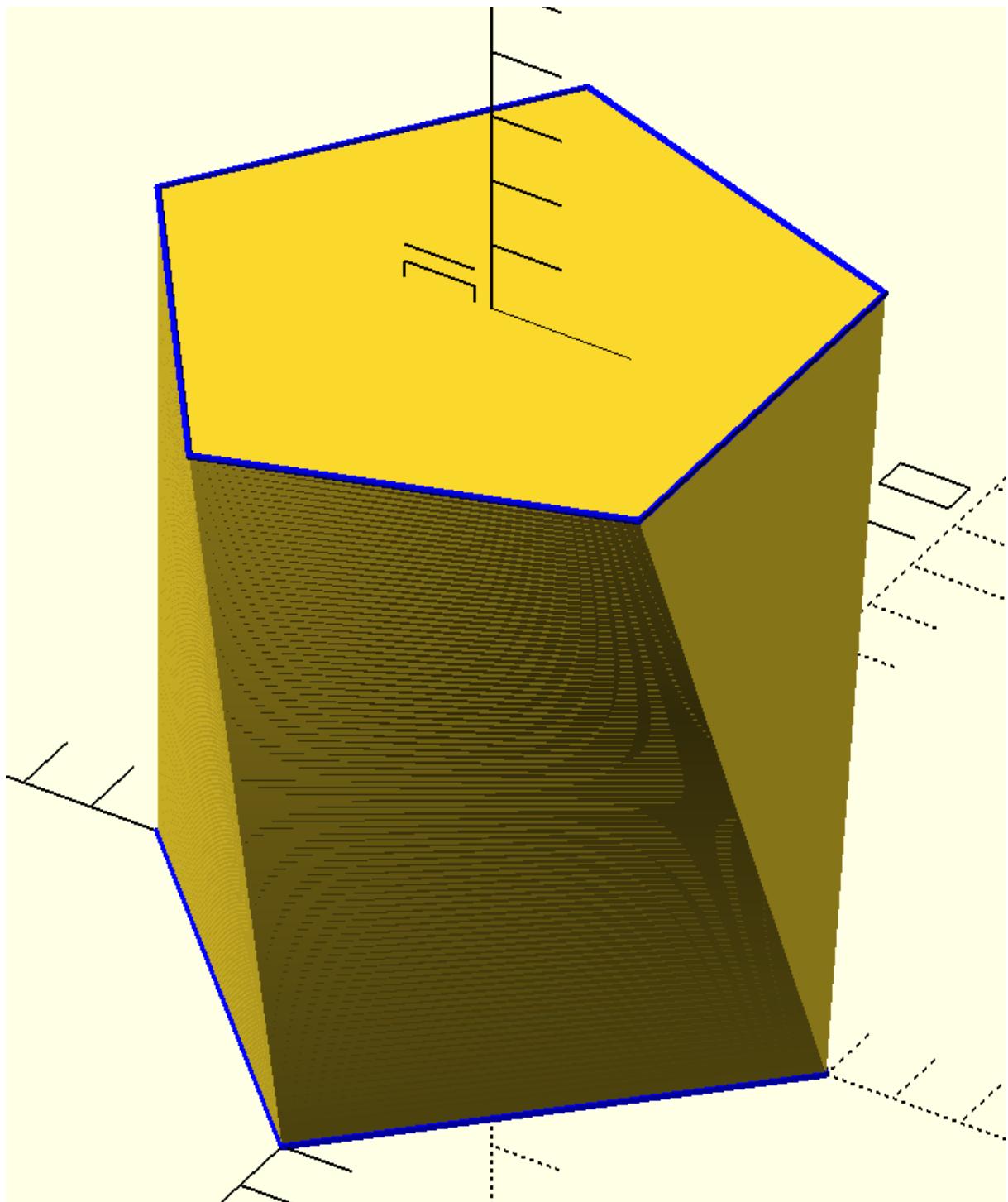
sqr1=circle(5,s=5)
pent1=circle(5,s=6)

sqr1=c2t3(sort_points(pent1,sqr1))
pent1=translate([0,0,10],pent1)
sol=[sqr1]+[pent1]
sol=align_sol(sol,1)
sol1=slice_sol(sol,100)

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>

{swp(sol1)}
color("blue")p_line3dc({sol[0]},.05);
color("blue")p_line3dc({sol[1]},.05);

'''')
```

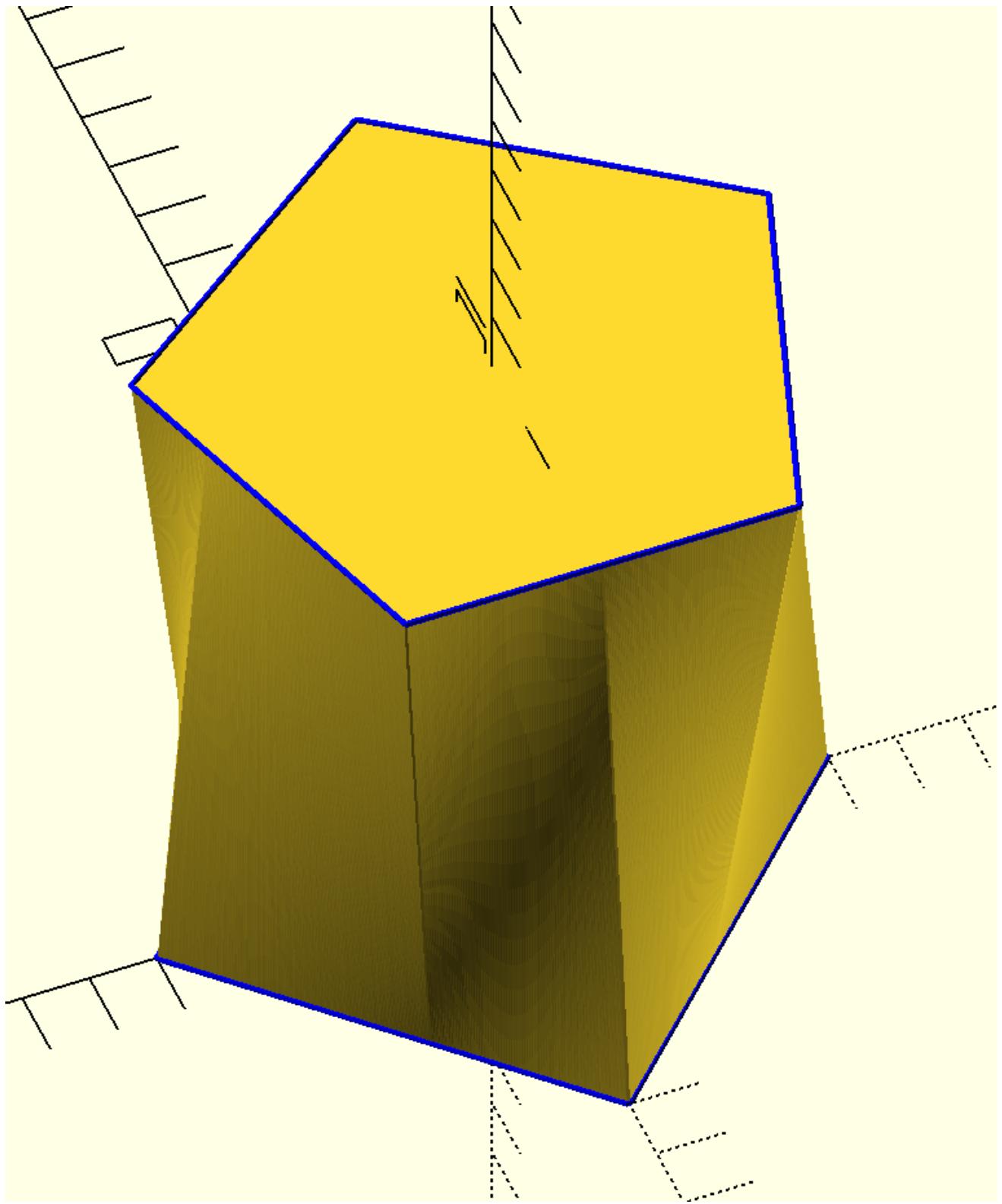


In [119]: # example of function align_sol(sol,angl)

```
sqr1=c2t3(equidistant_pathc(circle(5,s=5),200)[:200])
pent1=equidistant_pathc(circle(5,s=6),200)[:200]
pent1=translate([0,0,10],pent1)
sol=[sqr1]+[pent1]
sol=align_sol_1(sol)
sol1=slice_sol(sol,10)

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>

{swp(sol1)}
color("blue")p_line3dc({sol[0]},.05);
color("blue")p_line3dc({sol[1]},.05);
...)
```



```
In [4]: sqr1=circle(5,s=5)
pent1=circle(5,s=6)

sqr1,pent1=c2t3(equate_points(sqr1,pent1))
pent1=translate([0,0,10],q_rot(['y-30'],pent1))
sol=[sqr1]+[pent1]
sol=align_sol(sol,.1)
sol1=slice_sol(sol,100)

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>

{swp(sol1)}
color("blue")p_line3dc({sol[0]},.05);
color("blue")p_line3dc({sol[1]},.05);
color("magenta")p_line3dc({pent1},.05);

...)
```

```
In [7]: # example of function align_sol(sol, ang=10)
t0=time.time()

path=cr_3d([[0,0,0,3],[15,0,0,4],[5,3,15,3],[0,10,0,3],[-5,3,-15,4],[-15,0,0,3]],10)
sec=pts([[-1.5,-1.25],[3,0],[-1.5,2.5]])
sol=path_extrude_closed(sec,path)
# sol=slice_sol(sol,2)
sol=align_sol(sol,1)

sol=slice_sol(sol,10)
with open('trial.scad','w+') as f:
    f.write(f'''  

include<dependencies2.scad>  

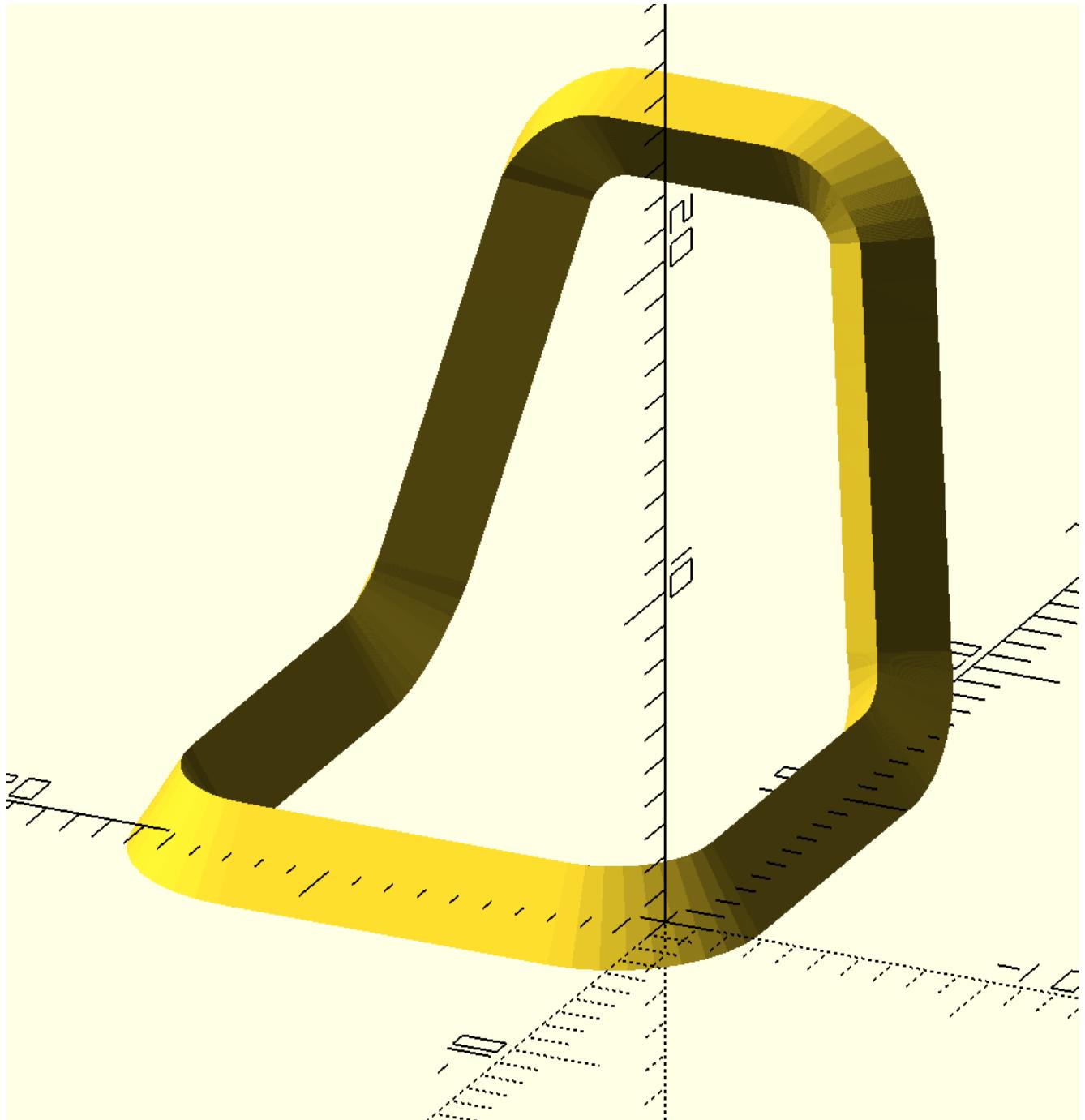
  

{swp_c(sol)}  

...''')
t1=time.time()
t1-t0
```

Out[7]: 1.2172250747680664



In [121...]: # approach to align absolutely different shapes

```
sec=corner_radius(pts1([[0,0,1],[8,3,3],[5,7,1],[-8,0,2],[-5,20,1]]),30)
sec1=scl2d_c(corner_radius(pts1([[-15,0,2.4],[0,15,3],[30,0,3],[0,-15,2.4],[5,0,2.4],[0,20,7],[-40,0,7]
```

```

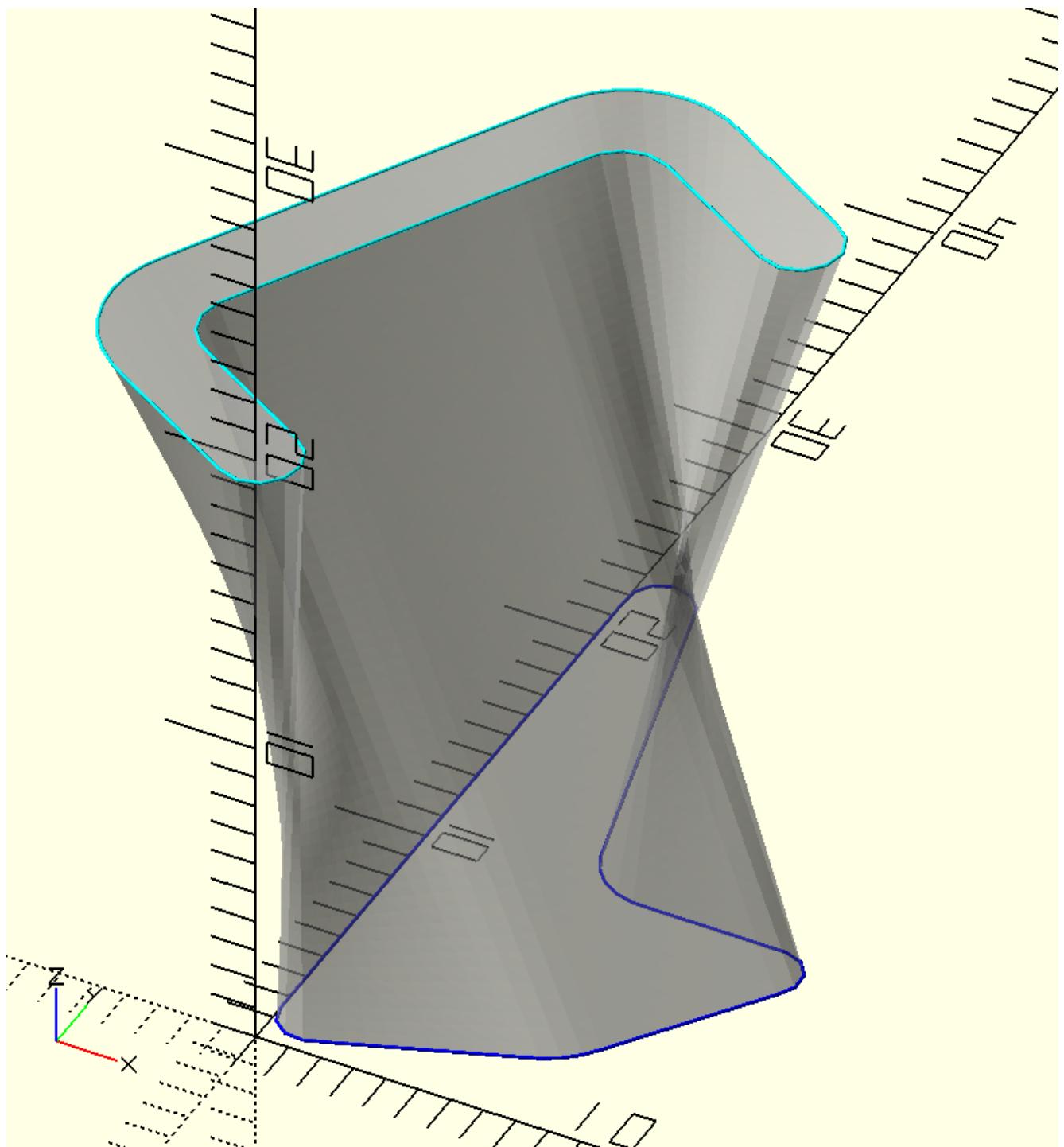
sec2=c2t3(equidistant_pathc(sec,100))
sec3=translate([0,0,20],equidistant_pathc(sec1,100))
sol=[sec2,sec3]
sol=slice_sol(align_sol(sol),50)

t0=time.time()
with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>
difference() {{
    {swp(sol)}
    //cube(10);
}}
//color("blue")p_line3dc({sol[0]},.05);
//color("cyan")p_line3dc({sol[-1]},.05);

''')
t1=time.time()
t1-t0

```

Out[121... 0.04557490348815918



align_sol_1

In [122...]

```
# example of function align_sol_1(sol)

t0=time.time()

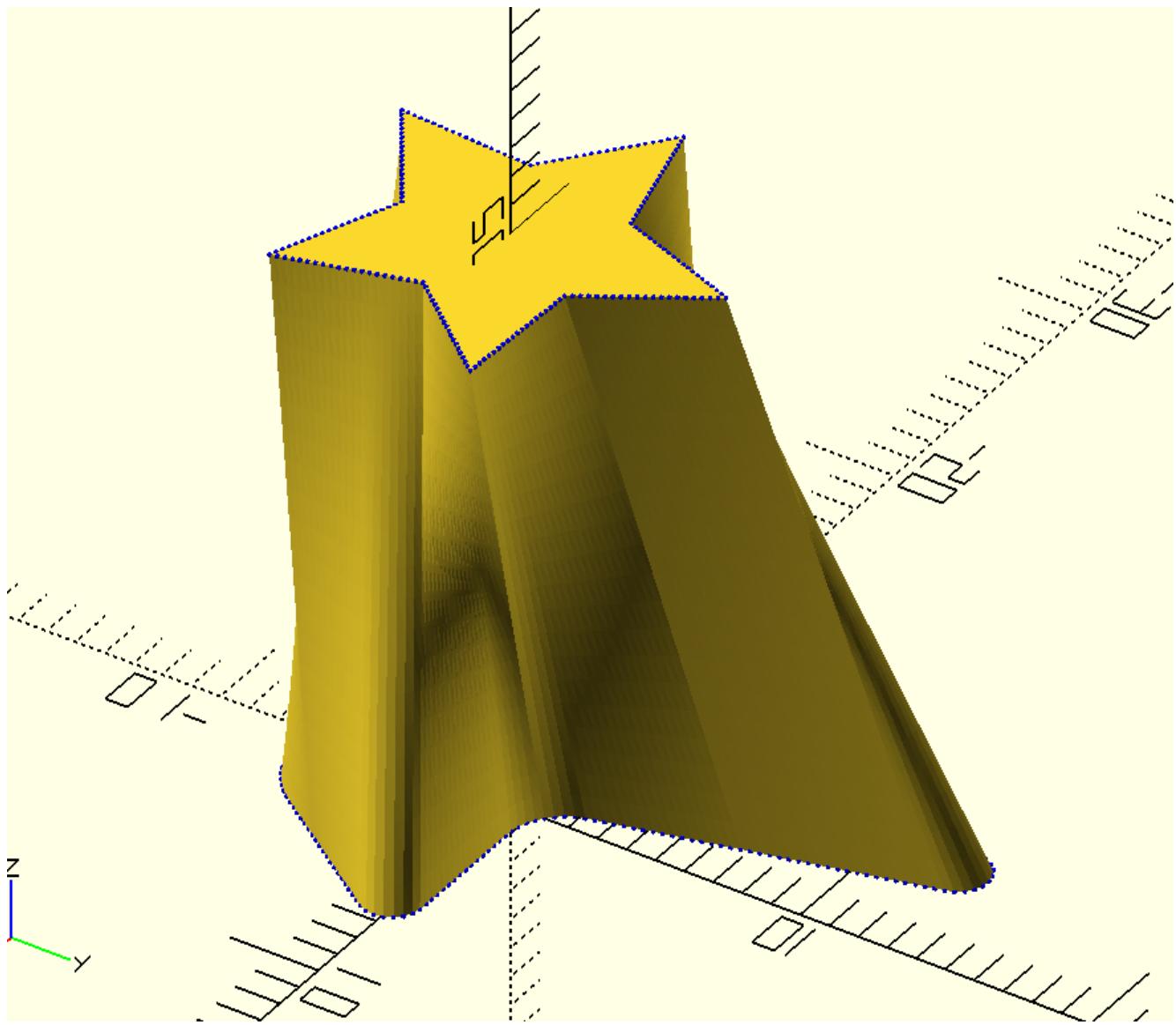
sec=corner_radius(pts1([[0,0,1],[8,3,3],[5,7,1],[-8,0,2],[-5,20,1]]),20)
cp1=array(sec).mean(0)

pent1=circle(7,s=6)
pent2=c3t2(q_rot([f'z{360/5/2}'],circle(3.5,s=6)))
sec1=concatenate(cpo([pent1]+[pent2])).tolist()

sec=translate(-cp1,(equidistant_pathc(sec,200)))
sec1=translate([0,0,20],equidistant_pathc(sec1,200))
sol=slice_sol(align_sol_1([sec,sec1]),20)

with open('trial.scad','w+') as f:
    f.write(f'''  
include<dependencies2.scad>  
  
sec={sec};  
sec1={sol[1]};  
{swp(sol)}  
color("blue")points({sol[-1]},.1);  
color("blue")points({sol[0]},.1);  
//for(i=[0:len(sec)-1])translate(sec[i])linear_extrude(.1)text(str(i),.2);  
//for(i=[0:len(sec1)-1])translate(sec1[i])linear_extrude(.1)text(str(i),.2);  
  
'''')  
  
t1=time.time()  
t1-t0
```

Out [122... 0.12352514266967773



```
In [10]: # very fine merging of 2 very different shapes
t0=time.time()
sec=corner_radius(pts1([[0,0,1],[8,3,3],[5,7,1],[-8,0,2],[-5,20,1]]),20)
sec1=corner_radius(pts1([[0,0,1],[7,5,2],[5,7,3],[-5,7,2],[-7,5,3]]),20)
sec1=corner_radius(pts1([[-15,0,2.4],[0,15,3],[30,0,3],[0,-15,2.4],[5,0,2.4],[0,20,7],[-40,0,7],[0,-20,3]]),20)

s1=translate([0,0,20],equidistant_pathc(sec,1000))
s2=c2t3(equidistant_pathc(sec1,1000))

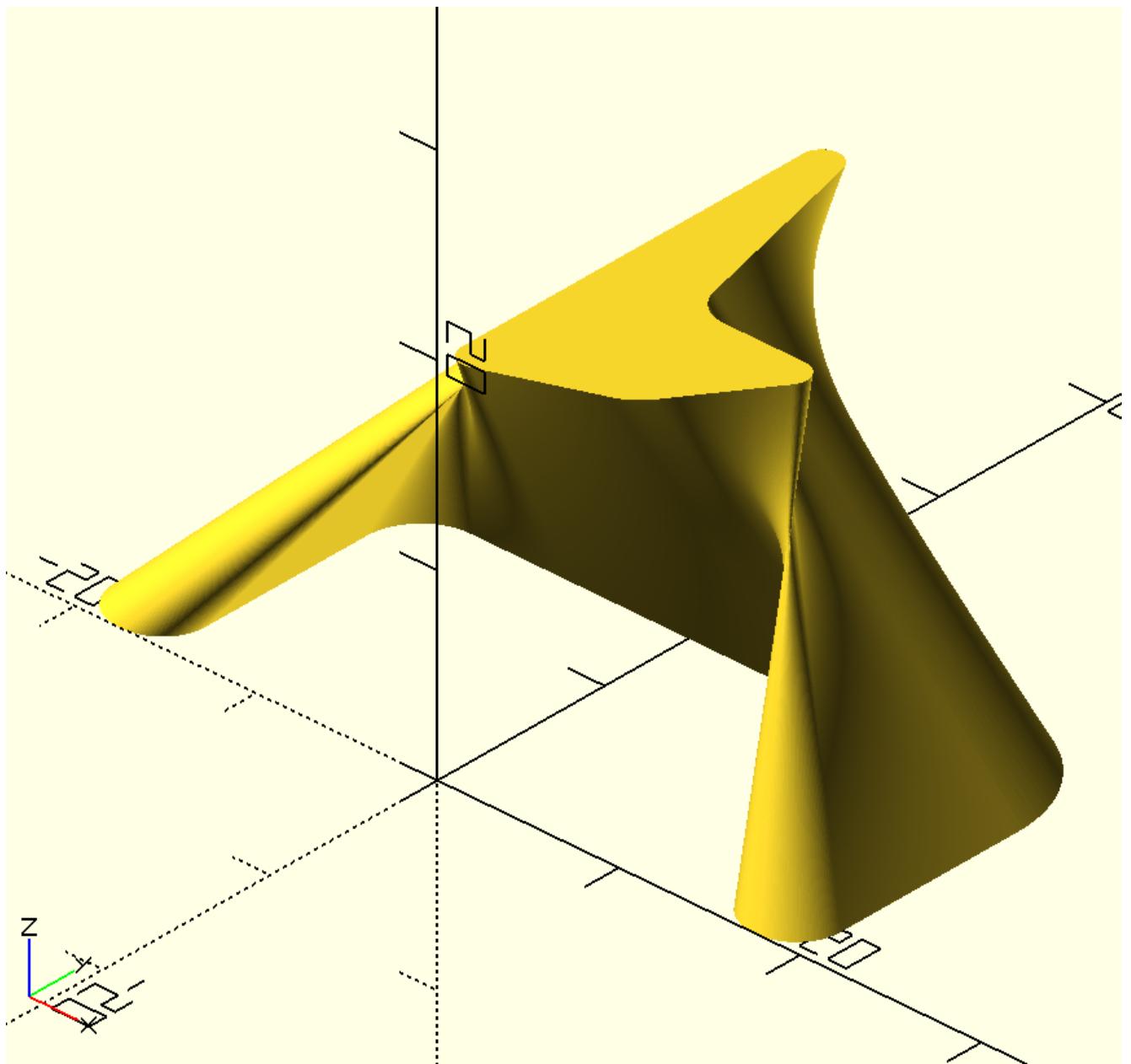
s3=slice_sol(align_sol_1([s2]+[s1]),100)

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>

swp({s3});

...))
t1=time.time()
t1-t0
```

Out[10]: 1.2676680088043213



end_cap

```
In [11]: # example of function path_extrude_open and end_cap
```

```
i_t=time.time()
sec=corner_radius(pts1([[-1.5,-1.5,.5],[3,0,.5],[0,3,1.49],[-3,0,1.49]]),10)
path=q_rot(['x-90'],cr_3d([[0,0,0,0],[3,5,10,5],[17,-2,5,6],[1,-10,10,0]],10))
sol=q_rot(['x90'],path_extrude_open(sec,path))
sol=align_sol_1(sol)
sol=slice_sol(sol,10)

e_cap=end_cap(sol,1)
f3=e_cap[0]
f4=e_cap[1]

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>

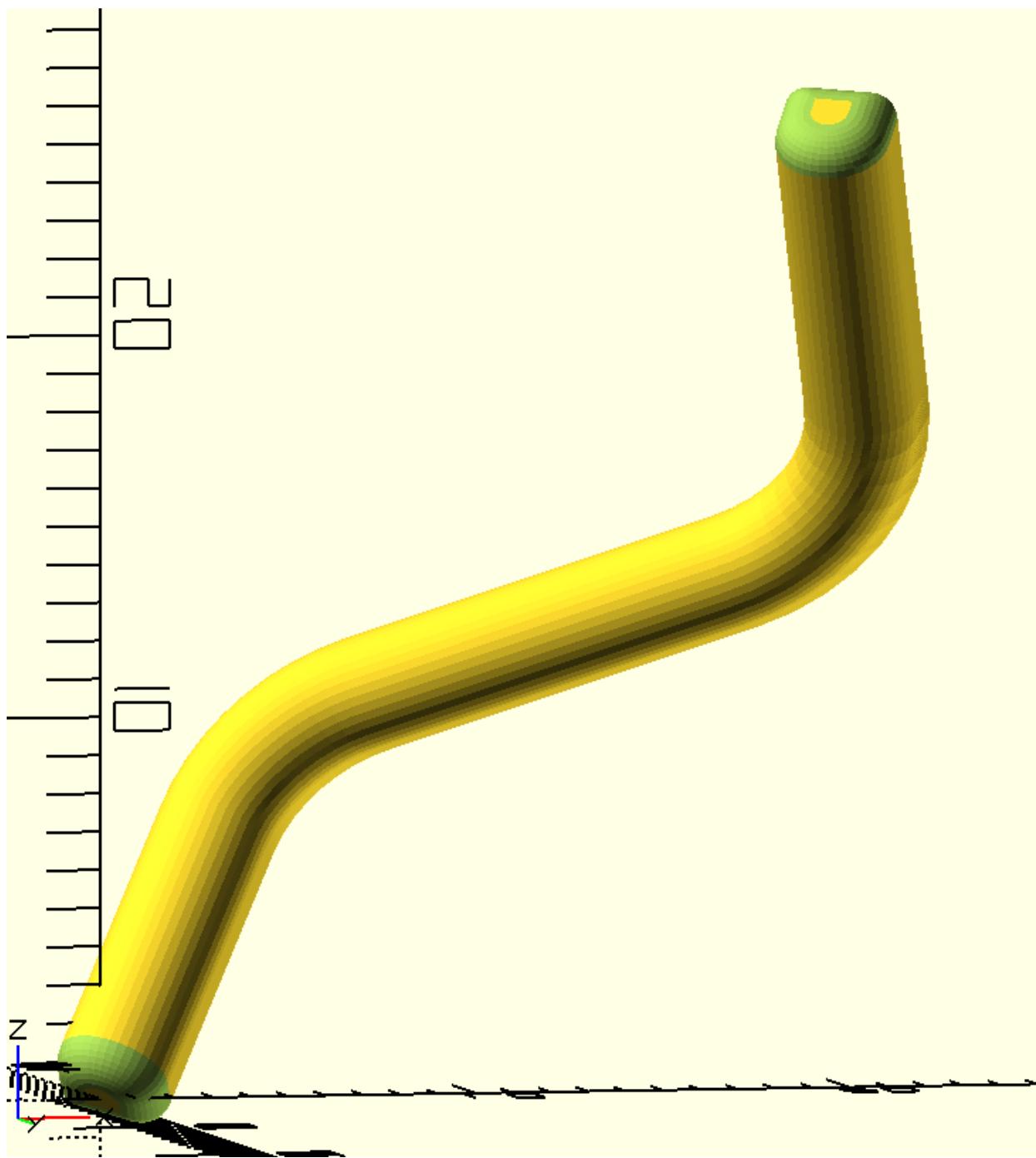
difference() {{
{swp(sol)}
{swp_c(f3)}
{swp_c(f4)}

}}\n
...)
```

```
f_t=time.time()  
f_t-i_t
```

```
/Users/sanjeevprabhakar/openscad/openscad1.py:5639: RuntimeWarning: invalid value encountered in divide  
u1=v1/norm(v1)
```

```
Out[11]: 1.00459885597229
```



sec2vector

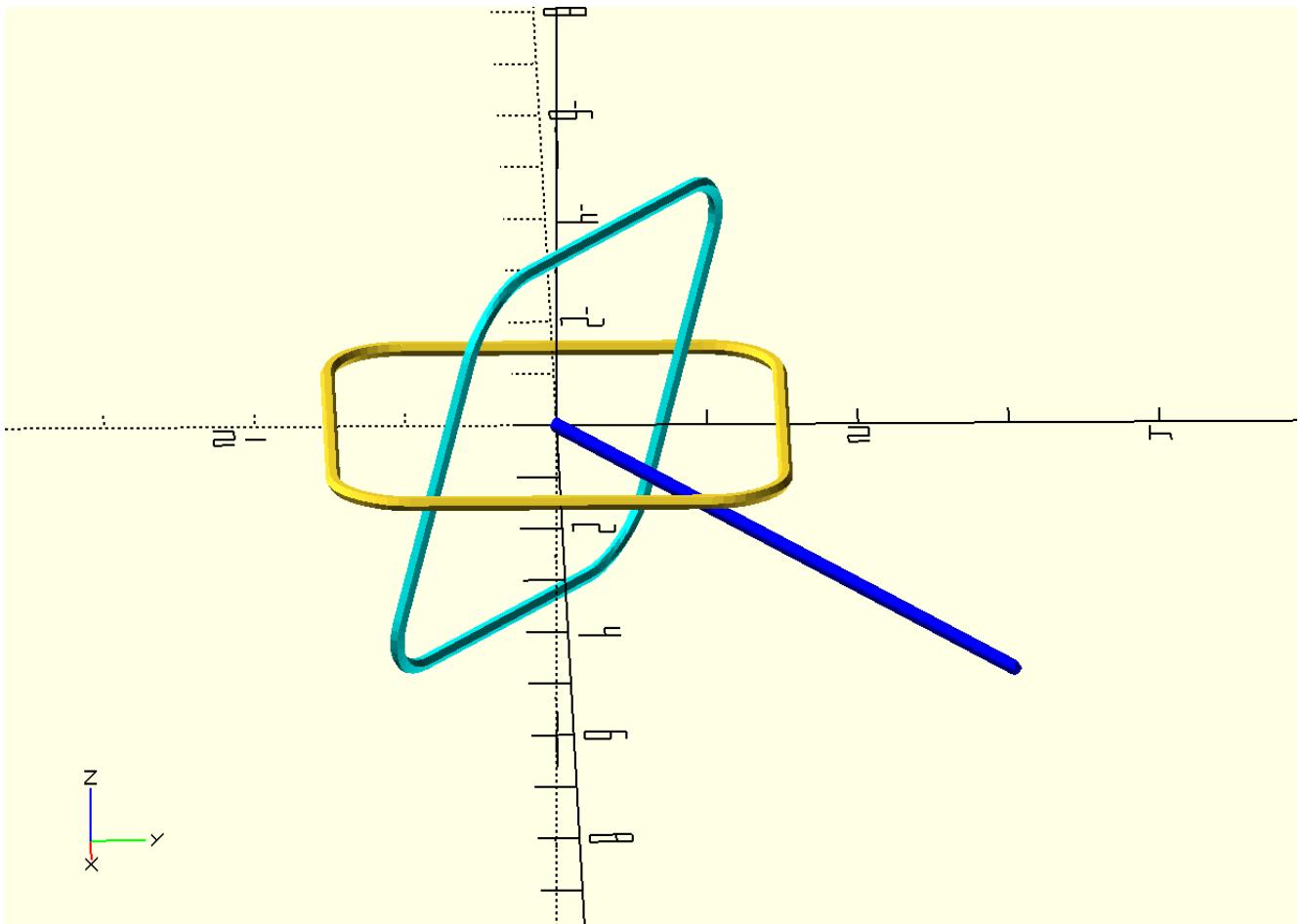
In [125...]

```
# example of function sec2vector(v1,sec)  
i_t=time.time()  
sec=corner_radius(pts1([[-1.5,-1.5,.5],[3,0,.5],[0,3,.5],[-3,0,.5]]),10)  
v1=[1,2,-1]  
vector1=[[0,0,0],v1]  
sec1=sec2vector(v1,sec)  
  
with open('trial.scad','w+') as f:  
    f.write(f'''  
include<dependencies2.scad>  
// original section  
p_line3dc({sec},.05);  
  
//aligned section in cyan color  
color("cyan")p_line3dc({sec1},.05);
```

```
//vector with whom the section is expected to be aligned with in blue color
color("blue")p_line3d({vector1},.05);

...)
f_t=time.time()
f_t-i_t
```

Out [125... 0.01011204719543457



cut_plane

```
# example of function cut_plane(nv, radius, thickness, trns)

t0=time.time()

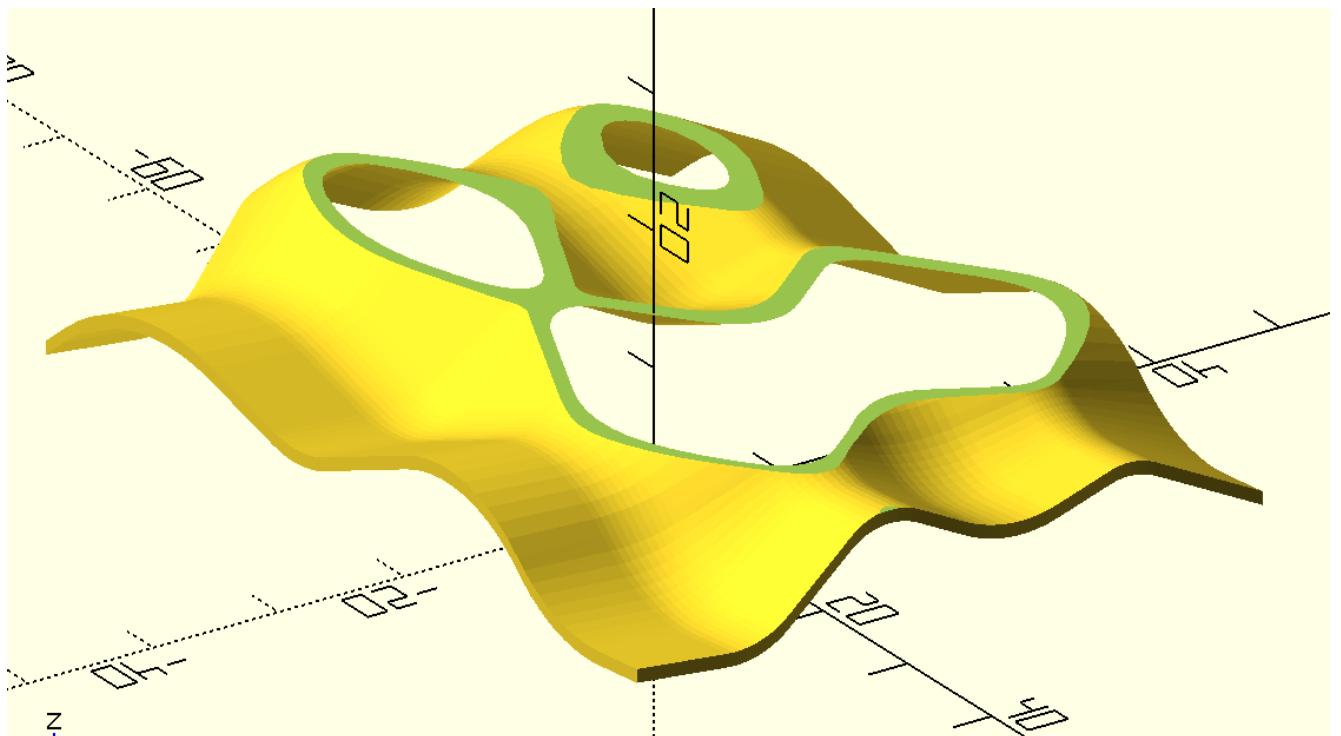
sec2=corner_radius(pts1([[-25,0],[10,5,5],[10,-3,10],[10,5,5],[10,-8,7],[10,1]]),10)
path2=cytz(corner_radius(pts1([[-35,5,0],[10,8,20],[20,-5,10],[20,8,20],[10,-9,20],[10,1,0]]),10))
surf2=surf_extrude(sec2,path2)
surf3=surf_extrudef(surf2,t=-1)

with open('trial.scad','w+') as f:
    f.write(f''''
include<dependencies2.scad>

difference(){
{swp(surf3)}
{swp(cut_plane([0,0,1],[100,100],10,14))}

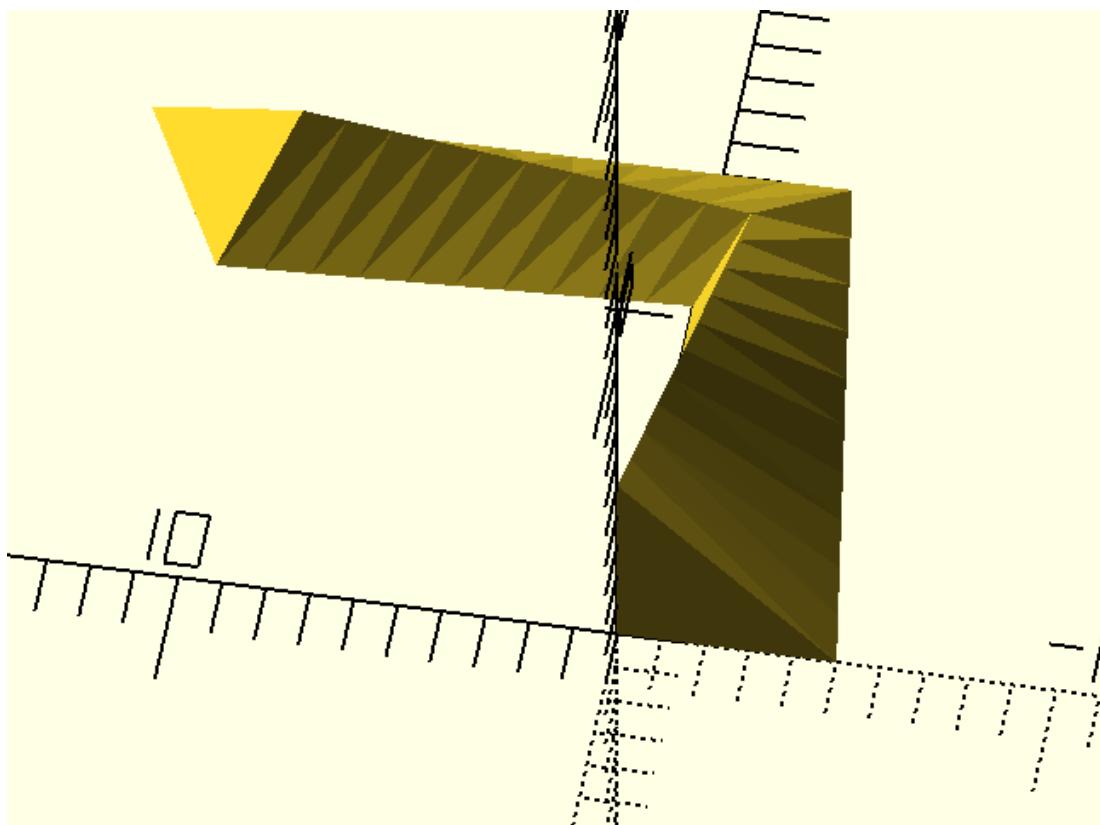
...)
t1=time.time()
total=t1-t0
total
```

Out [126... 0.11357808113098145



In [127]:

```
sec=[[0,0],[5,0],[0,5]]
path=[[0,0,0],[10,0,0],[5,10,5]]
sol=path_extrude_open(sec,path)
# sol=align_sol_1(sol)
sol2=slice_sol(sol,20)
with open('trial.scad','w+') as f:
    f.write(f'''  
include<dependencies2.scad>  
  
difference(){{
    {swp(sol2)}  
  
//{swp(cut_plane([1,1,1],30,7,-3))}  
}}  
''' )
```



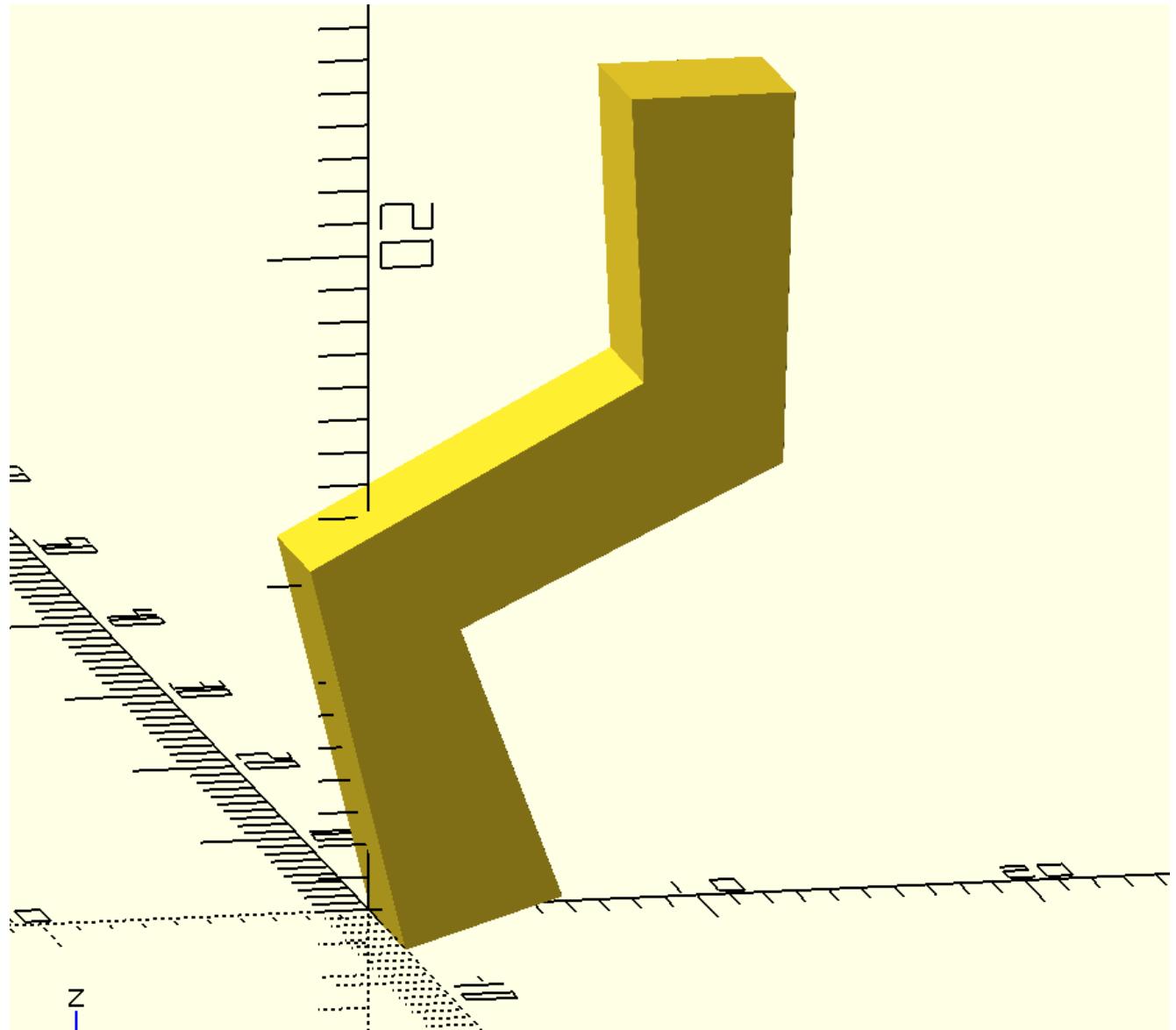
slice_sol

In [12]:

```
sec=square(5,True)
path=q_rot(['x-90'],[[3,0,0],[0,0,10],[10,0,15],[10,0,25]])
sol=q_rot(['x90'],path_extrude_open(sec,path))
# sol=align_sol(sol)
sol2=slice_sol(sol,10)
with open('trial.scad','w+') as f:
    f.write(f'''  
include<dependencies2.scad>
//color("blue")p_line3dc({sec},.1,1);
//color("blue")p_line3d({cytz(path)},.1,1);
//color("cyan")points({cytz(path)},.5);

{swp(sol2)}
//color("magenta")for(p={sol2})p_line3dc(p,.1,1);

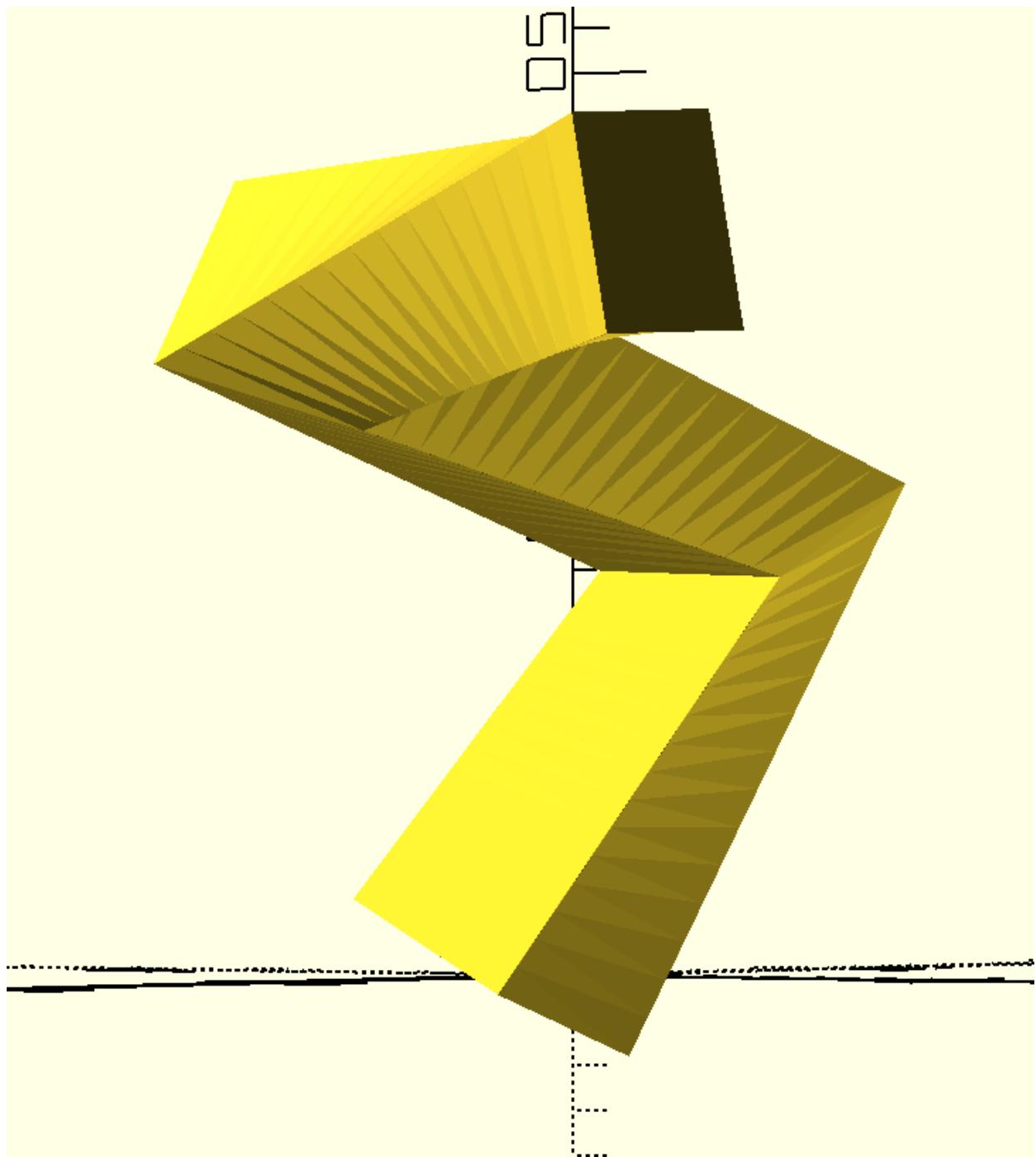
...)
```



In [10]:

```
sec=square(5,True)
path=q_rot(['x0'],pts2([[3,0,0],[-3,5,10],[10,-5,5],[0,10,2]]))
sol=path_extrude_open(sec,path)
sol=align_sol(sol,1)
sol2=slice_sol(sol,5)
sol3=offset_sol(sol2,.5,1)
sol4=swp_prism_h(sol3,sol2)
with open('trial.scad','w+') as f:
    f.write(f'''  
include<dependencies2.scad>
color("magenta")p_line3d({path},.1,rec=1);
//color("cyan")for(p={sol2})p_line3dc(p,.1);
{swp_c(sol4)}

...)
```



o_solid

In [130]: # example of function o_solid(*nv, sec, thickness, trns1, trns2, trns3*)

```
# sec=corner_radius(pts1([[0,0,1],[8,3,3],[5,7,1],[-8,0,2],[-5,20,1]]),30)
# sec=corner_radius(pts1([[0,0,1],[7,5,2],[5,7,3],[-5,7,5],[-7,5,5]]),10)
sec=corner_radius(pts1([[-15,0,2.4],[0,15,3],[30,0,3],[0,-15,2.4],[5,0,2.4],[0,20,7],[-40,0,7],[0,-20,
# sec=corner_radius(pts1([[0,0,1],[10,0,1],[0,5,1],[-10,0,1]]),10)
# sec=circle(10)

sol=o_solid([1,0,0],sec,20,-10,0,0,theta=[0,0,0])
with open('trial.scad','w+') as f:
    f.write(f'''  

include<dependencies2.scad>  

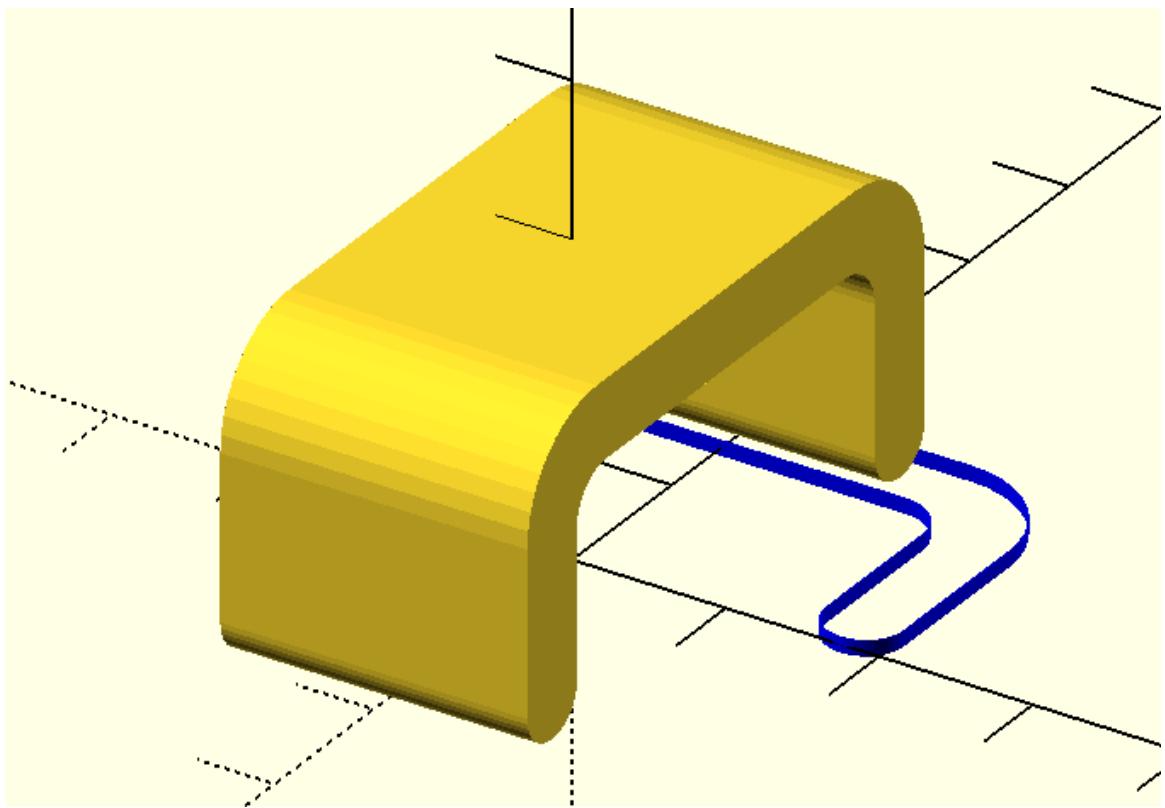
  

{swp(sol)}  

color("blue")p_line({sec},.05);  

...)
```



```
In [131...]
t0=time.time()
sec=corner_radius(pts1([[-5,-5,1],[10,0,1],[0,10,1],[-10,0,1]]),5)
sec1=corner_radius(pts1([[-5,-5,1],[10,0,1],[0,10,1],[-10,0,1]]),5)

path=corner_radius(pts1([[-4,0],[4,0,1],[0,10,1],[-4,0]]),5)
# path=equidistant_path(path,100)
sol1=prism(sec,path)

sol2=o_solid([1,0,.1],circle(2,s=15),15,-7,0,10,[-90,0,0])

with open('trial.scad','w+')as f:
    f.write(f''''
        include<dependencies.scad>

{swp(sol1)}
{swp(sol2)}

    ''')
t1=time.time()
t1-t0
```

Out[131...]

0.12895798683166504

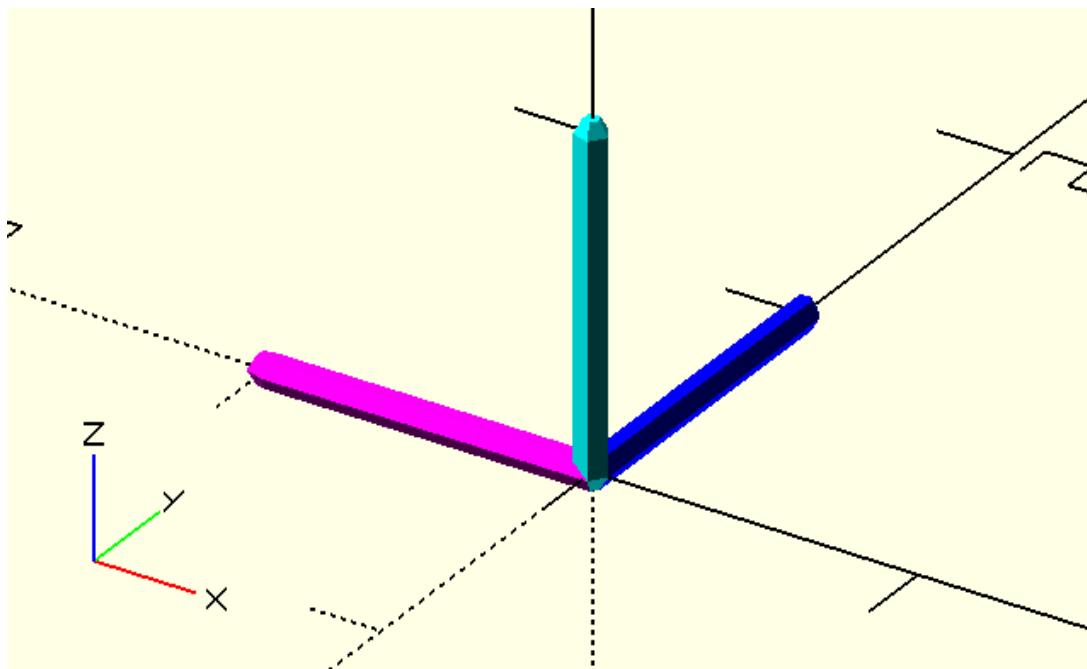
```
In [132...]
# example of defining normal vectors to a given vector 'v1'

v1=[0,1,0]
u1=v1/norm(v1)
ua=ua=array([0,0,-1]) if u1[2]==0 else array([0,-1,0]) if (u1==[0,0,1]).all() else array([-1,0,0]) if
v2=cross(u1,ua)
u2=v2/norm(v2)
u3=array(q(u2,u1,-90))

u1,u2,u3=array([u1,u2,u3]).tolist()
with open('trial.scad','w+') as f:
    f.write(f''''
        include<dependencies2.scad>

color("blue")p_line3d({[[0,0,0],u1]},.05);
color("magenta")p_line3d({[[0,0,0],u2]},.05);
color("cyan")p_line3d({[[0,0,0],u3]},.05);

    ''')
```



```
In [133]: with open('trial.scad', 'w+') as f:
    f.write(f'''  
  

cylinder (d=30,h=30); // hub.  

linear_extrude (height=30, twist=100, $fn=100)  

    for (a=[0:120:359]) rotate (a) translate ([15,-1]) square ([45,2]);  
  

...''')
```

ppplane

```
In [134]: # points projected on a plane
t0=time.time()

p0=random.random([10,3))*(20-(-20))+(-20)
loc=[0,10,1]
v1=[2,3,4]

sec=pts([[50/2,-50/2],[50,0],[0,50],[-50,0]])
plane1=translate(loc,o_solid(v1,sec,.001))

ip1=ppplane(p0,v1,loc)
lines1=array([p0,ip1]).transpose(1,0,2).tolist()

with open('trial.scad', 'w+') as f:
    f.write(f'''  
  

include<dependencies2.scad>
//p_line({sec},.05);

