INTERNAL GEAR PUMP

Instructional tutorial for the Lua script for IceSl

at Deggendorf Institute of Technology
at the Faculty of Mechanical Engineering and Mechatronics
Case Study Cyber Physical Production Systems using Additive manufacturing

Principal Supervisor: Prof. Dr.-Ing. Stefan Scherbarth

Author: 3D it Group
Arash Kadkhodaei Elyaderani
Mohammadmahdi Ataei
www.3dit.engineer
info@3dit.engineer

Submission: 8th May 2022

Contents

1 Introduction

This tutorial is going to help the user to develop the internal gear components using Lua scripts and prepare components for 3D printing and also guidance for the assembly. We tried to write this script in a flexible and easy-to-use way so everyone can understand and work with it. Every important variable can change and the final result can be in your desired shape. Mainly, these scripts produce four main components:

- Rotor

- Idler

- crescent

- Gear housing

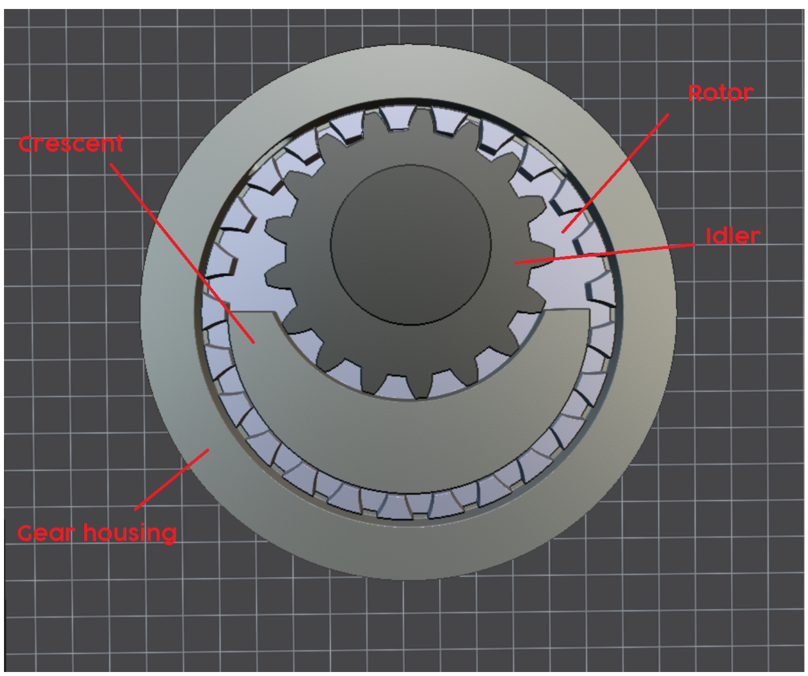

Figure 1 Main Parts

2 Script description

In order to explain the script we will focus on two main parts:

1- Parameters 2- Functions

2.1 Input Parameters

Firstly, we get the input variables through the interface box:

```
---- Input parameters using Interface
z_2 = ui_numberBox("Number Of Teeth for Rotor Gear", 25);  --number of
    teeth Rotor
z_1 = ui_numberBox("Number Of Teeth for Idler Gear", 17);  --number of
    teeth Idler
m = ui_numberBox("Module Of Gear", 4);  -- Module
alpha_t = ui_scalarBox("Pressure Angle", 20, 1);  -- Pressure angle
h_a_coef_p = ui_scalarBox("Addendum Coef(mm)", 1, 0.05);  -- Addendum
    height
h_f_coef_p = ui_scalarBox("Dedendum Coef(mm)", 1.25, 0.05);  --
    Dedendum height
x_coef_int = ui_scalarBox("Internal Profile Shift(mm)", 0.1, 0.05);  --
     Profile shift factor for internal gear
x_coef_ext = ui_scalarBox("External Profile Shift(mm)", 0, 0.05);  --
    Profile shift factor for external gear
b = ui_numberBox("Width(mm)", 10);  -- Thickness of the gear
rotation = ui_numberBox("Rotate", 0);  -- Rotation
```

As you can see the input variables in figure below and their default values which user can change them easily.

