Summer Fellowship Report

On

Creating CAD modules for OSDAG section modeller with pythonOCC

Submitted by

Mosam Patel

Under the guidance of

Prof.Sidhartha Ghosh
Civil Engineering Department
IIT Bombay

Under the Mentorship of

Anand Swaroop
Project Research Associate

June 26, 2020

Acknowledgment

I would like to thank FOSSEE for providing me a platform to work on something I am very interested in. I am thankful to everyone who thought of having and involved in selection process based on screening tasks. I am grateful to be a part of team which promotes open source software.

I thank all the Osdag members, who are wonderful mentors and great team. I thank Anand Swaroop (Project Research Associate), Danish Ansari (Project Research Assistant), Deepti Reddy(Project Research Associate), Sourabh Das (Project Research Associate), Ajmal Babu MS (Project Research Associate), Yash Lokhande (Project Research Assistant), Darshan Viswakarma (Project Research Associate), Anjali Jatav (Project Research Assistant) and whole team, who made us feel welcome and planned all the tasks meticulously during this period.

I am grateful that I got a chance to work under Prof. Sidharth Ghosh, who took time to mentor us and monitored individual contributions as well.

Contents

Chapter 1

Introduction

1.1 Osdag Internship

Osdag internship is provided under the FOSSEE project. FOSSEE project promotes the use of FOSS (Free/Libre and Open Source Software) tools to improve quality of education in our country. FOSSEE encourages the use of FOSS tools through various activities to ensure availability of competent free software equivalent to commercial (paid) softwares.
The FOSSEE project is a part of the National Mission on Education through Infrastructure and Communication Technology(ICT), Ministry of Human Resources and Development, Government of India.
Osdag is one such open source software which comes under the FOSSEE project. Osdag internship is provided through FOSSEE project. Any UG/PG/PhD holder can apply for this internship. And the selection will be based on a screening task.

1.2 What is Osdag?

Osdag is Free/Libre and Open Source Software being developed for design of steel structures. Its source code is written in Python, 3D CAD images are developed using PythonOCC. Github is used to ensure smooth workflow between different modules and team members. It is in a path where people from around the world would be able to contribute to its development. FOSSEE's "Share alike" policy would improve the standard of the software when the source code is further modified based on the industrial and educational needs across the country.

Design and Detailing Checklist (DDCL) for different connections, members and structure designs is one of the important bi-products of this project. It would create a repository and design guide book for steel construction based on Indian Standard codes and best industry practices.

1.3 Who can use ?

Osdag is created both for educational purpose and industry professionals. As Osdag is currently funded by MHRD, Osdag team is developing software in such a way that it can be used by the students during their academics and to give them a better insight look in the subject.
Osdag can be used by anyone starting from novice to professionals. It's simple user interface makes it flexible and attractive than other software. Video tutorials are available to help get started. The video tutorials of Osdag can be accessed here.

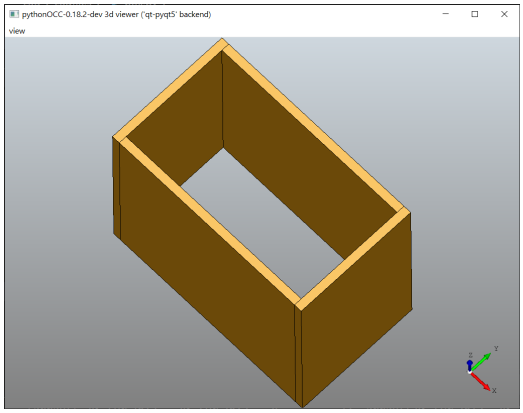
Chapter 2

Creating CAD modules

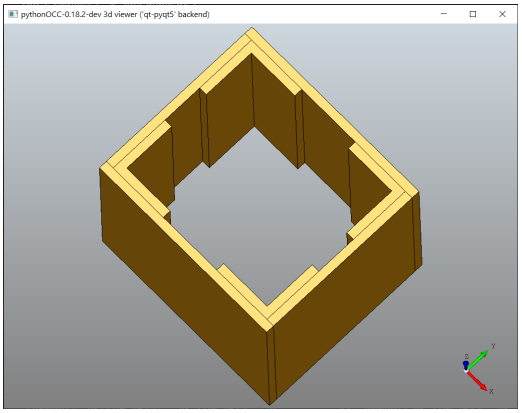
I was given a specification for several CAD modules and from that i have to create CAD modules. I have created all the CAD modules from the given specs using pythonocc.

2.1 Creating CAD modules for section modeller

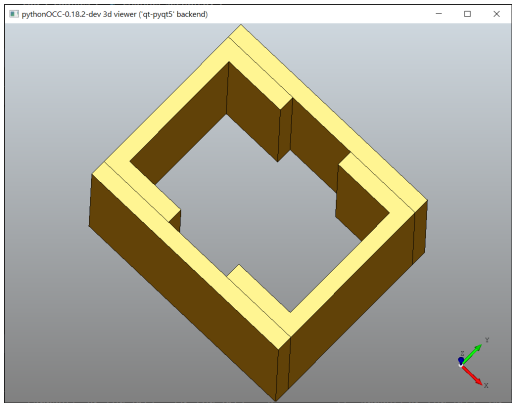
There are already some basic CAD modules are available in CAD.items folder like Isection, channel, angle, plate, nut etc. I have used those premade modules to create now modules for section modeller. There are around twelve CAD modules are created for section modeller which includes channel section (side-side & back-back), isection coverplate (side-side), box, box angle section, cross isection, bulid up section, angle section (4 angle, 2 angle, same side, opposite side), compound section ect. each modules are created in the same way that previous modules are created. they all consisit main three methodes (place, calculate_params, create_model). I have included the images of all the designed CAD modules in fig. 2.1.


pythonOCC-0.18.2-dev 3d viewer ('qt-pyqt5' backend)
view

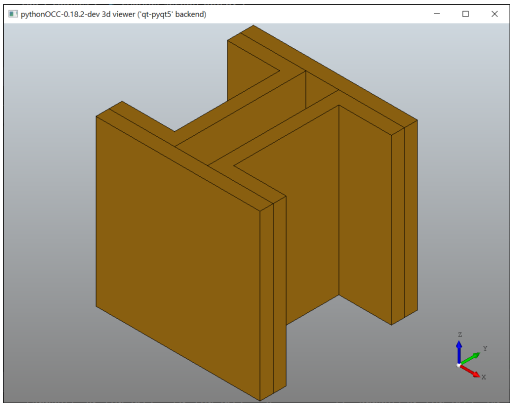
(a)


pythonOCC-0.18.2-dev 3d viewer ('qt-pyqt5' backend)

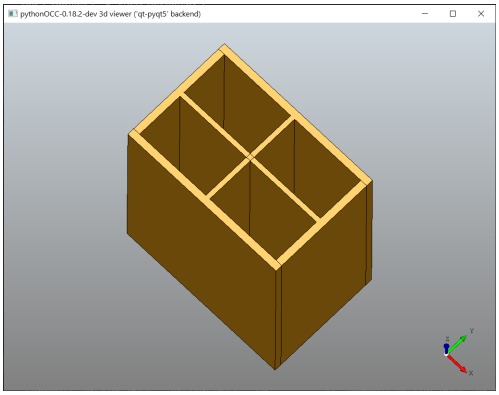
(b)


pythonOCC-0.18.2-dev 3d viewer ('qt-pyqt5' backend)

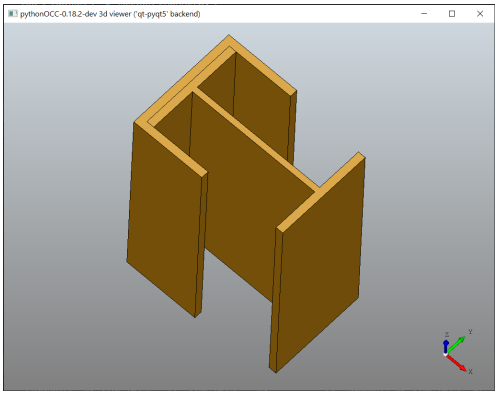
(c)


pythonOCC-0.18.2-dev 3d viewer ('qt-pyqt5' backend)

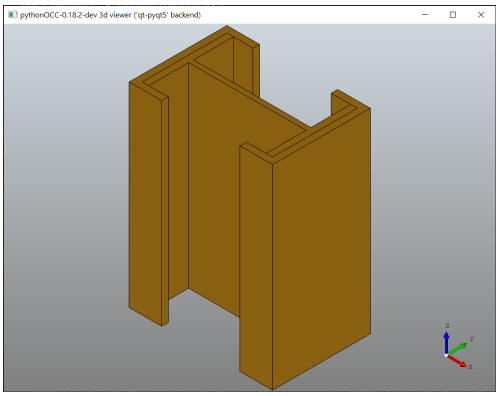
(d)


pythonOCC-0.18.2-dev 3d viewer ('qt-pyqt5' backend)

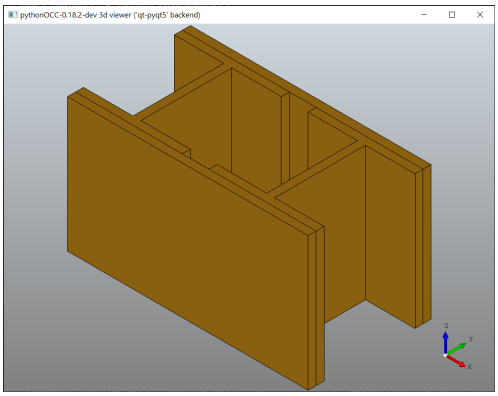
(e)


pythonOCC-0.18.2-dev 3d viewer ('qt-pyqt5' backend)

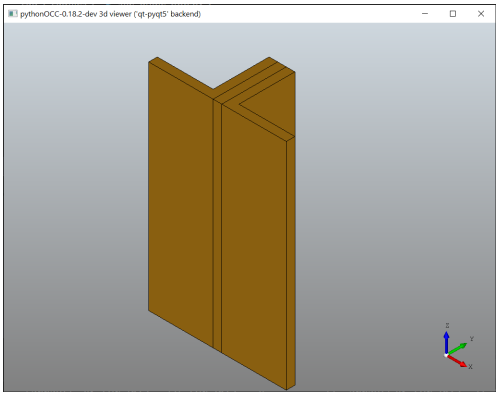
(f)



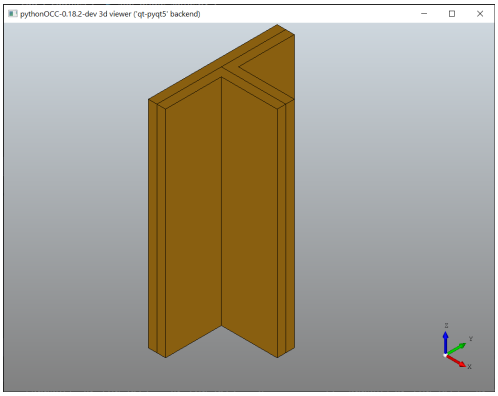
(g)



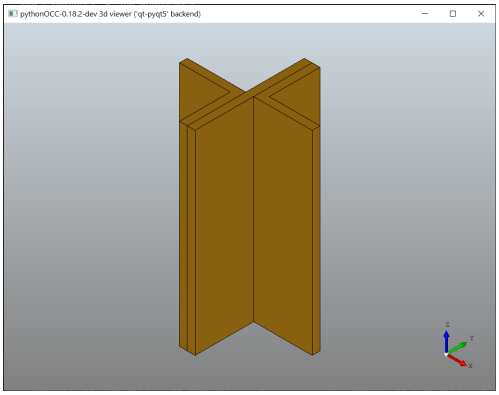
(h)



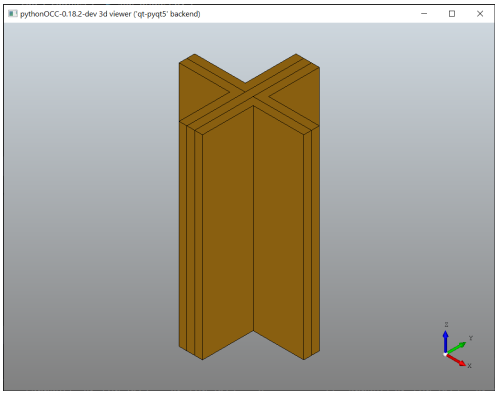
(i)



(j)



(k)



(l)

Figure 2.-4: CAD modules images for section modeller

2.2 Creating hollow section CAD modules

I have also created several hollow section modules from given specification like, square hollow section, rectangular hollow section and circular hollow section. Fig. 2.1, 2.3 and 2.3 shows the designed circular, rectangle and square hollow section modules respectively.