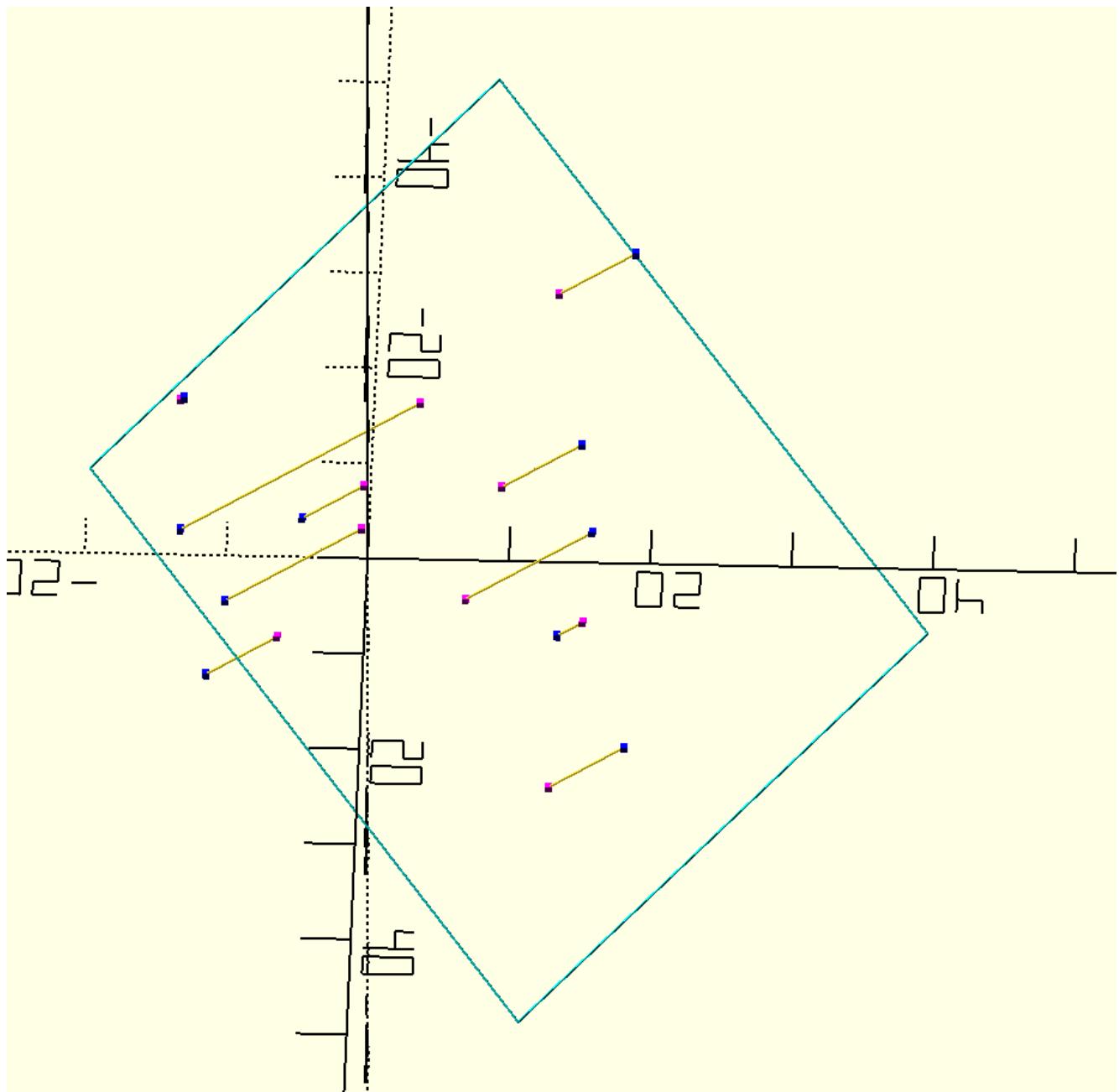
color("cyan")p_line3dc({plane1[0]},.1);
color("blue")points({p0.tolist()},.5);

color("magenta")points({ip1},.5);

for(p={lines1})p_line3d(p,.1);

...''')
t1=time.time()
t1-t0
```

Out [134]: 0.012050867080688477



ppesec

In [135...]: # points projected on an enclosed section in 3d space
t0=time.time()

```
p0=random.random([10,3])*(20-(-20))+(-20)
loc=[0,10,1]
v1=[2,3,4]

sec=pts([[50/2,-50/2],[50,0],[0,50],[-50,0]])
plane1=translate(loc,o_solid(v1,sec,.001))

ip1=ppesec(p0,plane1[0])[0]

lines1=array(ppesec(p0,plane1[0])).transpose(1,0,2).tolist()

with open('trial.scad','w+') as f:
    f.write(f'''

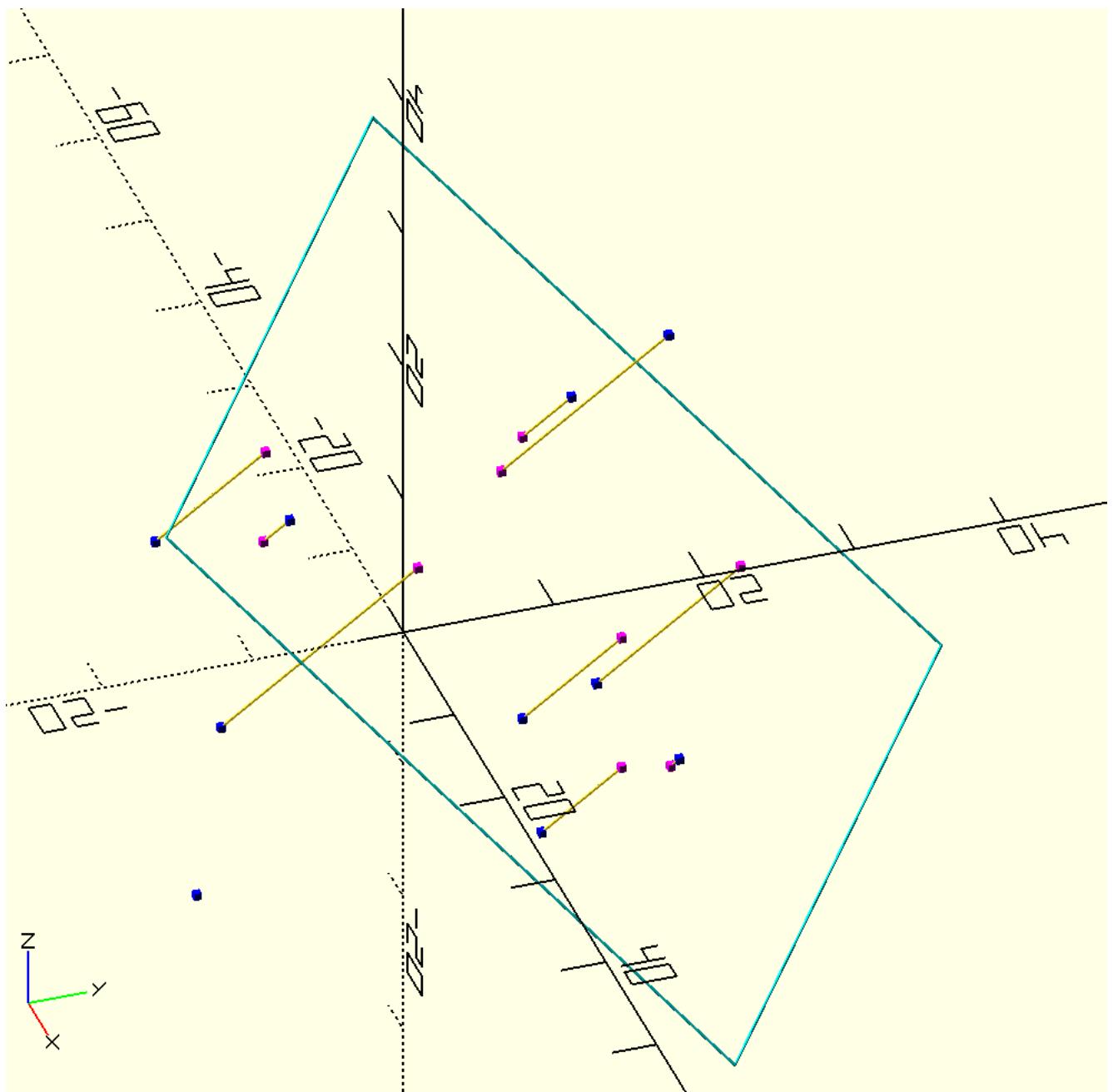
include<dependencies2.scad>
//p_line({sec},.05);

color("cyan")p_line3dc({plane1[0]},.1);
color("blue")points({p0.tolist()},.5);

color("magenta")points({ip1},.5);
for(p={lines1})p_line3d(p,.1);
```

```
...)  
t1=time.time()  
t1-t0
```

Out[135... 0.005750179290771484

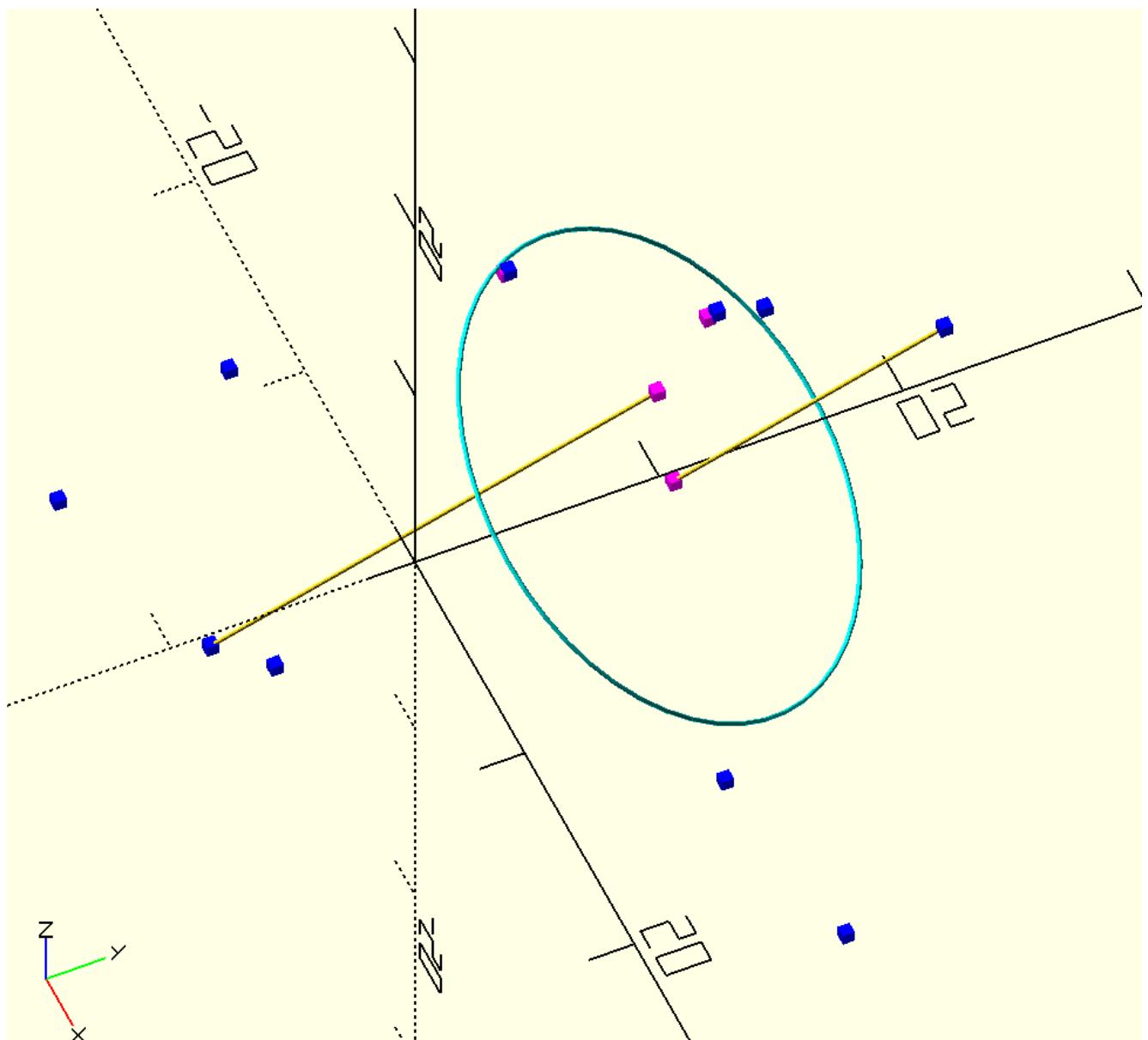


In [136... # another example of points projected on an enclosed section in 3d space
t0=time.time()

```
p0=random.random([10,3])*(20-(-20))+(-20)  
v1=[2,3,4]  
sec=circle(10)  
loc=[0,10,0]  
plane1=translate(loc,o_solid(v1,sec,.001))  
  
ip1=ppesec(p0,plane1[0])[0]  
lines1=cpo(ppesec(p0,plane1[0]))  
with open('trial.scad','w+') as f:  
    f.write(f'''  
  
include<dependencies2.scad>  
//p_line({sec},.05);  
  
color("cyan")p_line3dc({plane1[0]},.1);  
color("blue")points({p0.tolist()},.5);  
  
color("magenta")points({ip1},.5);  
for(p={lines1})p_line3d(p,.1);
```

```
...)  
t1=time.time()  
t1-t0
```

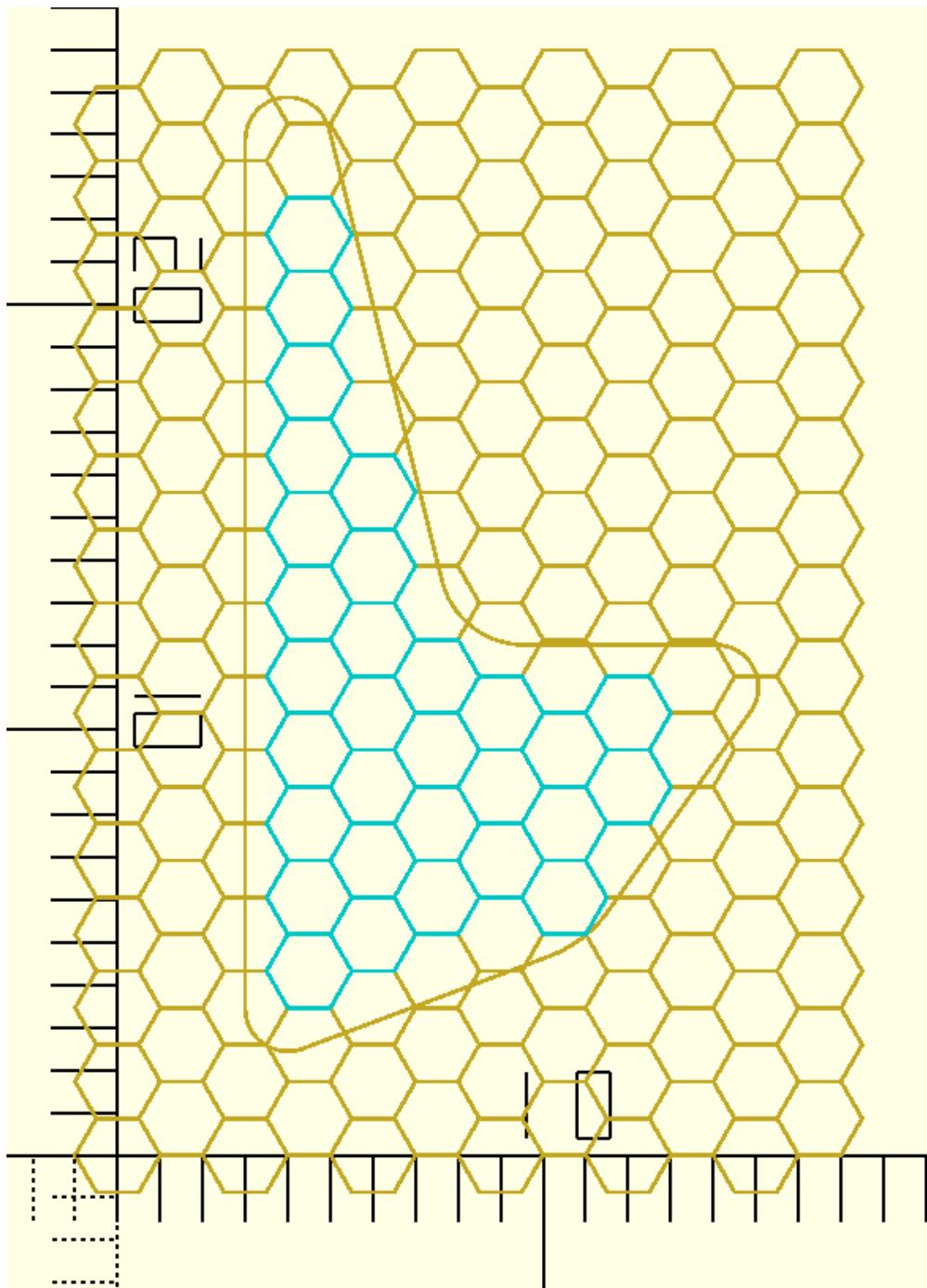
Out [136... 0.01401376724243164



honeycomb

In [137... # honeycomb structure

```
sec4=honeycomb(1,6,15)  
sec=corner_radius(pts1([[3,2,1],[8,3,3],[5,7,1],[-8,0,2],[-5,20,1]]),30)  
pnts1=[p for p in sec4 if len(pies1(sec,p))==6]  
  
with open('trial.scad','w+') as f:  
    f.write(f'''  
include<dependencies2.scad>  
  
for(p={sec4})p_line(p,.1);  
color("blue")p_line({sec},.1);  
color("cyan")for(p={pnts1})p_line(p,.1);  
  
...)
```



```
In [138]: # honeycomb structure with intersection option
t0=time.time()

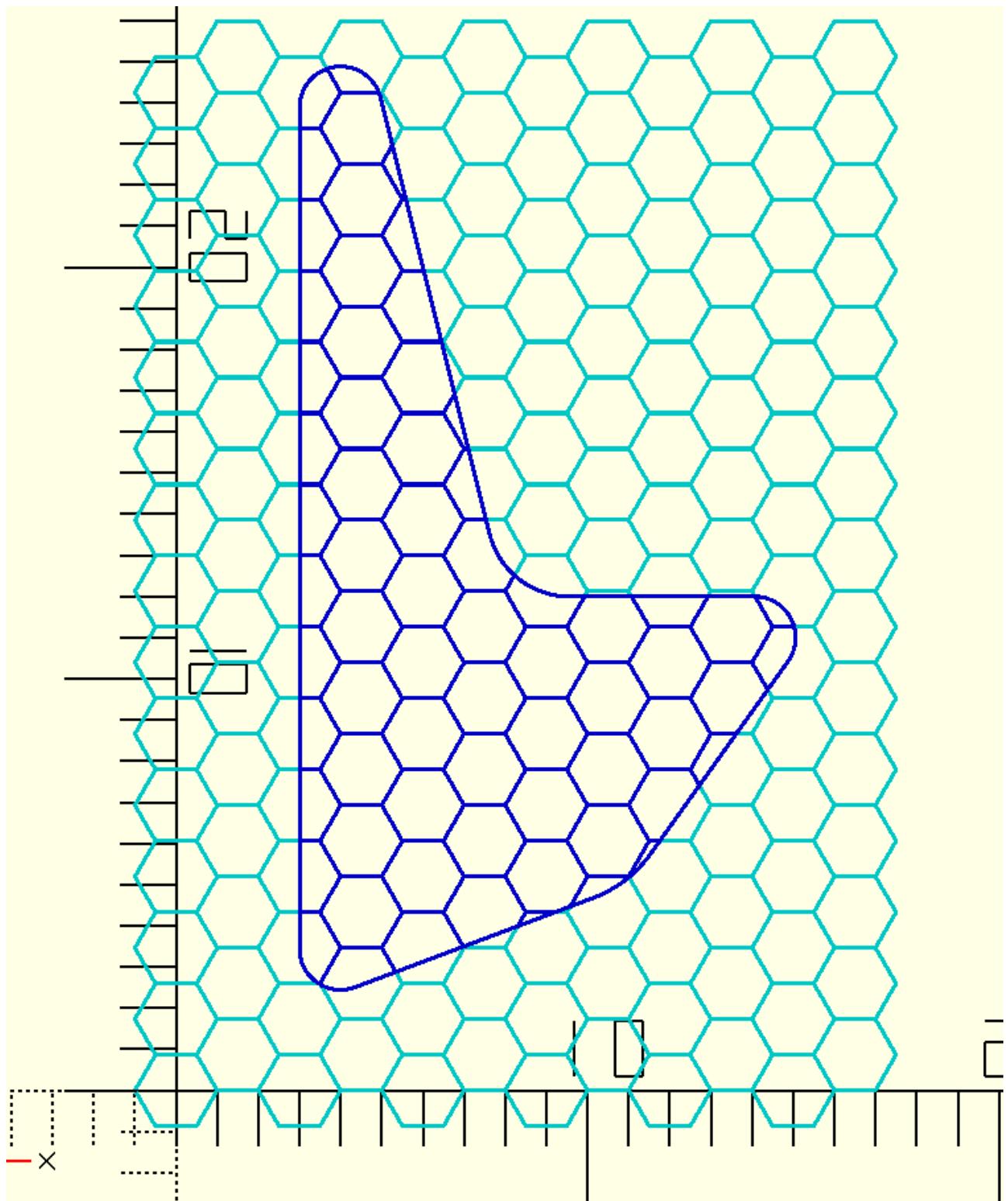
sec4=honeycomb(1,6,15)
sec=corner_radius(pts1([[3,2,1],[8,3,3],[5,7,1],[-8,0,2],[-5,20,1]]),30)
# pts1=[p for p in sec4 if len(pies1(sec,p))==6]

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>
color("cyan")linear_extrude(1)for(p={sec4})p_line(p,.1);

color("blue")
linear_extrude(1){{{
for(p={sec4}){{{
intersection(){{{
p_line(p,.1);
polygon({sec});
}}}
p_line({sec},.1);
}}}

...
t1=time.time()
t1-t0
```

Out [138... 0.023134231567382812



chimney-panel-support

In [139...]

```
# chimney panel support

sec=corner_radius(pts1([[0,5],[0,-5,3.5],[10,0,4],[2,1,1],[34,0,4],[1,3.5]]),10)
sec1=path_offset(sec,-3)
path=bezier([[-27.5,0,3]]+arc_3p_3d([[-27,0,3],[0,0,0],[27,0,3]],100)+[[27.5,0,3]],100)
sol=path_extrude_open(sec,path)
sol1=path_extrude_open(sec1,path[1:-1])
sec2=corner_radius(pts1([[0,0],[25,0,.5],[0,4,.5],[-1,3,.5],[-1.5,0,.5],[-2,-3,1],[-20.5,0]]),10)
sol2=translate([0,-17,1],path_extrude_open(sec2,path[47:-47]))
sec3=corner_radius(pts1([[-6,-5,5],[12,0,5],[0,10.0001,5],[-12,0,5]]),10)
sol3=o_solid([0,0,1],sec3,5,0,0,-37)
sec4=corner_radius(pts1([[0,0],[5,0,10],[7,7,10],[14,0]]),20)
sec5=path_offset(sec4,-3)
sec6=sec4+flip(sec5)
sol4=translate([0,-25,1],path_extrude_open(sec6,path[16:30]))
sol5=translate([0,-25,1],path_extrude_open(sec6,path[-30:-16]))
```

```

sol6=translate([12,-25,2],q_rot(['x90','z-90']),linear_extrude(sec6[20:-20],39))

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>
module chimeny_support(){
difference(){
{swp(sol)}
{swp(sol1)}
{swp(sol3)}
}

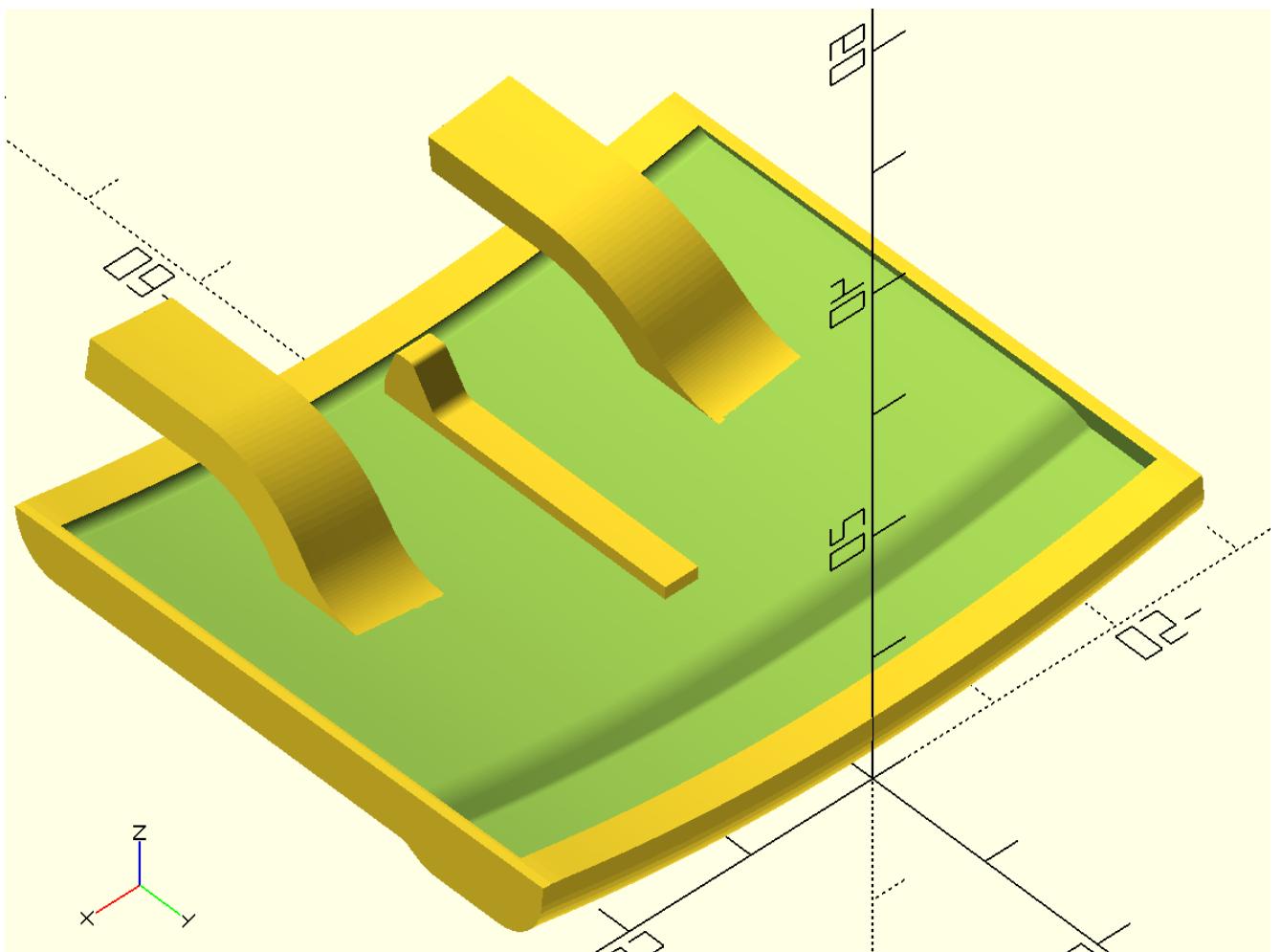
{swp(sol2)}

{swp(sol4)}
{swp(sol5)}

}>

difference(){
translate([0,0,28])
rotate([0,-90,0])
chimney_support();
//{swp(cut_plane([0,0,1],60,50,12,0,22))}
}
//support for 3d printing the part
//translate([-15,-55,0])
//cube([25,60,2]);
//translate([0,0,27])
//rotate([0,-90,0]){
////{swp(sol3)}
//{swp(sol6)}
//}}
''' )

```



offset_3d

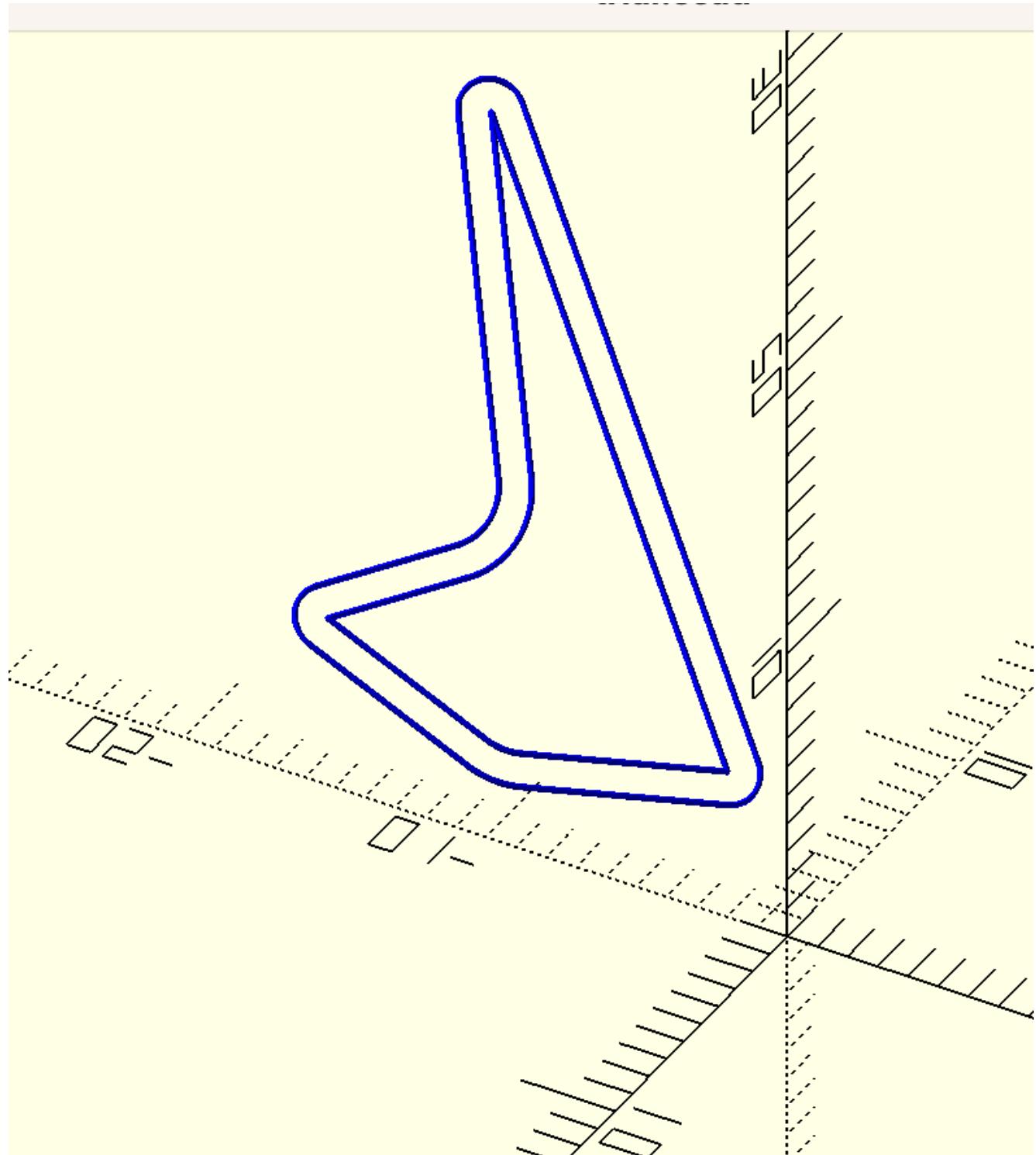
In [140...]

```
# example of offset_3d(sec,d)
sec=corner_radius(pts1([[3,2,1],[8,3,3],[5,7,1],[-8,0,2],[-5,20,1]]),30)
sec1=o_solid([2,3,4],sec,1,10)[0]
sec2=offset_3d(sec1,-1)

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>

color("blue")p_line3dc({sec1},.1);
color("blue")p_line3dc({sec2},.1);

...)
```



convert_3lines2fillet

In [141...]

```
# example of convert_3lines2fillet(pnt1,pnt2,pnt3,f=1.9,s=10)
t0=time.time()

sec=[[0,0],[5,0],[2.5,5]]
```

```

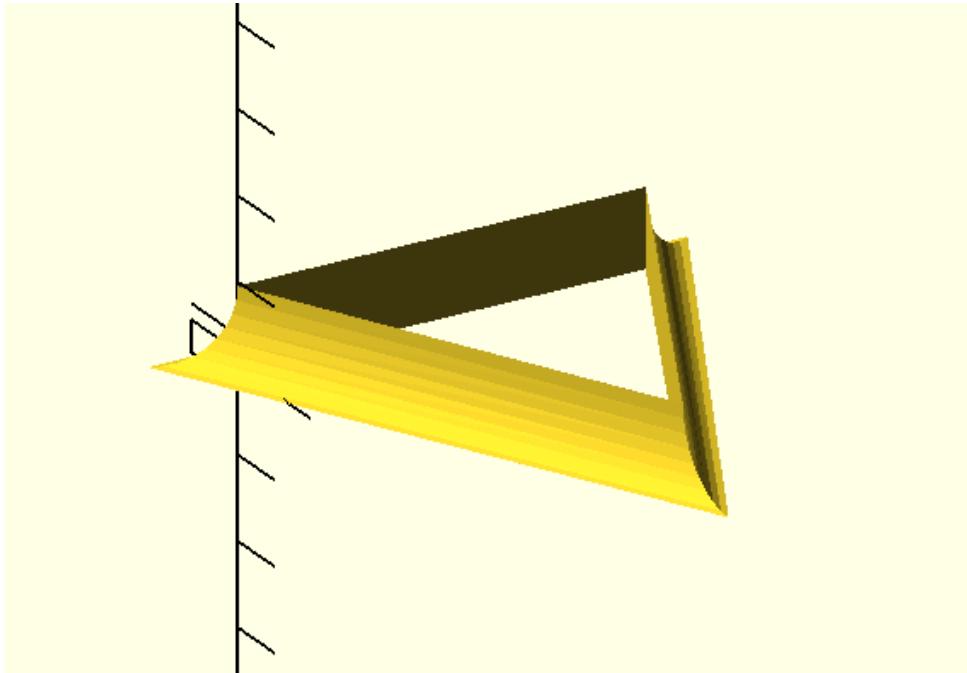
sol=o_solid([0,0,1],sec,1,10)
line1=sol[0]
line2=sol[1]
line3=offset_3d(line1,1)
fillet1=convert_3lines2fillet(line3,line2,line1)
fillet1=fillet1+[fillet1[0]]
# fillet1=flip(cpo(fillet1)[1:])
with open('trial.scad','w+') as f:
    f.write(f'''')
include<dependencies2.scad>

color("blue")p_line3dc({line1},.05);
color("cyan")p_line3dc({line2},.05);
color("magenta")p_line3dc({line3},.05);
{swp_c(fillet1)}

''')
t1=time.time()
t1-t0

```

Out [141]: 0.012671947479248047



sunflower

```

In [12]: # example of offset_3d when the section is not in 1 plane
# example of path_extrude2msec(sec_list, path)
t0=time.time()

t=1.1 # thickness
s=10 # number of sides of the star
d=20 # outer diameter of the star
h=50 # height of the star prism
cir1=circle(d,s=(s+1))
cir2=c3t2(q_rot([f"z{360/(s+1)/2}"],circle(d/4,s=(s+1))))
sec1=array(c2t3([cir1,cir2]))
sec1=sec1.transpose(1,0,2).reshape(-1,3)
sec1=sec1
sec1=[(sec1[i]+[0,0,1]).tolist() if i%2==0 else (sec1[i]+[0,0,t]).tolist() for i in range(len(sec1)) ]
sec1=m_points(corner_radius(sec1,20),1.1)
sol1=linear_extrude(sec1,h)

line1=corner_radius(pts1([[-20,0],[20,15,30],[20,-15]]),40)
line2=cytz(line1)
surf1=surf_extrude(line1,line2)

ip1=ip_surf(surf1,sol1)
ip2=offset_3d(ip1,-2.01)

surf4=[ip2,ip1,translate([0,0,-t],ip1),translate([0,0,-t],ip2),ip2]
avg2=array(surf4).mean(0).mean(0)

```

```

sec3=circle(5)
path3=m_points_o(corner_radius(pts1([[-4,0],[0,15,2],[3.5,4,2],[0,1]]),5),.5)

sol3=prism(sec3,path3)

path4=q_rot(['z90','x90'],cytz([[i,3*sin(d2r(i*20))]] for i in linspace(0,20,44)])
sol4=sol2path(sol3,path4)
v1==array(nv(sol4[-1]))
avg1=array(sol4[-1]).mean(0)
surf4=translate(-avg2+[0,0,-.5],surf4)
surf4=sol2vector(v1,surf4,avg1)

arc1=arc_2p([0.01,0],[4.65,0],3,-1)
arc2=[q_rot(['z{i}'],arc1) for i in arange(0,360,360/50)]
arc2p=translate([0,0,5],arc2)
arc2=array([arc2,arc2p]).transpose(1,0,2,3)

arc3=arc_2p([0.01,0],[4.65,0],3,1)
arc4=[q_rot(['z{i}'],arc3) for i in arange(0,360,360/50)]
arc4p=translate([0,0,5],arc4)
arc4=array([arc4,arc4p]).transpose(1,0,2,3)

arc5=circle(4.5)
path5=arc_2p([0,0],[-4.5,2],10,-1)
sol5=prism(arc5,path5)

ip1=[ip_sol2sol(sol5,p) for p in arc2]
ip1=[[p1[-1] for p1 in p] for p in ip1]
ip2=[ip_sol2sol(sol5,p) for p in arc4]
ip2=[[p1[-1] for p1 in p] for p in ip2]

ip1=sol2vector(v1,ip1,avg1)
ip2=sol2vector(v1,ip2,avg1)
sol5=sol2vector(v1,sol5,avg1)

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>

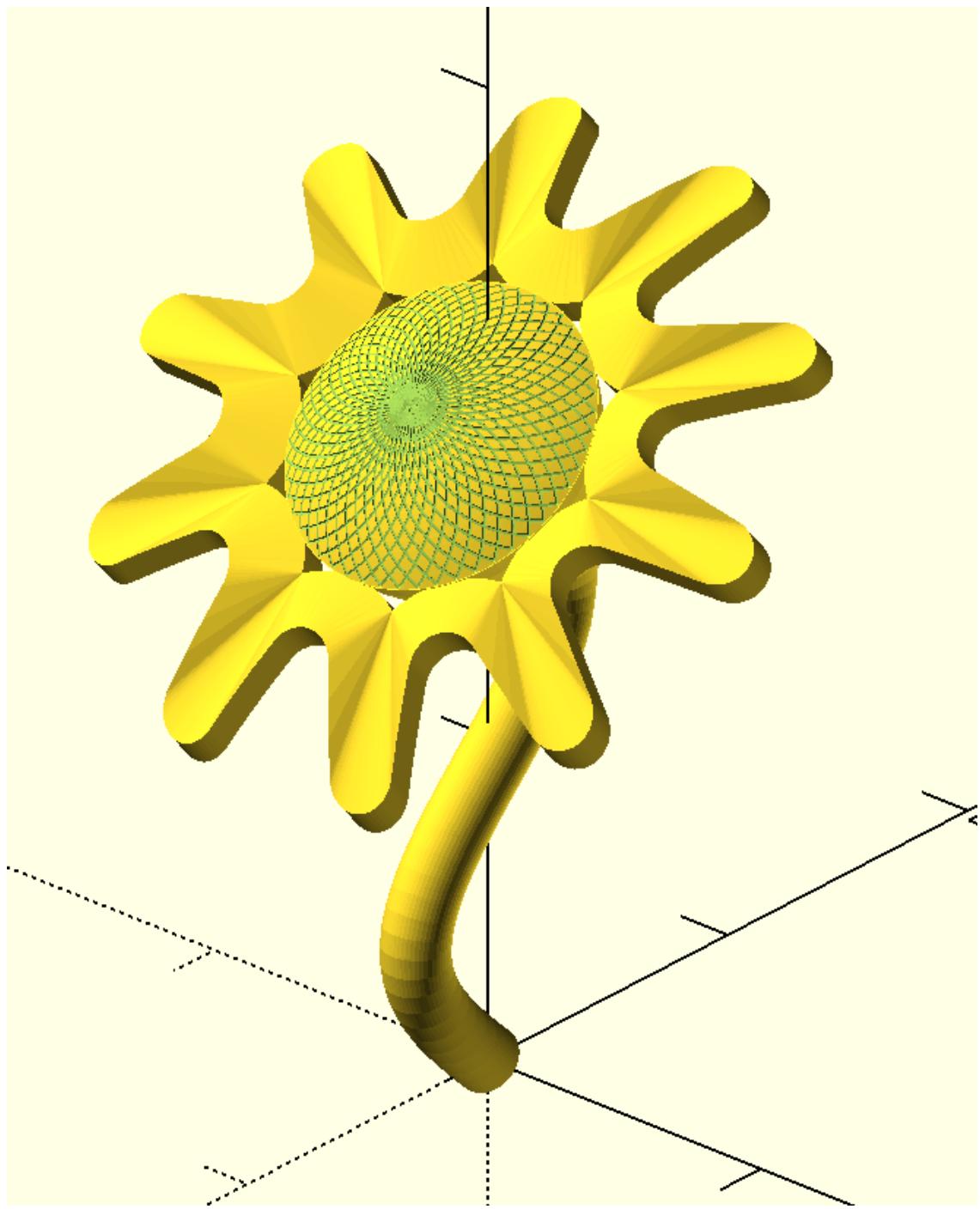
{swp_c(flip(surf4))}
{swp(sol4)}
//color("blue")p_line3dc({sol4[-1]},.2);
//color("magenta")p_line3d({path4},.2);

difference(){
{swp(flip(sol5))}
for(p={ip1})p_line3d(p,.05,rec=1);
for(p={ip2})p_line3d(p,.05,rec=1);
}

''' )
t1=time.time()
t1-t0

```

Out[12]: 2.993295669555664



sol2vector

In [143]:

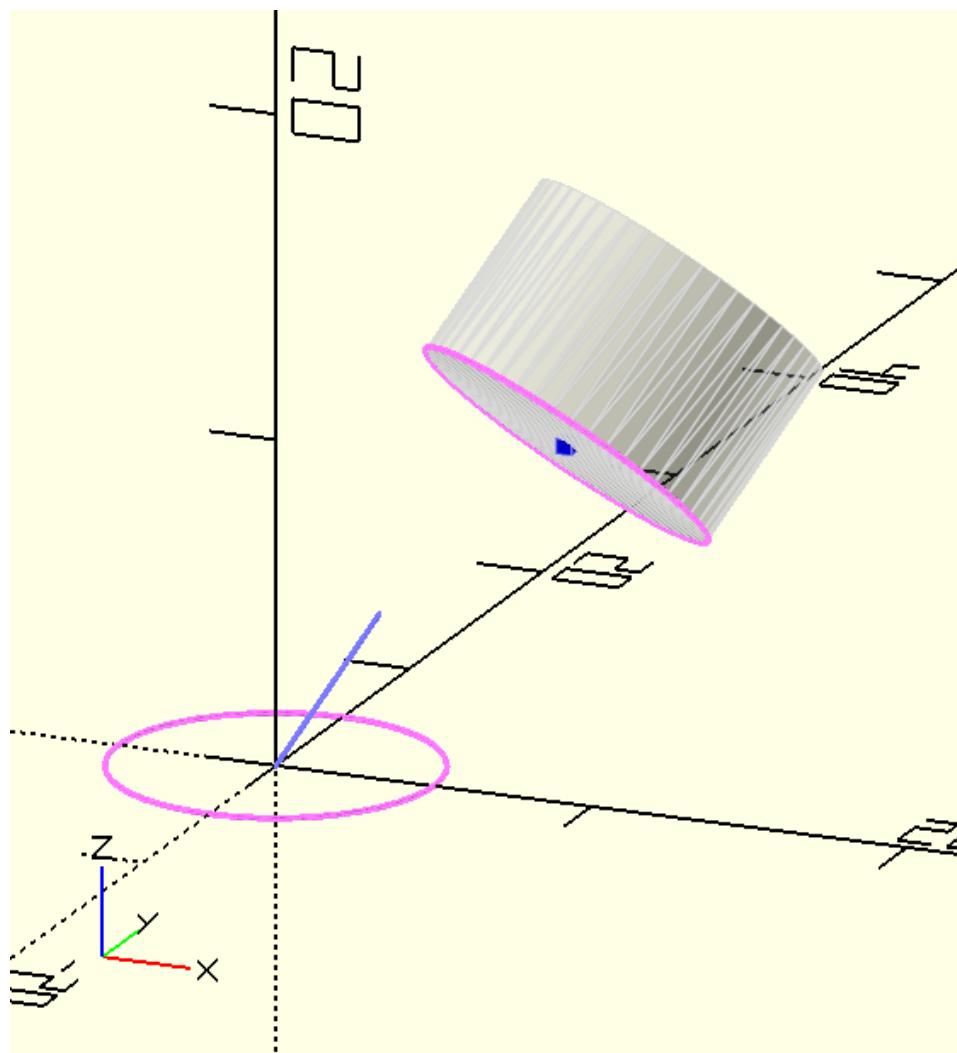
```
# checking orientation of solid w.r.t. vector
t0=time.time()

sol=linear_extrude(circle(5),6)
v1=[2,3,4]
loc=[5,10,7.5]
sol1=sol2vector(v1,sol,loc)

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>
%{swp(sol1)}
color("blue")points({[loc]},.5);
color("magenta")p_line3dc({sol[0]},.1);
color("magenta")p_line3dc({sol1[0]},.1);
color("blue")p_line3d({[[0,0,0],v1]},.1);

''')
t1=time.time()
t1-t0
```

Out [143... 0.008891105651855469



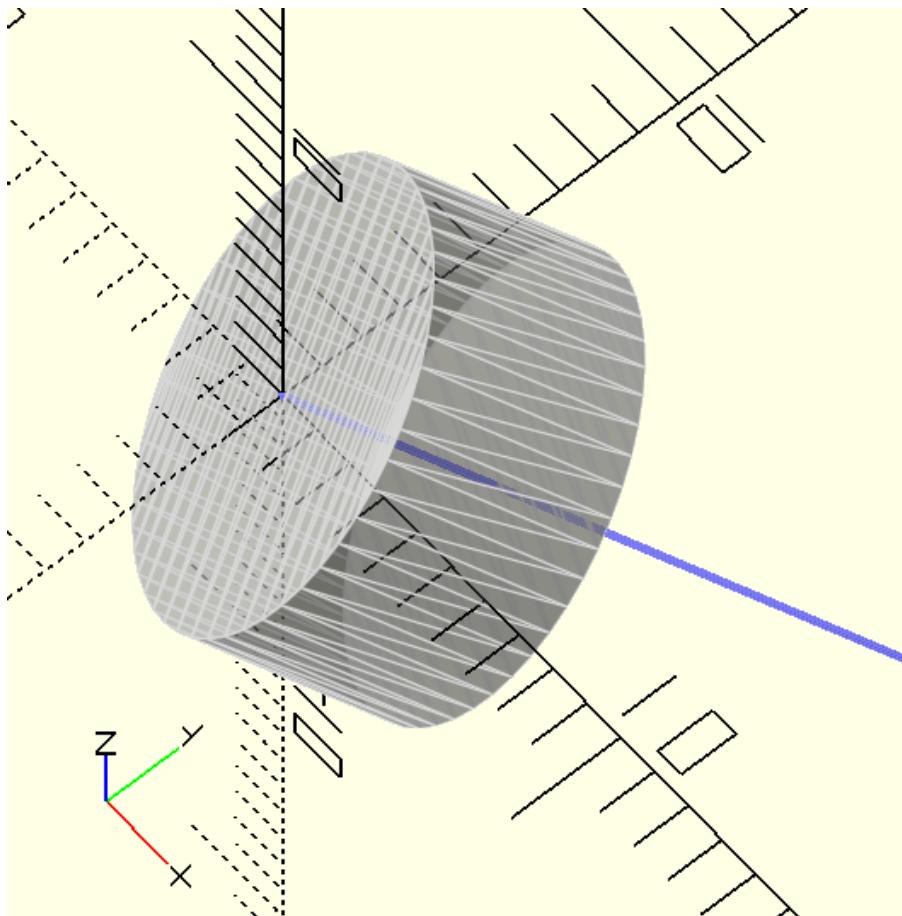
In [144... # checking orientation of solid w.r.t. vector
t0=time.time()

```
sec=circle(5)
v1=(array([1,1,-1])*10).tolist()
sol=o_solid(v1,sec,5,0,0,0)

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>
%{swp(sol)}
color("blue")p_line3d({[[0,0,0]},v1],.1);

''')
t1=time.time()
t1-t0
```

Out [144... 0.010133028030395508



```
In [13]: # difficult fillet
t0=time.time()

sec1=circle(10,s=6)
pent1=circle(7,s=6)
pent2=c3t2(q_rot([f'z{360/5/2}'],circle(3.5,s=6)))
sec2=concatenate(cpo([pent1]+[pent2])).tolist()

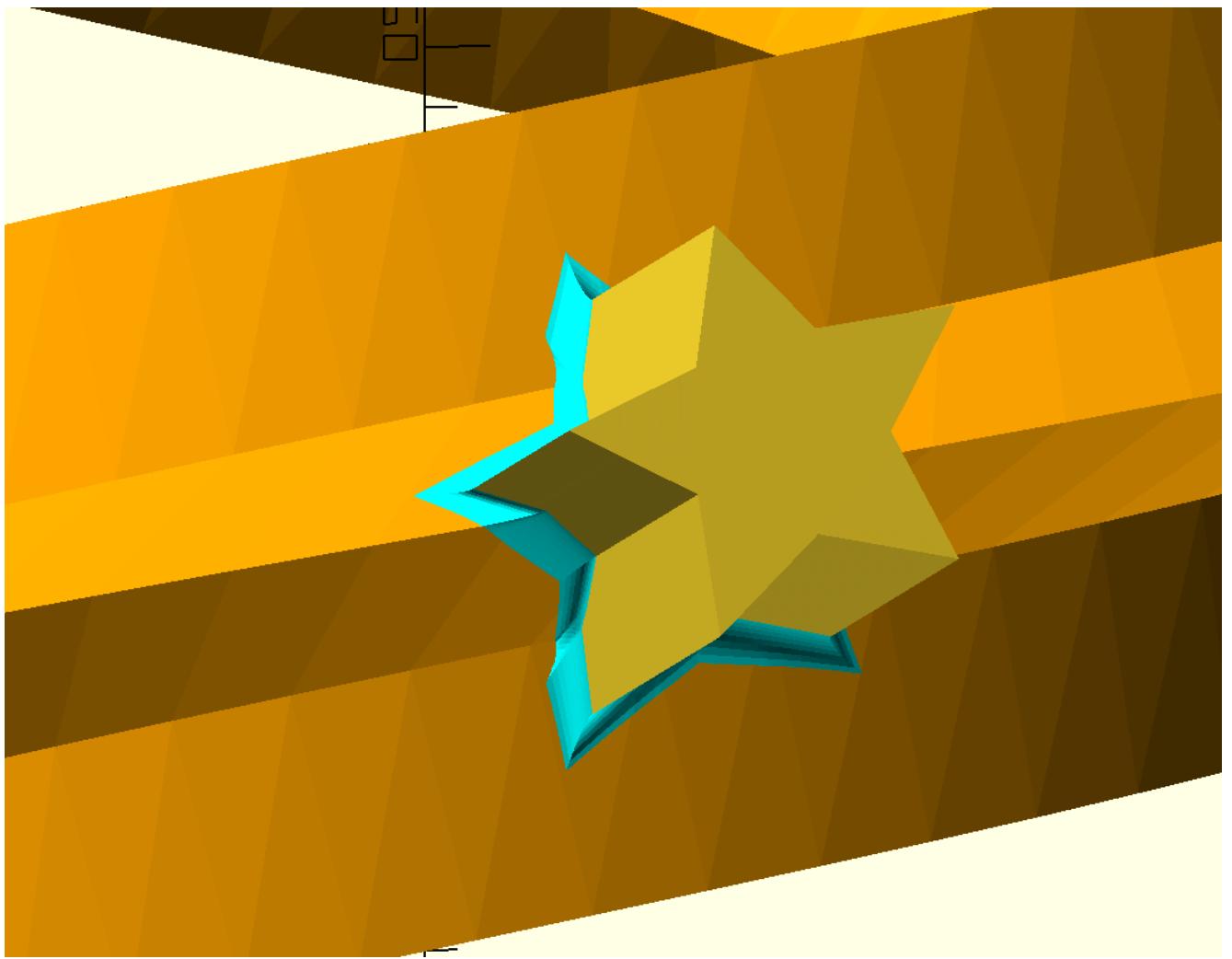
sec3=offset(sec2,-1.5)
sec4=offset(sec3,1)
path1=helix(20,30,1,5)
path2=[[0,0,10],[-30,20,13]]
sol=path_extrude_open(sec2,path1)
sol1=path_extrude_open(m_points1(sec3,50),path2)
sol2=path_extrude_open(m_points1(offset(sec3,.3),50),path2)
v1=array(path2[1])-array(path2[0])
u1=v1/norm(v1)
ip1=[ip_sol2line(sol,p)[-1] for p in cpo(sol1)]
ip2=translate(u1,ip1)
ip3=[ip_sol2line(sol,p)[-1] for p in cpo(sol2)]
fillet1=convert_3lines2fillet(ip3,ip2,ip1)
fillet1=fillet1+[fillet1[0]]
with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>

color("orange"){swp(sol)}
%{swp(sol1)}

//color("blue")p_line3dc({ip1},.05);
//color("blue")p_line3dc({ip2},.05);
//color("blue")p_line3dc({ip3},.05);
color("cyan"){swp(fillet1)}

''')
t1=time.time()
t1-t0
```

Out [13]: 4.588769912719727



ip_sol2sol

```
In [146]: # example of function ip_sol2line(sol,line)
t0=time.time()

sec=circle(10)
path=corner_radius(pts1([[-10+.1,0],[12,0],[-2,0,2],[0,10,3],[-10,0]]),5)
sol=prism(sec,path)

sol1=o_solid([1,0,1],circle(3),20,-5)

pa=ip_sol2sol(sol,sol1)
ip1=[p[-1] for p in pa]

ip2=[p[0] for p in pa]

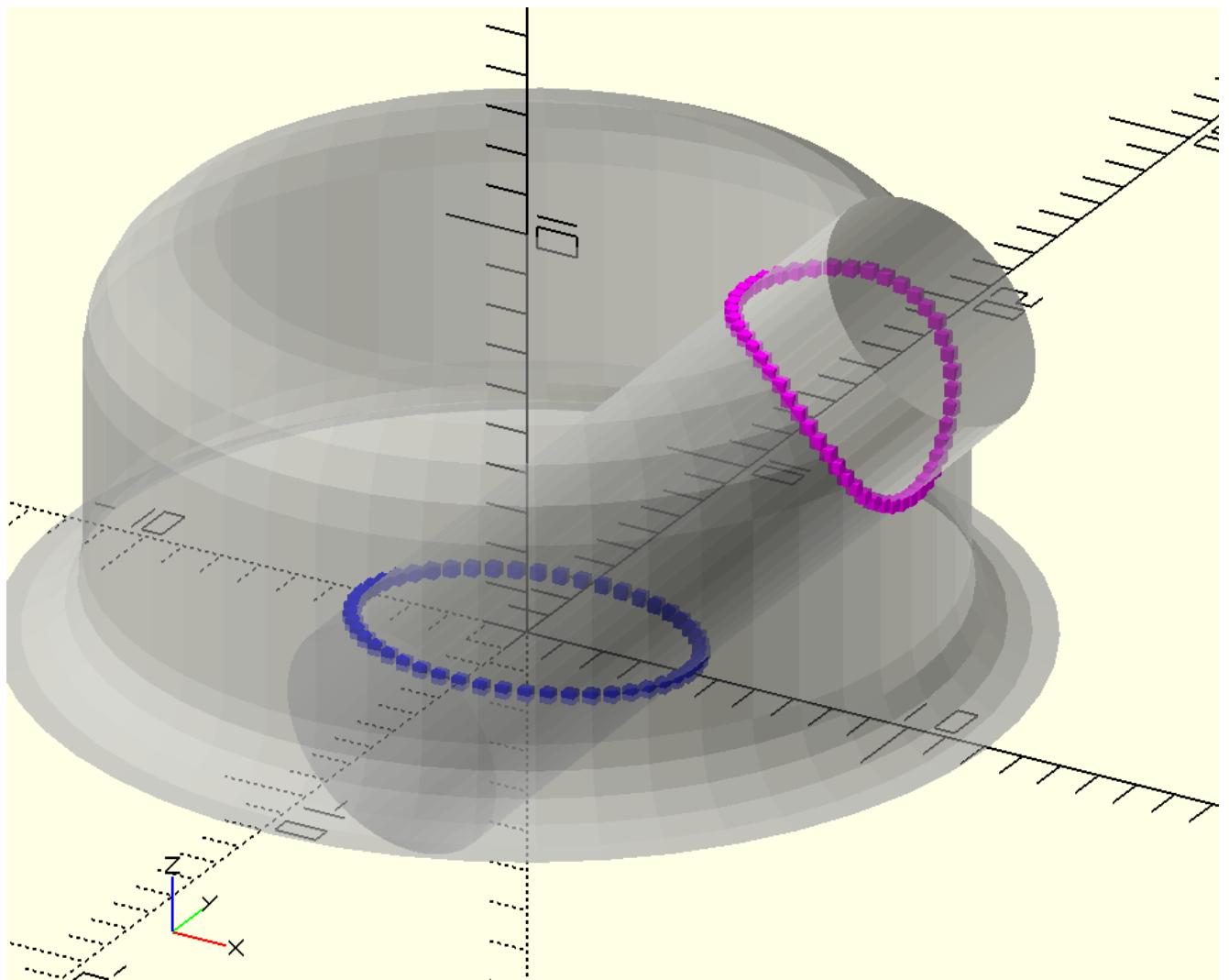
with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>

${swp(sol)}
${swp(sol1)}

color("magenta")points({ip1},.3);
color("blue")points({ip2},.3);

''')
t1=time.time()
t1-t0
```

Out [146]: 0.09132504463195801



ip_sol2line

In [147]:

```
# example of function ip_sol2line(sol,line)
t0=time.time()

sec=circle(10)
path=corner_radius(pts1([[-10+.1,0],[12,0],[-2,0,2],[0,10,3],[-10,0]]),5)
sol=prism(sec,path)

line=ls([[0,0,-1],[20,20,10]],10)

ip1=ip_sol2line(sol,line)[-1]
ip2=ip_sol2line(sol,line)[0]

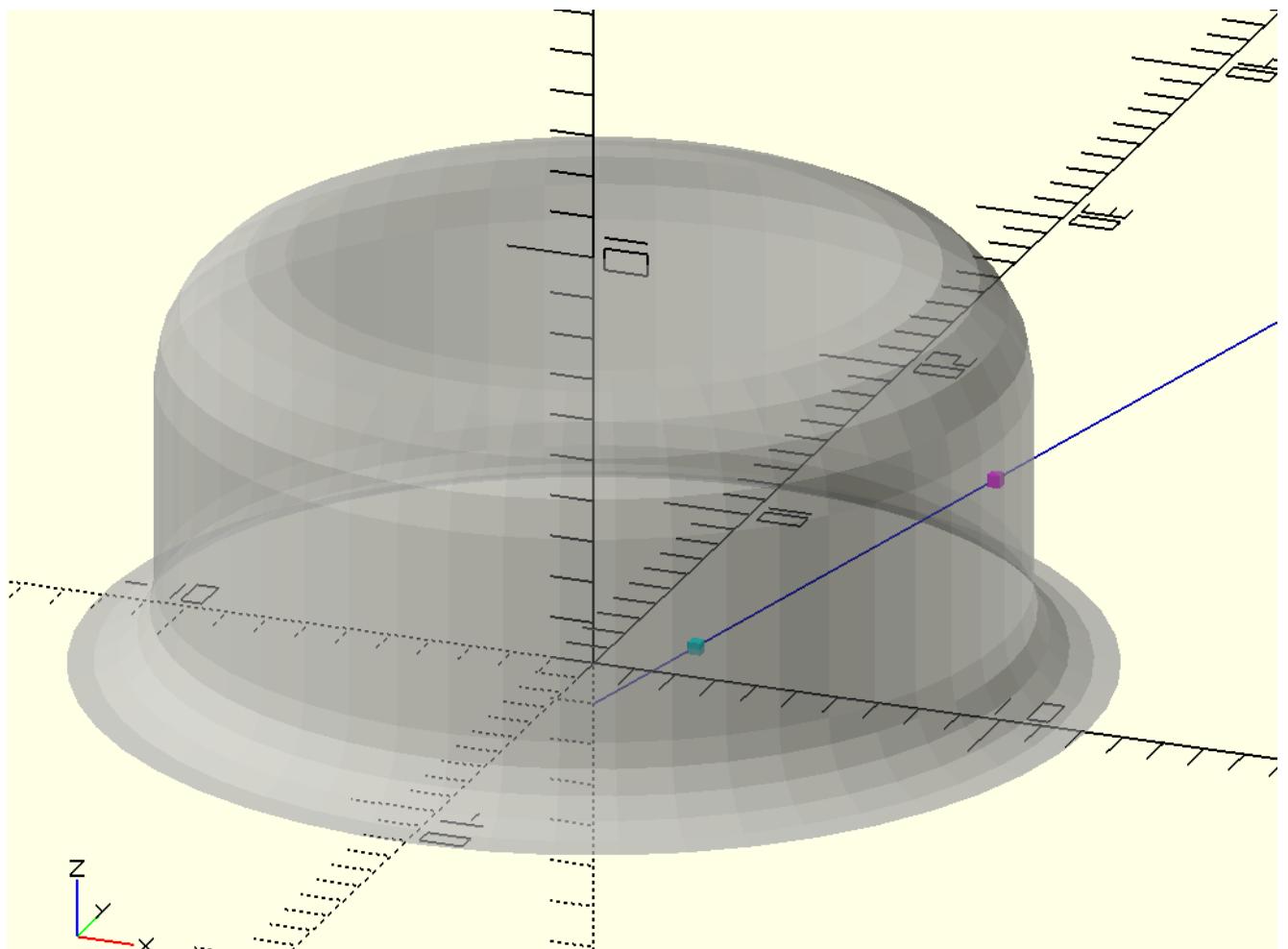
with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>
%{swp(sol)}

color("blue")p_line3dc({line},.05);
color("magenta")points({[ip1]},.3);
color("cyan")points({[ip2]},.3);

''')
t1=time.time()
t1-t0

ip1
```

Out [147... [7.1354892496692255, 7.1354892496692255, 2.924519087318074]



In [148... t0=time.time()

```
c1=l_cir_ip([[0,0],[0,5]],circle(2,[1,2.5]))
sec1=arc_long_2p(c1[0],c1[1],2,-1,50)
sol1=o_solid([0,0,1],sec1,5)

c1=l_cir_ip([[0,0],[0,5]],circle(2.3,[1,2.5]))
sec1=arc_long_2p(c1[0],c1[1],2.3,-1,50)
sol2=o_solid([0,0,1],sec1,5)

c1=l_cir_ip([[-0.3,0],[-0.3,5]],circle(2,[1,2.5]))
sec1=arc_long_2p(c1[0],c1[1],2,-1,50)
sol3=o_solid([0,0,1],sec1,5.6,-.3)
lines=array([sol1,sol3,sol2]).transpose(1,0,2,3)[1]
a,b,c=array([lines[0],lines[1],lines[2]]).tolist()
lines1=array([sol1,sol3,sol2]).transpose(1,0,2,3)[0]
h,i,j=array([lines1[0],lines1[1],lines1[2]]).tolist()
k,l,m=a+flip(h),b+flip(i),c+flip(j)

fillet1=convert_3lines2fillet(m,l,k)
fillet1=translate([0,5,0],q_rot(['z90'],fillet1+[fillet1[0]]))

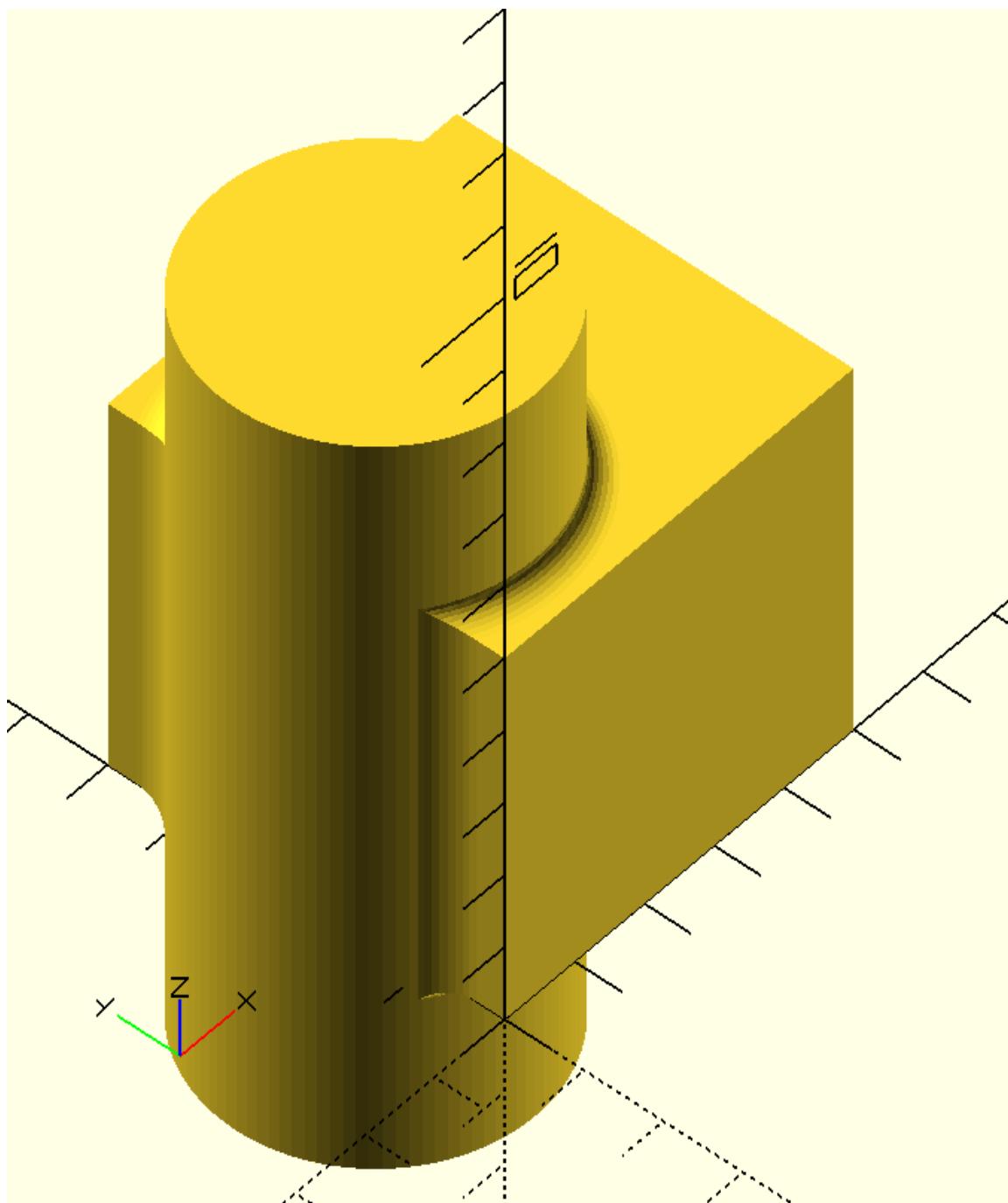
with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>

%cube(5);
%translate([1,2.5,-2.5])cylinder(r=2,h=10,$fn=100);

{swp(fillet1)}
translate([0,5,0])
rotate([0,0,90]){
color("magenta")p_line3dc({k},.02);
color("magenta")p_line3dc({l},.02);
color("magenta")p_line3dc({m},.02);
}
'''')
```

```
t1=time.time()  
t1-t0
```

Out [148]: 0.07036113739013672



```
In [14]: t0=time.time()  
sec1=circle(10,s=6)  
pent1=circle(7,s=6)  
pent2=c3t2(q_rot([f'z{360/5/2}'],circle(3.5,s=6)))  
sec2=concatenate(cpo([pent1]+[pent2])).tolist()  
sec2=corner_radius(array(c2t3(sec2))+[0,0,.5],15)  
sec3=concatenate(cpo([pent1]+[pent2])).tolist()  
sec3=offset(sec3,-1)  
sec3=corner_radius(array(c2t3(sec3))+[0,0,.2],10)  
path1=helix(20,30,1,5)  
path2=[[0,0,10],[-30,20,13]]  
sol=path_extrude_open(sec2,path1)  
sol1=path_extrude_open(equidistant_pathc(sec3,300),path2)  
  
sol2=sol[22:33]  
sol3=slice_sol(sol1,20)  
sol4=path_extrude_open(equidistant_pathc(offset(sec3,1),300),path2)  
sol4=slice_sol(sol4,20)  
  
i_p1=ip_sol2sol(sol2,sol3[12:17])  
i_p1=[p[0] for p in i_p1]  
i_p2=i_p_p(sol3,i_p1,1)  
i_p3=ip_sol2sol(sol2,sol4[12:17])
```

```

i_p3=[p[0] for p in i_p3]
i_p1,i_p2,i_p3=align_sol_1([i_p1,i_p2,i_p3])
fillet1=convert_3lines2fillet_closed(i_p3,i_p2,i_p1)
# fillet1=fillet_sol2sol(sol2,sol3[12:16],1,s=20)

with open('trial.scad','w+') as f:
    f.write(f''''
include<dependencies2.scad>
//color("blue")for(p={sol3[12:16]})points(p,.1);

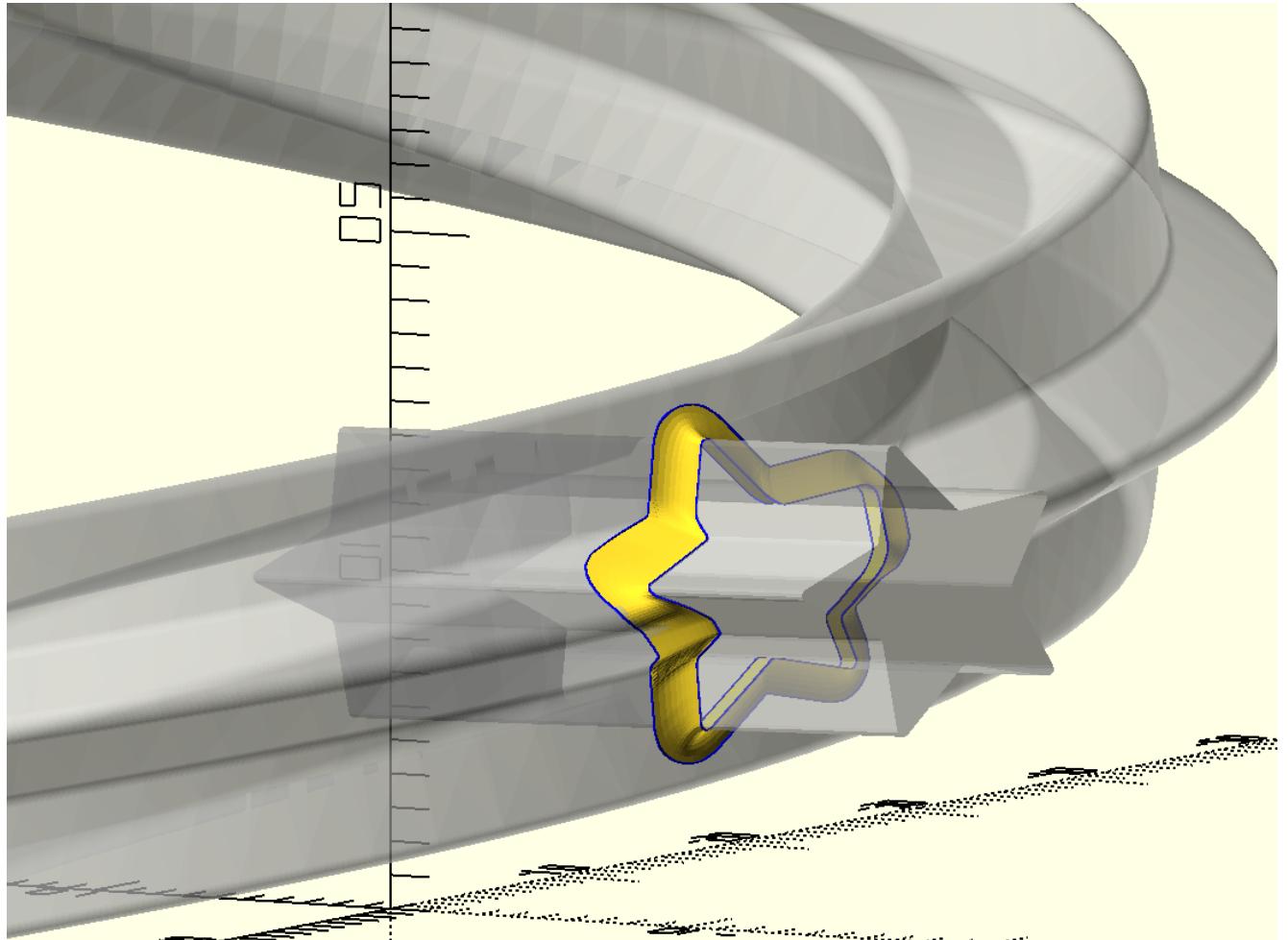
{swp(sol)}
{swp(sol3)}
//color("blue")for(p={sol4[12:16]})p_line3dc(p,.1,rec=1);
color("blue")p_line3dc({i_p1},.05,rec=1);
color("blue")p_line3dc({i_p2},.05,rec=1);
color("blue")p_line3dc({i_p3},.05,rec=1);

{swp_c(fillet1)}


'''')
t1=time.time()
t1-t0

```

Out[14]: 2.2179057598114014



equidistant_path

In [150...]