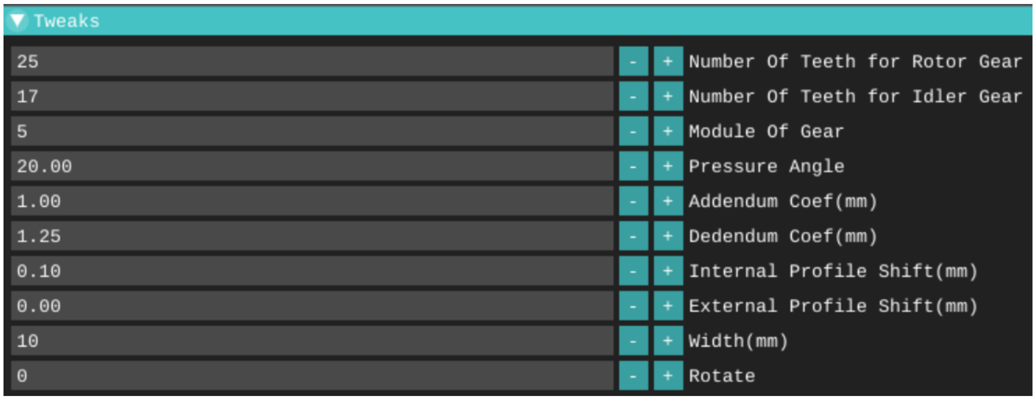

Figure 2 User Interface

2.2 Functions

before the main parts of the script we defined some functions which help us through the script to avoid repeating some tasks:

- **Involute Profile Function**: This function uses the basic formulas for calculating the involute points.

```
function involute_t(r_b, inv_alpha) -- r_b
    return v(r_b * (math.sin(inv_alpha) - inv_alpha * math.cos(
        inv_alpha)), r_b * (math.cos(inv_alpha) + inv_alpha * math.sin(
        inv_alpha)))
end
```

Then we have three simple functions for finding involute angle, rotation and mirroring the points.

```
---- Function for involute angle
function involute_angle(r_p1, r_p2)
    return (math.sqrt((r_p2 * r_p2 - r_p1 * r_p1) / (r_p1 * r_p1)))
end

---- Function for rotation matrix
function p_rotate(a, points) -- a = angle
    return v(math.cos(a) * points.x + math.sin(a) * points.y, math.cos(
        a) * points.y - math.sin(a) * points.x)
```

```
end

---- Function for mirroring
function mirror(point)
    return v(-point.x , point.y)
end
```

- **Gear Profile Function**:The most important function in this section is the `gearProfile` this function contains formulas and equations needed for the calculation of the involutes. we do not explain in detail about them here, but you can read more in the technical report.[1] This function contains the formulas and it calculates both the internal and external gear regarding the inputs. After the formulas, there are some loops to put all points together as output.

```
 -- loops for generation points
    local steps = 30;
    for i = 1, z do
        local th = 2 * math.pi
        local AngelE = involute_angle(r_TIF, r_a);  -- End asngle
        for j = 1, steps do
            inv_xy[#inv_xy + 1] = p_rotate(th * i / z, involute_t(r_TIF
                , ((AngelE) * j / steps))) -- firsr involute face
        end
        for j = steps, 1, -1 do
            inv_xy[#inv_xy + 1] = p_rotate(th * i / z, p_rotate(angel,
                mirror(involute_t(r_TIF, ((AngelE) * j / steps))))) --
                Mirroring
        end
    end
    inv_xy[#inv_xy + 1] = inv_xy[1] --table of values
    return linear_extrude(v(0, 0, b), inv_xy)
end
```

- **Circle Function**: This function is to make circles :

```
function circle(r)
    local x, y = 0, 0
    local XY = {}
    for i = 1, 360 do
        local angle = i * math.pi / 180
       XY[i] = v(x + r * math.cos(angle), y + r * math.sin(angle))
    end
    return XY
```

```
end
```

- **Working Pressure angle Function**: According to 1992 [Harry Cheng][2], from the result of the derivation of an explicit solution of the inverse involute function, we can find the working pressure angle.

```
function WorkingAngelF(x)
    return (((math.pow(3 * x, (1 / 3)))) - (2 * x / 5) + (math.pow(9 /
        175 * 3, (2 / 3))) * (math.pow(x, (5 / 3))) - (math.pow(2 / 175
        * 3, (1 / 3))) * (math.pow(x, (7 / 3))) - ((144 / 67375) * (
        math.pow(x, (9 / 3))) + (3258 / 3128215) * (math.pow(3, (2 / 3))
        ) * (math.pow(x, (11 / 3))) - (49711 / 153278125) * (math.pow(3,
         (1 / 3))) * (math.pow(x, (13 / 3))) - (1130112 / 9306171875) *
        (math.pow(x, (15 / 3))) + (5169659643 / 95304506171875) * (
        math.pow(3, (2 / 3))) * (math.pow(x, (17 / 3)))))
end
```