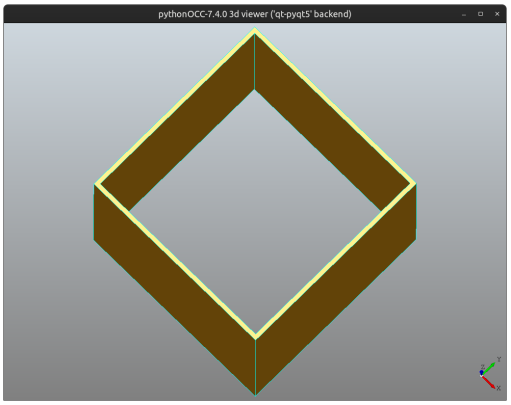


Figure 2.-3: square hollow section

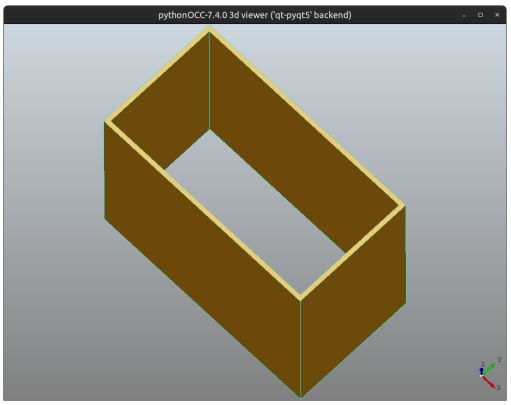


Figure 2.-2: rectangle hollow section

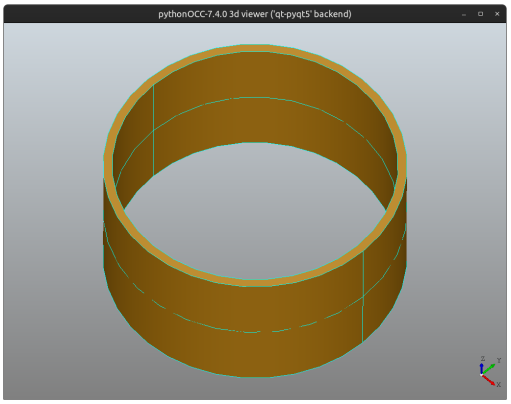
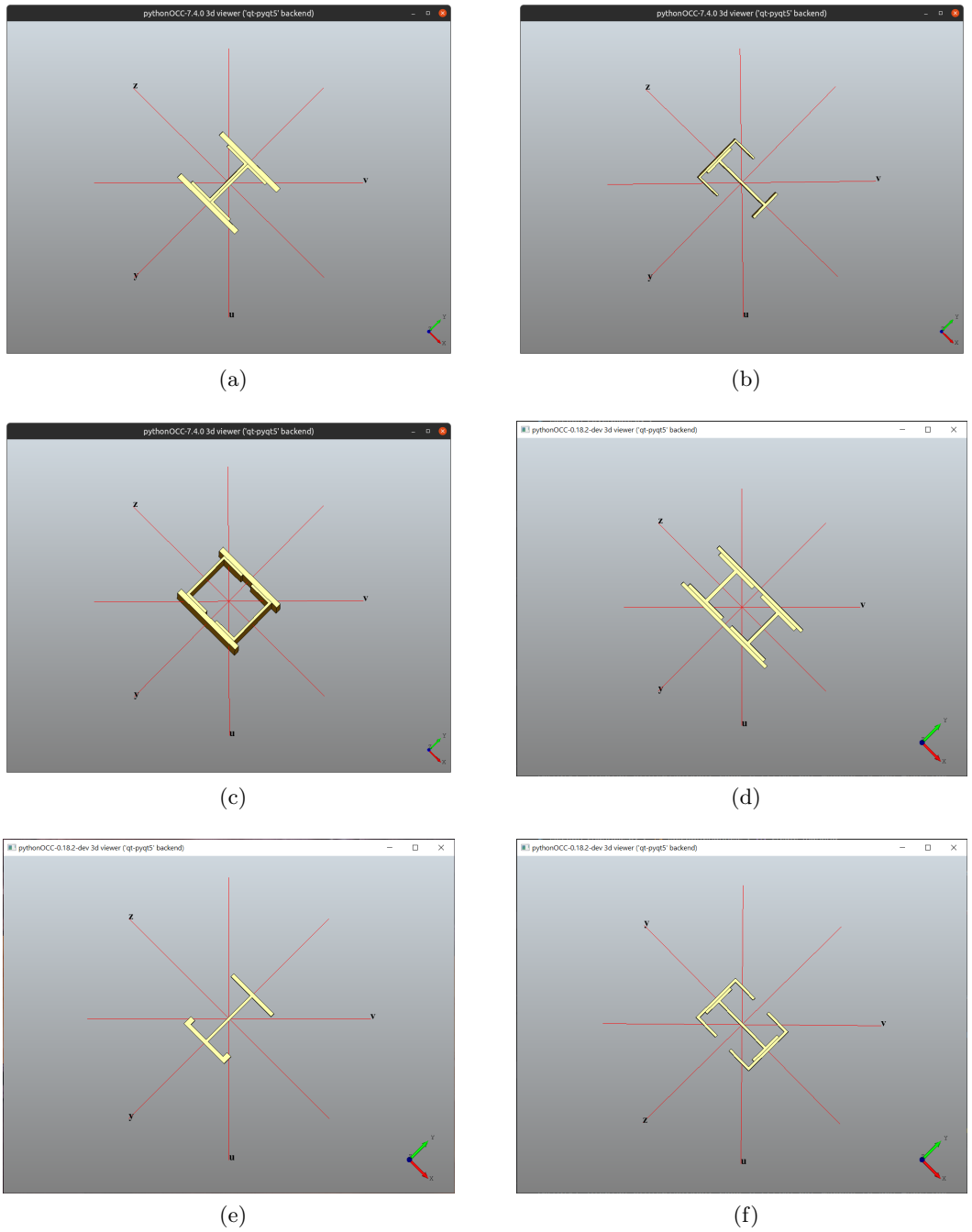


Figure 2.-1: circular hollow section

2.3 Creating marking and labeling for CAD modules

I have detailed and labeled all newly created modules with major and minor axis marking. this includes two major axis y and z and two minor axis u and v. I have included the some figures indicating the detailing of the CAD modules.



(a) (b)

(c) (d)

(e) (f)

Figure 2.0: axis marking example figures

2.4 Drawing ASCII diagrams for CAD files

I have drawn ASCII diagram for some CAD modules. ASCII diagram is a kind of documented diagram made up with some characters like ., /, —, -, _, + etc. By looking at the ASCII diagram one can understand the CAD module code.

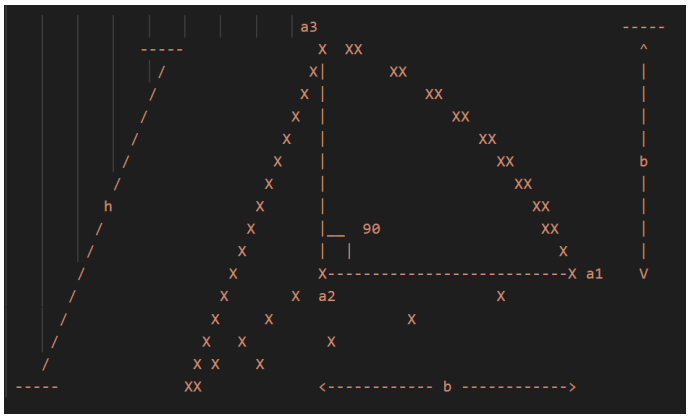


Figure 2.1: ASCII diagram of quarter cone module

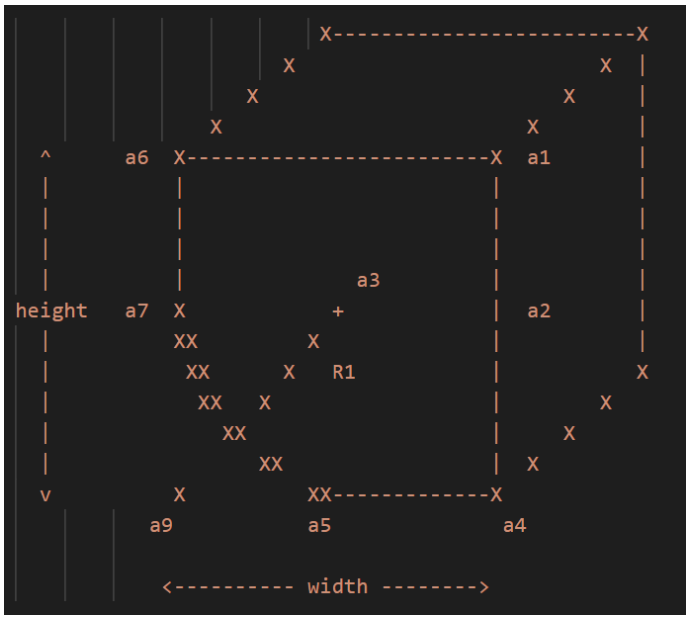


Figure 2.2: ASCII diagram of notch module

2.5 Making CAD files to run independently

I have also made CAD code files run independently. So, by just executing require CAD code file separately one can easily see output of the CAD module in the form of OCC viewer.

Chapter 3

Converting 3D CAD modules to 2D

In this task I have to figure out a way to convert 3D CAD shape into 2D, That means i have to project given 3D shape into XY, YZ, and ZX plane. In other words I have to take top view, front view and side view from given 3D shape. I solved this problem by writing a script that can take 3D shape as input and gives three 2D plane (XY, YZ, ZX) as an output. I included the Isection module example where fig. 3.1 shows 3D view of Isection and fig. 3.2, 3.3 and 3.4 shows top view, front view and side view of Isection respectively.