```

t0=time.time()
path=helix(15,20/10,10,5)
# path1=[[0,0,p[2]] for p in path]
path1=helix(0,20/10,10,5)
sec=circle(10)
path2=arc(10,-90,90,[-10+.1,10],50)
sp1=prism(sec,path2)
sol1=[path1,path]

```

```

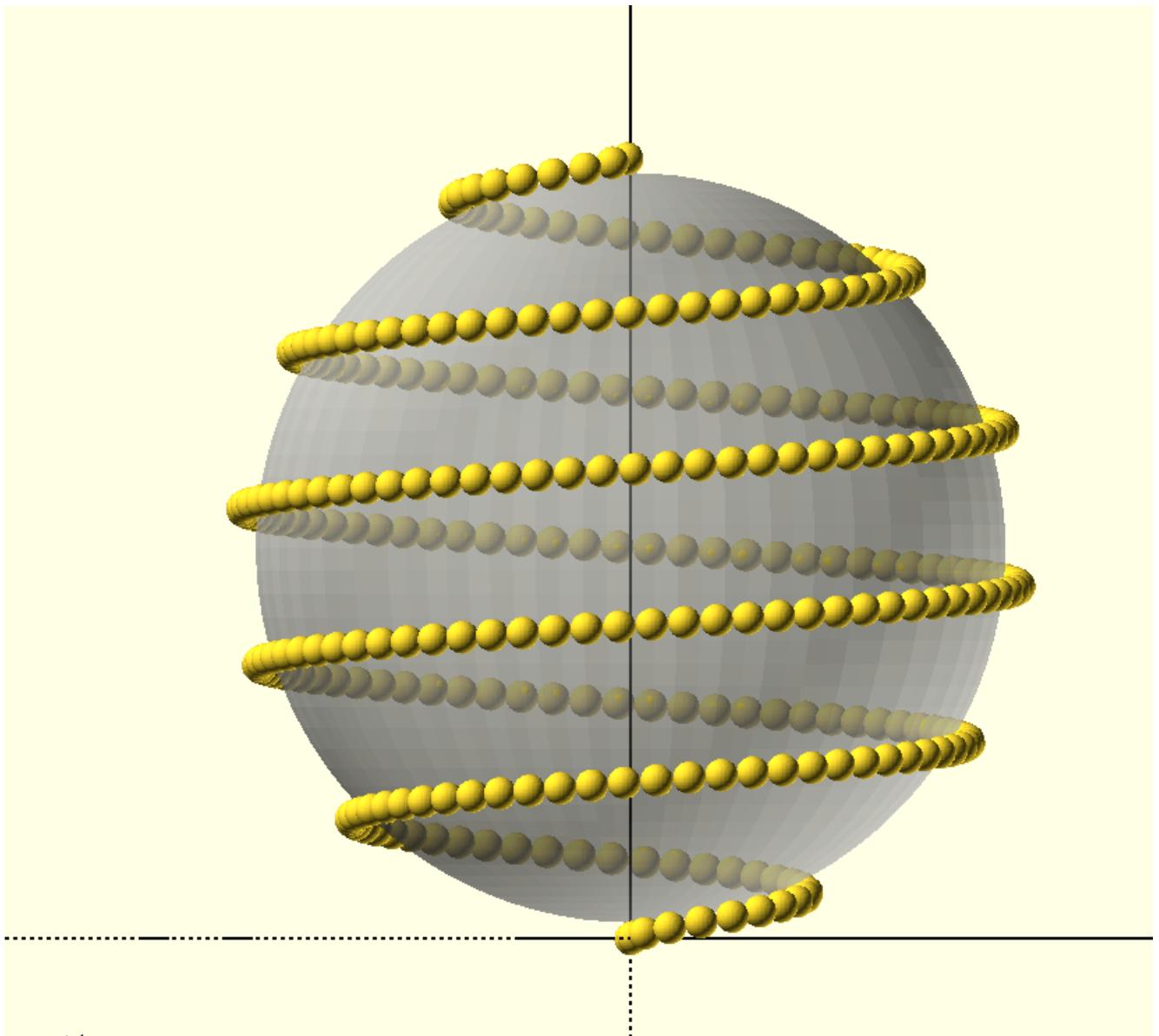
i_p1=ip_sol2sol(sp1,sol1)
i_p1=[p[0] for p in i_p1]
i_p1=equidistant_path(i_p1,500)
d=l_len([i_p1[0],i_p1[1]])
sp2=translate([0,0,10],sphere(10-d/2))
sp3=sphere(d/2)

with open('trial.scad','w+') as f:
    f.write(f'''
        include<dependencies2.scad>
        %{swp(sp2)}
        color("blue")p_line3d({i_p1},.05);
        for(p={i_p1})translate(p){swp(sp3)}

    ''')
t1=time.time()
t1-t0

```

Out [150]: 1.2135868072509766



iterative-approach-towards-creating-fillets

In [4]: # iterative approach towards creating fillets

```

t0=time.time()
sec1=circle(10,s=6)
pent1=circle(7,s=6)
pent2=c3t2(q_rot(['f' z{360/5/2}]),circle(3.5,s=6)))
sec2=concatenate(cpo([pent1]+[pent2])).tolist()
sec2=corner_radius(array(c2t3(sec2))+[0,0,.1],5)
sec3=concatenate(cpo([pent1]+[pent2])).tolist()
sec3=offset(sec3,-1)

```

```

sec3=corner_radius(array(c2t3(sec3))+[0,0,.1],5)
path1=helix(20,30,1,5)
path2=[[0,0,10],[-30,20,13]]
sol=path_extrude_open(sec2,path1)
sol1=path_extrude_open(sec3,path2)
sol2=sol[25:35]
a=array([[1-cos(d2r(i)),1-sin(d2r(i))] for i in linspace(0,90,90)])*1
b=[offset_sol(sol2,i,1) for i in a[:,0].round(2)]
c=[offset_sol(sol1,i) for i in a[:,1].round(2)]
```

with open('trial.scad','w+') **as** f:

for i **in** range(len(b)):

f.write(f'''

intersection(){

{swp(b[i])}

{swp(c[i])}

}}

'''')

f.write(f'''

include<dependencies2.scad>

color("cyan")

{swp(sol)}

color("blue")

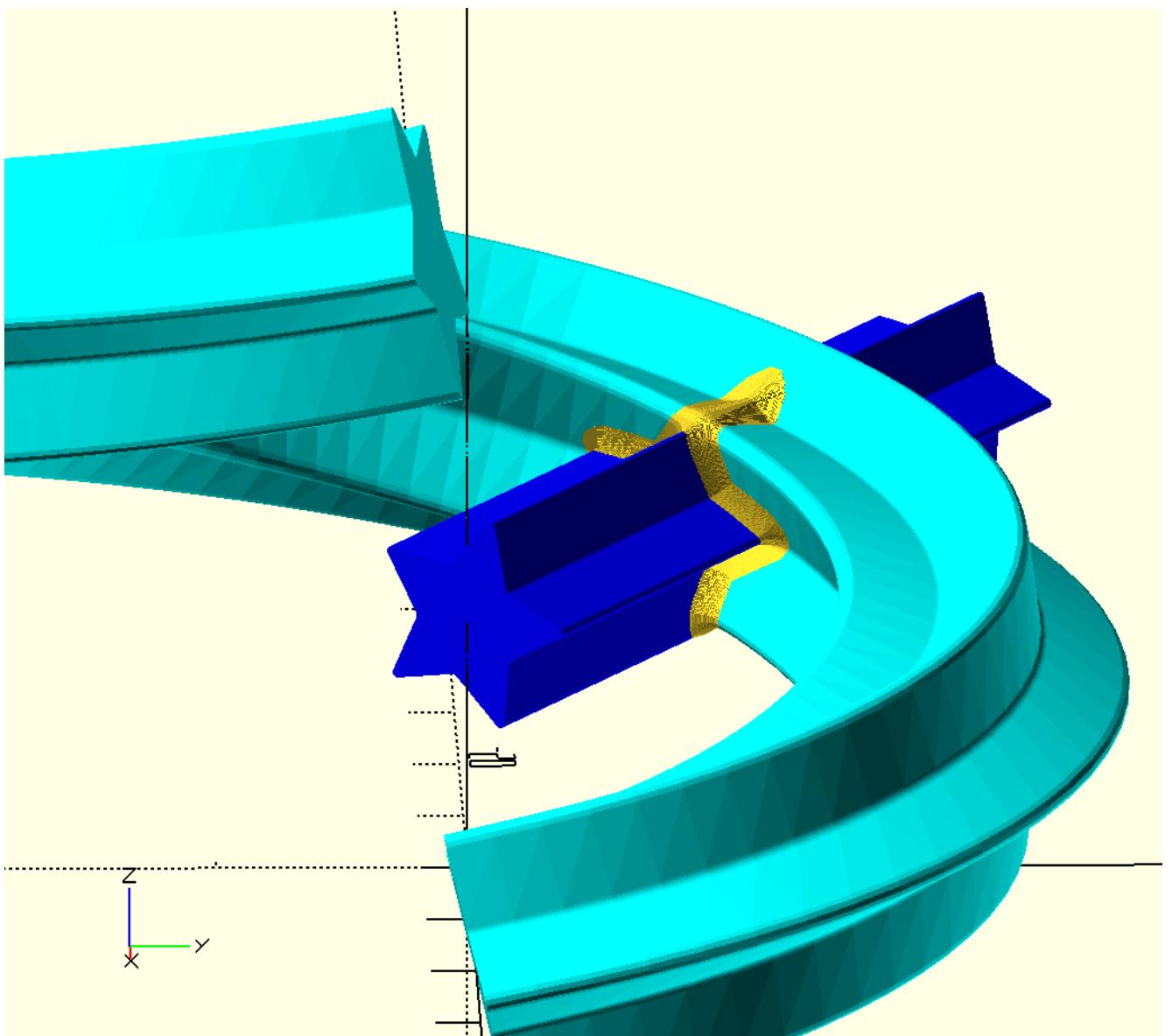
{swp(sol1)}

'''')

t1=time.time()

t1-t0

Out[4]: 7.65777587890625



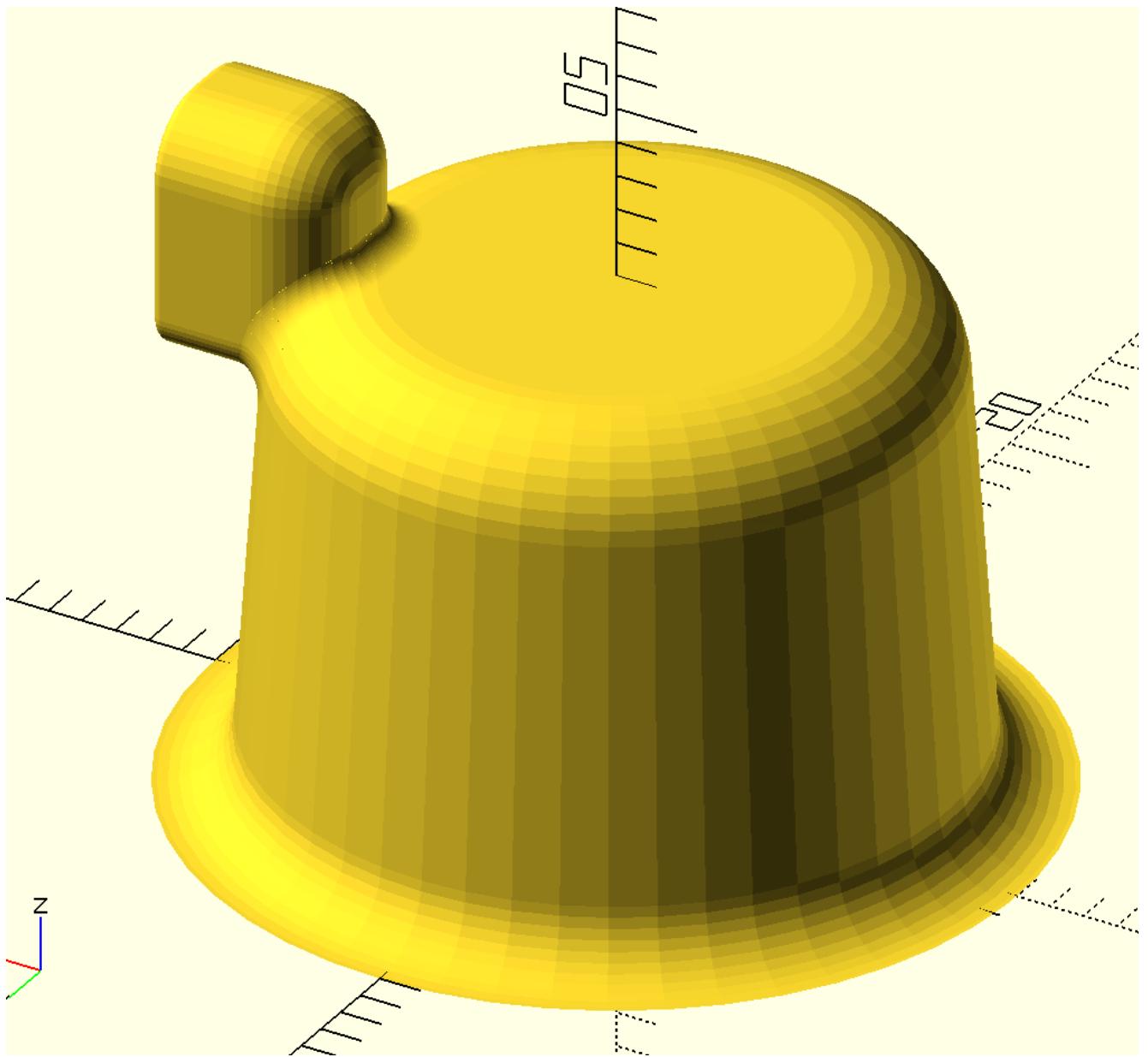
In [152...]: # iterative approach for creating fillets, example-2

```
t0=time.time()
sec=circle(10)
path=corner_radius(pts1([[2,0],[-2,0,2],[-1,15,3],[-8,0]]),10)
sol1=q_rot(['z90'],prism(sec,path))

sec1=corner_radius(pts1([[0,0,1],[5,0,1],[0,7,2],[-5,0,2]]),10)
path1=corner_radius(pts1([[-2,0],[2,0,2],[0,5,.3],[-.5,0]]),10)
sol2=translate([6,0,12],q_rot(['x90','z90'],prism(sec1,path1)))
a=array([[1-cos(d2r(i)),1-sin(d2r(i))]*.75 for i in linspace(0,90,20)])
b=[rsz3dc(sol1,array(bb(sol1))+i*2) for i in a[:,0]]
c=[rsz3dc(sol2,array(bb(sol2))+i*2) for i in a[:,1]]

with open('trial.scad','w+') as f:
    for i in range(len(a)-1):
        f.write(f'''  
include<dependencies2.scad>  
hull(){{  
intersection(){{  
{swp(b[i])}  
{swp(c[i])}  
}}  
  
intersection(){{  
{swp(b[i+1])}  
{swp(c[i+1])}  
}}  
}}  
''' )
    f.write(f'''  
    {swp(sol1)}  
    {swp(sol2)}  
''' )
t1=time.time()
t1-t0
```

Out[152...]: 1.2047343254089355

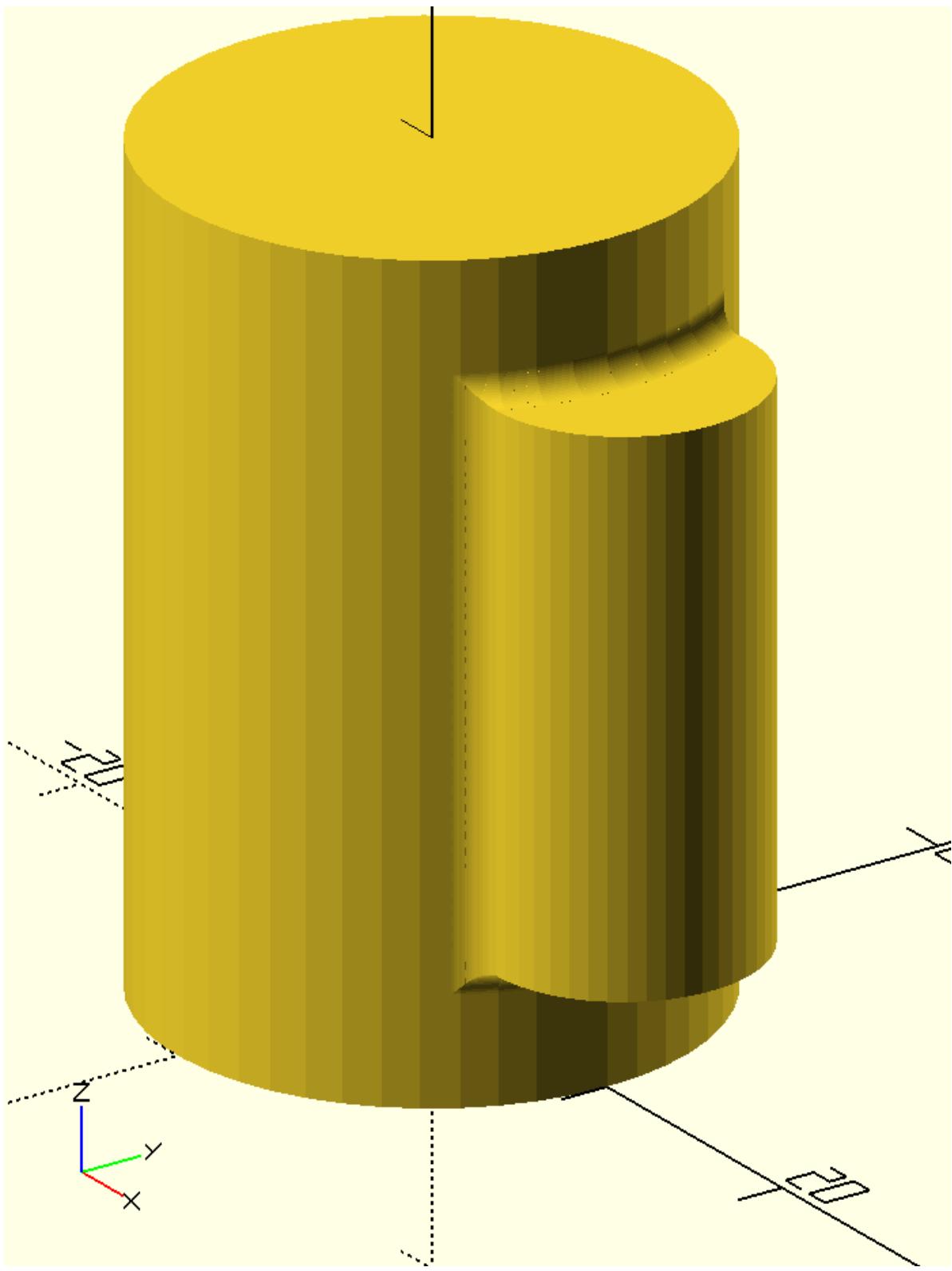


```
In [5]: cyl1=linear_extrude(circle(10,s=200),30)
cyl2=translate([10,0,5],linear_extrude(circle(5,s=100),20))
# cyl2=c2ro(cyl2,1)
a=array([[1-cos(d2r(i)),1-sin(d2r(i))] for i in linspace(0,90,20)])*1
b=[rsz3dc(cyl1,array(bb(cyl1)+i*2) for i in a[:,0]])
c=[rsz3dc(cyl2,array(bb(cyl2)+i*2) for i in a[:,1]])

with open('trial.scad','w+') as f:
    for i in range(len(a)-1):
        f.write(f'''')
include<dependencies2.scad>
hull(){{}
intersection(){{
{swp(b[i])}
{swp(c[i])}
}}}

intersection(){{
{swp(b[i+1])}
{swp(c[i+1])}
}}}
}'''

    f.write(f'''
{swp(cyl1)}
{swp(cyl2)}
'''')
```



```
In [5]: # iterative approach to filleting example

t0=time.time()

sec=pts([[0,0],[5,0],[0,5],[-5,0]])
path=corner_radius(pts1([[-.4,0],[.4,0,0.2],[0,5,0.2],[-.4,0]]),10)
sol=prism(sec,path)

sec1=circle(1.5,s=150)
# sol1=o_solid([0,0,1],sec1,10,-1.5,0,-2.5)
sol1=o_solid([1,0,10],sec1,10,-1.5,-2.5,0)

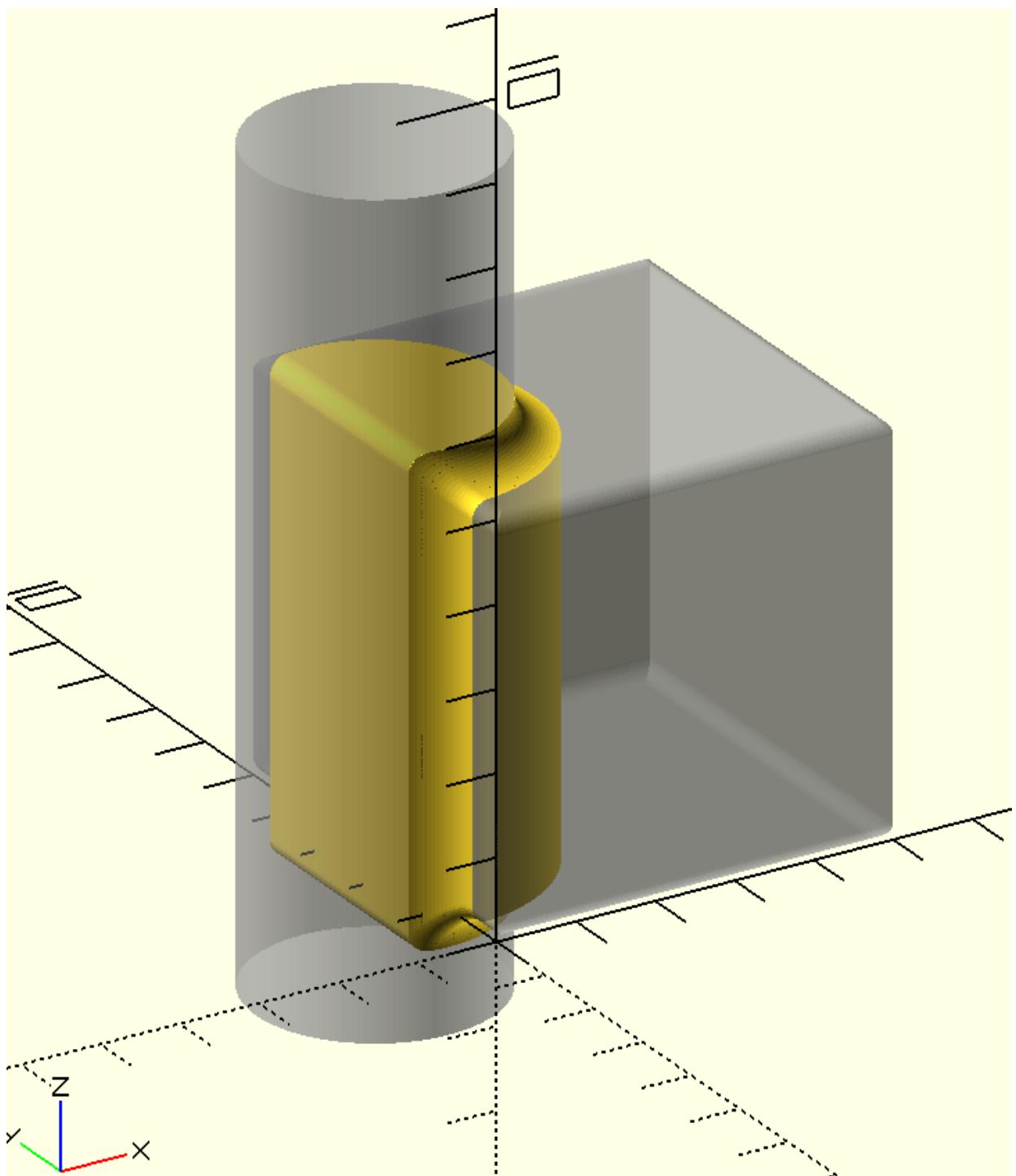
# sol1=slice_sol(sol1,20)

a=array([[1-cos(d2r(i)),1-sin(d2r(i))]] for i in linspace(0,90,20)])*.5
b=[rsz3dc(sol,array(bb(sol))+i*2) for i in a[:,0]]
c=[offset_sol(sol1,i) for i in a[:,1]]

with open('trial.scad','w+') as f:
    for i in range(len(a)-1):
        f.write(f'''
```

```
hull(){{  
intersection(){{  
{swp(b[i])}  
{swp(c[i])}  
}}  
  
intersection(){{  
{swp(b[i+1])}  
{swp(c[i+1])}  
}}  
}}  
  
...)  
  
f.write(f'''  
include<dependencies2.scad>  
  
{swp(sol)}  
{swp(sol1)}  
  
...)  
  
t1=time.time()  
t1-t0
```

Out[5]: 0.46150994300842285



In [155...]

```
t0=time.time()

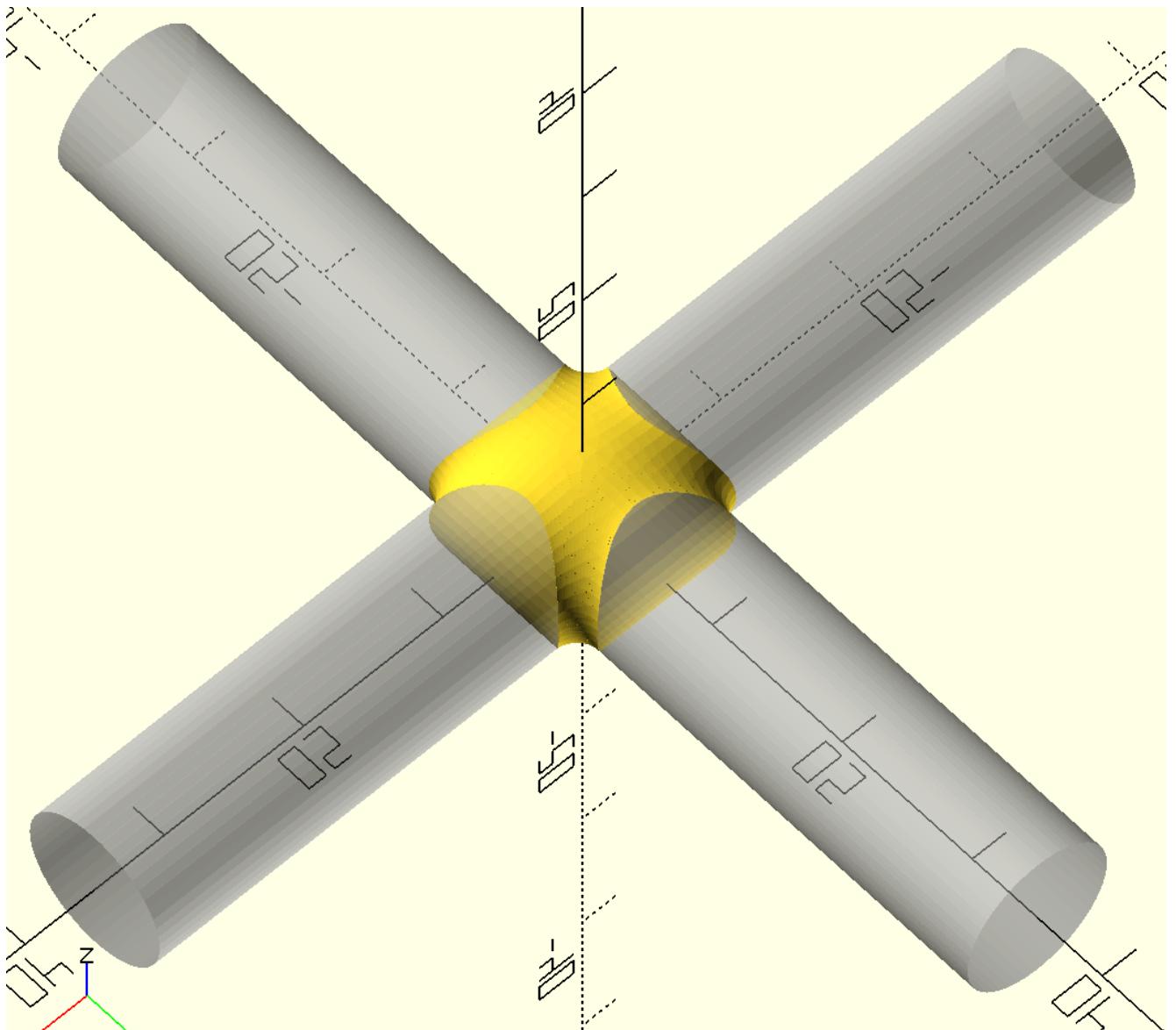
cyl1=o_solid([0,1,0],circle(5),70,-35)
cyl2=o_solid([1,0,0],circle(5),70,-35)

a=array([[1-cos(d2r(i)),1-sin(d2r(i))] for i in linspace(0,90,20)])*1.5
b=[rsz3dc(cyl1,array(bb(cyl1)+i*2) for i in a[:,0]]
c=[rsz3dc(cyl2,array(bb(cyl2)+i*2) for i in a[:,1]]

with open('trial.scad','w+') as f:
    for i in range(len(a)-1):
        f.write(f'''  
        hull(){{  
intersection(){{  
{swp(b[i])}  
{swp(c[i])}  
}}  
  
intersection(){{  
{swp(b[i+1])}  
{swp(c[i+1])}  
}}  
}}  
        ''')
    f.write(f'''  
include<dependencies2.scad>  
  
{swp(cyl1)}  
{swp(cyl2)}  
'''  
)
t1=time.time()
t1-t0
```

Out [155...]

0.054795026779174805



```
In [156... cyl1=linear_extrude(circle(5),20)
with open('trial.scad','w+') as f:
    f.write(f'''  

{swp1(cyl1)}  

''')
```

```
In [157... triangle=[[0,0],[5,0],[0,5]]
p=[0,3]
tx=triangulate_4p(triangle,p)

p0,p1,p2,p3=triangle+[p]

with open('trial.scad','w+') as f:
    f.write(f'''  

include<dependencies2.scad>  

color("blue")points({[p0,p1,p2,p3]},.2);
//p_line({tx},.05);
for(p={tx})p_line(p,.05);
color("magenta")p_line({cir_3p(tx[0][0],tx[0][1],tx[0][2],s=72)},.05);
color("cyan")p_line({cir_3p(tx[1][0],tx[1][1],tx[1][2],s=72)},.05);

''')
tx
```

```
Out[157... [[[0, 5], [0, 3], [5, 0]], [[5, 0], [0, 3], [0, 0]]]
```

lexicographic_sort_xy

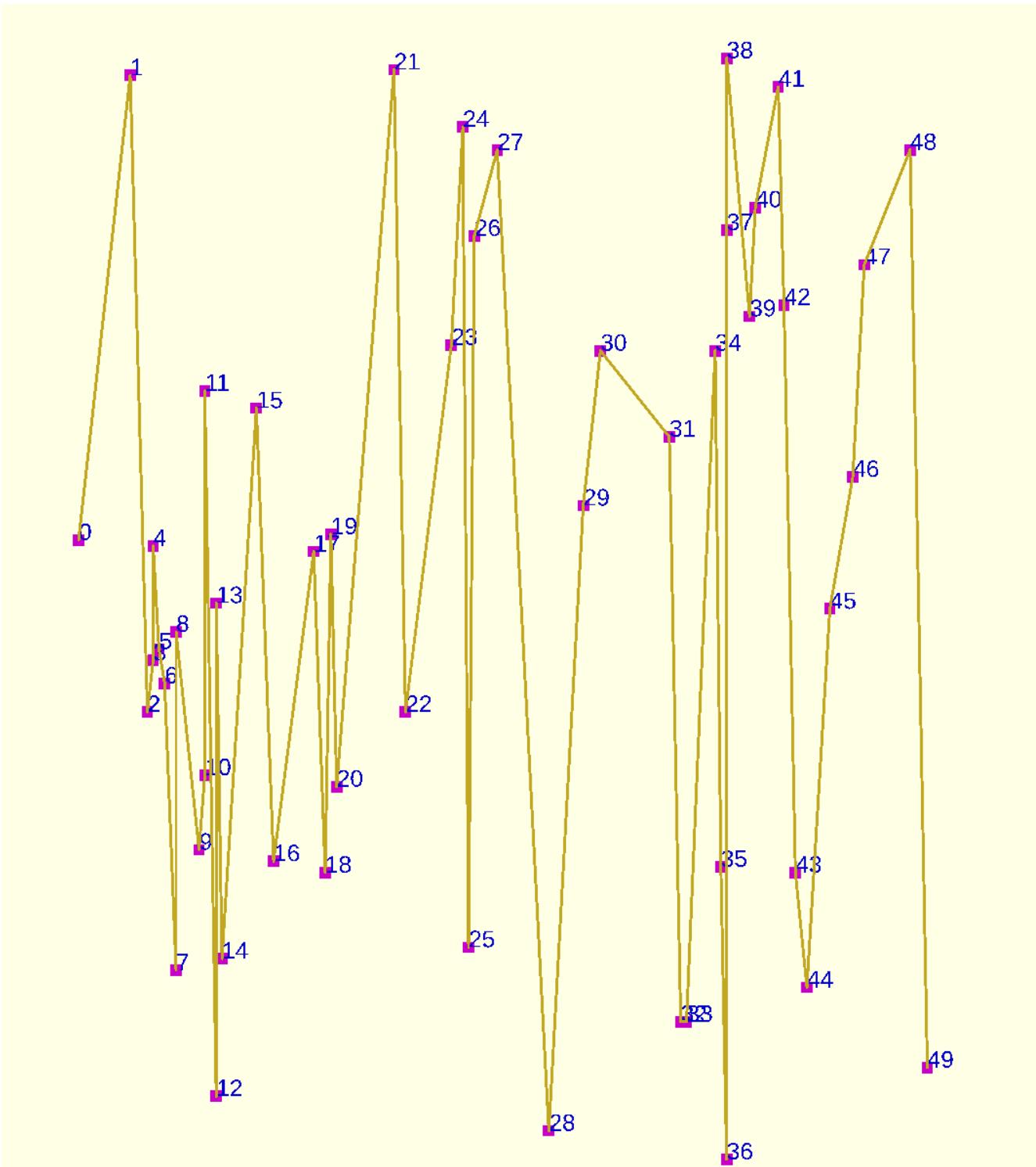
In [158...]

```
# example of function lexicographic_sort_xy(p)
t0=time.time()
a=random.random(50)*(20-5)+5
b=random.random(50)*(30-10)+10
p=array([a.round(1),b.round(1)]).transpose(1,0)

with open('trial.scad','w+')as f:
    f.write(f'''  
include<dependencies2.scad>
p2={lexicographic_sort_xy(p)};
color("blue")for(i=[0:len(p2)-1])translate(p2[i])text(str(i),.3);
p_lineo(p2,.05);
color("magenta")points(p2,.2);

''')
t1=time.time()
t1-t0
```

Out [158...]



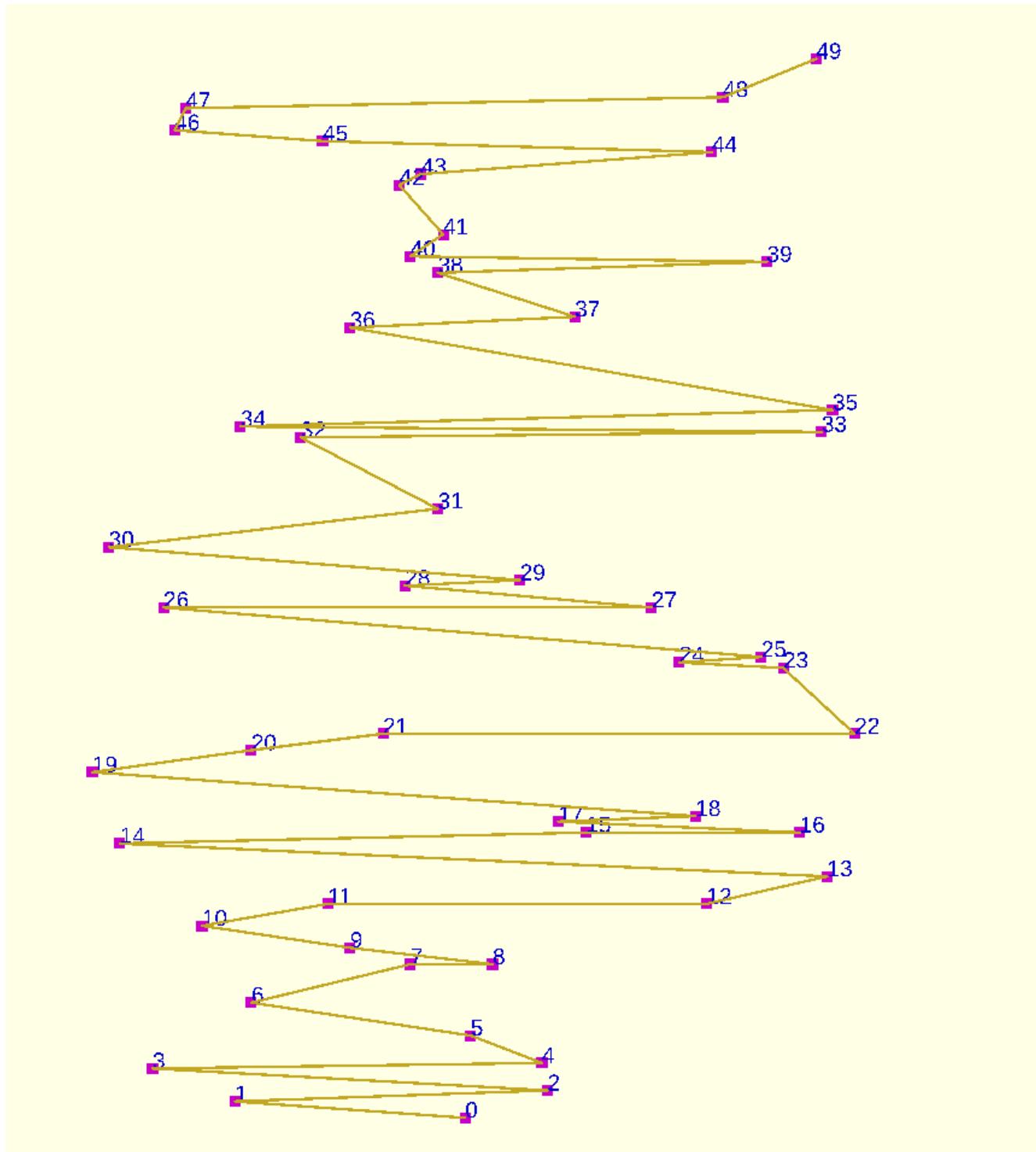
lexicographic_sort_yx

```
In [159]: # example of function lexicographic_sort_yx(p)
t0=time.time()
a=random.random(50)*(20-5)+5
b=random.random(50)*(30-10)+10
p=array([a.round(1),b.round(1)]).transpose(1,0)

with open('trial.scad','w+')as f:
    f.write(f'''
include<dependencies2.scad>
p2={lexicographic_sort_yx(p)};
color("blue")for(i=[0:len(p2)-1])translate(p2[i])text(str(i),.3);
p_lineo(p2,.05);
color("magenta")points(p2,.2);

    ''')
t1=time.time()
t1-t0
```

Out [159]: 0.007342815399169922



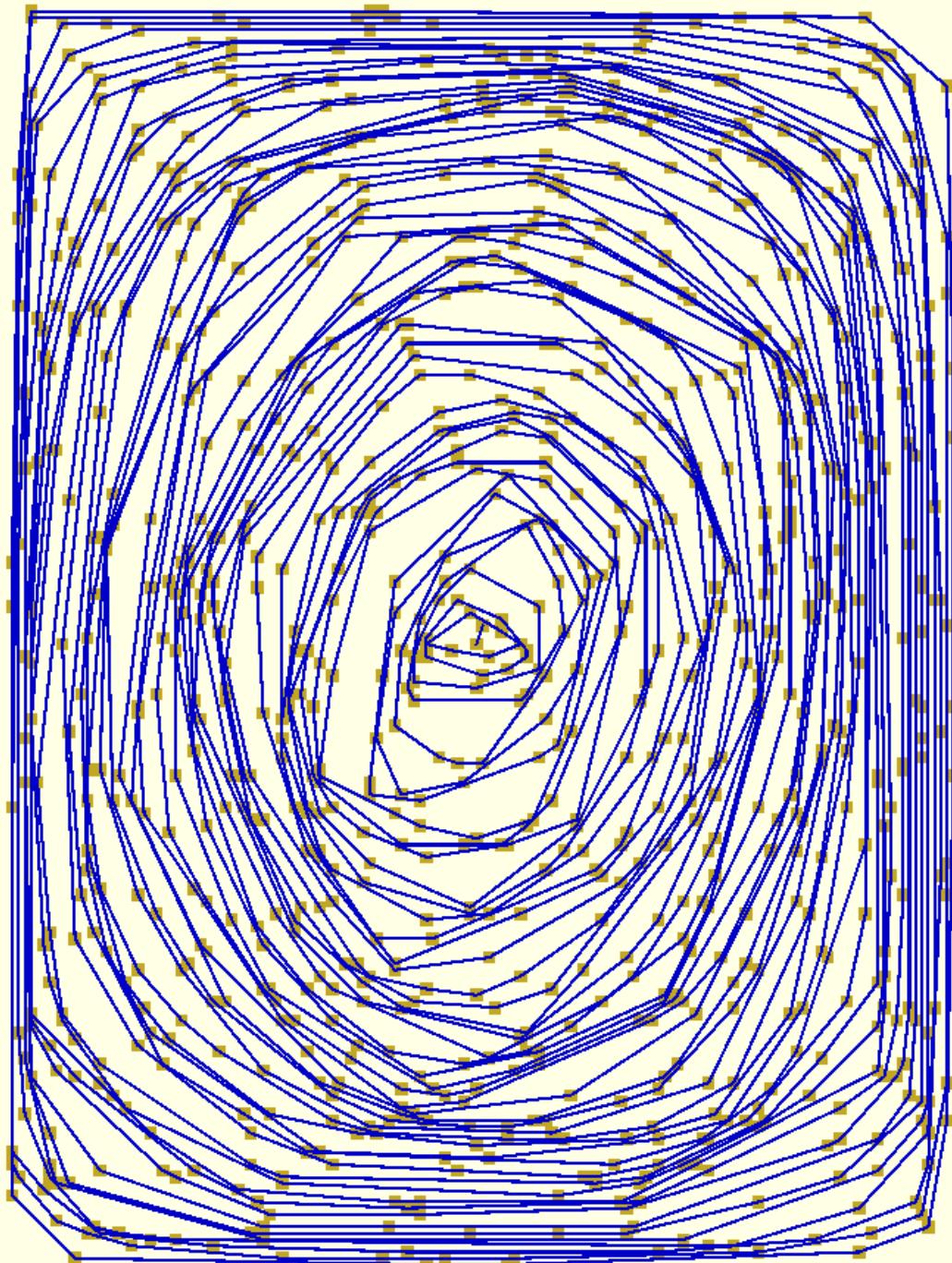
convex_hull

In [15]:

```
t0=time.time()
a=random.random(1000)*(20-5)+5
b=random.random(1000)*(30-10)+10
px=array([a.round(1),b.round(1)]).transpose(1,0).tolist()
p=px
sec=[]
while(p!=[]):
    p1=convex_hull(p)
    sec.append(p1)
    p=exclude_points(p,p1)
    if p==[]:
        break

with open('trial.scad','w+')as f:
    f.write(f'''  
include<dependencies2.scad>  
points({{px}},.2);  
  
color("blue")for (p={{sec}})p_line(p,.05);  
  
''')
t1=time.time()
t1-t0
```

Out[15]: 5.7603440284729



equivalent_rot_axis

```
In [161]: # example of function equivalent_rot_axis(rl[])
a,b,c,d,e,f1,g,h=20,-20,70,40,50,-50,70,10
r1=[ f"x{a}",f"y{b}",f"z{c}",f"y{d}",f"x{e}",f"y{f1}",f"z{g}",f"y{h}"]
sol=cylinder(h=50)
v2,theta=equivalent_rot_axis(r1)
sol1=axis_rot(v2,sol,theta)
sol2=axis_rot(v2,sol1,-theta)
```

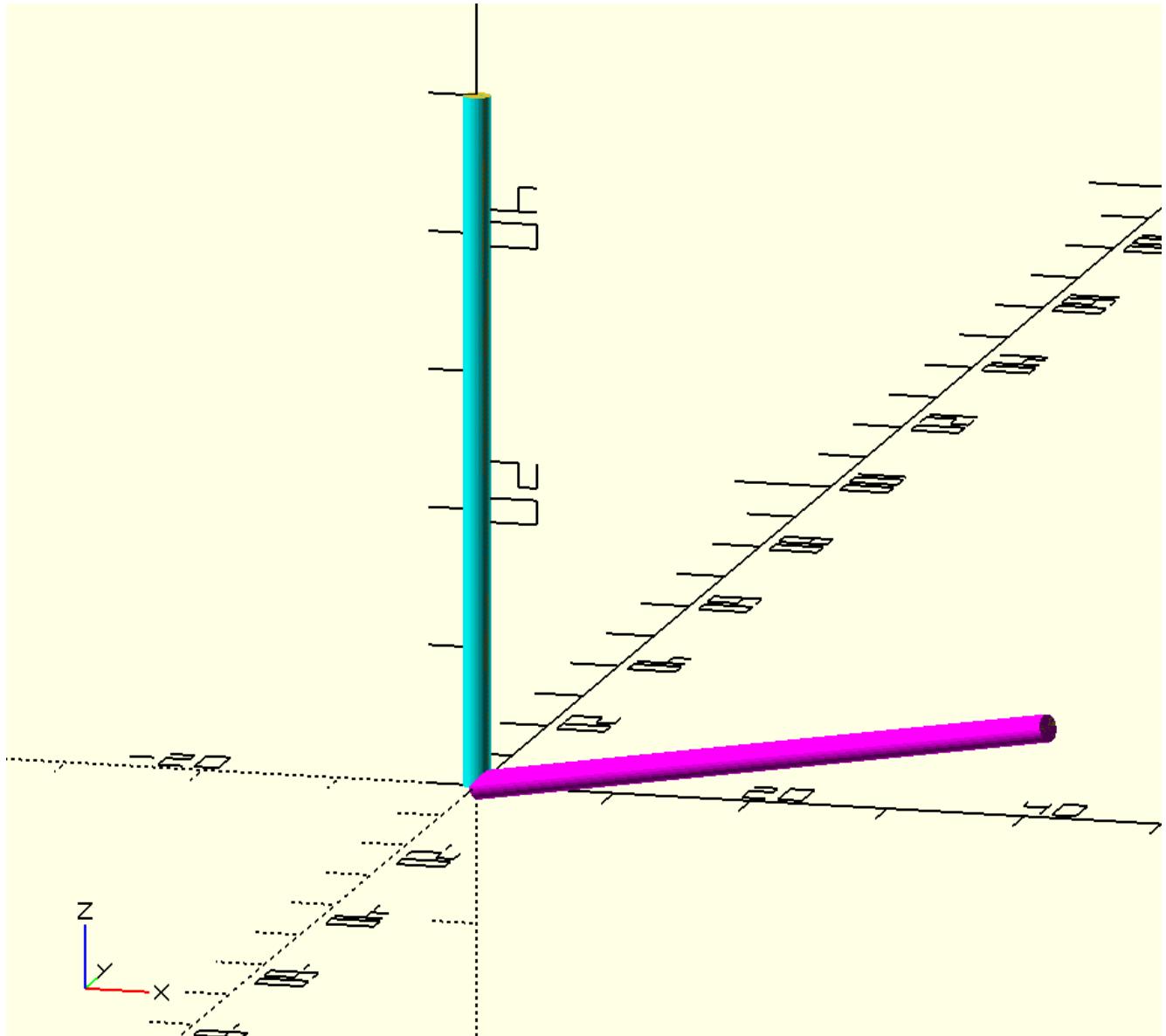
```
with open('trial.scad','w+')as f:
    f.write(f'''
include<dependencies.scad>

// pure openscad way of rotating the 3d object
color("magenta")
rotate({[0,h,0]})
```

```

// rotation by function equivalent_rot_axis
r1=[ "x{a}","y{b}","z{c}","y{d}","x{e}","y{f1}","z{g}","y{h}"];
theta=equivalent_rot_axis(r1)[0];
v2=equivalent_rot_axis(r1)[1];
rotate(-theta,v2)
rotate(theta,v2)
cylinder(r=1.01,h=50,$fn=30);
'''')

```



```

In [162...]
s_y=20+5/sin(d2r(45))
h_z=10/20*s_y
s_y1=(s_y**2-h_z**2)**0.5
sec=corner_radius(pts1([[20,s_y,5],[-20,20,5],[-20,-20,5],[20,-20,5]]),20)
sec1=offset(sec,5)

path=cr_3d([[0,0,h_z,0],[0,s_y1,-h_z,5],[0,s_y1,h_z,0]],20)
sec2=wrap_around(sec,path)
sec3=wrap_around(sec1,path)

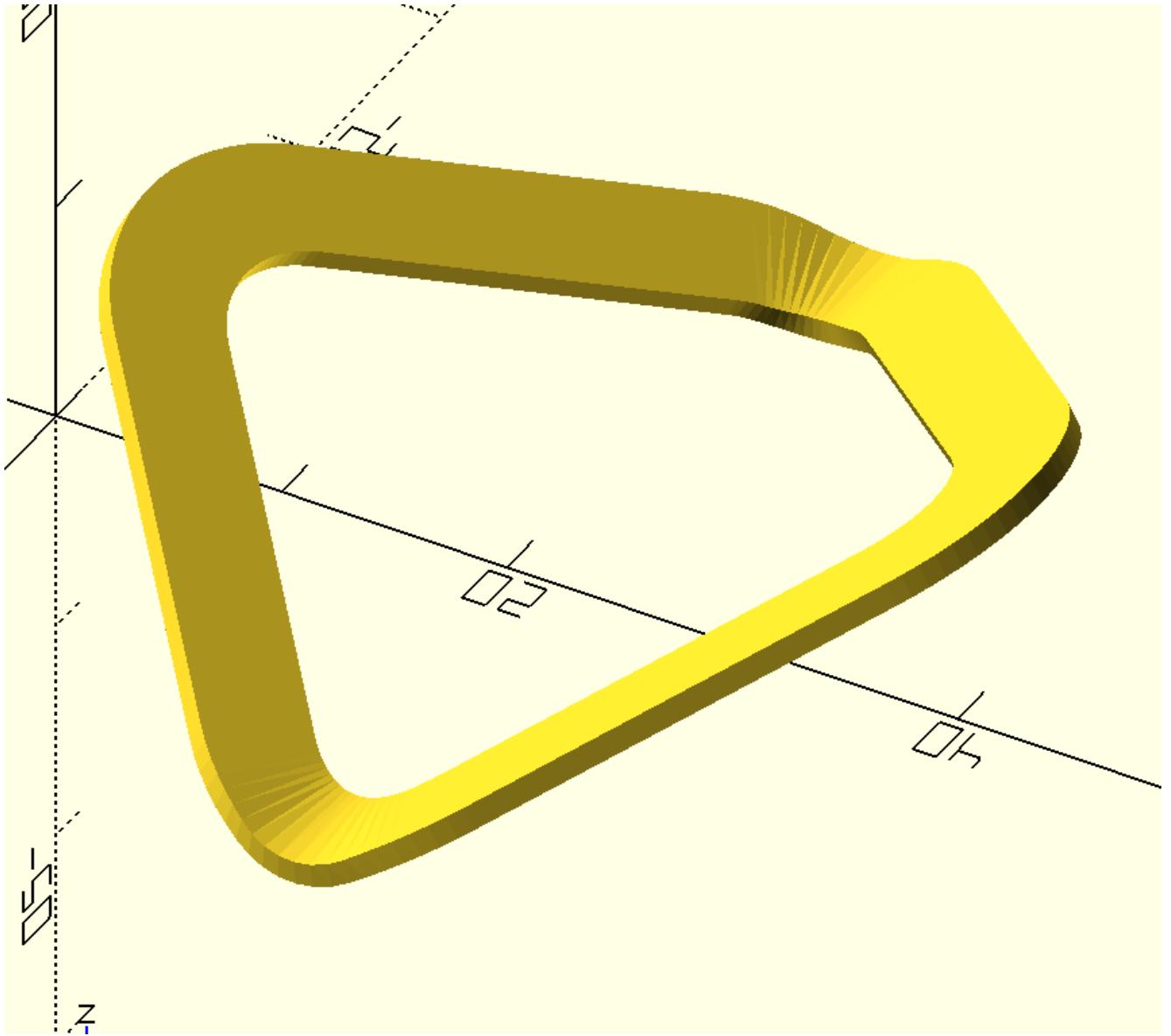
surf1=[sec2,sec3]
surf2=surf_offset(surf1,-1)
sol=surf1+flip(surf2)+[surf1[0]]
d=s_y1/s_y*20
path2=cr_3d([[20,s_y1+.25,0,3.5],[-20,d,10,5],[-20,-d,-10,3.5],[20,-d,10,5]],20)

with open('trial.scad','w+')as f:
    f.write(f'''
include<dependencies2.scad>

//color("blue")for(p={cpo(sol)})p_line3dc(p,.05);