- **Center Function**:Next function is center function which is responsible for meshing and finding the center point according to coefficients. More about formulas can be find on [1].

```
function center()
    local alpha_rad = alpha_t * math.pi / 180;  -- Preassure angle
    local inv_a = math.tan(alpha_rad) - alpha_rad;  -- Involute
        function
    local inv_aw = ((2 * math.tan(alpha_rad) * (x_coef_ext - x_coef_int
        )) / (z_2 - z_1)) + inv_a;  -- Involute function working
        pressure angle

    -- Working preassure angle
    local alpha_aw = WorkingAngelF(inv_aw);
    -- Centre distance coeficiant factor
    local y = ((z_2 - z_1) * (math.cos(alpha_rad) - math.cos(alpha_aw))
        ) / (2 * math.cos(alpha_aw));
    -- Center distance between Gears
    local a_x = (((z_2 - z_1) / 2) + y) * m;  -- Center distance
    meshDistance = a_x;
end
```

- **Extrude Function**: will extrude a Contour to a shape by turning the contour to angle in `z_steps`, a dircetion given by the vector `dir_v`, and with a scaling factor given by vector `scale_v`. `Contour`: a table of vectors as a closed contour (start point and end point is the same) `angle`: roation angle of contour along `z_steps` in deg `dir_v`: vector(x,y,z) direction of extrusion `sacle_v`: vector(x,y,z) scaling factors for extrudion `z_steps`: number of steps for the shape, mostly `z_steps=2` if angle is equal zero we will use it to extrude our 2d gear model to 3d with it.

- **Gear Formation Function**: For gear formation with this code we need to use `gear_formation ()` function.

In this function we can see that Rotor, Idler, Crescent, Gear housing will formed. For creating the Rotor gear we need to call `function gearProfile(z, m_n, alpha_t, x_coef , h_a_coef, h_f_coef, b)`. it will get `z` as number of teeth, `m_n` as module, `alpha_t` as pressure angle, `x_coef` as profile shift, `h_a_coef` as addendum height, `h_f_coef` as dedendum height, `b` as thickness of the gear. Now we have it on `rotor_gear` variable and we need to difference it from an cylinder to shape it as the rotor gear.

We also have `rot_rotor` variable for creating the rotation effect in the IceSL application. Then we start to emitting an cylinder for the the rotor gear base and then the rotor gear and also we set a color for them.

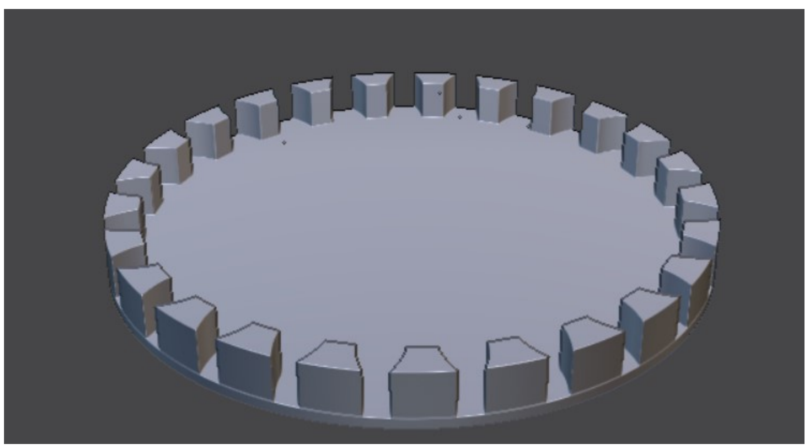

Figure 3 Rotor gear with base

```
local rotor_gear = gearProfile(z_2, m, alpha_t, x_coef_int, h_a_coef_p,
     h_f_coef_p, b);  -- Rotor gear
local rot_rotor = rotate(0, 0, rotation);  -- Rotation of rotor gear
emit(translate(0, 0, -0.1) * cylinder(r_a, 2), 1); -- Rotor gear Base
    formation
```

```
emit(rot_rotor * difference(extrude(circle(r_a - 0.1), 0, v(0, 0, b), v
    (1, 1, 1), 20), rotor_gear), 1); -- Rotor gear formation
set_brush_color(1, 0.3, 0.3, 0.4); -- Set color for rotor gear
```