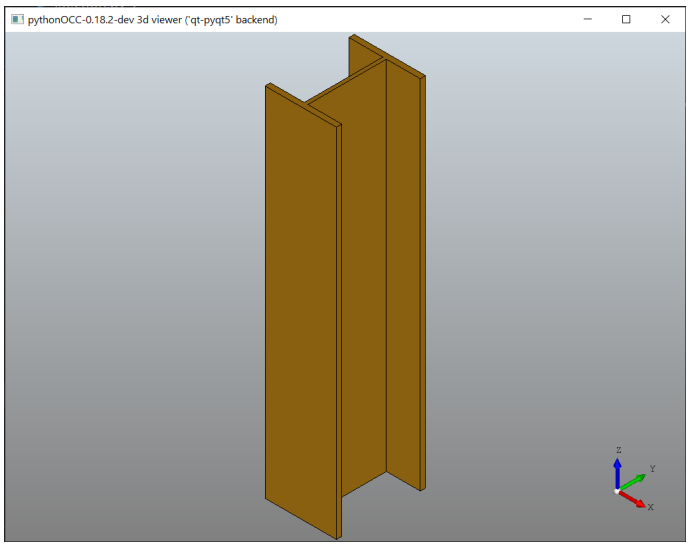


Figure 3.1: Isection 3D module

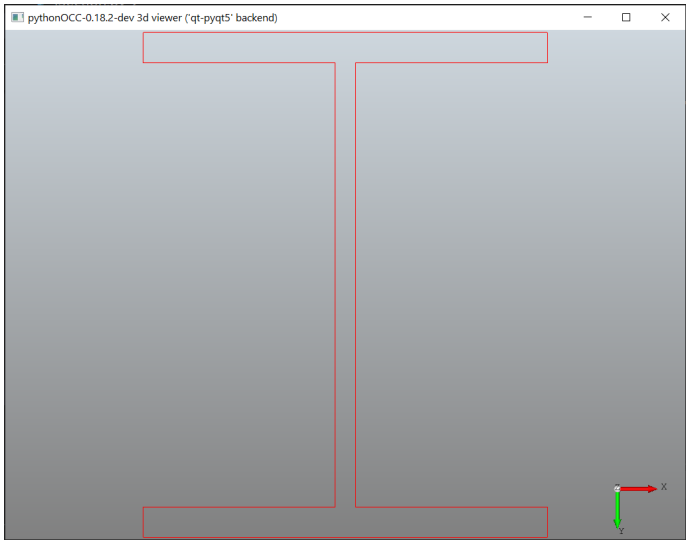


Figure 3.2: Top view of Isection module

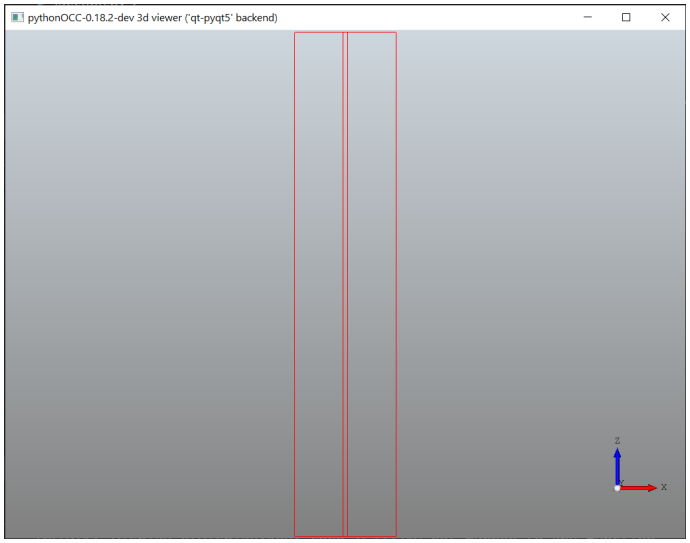


Figure 3.3: Front view of Isection module

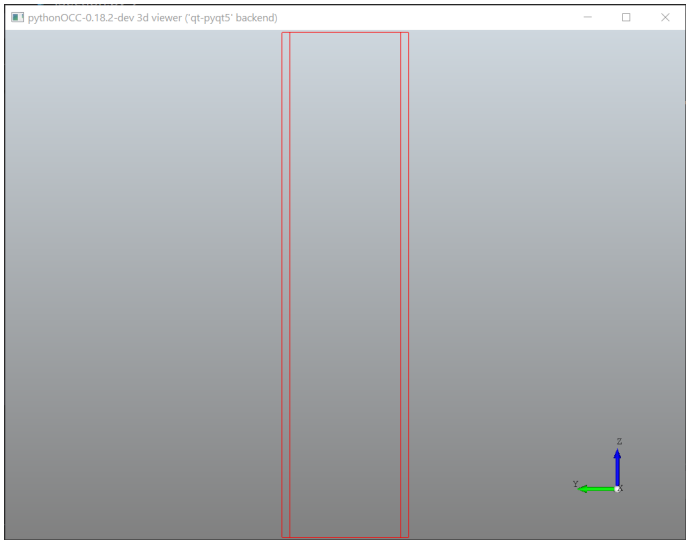


Figure 3.4: side view of Isection module

Appendices

Appendix A

Code of the CAD modules for section modeller

box hollow section

```
import numpy
from cad1.items.ModelUtils import *
from OCC.Core.BRepAlgoAPI import BRepAlgoAPI_Fuse

from anglebar import Angle

from cad1.items.plate import Plate

class Box(object):
    def __init__(self, A, B, t, H, s, s1):
        self.A = A
        self.B = B
        self.H = H
        self.t = t
        self.s = s
        self.B = s + t
        self.s1 = s1
        self.A = s1 + t

        self.sec_origin = numpy.array([0, 0, 0])
        self.uDir = numpy.array([1.0, 0, 0])
        self.wDir = numpy.array([0.0, 0, 1.0])
        self.vDir = self.wDir * self.uDir

        self.plate1 = Plate(self.B, H, t)
        self.plate2 = Plate(t, H, self.A)
        self.plate3 = Plate(self.B, H, t)
        self.plate4 = Plate(t, H, self.A)

    def place(self, secOrigin, uDir, wDir):
        self.sec_origin = secOrigin
        self.uDir = uDir
        self.wDir = wDir
        origin5 = numpy.array([0.,0.,0.])
        origin5 = numpy.array([-self.A/2-self.t/2, 0., 0.])
        self.plate1.place(origin5, self.uDir, self.wDir)
```

```
37          origin6 = numpy.array([0., -self.B/2+self.t/2., 0.])
38          self.plate2.place(origin6, self.uDir, self.wDir)
39          origin7 = numpy.array([self.A/2+self.t/2., 0., 0.])
40          self.plate3.place(origin7, self.uDir, self.wDir)
41          origin8 = numpy.array([0., self.B/2-self.t/2, 0.])
42          self.plate4.place(origin8, self.uDir, self.wDir)
43
44      def compute_params(self):
45          self.plate1.compute_params()
46          self.plate2.compute_params()
47          self.plate3.compute_params()
48          self.plate4.compute_params()
49
50      def create_model(self):
51          prism1 = self.plate1.create_model()
52          prism2 = self.plate2.create_model()
53          prism3 = self.plate3.create_model()
54          prism4 = self.plate4.create_model()
55
56          prism = BRepAlgoAPI_Fuse(prism1, prism2).Shape()
57          prism = BRepAlgoAPI_Fuse(prism, prism3).Shape()
58          prism = BRepAlgoAPI_Fuse(prism, prism4).Shape()
59          return prism
60
61      def create_marking(self):
62          middel_pnt = []
63          line = []
64          labels = ["z","y","u","v"]
65          offset = 100
66          uvoffset = offset/numpy.sqrt(2)
67
68          z_points = [numpy.array([-offset,0.,self.H/2]),
            ↪  numpy.array([offset,0.,self.H/2])]
69          line.append(makeEdgesFromPoints(z_points))
70
71          y_points = [numpy.array([0.,-offset,self.H/2]),
            ↪  numpy.array([0,offset,self.H/2])]
72          line.append(makeEdgesFromPoints(y_points))
73
74          u_points = [numpy.array([-uvoffset,uvoffset,self.H/2]),
            ↪  numpy.array([uvoffset,-uvoffset,self.H/2])]
75          line.append(makeEdgesFromPoints(u_points))
76
77          v_points = [numpy.array([-uvoffset,-uvoffset,self.H/2]),
            ↪  numpy.array([uvoffset,uvoffset,self.H/2])]
78          line.append(makeEdgesFromPoints(v_points))
79
80          middel_pnt =
            ↪  [[-offset,0,self.H/2],[0,-offset,self.H/2],[uvoffset,-uvoffset,self.H/2],[uvoffset,u
81
82          return line, middel_pnt, labels
83
84
85  if __name__ == '__main__':
86
87      from OCC.Display.SimpleGui import init_display
```

```
    display, start_display, add_menu, add_function_to_menu = init_display()

    def display_lines(lines, points, labels):
        for l,p,n in zip(lines,points, labels):
            display.DisplayShape(l, update=True)
            display.DisplayMessage(getGpPt(p), n,message_color=(0,0,0))

    def view_bottom(event=None):
        display.View_Bottom()
        display.FitAll()

    def view_front(event=None):
        display.View_Front()
        display.FitAll()

    def view_left(event=None):
        display.View_Left()
        display.FitAll()

    def view_right(event=None):
        display.View_Right()
        display.FitAll()

    add_menu('view')
    add_function_to_menu('view',view_bottom)
    add_function_to_menu('view',view_front)
    add_function_to_menu('view',view_left)
    add_function_to_menu('view',view_right)

    A = 50
    B = 30
    H = 50
    t = 2
    s = 30
    s1 = 50


    origin = numpy.array([0.,0.,0.])
    uDir = numpy.array([1.,0.,0.])
    wDir = numpy.array([0.,0.,1.])

    box = Box(A, B, t, H, s, s1)
    _place = box.place(origin, uDir, wDir)
    point = box.compute_params()
    prism = box.create_model()
    lines, m_pnt, labels = box.create_marking()
    display.DisplayShape(prism, update=True)
    display_lines(lines, m_pnt, labels)
    display.DisableAntiAliasing()
    start_display()
```

box angle section

```
import numpy
from cad1.items.ModelUtils import *
```

```
from OCC.Core.BRepAlgoAPI import BRepAlgoAPI_Fuse
from anglebar import Angle
from cad1.items.plate import Plate

class BoxAngle(object):
    def __init__(self, a, b, t, l, t1, l1, H, s, s1):
        self.l = l
        self.a = a
        self.b = b
        self.t = t
        self.s = s
        self.s1 = s1
        self.t1 = t1
        self.l1 = l1
        self.H = H

        self.sec_origin = numpy.array([0, 0, 0])
        self.uDir = numpy.array([1.0, 0, 0])
        self.wDir = numpy.array([0.0, 0, 1.0])
        self.vDir = self.wDir * self.uDir

        self.angle1 = Angle(H, a, b, t, 0, 0)
        self.angle2 = Angle(H, b, a, t, 0, 0)
        self.angle3 = Angle(H, a, b, t, 0, 0)
        self.angle4 = Angle(H, b, a, t, 0, 0)
        self.plate1 = Plate(l, H, t1)
        self.plate2 = Plate(t1, H, l1)
        self.plate3 = Plate(l, H, t1)
        self.plate4 = Plate(t1, H, l1)

    def place(self, secOrigin, uDir, wDir):
        self.sec_origin = secOrigin
        self.uDir = uDir
        self.wDir = wDir
        verti = -self.t - self.s1/2#-self.s1/2 - self.b
        hori = -self.t - self.s/2
        t = self.t1/2

        origin1 = numpy.array([verti, hori, 0.])
        self.angle1.place(origin1, self.uDir, self.wDir)
        origin2 = numpy.array([hori, verti, 0.])
        self.angle2.place(origin2, self.uDir, self.wDir)
        origin3 = numpy.array([verti, hori, 0.])
        self.angle3.place(origin3, self.uDir, self.wDir)
        origin4 = numpy.array([hori, verti, 0.])
        self.angle4.place(origin4, self.uDir, self.wDir)

        origin5 = numpy.array([-self.l1/2+t, 0., 0.])
        self.plate1.place(origin5, self.uDir, self.wDir)
        origin6 = numpy.array([0., -self.l/2-t, 0.])
        self.plate2.place(origin6, self.uDir, self.wDir)
        origin7 = numpy.array([self.l1/2-t, 0., 0.])
        self.plate3.place(origin7, self.uDir, self.wDir)
        origin8 = numpy.array([0., self.l/2+t, 0.])
        self.plate4.place(origin8, self.uDir, self.wDir)