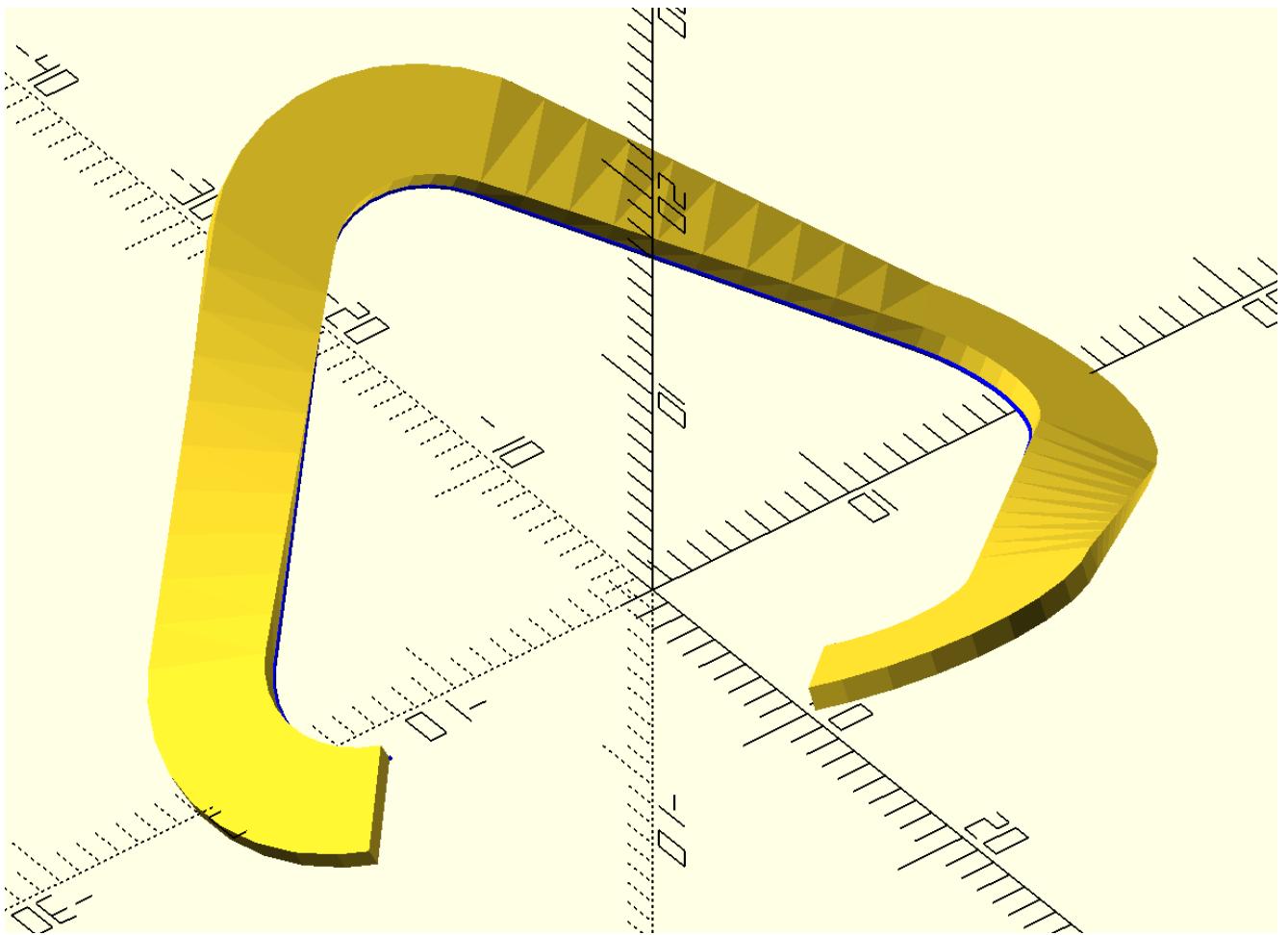
```

```
//color("magenta")p_line3dc({path2},.05);
{swp_c(sol)}
    ''')
```



path_extrude_open

```
In [163]: # example of path_extrude_open(sec,path,twist=0) which adds a twist to the section
sec=pts([[0,0],[5,0],[0,1],[-5,0]])
path=cr_3d([[20,0,10,5],[-20,20,-10,5],[-20,-20,10,5],[20,-20,-10,5]],10)
# path=q_rot(['x30'],circle(20))
# path=helix(20,10,1,5.1)
sol=path_extrude_open(sec,path,1)
sol=slice_sol(sol,10)
with open('trial.scad','w+')as f:
    f.write(f'''
include<dependencies2.scad>
color("blue")p_line3d({path},.1);
{swp(sol)}
    ''')
```



`tangents_along_path`

`normals_along_path`

`orthos_along_path`

In [164]:

```
# path=q_rot(['x90.001'],circle(10))
# path=cr_3d([[0,0,0,3],[15,0,0,4],[5,3,15,3],[0,10,0,3],[-5,3,-15,4],[-15,0,0,3]],10)
path=cr_3d([[20,0,10,5],[-20,20,-10,5],[-20,-20,10,5],[20,-20,-10,5]],20)
# path=helix(10,5,3,5.001)

# sec=pts([[-1.5,-1.25],[3,0],[-1.5,2.5]])
# sec=pts([[0,0],[5,0],[0,1],[-5,0]])
sec=[[0,0],[1,0],[0,1]]
sol=path_extrude_open(sec,path)
# sol=slice_sol(sol,10)
# sol=align_sol(sol,1)

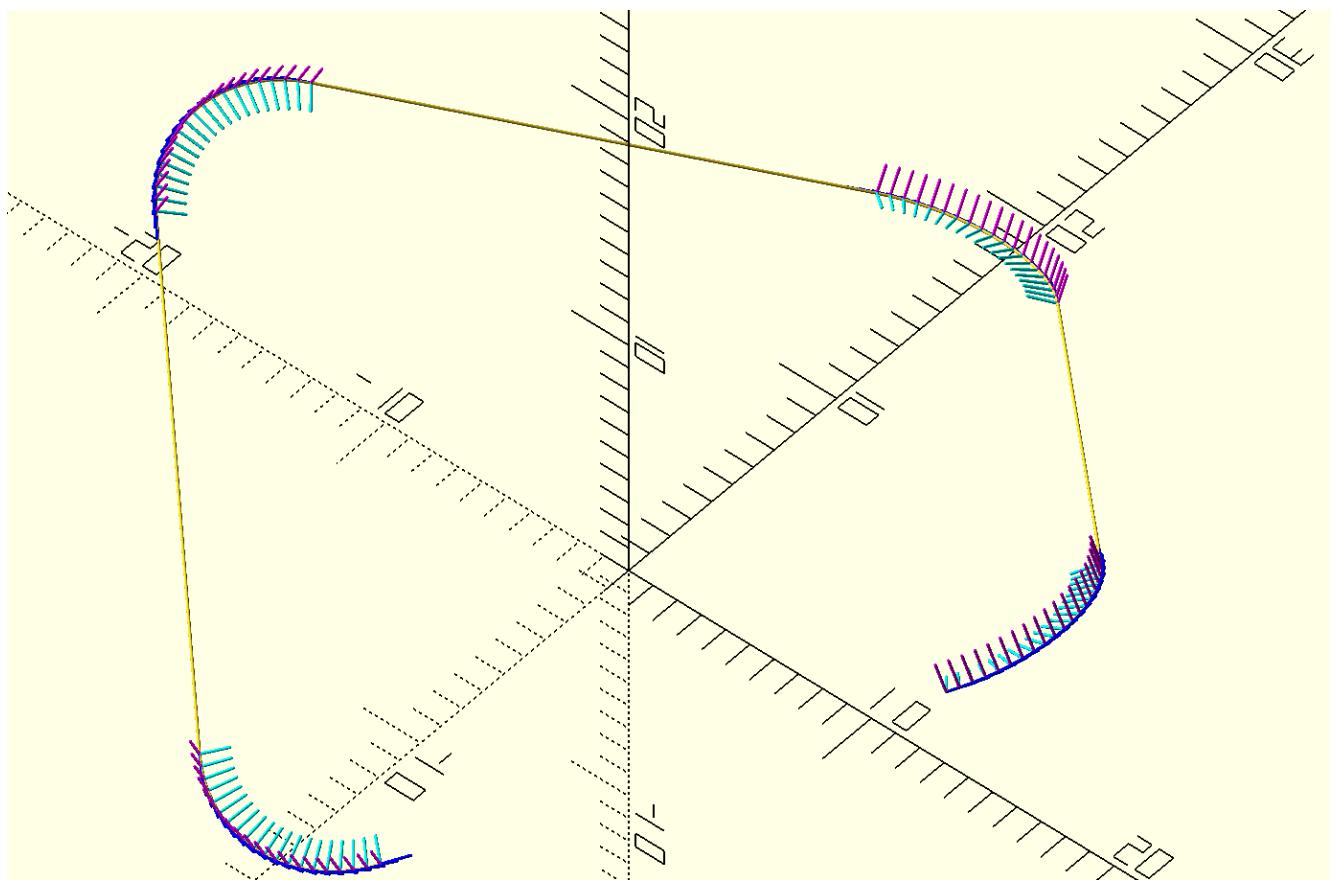
t_v1=tangents_along_path(path,1)
n_v1=normals_along_path(path,1)
o_v1=orthos_along_path(path,1)

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>

p_line3d({path},.05);
color("blue")for(p={t_v1})p_line3d(p,.05);
color("magenta")for(p={n_v1})p_line3d(p,.05);
color("cyan")for(p={o_v1})p_line3d(p,.05);

//{swp(sol)}
```

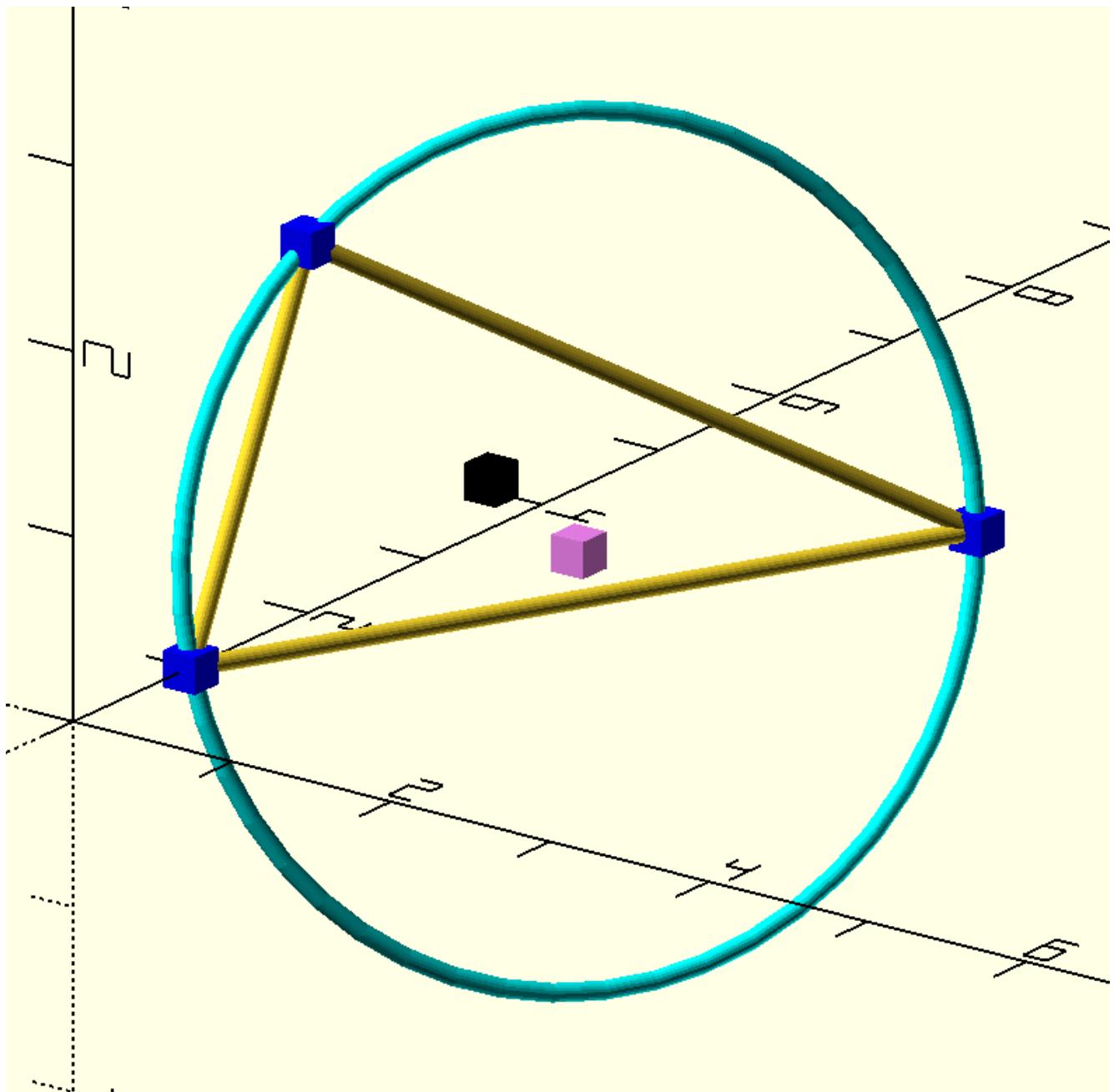
...)



```
In [165]: # sec2=[[0,0,0],[2,0,0],[0,0,2]]  
sec2=[[0,1,0],[2,3,0],[0,3,2]]  
  
cir1=cir_3p_3d(sec2,100)  
cp=cp_cir_3d(sec2)  
with open('trial.scad','w+') as f:  
    f.write(f'''  
include<dependencies2.scad>  
color("blue")points({sec2},.2);  
p_line3dc({cir1},.05);  
  
color("magenta")points({[cp]},.2);  
...)
```

cp_cir_3d

```
In [166]: # explanation of functions to draw circle from 3 points in 3d space  
# function to get center of the circle drawn with 3 points. Note that the center of circle is different  
# Also note that the centroid of the triangle is same as mean of the 3 points  
  
sec=[[0,1,0],[2,5,0],[0,2,2]] # 3 points in 3d space  
  
cir1=cir_3p_3d(sec,50) #function to draw circle from 3 points in 3d space  
cp=cp_cir_3d(sec) # function calculates the center of the circle  
cp1=centroid_3p_3d(sec)  
cp2=array(sec).mean(0).tolist()  
with open('trial.scad','w+') as f:  
    f.write(f'''  
include<dependencies2.scad>  
color("blue")points({sec},.2);  
p_line3dc({sec},.05);  
  
color("cyan")p_line3dc({cir1},.05);// circle drawn by 3 points input  
color("violet")points({[cp]},{.2});// center point of the circle containing the 3 points as input  
color("magenta")points({[cp1]},{.2});// centroid of a triangle  
color("black")points({[cp2]},{.2});// mean is the centroid  
...)
```



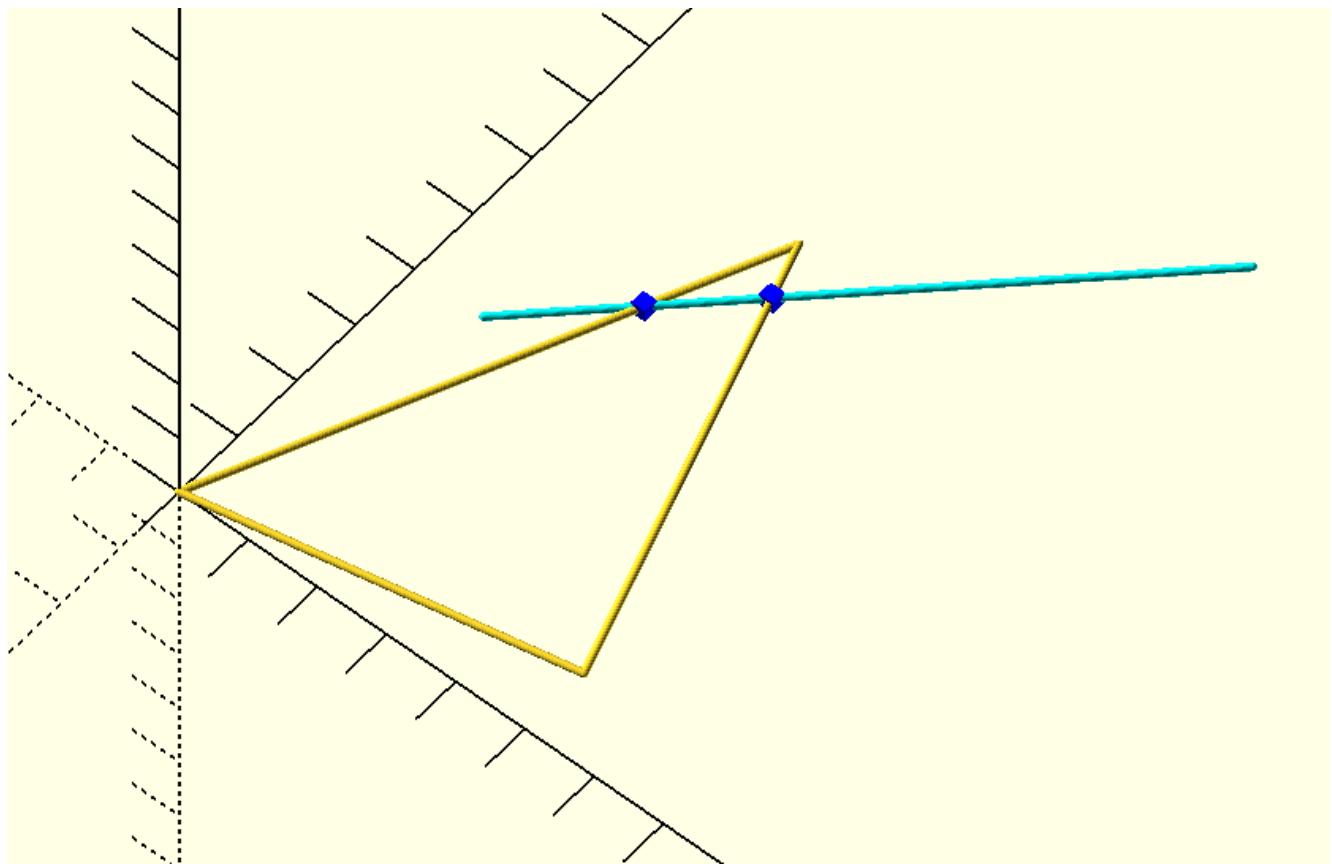
l_sec_ip

```
In [167]: # example of line to section intersection point function
line=[[1,4],[7,10]]
sec=[[0,0],[5,1],[3,7]]

i_p1=l_sec_ip(line,sec)
with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>

p_line3dc({sec},.05);
color("cyan")p_line3dc({line},.05);
color("blue")points({i_p1},.2);

...)
```



i_line_planes

```
In [168...]: # example of intersection line between 2 planes
t0=time.time()

p1=translate([-2,0,-8.22],[[2,4,5],[7,9,15],[1,10,5]])

p2=translate([-3,0,-5],[[5,10,3],[5,-7,5],[-10,5,10]])

i_p1=i_line_planes(p2,p1)

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>

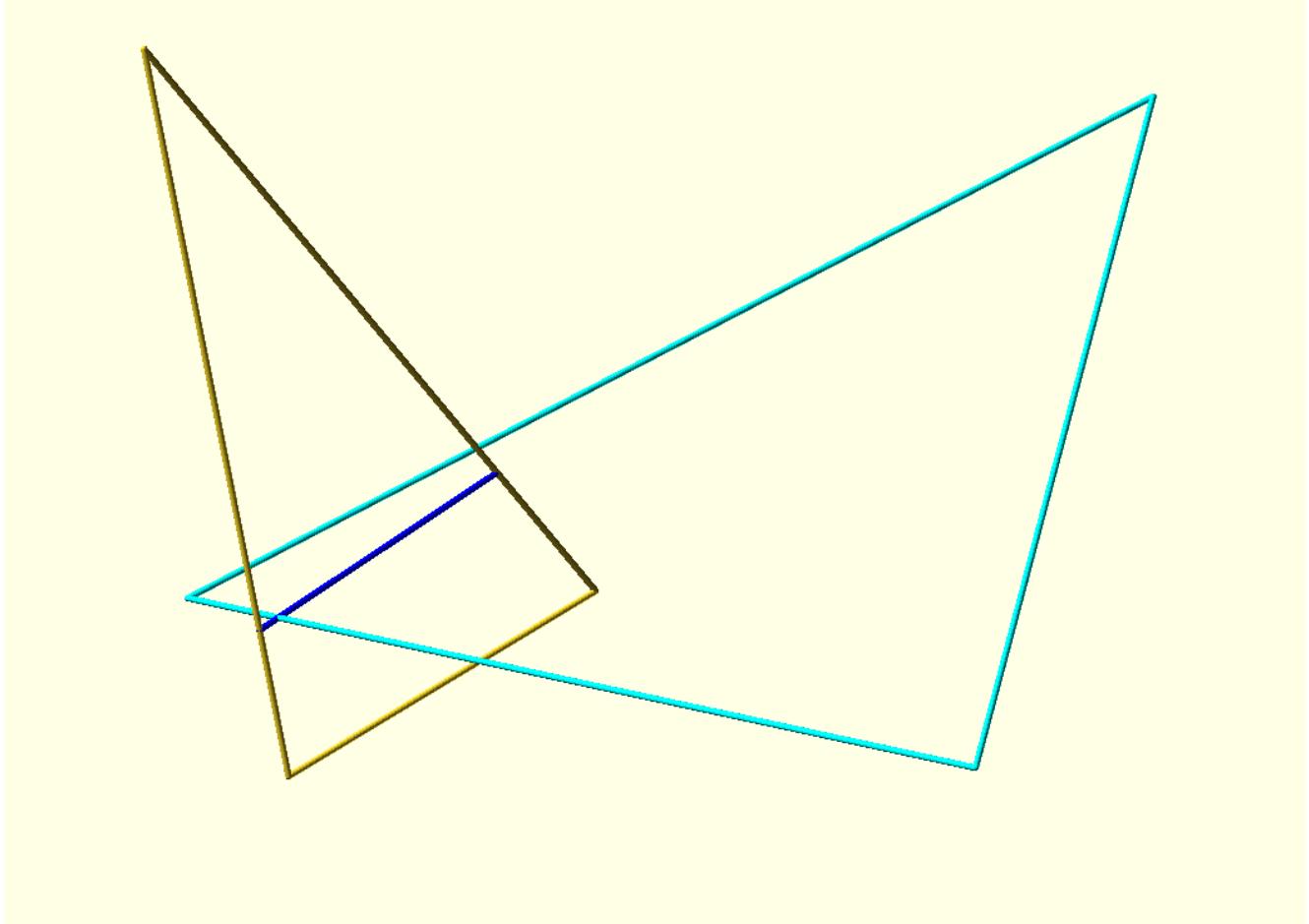
p_line3dc({p1},.05);
color("cyan")p_line3dc({p2},.05);

color("blue")p_line3d({i_p1},.05);

''')

t1=time.time()
t1-t0
```

Out [168...]: 0.016090869903564453



surface_for_fillet

```
In [16]: # example of function surface_for_fillet(sol1=[],sol2=[],factor1=50,factor2=10,factor3=1,factor4=100,a
t0=time.time()

sec=circle(10)
path=corner_radius(pts1([[-8,0],[10,0],[-2,0,2],[-1,15,3],[-8.9,0]]),10)
path=equidistant_path(path,100)
sol1=q_rot(['z90'],prism(sec,path))

sec1=corner_radius(pts1([[0,0,1],[5,0,1],[0,7,2.3],[-5,0,2.3]]),10)
path1=corner_radius(pts1([-2.4,0],[2.4,0,2],[0,5,.3],[-.5,0])),10)
path1=equidistant_path(path1,30)
sol2=translate([6,0,12],q_rot(['x90','z90'],prism(sec1,path1)))

sol3=surface_for_fillet(sol1,sol2,100,20,4,23,8)
ip2=ip_sol2sol(sol2,sol3)
ip2=[p[-1] for p in ip2]
# ip3=o_3d(ip2,sol2,1)
# ip4=o_3d(ip2,i_p1,1,1)
# fillet1=convert_3lines2fillet_closed(ip2,ip3,ip4)

fillet1=i_line_fillet(sol2,sol3,ip2,1,-1)
with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>
%{swp(sol1)}
%{swp(sol2)}
color("cyan")for(p={sol2})p_line3dc(p,.01,rec=1);
color("cyan")for(p={cpo(sol2)})p_line3d(p,.01,rec=1);

color("blue")for(p={sol3})p_line3dc(p,.01,rec=1);
color("blue")for(p={cpo(sol3)})p_line3d(p,.01,rec=1);
color("magenta")p_line3dc({ip2},.02,rec=1);

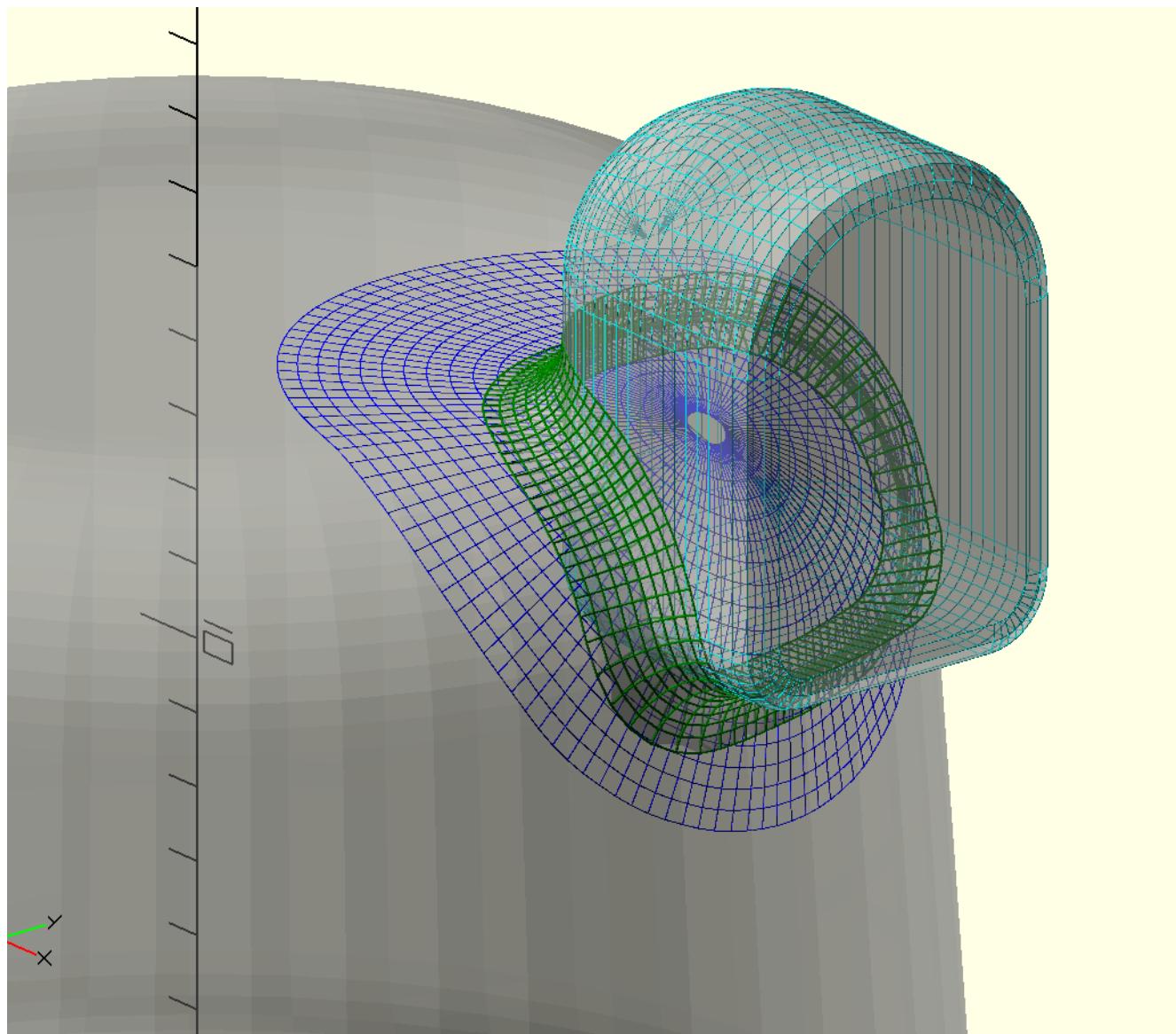
{swp_c(fillet1)}
color("green")for(p={fillet1})p_line3dc(p,.02,rec=1);
color("green")for(p={cpo(fillet1)})p_line3dc(p,.02,rec=1);
```

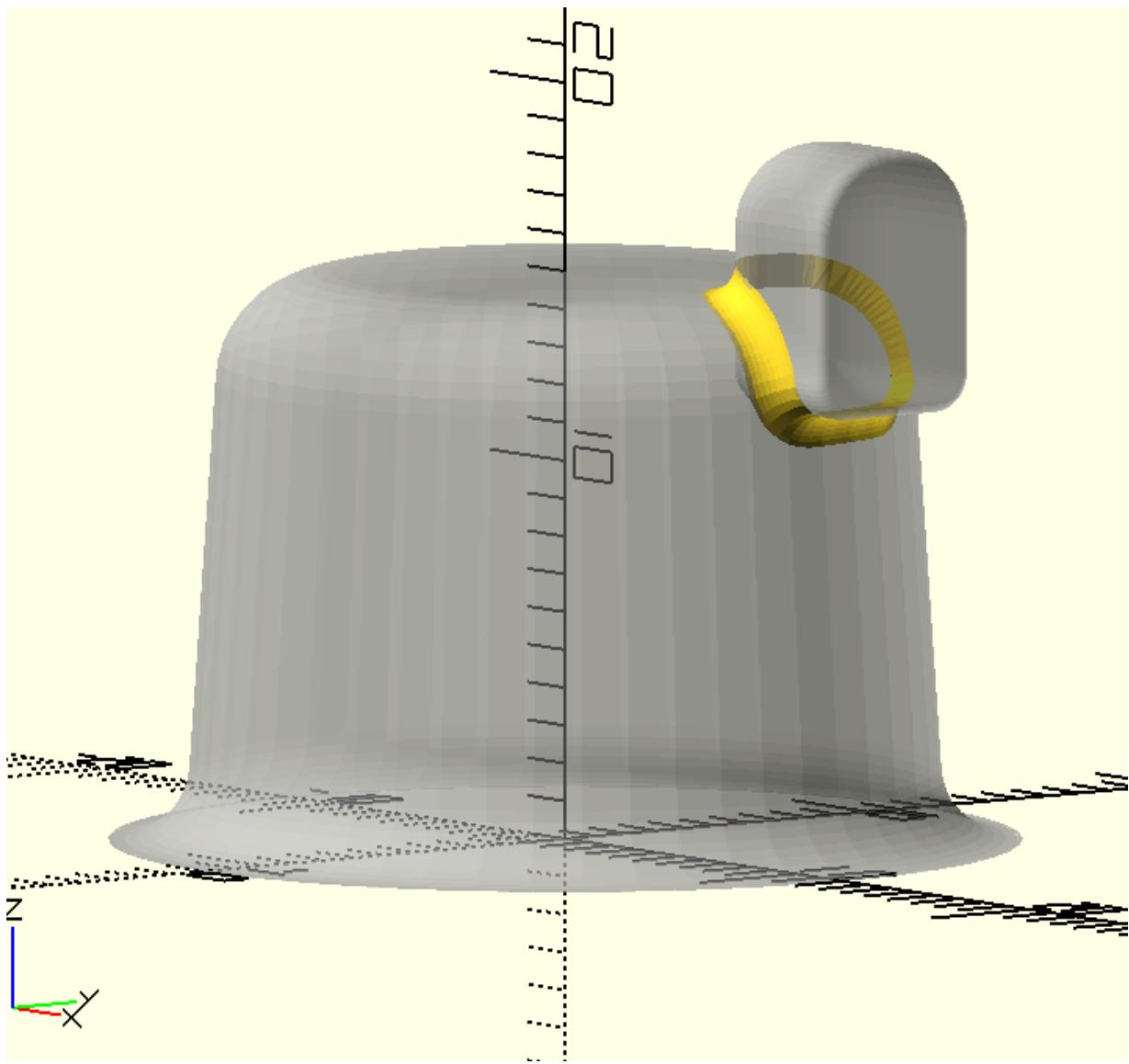
```
...)
```

```
t1=time.time()  
t1-t0
```

```
/Users/sanjeevprabhakar/openscad/openscad1.py:4629: RuntimeWarning: invalid value encountered in divide  
    t=einsum('kl,ijkl->ijk',cross(p01,p02),la[:, :,None]-p0)/(einsum('ijl,kl->ijk',(-lab),cross(p01,p02))  
+.00000)  
/Users/sanjeevprabhakar/openscad/openscad1.py:4630: RuntimeWarning: divide by zero encountered in divid  
e  
    u=einsum('ijkl,ijkl->ijk',cross(p02[None,None,:,:],(-lab)[:, :,None,:]),(la[:, :,None,:]-p0[None,Non  
e,:,:]))/(einsum('ijl,kl->ijk',(-lab),cross(p01,p02))+.00000)  
/Users/sanjeevprabhakar/openscad/openscad1.py:4631: RuntimeWarning: divide by zero encountered in divid  
e  
    v=einsum('ijkl,ijkl->ijk',cross((-lab)[:, :,None,:],p01[None,None,:,:]),(la[:, :,None,:]-p0[None,Non  
e,:,:]))/(einsum('ijl,kl->ijk',(-lab),cross(p01,p02))+.00000)  
/Users/sanjeevprabhakar/openscad/openscad1.py:4632: RuntimeWarning: invalid value encountered in add  
    condition=(t>=0)&(t<=1)&(u>=0)&(u<=1)&(v>=0)&(v<=1)&(u+v<1)  
/Users/sanjeevprabhakar/openscad/openscad1.py:4629: RuntimeWarning: divide by zero encountered in divid  
e  
    t=einsum('kl,ijkl->ijk',cross(p01,p02),la[:, :,None]-p0)/(einsum('ijl,kl->ijk',(-lab),cross(p01,p02))  
+.00000)  
/Users/sanjeevprabhakar/openscad/openscad1.py:4630: RuntimeWarning: invalid value encountered in divide  
    u=einsum('ijkl,ijkl->ijk',cross(p02[None,None,:,:],(-lab)[:, :,None,:]),(la[:, :,None,:]-p0[None,Non  
e,:,:]))/(einsum('ijl,kl->ijk',(-lab),cross(p01,p02))+.00000)  
/Users/sanjeevprabhakar/openscad/openscad1.py:4634: RuntimeWarning: invalid value encountered in multip  
ly  
    a=(la[:,None,:,None,:]+lab[:,None,:,None,:]*t[:,None,:,:None])
```

```
Out[16]: 2.3262360095977783
```





```
In [170]: t0=time.time()
sec1=circle(55,s=70)
path1=corner_radius(pts1([[-50,30],[56,0],[0,8,5],[-4,3,5],[0,18,10],[8,5,10],[0,12],[-50,0]]),10)
sol1=q_rot(['z0'],prism(sec1,path1))

sec2=circle(7.5)
path2=corner_radius(pts1([[-5,0],[5,0,5],[0,35,3],[-4,0]]),10)
sol2=translate([58,0,35],prism(sec2,path2))

surf1=surface_for_fillet(sol1,sol2,70,20,5,200,50)

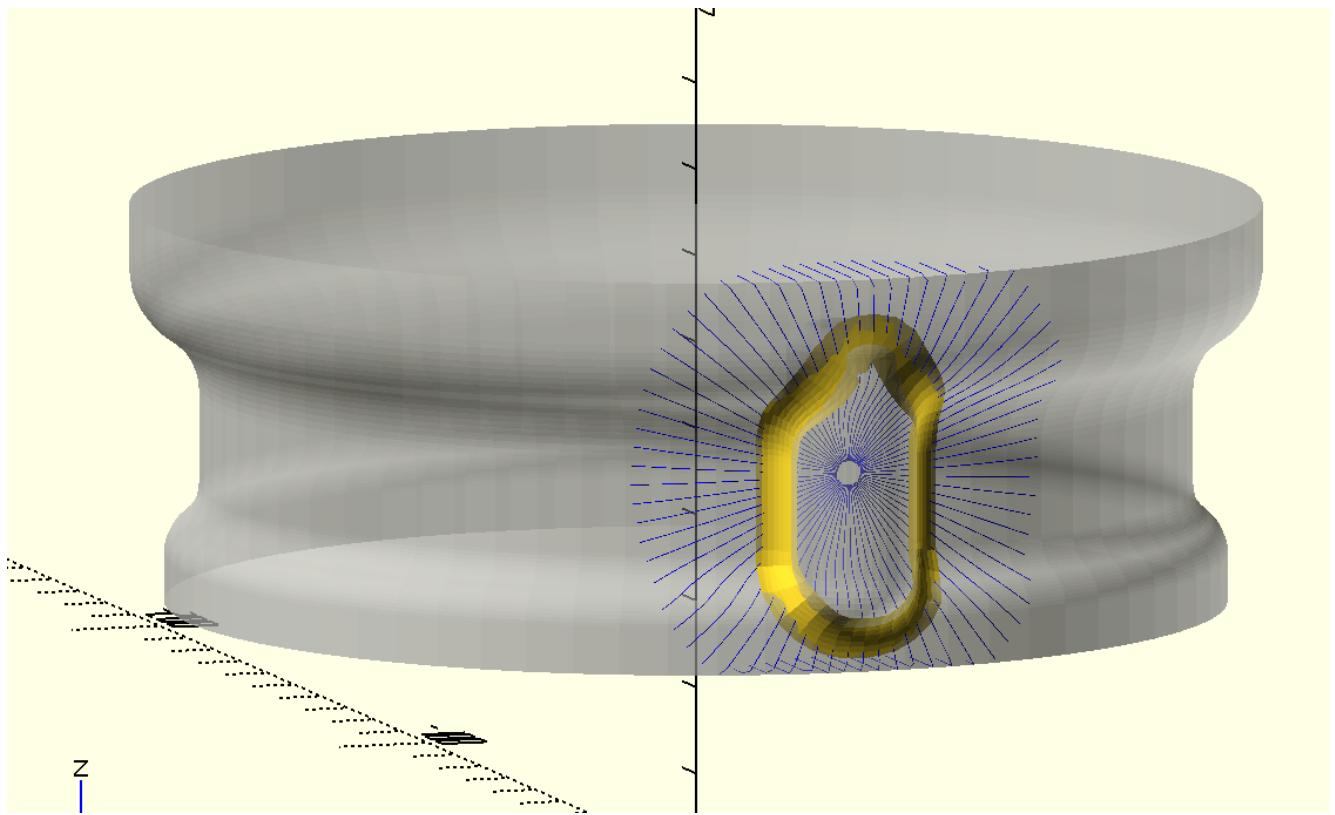
fillet1=fillet_sol2sol(sol2,surf1,4)
with open('trial.scad','w+') as f:
    f.write(f'''  
include<dependencies2.scad>
%{swp(sol1)}
%{swp(sol2)}

color("blue")for(p={cpo(surf1)})p_line3d(p,.05);
{swp_c(fillet1)}
```

```
'''')
t1=time.time()
total=t1-t0
total
```

```
/Users/sanjeevprabhakar/iCloud Drive (Archive)/Documents/Download/python/openscad1.py:2403: RuntimeWarning: invalid value encountered in arcsin
a=arcsin(r1/d1)*180/pi
```

```
Out[170]: 1.9271330833435059
```



i_line_fillet

```
In [17]: i_t=time.time()
sec1=circle(55)
path1=corner_radius(pts1([[-50,0],[50,0,.2],[0,30,3],[6,1,3],[0,6,3],[-4,2,3],[0,22,6],[8,2,6],[0,10,0]]),10)
# path1=equidistant_path(path1,200)
sec2=circle(7.5)
path2=corner_radius(pts1([[-5,0],[5,0,5],[0,35,2],[-5,0]]),10)
path2=equidistant_path(path2,50)
sol1=prism(sec1,path1)
sol2=translate([57.5,0,37],prism(sec2,path2))
sol2=axis_rot_o([0,0,1],sol2,180)
i_p1=ip_sol2sol(sol1,cpo(sol2))
p1=[p[0] for p in i_p1]
p2=[p[-1] for p in i_p1]
p3=flip(p1)+p2

fillet1=i_line_fillet(sol1,sol2,p3,3,-3)

with open('trial.scad','w+') as f:
    f.write(f'''
        include<dependencies2.scad>

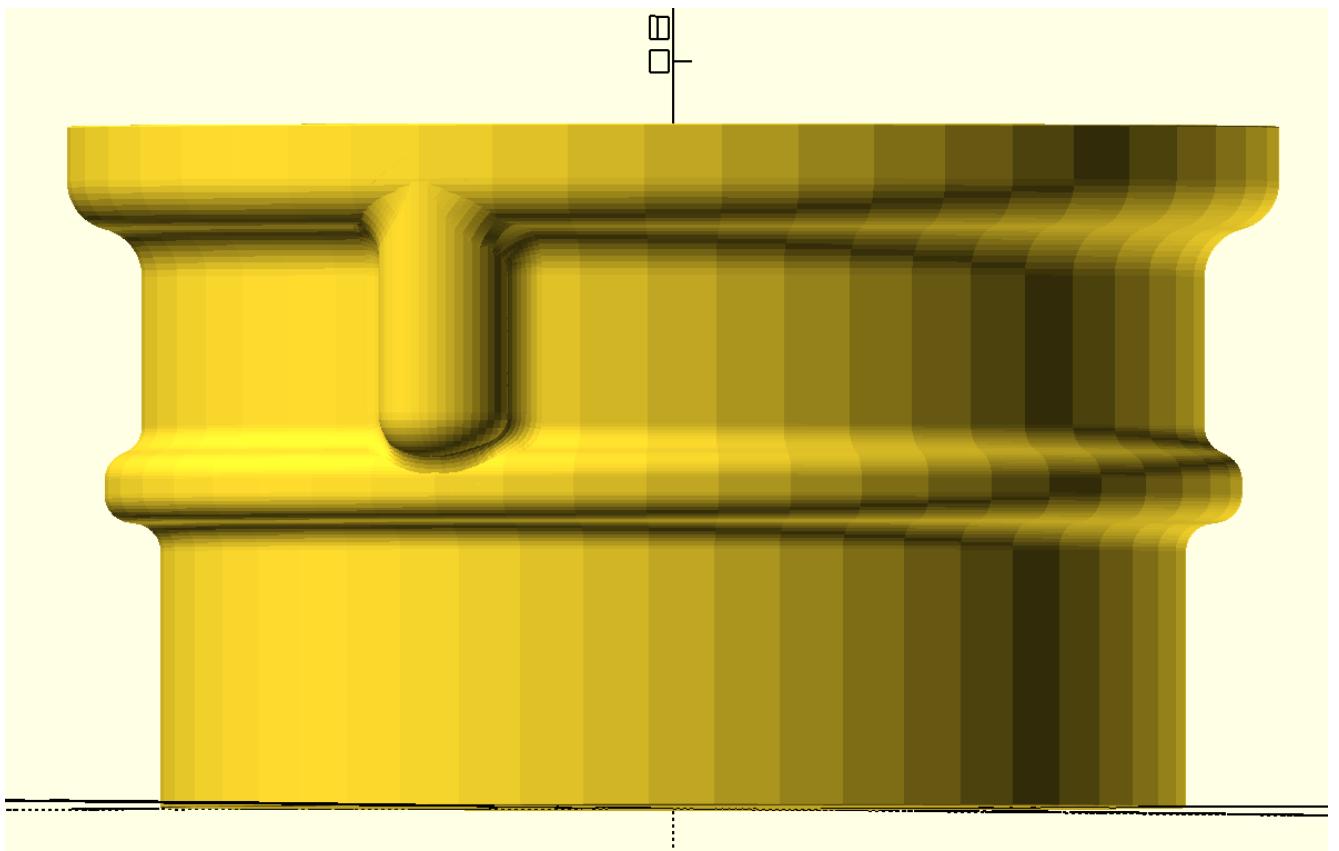
        %swp(sol1)
        %swp(sol2)

        //color("blue")for(p={cpo(sol2)})p_line3d(p,.1,rec=1);
        color("magenta")points({p1},.3);
        color("magenta")points({p2},.3);
        color("cyan")p_line3dc({p3},.1,rec=1);
        {swp_c(fillet1)}

    ''')

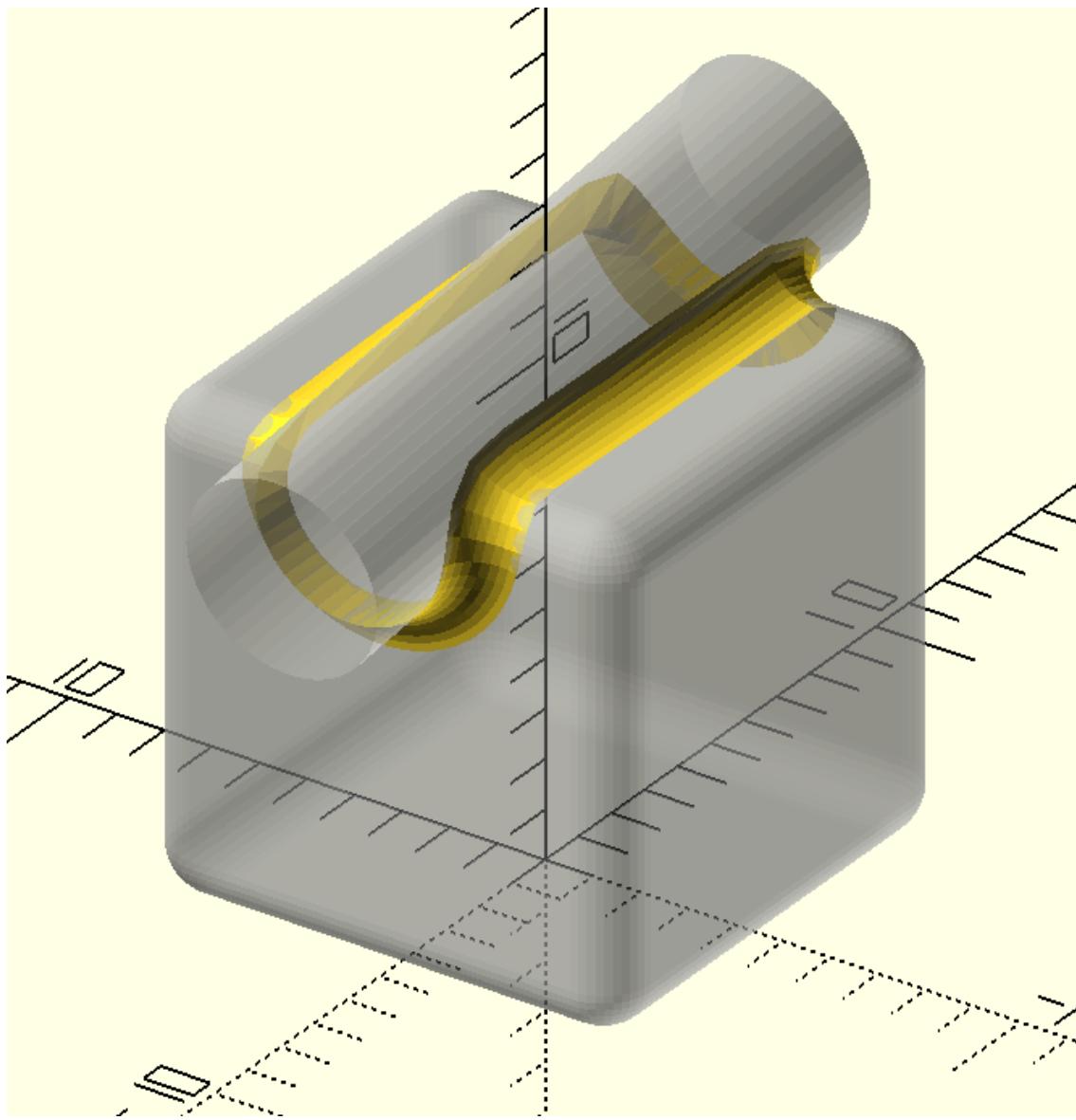
f_t=time.time()
f_t-i_t
# len(p1),len(p2),len(p3)
```

Out [17]: 4.994019031524658



```
In [172...]  
i_t=time.time()  
sec=corner_radius(pts1([[-5,-5,1],[10,0,1],[0,10,1],[-10,0,1]]),10)  
path=corner_radius(pts1([[-4,0],[4,0,1],[0,10,1],[-4,0]]),10)  
sol1=prism(sec,path)  
sol2=o_solid([1,0,.1],circle(2,s=100),15,-7,0,10,[-90,0,0])  
sol2=slice_sol(sol2,2)  
pa=ip_sol2sol(sol1,sol2)  
p1=[p[0] for p in pa]  
p2=[p[-1] for p in pa]  
  
p3=flip(p1)+p2  
# p4=o_3d(p3,sol2,-1)  
# p5=o_3d(p3,sol1,1)  
# fillet1=convert_3lines2fillet_closed(p3,p5,p4)  
fillet1=i_line_fillet(sol1,sol2,p3,1,-1)  
  
with open('trial.scad','w+')as f:  
    f.write(f'''  
        include<dependencies2.scad>  
  
        %swp(sol1)  
        %swp(sol2)  
        color("blue")p_line3d({p1},.04,rec=1);  
        color("magenta")p_line3d({p2},.04,rec=1);  
  
        {swp_c(fillet1)}  
        ''')  
f_t=time.time()  
f_t-i_t  
# len(p3),len(p4),len(p5)
```

Out [172... 0.7792840003967285



```
In [173]: i_t=time.time()

sec=pts([[0,0],[5,0],[0,5],[-5,0]])
path=corner_radius(pts1([[-2.49,0],[2.49,0,0.25],[0,5,0.25],[-2.49,0]]),10)
sol1=prism(sec,path)
v1=[2,0,8]
sec1=axis_rot([0,0,1],circle(1.5,s=100),90)
sol2=o_solid(v1,sec1,10,-1.5,-2.5,0)
pa=ip_sol2sol(sol1,sol2)
p1=[p[0] for p in pa]
p2=[p[-1] for p in pa]
p3=flip(p1)+p2
# p4=equidistant_pathc(o_3d(p3,sol1,.5),200)
# p4=sort_points(p3,p4)
# # p4=path2path1(p3,p4)
# p5=equidistant_pathc(o_3d(p3,sol2,-.5),200)
# p5=sort_points(p3,p5)
# # p5=path2path1(p3,p5)
# p3,p4,p5=align_sol_1([p3,p4,p5])
# fillet1=convert_3lines2fillet_closed(p3,p5,p4)

fillet1=i_line_fillet(sol1,sol2,p3,.5,-.5)

with open('trial.scad','w+') as f:
    f.write(f'''  

include<dependencies2.scad>  

%{swp(sol1)}  

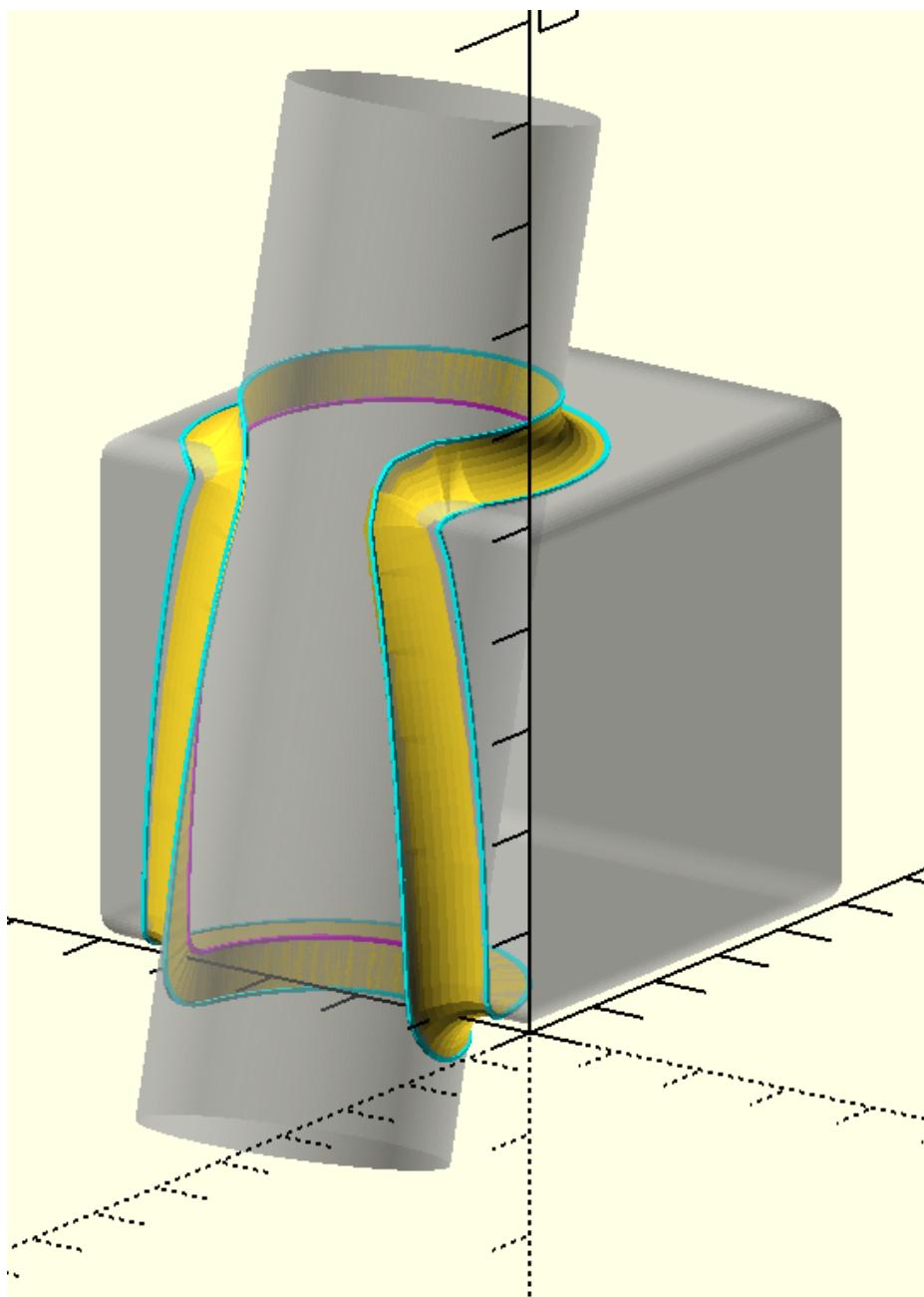
%{swp(sol2)}  

{swp_c(fillet1)}  

...)
```

```
f_t=time.time()  
f_t-i_t  
  
# len(p3),len(p4),len(p5)
```

Out[173... 0.9269061088562012



```
In [214... import sys  
set_printoptions(threshold=sys.maxsize)
```

align_sol_1

```
# merging 2 different shapes  
t0=time.time()  
  
sec=corner_radius(pts1([[1.5,8,.9],[2,0,.9],[0,-3,1.4],[3,0,1.4],[0,4,1.9],[-5,0,2.5],[0,6,2.5],  
[7,0,2.5],[0,-4,.9],[-2,0,.9],[0,3,1.4],[-3,0,1.4],[0,-4,1.9],  
[5,0,2.5],[0,-6,2.5],[-7,0,2.5]]),20)  
  
cp1=array(c2t3(sec)).mean(0)+[0,0,10]  
sec=translate(cp1,sec)  
sec1=c2t3(pts([[-5,-10],[10,0],[0,20],[-10,0]]))  
  
sec=equidistant_pathc(sec,300)  
sec1=equidistant_pathc(sec1,300)  
  
sol=align_sol_1([sec1,sec])
```

```

sol=slice_sol(sol,30)

sol1=array(sol).reshape(-1,3)

with open('trial.scad','w+')as f:
    f.write(f'''  

        include<dependencies.scad>  

difference(){{  

    {swp(sol)}  

    translate([0,0,-.01])cube(5);  

}}  

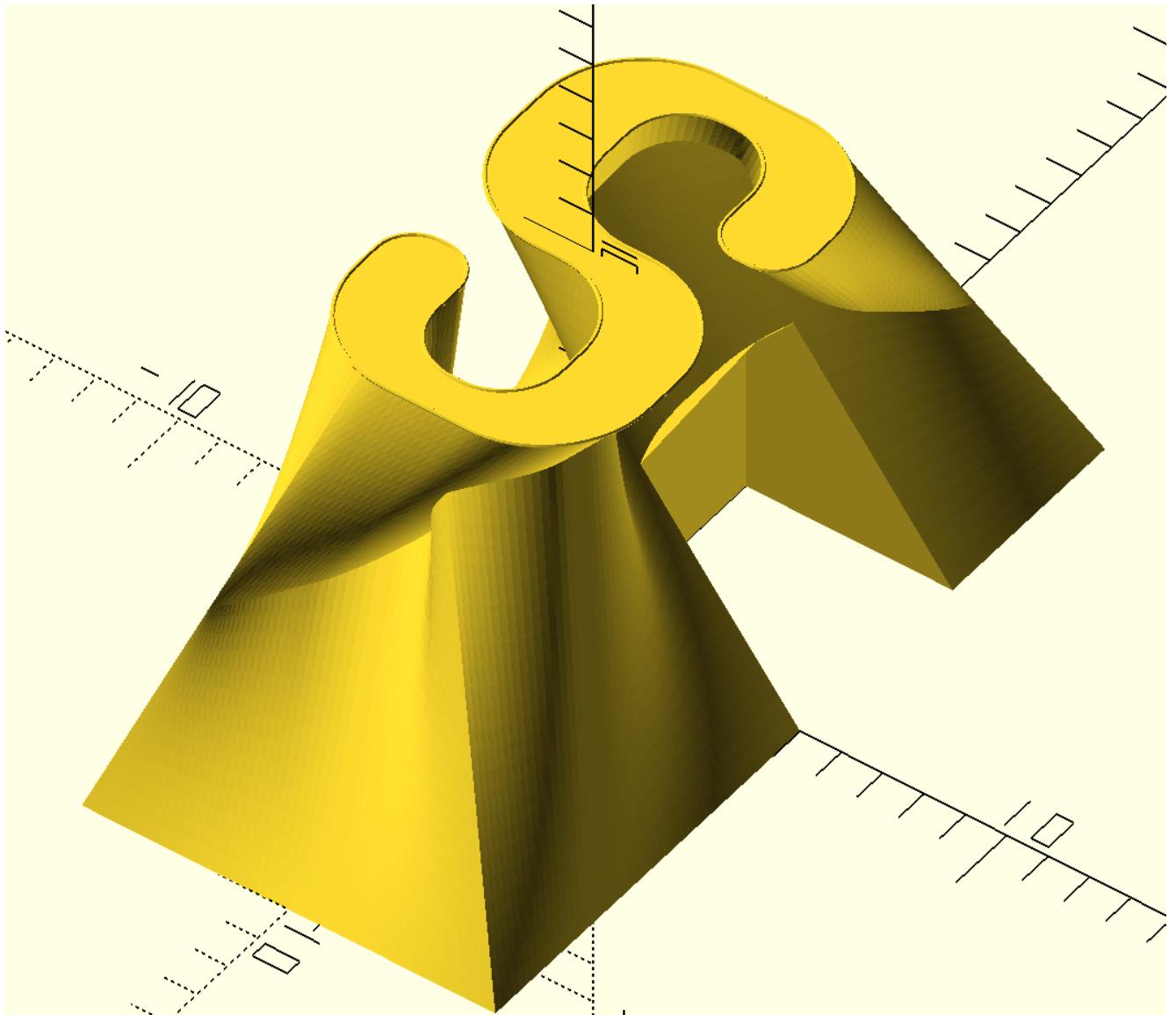
  

color("blue")p_line3dc({sol[-1]},.05);
'''')

t1=time.time()
t1-t0

```

Out[174... 0.2719290256500244



coil-example

```

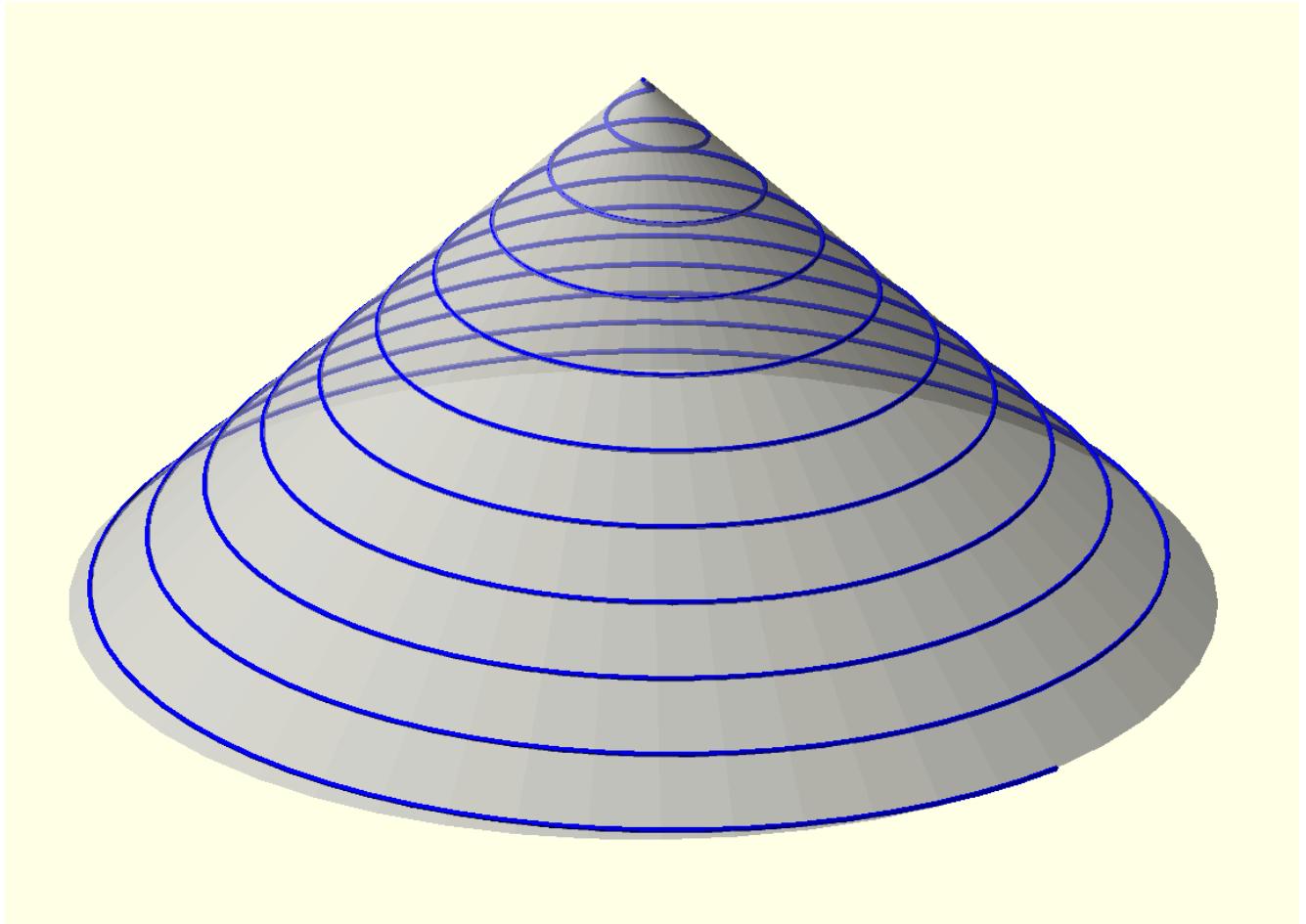
In [175... coil=array([i/360*array([cos(d2r(i)),sin(d2r(i)),-1]) for i in linspace(0,3600,720)]).tolist()
cyl1=translate([0,0,-10],cylinder(r1=10,r2=0.1,h=10))
with open('trial.scad','w+')as f:
    f.write(f'''  

        include<dependencies.scad>  

translate([0,0,10]){{  

    color("blue")p_line3d({coil},.05);
%{swp(cyl1)}
}}
```

```
}}  
...)
```



In [178]:

```
x,y=sym.symbols('x y')  
  
f1=sym.lambdify(x,10*sym.cos(x),'numpy')  
f2=sym.lambdify(x,10*sym.sin(x),'numpy')  
  
a=linspace(0,2*pi,100)  
cir1=array([f1(a),f2(a)]).transpose(1,0).tolist()  
  
with open('trial.scad','w+')as f:  
    f.write(f'''  
        include<dependencies.scad>  
        color("blue")p_line({cir1},.05);  
  
    ''')
```

In [44]:

```
v1x,v1y,v2x,v2y,p0x,p0y,p1x,p1y,t1,t2=sym.symbols('v1x,v1y,v2x,v2y,p0x,p0y,p1x,p1y,t1,t2')  
  
p0,v1=array([[2,3],[4,5]])  
p1,v2=array([[10,7],[-3,7]])  
  
# p0+v1*t1=p1+v2*t2  
# v1*t1-v2*t2=p1-p0  
# v1x*t1-v2x*t2=(p1-p0)x  
# v1.y*t1-v2.y*t2=(p1-p0).y  
  
sp.linsolve([v1x*t1-v2x*t2-(p1x-p0x),v1y*t1-v2y*t2-(p1y-p0y)],t1,t2)
```

Out [44]:

$$\frac{-p_0 v_2 y + p_0 y v_2 x + p_1 x v_2 y - p_1 y v_2 x}{v_1 x v_2 y - v_1 y v_2 x}$$

In [50]:

```
# p0+v1*t1=p2+v2*t2+v3*t3  
# v1*t1-v2*t2-v3*t3=p2-p0  
  
v1x,v1y,v1z,v2x,v2y,v2z,v3x,v3y,v3z,p0x,p0y,p0z,p2x,p2y,p2z,t1,t2,t3=\nsp.symbols('v1x,v1y,v1z,v2x,v2y,v2z,v3x,v3y,v3z,p0x,p0y,p0z,p2x,p2y,p2z,t1,t2,t3')
```

```
f=sp.linsolve([v1x*t1-v2x*t2-v3x*t3-(p2x-p0x),v1y*t1-v2y*t2-v3y*t3-(p2y-p0y),v1z*t1-v2z*t2-v3z*t3-(p2z
```

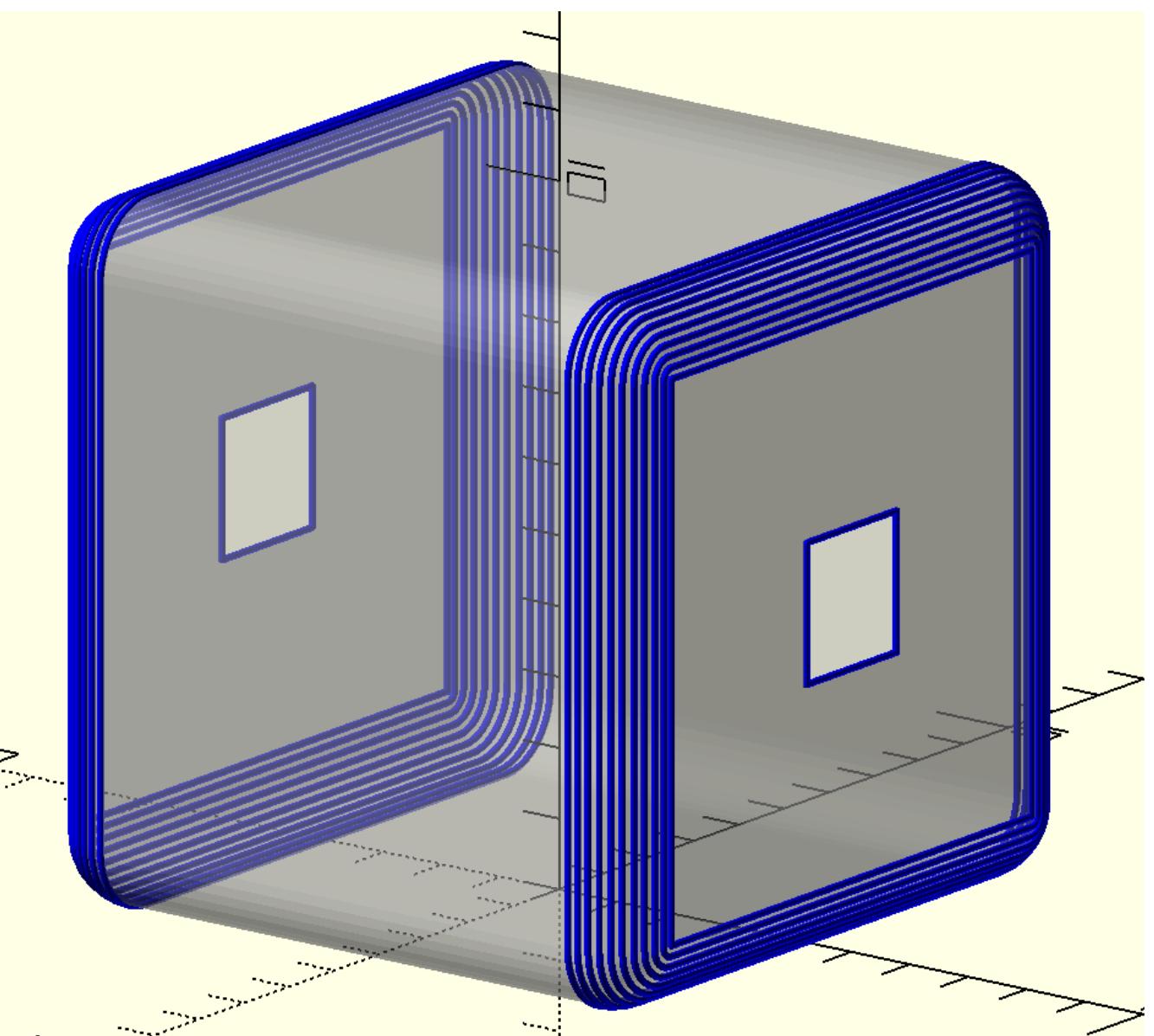
axis_rot_o

In [186...]