We need two of the variables that has been created in rotor gear part for crescent and we will use them later.

```
local crescent_outer_radius = r_TIF;  -- Radius of the rotor gear
local crescent_root_radius = r_f;  -- Root radius of the rotor gear
```

For gear housing we need just a simple cylinder with emptied inlet and outlet from it. `gear_housing` variable is just for the emptied cylinder and `h` is for height of the inlet and outlet. We need to create an translate for inlet and outlet with `inlet_trans` and `outlet_trans`. now for creating the inlet we will create two cylinder ((`incyl1` and `incyl2`)) and one cube (`incube`) and difference it out of the `gear_housing` to create `inlet`. We have the same as inlet for `outcyl1`, `outcyl2`, `outcube`, `outlet`.

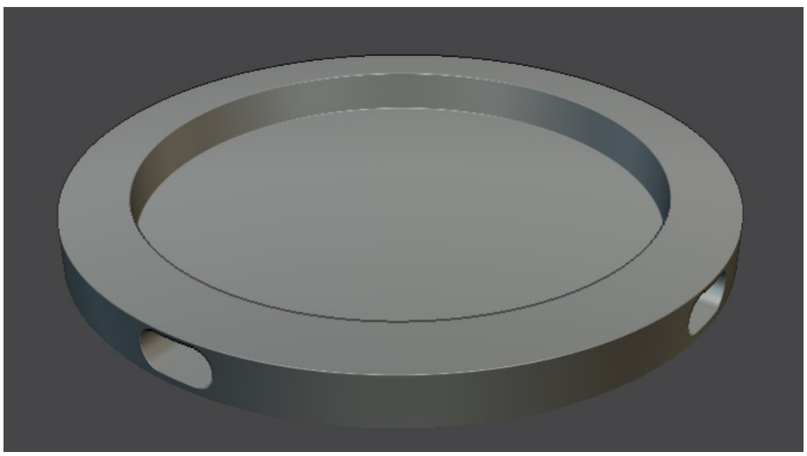

Figure 4 Gear housing

```
local gear_housing = difference(cylinder(r_a + 16, b), cylinder(r_a + 1
    .75, b)); -- Gear housing cylinder
local h = (z_2 * m) / 2 + m * 2 + 16;
local inlet_trans = rotate(45, Z) * translate(h, 0, b / 2 + 0.2) *
    rotate(270, Y); -- Inlet translate
local outlet_trans = rotate(45, Z) * translate(0, h, b / 2 + 0.2) *
    rotate(270, -X) * rotate(90, -Z); -- Outlet translate
```

```
local incyl1 = inlet_trans * cylinder(b / 2 - 1, h);  -- Inlet cylinder
  1
local incyl2 = inlet_trans * translate(0, b - 2, 0) * cylinder(b / 2 -
  1, h);  -- Inlet cylinder 2
local incube = inlet_trans * translate(0, b / 2 - 1, 0) * cube(b - 2, b
  - 1, h); -- Inlet cube
local inlet = union(union(incyl1,incyl2),incube); -- Inlet hole
local outcyl1 = outlet_trans * cylinder(b / 2 - 1, h);  -- Outlet
  cylinder 1
local outcyl2 = outlet_trans * translate(0, b - 2, 0) * cylinder(b / 2
  - 1, h);  -- Outlet cylinder 2
local outcube = outlet_trans * translate(0, b / 2 - 1, 0) * cube(b - 2,
  b - 1, h); -- Outlet cube
local outlet = union(union(outcyl1,outcyl2),outcube); -- Outlet hole
emit(intersection(difference(gear_housing, inlet), difference(
  gear_housing, outlet)), 0); -- Gear housing formation
emit(translate(0, 0, -0.5) * cylinder(r_a + 16, 0.5), 0); -- Base for
  Gear housing formation
```

Also we can add a front door that can connect to our crescent and make it hold still. It just need to have a hole for idler shaft and some holes for bolts.