```

```
59      def compute_params(self):
60          self.angle1.computeParams()
61          self.angle2.computeParams()
62
63          self.angle2.points = self.rotate(self.angle2.points, numpy.pi/2)
64          self.angle3.computeParams()
65          self.angle3.points = self.rotate(self.angle3.points, numpy.pi)
66          self.angle4.computeParams()
67          self.angle4.points = self.rotate(self.angle4.points, 3*numpy.pi/2)
68
69          self.plate1.compute_params()
70          self.plate2.compute_params()
71          self.plate3.compute_params()
72          self.plate4.compute_params()
73
74      def create_model(self):
75          prism1 = self.angle1.create_model()
76          prism2 = self.angle2.create_model()
77          prism3 = self.angle3.create_model()
78          prism4 = self.angle4.create_model()
79
80
81          prism5 = self.plate1.create_model()
82          prism6 = self.plate2.create_model()
83          prism7 = self.plate3.create_model()
84          prism8 = self.plate4.create_model()
85
86          prism = BRepAlgoAPI_Fuse(prism1, prism2).Shape()
87          prism = BRepAlgoAPI_Fuse(prism, prism3).Shape()
88          prism = BRepAlgoAPI_Fuse(prism, prism4).Shape()
89          prism = BRepAlgoAPI_Fuse(prism, prism5).Shape()
90          prism = BRepAlgoAPI_Fuse(prism, prism6).Shape()
91          prism = BRepAlgoAPI_Fuse(prism, prism7).Shape()
92          prism = BRepAlgoAPI_Fuse(prism, prism8).Shape()
93          return prism
94
95      def rotate(self, points, x):
96          rotated_points = []
97          rmatrix = numpy.array([[numpy.cos(x), -numpy.sin(x), 0],
98                                 [numpy.sin(x), numpy.cos(x), 0],
99                                 [0, 0, 1]])
100         for point in points:
101             point = numpy.matmul(rmatrix, point)
102             rotated_points.append(point)
103         return rotated_points
104
105     def create_marking(self):
106         middel_pnt = []
107         line = []
108         labels = ["z","y","u","v"]
109         offset = 100
110         uvoffset = offset/numpy.sqrt(2)
111
112         z_points = [numpy.array([-offset,0.,self.H/2]),
            ↪  numpy.array([offset,0.,self.H/2])]
113         line.append(makeEdgesFromPoints(z_points))
```

```
114
115         y_points = [numpy.array([0.,-offset,self.H/2]),
            ↪  numpy.array([0,offset,self.H/2])]
116         line.append(makeEdgesFromPoints(y_points))
117
118         u_points = [numpy.array([-uvoffset,uvoffset,self.H/2]),
            ↪  numpy.array([uvoffset,-uvoffset,self.H/2])]
119         line.append(makeEdgesFromPoints(u_points))
120
121         v_points = [numpy.array([-uvoffset,-uvoffset,self.H/2]),
            ↪  numpy.array([uvoffset,uvoffset,self.H/2])]
122         line.append(makeEdgesFromPoints(v_points))
123
124         middel_pnt =
            ↪  [[-offset,0,self.H/2],[0,-offset,self.H/2],[uvoffset,-uvoffset,self.H/2],[uvoffset,u
125
126         return line, middel_pnt, labels
127
128
129 if __name__ == '__main__':
130
131     from OCC.Display.SimpleGui import init_display
132     display, start_display, add_menu, add_function_to_menu = init_display()
133
134     def display_lines(lines, points, labels):
135         for l,p,n in zip(lines,points, labels):
136             display.DisplayShape(l, update=True)
137             display.DisplayMessage(getGpPt(p), n,message_color=(0,0,0))
138
139     #def view_markings():
140      #   display_lines(lines, m_pnt, labels)
141
142     #add_menu('view')
143     #add_function_to_menu('view',view_markings)
144
145     l = 40
146     l1 = 50
147     a = 15
148     b = 15
149     t = 2
150     t1 = 2
151     s = l  - 2*t1
152     s1 = l1 - 2*t1 - 2*t
153     H = 50
154
155
156     origin = numpy.array([0.,0.,0.])
157     uDir = numpy.array([1.,0.,0.])
158     wDir = numpy.array([0.,0.,1.])
159
160     box_angle = BoxAngle(a, b, t, l, t1, l1, H, s, s1)
161     _place = box_angle.place(origin, uDir, wDir)
162     point = box_angle.compute_params()
163     prism = box_angle.create_model()
164     lines, m_pnt, labels = box_angle.create_marking()
165     display.DisplayShape(prism, update=True)
```

```
    display_lines(lines, m_pnt, labels)
    display.DisableAntiAliasing()
    start_display()
```

channel section

```
import numpy
from cad1.items.ModelUtils import *
from OCC.Core.BRepAlgoAPI import BRepAlgoAPI_Fuse
from cad1.items.channel import Channel
from cad1.items.plate import Plate

class ChannelSection(object):

    def __init__(self, D, B, T, t, s, l, t1, H):
        self.B = B
        self.T = T
        self.D = D
        self.t = t
        self.l = l
        self.s = s
        self.t1 = t1
        self.H = H

        self.sec_origin = numpy.array([0, 0, 0])
        self.uDir = numpy.array([1.0, 0, 0])
        self.wDir = numpy.array([0.0, 0, 1.0])

        self.Plate1 = Plate(t1, H, l)
        self.Plate2 = Plate(t1, H, l)
        self.channel1 = Channel(B, T, D, t, 0, 0, H)
        self.channel2 = Channel(B, T, D, t, 0, 0, H)
        #self.compute_params()

    def place(self, sec_origin, uDir, wDir):
        self.sec_origin = sec_origin
        self.uDir = uDir
        self.wDir = wDir
        space = -self.s/2+self.B-self.t
        origin = numpy.array([space,0.,0.])
        self.channel1.place(origin, self.uDir, self.wDir)
        origin1 = numpy.array([space,0.,0.])
        self.channel2.place(origin1, self.uDir, self.wDir)
        origin2 = numpy.array([0., -self.t1/2,0.])
        self.Plate1.place(origin2, self.uDir, self.wDir)
        origin3 = numpy.array([0.,self.D+self.t1/2,0.])
        self.Plate2.place(origin3, self.uDir, self.wDir)
        #self.compute_params()

    def compute_params(self):
        self.channel1.compute_params()
        self.channel2.compute_params()
        self.channel2.points = self.rotateY(self.channel2.points)
        self.channel2.points = self.rotateY(self.channel2.points)
        self.Plate1.compute_params()
```

```
50          self.Plate2.compute_params()
51
52      def create_model(self):
53          prism1 = self.channel1.create_model()
54          prism2 = self.channel2.create_model()
55
56          prism3 = self.Plate1.create_model()
57          prism4 = self.Plate2.create_model()
58
59          prism = BRepAlgoAPI_Fuse(prism1, prism2).Shape()
60          prism = BRepAlgoAPI_Fuse(prism, prism3).Shape()
61          prism = BRepAlgoAPI_Fuse(prism, prism4).Shape()
62          return prism
63
64      def rotateY(self, points):
65          rotated_points = []
66          rmatrix = numpy.array([[0, 0, 1],[0, 1, 0],[-1, 0, 0]])
67          for point in points:
68              point = numpy.matmul(rmatrix, point)
69              rotated_points.append(point)
70          return rotated_points
71
72      def create_marking(self):
73          middel_pnt = []
74          line = []
75          labels = ["z","y","u","v"]
76          offset = 100
77          uvoffset = offset/numpy.sqrt(2)
78
79          z_points = [numpy.array([-offset,self.D/2,self.H/2]),
            ↪  numpy.array([offset,self.D/2,self.H/2])]
80          line.append(makeEdgesFromPoints(z_points))
81
82          y_points = [numpy.array([0.,-offset+self.D/2,self.H/2]),
            ↪  numpy.array([0,offset+self.D/2,self.H/2])]
83          line.append(makeEdgesFromPoints(y_points))
84
85          u_points = [numpy.array([-uvoffset,uvoffset+self.D/2,self.H/2]),
            ↪  numpy.array([uvoffset,-uvoffset+self.D/2,self.H/2])]
86          line.append(makeEdgesFromPoints(u_points))
87
88          v_points = [numpy.array([-uvoffset,-uvoffset+self.D/2,self.H/2]),
            ↪  numpy.array([uvoffset,uvoffset+self.D/2,self.H/2])]
89          line.append(makeEdgesFromPoints(v_points))
90
91          middel_pnt =
            ↪  [[-offset,self.D/2,self.H/2],[0,-offset+self.D/2,self.H/2],[uvoffset,-uvoffset+self.
92
93          return line, middel_pnt, labels
94
95  if __name__ == '__main__':
96      from OCC.Display.SimpleGui import init_display
97
98      display, start_display, add_menu, add_function_to_menu = init_display()
99
100     def display_lines(lines, points, labels):
```

```
        for l,p,n in zip(lines,points, labels):
            display.DisplayShape(l, update=True)
            display.DisplayMessage(getGpPt(p), n,message_color=(0,0,0))

    B = 20
    T = 4
    D = 40
    t = 4
    t1 = 4
    s = 50
    l = s + 2*t
    H = 50

    origin = numpy.array([0.,0.,0.])
    uDir = numpy.array([1.,0.,0.])
    shaftDir = numpy.array([0.,0.,1.])

    channel_section = ChannelSection(D, B, T, t, s, l, t1, H)
    _place = channel_section.place(origin, uDir, shaftDir)
    point = channel_section.compute_params()
    prism = channel_section.create_model()
    lines, m_pnt, labels = channel_section.create_marking()
    display.DisplayShape(prism, update=True)
    display_lines(lines, m_pnt, labels)
    display.DisableAntiAliasing()
    start_display()
```

