Inhaltsverzeichnis

Prozeduale CAD- Modelle erstellen mit CadQuery

CadQuery ist eine Python- Bibliothek, die auf dem Datenmodell von FreeCAD aufsetzt und es ermöglicht, mit Python 3D- CAD- Modelle zu erzeugen.

Jedenfalls in der Theorie.. In der Praxis sind selbst scheinbar simple Operationen ein tagelanges Try & Error, um was man aber nicht wirklich umrumkommt, wenn man z.B. aus Parameterlisten komplexe Bohrlochplatten machen will.

So entstand unter Schmerzen letztlich dieses Programm (im QC-Editor), was Bohrlöcher anhand einer Parameterliste setzt:

pruefmodul.py

```
import cadquery as cq

plate_width=200.0  /1000.0
plate_height=150.0 /1000.0
thickness=30.0 /1000.0

m3_hole = 3.5
m3_head = 5.5
m3_depth = 20.0

connectors={
    # this is a kind of library for the different connector footprints
    "MGW":[ # each connector has one or more areas of a pin grid
        { #where each area has a position inside the connector
            # its relative position inside the connector
            "x":0.1,
            "y":0.1,
            # and a set of column dimensions, relative to each other
            "cols":[6.5,8.5, 8.5],
            #the related row dimensions
            "rows":[7.45, 9.9, 18.2],
            #""rows":[7.45, 9.9, 8.2],
            # and then finally the property for each pin
            # the hole diameter, the borehole diameter,
            # the borehole depth, the long hole length,
            # the long hole diameter and the long hole orientation
            # if long hole diameter or length is 0, the hole is not
made
            "pins":[
            # first row
            [m3_hole, m3_head, m3_depth, 3.0, 3.0, "v"],
            [m3_hole, m3_head, m3_depth, 6.5, 3.0, "v"],
            [m3_hole, m3_head, m3_depth, 3.0, 3.0, "v"],
```

```
            # second row
            [m3_hole, m3_head, m3_depth, 6.5, 3.0, "v"],
            [m3_hole, m3_head, m3_depth, 6.5, 3.0, "h"],
            [m3_hole, m3_head, m3_depth, 6.5, 3.0, "v"],
            # third row
            [m3_hole, m3_head, m3_depth, 3.0, 3.0, "v"],
            [m3_hole, m3_head, m3_depth, 8.0, 3.0, "h"],
            [m3_hole, m3_head, m3_depth, 3.0, 3.0, "v"],
            ]
        }
    ]

}

plate=[ # a number of connectors
       { #each connector consists of
        "x":20.0,
        "y":0.0,
        "type":"MGW"
        },
           { #each connector consists of
        "x":100.0,
        "y":0.0,
        "type":"MGW"
       },
]

x_offset=-plate_height/2
y_offset=-plate_width/2
# make the base
print("box:",plate_height, plate_width , thickness)
result = cq.Workplane("XY").box(plate_height, plate_width , thickness)

for single_connector in plate:
    connector=connectors[single_connector["type"]]
    connector_x=single_connector["x"] /1000
    connector_y=single_connector["y"] /1000
    for area in connector:
        area_x=area["x"] /1000 + connector_x
        area_y=area["y"] /1000 + connector_y
        print("area",area_x,area_y)
        rows=area["rows"]
        cols=area["cols"]
        row_count=len(rows)
        col_count=len(cols)
        #row_pos=area_x
        row_pos=area_y
        for row_index in range(row_count):
            row_pos += (rows[row_index] /1000)
            #col_pos = area_y
            col_pos = area_x
```

```
            for col_index in range(col_count):
                col_pos += cols[col_index]/1000
                pin_index= row_index * col_count + col_index
                print("pin pos", pin_index, row_pos, col_pos)
                pin=area["pins"][pin_index]
                print(pin)
                pin_hole_size=pin[0] /1000 /2 # diameter -> radius
                pin_screw_size=pin[1] /1000 /2 # diameter -> radius
                pin_screw_depth=pin[2] /1000
                pin_slot_length=pin[3] /1000
                pin_slot_width=pin[4] /1000
                pin_type=pin[5]

                # aus purer verzweiflung: Vertauschen von x und y :-(
                x_pos, y_pos= row_pos, col_pos


                #wp.center(row_pos,col_pos).hole(5.0)
                result = result.workplane(offset=0,
centerOption="CenterOfBoundBox").center(x_pos + x_offset, y_pos
+y_offset ).circle(pin_hole_size).cutThruAll()
                result = result.workplane(offset=0,
centerOption="CenterOfBoundBox").center(x_pos + x_offset, y_pos
+y_offset ).circle(pin_screw_size).cutBlind(pin_screw_depth)
                if pin_slot_length> 0 and pin_slot_width>0:
                    if pin_type.lower()=="v":
                        result = result.workplane(offset=0,
centerOption="CenterOfBoundBox").center(x_pos + x_offset, y_pos
+y_offset ).move( -pin_slot_length,-
pin_slot_width/2).lineTo(pin_slot_length,-
pin_slot_width/2).lineTo(pin_slot_length,pin_slot_width/2).lineTo(-
pin_slot_length,pin_slot_width/2).close().cutBlind(pin_screw_depth)
                        result = result.workplane(offset=0,
centerOption="CenterOfBoundBox").center(x_pos + x_offset +
pin_slot_length, y_pos  +y_offset ).circle(pin_slot_width / 1.99
).cutBlind(pin_screw_depth)
                        result = result.workplane(offset=0,
centerOption="CenterOfBoundBox").center(x_pos + x_offset -
pin_slot_length, y_pos  +y_offset ).circle(pin_slot_width / 1.99
).cutBlind(pin_screw_depth)
                    else:
                        result = result.workplane(offset=0,
centerOption="CenterOfBoundBox").center(x_pos + x_offset, y_pos
+y_offset ).move(-pin_slot_width/2, -pin_slot_length).lineTo(-
pin_slot_width/2,pin_slot_length).lineTo(pin_slot_width/2,pin_slot_leng
th).lineTo(pin_slot_width/2,-
pin_slot_length).close().cutBlind(pin_screw_depth)
                        result = result.workplane(offset=0,
centerOption="CenterOfBoundBox").center(x_pos + x_offset , y_pos
+y_offset + pin_slot_length).circle(pin_slot_width / 1.99
).cutBlind(pin_screw_depth)
```

```
                        result = result.workplane(offset=0,
centerOption="CenterOfBoundBox").center(x_pos + x_offset , y_pos
+y_offset - pin_slot_length).circle(pin_slot_width / 1.99
).cutBlind(pin_screw_depth)



# Render the solid
show_object(result)
cq.exporters.export(result,'/media/ram/result.stl')
cq.exporters.export(result.section(),'/media/ram/result.dxf')
```