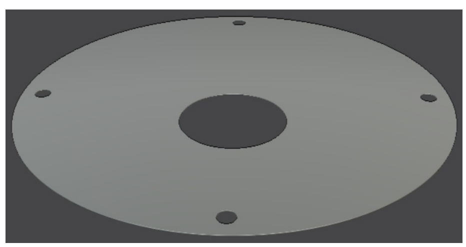

Figure 5 Idler gear

```
local front_door = translate(0, 0, b + 0.2) * cylinder(r_a + 16, 0.5);
  -- Front door for crescent
local hole_shaft = translate(0, meshDistance, b + 0.2) * extrude(circle
  (m * 4 + 0.5), 0, v(0, 0, 0.5), v(1, 1, 1), 20);  -- Hole of the
  idler shaft
local total_hole = (translate(r_a + 9, 0, -1) * cylinder(2.5, b + 5));
  -- Holes for bolts
for i = 360 / 4, 360, 360 / 4 do
  total_hole = union { total_hole, rotate(0, 0, i) * total_hole };
    -- Repeat for 4 holes
end
```

```
emit(intersection((difference(front_door, hole_shaft)), (difference(
    front_door, total_hole))), 0); --Front door formation
```

For the idler gear we have the same structure as rotor gear but we just add a shaft to it.

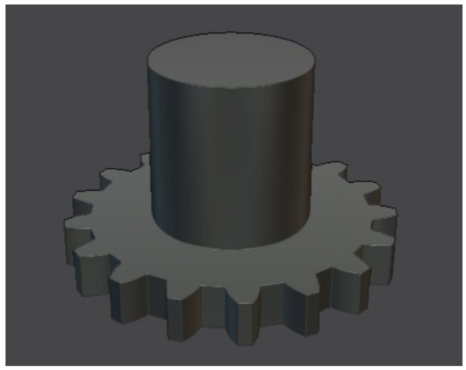

Figure 6 Idler gear

```
local idler_gear = gearProfile(z_1, m, alpha_t, x_coef_ext, h_a_coef_p,
     h_f_coef_p, b); -- Idler gear
local rot_idler = rotate(0, 0, rotation * z_2 / z_1); -- Rotation of
    idler gear
emit(translate(0, meshDistance, 0) * rot_idler * cylinder(2 + r_b / 2,
    r_b + 15), 2); -- Shaft formation
emit(translate(0, meshDistance, 0) * rot_idler * difference(idler_gear,
     extrude(circle(r_b / 2), 0, v(0, 0, b), v(1, 1, 1), 20)), 2); --
    Idler gear formation
set_brush_color(2, 0, 0, 0); -- Set color for idler gear
```

For the crescent we just need to difference the idler circle (`crescent_circle`) from the rotor circle (`ccylinder(crescent_outer_radius - c * 2, b)`) and remove the sharp edges (`crescent_cube_right` and `crescent_cube_left`).

```
local crescent_circle = translate(0, meshDistance, 0) * ccylinder(r_a +
     c * 2, b); -- Cylinder for crescent
local crescent_cube_right = translate(r_a + c, meshDistance + m * 2, 0)
     * cube(crescent_root_radius + c, crescent_root_radius + c, b + 0.1)
    ; -- Right cube to remove sharp edges of the crescent
local crescent_cube_left = translate(-(r_a), meshDistance + m * 2, 0) *
     cube(crescent_root_radius + c, crescent_root_radius + c, b + 0.1);
     -- Left cube to remove sharp edges of the crescent
```

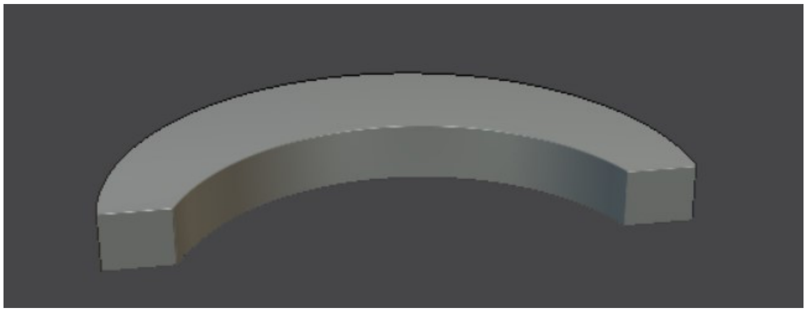

Figure 7 Crescent

```
local crescent_main = translate(0, 0, b / 2 + 0.1) * difference(
    ccylinder(crescent_outer_radius - c * 2, b), crescent_circle); --
    Crescent with sharp edges
emit(intersection(difference(crescent_main, crescent_cube_right),
    difference(crescent_main, crescent_cube_left)), 0); -- Crescent
    formation without sharp edges
set_brush_color(0, 0.2, 0.2, 0.2); -- Set color for crescent and gear
    housing
```