```
t0=time.time()
sec=corner_radius(pts1([[-5,-5,1],[10,0,1],[0,10,1],[-10,0,1]]),10)
path=corner_radius(pts1([[-4,0],[4,0,1],[0,10,1],[-4,0]]),10)
sol1=axis_rot_o([0,1,0],prism(sec,path),90)

with open('trial.scad','w+')as f:
    f.write(f'''  
    include<dependencies.scad>  
  
    %{swp_c(sol1)}  
    color("blue")for(p={sol1})p_line3dc(p,.1);  
    ''')
t1=time.time()
t1-t0
```

Out [186...]



In [180...]

```
a=random.random(1000)*(10-1)+1
b=random.random(1000)*(10-3)+3
```

In []:

```
a=random.random(1000)*(10-1)+1
b=random.random(1000)*(10-3)+3
px=array([a,b]).transpose(1,0)
p_l=array([a,b]).transpose(1,0)
p_l=p_l.tolist()
```

In [403...]

```
k,n=3,10
s1=ch1(pnts,k)
py=exclude_points(pnts,s1)
pz=exclude_points(py,pies1(s1,py))

while (pz!=[] and k<=n):
    k=k+1
    s1=ch1(pnts,k)
    py=exclude_points(pnts,s1)
    x=pies1(s1,py)
    pz=exclude_points(py,x) if x!=[] else py
```

In [227...]

```
# for x in arange(10):
#     for y in arange(10):
#         for z in arange(10):
#             print (x,y,z)
#             if x*y*z == 30:
#                 break
#             else:
#                 continue
#             break
#         else:
#             continue
#     break
```

sphere-through-parameteric-equation

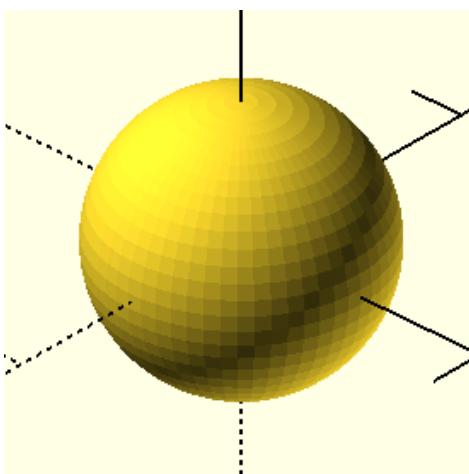
In [187...]

```
# sphere through parameteric equation
t0=time.time()
s=30
cp=[0,0,0]
theta=linspace(0,180,s)
phi=linspace(0,360,s*2)
r=5
x=lambda cp,r,theta,phi :cp[0]+r*sin(d2r(theta))*cos(d2r(phi))
y=lambda cp,r,theta,phi :cp[1]+r*sin(d2r(theta))*sin(d2r(phi))
z=lambda cp,r,theta :cp[2]+r*cos(d2r(theta))

sp1=[[x(cp,r,j,i),y(cp,r,j,i),z(cp,r,j)] for i in phi] for j in theta]

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>
{swp(sp1)}
''')
t1=time.time()
t1-t0
```

Out [187...]



cone-through-parameteric-equation

```
In [71]: # Cone through parametric equation
t0=time.time()

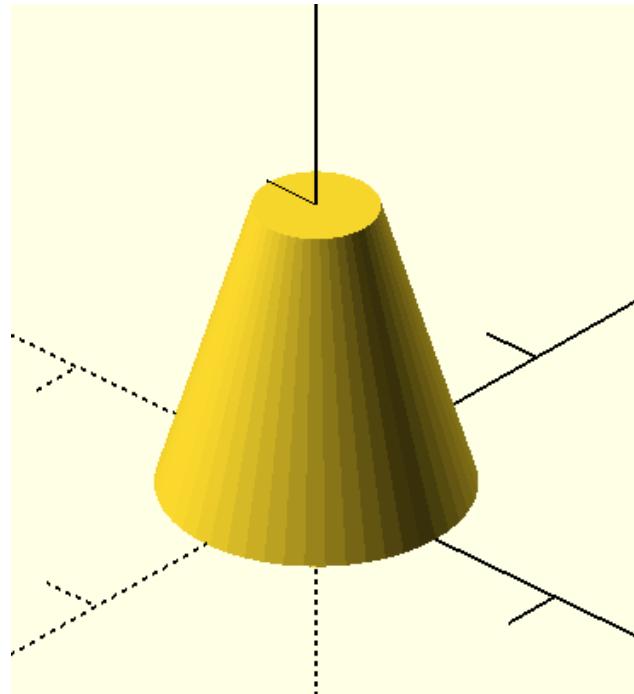
x=lambda cp,r,theta: cp[0]+r*cos(d2r(theta))
y=lambda cp,r,theta: cp[1]+r*sin(d2r(theta))
z=lambda cp,r:cp[2]-r

s=50
r1,r2=5,2 #bottom and the top radius respectively
r2=r2+.00001
theta=linspace(0,360,s)
r=linspace(r1,r2,10)
h=10
cp=[0,0,h/(r1-r2)*r2+h]

cn1=[[x(cp,i,j),y(cp,i,j),z(cp,i/(r1-r2)*h)] for j in theta] for i in r]

with open('trial.scad','w+') as f:
    f.write(f'''include<dependencies2.scad>
{swp(cn1)}
''')
t1=time.time()
t1-t0
```

Out[71]: 0.011339902877807617



torus-through-parameteric-equation

```
In [72]: # Torus through parametric equation
t0=time.time()
s=60
cp=[0,0,0]
theta=linspace(0,360,s)
phi=linspace(0,360,s*2)
R=6
r=2
x=lambda cp,r,theta,phi :cp[0]+(R+r*sin(d2r(theta)))*cos(d2r(phi))
y=lambda cp,r,theta,phi :cp[1]+(R+r*sin(d2r(theta)))*sin(d2r(phi))
z=lambda cp,r,theta :cp[2]+r*cos(d2r(theta))+.00001

sp1=[[x(cp,r,j,i),y(cp,r,j,i),z(cp,r,j)] for j in theta] for i in phi]

with open('trial.scad','w+') as f:
    f.write(f'''
```

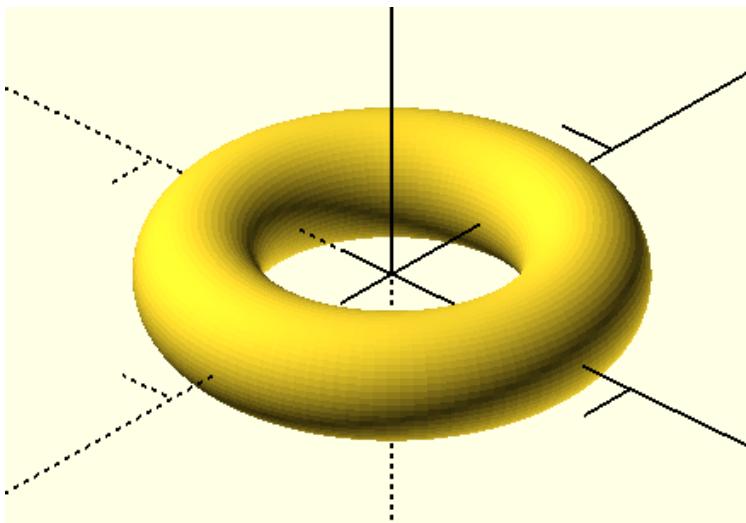
```

include<dependencies2.scad>

{swp_c(sp1)}
...
t1=time.time()
t1-t0

```

Out[72]: 0.17312073707580566



handling-trolley

In [188...]

```

# frame
sec1=circle(12.5)
path1=c2t3(corner_radius(pts1([[0,0,5],[900,0,5],[0,600,5],[-900,0,5]]),10))

sol1=path_extrude_closed(sec1,path1)

path2=cr_3d([[600,0,0,5],[0,600,0,5],[0,0,500,5],[0,-600,0,5]],10)
sol2=align_sol_1(path_extrude_closed(sec1,path2))

sol3=translate([-100,0,0],sol2)

# hinge supports
fillet1=fillet_l_cir([[0,75],[200,75]],circle(40,[100,120]),20)[1]
fillet2=flip(fillet_l_cir([[0,75],[200,75]],circle(40,[100,120]),20)[0])
arc1=arc_long_2p(fillet1[-1],fillet2[0],40,-1)
sec2=[[0,0],[200,0],[200,75]]+fillet1+arc1+fillet2+[[0,75]]
sol4=translate([450,250,465],q_rot(['x90'],linear_extrude(sec2,20)))
sol5=translate([450,350,465],q_rot(['x90'],linear_extrude(sec2,20)))

# long arm
x1=o_solid([-1,0,0],circle(30),1800,-900,290,590,[0,0,0])
x2=o_solid([-1,0,0],circle(25),1800,-900,290,590,[0,0,0])
sol6=swp_prism_h(x1,x2)
# sol6=axis_rot_1(sol6,[0,1,0],[550,290,590],10)

# frame
sol7=o_solid([0,1,0],circle(12.5),600,0,100)
sol8=o_solid([0,1,0],circle(12.5),600,0,200)

# cylinder body
x3=o_solid([0,0,1],circle(40),300,135,150,-290)
x4=o_solid([0,0,1],circle(35),300,135,150,-290)
sol9=swp_prism_h(x3,x4)
# sol9=axis_rot_1(sol9,[0,1,0],[150,290,100],1)

# cylinder bottom cover
x5=o_solid([0,0,1],circle(40),20,135,150,-290)
# x5=axis_rot_1(x5,[0,1,0],[150,290,100],1)

# hinge of cylinder
sec1=corner_radius(pts1([[-20,-35,20],[40,0,20],[0,70],[-40,0]]))
sol10=o_solid([0,1,0],sec1,20,-10+290,150,125-20)
# sol10=axis_rot_1(sol10,[0,1,0],[150,290,100],2)

```

```

# piston
x6=o_solid([0,0,1],circle(35),20,175,150,-290)
# x6=axis_rot_1(x6,[0,1,0],[150,290,100],1)

# piston rod
x7=o_solid([0,0,1],circle(15),350,175,150,-290)
# x7=axis_rot_1(x7,[0,1,0],[150,290,100],1)

# cylinder top cover
x8=o_solid([0,0,1],circle(40),20,425,150,-290)
# x8=axis_rot_1(x8,[0,1,0],[150,290,100],1)

# c-clamp for hinge support
sec1=corner_radius(pts1([[20,0,20],[40,0,20],[0,260,20],[-40,0,20]]),10)
sec1=equidistant_pathc(sec1,300)
sec2=offset(sec1,-19)

path1=corner_radius(pts1([[0,100],[0,-100,10],[70,0,10],[0,120]]),10)
path1=equidistant_path(path1,300)
path1=translate([0,0,0],q_rot(['x90','z90'],path1))
fold1=wrap_around(sec1,path1)

fold2=wrap_around(sec2,path1)
surf1=[fold1]+[fold2]
surf2=surf_offset(surf1,-5)
sol11=[surf1[1]]+[surf1[0]]+[surf2[0]]+[surf2[1]]
sol11=translate([150,255,520],sol11)
# sol11=axis_rot_1(sol11,[0,1,0],[550,290,590],10)
# sol11=axis_rot_1(sol11,[0,1,0],[170,290,670],-9)

# c-clamp for counterweight
x9=[surf1[1]]+[surf1[0]]+[surf2[0]]+[surf2[1]]
x9=translate([850,255,520],x9)

# catcher
arc1=c2t3(arc(200,-90,90,s=100))

sol12=path_extrude_open(circle(10),arc1)
sol12=translate([-1090,290,590],sol12)
# sol12=axis_rot_1(sol12,[0,1,0],[550,290,590],10)

# hinge pin c-clamp
sol13=o_solid([0,1,0],circle(5),100,240,150,590)
# sol13=axis_rot_1(sol13,[0,1,0],[550,290,590],10)

# hinge pin cylinder mounting
sol14=o_solid([0,1,0],circle(5),100,240,150,90)

# hinge pin long arm
sol15=o_solid([0,1,0],circle(5),140,220,550,590)

# hinge pin counterweight
x10=o_solid([0,1,0],circle(5),100,240,850,590)

# rod for counterweight
x11=o_solid([0,0,1],circle(15),300,225,850,-290)

# counterweight
s1=corner_radius(pts1([[100,20],[-120,0,19],[0,-40,19],[120,0]]),10)+arc_long_2p([100,-20],[100,20],10)
x12=o_solid([0,0,1],s1,100,235,850,-290,[90,0,0])

# support for counterweight
x13=o_solid([0,0,1],circle(100),10,225,850,-290)

# trolley wheel
s1=circle(15)
p1=q_rot(['x90'],circle(50))
wh=path_extrude_closed(s1,p1)
wh=align_sol_1(wh)
s2=circle(4)
p2=[[50,0,0],[50,0,0]]
spk=[axis_rot([0,1,0],path_extrude_open(s2,p2),i) for i in linspace(0,360,6)[:-1]]

```

```
with open('trial.scad','w+') as f:  
    f.write(f'''  
include<dependencies2.scad>  
// frame  
swp_c({sol1});  
swp_c({sol2});  
swp_c({sol3});  
swp({sol7});  
swp({sol8});  
  
// hinge supports  
swp({sol4});  
swp({sol5});  
  
for(i=[20,80])  
translate([-400,i,-500])  
swp({sol4});  
swp_c({sol6});  
  
//cylinder body  
sol9={sol9};  
%swp_c(sol9);  
//cylinder bottom cover  
x5={x5};  
swp(x5);  
  
//hinge for cylinder  
swp({sol10});  
  
//piston  
x6={x6};  
swp(x6);  
  
color("cyan")  
union(){  
//piston rod  
x7={x7};  
swp(x7);  
  
//cylinder top cover  
x8={x8};  
swp(x8);  
}  
  
// c-clamp for pivot point hinge support  
sol11={sol11};  
swp(sol11);  
  
// catcher  
sol12={sol12};  
swp(sol12);  
  
//hinge pin  
sol13={sol13};  
color("cyan")swp(sol13);  
  
//hinge pin cylinder  
sol14={sol14};  
color("cyan")swp(sol14);  
  
//hinge pin long arm  
sol15={sol15};  
color("cyan")swp(sol15);  
  
//c-clamp for counterweight  
x9={x9};  
swp(x9);  
  
// hinge pin counterweight  
x10={x10};  
color("cyan")swp(x10);  
  
// rod for counterweight  
x11={x11};
```

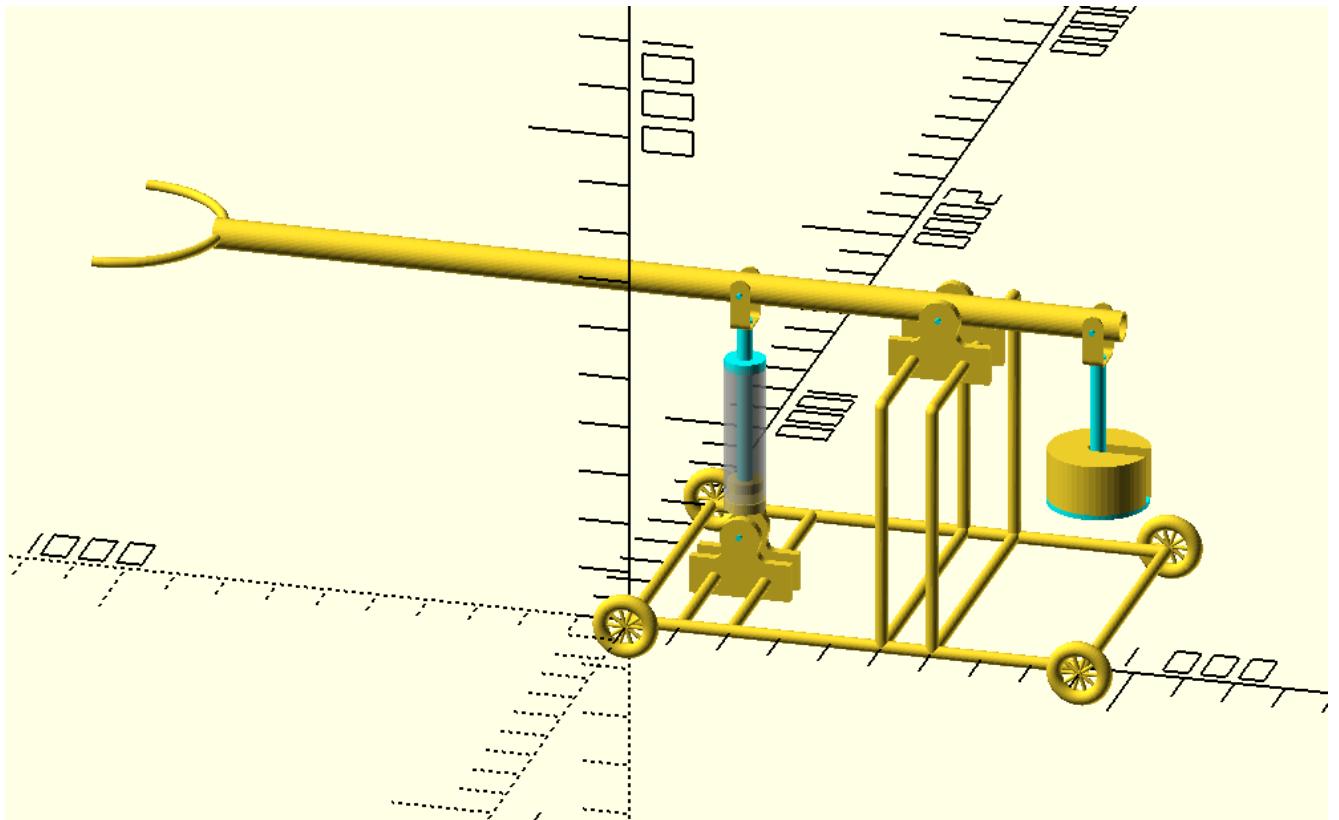
```

color("cyan")swp(x11);

//counterweight
x12={x12};
swp(x12);

//counterweight
x13={x13};
color("cyan")swp(x13);

//trolley wheels
for(j=[0,900])
for(i=[-30,630])
translate([j,i,0]){{}
swp_c({wh});
for(p={spk})swp(p);
}
}
'''
```



r_sec

```

In [189]: # example of function r_sec(r1,r2,cp1,cp2)

t0=time.time()

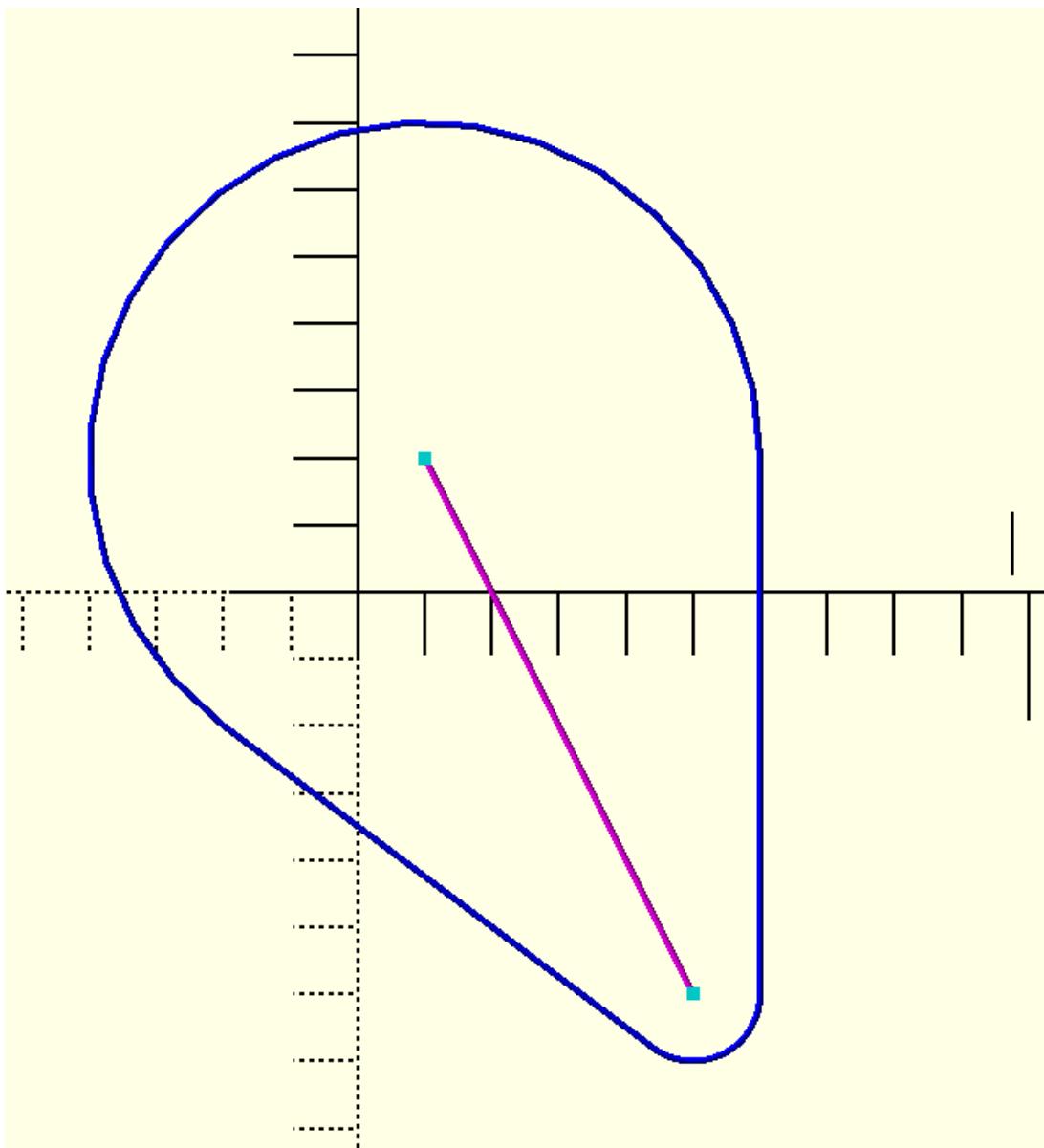
line=[[1,2],[5,-6]]
r1,r2=5,1

sec1=r_sec(r1,r2,line[0],line[1])

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>
color("blue")p_line3dc({sec1},.1);
color("magenta")p_line3d({line},.1);
color("cyan")points({line},.2);

''')
t1=time.time()
t1-t0
```

Out [189]: 0.004086017608642578



3d-knots

```
In [18]: # 3d knots various types
t0=time.time()
# trefoil knot

path=[[10*(sin(t)+2*sin(2*t)),
      10*(cos(t)-2*cos(2*t)),
      -10*sin(3*t)] for t in d2r(arange(0,360))]

# circular sin theta knot

# path=[[60*(cos(t)),
#        60*(sin(t)),
#        20*sin(4*t)*cos(4*t)] for t in d2r(arange(0,360))]

# random knot

# path=[[20*(-0.22*cos(t) - 1.28*sin(t) - 0.44*cos(3*t) - 0.78*sin(3*t)),
#        20*(-0.1*cos(2*t) - 0.27*sin(2*t) + 0.38*cos(4*t) + 0.46*sin(4*t)),
#        20*(0.7*cos(3*t) - 0.4*sin(3*t))] for t in d2r(arange(0,360))]

# torus knots

# path=[[10*cos(3*t)*(3+cos(4*t)),
#        10*sin(3*t)*(3+cos(4*t)),
#        10*sin(4*t)] for t in d2r(arange(0,360))]

# cinquefoil torus knots
a,p,q=3,11,12
d=10
```

```

# explanation
# radius of the torus = a*d
# section radius of the torus = d
# p in number of cycles of the wrapping coil over torus
# q in the number of turns of the wrapping coil over torus

path=[[d*cos(p*t)*(a+cos(q*t)),
d*sin(p*t)*(a+cos(q*t)),
-d*sin(q*t)] for t in d2r(arange(0,360,.25))]

# Lissajous knots

# path=[[10*cos(3*t+5),
# 10*cos(3*t+10),
# 10*cos(3*t+2)]for t in d2r(arange(0,360))]
r=2
sec=circle(r)
sol=align_sol_1(path_extrude_closed(sec,path))

sec1=circle(d-r)
path1=c2t3(circle(a*d))
sol1=path_extrude_closed(sec1,path1)

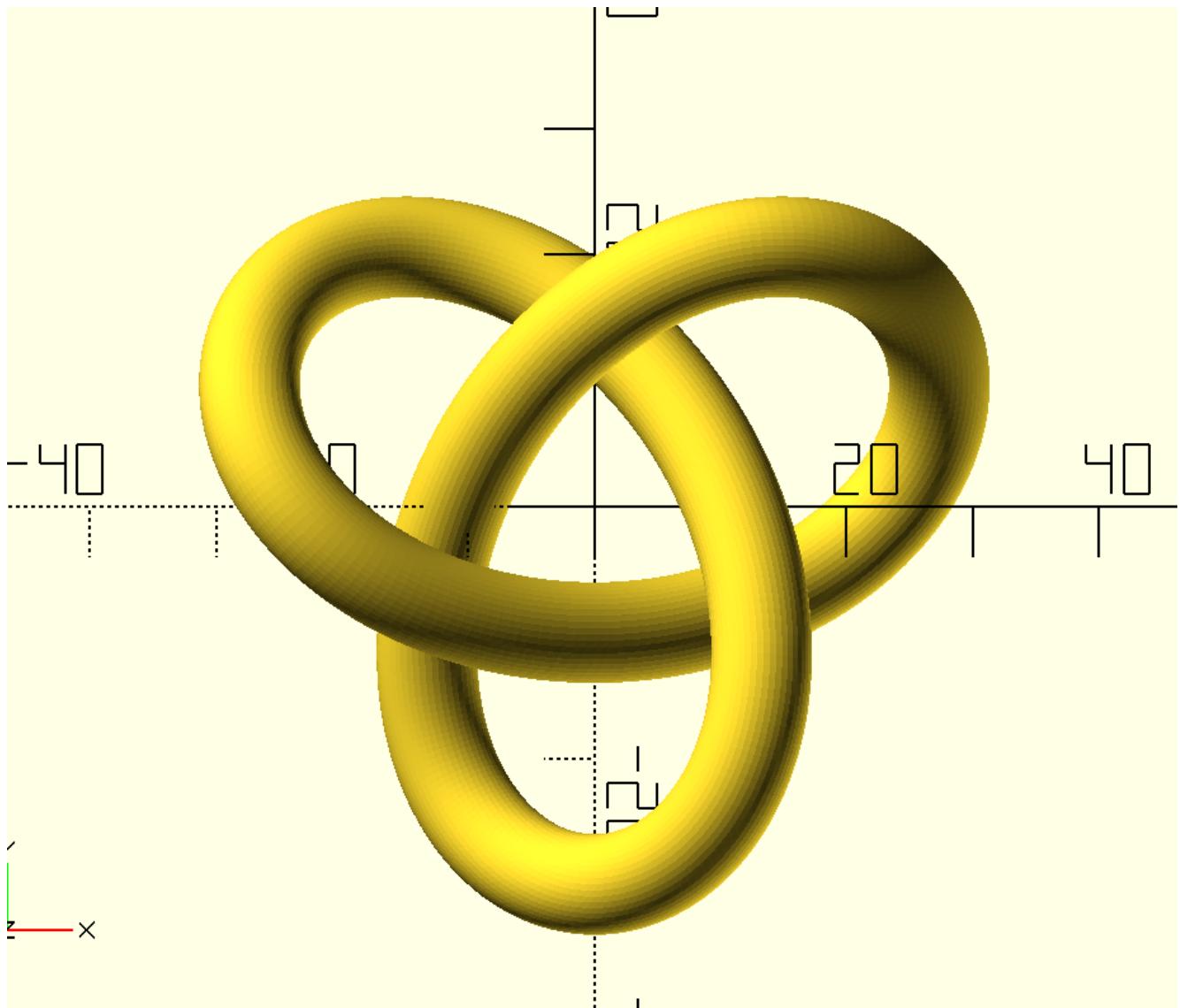
# sol2=o_solid([0,0,-1],circle(38),2,10)
# sol3=o_solid([0,0,-1],circle(20),2,10)
# sol2=swp_prism_h(sol2,sol3)

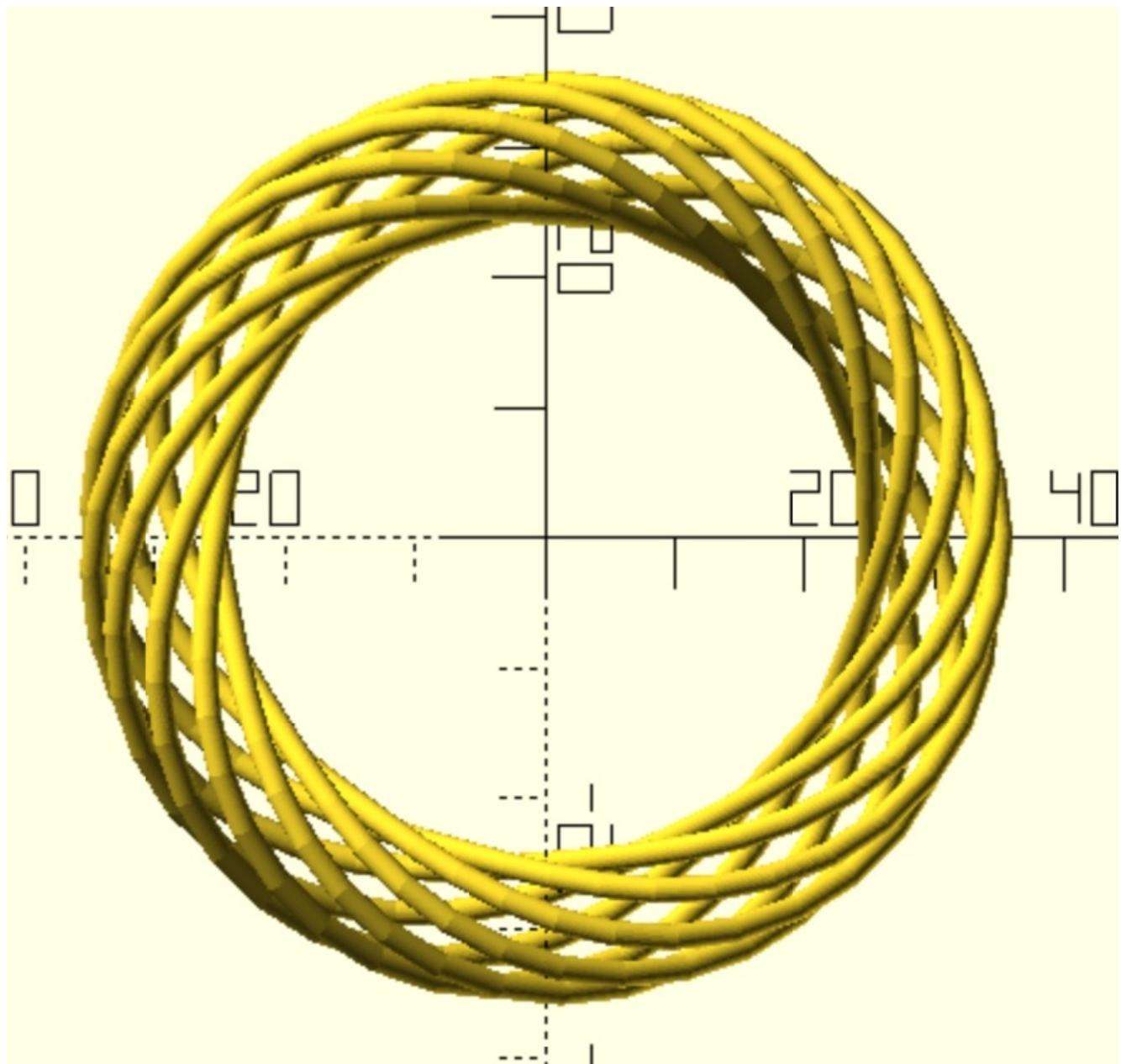
with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>
difference(){
{swp(sol)}
//{swp(cut_plane([0,-1,0],[100,100],100))}
}

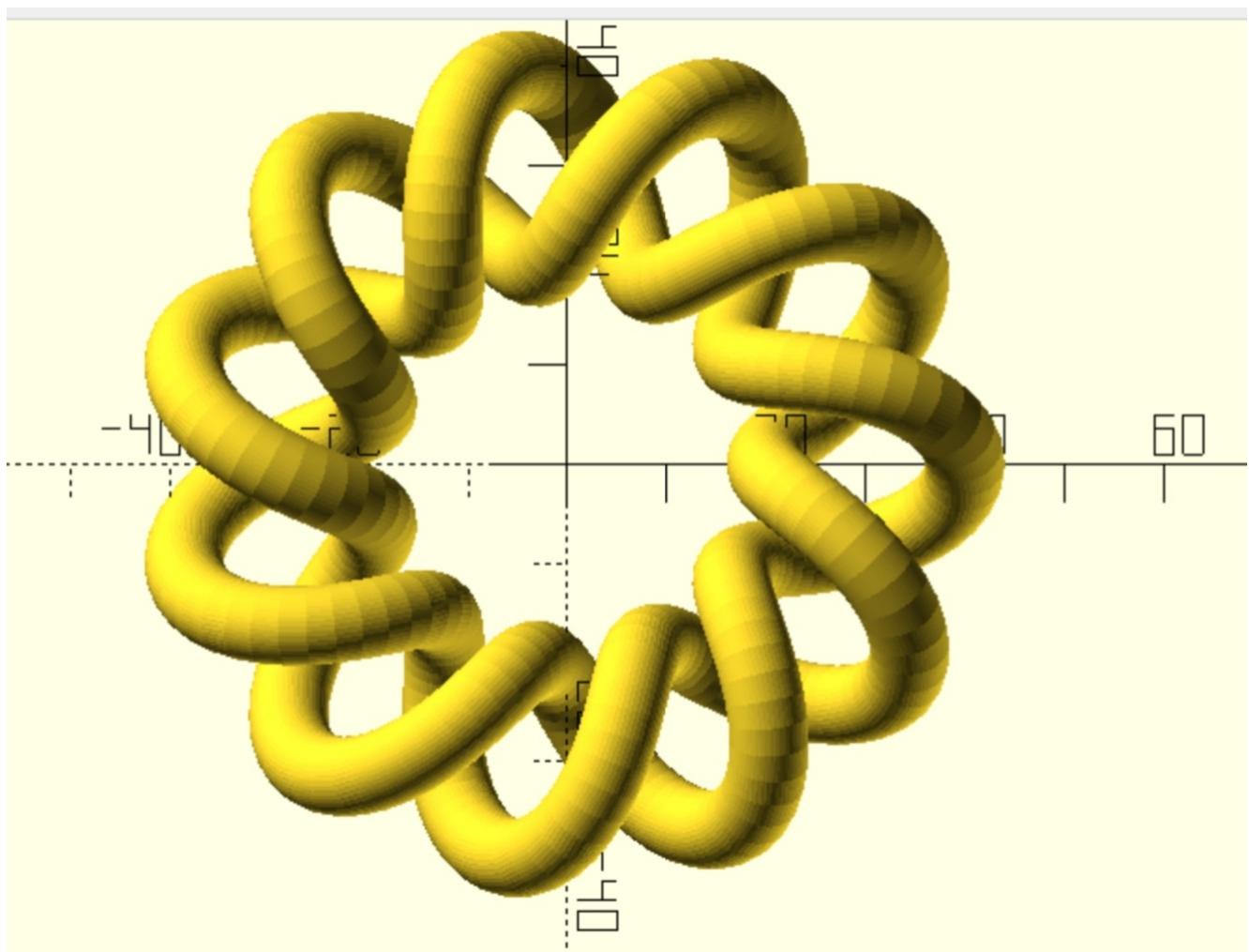
...)
t1=time.time()
t1-t0

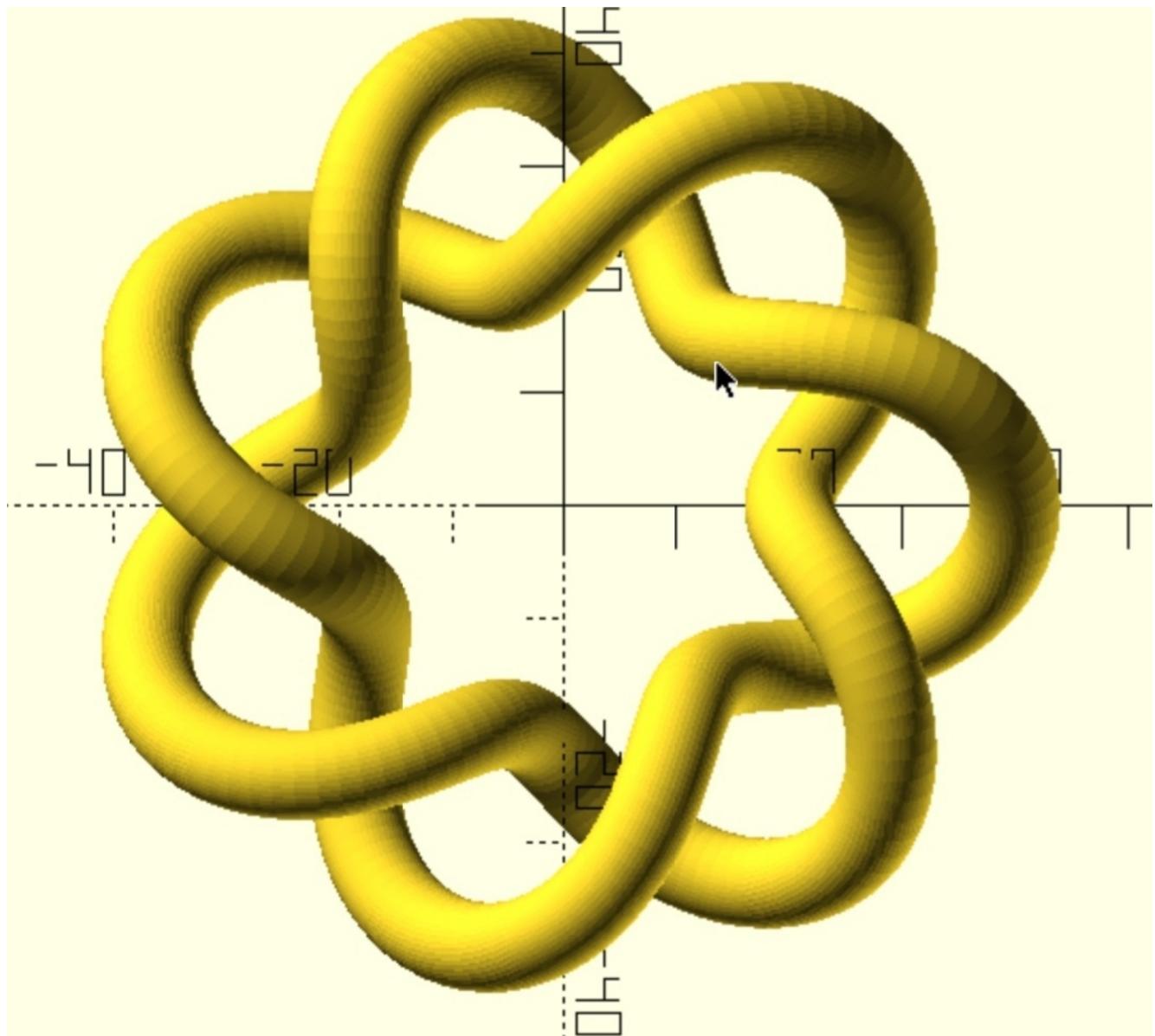
```

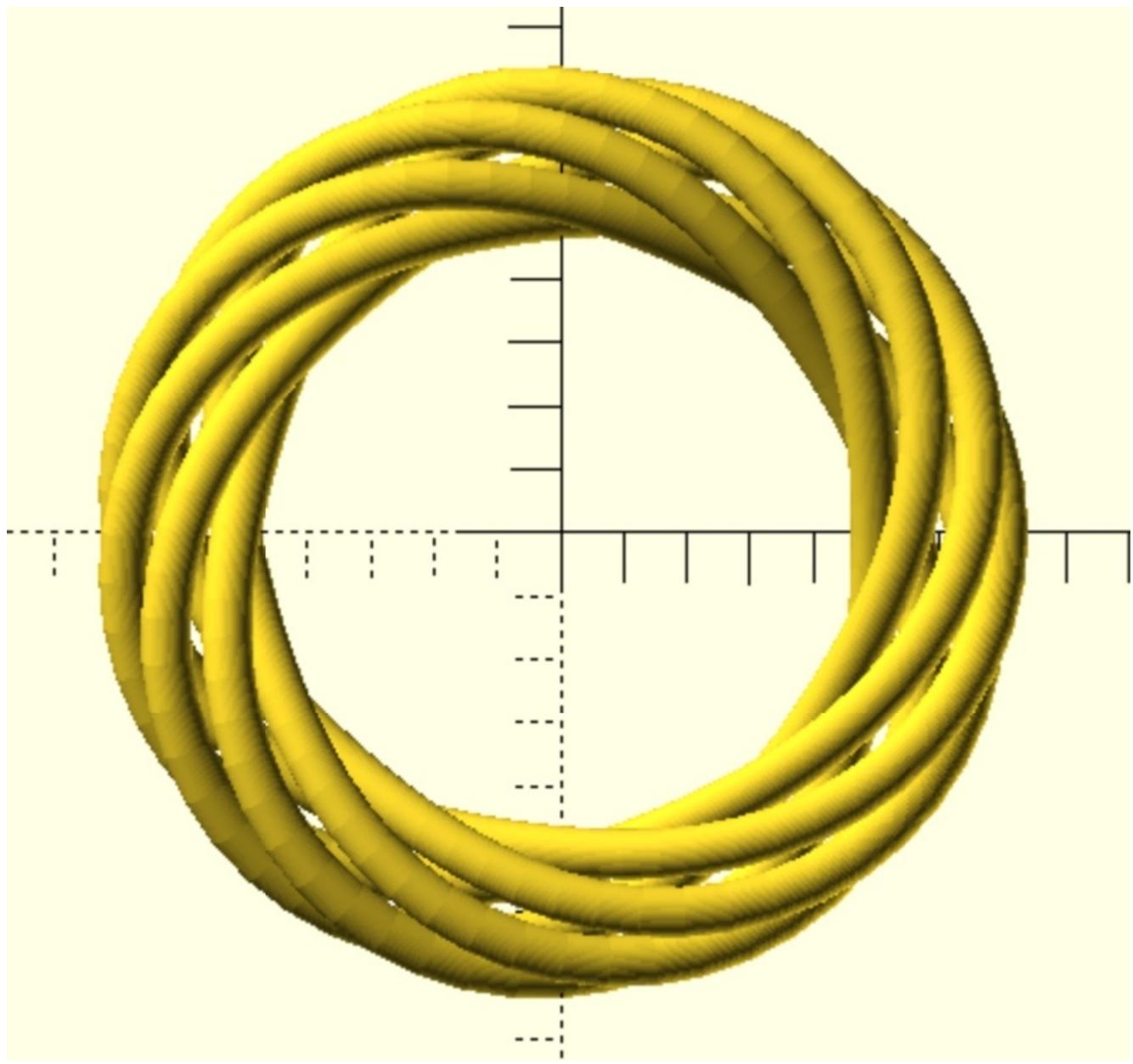
Out[18]: 3.011662006378174







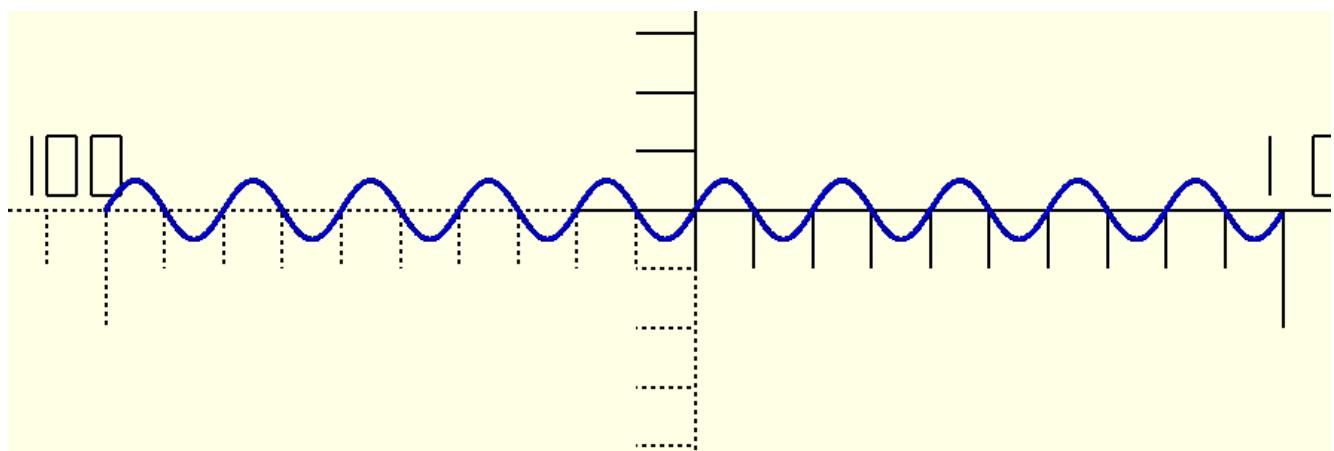




```
In [179...]: # sinwave
length=200
starting_point==100
number_of_waves=10
amplitude=5
a=[[r2d(i)/360*length+starting_point, amplitude*sin(number_of_waves*i)] for i in d2r(arange(0,360))]

with open('trial.scad','w+') as f:
    f.write(f''''
include<dependencies2.scad>
color("blue")p_line3d({a},1);

...)
```

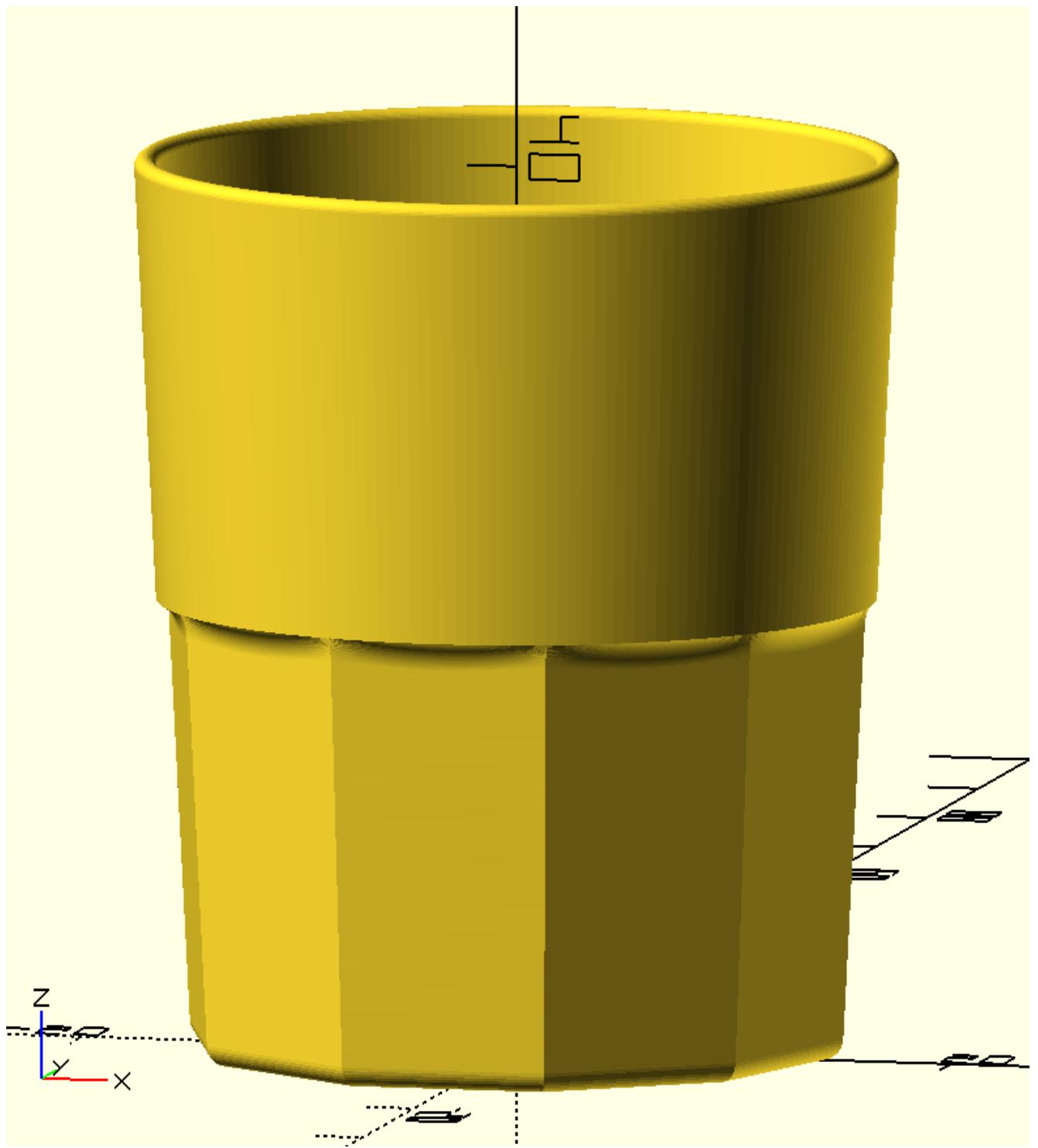


glass-model

In [9]:

```
# glass model
r=15
a1=1/20 #taper of the glass
h1=20
h2=20
s1=10
cir1=circle(r,s=s1+1)
path1=corner_radius(pts1([[-1,0],[1,0,1],[a1*h1,h1]]),10)
sol1=prism(cir1,path1)
sol1=align_sol_1([equidistant_pathc(p,200) for p in sol1])
sol1=slice_sol(sol1,30)

cir2=circle((r+a1*h1)/cos(d2r(360/(s1+1)/4)),s=201)
path2=corner_radius(pts1([[0,h1+.25],[a1*h2,h2,.49],[-1,0,.49],[-a1*h2,-h2,.5],[-.5,-.5,.5],[-a1*(h1-3
sol2=prism(cir2,path2)
sol3=sol1+sol2
p0=sol1[-1]
p1=sol2[0]
p2=sol1[-2]
fill=convert_3lines2fillet(p2,p1,p0,s=30)
fill=fill+[fill[0]]
sec=square(100,center=True)
cut1=o_solid([0,-1,0],sec,100,theta=[0,0,10])
with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>
difference() {{
union() {{
{swp(sol3)}
{swp_c(fill)}
}}
{swp(cut1)}
}}
'''')
```



`q(vector=[1,0,0],point=[0,5,0],theta=0)`

In [2]: `q(vector=[1,0,0],point=[0,5,0],theta=90)`

Out[2]: `[0.0, 8.881784197001252e-16, 5.0]`

`q_rot`

```
In [10]: line=[[0,0,0],[10,0,0]]
line1=q_rot(['z30','x90'],line)

pnt=[20,0]
pnt1=q_rot(['z45'],pnt)
with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>

p_line3d({line},1);
color("blue")p_line3d({line1},1);
color("magenta")points({[pnt]},1);
```

```
color("magenta")points({[pnt1]},1);  
...)
```

ang(x,y)

```
In [196... # example of function ang(x,y)  
pnt=[20,0]  
pnt1=q_rot(['z125'],pnt)  
ang(pnt1[0],pnt1[1])
```

```
Out[196... 125.0
```

l_len

```
In [197... l_len([[0,0,0],[10,0,0]])
```

```
Out[197... 10.0
```

l_lenv

```
In [198... l_lenv([[0,0,0],[10,0,0],[10,5,0],[0,5,0]])
```

```
Out[198... 30.0
```

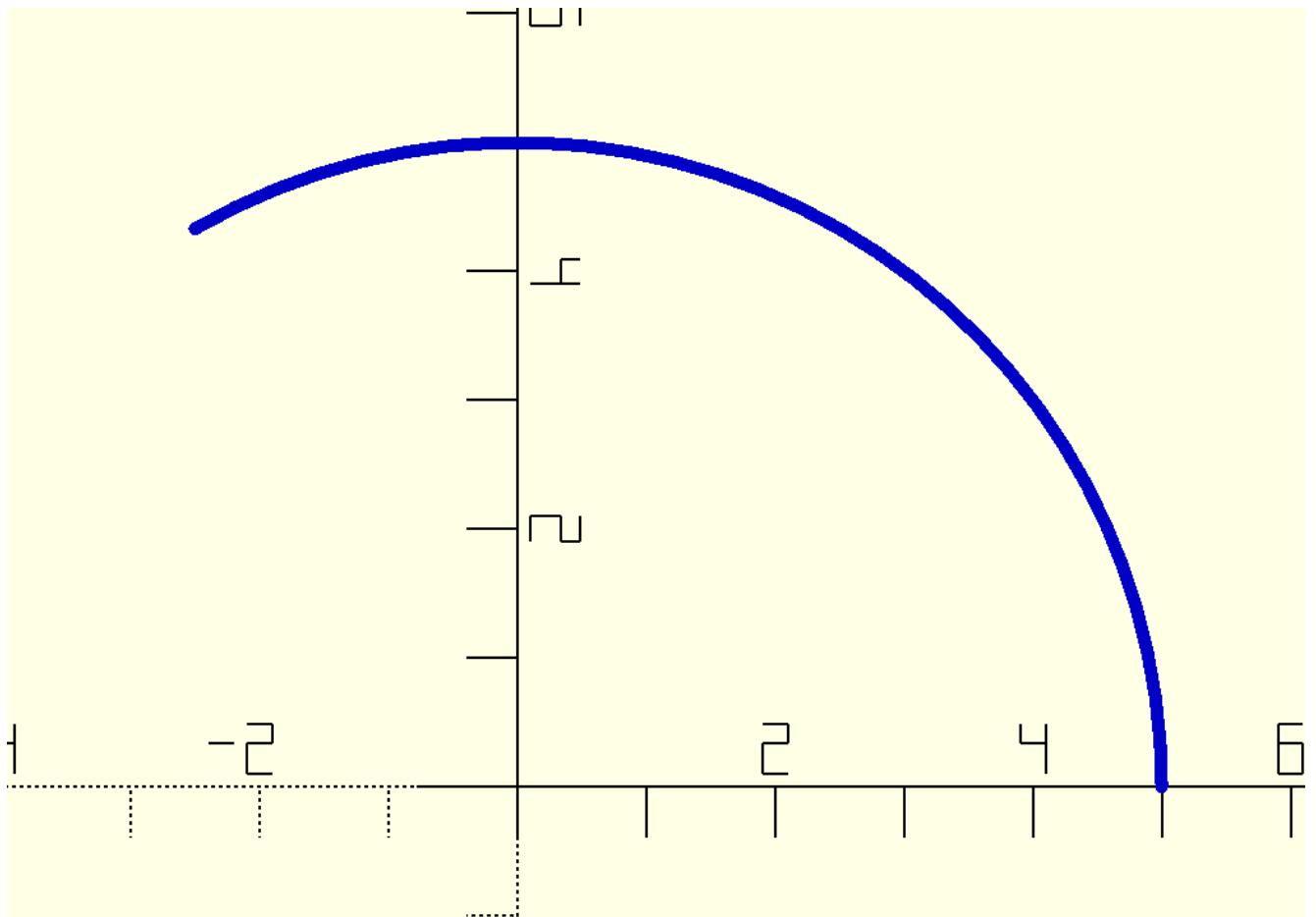
l_lenv_o

```
In [199... l_lenv_o([[0,0,0],[10,0,0],[10,5,0],[0,5,0]])
```

```
Out[199... 25.0
```

arc

```
In [200... arc1=arc(radius=5,start_angle=0,end_angle=120,cp=[0,0],s=30)  
with open('trial.scad','w+') as f:  
    f.write(f'''  
include<dependencies2.scad>  
  
color("blue")p_lineo({arc1},.1);  
...)
```



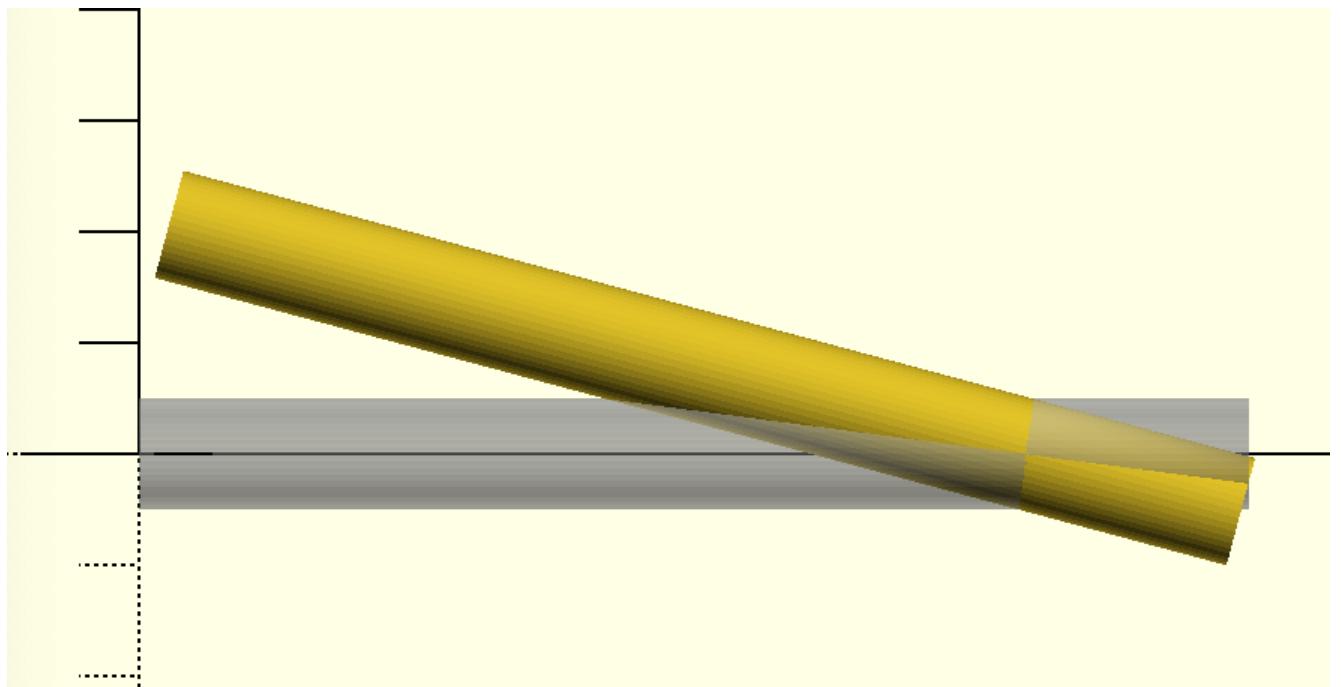
pts