channel section opposite

```
import numpy
from cad1.items.ModelUtils import *
from OCC.Core.BRepAlgoAPI import BRepAlgoAPI_Fuse
from cad1.items.channel import Channel
from cad1.items.plate import Plate

class ChannelSectionOpposite(object):

    def __init__(self, D, B, T, t, s, l, t1, H):
        self.B = B
        self.T = T
        self.D = D
        self.t = t
        self.l = l
        self.s = s
        self.t1 = t1
        self.H = H

        self.sec_origin = numpy.array([0, 0, 0])
        self.uDir = numpy.array([1.0, 0, 0])
        self.wDir = numpy.array([0.0, 0, 1.0])

        self.Plate1 = Plate(t1, H, l)
        self.Plate2 = Plate(t1, H, l)
        self.channel1 = Channel(B, T, D, t, 0, 0, H)
```

```
        self.channel2 = Channel(B, T, D, t, 0, 0, H)
        #self.compute_params()

    def place(self, sec_origin, uDir, wDir):
        self.sec_origin = sec_origin
        self.uDir = uDir
        self.wDir = wDir
        space = self.s/2+self.B
        origin = numpy.array([space,0.,0.])
        self.channel1.place(origin, self.uDir, self.wDir)
        origin1 = numpy.array([space,0.,0.])
        self.channel2.place(origin1, self.uDir, self.wDir)
        origin2 = numpy.array([0., -self.t1/2,0.])
        self.Plate1.place(origin2, self.uDir, self.wDir)
        origin3 = numpy.array([0.,self.D+self.t1/2,0.])
        self.Plate2.place(origin3, self.uDir, self.wDir)
        #self.compute_params()

    def compute_params(self):
        self.channel1.compute_params()
        self.channel2.compute_params()
        self.channel2.points = self.rotateY(self.channel2.points)
        self.channel2.points = self.rotateY(self.channel2.points)
        self.Plate1.compute_params()
        self.Plate2.compute_params()

    def create_model(self):
        prism1 = self.channel1.create_model()
        prism2 = self.channel2.create_model()

        prism3 = self.Plate1.create_model()
        prism4 = self.Plate2.create_model()

        prism = BRepAlgoAPI_Fuse(prism1, prism2).Shape()
        prism = BRepAlgoAPI_Fuse(prism, prism3).Shape()
        prism = BRepAlgoAPI_Fuse(prism, prism4).Shape()
        return prism

    def rotateY(self, points):
        rotated_points = []
        rmatrix = numpy.array([[0, 0, 1],[0, 1, 0],[-1, 0, 0]])
        for point in points:
            point = numpy.matmul(rmatrix, point)
            rotated_points.append(point)
        return rotated_points

    def create_marking(self):
        middel_pnt = []
        line = []
        labels = ["z","y","u","v"]
        offset = 100
        uvoffset = offset/numpy.sqrt(2)

        z_points = [numpy.array([-offset,self.D/2,self.H/2]),
        ↪  numpy.array([offset,self.D/2,self.H/2])]
        line.append(makeEdgesFromPoints(z_points))
```

```
81
82          y_points = [numpy.array([0.,-offset+self.D/2,self.H/2]),
            ↪  numpy.array([0,offset+self.D/2,self.H/2])]
83          line.append(makeEdgesFromPoints(y_points))
84
85          u_points = [numpy.array([-uvoffset,uvoffset+self.D/2,self.H/2]),
            ↪  numpy.array([uvoffset,-uvoffset+self.D/2,self.H/2])]
86          line.append(makeEdgesFromPoints(u_points))
87
88          v_points = [numpy.array([-uvoffset,-uvoffset+self.D/2,self.H/2]),
            ↪  numpy.array([uvoffset,uvoffset+self.D/2,self.H/2])]
89          line.append(makeEdgesFromPoints(v_points))
90
91          middel_pnt =
            ↪  [[-offset,self.D/2,self.H/2],[0,-offset+self.D/2,self.H/2],[uvoffset,-uvoffset+self.
92
93          return line, middel_pnt, labels
94
95  if __name__ == '__main__':
96      from OCC.Display.SimpleGui import init_display
97
98      display, start_display, add_menu, add_function_to_menu = init_display()
99
100     def display_lines(lines, points, labels):
101         for l,p,n in zip(lines,points, labels):
102             display.DisplayShape(l, update=True)
103             display.DisplayMessage(getGpPt(p), n,
                ↪  height=24,message_color=(0,0,0))
104
105     B = 20
106     T = 4
107     D = 40
108     t = 4
109     t1 = 4
110     s = 10
111     l = s + 2*B
112     H = 50
113
114     origin = numpy.array([0.,0.,0.])
115     uDir = numpy.array([1.,0.,0.])
116     shaftDir = numpy.array([0.,0.,1.])
117
118     channel_section_opp = ChannelSectionOpposite(D, B, T, t, s, l, t1, H)
119     _place = channel_section_opp.place(origin, uDir, shaftDir)
120     point = channel_section_opp.compute_params()
121     prism = channel_section_opp.create_model()
122     lines, m_pnt, labels = channel_section_opp.create_marking()
123     display.DisplayShape(prism, update=True)
124     display_lines(lines, m_pnt, labels)
125     display.DisableAntiAliasing()
126     start_display()
```

cross isection

```
1  import numpy
```

```
from cad1.items.ModelUtils import *
from OCC.Core.BRepAlgoAPI import BRepAlgoAPI_Fuse
#from notch import Notch
from cad1.items.plate import Plate
from cad1.items.ISection import ISection

class cross_isection(object):

    def __init__(self, D, B, T, t, H, s, d):
        self.B = B
        self.T = T
        self.D = D
        self.t = t
        self.H = H
        self.s = s
        self.d = d

        self.Isection1 = ISection(2*s+t+2*T, T, 2*d+2*T+t, t, 0, 0, None, H,
        ↪  None)
        self.Isection2 = ISection(2*d+t, T, 2*s+t+2*T, t, 0, 0, None, H,
        ↪  None)


    def place(self, sec_origin, uDir, wDir):
        self.sec_origin = sec_origin
        self.uDir = uDir
        self.wDir = wDir

        self.Isection1.place(self.sec_origin, self.uDir, self.wDir)
        self.Isection2.place(self.sec_origin, self.uDir, self.wDir)

    def compute_params(self):
        self.Isection1.compute_params()
        self.Isection2.compute_params()
        self.Isection2.points = self.retate(self.Isection2.points)


    def create_model(self):

        prism1 = self.Isection1.create_model()
        prism2 = self.Isection2.create_model()

        prism = BRepAlgoAPI_Fuse(prism1, prism2).Shape()
        return prism

    def retate(self, points):
        rotated_points = []
        rmatrix = numpy.array([[0, -1, 0],[1, 0, 0],[0, 0, 1]])
        for point in points:
            point = numpy.matmul(rmatrix, point)
            rotated_points.append(point)
        return rotated_points

    def create_marking(self):
        middel_pnt = []
        line = []
```

```
56            labels = ["z","y","u","v"]
57            offset = 100
58            uvoffset = offset/numpy.sqrt(2)
59
60            z_points = [numpy.array([-offset,0.,self.H/2]),
              ↪  numpy.array([offset,0.,self.H/2])]
61            line.append(makeEdgesFromPoints(z_points))
62
63            y_points = [numpy.array([0.,-offset,self.H/2]),
              ↪  numpy.array([0,offset,self.H/2])]
64            line.append(makeEdgesFromPoints(y_points))
65
66            u_points = [numpy.array([-uvoffset,uvoffset,self.H/2]),
              ↪  numpy.array([uvoffset,-uvoffset,self.H/2])]
67            line.append(makeEdgesFromPoints(u_points))
68
69            v_points = [numpy.array([-uvoffset,-uvoffset,self.H/2]),
              ↪  numpy.array([uvoffset,uvoffset,self.H/2])]
70            line.append(makeEdgesFromPoints(v_points))
71
72            middel_pnt =
              ↪  [[-offset,0,self.H/2],[0,-offset,self.H/2],[uvoffset,-uvoffset,self.H/2],[uvoffset,u
73
74            return line, middel_pnt, labels
75
76    if __name__ == '__main__':
77
78        from OCC.Display.SimpleGui import init_display
79        display, start_display, add_menu, add_function_to_menu = init_display()
80
81        def display_lines(lines, points, labels):
82            for l,p,n in zip(lines,points, labels):
83                display.DisplayShape(l, update=True)
84                display.DisplayMessage(getGpPt(p), n,message_color=(0,0,0))
85
86        B = 50
87        T = 3
88        D = 70
89        t = 2
90        H = 100
91        d = (B - 2*T - t)/2
92        s = (D - t)/2
93
94        CrossISec = cross_isection(D, B, T, t, H, s, d)
95
96        origin = numpy.array([0.,0.,0.])
97        uDir = numpy.array([1.,0.,0.])
98        shaftDir = numpy.array([0.,0.,1.])
99
100       CrossISec.place(origin, uDir, shaftDir)
101       CrossISec.compute_params()
102       prism = CrossISec.create_model()
103       lines, m_pnt, labels = CrossISec.create_marking()
104       display.DisplayShape(prism, update=True)
105       display_lines(lines, m_pnt, labels)
106       display.DisableAntiAliasing()
```

```
    start_display()
```