At the end when we assemble them, we have something like the shape below.

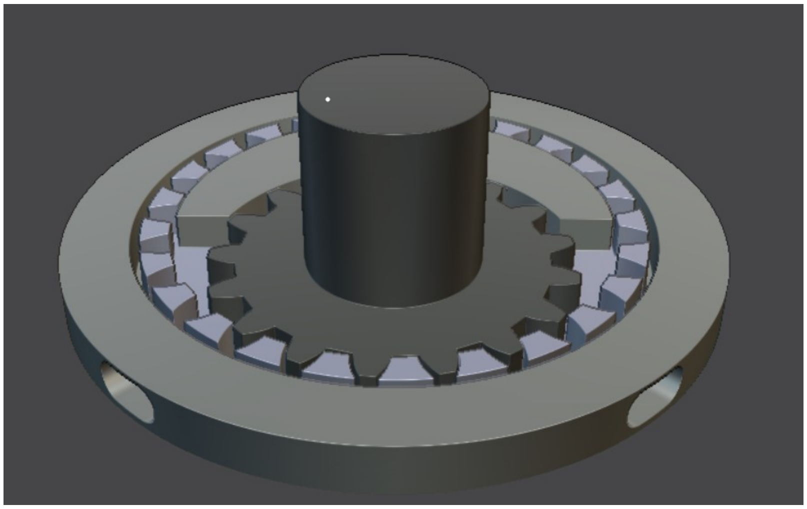

Figure 8 Internal gear pump

3 Slicing

We can use IceSL-slicer to export stl or gcode files. For stl file we export to mesh.

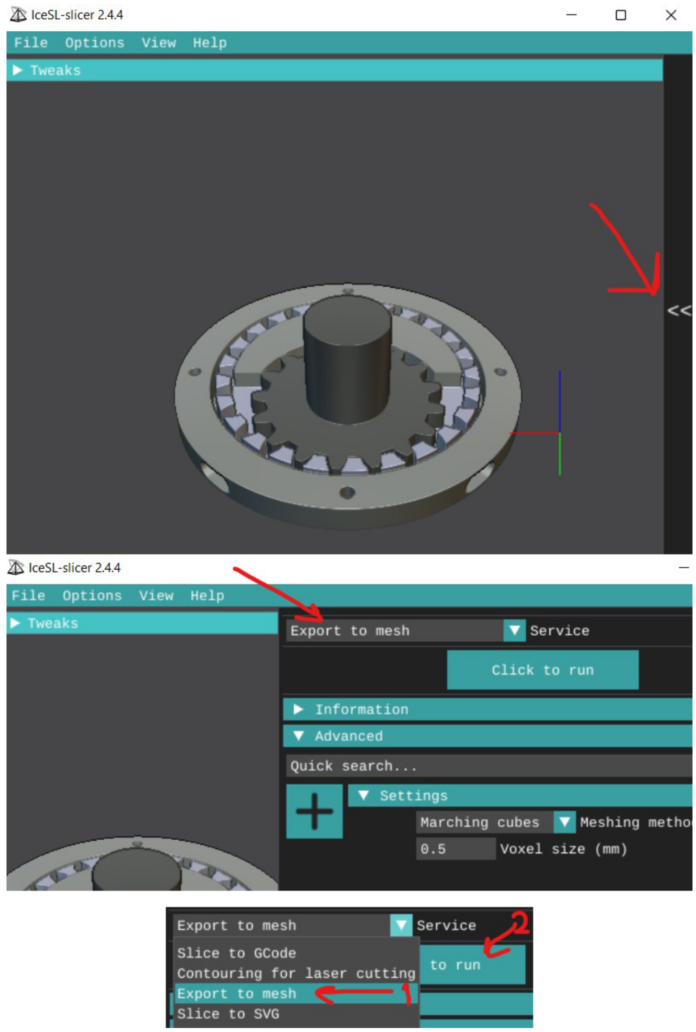

Figure 9 Slicing

4 Assembling

The final parts can 3d printed in 4 parts and assemble together like as the picture below. After that the user can put this in a sealed chamber and connect inlet and outlet to the relevant pipes and by rotating the shaft the pump will start the suction. For better fastening, we installed a door attached to the crescent with 4 bolts holes.