```
In [201]: # calculates the cumulative sum of points  
pts([[0,0],[4,0],[2,3],[5,-8]])
```

```
Out[201]: [[0, 0], [4, 0], [6, 3], [11, -5]]
```

axis_rot_1

```
In [11]: sol=o_solid([1,0,0],circle(5),100)  
sol1=axis_rot_1(sol,[0,1,0],[80,0,0],15)  
with open('trial.scad','w+') as f:  
    f.write(f'''  
include<dependencies2.scad>  
  
%{swp(sol)}  
{swp(sol1)}  
'''')
```



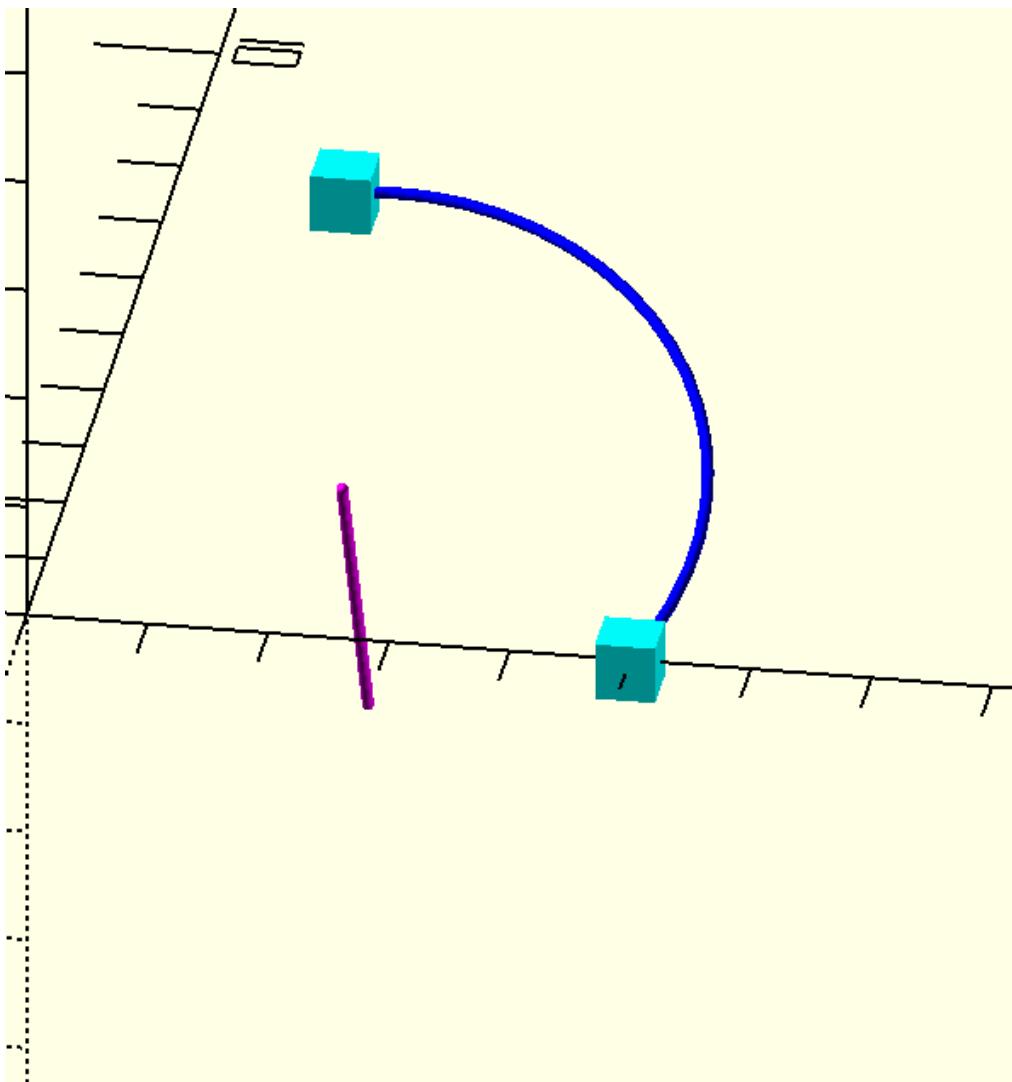
arc_2p_3d

arc_2p_3p_cp

In [9]:

```
'''  
    draws an arc through 2 points  
    n1: normal vector to define plane on which the arc will be drawn  
    r: radius of the arc  
    cw: '1' stands for clockwise and '-1' stands for counter-clockwise  
    's' is the number of segments of the circle  
'''  
  
p0=[5,0,0]  
p1=[2,4,2]  
p2=[0,0,0]  
n1=nv([p0,p1,p2])  
n2=(array(n1)*3).tolist()  
arc1=arc_2p_3d(n1,p0,p1,3,-1)  
cp=arc_2p_3d_cp(n1,p0,p1,3,-1)  
n_line=[cp,(array(cp)+array(n2)).tolist()]  
  
with open('trial.scad', 'w+') as f:  
    f.write(f'''  
include<dependencies2.scad>  
color("blue")p_line3d({arc1},.1);  
color("magenta")p_line3dc({n_line},.1);  
color("cyan")points({[p0,p1]},.5);  
'''')  
n1
```

Out [9]: [0.0, 0.4472135954999579, -0.8944271909999159]



c2t3

```
In [10]: # function to convert 2d to 3d, it just adds the z-coordinate to the points list
#     example:
list=c2t3([[1,2],[3,4],[6,7]])
list
```

```
Out[10]: [[1, 2, 0], [3, 4, 0], [6, 7, 0]]
```

c3t2

```
In [187...]: # function to convert 3d to 2d, it just removes the z-coordinate from the points list
#     example:
list=c3t2([[1,2,3],[3,4,5],[6,7,8]])
list
```

```
Out[187...]: [[1, 2], [3, 4], [6, 7]]
```

nv

```
In [188...]: # given 3 points ['p1','p2',p3] function calculates unit normal vector
#     example:
p1,p2,p3=[1,0,0],[0,10,0],[-5,0,0]
nv([p1,p2,p3]) #=> [0.0, 0.0, -1.0]
```

```
Out[188...]: [0.0, 0.0, -1.0]
```

cytz

```
In [189]: #      function to convert the y co-ordinates to z co-ordinates e.g.[x,y]=>[x,0,y]. 2d to 3d coordinate
list=cytz([[1,2],[3,4],[6,7]])
list
```

```
Out[189]: [[1, 0, 2], [3, 0, 4], [6, 0, 7]]
```

d2r

```
In [190]: # function to convert from degrees to radians
d2r(90)
```

```
Out[190]: 1.5707963267948966
```

r2d

```
In [191]: # function to convert from radians to degrees
r2d(1.57079)
```

```
Out[191]: 89.99963750135457
```

flip

```
In [11]: # function to flip the sequence of a list or a list of points
#      example:
list=[1,2,3,4,5]
flipped_list1=flip(list) #=> [5, 4, 3, 2, 1]

list=[[1,2,3],[4,5,6],[7,8,9]]
flipped_list2=flip(list) #=> [[7, 8, 9], [4, 5, 6], [1, 2, 3]]
flipped_list1, flipped_list2
```

```
Out[11]: ([5, 4, 3, 2, 1], [[7, 8, 9], [4, 5, 6], [1, 2, 3]])
```

gcd

```
In [212]: # calculates the greatest common divisor of 2 numbers 'a','b'
gcd(12,15)
```

```
Out[212]: 3
```

lcm

```
In [213]: # calculates the least common multiple of 2 numbers 'a','b'
lcm(12,15)
```

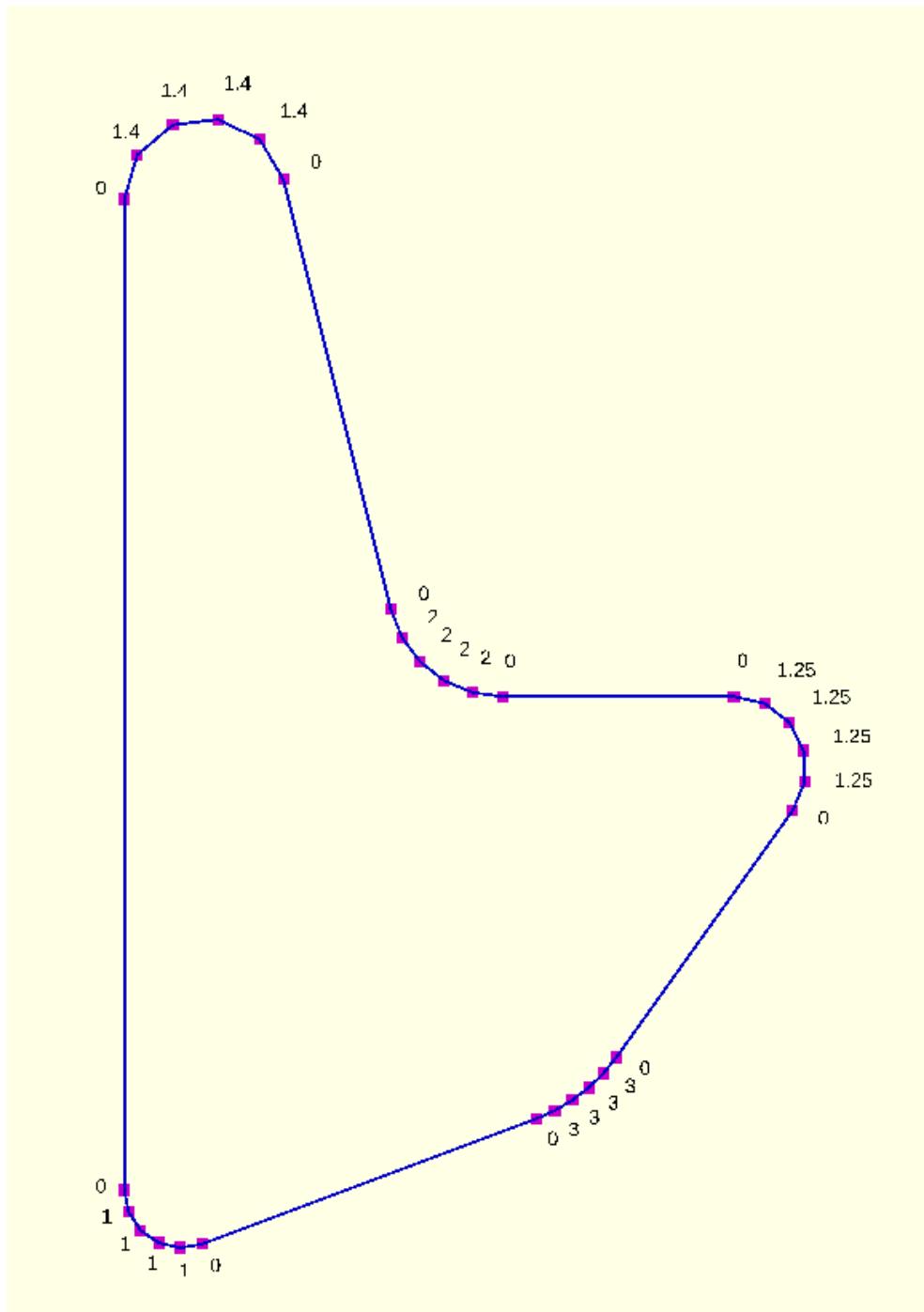
```
Out[213]: 60.0
```

list_r

```
In [12]: #      function list the corner radiiuses of a given section (only where the radius is specified)
#      example:
sec=corner_radius(pts1([[0,0,1],[8,3,3],[5,7,1.25],[-8,0,2],[-5,20,1.4]]),5)
r1=list_r(sec)
with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>
sec={sec};
r1={r1.tolist()};
color("blue")p_line({sec},.05);
color("magenta")points({sec},.2);
color("black")for(i=[0:len(sec)-1])translate({offset(sec,.5)}[i])text(str(r1[i]),.25);
```

```
r1
```

```
Out[12]: array([0. , 1. , 1. , 1. , 1. , 0. , 0. , 3. , 3. , 3. , 3. ,  
    0. , 0. , 1.25, 1.25, 1.25, 1.25, 0. , 0. , 2. , 2. , 2. , 2. ,  
    2. , 0. , 0. , 1.4 , 1.4 , 1.4 , 1.4 , 0. ])
```



ls(line,n)

```
In [194... # function to draw number of points 'n' in a line 'line'  
#     example:  
line=[[0,0],[10,0]]  
line1=ls(line,5) #=> [[0.0, 0.0], [2.0, 0.0], [4.0, 0.0], [6.0, 0.0], [8.0, 0.0], [10.0, 0.0]]  
line1
```

```
Out[194... [[0.0, 0.0], [2.0, 0.0], [4.0, 0.0], [6.0, 0.0], [8.0, 0.0]]
```

max_r(sec)

```
In [195... # function calculates the maximum radius in a given closed section  
#     example:  
sec=cr_c(pts1([[0,0,.2],[8,3,3],[5,7,1],[-8,0,2],[-5,20,1]]),20)  
max_r(sec) #=> 3.0
```

arc_long_2p_3d

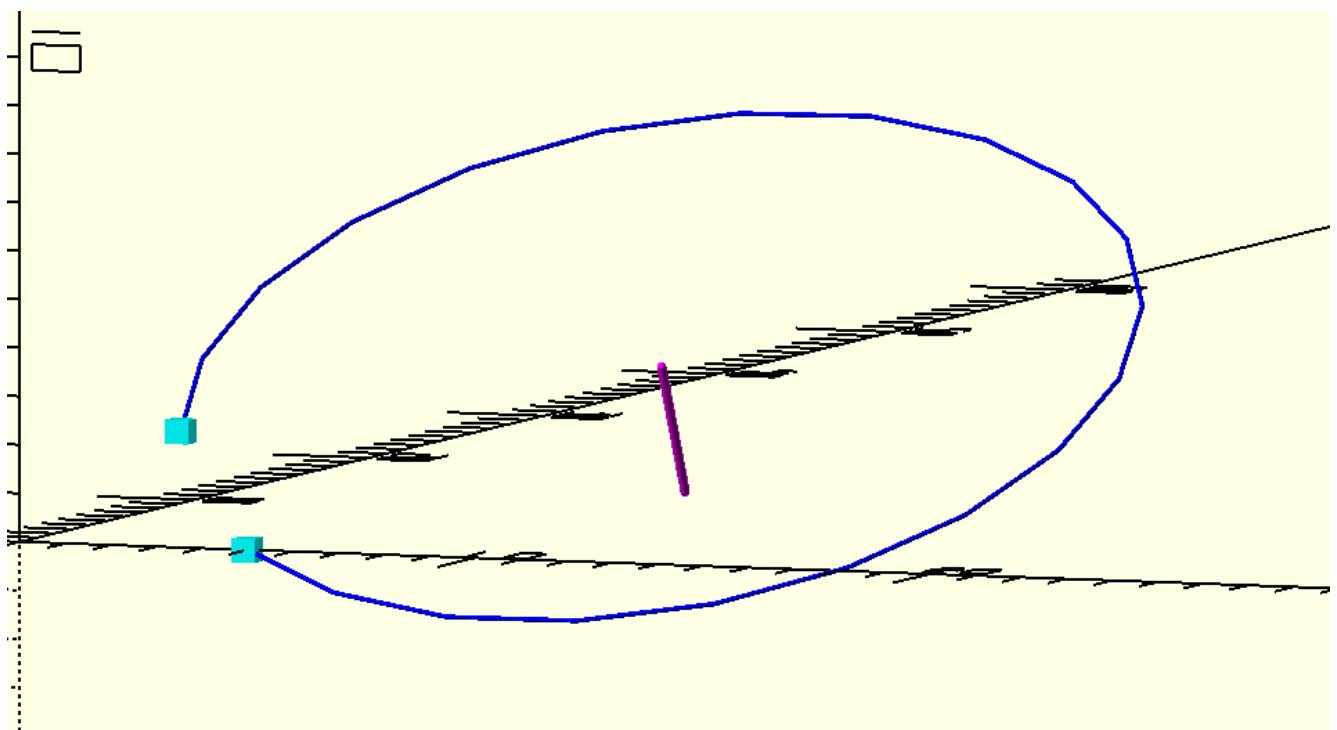
```
In [14]: # draws a long arc through 2 points
#      n1: normal vector to define plane on which the arc will be drawn
#      r: radius of the arc
#      cw: '1' stands for clockwise and '-1' stands for counter-clockwise
#      's' is the number of segments of the circle

p0=[5,0,0]
p1=[2,4,2]
p2=[0,0,0]
n1=nv([p0,p1,p2])
n2=(array(n1)*3).tolist()
arc1=arc_long_2p_3d(n1,p0,p1,10,-1)

cp=arc_2p_3d_cp(n1,p0,p1,10,1)
n_line=[cp,(array(cp)+array(n2)).tolist()]

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>
color("blue")p_line3d({arc1},.1);
color("magenta")p_line3dc({n_line},.2);
color("cyan")points([{p0,p1}],.5);

''' )
```



```
In [15]: t0=time.time()

sec=pts([[0,0],[5,0],[0,5],[-5,0]])
path=corner_radius(pts1([[-2.5,0],[2.5,0,0.25],[0,5.01,0.25],[-2.5,0]]),10)
sol=prism(sec,path)

v1=[2,0,7]
sec1=axis_rot([0,0,1],circle(1.5,s=100),90)
sol1=o_solid(v1,sec1,10,-1.5,-2.5,0)
sol2=[ip_sol2line(sol,p)[0] if ip_sol2line(sol,p)!=[] else p[1] for p in cpo(sol1) ]

sol2=translate([.01,0,0.01],[sol1[0]]+[sol2])
fillet1=fillet_sol2sol(sol,sol2,.3,o=1)[:2]
fillet1=flip(fillet1)

sol3=[ip_sol2line(sol,p)[0] if ip_sol2line(sol,p)!=[] else p[1] for p in cpo(flip(sol1))]
sol3=[flip(sol1)[0]]+[sol3]
sol3=translate([0.01,0,-.01],sol3)
fillet2=fillet_sol2sol(sol,sol3,.3)[1:-2]
```

```

fillet3=fillet1+fillet2+[fillet1[0]]

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>

{swp(sol)}
{swp(sol1)}
{swp_c(fillet3)}
//{swp_c(fillet2)}
//color("blue")p_line3dc({fillet1[0]},.05);
//color("blue")p_line3dc({fillet2[0]},.05);

'''')
t1=time.time()
t1-t0

```

Out[15]: 0.6492040157318115

ang_2lineccw

```

In [16]: p0=[0,0]
p1=[5,2]
p2=[7,8]

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>

// case 1: counter-clockwise angle between line p1p0 and p1p2 is 229.76 degrees
color("blue")points({[p0,p1,p2]},.2);
color("cyan")p_line3d({[p0,p1,p2]},.1);
color("magenta")translate([0,-.5]){{{
translate({p0})text("p0",.5);
translate({p1})text("p1",.5);
translate({p2})text("p2",.5);
}}}

// case 2: counter-clockwise angle between line p0p1 and p0p2 is 27.01 degrees

translate([10,0,0]){{{
color("blue")points({[p0,p1,p2]},.2);
color("cyan")p_line3d({[p1,p0,p2]},.1);

color("magenta")translate([0,-.5]){{{
translate({p0})text("p0",.5);
translate({p1})text("p1",.5);
translate({p2})text("p2",.5);
}}}
}}}

'''')
ang_2lineccw(p1,p0,p2),ang_2lineccw(p0,p1,p2)

```

Out[16]: (229.76364169072616, 27.012665347938547)

ang_2linecw

```

In [17]: p0=[0,0]
p1=[5,2]
p2=[7,8]

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>

// case 1: clockwise angle between line p1p0 and p1p2 is 130.23 degrees
color("blue")points({[p0,p1,p2]},.2);
color("cyan")p_line3d({[p0,p1,p2]},.1);
color("magenta")translate([0,-.5]){{{
translate({p0})text("p0",.5);
}}
```

```

translate({p1})text("p1",.5);
translate({p2})text("p2",.5);
}

// case 2: clockwise angle between line p0p1 and p0p2 is 332.98 degrees

translate([10,0,0]){{{
color("blue")points({[p0,p1,p2]},.2);
color("cyan")p_line3d({[p1,p0,p2]},.1);

color("magenta")translate([0,-.5]){{{
translate({p0})text("p0",.5);
translate({p1})text("p1",.5);
translate({p2})text("p2",.5);
}}}
}}}

''')
ang_2linecw(p1,p0,p2),ang_2linecw(p0,p1,p2)

```

Out[17]: (130.23635830927384, 332.98733465206146)

cir_2p

cp_arc

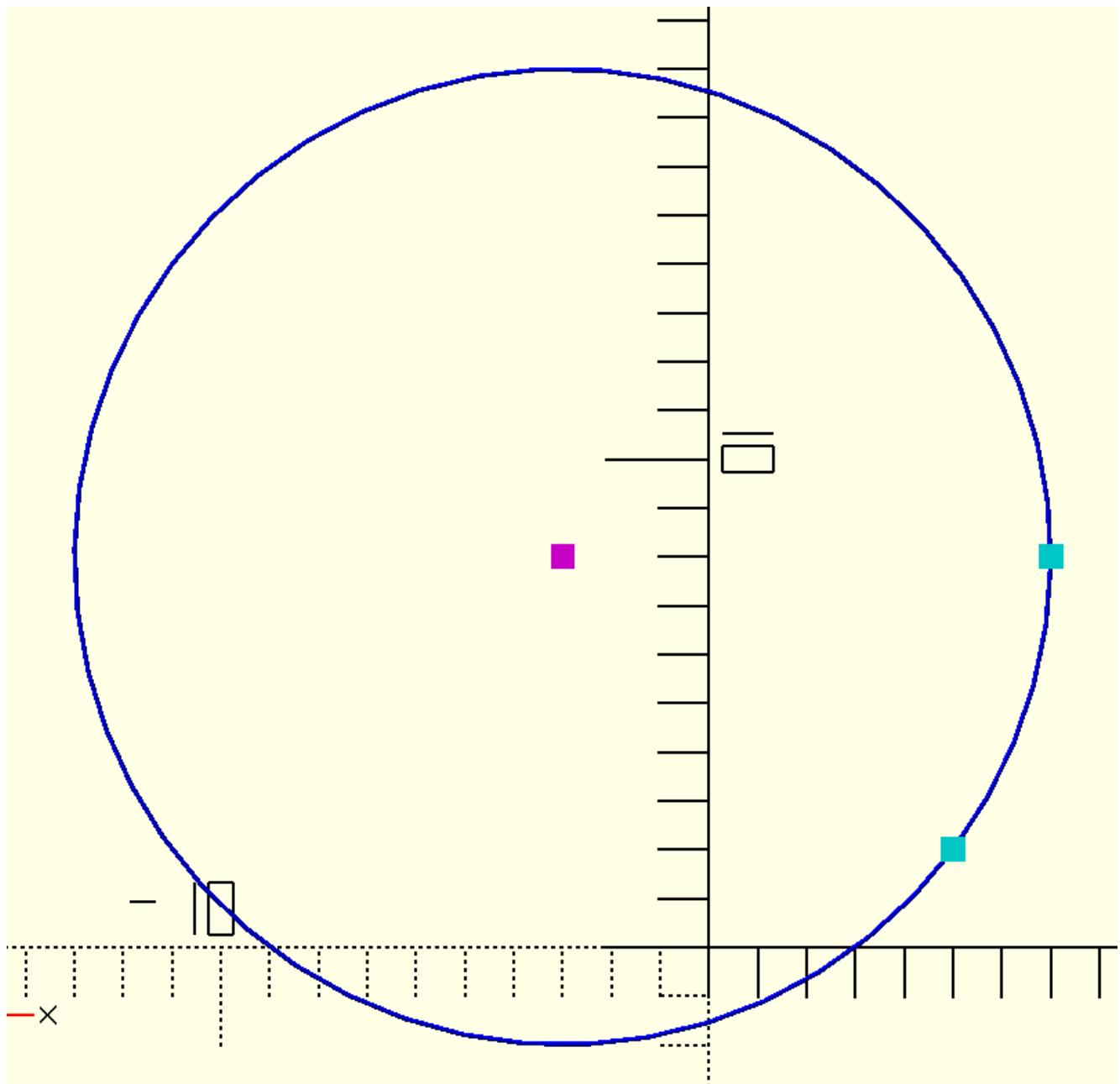
```

In [18]: p1=[5,2]
p2=[7,8]
r=10
c1=cir_2p(p1,p2,r,cw=-1,s=50)
cp1=cp_arc(c1)
with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>

color("cyan")points({[p1,p2]},.5);
color("blue")p_line3dc({c1},.1);
color("magenta")points({[cp1]},.5);

'''')

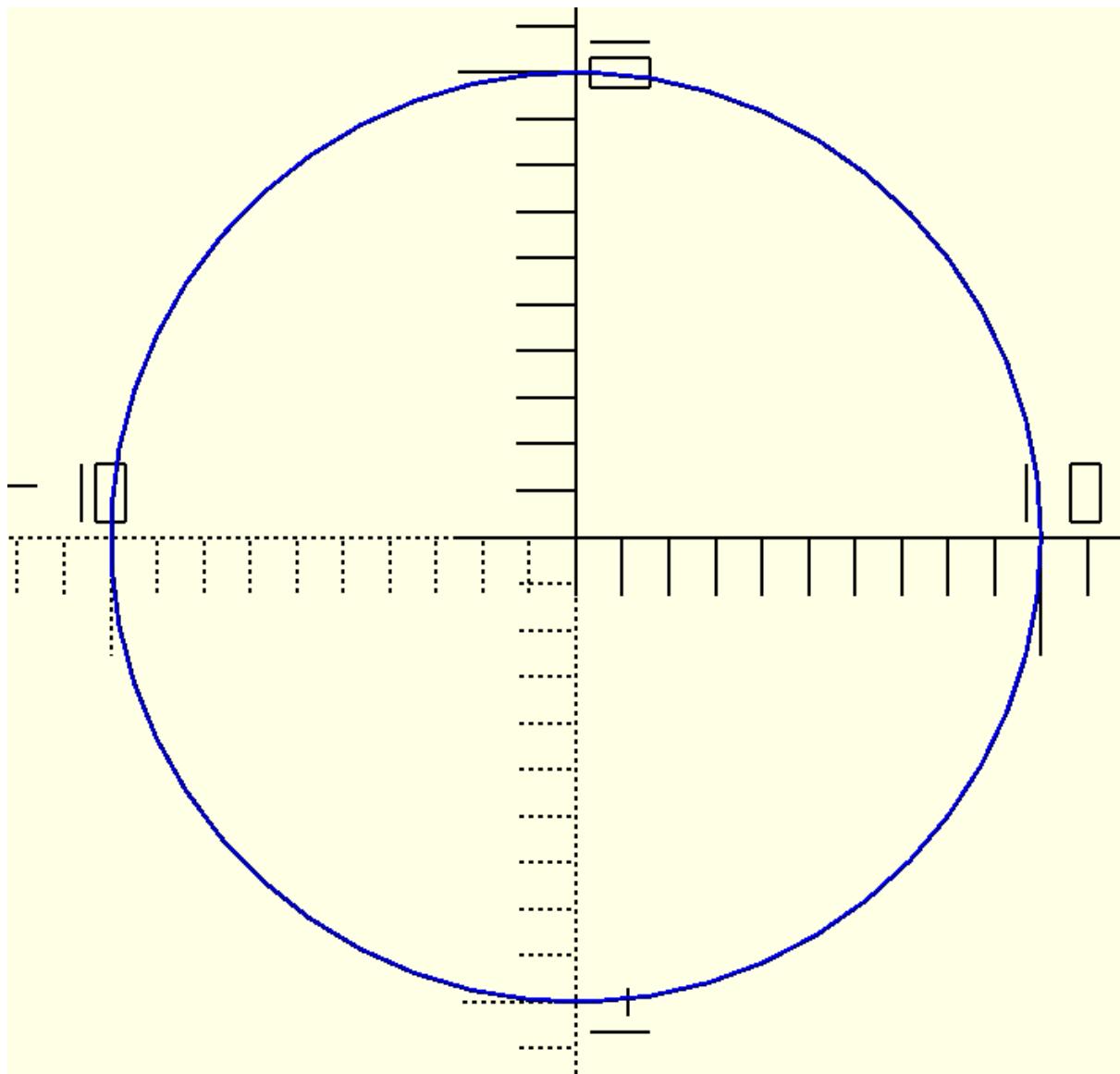
```



circle

```
In [201]: c1=circle(r=10,cp=[0,0],s=50)
with open('trial.scad','w+') as f:
    f.write(f''''
include<dependencies2.scad>
color("blue")p_line3dc({c1},.1);

...)
```



CW

```
In [202]: sec=corner_radius(pts1([[0,0,1],[8,3,3],[5,7,1],[-8,0,2],[-5,20,1]]),20)
cw(sec)
```

```
Out[202]: -1
```

CWV

```
In [203]: sec=corner_radius(pts1([[0,0,1],[8,3,3],[5,7,1],[-8,0,2],[-5,20,1]]),20)
cwv(sec)
```



```
1,
1,
1,
1,
1,
1,
1,
-1,
-1,
-1,
-1,
-1,
-1,
-1,
-1,
-1,
-1,
-1,
-1,
-1,
-1,
-1,
-1,
-1,
-1,
-1,
-1,
-1,
-1,
-1,
-1]
```

exclude_points

```
In [204]: p0=[[ 0,  1,  2],[ 3,  4,  5],[ 6,  7,  8],[ 9, 10, 11],[12, 13, 14],[15, 16, 17],
[18, 19, 20],[21, 22, 23],[24, 25, 26],[27, 28, 29]]
p1=[[0,1,2],[9,10,11]]
p2=exclude_points(p0,p1)
p2
```

```
Out[204]: [[3, 4, 5],
[6, 7, 8],
[12, 13, 14],
[15, 16, 17],
[18, 19, 20],
[21, 22, 23],
[24, 25, 26],
[27, 28, 29]]
```

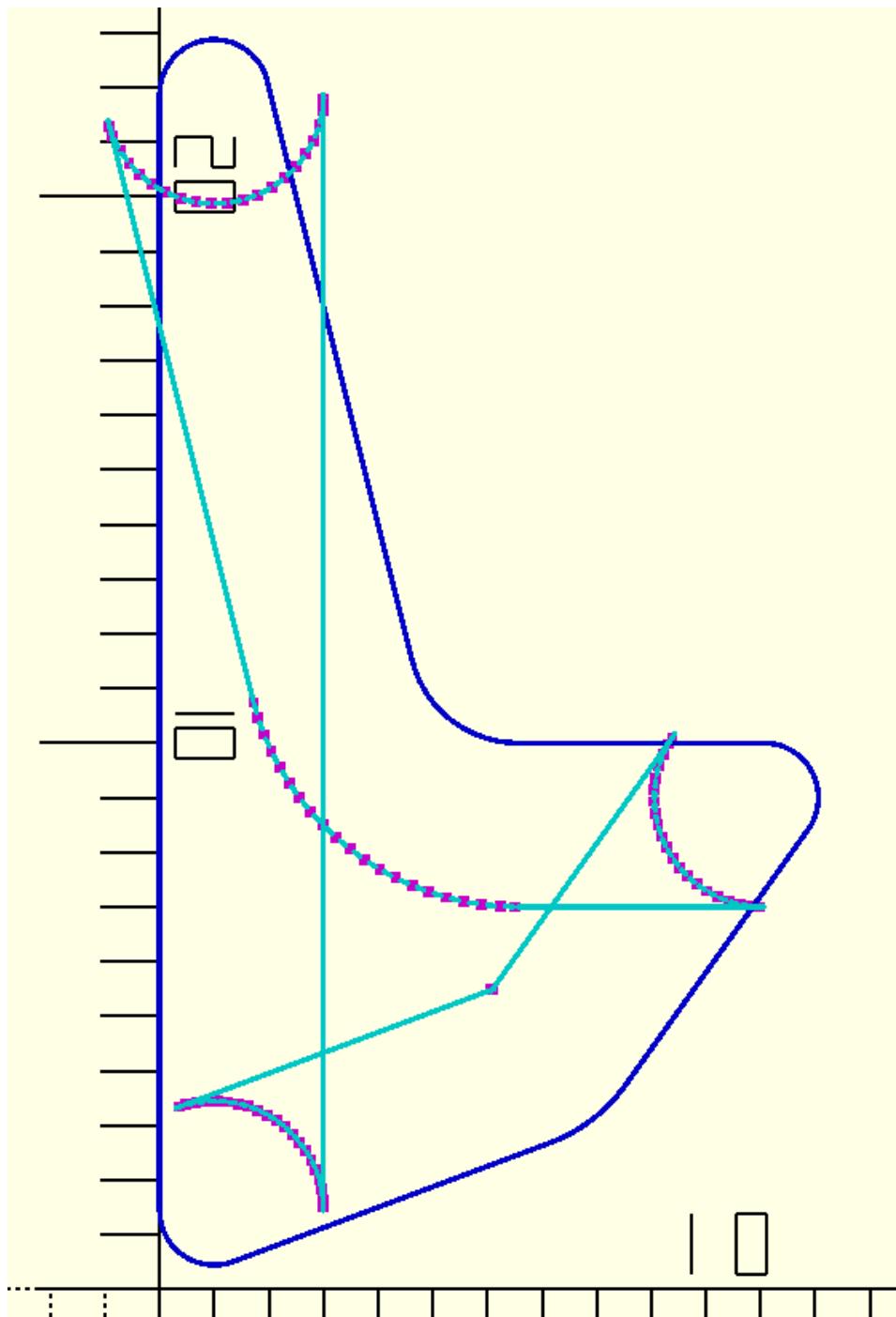
intersections

```
In [19]: sec=corner_radius(pts1([[0,0,1],[8,3,3],[5,7,1],[-8,0,2],[-5,20,1]]),20)
sec=sec=corner_radius(pts1([[0,0,.1],[7,5,2],[5,7,2],[-5,7,2],[-7,5,3]]),20)
sec=corner_radius(pts1([[-15,0,2.49],[0,15,3],[30,0,3],[0,-15,2.49],[5,0,2.49],[0,20,7],[-40,0,7],[0,-10,0]]),
# sec=circle(10)
# sec=pts([[0,0],[10,0],[0,10],[-10,0]])

offset_line_segments=offset_segv(sec,-2.5)
intersection_points=intersections(offset_line_segments)

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>
color("blue")p_line({sec},.1);
color("cyan")for(p={offset_line_segments})p_line(p,.1);
color("magenta")points({intersection_points},.2);

...)
```



```
In [20]: c1=c2t3(circle(10))
c2=translate([0,0,10],circle(5,s=70))

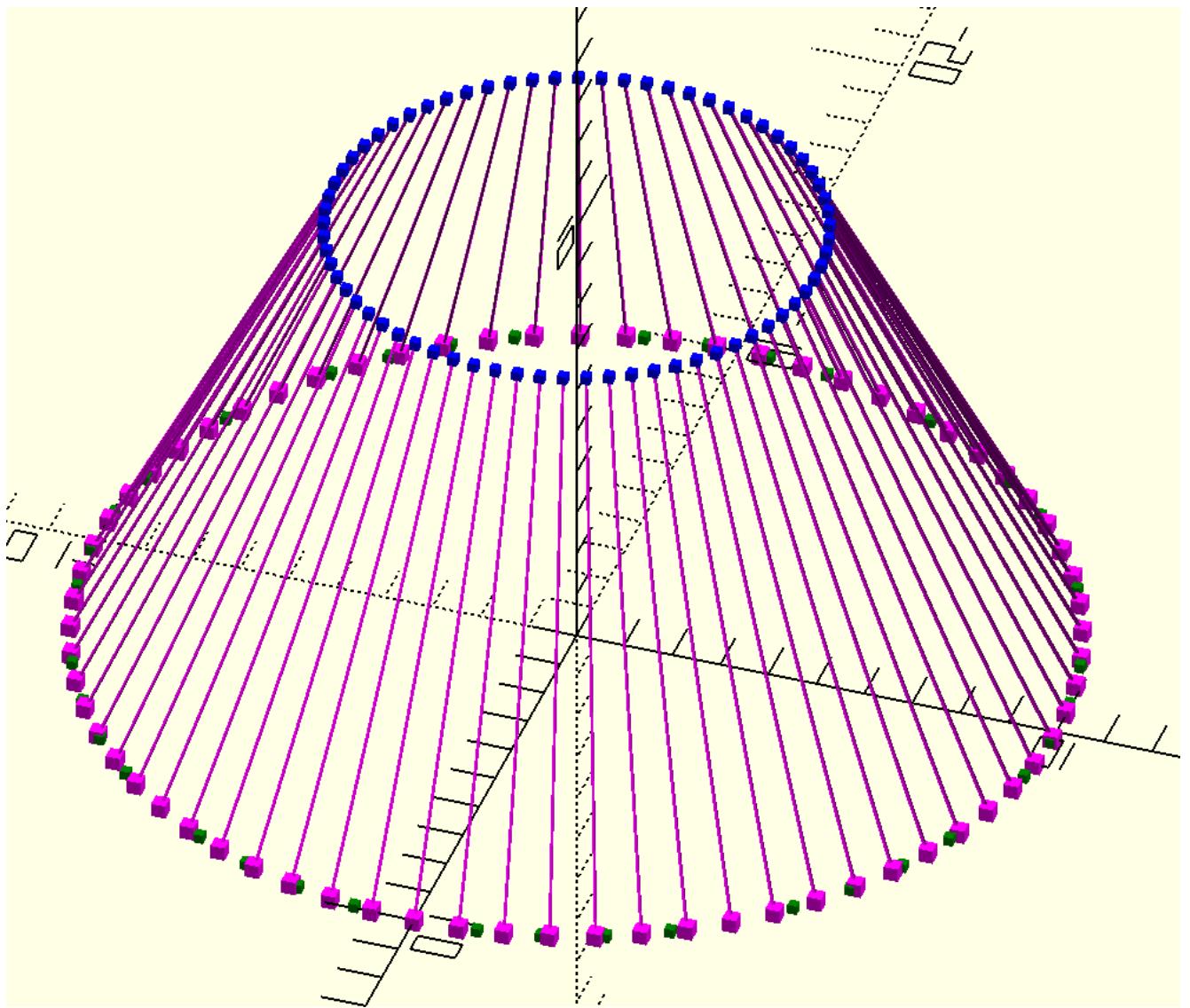
c3=path2path1_closed(c2,c1)
sol=cpo(align_sol_1([c3,c2]))

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>

color("blue")points({{c2}},.2);
color("cyan")points({{c3}},.2);
color("green")points({{c1}},.2);
color("magenta")points({{circle(10,s=70)}},.3);

color("magenta")for(p={{sol}})p_line3d(p,.05,rec=1);

'''')
```



sol2path

In [21]:

```
# example to extrude a solid along a path

# profile of the fan blade
t0=time.time()

p0=corner_radius(pts1([[0,0,.5],[20,-2,300],[20,-5,5],[5,5,4],[-5,5,5],[-20,0,300],[-20,-2,.5]]),20)

# blade profile smoothed with bezier
# sec=bezier(p0,300)
sec=equidistant_pathc(p0,300)

sol=linear_extrude(sec,45)
sol1=translate([45,0,0],q_rot(['y-90'],sol))
sol1=c2ro(sol1,1)
# zval=[p[0][2] for p in sol1]
# sol1=c3t2(sol1)

# generating helical path to extrude the profile
arc1=c2t3(arc(30,-10,0,s=2))
path=arc1+helix(30,120,.25,5)[3:]
path=bezier(path,100)

# code to extrude the solid along the given path
sol2=sol2path(sol1,path)
sol2=slice_sol(sol2,5)

# file path needs to be modified
with open('trial.scad','w+') as f:
    f.write(f'''  

        include<dependencies2.scad>  

difference(){{
```

```

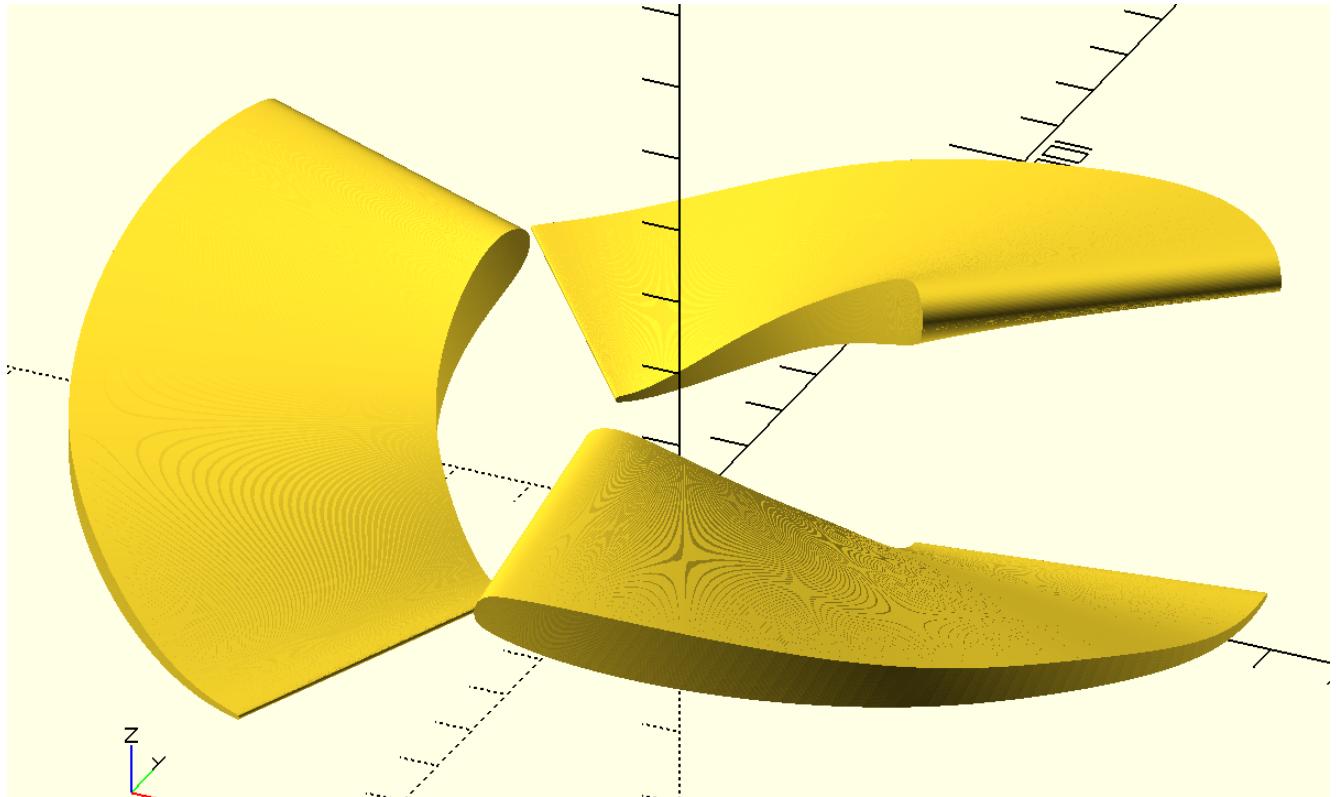
union(){{
for(i=[0:360/3:359])
rotate([0,0,i])
mirror([0,1,0]){{swp(sol2)}}
}}


//{swp(cut_plane([0,0,1],[1000,1000],100,20))}

color("blue")p_line3d({path},1,1);
'''')
t1=time.time()
t1-t0

```

Out[21]: 0.21719002723693848



In [22]:

```

# profile of the fan blade
t0=time.time()

p0=corner_radius(pts1([[0,0,.5],[20,-2,300],[20,-5,5],[5,5,4],[-5,5,5],[-20,0,300],[-20,-2,.5]]),20)

# blade profile smoothed with bezier
# sec=bezier(p0,200)
sec=equidistant_pathc(p0,300)

sol=linear_extrude(sec,45)
sol1=translate([45,0,0],q_rot(['y-90'],sol))
sol1=c2ro(sol1,1)
zval=[p[0][2] for p in sol1]
solx=c3t2(sol1)

sol2=[]
for p in solx:
    r1=array([l_len(p1) for p1 in seg(p)]).min()/5
    s1=corner_radius([[p[0][0],p[0][1],r1],[p[1][0],p[1][1],r1],[p[2][0],p[2][1],r1],[p[3][0],p[3][1],r1]])
    sol2.append(s1)

sol2=[translate([0,0,zval[i]],sol2[i]) for i in range(len(sol2))]

# generating helical path to extrude the profile
arc1=c2t3(arc(30,-10,0,s=3)[-1])
path=arc1+helix(30,120,.25,5)
path=bezier(pa2pb(path,zval),len(zval))

# extrude a solid to a different path

```

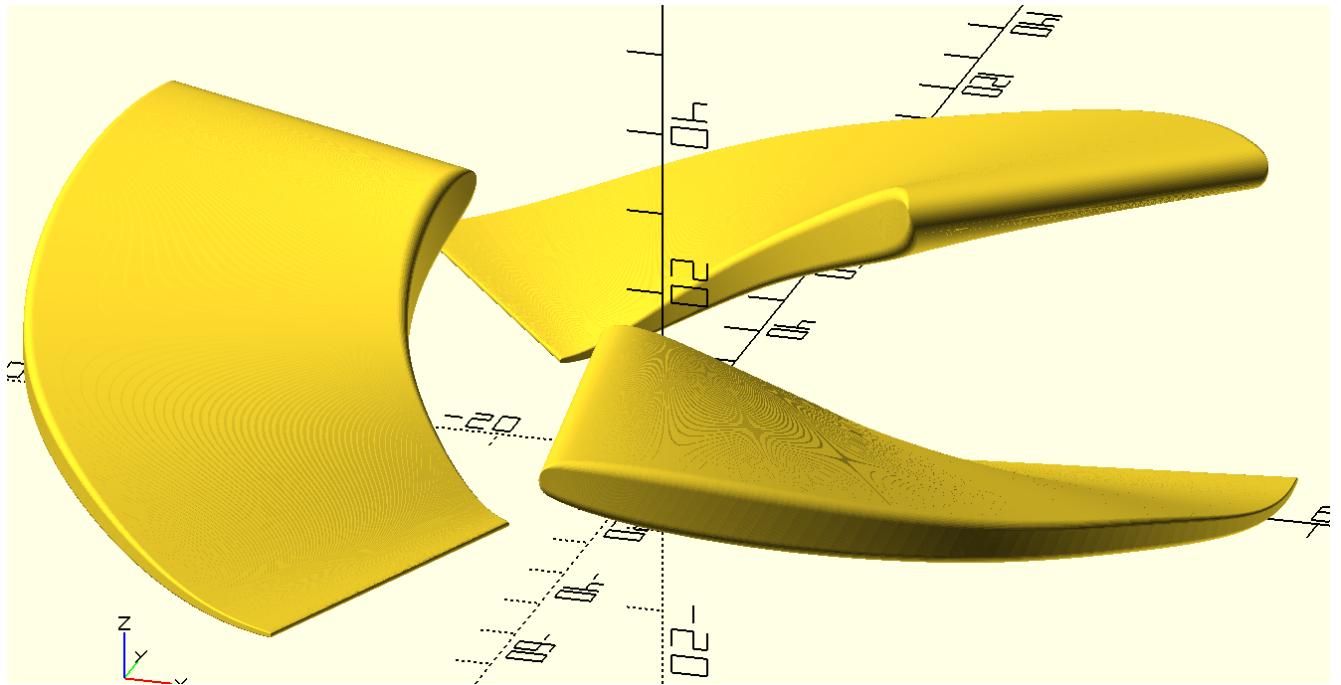
```

sol3=sol2path(sol2,path)
sol3=slice_sol(sol3,5)

with open('trial.scad','w+') as f:
    f.write(f''''
difference(){
union(){
for(i=[0:360/3:359])
rotate([0,0,i])
mirror([0,1,0]){\swp(sol3)}
}
//{\swp(cut_plane([0,0,1],[1000,1000],100,20))}
}
''')
t1=time.time()
t1-t0

```

Out[22]: 1.3731441497802734



```

In [23]: t0=time.time()

sec=circle(5)
path1=corner_radius(pts1([[-3,0],[3,0,3],[0,15,2],[-2,5,3],[2,5,2],[0,20,3],[-3,0]]),10)
path1=equidistant_path(path1,200)
sol=prism(sec,path1)
path=bspline_cubic(cr_3d([[10,0,0,0],[10,15,10,8],[0,0,15,6],[20,0,0,0]]),20)
sol2=sol2path(sol,path)

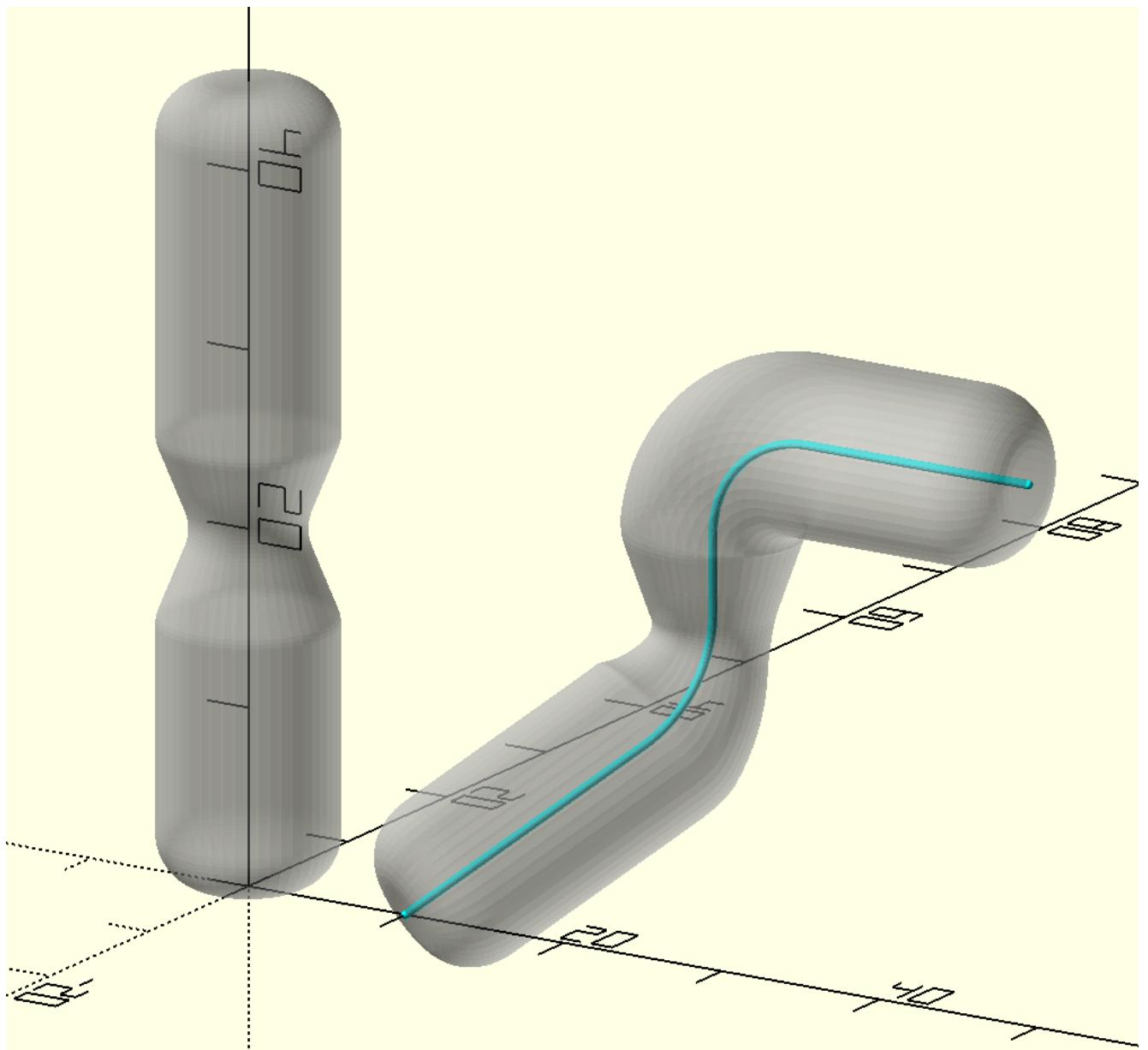
with open('trial.scad','w+') as f:
    f.write(f''''
include<dependencies2.scad>

%{swp(sol)}
%{swp(sol2)}
color("cyan")p_line3d({path},.5);
''')

t1=time.time()
t1-t0

```

Out[23]: 1.5306391716003418



```
In [25]: sec=circle(5)
path=bezier(pts=[[5,0],[15,10],[-26,10],[0,10],[5,6],[5,15],[4,4]]),50)
path1=q_rot(['x90'],corner_radius(pts1([[0,0],[0,45,15],[30,0]]),10))
sol=prism(sec,path)
sol1=sol2path(sol,path1)

with open('trial.scad','w+')as f:
    f.write(f''''
include<dependencies2.scad>

color("blue")p_line3d({path1},1);
{swp(sol1)}
'''')
```

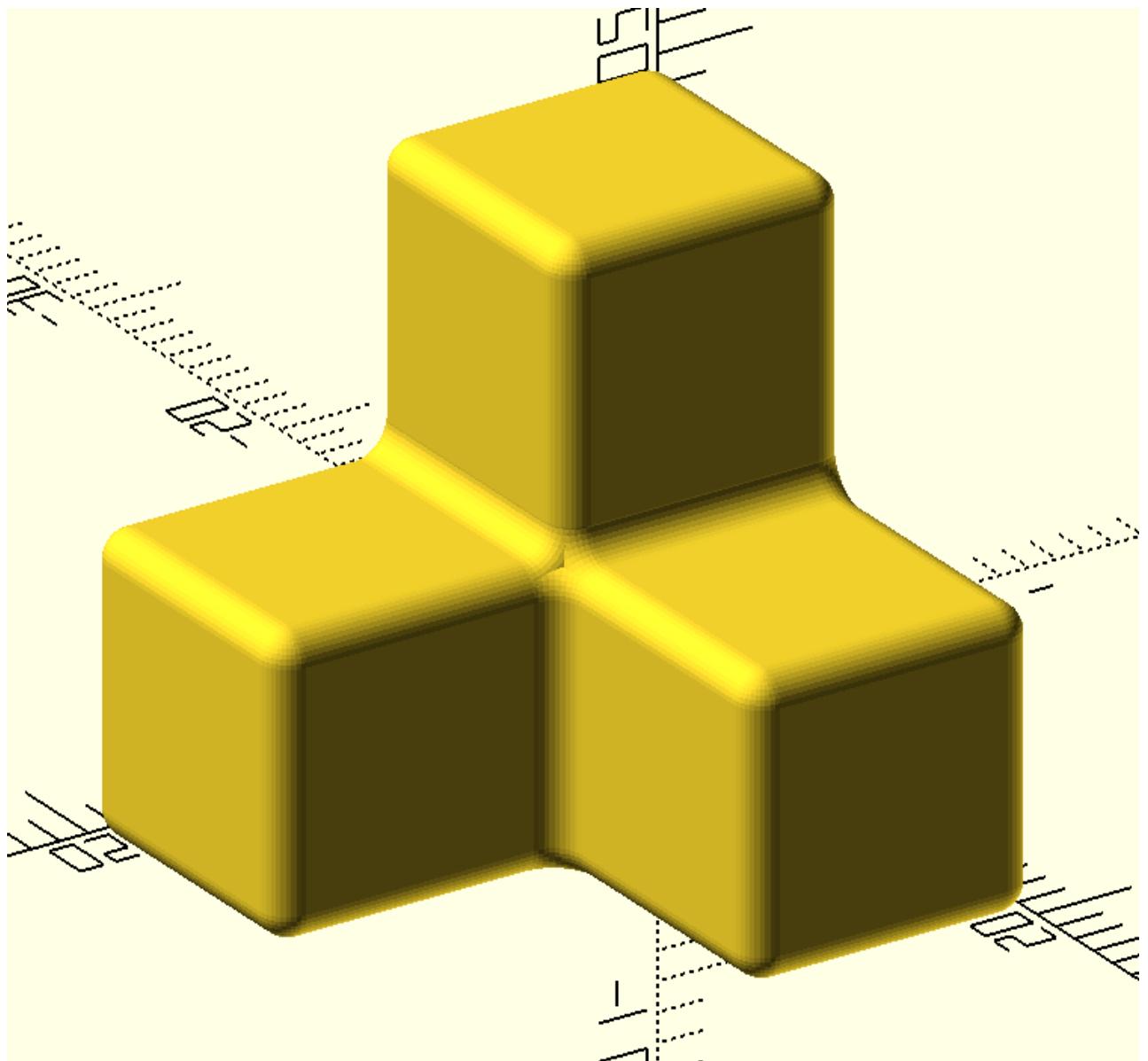
rounding-various-rounded-cubes

```
In [26]: t0=time.time()
r=1
sec=corner_radius(pts1([[0,0,r],[20,0,r],[0,10,r],[-10,0,r],[0,10,r],[-10,0,r]]),10)
path=corner_radius(pts1([[-r,0],[r,0,r],[0,10,r],[-r,0]]),10)
sol=prism(sec,path)

sec1=corner_radius(pts1([[0,0,r],[10,0,r],[0,10,r],[-10,0,r]]),10)
path1=corner_radius(pts1([[-r,10],[r,0,r],[0,10,r],[-r,0]]),10)
sol1=prism(sec1,path1)
l1=cr_3d([[10-r,0,10-r,0],[r,0,r,r],[0,10,0,r],[-r,0,-r,0]],10)
l2=cr_3d([[10-r,0,10+r,0],[r,0,0,r],[0,10,0,r],[-r,0,0,0]],10)
l3=cr_3d([[10+r,0,10-r,0],[0,0,r,r],[0,10,0,r],[0,0,-r,0]],10)
fillet1=convert_3lines2fillet(l3,l2,l1)

l1=cr_3d([[10,10-r,10-r,0],[0,r,r,r],[-10,0,0,r],[0,-r,-r,0]],10)
```

Out [26]: 0.33608102798461914



ip_triangle

In [27]:

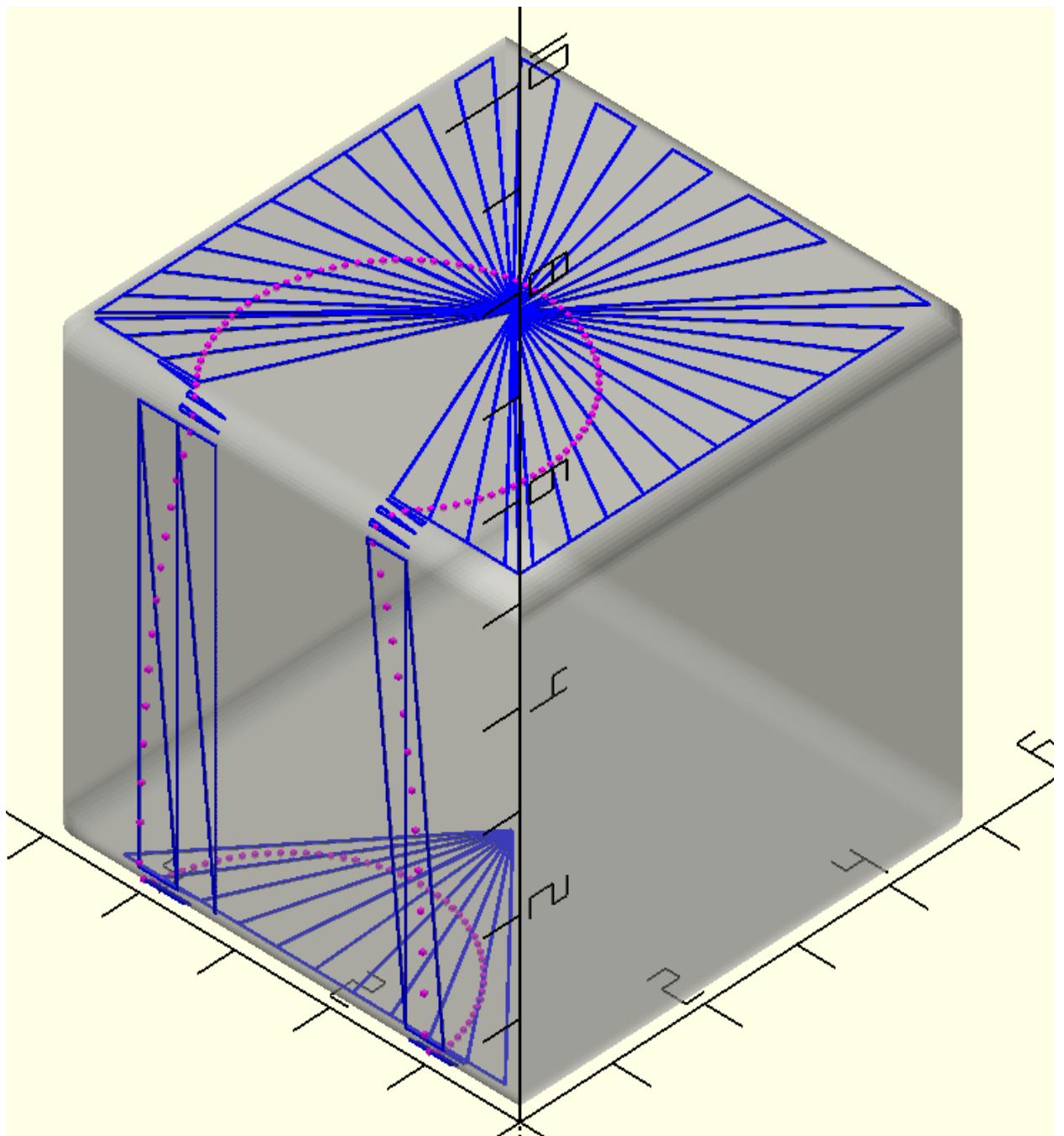
```
t0=time.time()

sec=pts([[0,0],[5,0],[0,5],[-5,0]])
sec=equidistant_pathc(sec,50)
path=corner_radius(pts1([-2.49,0],[2.49,0,0.25],[0,5,0.25],[-2.49,0])),10)
sol1=prism(sec,path)
v1=[2,0,8]
sec1=axis_rot([0,0,1],circle(1.5,s=100),90)
sol2=o_solid(v1,sec1,10,-1.5,-2.5,0)
pa=ip_sol2sol(sol1,sol2)
p1=[p[0] for p in pa]
p2=[p[-1] for p in pa]
p3=flip(p1)+p2

tri_1=ip_triangle(p3,sol1)
with open('trial.scad','w+') as f:
    f.write(f'''  
include<dependencies2.scad>  
  
%{swp(sol1)}  
  
//%{swp(sol2)}  
  
color("magenta")points({p3},.05);
color("blue")for(p={tri_1})p_line3dc(p,.04,rec=1);
''')
t1=time.time()
t1-t0
```