isection channel

```
import numpy
from cad1.items.ModelUtils import *
from OCC.Core.BRepAlgoAPI import BRepAlgoAPI_Fuse
from cad1.items.channel import Channel
from cad1.items.plate import Plate
from cad1.items.ISection import ISection

class ISectionChannel(object):

    def __init__(self, D, B, T, t, T1, t1, d, b, H, s):
        self.B = B
        self.T = T
        self.D = D
        self.t = t
        self.T1 = T1
        self.t1 = t1
        self.d = d
        self.b = b
        self.H = H
        self.s = s
        self.B = 2*self.s-2*T1
        self.d = 2*self.s

        self.sec_origin = numpy.array([0, 0, 0])
        self.uDir = numpy.array([1.0, 0, 0])
        self.wDir = numpy.array([0.0, 0, 1.0])

        self.channel1 = Channel(b, T1, self.d, t1, 0, 0, H)
        self.isection = ISection(self.B, T, D, t, 0, 0, 0, H, None)
        #self.compute_params()

    def place(self, sec_origin, uDir, wDir):
        self.sec_origin = sec_origin
        self.uDir = uDir
        self.wDir = wDir
        D = self.D/2
        origin = numpy.array([-D+self.b-self.t1,0.,0.])
        self.channel1.place(origin , self.uDir, self.wDir)
        origin1 = numpy.array([self.s,0.,0.])
        self.isection.place(origin1, self.uDir, self.wDir)

    def compute_params(self):
        self.channel1.compute_params()
        self.isection.compute_params()
        self.isection.points = self.rotateZ(self.isection.points)

    def create_model(self):
        prism1 = self.channel1.create_model()
        prism3 = self.isection.create_model()

        prism = BRepAlgoAPI_Fuse(prism1, prism3).Shape()
```

```
52          return prism
53
54      def rotateZ(self, points):
55          rotated_points = []
56          rmatrix = numpy.array([[0, -1, 0],[1, 0, 0],[0, 0, 1]])
57          for point in points:
58              point = numpy.matmul(rmatrix, point)
59              rotated_points.append(point)
60          return rotated_points
61
62      def create_marking(self):
63          middel_pnt = []
64          line = []
65          labels = ["z","y","u","v"]
66          offset = 100
67          uvoffset = offset/numpy.sqrt(2)
68
69          z_points = [numpy.array([-offset,self.d/2,self.H/2]),
            ↪  numpy.array([offset,self.d/2,self.H/2])]
70          line.append(makeEdgesFromPoints(z_points))
71
72          y_points = [numpy.array([0.,-offset+self.d/2,self.H/2]),
            ↪  numpy.array([0,offset+self.d/2,self.H/2])]
73          line.append(makeEdgesFromPoints(y_points))
74
75          u_points = [numpy.array([-uvoffset,uvoffset+self.d/2,self.H/2]),
            ↪  numpy.array([uvoffset,-uvoffset+self.d/2,self.H/2])]
76          line.append(makeEdgesFromPoints(u_points))
77
78          v_points = [numpy.array([-uvoffset,-uvoffset+self.d/2,self.H/2]),
            ↪  numpy.array([uvoffset,uvoffset+self.d/2,self.H/2])]
79          line.append(makeEdgesFromPoints(v_points))
80
81          middel_pnt =
            ↪  [[-offset,self.d/2,self.H/2],[0,-offset+self.d/2,self.H/2],[uvoffset,-uvoffset+self.
82
83          return line, middel_pnt, labels
84
85  if __name__ == '__main__':
86      from OCC.Display.SimpleGui import init_display
87
88      display, start_display, add_menu, add_function_to_menu = init_display()
89
90      def display_lines(lines, points, labels):
91          for l,p,n in zip(lines,points, labels):
92              display.DisplayShape(l, update=True)
93              display.DisplayMessage(getGpPt(p), n,
                ↪  height=24,message_color=(0,0,0))
94
95      B = 20
96      T = 2
97      D = 40
98      t = 1.5
99      T1 = 2
100     t1 = 2
101     H = 60
```

```
    b = 20
    d = 50
    s = 15

    origin = numpy.array([0.,0.,0.])
    uDir = numpy.array([1.,0.,0.])
    shaftDir = numpy.array([0.,0.,1.])

    isection_channel = ISectionChannel(D, B, T, t, T1, t1, d, b, H, s)
    print(isection_channel.B)
    _place = isection_channel.place(origin, uDir, shaftDir)
    point = isection_channel.compute_params()
    prism = isection_channel.create_model()
    lines, m_pnt, labels = isection_channel.create_marking()
    display.DisplayShape(prism, update=True)
    display_lines(lines, m_pnt, labels)
    display.DisableAntiAliasing()
    start_display()
```

isection coverplate

```
import numpy
from cad1.items.ModelUtils import *
from OCC.Core.BRepAlgoAPI import BRepAlgoAPI_Fuse
#from notch import Notch
from cad1.items.plate import Plate
from cad1.items.ISection import ISection

class IsectionCoverPlate(object):

    def __init__(self, D, B, T, t, s, l, t1, H):
        self.B = B
        self.T = T
        self.D = D
        self.t = t
        self.l = l
        self.s = s
        self.t1 = t1
        self.H = H

        self.Isection1 = ISection(B, T, D, t, 0, 0, 0, H, None)
        self.Isection2 = ISection(B, T, D, t, 0, 0, 0, H, None)
        self.Plate1 = Plate(t1, H, l)
        self.Plate2 = Plate(t1, H, l)

    def place(self, sec_origin, uDir, wDir):
        self.sec_origin = sec_origin
        self.uDir = uDir
        self.wDir = wDir

        origin = numpy.array([-self.s/2.,0.,0.])
        self.Isection1.place(origin, self.uDir, self.wDir)
        origin1 = numpy.array([self.s/2.,0.,0.])
        self.Isection2.place(origin1, self.uDir, self.wDir)
```

```
34          origin2 = numpy.array([0.,(self.D+self.t1)/2,0.])
35          self.Plate1.place(origin2, self.uDir, self.wDir)
36          origin3 = numpy.array([0.,-(self.D+self.t1)/2,0.])
37          self.Plate2.place(origin3, self.uDir, self.wDir)
38          #self.compute_params()
39
40      def compute_params():
41          self.Isection1.compute_params()
42          self.Isection2.compute_params()
43          self.Plate1.compute_params()
44          self.Plate2.compute_params()
45
46      def create_model(self):
47
48          prism1 = self.Isection1.create_model()
49          prism2 = self.Isection2.create_model()
50
51          prism3 = self.Plate1.create_model()
52          prism4 = self.Plate2.create_model()
53
54          prism = BRepAlgoAPI_Fuse(prism1, prism2).Shape()
55          prism = BRepAlgoAPI_Fuse(prism, prism3).Shape()
56          prism = BRepAlgoAPI_Fuse(prism, prism4).Shape()
57          return prism
58
59      def create_marking(self):
60          middel_pnt = []
61          line = []
62          labels = ["z","y","u","v"]
63          offset = 100
64          uvoffset = offset/numpy.sqrt(2)
65
66          #b = self.B/2+self.s/2
67          z_points = [numpy.array([-offset,0.,self.H/2]),
            ↪  numpy.array([offset,0.,self.H/2])]
68          line.append(makeEdgesFromPoints(z_points))
69
70          y_points = [numpy.array([0,-offset,self.H/2]),
            ↪  numpy.array([0,offset,self.H/2])]
71          line.append(makeEdgesFromPoints(y_points))
72
73          u_points = [numpy.array([-uvoffset,uvoffset,self.H/2]),
            ↪  numpy.array([uvoffset,-uvoffset,self.H/2])]
74          line.append(makeEdgesFromPoints(u_points))
75
76          v_points = [numpy.array([-uvoffset,-uvoffset,self.H/2]),
            ↪  numpy.array([uvoffset,uvoffset,self.H/2])]
77          line.append(makeEdgesFromPoints(v_points))
78
79          middel_pnt =
            ↪  [[-offset,0,self.H/2],[0,-offset,self.H/2],[uvoffset,-uvoffset,self.H/2],[uvoffset,u
80
81          return line, middel_pnt, labels
82
83  if __name__ == '__main__':
84
```

```
    from OCC.Display.SimpleGui import init_display
    display, start_display, add_menu, add_function_to_menu = init_display()

    def display_lines(lines, points, labels):
        for l,p,n in zip(lines,points, labels):
            display.DisplayShape(l, update=True)
            display.DisplayMessage(getGpPt(p), n,
            ↪  height=24,message_color=(0,0,0))

    B = 40
    T = 3
    D = 40
    t = 3
    s = 50
    l = B + s
    t2 = 3
    H = 50

    ISecPlate = IsectionCoverPlate(D, B, T, t, s, l, t2, H)

    origin = numpy.array([0.,0.,0.])
    uDir = numpy.array([1.,0.,0.])
    shaftDir = numpy.array([0.,0.,1.])

    ISecPlate.place(origin, uDir, shaftDir)
    prism = ISecPlate.create_model()
    lines, m_pnt, labels = ISecPlate.create_marking()
    display.DisplayShape(prism, update=True)
    display_lines(lines, m_pnt, labels)
    display.DisableAntiAliasing()
    start_display()
```

star with two angle

```
import numpy
from cad1.items.ModelUtils import *
from OCC.Core.BRepAlgoAPI import BRepAlgoAPI_Fuse
#from cad.items.angle import Angle
from anglebar import Angle
from cad1.items.plate import Plate

class StarAngle2(object):
    def __init__(self, a, b, t, l, t1, H):
        self.l = l
        self.a = a
        self.b = b
        self.t = t
        self.t1 = t1
        self.H = H

        self.sec_origin = numpy.array([0, 0, 0])
        self.uDir = numpy.array([1.0, 0, 0])
        self.wDir = numpy.array([0.0, 0, 1.0])
        self.vDir = self.wDir * self.uDir
```

```
21
22          self.angle1 = Angle(H, a, b, t, 0, 0)
23          self.angle2 = Angle(H, a, b, t, 0, 0)
24          self.plate1 = Plate(l, H, t1)
25          #self.plate2 = Plate(t, L, W)
26
27      def place(self, secOrigin, uDir, wDir):
28          self.sec_origin = secOrigin
29          self.uDir = uDir
30          self.wDir = wDir
31          origin1 = numpy.array([self.t1/2, 0., 0.])
32          self.angle1.place(origin1, self.uDir, self.wDir)
33          origin2 = numpy.array([self.t1/2, 0., 0.])
34          self.angle2.place(origin2, self.uDir, self.wDir)
35          self.plate1.place(self.sec_origin, self.uDir, self.wDir)
36          #self.plate2.place(self.sec_origin, self.uDir, self.wDir)
37
38      def compute_params(self):
39          self.angle1.computeParams()
40          self.angle2.computeParams()
41          self.angle2.points = self.rotate(self.angle2.points, numpy.pi)
42          self.plate1.compute_params()
43          #self.plate2.compute_params()
44
45      def create_model(self):
46          prism1 = self.angle1.create_model()
47          prism2 = self.angle2.create_model()
48
49          prism3 = self.plate1.create_model()
50          #prism4 = self.plate2.create_model()
51
52          prism = BRepAlgoAPI_Fuse(prism1, prism2).Shape()
53          prism = BRepAlgoAPI_Fuse(prism, prism3).Shape()
54          #prism = BRepAlgoAPI_Fuse(prism, prism4).Shape()
55          return prism
56
57      def rotate(self, points, x):
58          rotated_points = []
59          rmatrix = numpy.array([[numpy.cos(x), -numpy.sin(x), 0],
60                                 [numpy.sin(x), numpy.cos(x), 0],
61                                 [0, 0, 1]])
62          for point in points:
63              point = numpy.matmul(rmatrix, point)
64              rotated_points.append(point)
65          return rotated_points
66
67      def create_marking(self):
68          middel_pnt = []
69          line = []
70          labels = ["z","y","u","v"]
71          offset = 100
72          uvoffset = offset/numpy.sqrt(2)
73
74          z_points = [numpy.array([-offset,0.,self.H/2]),
            ↪  numpy.array([offset,0.,self.H/2])]
75          line.append(makeEdgesFromPoints(z_points))
```

```
76
77          y_points = [numpy.array([0.,-offset,self.H/2]),
            ↪  numpy.array([0,offset,self.H/2])]
78          line.append(makeEdgesFromPoints(y_points))
79
80          u_points = [numpy.array([-uvoffset,uvoffset,self.H/2]),
            ↪  numpy.array([uvoffset,-uvoffset,self.H/2])]
81          line.append(makeEdgesFromPoints(u_points))
82
83          v_points = [numpy.array([-uvoffset,-uvoffset,self.H/2]),
            ↪  numpy.array([uvoffset,uvoffset,self.H/2])]
84          line.append(makeEdgesFromPoints(v_points))
85
86          middel_pnt =
            ↪  [[-offset,0.,self.H/2],[0,-offset,self.H/2],[uvoffset,-uvoffset,self.H/2],[uvoffset,
87
88          return line, middel_pnt, labels
89
90  if __name__ == '__main__':
91
92      from OCC.Display.SimpleGui import init_display
93      display, start_display, add_menu, add_function_to_menu = init_display()
94
95      def display_lines(lines, points, labels):
96          for l,p,n in zip(lines,points, labels):
97              display.DisplayShape(l, update=True)
98              display.DisplayMessage(getGpPt(p), n,
                ↪  height=24,message_color=(0,0,0))
99
100     a = 15
101     b = 15
102     l = 2*a
103     t = 2
104     t1 = 2
105     H = 50
106
107     origin = numpy.array([0.,0.,0.])
108     uDir = numpy.array([1.,0.,0.])
109     wDir = numpy.array([0.,0.,1.])
110
111     star_angle = StarAngle2(a, b, t, l, t1, H)
112     _place = star_angle.place(origin, uDir, wDir)
113     point = star_angle.compute_params()
114     prism = star_angle.create_model()
115     lines, m_pnt, labels = star_angle.create_marking()
116     display.DisplayShape(prism, update=True)
117     display_lines(lines, m_pnt, labels)
118     display.DisableAntiAliasing()
119     start_display()
```