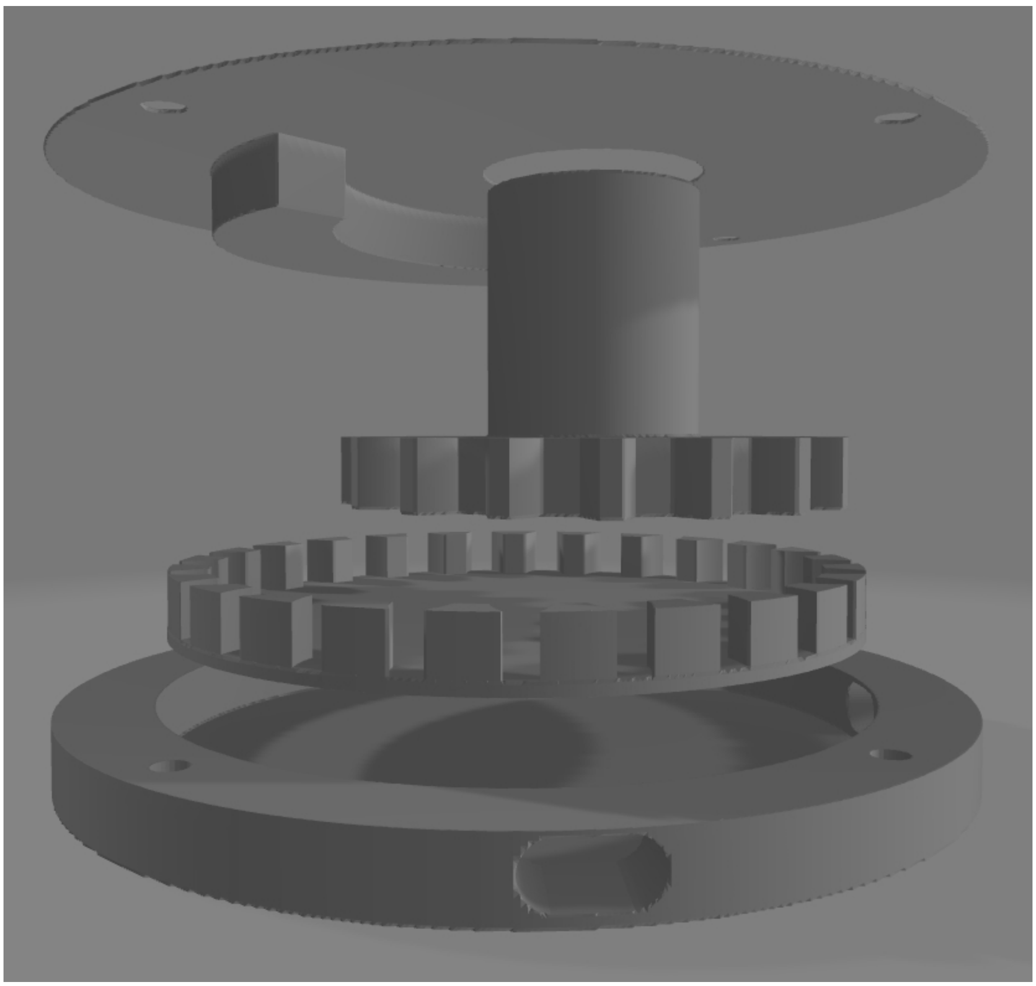

Figure 10 Assembling

References

[1] Kadkhodaei Elyaderani A. Ataei M. *INTERNAL GEAR PUMP*. Deggendorf Institute of Technology, 2022.

[2] Harry H. Cheng. *Derivation of the Explicit Solution of the Inverse Involute Function and its Application in gear Tooth Geometry*. Journal of Applied Mechanisms and Robotics, 1996.