```
/Users/sanjeevprabhakar/openscad/openscad1.py:4630: RuntimeWarning: divide by zero encountered in divide
  t=einsum('kl,ijkl->ijk',cross(p01,p02),la[:, :,None]-p0)/(einsum('ijl,kl->ijk',(-lab),cross(p01,p02))+.00000)
/Users/sanjeevprabhakar/openscad/openscad1.py:4630: RuntimeWarning: invalid value encountered in divide
  t=einsum('kl,ijkl->ijk',cross(p01,p02),la[:, :,None]-p0)/(einsum('ijl,kl->ijk',(-lab),cross(p01,p02))+.00000)
/Users/sanjeevprabhakar/openscad/openscad1.py:4631: RuntimeWarning: divide by zero encountered in divide
  u=einsum('ijkl,ijkl->ijk',cross(p02[None,None,:,:],(-lab)[:, :,None,:]),(la[:, :,None,:]-p0[None,None,:,:]))/(einsum('ijl,kl->ijk',(-lab),cross(p01,p02))+.00000)
/Users/sanjeevprabhakar/openscad/openscad1.py:4632: RuntimeWarning: divide by zero encountered in divide
  v=einsum('ijkl,ijkl->ijk',cross((-lab)[:, :,None,:],p01[None,None,:,:]),(la[:, :,None,:]-p0[None,None,:,:]))/(einsum('ijl,kl->ijk',(-lab),cross(p01,p02))+.00000)
/Users/sanjeevprabhakar/openscad/openscad1.py:4632: RuntimeWarning: invalid value encountered in divide
  v=einsum('ijkl,ijkl->ijk',cross((-lab)[:, :,None,:],p01[None,None,:,:]),(la[:, :,None,:]-p0[None,None,:,:]))/(einsum('ijl,kl->ijk',(-lab),cross(p01,p02))+.00000)
/Users/sanjeevprabhakar/openscad/openscad1.py:4633: RuntimeWarning: invalid value encountered in add
  condition=(t>=0)&(t<=1)&(u>=0)&(u<=1)&(v>=0)&(v<=1)&(u+v<1)
/Users/sanjeevprabhakar/openscad/openscad1.py:4635: RuntimeWarning: invalid value encountered in multiply
  a=(la[:,None,:,:]+lab[:,None,:,:]*t[:,None,:,:])
```

Out [27]: 0.2533078193664551



$$p_0 + v_0 t_0 = p_a + v_1 t_1 + v_2 * t_2$$

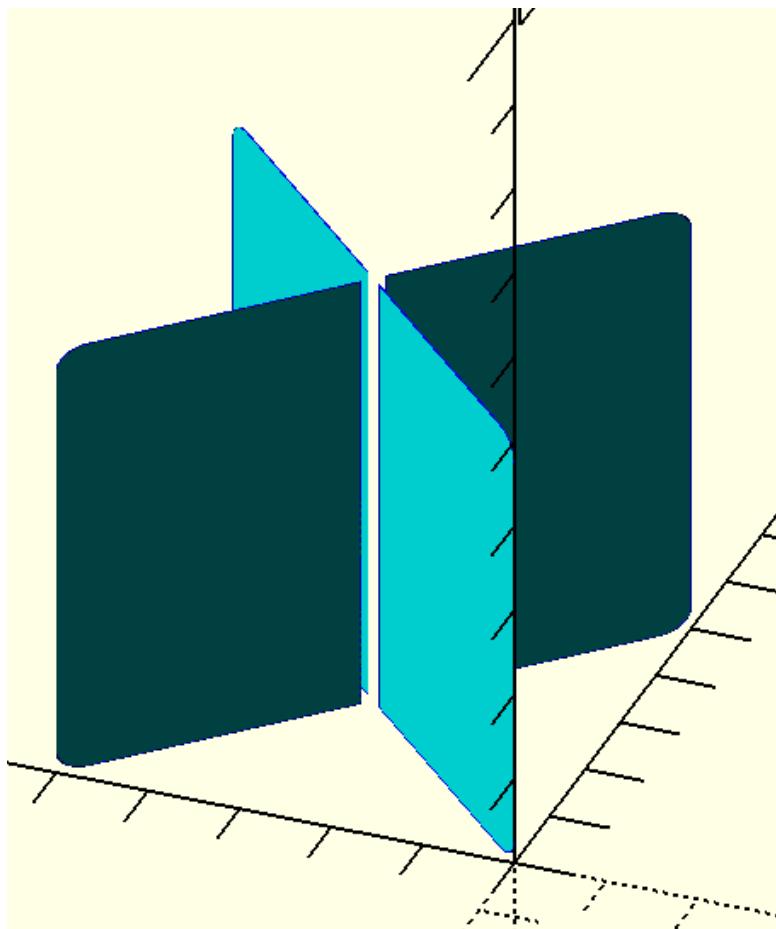
$$v_0 t_0 - v_1 t_1 - v_2 * t_2 = p_a - p_0$$

$$[v_0, -v_1, -v_2] * [t_0, t_1, t_2] = [p_a - p_0]$$

```
In [28]: sec=pts([[0,0],[5,0],[0,5],[-5,0]])
path=corner_radius(pts1([[-2.4,0],[2.4,0,0.25],[0,5,0.25],[-2.4,0]]),10)
sol1=prism(sec,path)
with open('trial.scad','w+') as f:
    f.write(f''''
include<dependencies2.scad>

color("blue")for(p={cpo(sol1)})p_line3dc(p,.01,rec=1);
color("cyan")for(p={cpo(sol1)})polyhedron(p,[{range(len(cpo(sol1)[0])).tolist()}]);

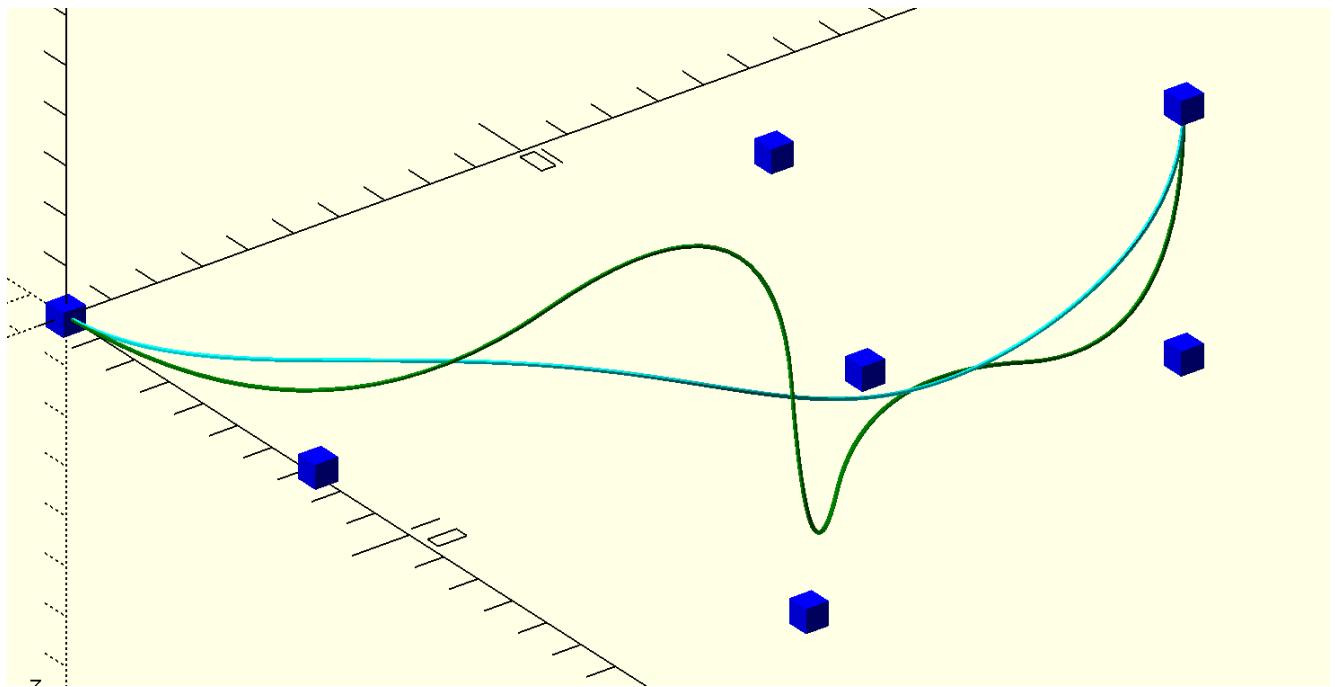
'''')
```



bspline_cubic

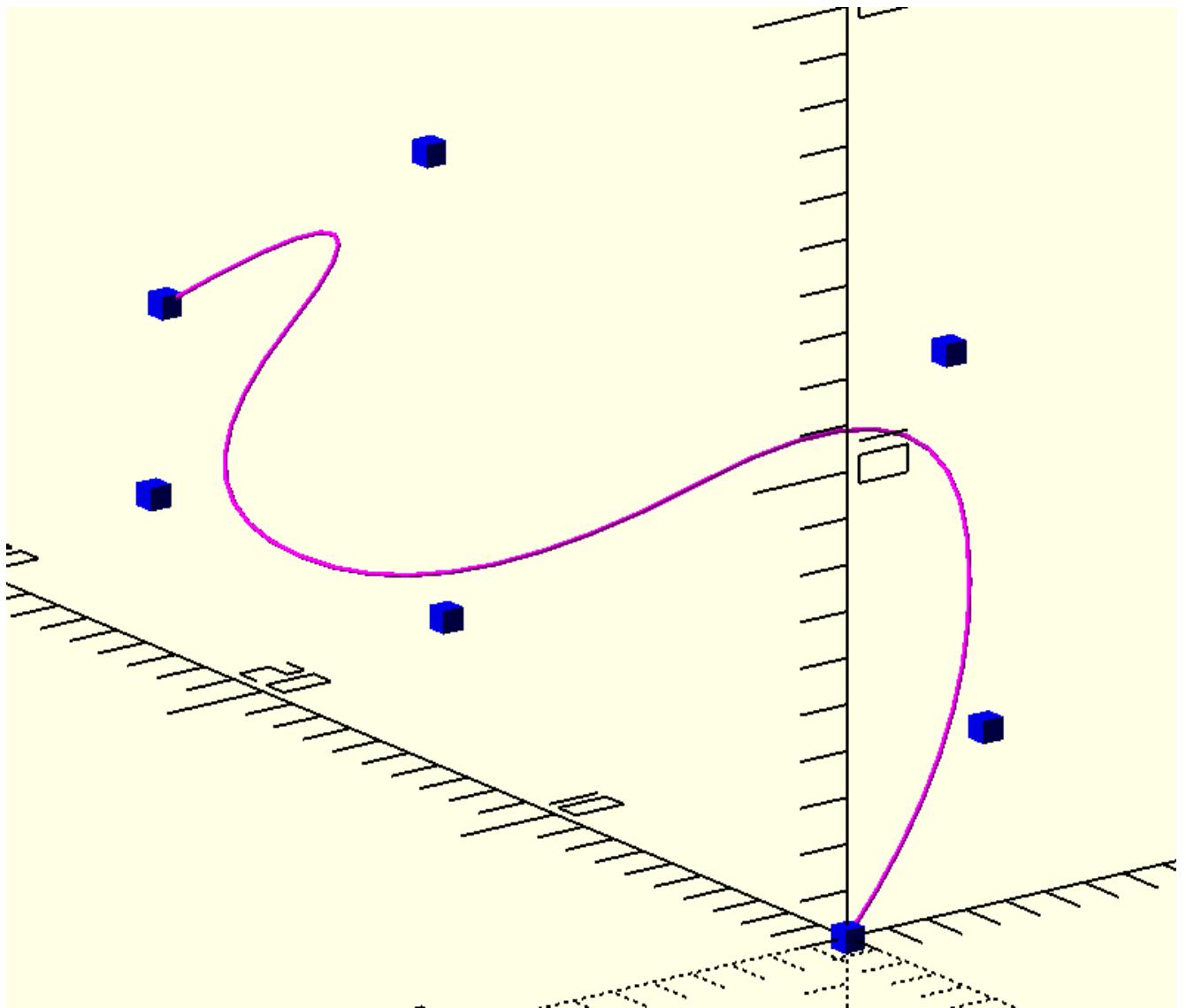
```
In [29]: p0=pts2([[0,0,0],[10,-2,2],[0,10,3],[5,-3,-6],[-5,5,1],[0,7,-2],[0,0,5]])  
  
c1=bezier(p0,100)  
  
s1=bspline_cubic(p0,20)  
  
with open('trial.scad','w+') as f:  
    f.write(f'''  
include<dependencies2.scad>  
  
color("blue")points({p0},.5);  
  
color("cyan")p_line3d({c1},.05,rec=1);  
  
color("green")p_line3d({s1},.05,rec=1);  
  
...)  
  
(len(p0)-2)*20,len(s1)
```

Out[29]: (100, 100)



```
In [30]: px=pts2([[0,0,0],[5,2,3],[5,8,5],[-15,-3,-2],[0,10,0],[7,0,6],[-3,5,-4]])  
  
b=bspline_cubic(px)  
c=bezier(px,100)  
  
with open('trial.scad','w+') as f:  
    f.write(f'''  
        include<dependencies2.scad>  
        color("blue")points({px},.5);  
        color("magenta")p_line3d({b},.1);  
        color("cyan")p_line3d({c},.1);  
  
    ''')  
  
len(b)
```

Out[30]: 50



```
In [31]: px=pts2([[0,0,0],[5,2,3],[5,8,5],[-15,-3,-2],[0,10,0],[7,0,6],[-3,5,-4]])
px=helix(10,5,1,5)
b=bspline_cubic(px)
c=bezier(px,100)

with open('trial.scad','w+') as f:
    f.write(f'''
        include<dependencies2.scad>
color("blue")points({{px}},.5);
color("magenta")p_line3d({{b}},{.1},rec=1);
color("cyan")p_line3d({{c}},{.1},rec=1);

    ''')
len(b)
```

Out[31]: 700

faces

```
In [32]: sec=pts([[0,0],[5,0],[0,5],[-5,0]])
# sec=equidistant_pathc(sec,50)
path=corner_radius(pts1([[-1,0],[1,0,0.25],[0,5,0.25],[-1,0]]),10)
sol1=prism(sec,path)
l,m,_=array(sol1).shape
v=array(sol1).reshape(-1,3).tolist()
f1=faces(l,m)

with open('trial.scad','w+') as f:
    f.write(f'''
        include<dependencies2.scad>

polyhedron({v},{f1});
```

...)

faces_1

```
In [33]: sec=pts([[0,0],[5,0],[0,5],[-5,0]])
# sec=equidistant_pathc(sec,50)
path=corner_radius(pts1([[-1,0],[1,0,0.25],[0,5,0.25],[-1,0]]),10)
sol1=prism(sec,path)
l,m,_=array(sol1).shape
v=array(sol1).reshape(-1,3).tolist()
f1=faces_1(l,m)

with open('trial.scad','w+') as f:
    f.write(f'''  
include<dependencies2.scad>  
  
polyhedron({v},{f1});  
'''')
```

```
In [34]: p0=[10,0,0]
p1=[5,2,3]
p2=[10,1,1]

l1,p2=array([p0,p1]),array(p2)

v1=l1[1]-l1[0]
u1=v1/norm(v1)
v2=p2-l1[0]

v2sint=norm(cross(v1,v2))/norm(v1)
v2cost=(v1@v2)/norm(v1)
t1=v2cost/norm(v1)
p3=l1[0]+v1*t1
p2=p2.tolist()
p3=p3.tolist()

with open('trial.scad','w+') as f:
    f.write(f'''  
include<dependencies2.scad>  
  
color("blue")points({[p0,p1,p2,p3]},.3);
color("cyan")p_line3d({[p0,p1]},.1);
color("magenta")p_line3d({[p2,p3]},.1);  
  
'''')
t1
```

Out [34]: 0.13157894736842107

p2p_intersection_line

l_sec_ip_3d

|2|_intersection

```
In [35]: i_t=time.time()
p0=[[1,2,1],[5,4,3],[2,8,5]]

p1=[[10,2,1],[5,1,3],[12,8,-5]]
line=p2p_intersection_line(p0,p1)
line1=l_sec_ip_3d(p0,line)
line2=l_sec_ip_3d(p1,line)
```

```

with open('trial.scad', 'w+') as f:
    f.write(f'''
        include<dependencies2.scad>

color("blue")p_line3dc({p0},.1,rec=1);
color("magenta")p_line3dc({p1},.1,rec=1);
color("cyan")p_line3d({line1},.1,rec=1);
color("green")p_line3d({line2},.1,rec=1);

'''')
f_t=time.time()
f_t-i_t

```

Out[35]: 0.0683140754699707

In [36]: i_t=time.time()

```

# p0=translate([-2,0,-8.22],[[2,4,5],[7,9,15],[1,10,5]])
# p1=translate([-3,0,-5],[[5,10,3],[5,-7,5],[-10,5,10]])

p0=[[-2,5,0],[-5,0,0],[0,5,10]]
p1=[[0,-5,0],[0,10,0],[5,3,15]]

line=p2p_intersection_line(p0,p1)
line1=l_sec_ip_3d(p0,line)
line2=l_sec_ip_3d(p1,line)

line3=[mean(p0,0).tolist(),add(mean(p0,0),multiply(nv(p0),10)).tolist()]
line4=[mean(p1,0).tolist(),add(mean(p1,0),multiply(nv(p1),10)).tolist()]
line5=array([array(p0).mean(0)-cross(nv(p0),nv(p1)),array(p0).mean(0)+cross(nv(p0),nv(p1))]).tolist()
line6=array([array(p1).mean(0)-cross(nv(p0),nv(p1)),array(p1).mean(0)+cross(nv(p0),nv(p1))]).tolist()
line7=array([array(p0).mean(0),array(p0).mean(0)+cross(nv(p0),cross(nv(p0),nv(p1)))*15]).tolist()
line8=array([array(sl_int1(p1,line3)[0]),array(sl_int1(p1,line3)[0])-cross(nv(p1),cross(nv(p0),nv(p1)))])
p2=sl_int1(p1,line3)[0]
p3=l2l_intersection(line7,line8)
p4=array(p0).mean(0).tolist()
p5=sl_int1(p0,line4)[0]

with open('trial.scad', 'w+') as f:
    f.write(f'''
        include<dependencies2.scad>

color("blue")p_line3dc({p0},.1,rec=1);
color("magenta")p_line3dc({p1},.1,rec=1);
//color("cyan")p_line3d({line},.1,rec=1);
//color("cyan")p_line3d({line1},.1,rec=1);
color("green")p_line3d({line2},.1,rec=1);
color("green")p_line3d({line3},.1,rec=1);
color("magenta")p_line3d({line4},.1,rec=1);
color("magenta")p_line3d({line5},.1,rec=1);
color([.2,.3,.7,.2])p_line3d({line6},.1,rec=1);
color([.1,.3,.9,1])p_line3d({line7},.1,rec=1);
color([.1,.3,.2,1])p_line3d({line8},.1,rec=1);
color([.3,.2,.6,1])points({[p2]},.3);
color([.3,.2,.1,1])points({[p3]},.3);
color([.1,.2,.9,1])points({[p4]},.3);
color([.5,.5,1,1])points({[p5]},.3);

polyhedron({p0},[[0,1,2]]);
polyhedron({p1},[[0,1,2]]);

'''')
f_t=time.time()
f_t-i_t

```

Out[36]: 0.14110994338989258

i_p_p

i_p_n

i_p_t

o_3d

ip_fillet

In [37]:

```
i_t=time.time()
sol=o_solid([1,0,0],circle(5),30,-15)
sol1=o_solid([0,1,0],circle(4.8),30,-15)
sol1=slice_sol(sol1,2)
fillet1=ip_fillet(sol,sol1,2,-2,o=-1)
fillet2=ip_fillet(flip(sol),flip(sol1),2,-2,o=-1)

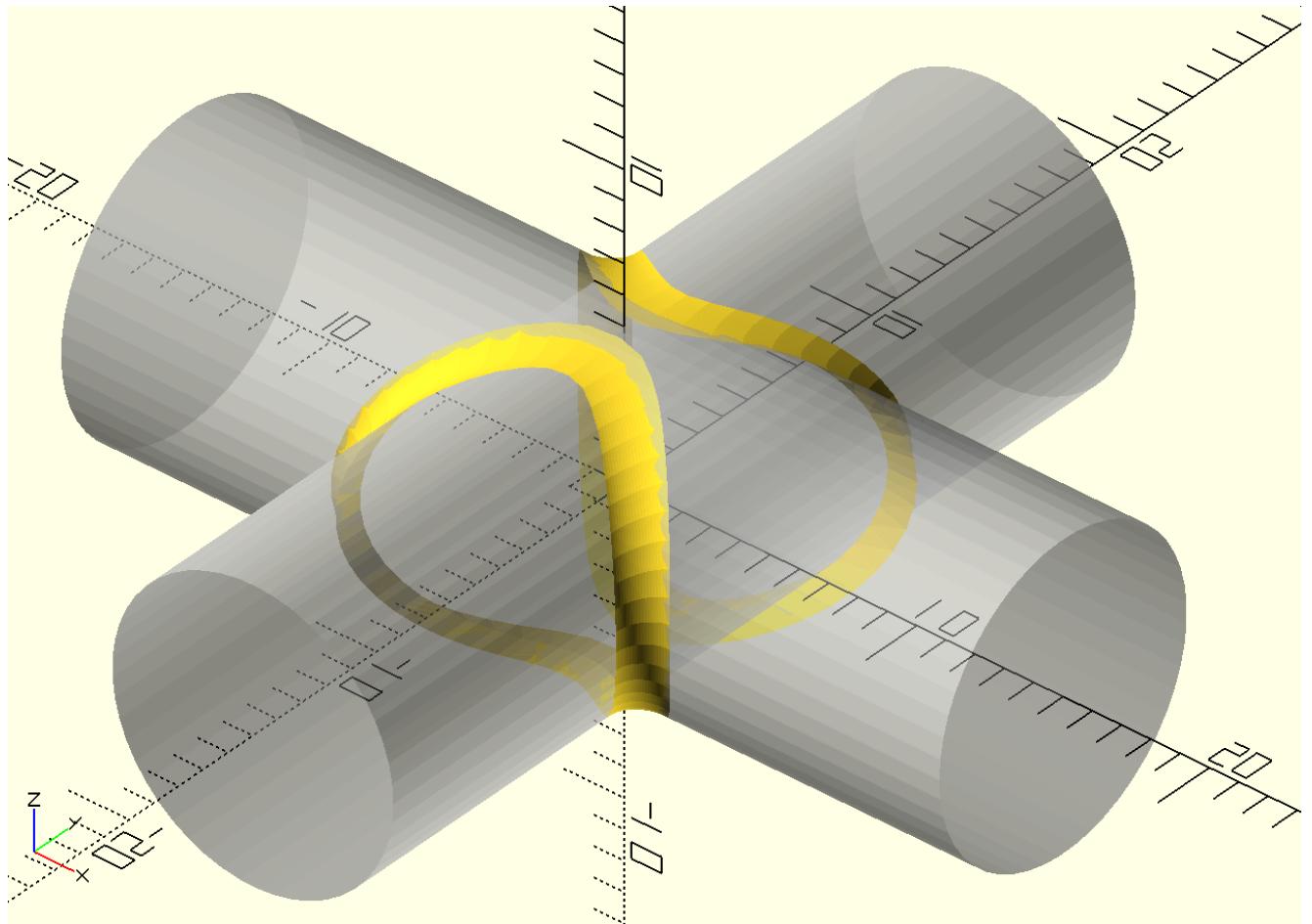
with open('trial.scad','w+') as f:
    f.write(f'''
        include<dependencies2.scad>

        %swp(sol)
        %swp(sol1)

        {swp_c(fillet1)}
        {swp_c(fillet2)}

    ''')
f_t=time.time()
f_t-i_t
# len(i_p),len(i_p1),len(i_p2)
```

Out [37]: 0.18747711181640625



In [38]:

```
i_t=time.time()
sec=corner_radius(pts1([[-5,-10,1],[10,0,1],[0,15,1],[-10,0,1]]),10)
path=corner_radius(pts1([[5,0],[5,0,1],[0,20,1],[-5,0]]),10)
sol=prism(sec,path)
sec1=circle(2,s=100)
```

```

path1=bezier(pts2([[0,-4,2],[0,5,10],[10,1,20]]),20)
sol1=path_extrude_open(sec1,path1)

p1=ip_sol2sol(sol,sol1)
p1=[p[-1] for p in p1]
p2=i_p_p(sol1,p1,1)
p3=o_3d(p1,sol,-1)

fillet1=convert_3lines2fillet_closed(p3,p2,p1)

with open('trial.scad','w+')as f:
    f.write(f'''  

    include<dependencies2.scad>  
  

    %{swp(sol)}  

    %{swp(sol1)}  

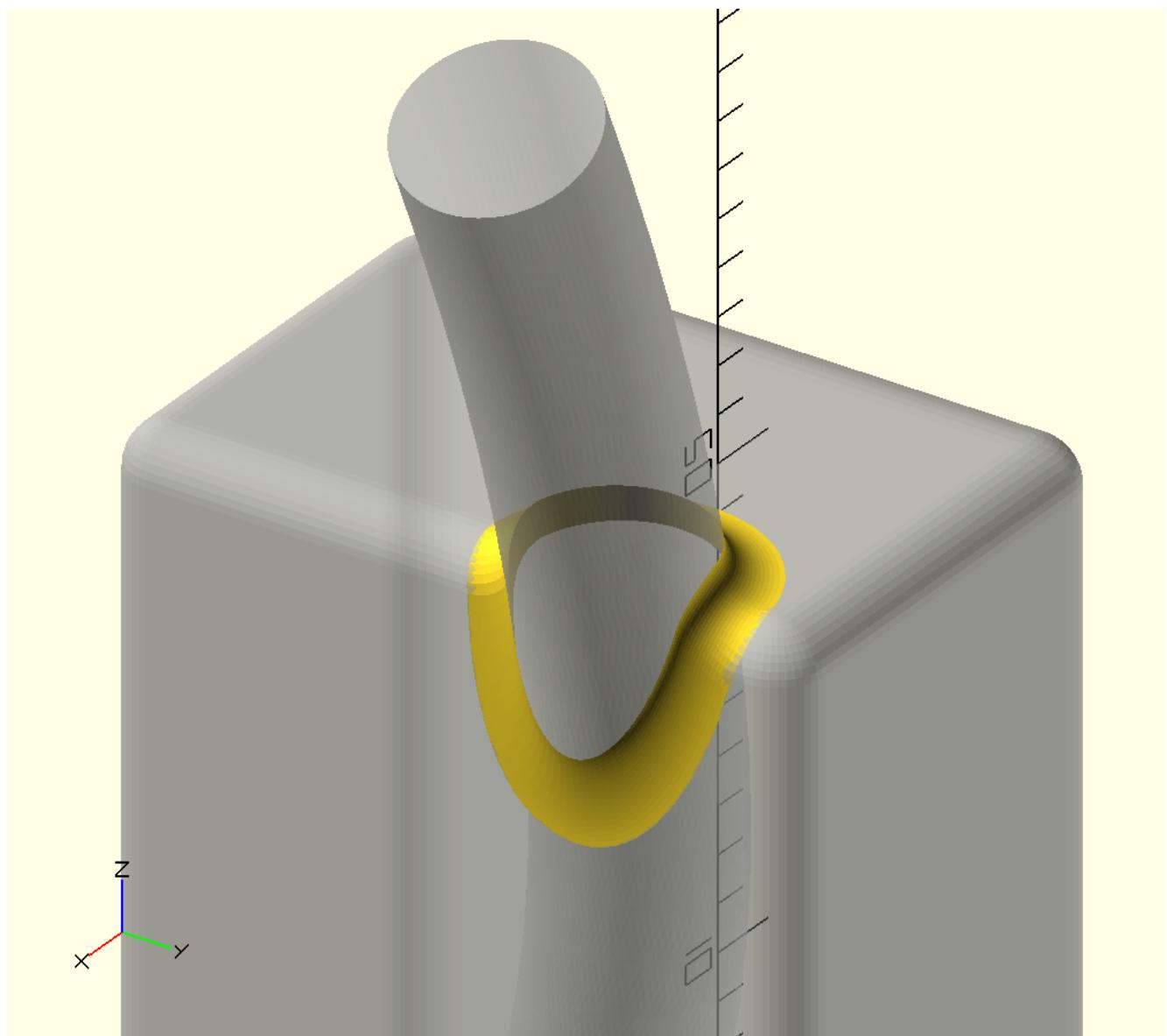
    {swp_c(fillet1)}  
  

    ''')
f_t=time.time()
f_t-i_t

```

```
/Users/sanjeevprabhakar/openscad/openscad1.py:4631: RuntimeWarning: invalid value encountered in divide
  u=einsum('ijkl,ijkl->ijk',cross(p02[None,None,:,:],(-lab)[:, :, None, :]),(la[:, :, None, :] - p0[None, Non
e, :, :]))/(einsum('ijl,kl->ijk',(-lab),cross(p01,p02))+.00000)
```

Out [38]: 1.031451940536499



```

In [39]: i_t=time.time()
p1=[[0,-31],[21-5,0,.2],[2,10,4],[35,0,10],
     [5,22,5],[-5,22,10],[-35,0,4],[-2,10,.2],[-21+5,0]]

path1=corner_radius(pts1(p1),10)
sec1=circle(5,s=72)
sol1=prism(sec1,path1)

```

```

sec5=corner_radius(pts1([[-20,-7.5,2.45],[5,0,2.45],[0,10,3],[15,2,70],[15,-2,3],[0,-10,2.45],
[5,0,2.45],[1,7.5,5],[-1,7.5,7],[-20,3,90],[-20,-3,7],[-1,-7.5,5]]),10)

sec5=equidistant_pathc(sec5,201)
sec6=scl2d_c(sec5,.6)
sol2=[c2t3(sec5)]+[translate([0,0,120],sec6)]
sol2=translate([0,41,0],q_rot(['x90','z190'],sol2))

# p1=ip_surf(sol1,sol2)
# p2=i_p_p(sol2,p1,6)
# p3=o_3d(p1,sol1,6)
# fillet1=convert_3lines2fillet_closed(p1,p2,p3)
fillet1=ip_fillet(sol1,sol2,6,-6)
with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>

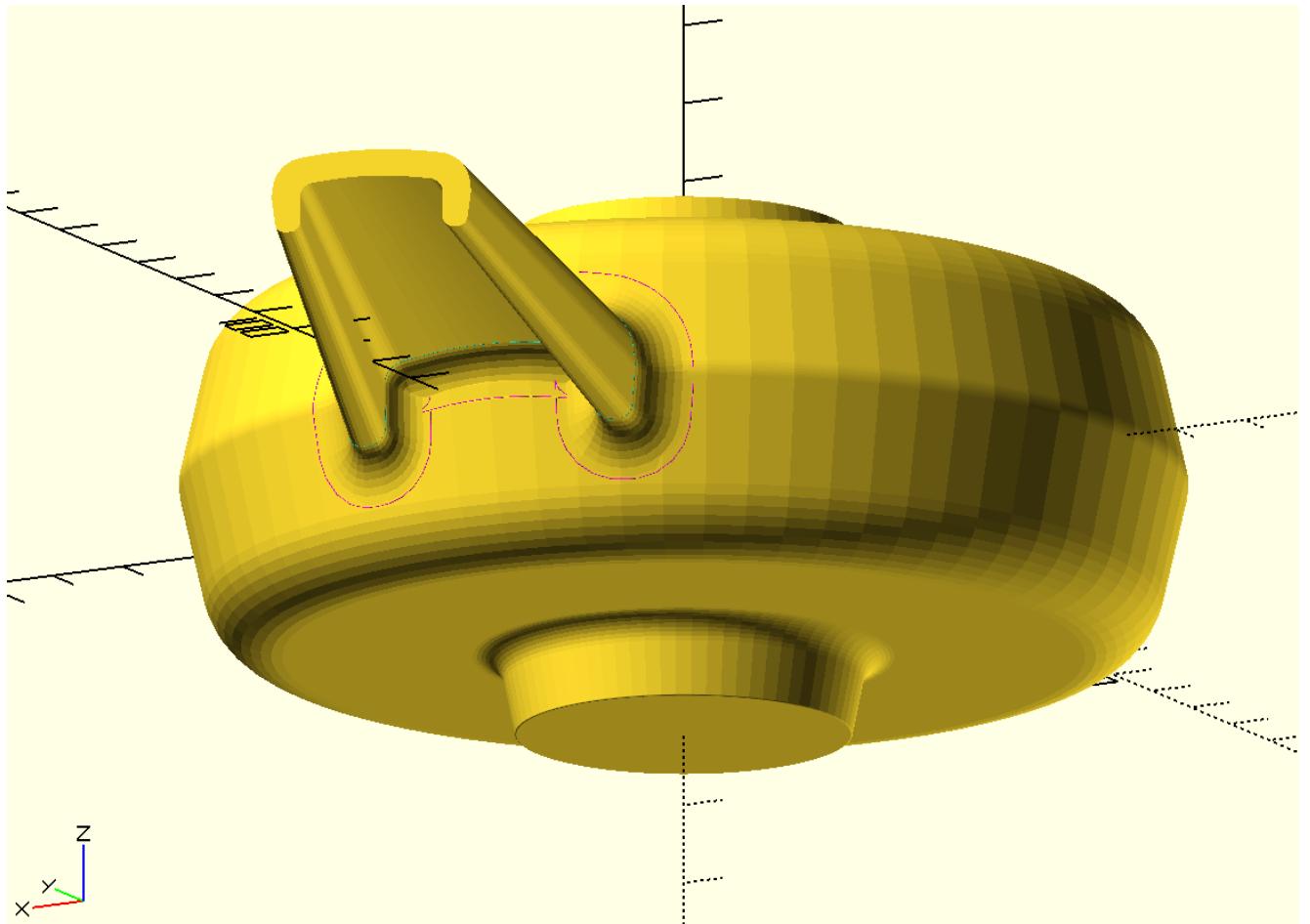
%{swp(sol1)}
%{swp(sol2)}
{swp_c(fillet1)}

'''')

f_t=time.time()
f_t-i_t
# len(p1),len(p2),len(p3)

```

Out[39]: 1.5249741077423096



In [40]: i_t=time.time()

```

p1=[[0,-31],[21-5,0,.2],[2,10,4],[35,0,10],
[5,22,5],[-5,22,10],[-35,0,4],[-2,10,.2],[-21+5,0]]

path1=corner_radius(pts1(p1),10)
sec1=circle(5,s=72)
sol1=prism(sec1,path1)

```

```

sec5=corner_radius(pts1([[-20,-7.5,2.49],[5,0,2.49],[0,10,3],[15,2,70],[15,-2,3],[0,-10,2.49],
[5,0,2.49],[1,7.5,5],[-1,7.5,7],[-20,3,90],[-20,-3,7],[-1,-7.5,5]]),10)

sec5=equidistant_pathc(sec5,200)
sec6=scl2d_c(sec5,.6)
sol2=[c2t3(sec5)]+[translate([0,0,120],sec6)]
sol2=translate([0,41,0],q_rot(['x90','z190'],sol2))
p1=ip(sol1,sol2)

p2=ip(sol1,offset_sol(sol2,8,1))
# p2=sort_points(p1,p2)
# p2=sort_points(p1,p2)
p3=i_p_p(sol2,p1,8)
# p3=sort_points(p1,p3)

fillet1=convert_3lines2fillet_closed(p2,p3,p1)

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>

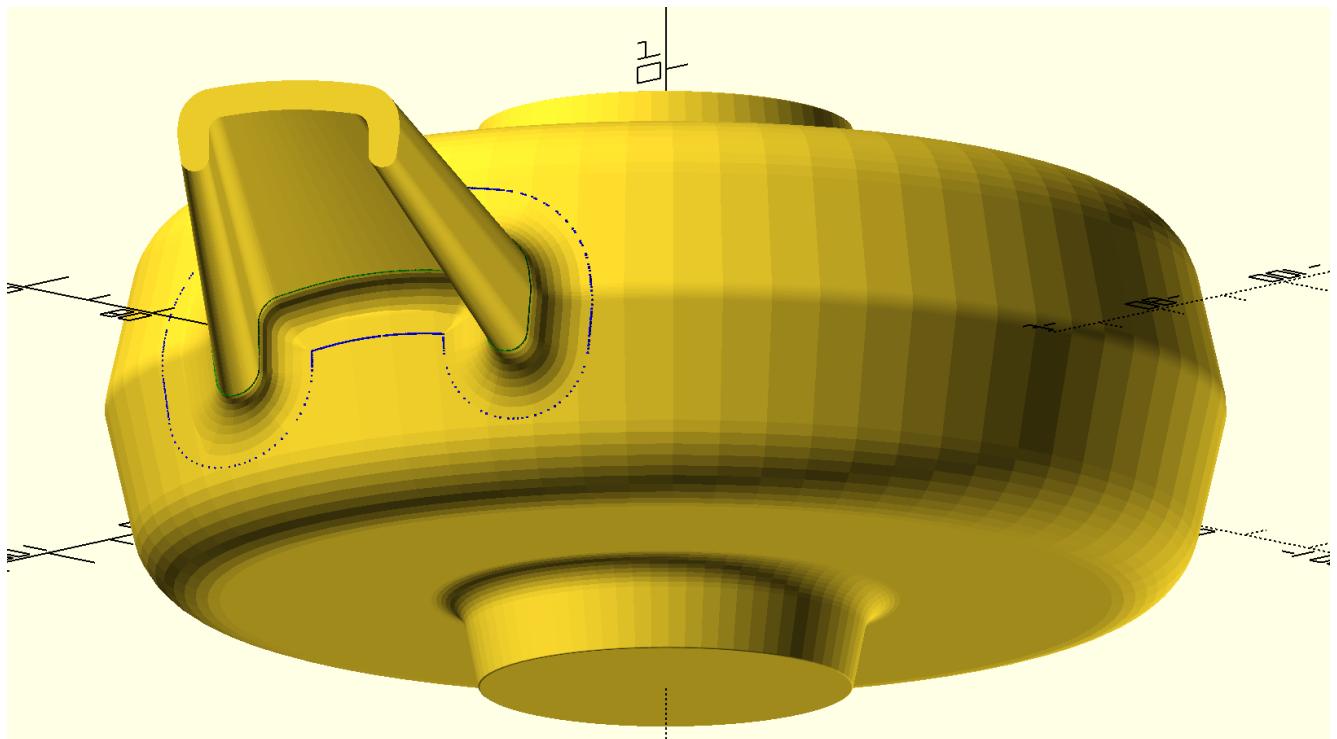
{swp(sol1)}
{swp(sol2)}
//%{swp(offset_sol(sol2,6,1))}
{swp_c(fillet1)}
color("blue")p_line3dc({p2},.2,rec=1);
color("green")p_line3dc({p3},.2,rec=1);

color("magenta")p_line3dc({p1},.2,rec=1);

''')
len(p1),len(p2),len(p3)
f_t=time.time()
f_t-i_t

```

Out[40]: 1.3715388774871826



In [41]:

```

# m10
t0=time.time()

sec1=arc(20,0,359,s=150)
path1=[[0,0],[-5,25]]

surf1=prism(sec1,path1)

sec2=corner_radius(pts1([-25,0],[10,5,5],[10,-3,10],[10,5,5],[10,-8,7],[10,1])),10)
path2=cytz(corner_radius(pts1([-35,5,0],[10,8,20],[20,-5,10],[20,8,20],[10,-9,20],[10,1,0])),10)

```

```

surf2=surf_extrude(sec2,path2)
surf3=surf_extrudef(surf2)
# p1=ip_surf(surf2,surf1)
# p2=o_3d_surf(p1,surf2,2)
# p3=i_p_p(surf1,p1,2)
# # # p=fillet_surf2sol(surf2,surf1,2,10,0)
# fillet1=convert_3lines2fillet_closed(p1,p3,p2)

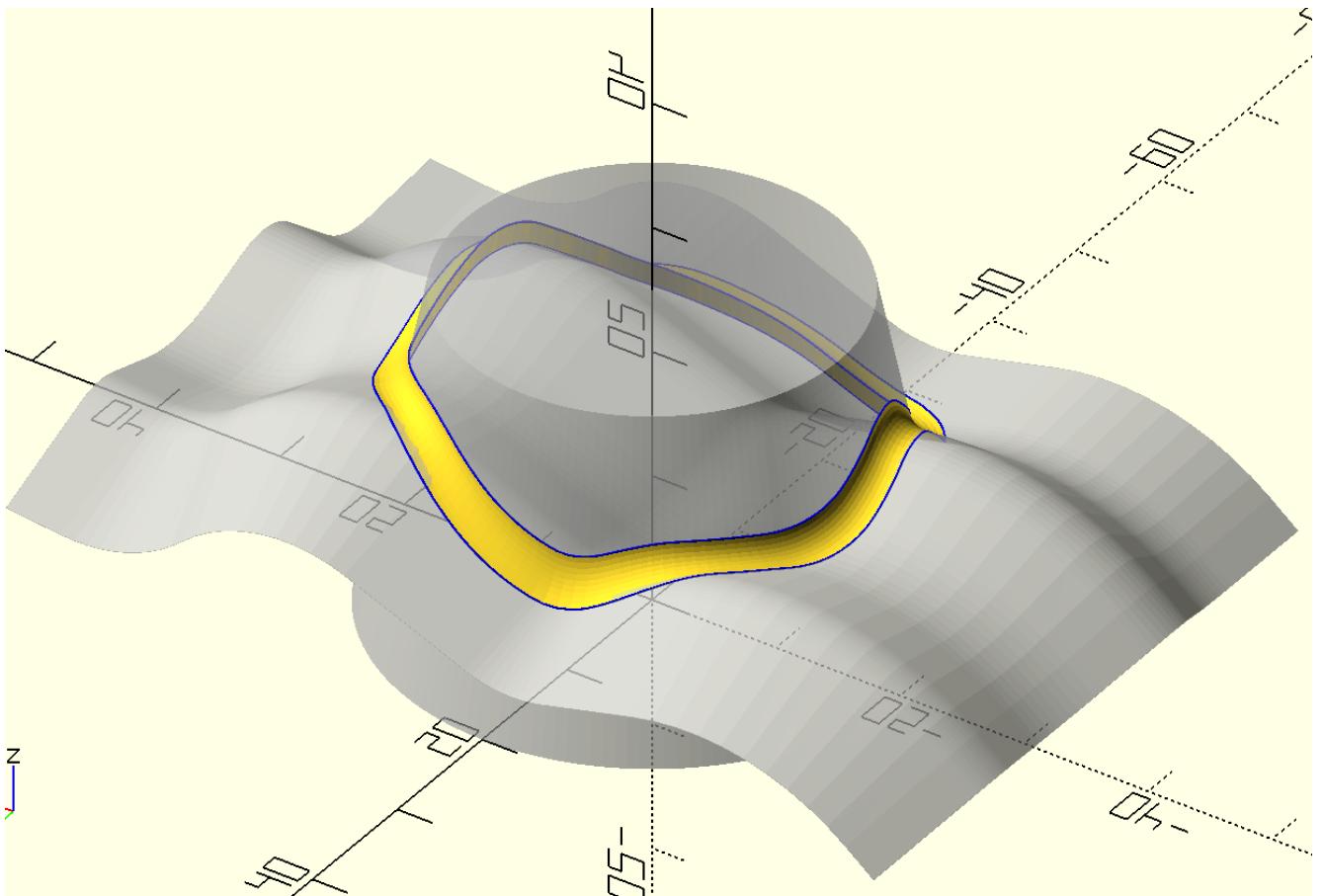
fillet1=ip_fillet_surf(surf2,surf1,2,-2)

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies.scad>
%{swp(surf1)}
%{swp_c(surf3)}
{swp_c(fillet1)}

'''')
t1=time.time()
total=t1-t0
total

```

Out[41]: 0.6061980724334717



```

In [42]: t1=time.time()

p1=[[0,-15,.5],[6,0,.3],[0,-16,.1],[1,-1,.1],[4,0,.2],[2,10,4],[35,0,10],
     [5,22,5],[-5,22,10],[-35,0,4],[-2,10,.2],[-4,0,.1],[-1,-1,.1],[0,-16,.3],
     [-6,0,.5]]
sec1=corner_radius(pts1(p1),10)
path1=c2t3(circle(10,s=72))
sol1=path_extrude_closed(sec1,path1)

sec3=circle(7,s=100)
path2=corner_radius(pts1([[2,0],[-2,31]]),10)

sol3=translate([51,0,12],prism(sec3,path2))
# px=ip_surf(sol1,sol3)
# py=i_p_p(sol3,px,3)
# pz=o_3d_surf(px,sol1,3)
# fillet1=convert_3lines2fillet_closed(px,py,pz)
fillet1=ip_fillet(sol1,sol3,3,-3)

```

```

f3=end_cap(sol3,2)[1]
with open('trial.scad','w+')as f:
    f.write(f'''  

include<dependencies2.scad>  

%{swp_c(sol1)}  

difference(){  

{swp(sol3)}  

{swp_c(f3)}  

}  

{swp_c(fillet1)}  

//color("blue")p_line3dc({sol3[0]},.2);  

''')  

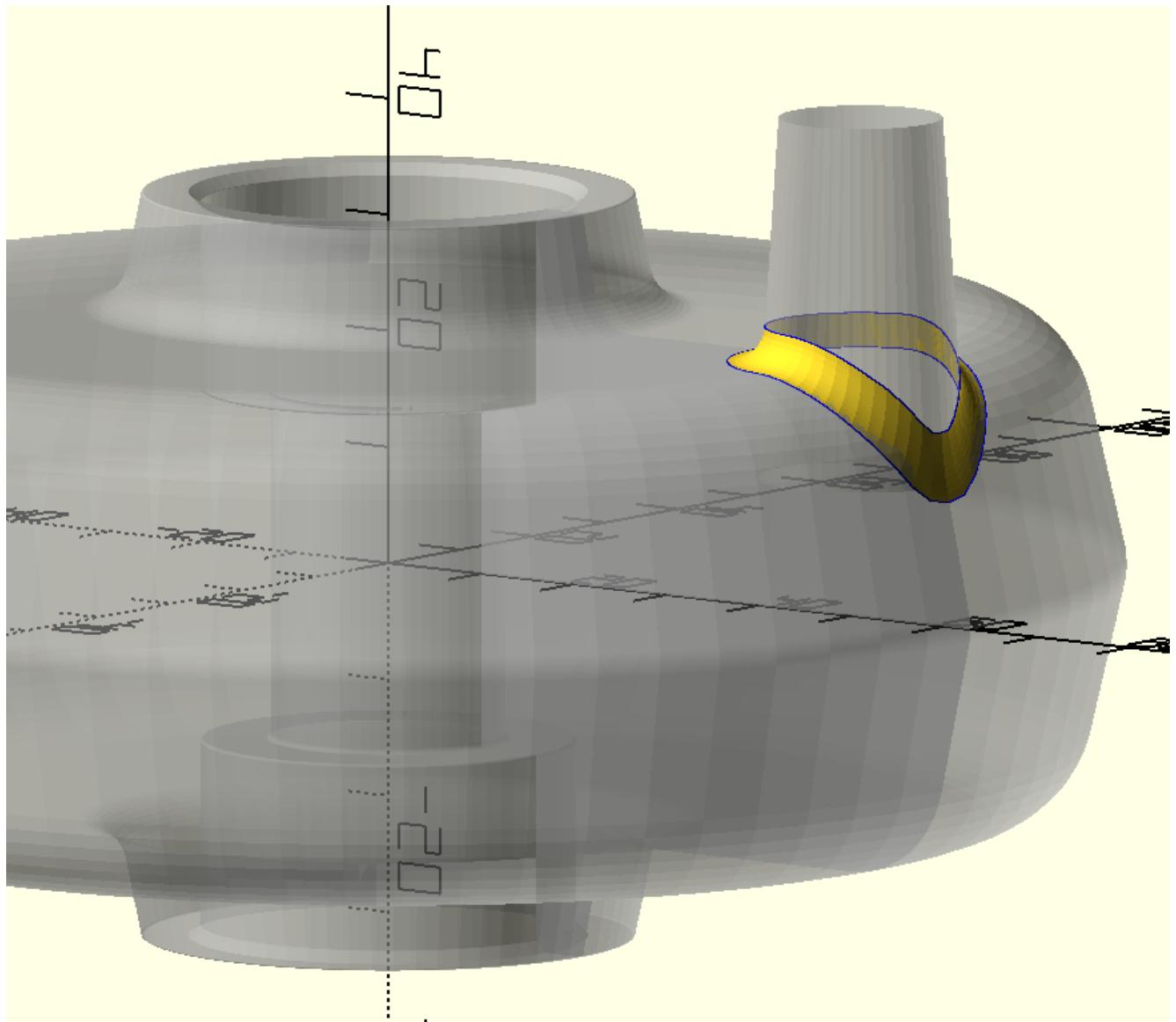
t2=time.time()  

t2-t1  

# len(p1),len(p2),len(p3)

```

Out [42]: 1.56980299949646



In [12]:

```

line=[[0,0],[10,0]]
pnts=[[10,15]]
d=1e-10

perp_points_d(line,pnts,d),perp_distance_within_line(line,pnts)

```

Out[12]: ([], [15.0])

In [43]:

```

t0=time.time()
sec=corner_radius(pts1([[0,0],[40,0],[0,40],[-40,0]]),10)
path=corner_radius(pts1([[-20,0],[20,0],[0,10],[-20,0]]),10)

sol1=prism(sec,path)

```

```

sec1=circle(7.5,s=100)
path1=corner_radius(pts1([[-7,0],[7,0],[-7.49,40]]),10)
sol2=prism(sec1,path1)
sol2=axis_rot_o([1,0,0],translate([-0.01,20,12],q_rot(['y90'],sol2)),180)

a=array([[1-cos(d2r(i)),1-sin(d2r(i))]] for i in linspace(0,90,100)])*5
b=[rsz3dc(sol1,array(bb(sol1))+i*2) for i in a[:,0]]
c=[rsz3dc(sol2,array(bb(sol2))+i*2) for i in a[:,1]]

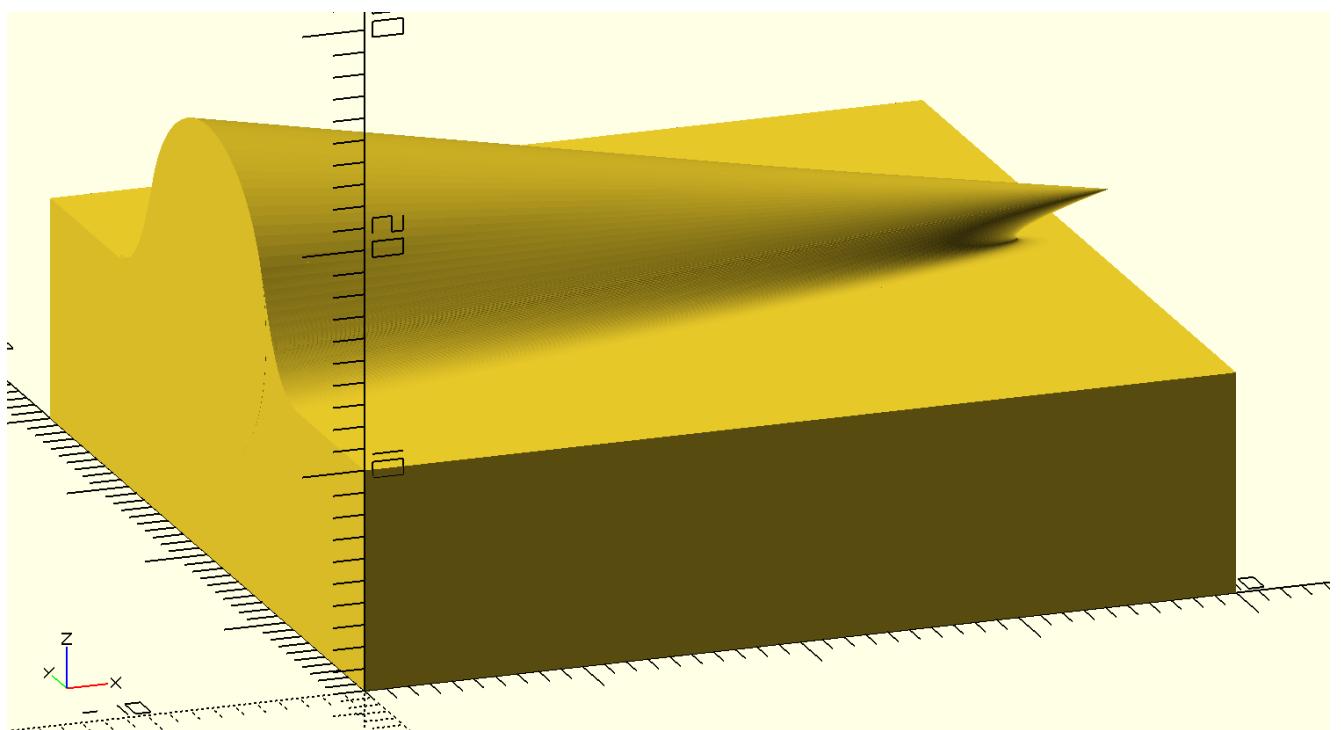
with open('trial.scad','w+') as f:
    f.write(f'''
difference(){
union(){
    ''
}
for i in range(len(a)-1):
    f.write(f'''
//include<dependencies2.scad>
hull(){
intersection(){
{swp(b[i])}
{swp(c[i])}
}

intersection(){
{swp(b[i+1])}
{swp(c[i+1])}
}
}
})
f.write(f'''

    })
translate([-10,-5,-2])cube([10,50,30]);
}
{swp(sol1)}
{swp(sol2)}
''')
t1=time.time()
t1-t0

```

Out[43]: 0.3255009651184082



partial_surface

shield

```
In [44]: i_t=time.time()
sec1=circle(55)
path1=corner_radius(pts1([[-50,0],[50,0,.2],[0,30,3],[6,1,3],[0,6,3],[-4,2,3],[0,22,6],[8,2,6],[0,10,.2]),10)
# path1=equidistant_path(path1,200)
sec2=circle(7.5)
path2=corner_radius(pts1([[-5,0],[5,0,5],[0,35,2],[-5,0]]),10)
path2=equidistant_path(path2,50)
sol1=prism(sec1,path1)
sol2=translate([57.5,0,37],prism(sec2,path2))
sol2=axis_rot_o([0,0,1],sol2,180)

s1=shield(sol1,sol2,50,10,4,135)
v,f1=partial_surface(sol1,prism_center(sol2),50)

with open('trial.scad','w+') as f:
    f.write(f'''
        include<dependencies2.scad>

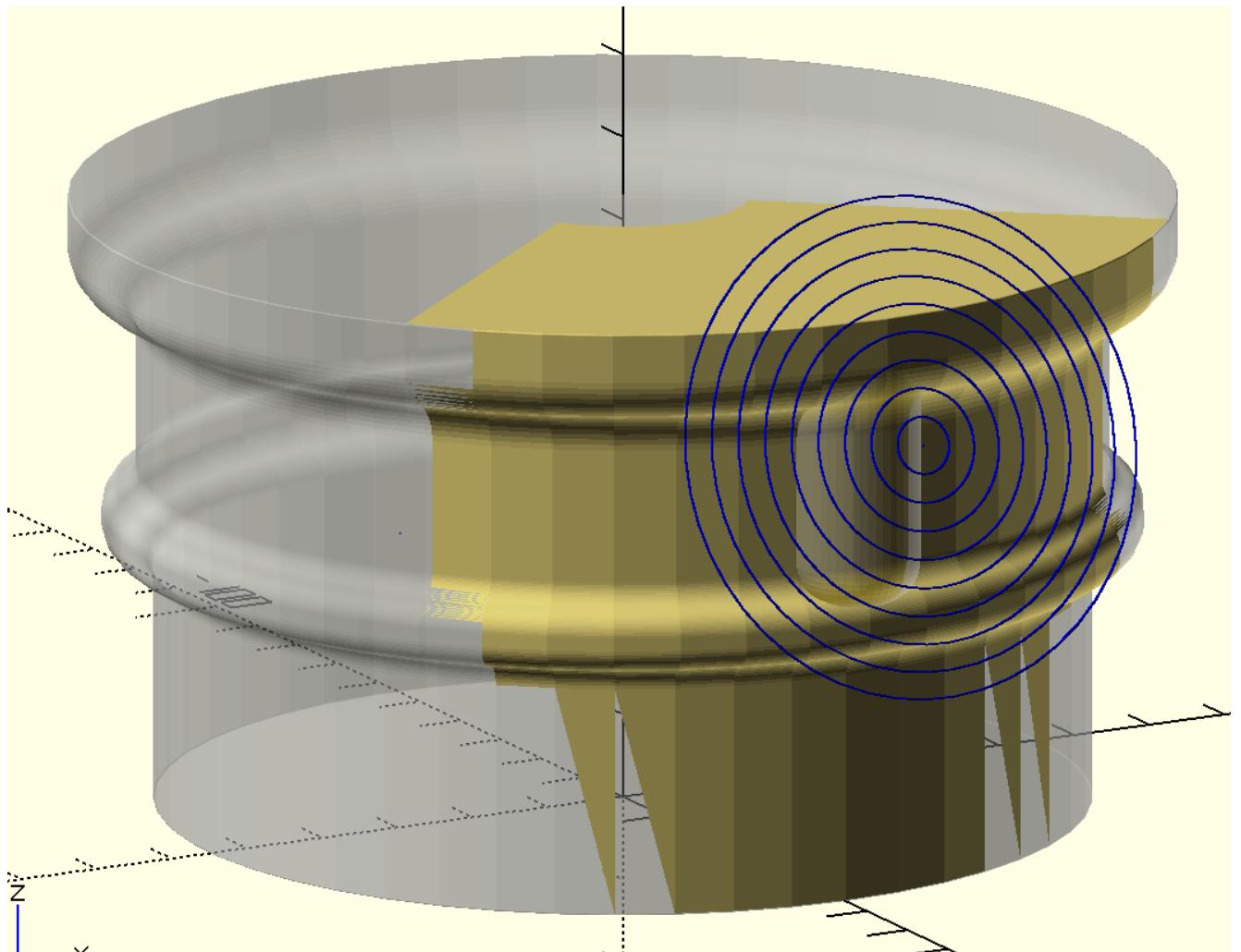
        %{swp(sol1)}
        %{swp(sol2)}

        color("blue")for(p={s1})for(p1=p)p_line3dc(p1,.2,rec=1);
        polyhedron({v},{f1});

    ''')

f_t=time.time()
f_t-i_t
# len(p1),len(p2),len(p3)
```

Out[44]: 0.5128381252288818



```
In [45]: # example of function surface_for_fillet(sol1=[],sol2=[],factor1=50,factor2=10,factor3=1,factor4=100,a  
t0=time.time()
```

```

sec=circle(10)
path=corner_radius(pts1([[-8,0],[10,0],[-2,0,2],[-1,15,3],[-8.9,0]]),10)
path=equidistant_path(path,100)
sol1=q_rot(['z90'],prism(sec,path))

sec1=corner_radius(pts1([[0,0,1],[5,0,1],[0,7,2.3],[-5,0,2.3]]),10)
path1=corner_radius(pts1([[-2.4,0],[2.4,0,2],[0,5,.3],[-.5,0]]),10)
path1=equidistant_path(path1,30)
sol2=translate([6,0,12],q_rot(['x90','z90'],prism(sec1,path1)))

i_p1=shield(sol1,sol2,100,20,4,23)
v,f1=partial_surface(sol1,prism_center(sol2),10)

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>
%{swp(sol1)}
%{swp(sol2)}
color("blue") for(p={i_p1})for(p1=p)p_line3dc(p1,.05,rec=1);
polyhedron({v},{f1});

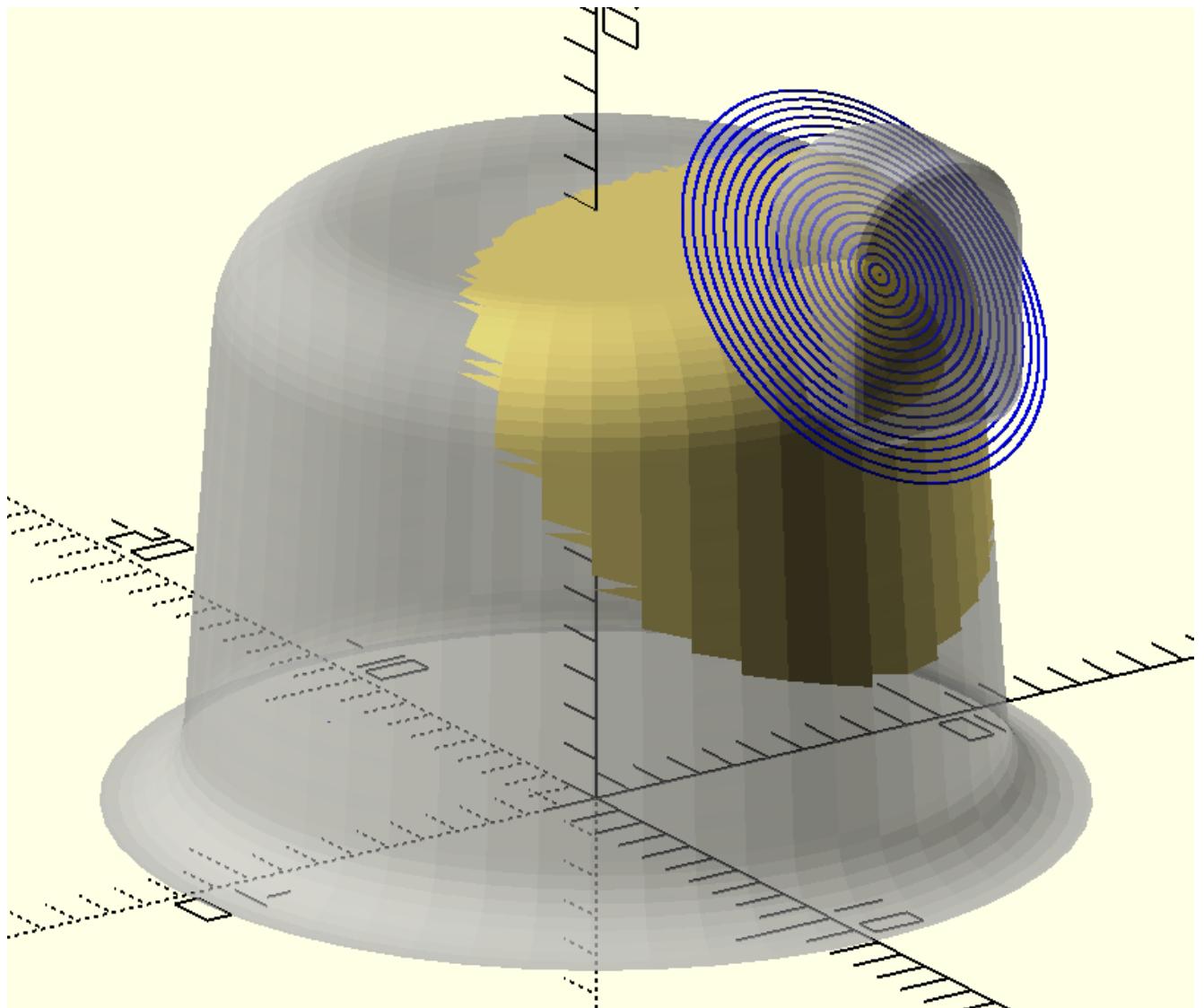
''')

t1=time.time()
t1-t0

# len(ip2),len(ip3),len(ip4)

```

Out[45]: 0.8456509113311768



In [46]:

```

i_t=time.time()
sec1=circle(55)
path1=corner_radius(pts1([[-50,0],[50,0,.2],[0,30,3],[6,1,3],[0,6,3],[-4,2,3],[0,22,6],[8,2,6],[0,10,..
# path1=equidistant_path(path1,200)
sec2=circle(7.5)
path2=corner_radius(pts1([[-5,0],[5,0,5],[0,35,2],[-5,0]]),10)
path2=equidistant_path(path2,100)

```

```

sol1=prism(sec1,path1)
sol2=translate([57.5,0,37],prism(sec2,path2))
sol2=axis_rot_o([0,0,1],sol2,180)

v,f1=partial_surface(sol1,prism_center(sol2),30)

i_p1=ip_tri2sol(v,f1,cpo(sol2))
p1=[p[0] for p in i_p1]
p2=[p[-1] for p in i_p1]
p3=flip(p1)+p2

fillet1=i_line_tri_fillet(v,f1,sol2,p3,3,-3,s=20)

with open('trial.scad','w+') as f:
    f.write(f'''
        include<dependencies2.scad>