star with four angle

```
1   import numpy
2   from cad1.items.ModelUtils import *
```

```
from OCC.Core.BRepAlgoAPI import BRepAlgoAPI_Fuse
#from cad.items.angle import Angle
from anglebar import Angle
from cad1.items.plate import Plate

class StarAngle4(object):
    def __init__(self, a, b, t, l, t1, H):
        self.l = l
        self.a = a
        self.b = b
        self.t = t
        self.t1 = t1
        self.H = H

        self.sec_origin = numpy.array([0, 0, 0])
        self.uDir = numpy.array([1.0, 0, 0])
        self.wDir = numpy.array([0.0, 0, 1.0])
        self.vDir = self.wDir * self.uDir

        self.angle1 = Angle(H, a, b, t, 0, 0)
        self.angle2 = Angle(H, b, a, t, 0, 0)
        self.angle3 = Angle(H, a, b, t, 0, 0)
        self.angle4 = Angle(H, b, a, t, 0, 0)
        self.plate1 = Plate(l, H, t1)
        #self.plate2 = Plate(t, L, W)

    def place(self, secOrigin, uDir, wDir):
        self.sec_origin = secOrigin
        self.uDir = uDir
        self.wDir = wDir
        #t = self.t/2
        origin1 = numpy.array([self.t1/2, 0., 0.])
        self.angle1.place(origin1, self.uDir, self.wDir)
        origin2 = numpy.array([0., self.t1/2, 0.])
        self.angle2.place(origin2, self.uDir, self.wDir)
        origin3 = numpy.array([self.t1/2, 0., 0.])
        self.angle3.place(origin3, self.uDir, self.wDir)
        origin4 = numpy.array([0., self.t1/2, 0.])
        self.angle4.place(origin4, self.uDir, self.wDir)
        self.plate1.place(self.sec_origin, self.uDir, self.wDir)
        #self.plate2.place(self.sec_origin, self.uDir, self.wDir)

    def compute_params(self):
        self.angle1.computeParams()
        self.angle2.computeParams()
        self.angle2.points = self.rotate(self.angle2.points, numpy.pi/2)
        self.angle3.computeParams()
        self.angle3.points = self.rotate(self.angle3.points, numpy.pi)
        self.angle4.computeParams()
        self.angle4.points = self.rotate(self.angle4.points, 3*numpy.pi/2)

        self.plate1.compute_params()
        #self.plate2.compute_params()

    def create_model(self):
        prism1 = self.angle1.create_model()
```

```
59          prism2 = self.angle2.create_model()
60          prism3 = self.angle3.create_model()
61          prism4 = self.angle4.create_model()
62
63
64          prism5 = self.plate1.create_model()
65          #prism6 = self.plate2.create_model()
66
67          prism = BRepAlgoAPI_Fuse(prism1, prism2).Shape()
68          prism = BRepAlgoAPI_Fuse(prism, prism3).Shape()
69          prism = BRepAlgoAPI_Fuse(prism, prism4).Shape()
70          prism = BRepAlgoAPI_Fuse(prism, prism5).Shape()
71          #prism = BRepAlgoAPI_Fuse(prism, prism6).Shape()
72          return prism
73
74
75      def rotate(self, points, x):
76          rotated_points = []
77          rmatrix = numpy.array([[numpy.cos(x), -numpy.sin(x), 0],
78                                 [numpy.sin(x), numpy.cos(x), 0],
79                                 [0, 0, 1]])
80          for point in points:
81              point = numpy.matmul(rmatrix, point)
82              rotated_points.append(point)
83          return rotated_points
84
85      def create_marking(self):
86          middel_pnt = []
87          line = []
88          labels = ["z","y","u","v"]
89          offset = 100
90          uvoffset = offset/numpy.sqrt(2)
91
92          z_points = [numpy.array([-offset,0.,self.H/2]),
            ↪  numpy.array([offset,0.,self.H/2])]
93          line.append(makeEdgesFromPoints(z_points))
94
95          y_points = [numpy.array([0.,-offset,self.H/2]),
            ↪  numpy.array([0,offset,self.H/2])]
96          line.append(makeEdgesFromPoints(y_points))
97
98          u_points = [numpy.array([-uvoffset,uvoffset,self.H/2]),
            ↪  numpy.array([uvoffset,-uvoffset,self.H/2])]
99          line.append(makeEdgesFromPoints(u_points))
100
101         v_points = [numpy.array([-uvoffset,-uvoffset,self.H/2]),
            ↪  numpy.array([uvoffset,uvoffset,self.H/2])]
102         line.append(makeEdgesFromPoints(v_points))
103
104         middel_pnt =
            ↪  [[-offset,0.,self.H/2],[0,-offset,self.H/2],[uvoffset,-uvoffset,self.H/2],[uvoffset,
105
106         return line, middel_pnt, labels
107
108 if __name__ == '__main__':
109
```

```
    from OCC.Display.SimpleGui import init_display
    display, start_display, add_menu, add_function_to_menu = init_display()

    def display_lines(lines, points, labels):
        for l,p,n in zip(lines,points, labels):
            display.DisplayShape(l, update=True)
            display.DisplayMessage(getGpPt(p), n,
            ↪  height=24,message_color=(0,0,0))

    a = 15
    b = 15
    l = 2*a
    t = 2
    t1 = 2
    H = 50

    origin = numpy.array([0.,0.,0.])
    uDir = numpy.array([1.,0.,0.])
    wDir = numpy.array([0.,0.,1.])

    star_angle = StarAngle4(a, b, t, l, t1, H)
    _place = star_angle.place(origin, uDir, wDir)
    point = star_angle.compute_params()
    prism = star_angle.create_model()
    lines, m_pnt, labels = star_angle.create_marking()
    display.DisplayShape(prism, update=True)
    display_lines(lines, m_pnt, labels)
    display.DisableAntiAliasing()
    start_display()
```

star angle opposite

```
import numpy
from cad1.items.ModelUtils import *
from OCC.Core.BRepAlgoAPI import BRepAlgoAPI_Fuse
from anglebar import Angle
from cad1.items.plate import Plate

class StarAngleOpposite(object):
    def __init__(self, a, b, t, l, t1, H):
        self.l = l
        self.a = a
        self.b = b
        self.t = t
        self.t1 = t1
        self.H = H

        self.sec_origin = numpy.array([0, 0, 0])
        self.uDir = numpy.array([1.0, 0, 0])
        self.wDir = numpy.array([0.0, 0, 1.0])
        self.vDir = self.wDir * self.uDir

        self.angle1 = Angle(H, a, b, t, 0, 0)
        self.angle2 = Angle(H, b, a, t, 0, 0)
```

```
23          self.plate1 = Plate(l, H, t1)
24
25      def place(self, secOrigin, uDir, wDir):
26          self.sec_origin = secOrigin
27          self.uDir = uDir
28          self.wDir = wDir
29          origin1 = numpy.array([self.t1/2., 0., 0.])
30          self.angle1.place(origin1, self.uDir, self.wDir)
31          origin2 = numpy.array([0., self.t1/2., 0])
32          self.angle2.place(origin2, self.uDir, self.wDir)
33          origin3 = numpy.array([0., self.a/2, 0.])
34          self.plate1.place(origin3, self.uDir, self.wDir)
35
36      def compute_params(self):
37          self.angle1.computeParams()
38          self.angle2.computeParams()
39          self.angle2.points = self.rotate(self.angle2.points)
40          self.plate1.compute_params()
41
42      def create_model(self):
43          prism1 = self.angle1.create_model()
44          prism2 = self.angle2.create_model()
45
46          prism3 = self.plate1.create_model()
47
48          prism = BRepAlgoAPI_Fuse(prism1, prism2).Shape()
49          prism = BRepAlgoAPI_Fuse(prism, prism3).Shape()
50          return prism
51
52      def rotate(self, points):
53          rotated_points = []
54          rmatrix = numpy.array([[0, -1, 0],[1, 0, 0],[0, 0, 1]])
55          for point in points:
56              point = numpy.matmul(rmatrix, point)
57              rotated_points.append(point)
58          return rotated_points
59
60      def create_marking(self):
61          middel_pnt = []
62          line = []
63          labels = ["z","y","u","v"]
64          offset = 100
65          uvoffset = offset/numpy.sqrt(2)
66
67          z_points = [numpy.array([-offset,self.t/2,self.H/2]),
            ↪  numpy.array([offset,self.t/2,self.H/2])]
68          line.append(makeEdgesFromPoints(z_points))
69
70          y_points = [numpy.array([0.,-offset+self.t/2,self.H/2]),
            ↪  numpy.array([0,offset+self.t/2,self.H/2])]
71          line.append(makeEdgesFromPoints(y_points))
72
73          u_points = [numpy.array([-uvoffset,uvoffset+self.t/2,self.H/2]),
            ↪  numpy.array([uvoffset,-uvoffset+self.t/2,self.H/2])]
74          line.append(makeEdgesFromPoints(u_points))
75
```

```
        v_points = [numpy.array([-uvoffset,-uvoffset+self.t/2,self.H/2]),
        ↪  numpy.array([uvoffset,uvoffset+self.t/2,self.H/2])]
        line.append(makeEdgesFromPoints(v_points))

        middel_pnt =
        ↪  [[-offset,self.t/2,self.H/2],[0,-offset+self.t/2,self.H/2],[uvoffset,-uvoffset+self.

        return line, middel_pnt, labels


if __name__ == '__main__':

    from OCC.Display.SimpleGui import init_display
    display, start_display, add_menu, add_function_to_menu = init_display()

    def display_lines(lines, points, labels):
        for l,p,n in zip(lines,points, labels):
            display.DisplayShape(l, update=True)
            display.DisplayMessage(getGpPt(p), n,
            ↪  height=24,message_color=(0,0,0))

    a = 15
    b = 15
    l = a
    t = 2
    t1 = 2
    H = 50

    origin = numpy.array([0.,0.,0.])
    uDir = numpy.array([1.,0.,0.])
    wDir = numpy.array([0.,0.,1.])

    star_angle_opposite = StarAngleOpposite(a, b, t, l, t1, H)
    _place = star_angle_opposite.place(origin, uDir, wDir)
    point = star_angle_opposite.compute_params()
    prism = star_angle_opposite.create_model()
    lines, m_pnt, labels = star_angle_opposite.create_marking()
    display.DisplayShape(prism, update=True)
    display_lines(lines, m_pnt, labels)
    display.DisableAntiAliasing()
    start_display()
```