%{swp(sol1)}
%{swp(sol2)}
//polyhedron({v},{f1});

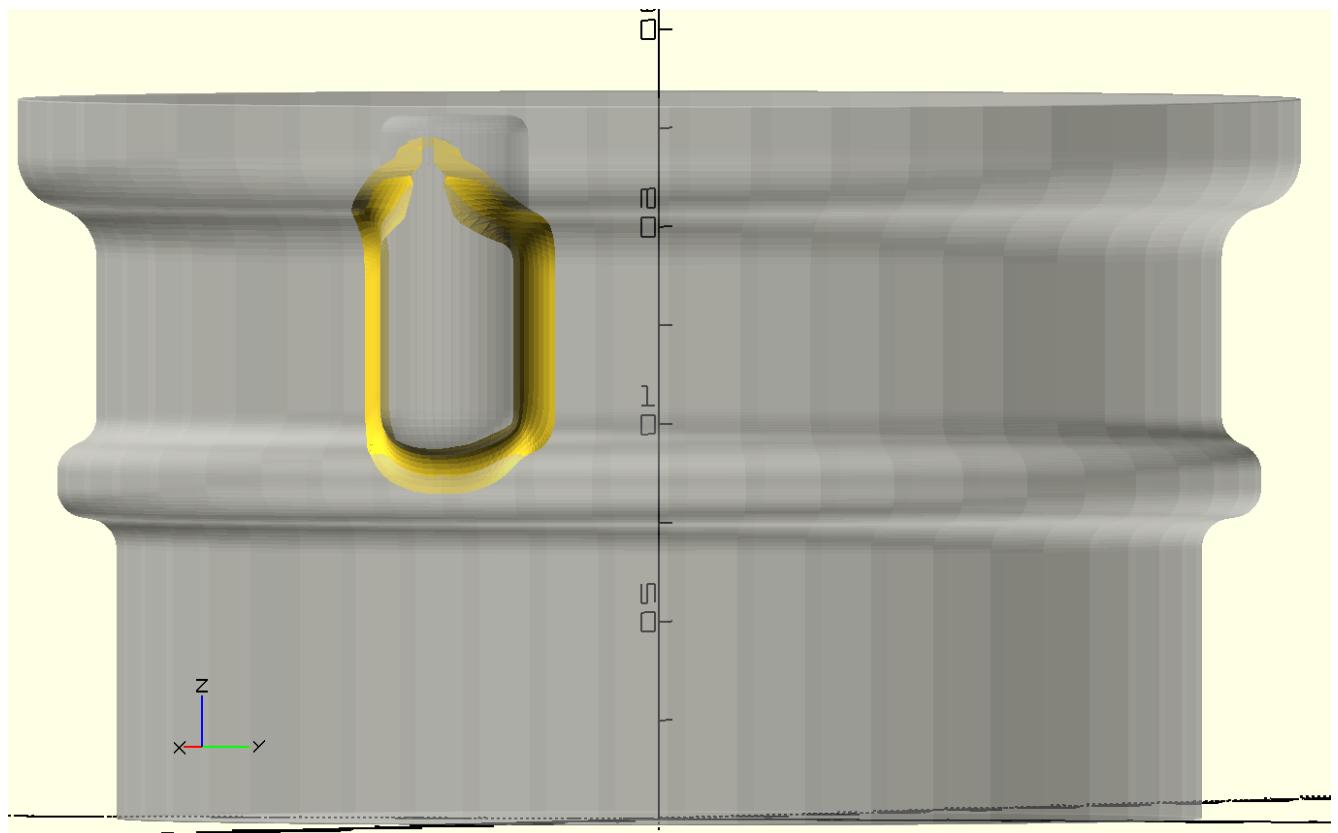
{swp_c(fillet1)}

'''')

f_t=time.time()
f_t-i_t
# len(p3),len(p4),len(p5)

```

Out [46]: 2.096005916595459



In [49]: t1=time.time()

```

p1=[[0,-15,.5],[6,0,.3],[0,-16,.1],[1,-1,.1],[4,0,.2],[2,10,4],[35,0,10],
    [5,22,5],[-5,22,10],[-35,0,4],[-2,10,.2],[-4,0,.1],[-1,-1,.1],[0,-16,.3],
    [-6,0,.5]]
sec1=corner_radius(pts1(p1),10)
path1=c2t3(circle(10,s=72))
sol1=path_extrude_closed(sec1,path1)

sec3=circle(7,s=100)
path2=corner_radius(pts1([[2,0],[-2,31]]),10)

sol3=translate([51,0,12],prism(sec3,path2))
v,f1=partial_surface(sol1,prism_center(sol3),30)

```

```

p2=ip_tri2sol(v,f1,sol3)
p2=[p[0] for p in p2]
fillet1=i_line_trifillet(v,f1,sol3,p2,3,-3)

f3=end_cap(sol3,2)[1]
with open('trial.scad','w+')as f:
    f.write(f''''
include<dependencies2.scad>

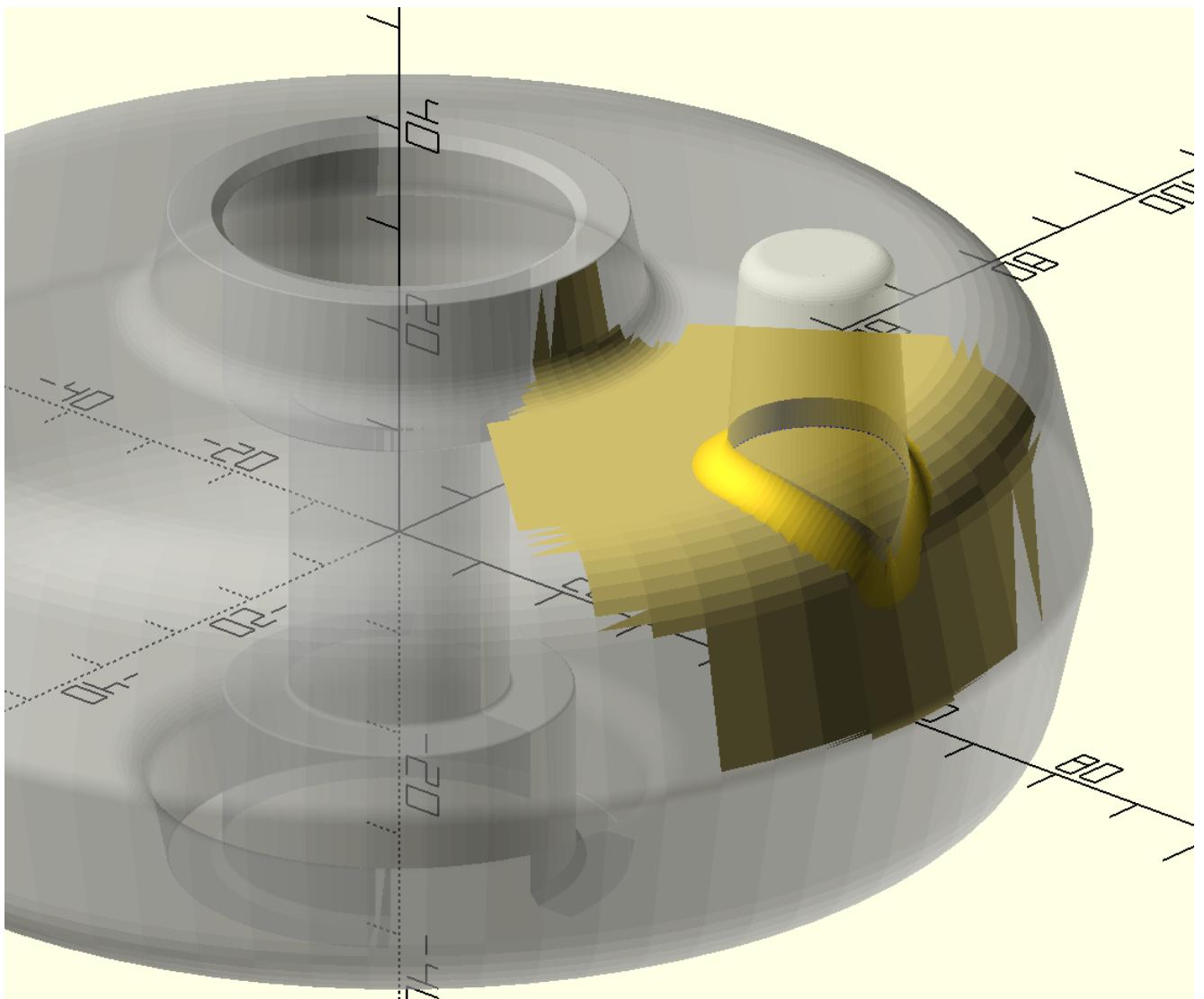
%{swp_c(sol1)}
%difference(){{{
{swp(sol3)}
{swp_c(f3)}
}}
{swp_c(fillet1)}
polyhedron({v},{f1});
color("blue")points({p2},.2);

'''')

t2=time.time()
t2-t1
# len(p1),len(p2),len(p3)

```

Out [49]: 0.4958629608154297



In [50]: i_t=time.time()

```

r=26.27
cp1,cp2,cp3,cp4=[0,0],[0,r],[r*cos(d2r(90+120)),r*sin(d2r(90+120))],[r*cos(d2r(90+240)),r*sin(d2r(90+240))]
r1,r2=22.5,6
a1=t_circarc(r1,r2,cp1,cp2,7.5,0)
a2=t_circarc(r1,r2,cp1,cp2,7.5,1)
a3=t_circarc(r1,r2,cp1,cp3,7.5,0)
a4=t_circarc(r1,r2,cp1,cp3,7.5,1)
a5=t_circarc(r2,r1,cp4,cp1,7.5,1)

```

```

a6=t_cir_tarc(r2,r1,cp4,cp1,7.5,0)

a12=arc_2p(a1[-1],a2[0],r2,-1)[1:-1]
a23=arc_2p(a2[-1],a3[0],r1,-1)[1:-1]

a34=arc_2p(a3[-1],a4[0],r2,-1)[1:-1]
a45=arc_2p(a4[-1],a5[0],r1,-1)[1:-1]

a56=arc_2p(a5[-1],a6[0],r2,-1)[1:-1]
a61=arc_2p(a6[-1],a1[0],r1,-1)[1:-1]

sec1=a1+a12+a2+a23+a3+a34+a4+a45+a5+a56+a6+a61

path1=corner_radius(pts1([[-22,0],[22,0],[0,3],[-22,0]]),10)
sol1=prism(sec1,path1)

sec2=circle(33.75/2,s=100)
path2=corner_radius(pts1([[0,0.001],[0,8.63],[-1.125,0],[0,3.75],[1.125,0],[0,7.5],[-1.125,0],[0,3.75]])
sol2=prism(sec2,path2)

fillet1=ip_fillet(sol1,sol2,1.88,-1.88)

sol3=translate([0,0,-.1],linear_extrude(circle(14.06,s=100),30))

sol4=translate([0,0,-.1],linear_extrude(circle(2.81,[0,26.27]),4))
sol4=[q_rot([f'z{i}'],sol4) for i in [0,120,240]]

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>
//color("magenta")p_line({sec1},.1);

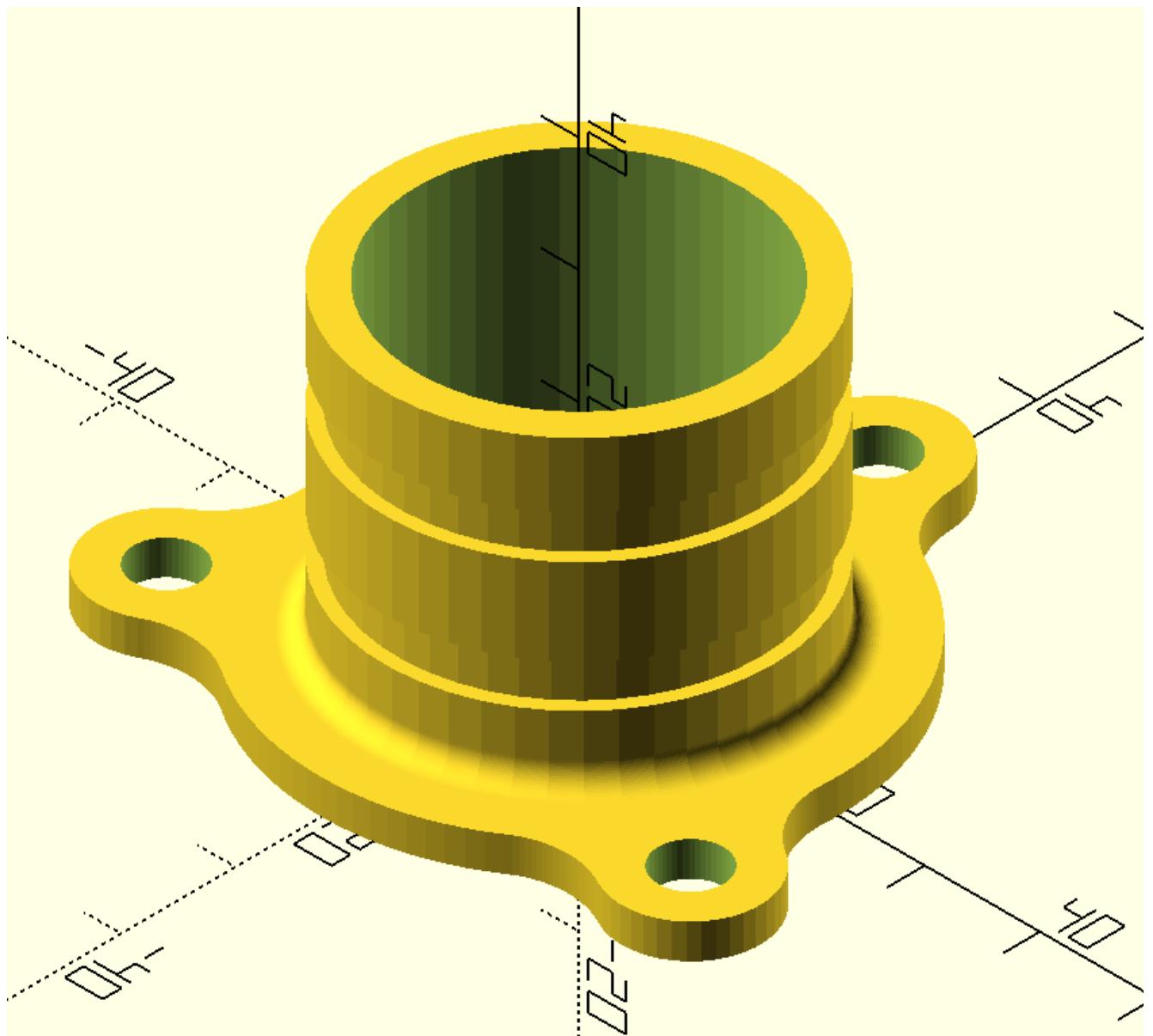
difference(){
union(){
{swp(sol1)}
{swp(sol2)}
}
{swp(sol3)}
for(p={sol4})swp(p);

}
{swp_c(fillet1)}
''')

f_t=time.time()
f_t-i_t

```

Out[50]: 2.7332160472869873



```
In [51]: i_t=time.time()

sec1=r_sec(60,60,[-55/2,0],[55/2,0],s=50)[:-1]
path1=corner_radius(pts1([[0,0],[0,8],[-14.5,0,2],[0,24,20],[-25,0]]),10)

sol1=prism(sec1,path1)
sol1=sol1+[sort_points(sol1[-1],[[0,0,24+8]])]
sec2=offset(sec1,-22.5)
path2=corner_radius(pts1([[0,-0.01],[0,24.01,12],[-12-25.5,0]]),10)
sol2=prism(sec2,path2)
sol2=sol2+[sort_points(sol2[-1],[[0,0,24]])]

sec3=circle(8)
path3=corner_radius(pts1([[0,16],[0,27,2],[7,0,2],[0,5]]),10)
sol3=prism(sec3,path3)

fillet1=ip_fillet(sol1,sol3,2,-2)
fillet2=ip_fillet(sol2,flip(sol3),2,-2)

sol4=translate([34.75,0,28],linear_extrude(circle(9,8)))
fillet3=ip_fillet(sol1,sol4,2,-2)

sol5=translate([0,0,16-.5],linear_extrude(circle(4,33)))
sol6=translate([34.75,0,24-.5],linear_extrude(circle(5,13)))

sol7=translate([0,52.5,-.1],linear_extrude(circle(4,9)))
sol8=[translate([-55/2,0,0],q_rot([f'z{i}'],sol7)) for i in [0,60,120,180]]
sol9=[translate([55/2,0,0],q_rot([f'z{i}'],sol7)) for i in [0,-60,-120,-180]]

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>
//color("blue")p_line({sec1},.05);
```

```

//color("blue")p_lineo({path1},.2);
sol8={sol8};
sol9={sol9};
difference(){{
union(){{
difference(){{
{swp(sol1)}
{swp(sol2)}
for(i=[0,1,2,3])swp(sol8[i]);
for(i=[0,1,2,3])swp(sol9[i]);
}}
{swp(sol3)}
{swp(sol4)}
}}
{swp(sol5)}
{swp(sol6)}
}};

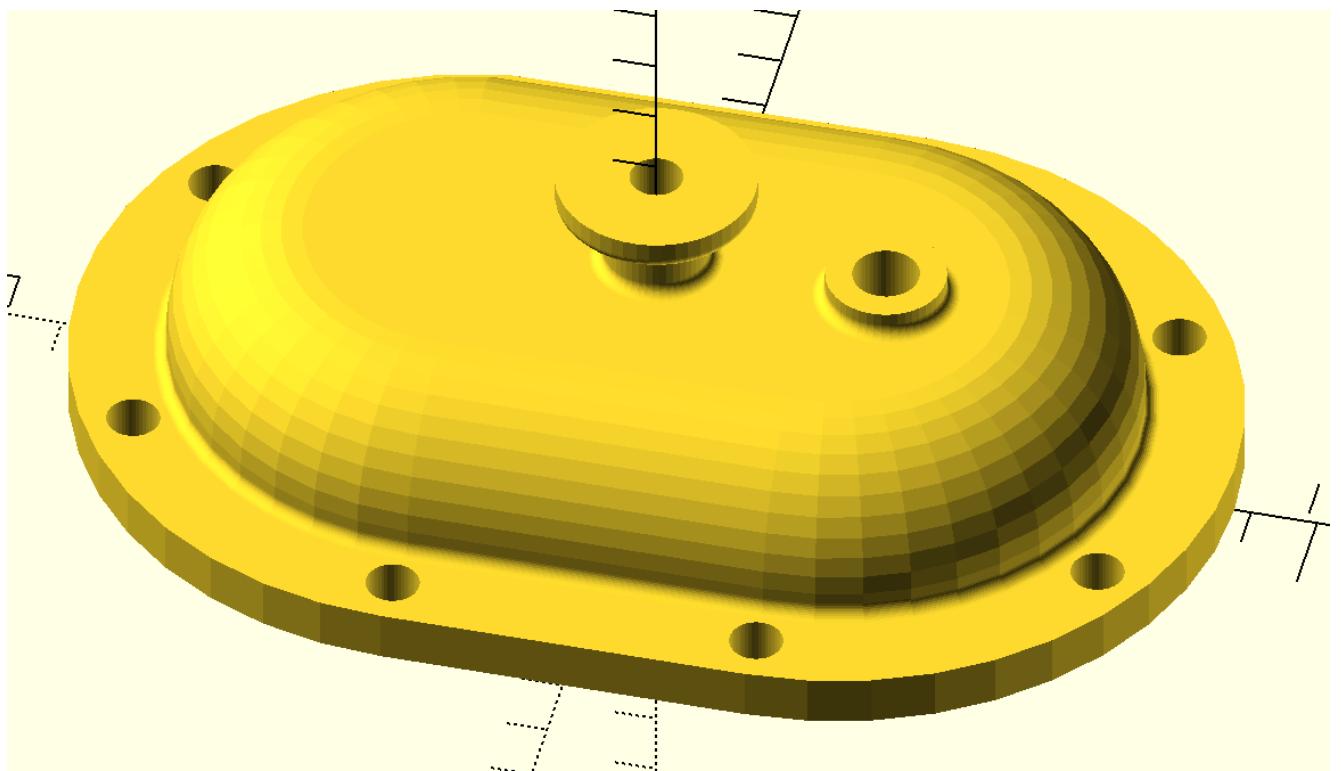
{swp_c(fillet1)}
{swp_c(fillet2)}
{swp_c(fillet3)}

//color("blue")for(p={sol1})p_line3dc(p,.2,rec=1);
'''')

f_t=time.time()
f_t-i_t

```

Out[51]: 2.5401549339294434



In [52]: i_t=time.time()

```

a1=c2t3(arc_2p([0,0],[5,28],58,-1))
a2=flip(c2t3(arc_2p([-1,2],[3,28],56,-1)))

a3=[[-5,0,0],[0,0,3]]+a1[2:-1]+[[5,28,.9],[3,28,.9]]+a2[1:-1]+[[-1,2,2],[-5,2,0]]
path1=corner_radius(a3,10)
sec1=circle(43/2,s=100)
sol1=prism(sec1,path1)

c1=circle(7.75,[10,18])
p0=[0,8]
p1=p_cir_t(p0,c1)
p2=c1[19]
a4=arc_long_2p(p1,p2,7.75,-1,s=50)
path2=cytz([p0]+a4)
sec2=circle(2)
sol2=translate([18.7,0,-3],align_sol_1(path_extrude_open(sec2,path2)))

```

```

a5=[[-5,0,0],[0,0,5]]+a1[1:-1]+[[5,28,1],[3,28,0]]
path3=corner_radius(a5,10)
sol3=prism(sec1,path3)
fillet1=ip_fillet(sol3,sol2[:-10],2,-2)
fillet2=ip_fillet(sol3,flip(sol2[30:]),2,2)

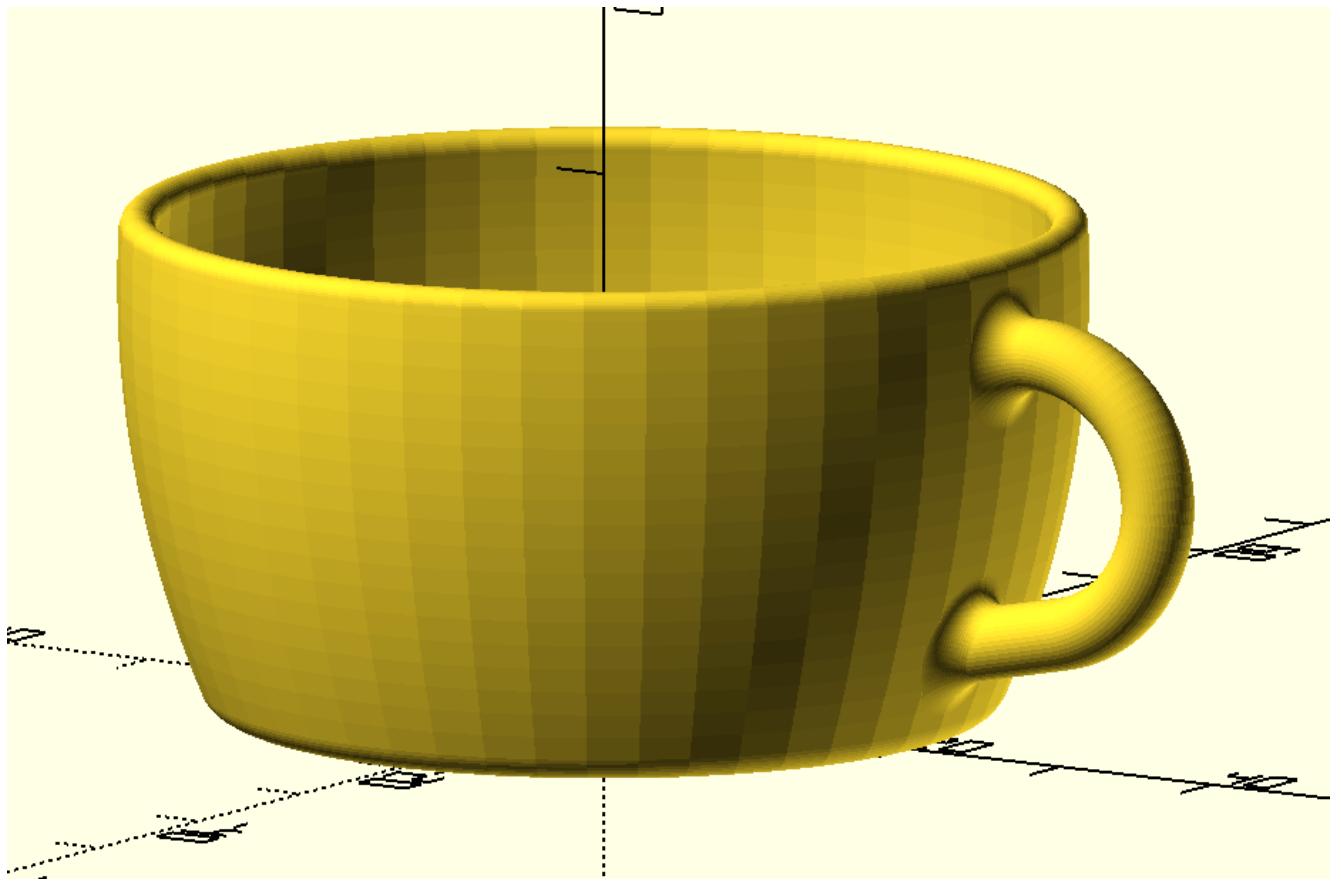
sol4=cut_plane([0,-1,0],[100,100],100)
with open('trial.scad','w+') as f:
    f.write(f'''

difference() {{
union() {{
{swp(sol1)}
difference() {{
{swp(sol2)}
{swp(sol3)}
}}
{swp(cpo(fillet1)[:-1])}
{swp(cpo(fillet2)[:-1])}
}}
//{swp(sol4)}
}}
''')

f_t=time.time()
f_t-i_t

```

Out[52]: 6.507597208023071



```

In [53]: sec=corner_radius(pts1([[-2.5,-2.5,0],[5,0,2.5],[0,2.5,0.5],[5,0,0]]),10)

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>
color("blue")p_lineo({sec},.05);

'''')

```

```

In [23]: s=100
x=2
a=linspace(0,4,s)
b=sin(d2r(720/x*a))

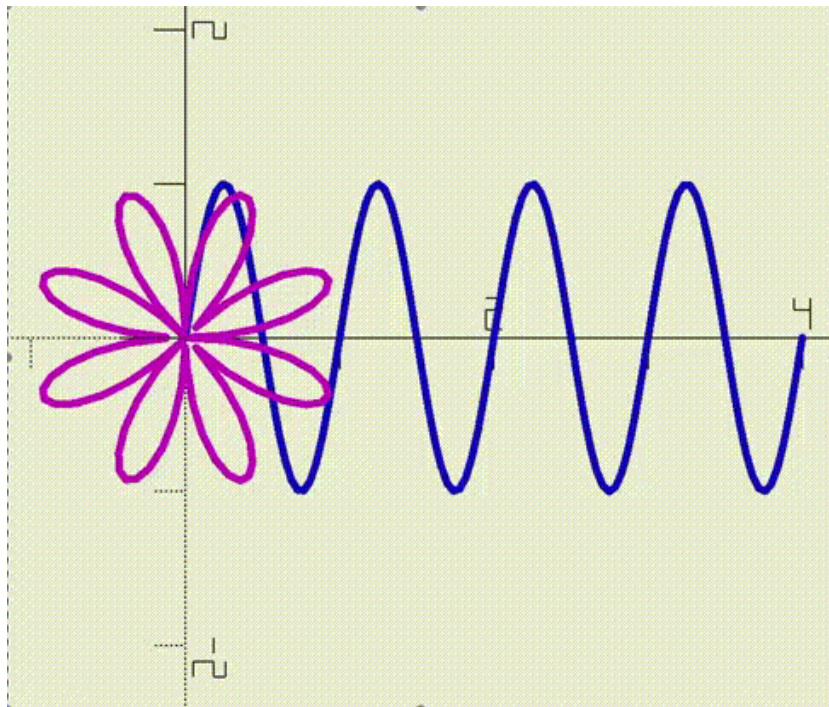
```

```

theta=720/x*a
c=cos(d2r(theta))
d=sin(d2r(theta))
e=array([[cos(d2r(i)),sin(d2r(i))] for i in linspace(0,360,s)])
e=einsum('ij,i->ij',e,abs(b)).tolist()
sw=array([a,b]).transpose(1,0).tolist()

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies.scad>
e={e};
sw={sw};
color("blue")p_lineo(loop(sw,0,100*$t),.05);
color("magenta")p_lineo(loop(e,0,100*$t),.05);

...)
```



```

In [24]: s=200
x=2
a=linspace(0,4,s)
b=sin(d2r(270/x*a))

theta=270/x*a
c=cos(d2r(theta))
d=sin(d2r(theta))
e=array([[cos(d2r(i)),sin(d2r(i))] for i in linspace(0,360,s)])
e=einsum('ij,i->ij',e,abs(b)).tolist()
sw=array([a,b]).transpose(1,0).tolist()

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies.scad>
e={e};
sw={sw};
color("blue")p_lineo(loop(sw,0,200*$t),.05);
color("magenta")p_lineo(loop(e,0,200*$t),.05);

...)
```

```

In [18]: # 3d knots various types
t0=time.time()
# trefoil knot

path=[[10*(sin(t)+2*sin(2*t)),
       10*(cos(t)-2*cos(2*t)),
```

```

-10*sin(3*t)] for t in d2r(arange(0,360))]

# circular sin theta knot

# path=[[60*(cos(t)),
#        60*(sin(t)),
#        20*sin(4*t)*cos(4*t)] for t in d2r(arange(0,360))]

# random knot

# path=[[20*(-0.22*cos(t) - 1.28*sin(t) - 0.44*cos(3*t) - 0.78*sin(3*t)),
# 20*(-0.1*cos(2*t) - 0.27*sin(2*t) + 0.38*cos(4*t) + 0.46*sin(4*t)),
# 20*(0.7*cos(3*t) - 0.4*sin(3*t))] for t in d2r(arange(0,360))]

# torus knots

# path=[[10*cos(3*t)*(3+cos(4*t)),
# 10*sin(3*t)*(3+cos(4*t)),
# 10*sin(4*t)] for t in d2r(arange(0,360))]

# cinquefoil torus knots
a,p,q=3,3,16
d=10

# explanation
# radius of the torus = a*d
# section radius of the torus = d
# p in number of cycles of the wrapping coil over torus
# q in the number of turns of the wrapping coil over torus

path=[[d*cos(p*t)*(a+cos(q*t)),
d*sin(p*t)*(a+cos(q*t)),
-d*sin(q*t)] for t in d2r(arange(0,360,1))]

# Lissajous knots

# path=[[10*cos(3*t+5),
# 10*cos(3*t+10),
# 10*cos(3*t+2)] for t in d2r(arange(0,360))]
r=2
sec=circle(r)
sol=align_sol_1(path_extrude_closed(sec,path))

sec1=circle(d-r)
path1=c2t3(circle(a*d))
sol1=path_extrude_closed(sec1,path1)

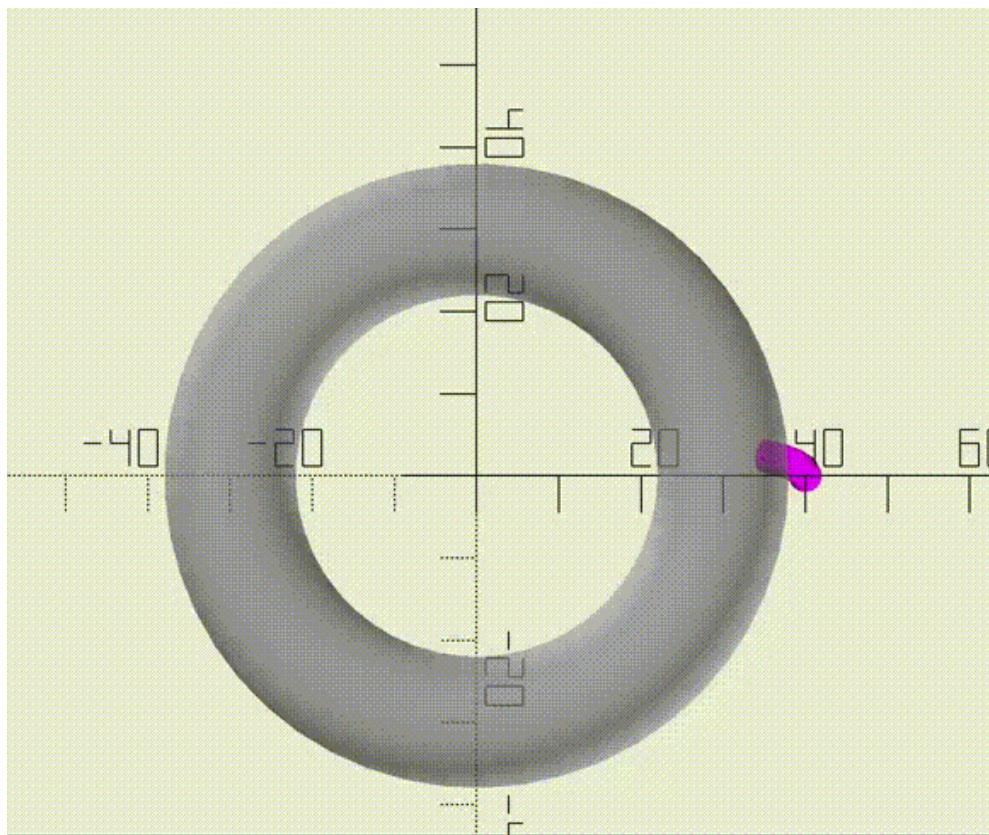
with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies.scad>
//color("blue")p_line3d({path},4);
sol={sol};
//color("magenta")swp_c(loop(sol,0,360*$t));
color([.2,.5,.7,.3]){\swp_c(sol)}

//%{\swp_c(sol1)}

''')
t1=time.time()
t1-t0

```

Out[18]: 8.435809850692749



```
In [54]: sec=circle(25)
path=corner_radius(pts1([[0,0,0],[0,15,1],[-6,50,1],[0,18,0]]),10)
sol=prism(sec,path)
sol1=prism(circle(24),path)
sol0=swp_prism_h(sol,sol1)
sec2=circle(15)
path2=cr_3d([[0,0,15,0],[-60,0,40,20],[0,0,30,0]],10)
sol2=path_extrude_open(sec2,path2)
sol3=path_extrude_open(offset(sec2,-1),path2)
sol02=swp_prism_h(sol2,sol3)
fillet1=ip_fillet(sol,sol2,5,-5)

sec3=corner_radius(pts1([[0,0],[-2,1,.2],[2,1]]),10)
path3=helix(14,2.5,6,5)
sol4=path_extrude_open(sec3,path3)

sol4=sol2vector([0,0,1],sol4,[-60,0,70])
with open('trial.scad','w+') as f:
    f.write(f'''
difference(){
{swp_c(sol0)}
{swp(sol3)}
}
difference(){
{swp_c(sol02)}
{swp(sol)}
}
{swp_c(fillet1)}
intersection(){
color("blue"){swp(sol4)}
{swp(sol3)}
}
...
''' )
```

sinwave-box

```
In [3]: i_t=time.time()
# sinwave glass
height=125
dia=100
width=pi*dia
factor=round(width/height,0)
sec=[[i,j,1*sin(d2r(i*360/height*3))*sin(d2r(j*360/width*3*factor))] for j in linspace(0,width,150)]
```

```

# path=translate([0,100/pi/2,100/pi/2],q_rot(['y90'],arc(100/pi/2,0,400,s=200)))
path=q_rot(['y90'],arc(dia/2,0,400,s=200))

surf1=[wrap_around(p,path)[:-1] for p in sec]
surf2=offset_sol(surf1,-2)
sol1=q_rot(['y-90'],swp_prism_h(surf2,surf1))
sol2=q_rot(['y-90'],surf1[:2])

p0=sol1[2][:-1]
p1=sol1[15][:-1]
p2=offset_3d(p0,-15)
fillet1=convert_3lines2fillet_closed(p2,p1,p0,s=30)

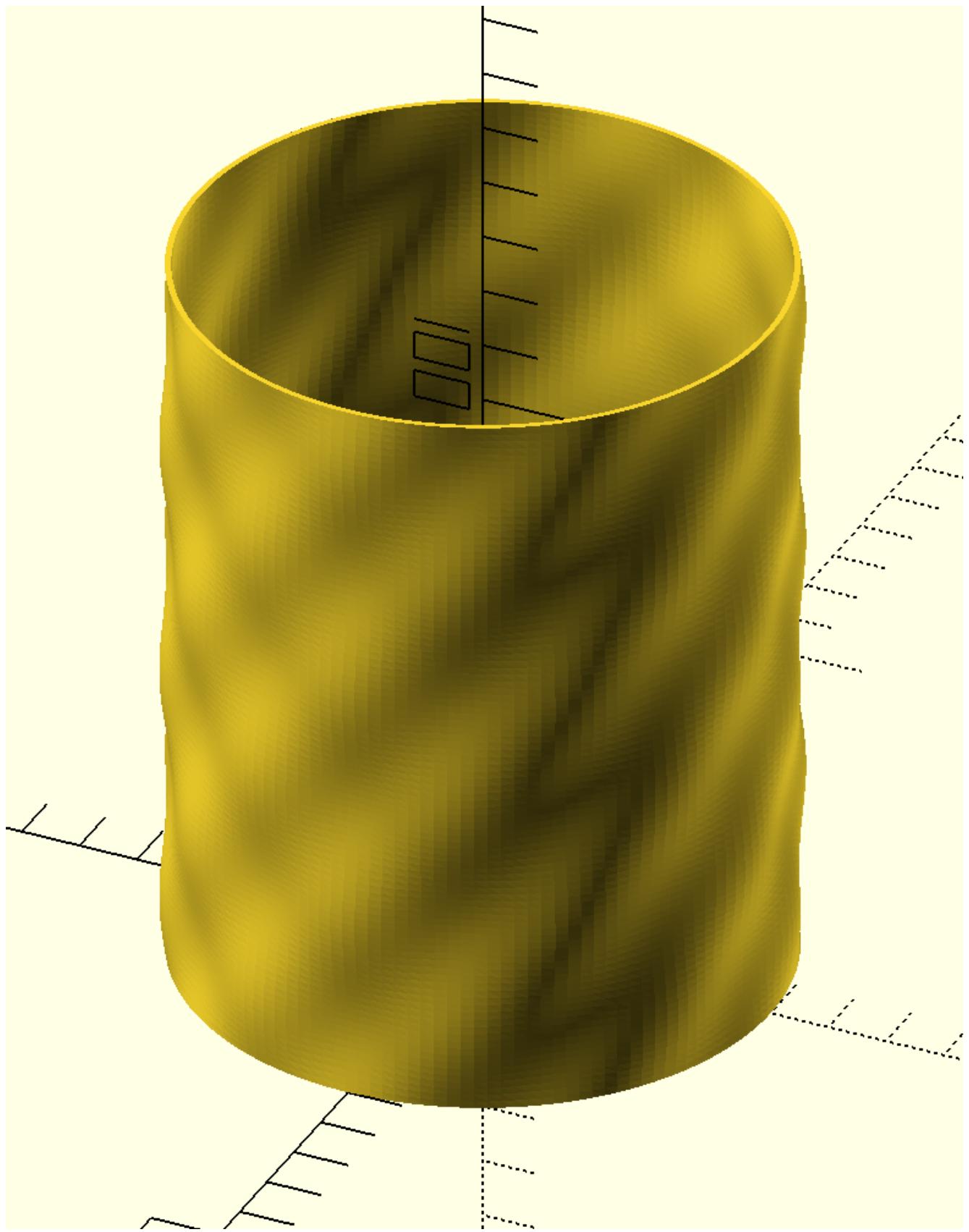
sol3=flip(sol1)[:-15]+flip(cpo(fillet1)[1:-1])

sol4=cut_plane([-1,0,0],[300,300],300,0,0,0)
with open('trial.scad','w+') as f:
    f.write(f'''
        include<dependencies2.scad>
        difference(){{
            {swp(sol3)}
            //{swp(sol4)}

        }}
    ''')
f_t=time.time()
f_t-i_t

```

Out [3]: 2.672560930252075



```
In [56]: i_t=time.time()
# sinwave-box
d=100
s=30
path=corner_radius(pts1([[-d/2,-d/2,d/5],[d,0,d/5],[0,d,d/5],[-d,0,d/5]]),s)
path=path[s+1:]+path[:s]
path=equidistant_pathc(q_rot(['y90'],path),500)
path=path+[path[0]]
l1=int(l_lenv(path))
sec=[[i,j,1*sin(d2r(i*360/125*3))*sin(d2r(j*360/l1*6))] for j in linspace(0,l1,300) for i in linspace(-15,15,30)]
surf1=[wrap_around(p[:-1],path) for p in sec]
surf2=offset_sol(surf1,-2)
sol1=q_rot(['y-90'],swp_prism_h(surf1,surf2))

p0=sol1[-4][-1]
p1=sol1[-15][-1]
```

```

p2=offset_3d(p0,-11)
fillet1=convert_3lines2fillet_closed(p2,p1,p0,s=30)
sol2=sol1[:-12]+flip(cpo(fillet1)[1:-1])

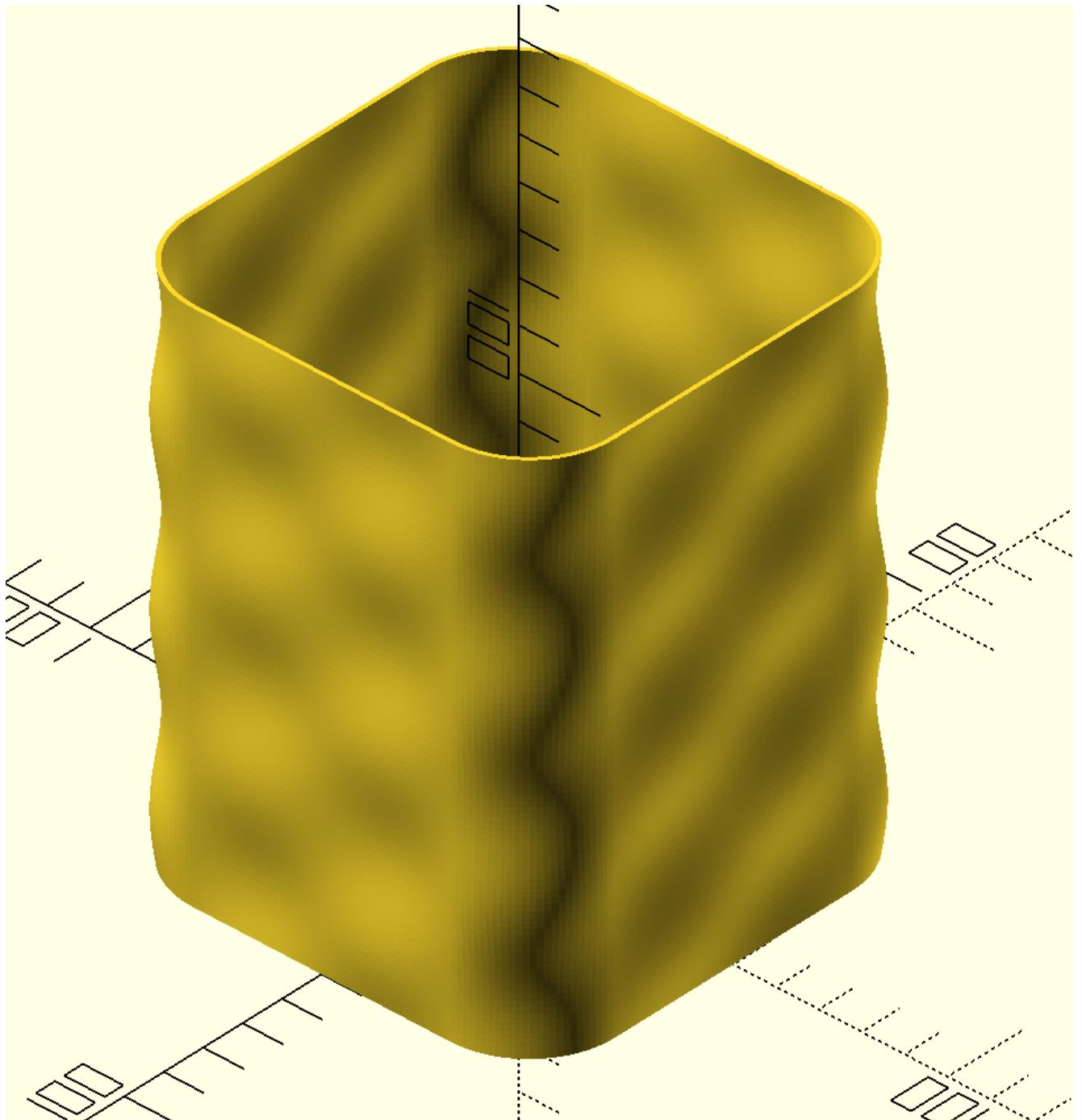
sol3=cut_plane([0,-1,0],[300,300],300,theta=[0,0,45])
with open('trial.scad','w+') as f:
    f.write(f''''
        include<dependencies2.scad>

        difference() {{
            {swp(sol2)}
        //{swp(sol3)}

    }}
    ''')
# l_lenv_o(path)
f_t=time.time()
f_t-i_t

```

Out[56]: 7.478494882583618



q_rot2d

translate_2d

In [57]:

```

c1=circle(22.5)
c2=translate_2d([26.25,0],circle(6))
c3=[q_rot2d(i,c2)  for i in [0,120,240]]

a1=t_cir_tarc(22.5,6,[0,0],[26.25,0],7.5)
p0,p1=a1[0],a1[-1]

a2=t_cir_tarc(22.5,6,[0,0],[26.25,0],7.5,side=1)
p2,p3=a2[0],a2[-1]
p4=q_rot2d(120,p0)

sec=arc_2p(p0,p1,7.5,cw=1)+arc_2p(p1,p2,6,cw=-1)+ \
arc_2p(p2,p3,7.5,cw=1)+arc_2p(p3,p4,22.5,cw=-1)

sec=[q_rot2d(i,sec)  for i in [0,120,240]]

sec=remove_extra_points(concatenate(sec).round(5))

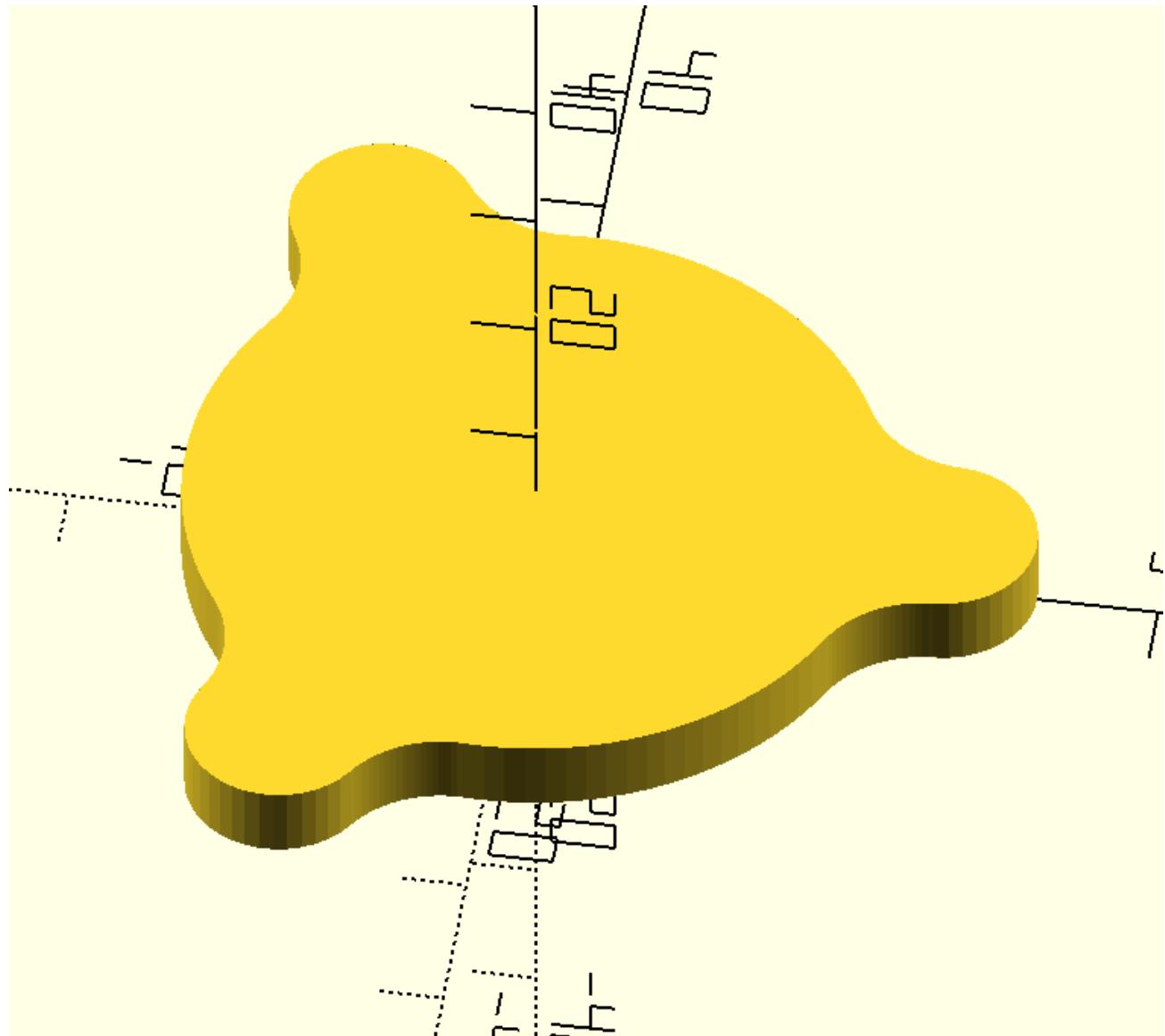
sol=linear_extrude(sec,5)

with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>
//color("blue",.2)p_line3dc({c1},.3);
//color("magenta",.2)for(p={c3})p_line3dc(p,.3);

//color("cyan")p_line3dc({sec},.5);
//color("blue")points({[p4]},1);
{swp(sol)}

    ''')

```



In [60]:

```

peri=400
height=100

```

```

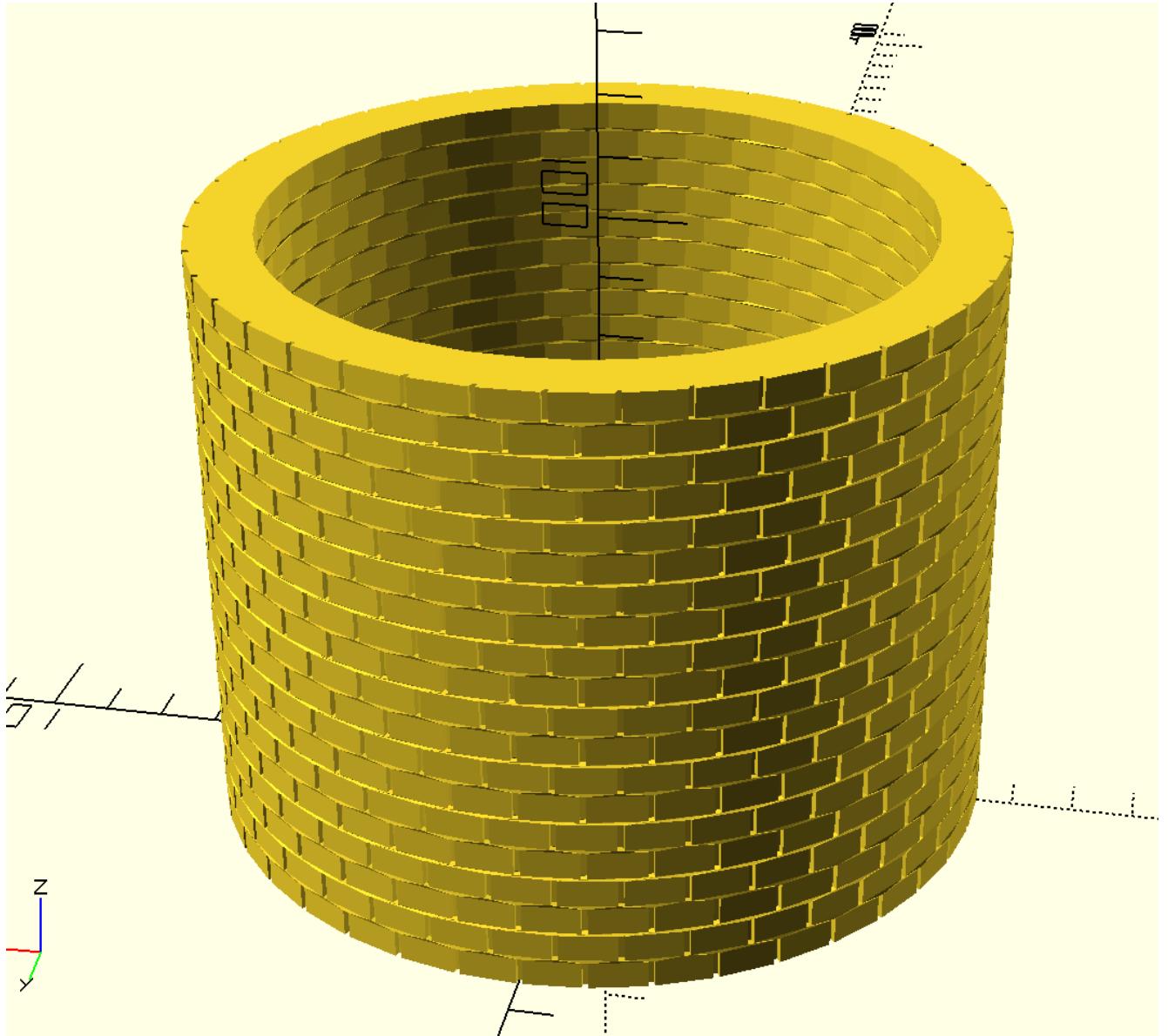
sec=corner_radius(pts1([[0,0],[9,0],[0,1],[1,0]]),10)
sec=concatenate([translate_2d([i,0],sec) for i in arange(0,peri,10)]).tolist()
sec1=translate_2d([4.5,5],sec)
sec2=[sec]+[sec1]
sec=concatenate([translate_2d([0,i],sec2) for i in arange(0,height,10)]).tolist()
sec=[translate(array(p).mean(0),q_rot(['x90']),translate_2d(-array(p).mean(0),p))) for p in sec]
sec1=translate([0,4.5,0],sec)
sec=array([sec,sec1]).transpose(1,0,2,3)
_,_,a,b=sec.shape
sec=sec.reshape(-1,a,b).tolist()
sec=translate([0,.1,0],q_rot(['z90']),sec))
path=circle(peri/(2*pi),s=100)
path=path+path[:10]
path=q_rot(['y90'],path)
sol=q_rot(['y90'],[wrap_around(p,path) for p in sec])
sol1=offset_sol(sol,-1.5)

sol2=swp_prism_h(sol,sol1)
with open('trial.scad','w+') as f:
    f.write(f'''
include<dependencies2.scad>
//color("blue")for(p={sol})p_line3d(p,.2);
//color("magenta")p_line3d({path},.2);

{swp_c(sol2)}\n
    ''')

```

''')



In []:

```

sec=corner_radius(pts1([[0,0],[70,0],[0,28],[-17.5,0,7],[0,24.5,17.5],
                      ,[-35.01,0,17.5],[0,-24.5,7],[-17.5,0]]),20)

cp_1=cp_arc(sec[26:30])
c1=circle(10,cp_1)
c2=circle(2.5,[7.5,14])

```

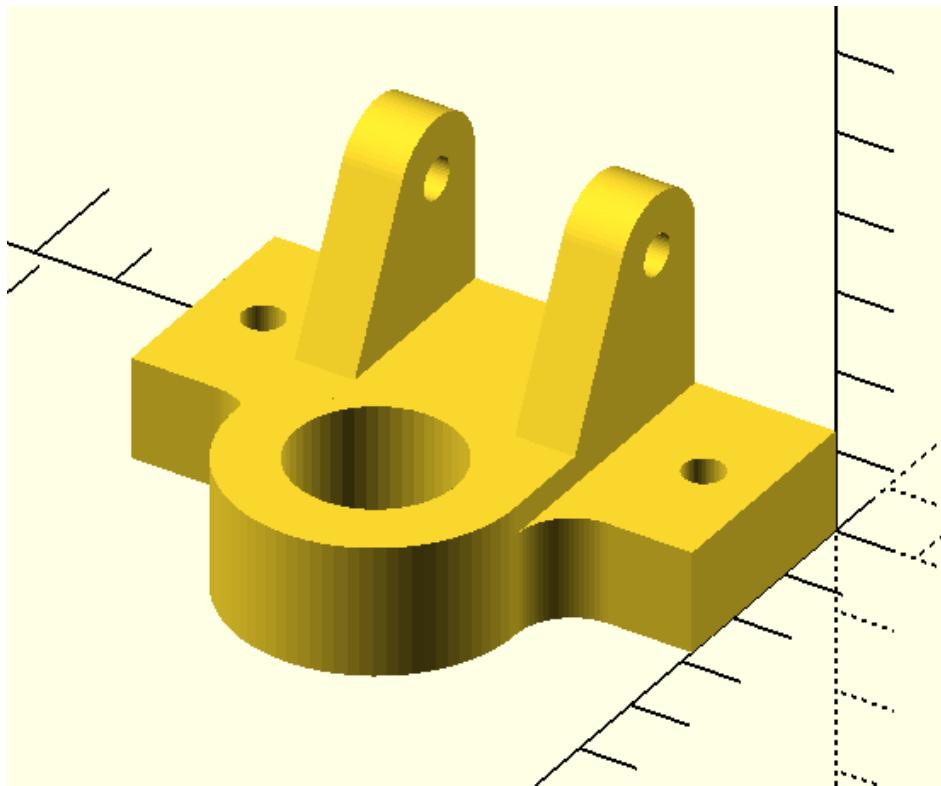
```

c3=circle(2.5,[62.5,14])
sol1=linear_extrude(sec,16)
sol2=[translate([0,0,-.5],linear_extrude(p,17)) for p in [c1,c2,c3]]
sol3=translate([-52.5+29,16],cube([18,40,20]))
sol4=translate([70-17.5,-.5,12.5],cube([18,40,20]))
c4=circle(7.5,[-7.5,32.5])
p0=[0,0]
p1=p_cir_t(p0,c4)
p3=[-52.5+29,16]
p2=cir_p_t(c4,p3)
p4=[-52.5+17.5,16]
p5=[-52.5+17.5,0]
c5=circle(2.5,[-7.5,32.5])
sec1=[p0]+arc_2p(p1,p2,7.5,-1,30)+[p3,p4,p5]

sol5=linear_extrude(sec1,7.5)
sol6=translate([0,0,-.5],linear_extrude(c5,9))
with open('trial.scad','w+') as f:
    f.write(f'''
        include<dependencies2.scad>
//color("blue")p_line3d({sec},.2,1);
//color("magenta")p_line3d({c1},.2,1);
//color("magenta")p_line3d({c2},.2,1);
//color("magenta")p_line3d({c3},.2,1);
//color("blue")points({[p0,p1,p2,p3,p4,p5]},.5);
//color("magenta")p_line3dc({sec1},.2,1);
for(i=[17.5,17.5+35-7.5])
translate([i+7.5,0,0])
rotate([90,0,-90])
difference() {{
{swp(sol5)}
{swp(sol6)}
}}


difference() {{
{swp(sol1)}
for(p={sol2})swp(p);
{swp(sol3)}
{swp(sol4)}
}}
    ''')

```



sinewave

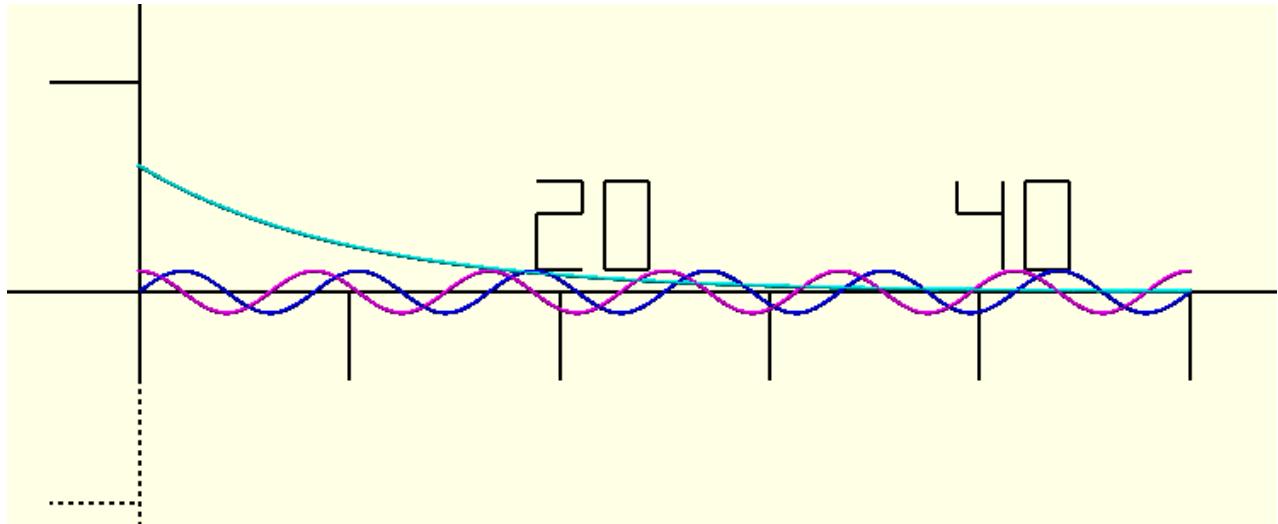
cosinewave

e_wave

```
In [9]: wav1=sinewave(50,6,1,200)
wav2=cosinewave(50,6,1,200)
wav3=e_wave(50,6,.1,200)
```

```
with open('trial.scad','w+') as f:
    f.write(f'''
        include<dependencies2.scad>
color("blue") p_line3d({{wav1}},.2);
color("magenta") p_line3d({{wav2}},.2);
color("cyan") p_line3d({{wav3}},.2);
```

```
'''')
```



```
In [ ]: wav1=sinewave(50,6,1,200)
wav2=sinewave(50,9,1,200)

a=[[wav1[i][0] , wav1[i][1]*wav2[i][1] ] for i in range(len(wav1))]
# b=[[wav1[i][0] , wav1[i][1]*wav2[i][1] ] for i in range(len(wav1))]

b=q_rot(['x90'],a)
c=q_rot(['x90','z90'],a)

surf_1=surface_from_2_waves(b,c,1)
surf_2=surf_extrudef(surf_1,-.1)

path=circle(50/2/pi+.01,s=100)
path=path+path[:3]
path=q_rot(['y90'],path)
sol=[ wrap_around(p,path) for p in surf_1]
sol=q_rot(['y-90'],sol)
h=[[0,0,p[0][2]] for p in sol]
s=c3t2(sol)
s1=[oset(p,-.5) for p in s]
sol1=[translate(h[i],s1[i]) for i in range(len(s1))]
sol2=swp_prism_h(sol,sol1)
sol2=scl3d(sol2,2)
with open('trial.scad','w+') as f:
    f.write(f'''
        include<dependencies2.scad>
{s1}
{sol2}
'''')
```