star angle same

```
import numpy
from cad1.items.ModelUtils import *
from OCC.Core.BRepAlgoAPI import BRepAlgoAPI_Fuse
#from cad.items.angle import Angle
from anglebar import Angle
from cad1.items.plate import Plate

class StarAngleSame(object):
    def __init__(self, a, b, t, l, t1, H):
        self.l = l
```

```
        self.a = a
        self.b = b
        self.t = t
        self.t1 = t1
        self.H = H

        self.sec_origin = numpy.array([0, 0, 0])
        self.uDir = numpy.array([1.0, 0, 0])
        self.wDir = numpy.array([0.0, 0, 1.0])
        self.vDir = self.wDir * self.uDir
        self.angle1 = Angle(H, a, b, t, 0, 0)
        self.angle2 = Angle(H, b, a, t, 0, 0)
        self.plate1 = Plate(l, H, t1)

    def place(self, secOrigin, uDir, wDir):
        self.sec_origin = secOrigin
        self.uDir = uDir
        self.wDir = wDir
        origin1 = numpy.array([self.t1/2., 0., 0.])
        self.angle1.place(origin1, self.uDir, self.wDir)
        origin2 = numpy.array([0., self.t1/2., 0])
        self.angle2.place(origin2, self.uDir, self.wDir)
        self.plate1.place(self.sec_origin, self.uDir, self.wDir)

    def compute_params(self):
        self.angle1.computeParams()
        self.angle2.computeParams()
        self.angle2.points = self.rotate(self.angle2.points)
        self.angle2.points = self.rotate(self.angle2.points)
        self.angle2.points = self.rotate(self.angle2.points)
        self.plate1.compute_params()

    def create_model(self):
        prism1 = self.angle1.create_model()
        prism2 = self.angle2.create_model()

        prism3 = self.plate1.create_model()

        prism = BRepAlgoAPI_Fuse(prism1, prism2).Shape()
        prism = BRepAlgoAPI_Fuse(prism, prism3).Shape()
        return prism

    def rotate(self, points):
        rotated_points = []
        rmatrix = numpy.array([[0, -1, 0],[1, 0, 0],[0, 0, 1]])
        for point in points:
            point = numpy.matmul(rmatrix, point)
            rotated_points.append(point)
        return rotated_points

    def create_marking(self):
        middel_pnt = []
        line = []
        labels = ["z","y","u","v"]
        offset = 100
        uvoffset = offset/numpy.sqrt(2)
```

```
67
68          z_points = [numpy.array([-offset,0.,self.H/2]),
            ↪  numpy.array([offset,0.,self.H/2])]
69          line.append(makeEdgesFromPoints(z_points))
70
71          y_points = [numpy.array([0.,-offset,self.H/2]),
            ↪  numpy.array([0,offset,self.H/2])]
72          line.append(makeEdgesFromPoints(y_points))
73
74          u_points = [numpy.array([-uvoffset,uvoffset,self.H/2]),
            ↪  numpy.array([uvoffset,-uvoffset,self.H/2])]
75          line.append(makeEdgesFromPoints(u_points))
76
77          v_points = [numpy.array([-uvoffset,-uvoffset,self.H/2]),
            ↪  numpy.array([uvoffset,uvoffset,self.H/2])]
78          line.append(makeEdgesFromPoints(v_points))
79
80          middel_pnt =
            ↪  [[-offset,0.,self.H/2],[0,-offset,self.H/2],[uvoffset,-uvoffset,self.H/2],[uvoffset,
81
82          return line, middel_pnt, labels
83
84  if __name__ == '__main__':
85
86      from OCC.Display.SimpleGui import init_display
87      display, start_display, add_menu, add_function_to_menu = init_display()
88
89      def display_lines(lines, points, labels):
90          for l,p,n in zip(lines,points, labels):
91              display.DisplayShape(l, update=True)
92              display.DisplayMessage(getGpPt(p), n,
                ↪  height=24,message_color=(0,0,0))
93
94      a = 15
95      b = 15
96      l = 2*a
97      t = 2
98      t1 = 2
99      H = 50
100
101     origin = numpy.array([0.,0.,0.])
102     uDir = numpy.array([1.,0.,0.])
103     wDir = numpy.array([0.,0.,1.])
104
105     star_angle_same = StarAngleSame(a, b, t, l, t1, H)
106     _place = star_angle_same.place(origin, uDir, wDir)
107     point = star_angle_same.compute_params()
108     prism = star_angle_same.create_model()
109     lines, m_pnt, labels = star_angle_same.create_marking()
110     display.DisplayShape(prism, update=True)
111     display_lines(lines, m_pnt, labels)
112     display.DisableAntiAliasing()
113     start_display()
```

bulid up section

```
import numpy
from cad1.items.ModelUtils import *


class TISection(object):


    def __init__(self, D, B, T, t, P, Q, H):
        self.B = B
        self.T = T
        self.D = D
        self.t = t
        self.d = P
        self.b = Q
        self.length = H

        self.sec_origin = numpy.array([0, 0, 0])
        self.uDir = numpy.array([1.0, 0, 0])
        self.wDir = numpy.array([0.0, 0, 1.0])
        self.compute_params()

    def place(self, sec_origin, uDir, wDir):
        self.sec_origin = sec_origin
        self.uDir = uDir
        self.wDir = wDir
        self.compute_params()

    def compute_params(self):
        self.vDir = numpy.cross(self.wDir, self.uDir)
        self.a1 = self.sec_origin + (self.t / 2.0) * self.uDir + ((self.D /
        ↪  2.0) - self.T) * self.vDir
        self.b1 = self.sec_origin + (self.B / 2.0) * self.uDir + ((self.D /
        ↪  2.0) - self.T) * self.vDir
        self.c1 = self.sec_origin + (self.B / 2.0) * self.uDir + (self.D /
        ↪  2.0) * self.vDir
        self.a2 = self.sec_origin + (-self.t / 2.0) * self.uDir + ((self.D /
        ↪  2.0) - self.T) * self.vDir
        self.b2 = self.sec_origin + (-self.B / 2.0) * self.uDir + ((self.D /
        ↪  2.0) - self.T) * self.vDir
        self.c2 = self.sec_origin + (-self.B / 2.0) * self.uDir + (self.D /
        ↪  2.0) * self.vDir
        self.a3 = self.sec_origin + (-self.t / 2.0) * self.uDir + -((self.D /
        ↪  2.0) - self.T) * self.vDir
        self.d5 = self.sec_origin + ((-self.B / 2.0) + self.b) * self.uDir +
        ↪  -((self.D / 2.0) -self.T - self.d) * self.vDir
        self.d7 = self.sec_origin + ((-self.B / 2.0) + self.b) * self.uDir +
        ↪  -((self.D / 2.0) - self.T) * self.vDir
        self.b3 = self.sec_origin + (-self.B / 2.0) * self.uDir + -((self.D /
        ↪  2.0) - self.T - self.d) * self.vDir
        self.c3 = self.sec_origin + (-self.B / 2.0) * self.uDir + -(self.D /
        ↪  2.0) * self.vDir
        self.a4 = self.sec_origin + (self.t / 2.0) * self.uDir + -((self.D /
        ↪  2.0) - self.T) * self.vDir
```

```
42          self.d6 = self.sec_origin + ((self.B / 2.0) - self.b) * self.uDir +
            ↪  -((self.D / 2.0) - self.T) * self.vDir
43          self.d4 = self.sec_origin + ((self.B / 2.0) - self.b) * self.uDir +
            ↪  -((self.D / 2.0) -self.T - self.d) * self.vDir
44          self.b4 = self.sec_origin + (self.B / 2.0) * self.uDir + -((self.D /
            ↪  2.0) - self.T - self.d) * self.vDir
45          self.c4 = self.sec_origin + (self.B / 2.0) * self.uDir + -(self.D /
            ↪  2.0) * self.vDir
46
47
48
49
50          self.points = [self.a1, self.b1, self.c1,
51                         self.c2, self.b2, self.a2,
52                         self.a3, self.d7, self.d5,
53                         self.b3, self.c3, self.c4,
54                         self.b4, self.d4, self.d6,
55                         self.a4]
56          print(self.d4)
57
58      def create_model(self):
59
60          edges = makeEdgesFromPoints(self.points)
61          wire = makeWireFromEdges(edges)
62          aFace = makeFaceFromWire(wire)
63          extrudeDir = self.length * self.wDir  # extrudeDir is a numpy array
64          prism = makePrismFromFace(aFace, extrudeDir)
65
66          return prism
67
68      def create_marking(self):
69          middel_pnt = []
70          line = []
71          labels = ["z","y","u","v"]
72          offset = 100
73          uvoffset = offset/numpy.sqrt(2)
74
75          z_points = [numpy.array([-offset,0.,self.length/2]),
            ↪  numpy.array([offset,0.,self.length/2])]
76          line.append(makeEdgesFromPoints(z_points))
77
78          y_points = [numpy.array([0.,-offset,self.length/2]),
            ↪  numpy.array([0,offset,self.length/2])]
79          line.append(makeEdgesFromPoints(y_points))
80
81          u_points = [numpy.array([-uvoffset,uvoffset,self.length/2]),
            ↪  numpy.array([uvoffset,-uvoffset,self.length/2])]
82          line.append(makeEdgesFromPoints(u_points))
83
84          v_points = [numpy.array([-uvoffset,-uvoffset,self.length/2]),
            ↪  numpy.array([uvoffset,uvoffset,self.length/2])]
85          line.append(makeEdgesFromPoints(v_points))
86
87          middel_pnt =
            ↪  [[-offset,0,self.length/2],[0,-offset,self.length/2],[uvoffset,-uvoffset,self.length
88
```

```
89          return line, middel_pnt, labels
90
91  if __name__ == '__main__':
92
93      from OCC.Display.SimpleGui import init_display
94      display, start_display, add_menu, add_function_to_menu = init_display()
95
96      def display_lines(lines, points, labels):
97          for l,p,n in zip(lines,points, labels):
98              display.DisplayShape(l, update=True)
99              display.DisplayMessage(getGpPt(p), n,
                ↪  height=24,message_color=(0,0,0))
100
101     B = 40
102     T = 3
103     D = 50
104     t = 2
105     P = 8
106     Q = 4
107     H = 100
108
109     origin = numpy.array([0.,0.,0.])
110     uDir = numpy.array([1.,0.,0.])
111     shaftDir = numpy.array([0.,0.,1.])
112
113     TISec = TISection(D, B, T, t, P, Q, H)
114     _place = TISec.place(origin, uDir, shaftDir)
115     point = TISec.compute_params()
116     prism = TISec.create_model()
117     lines, m_pnt, labels = TISec.create_marking()
118     display.DisplayShape(prism, update=True)
119     display_lines(lines, m_pnt, labels)
120     display.DisableAntiAliasing()
121     start_display()
```