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.

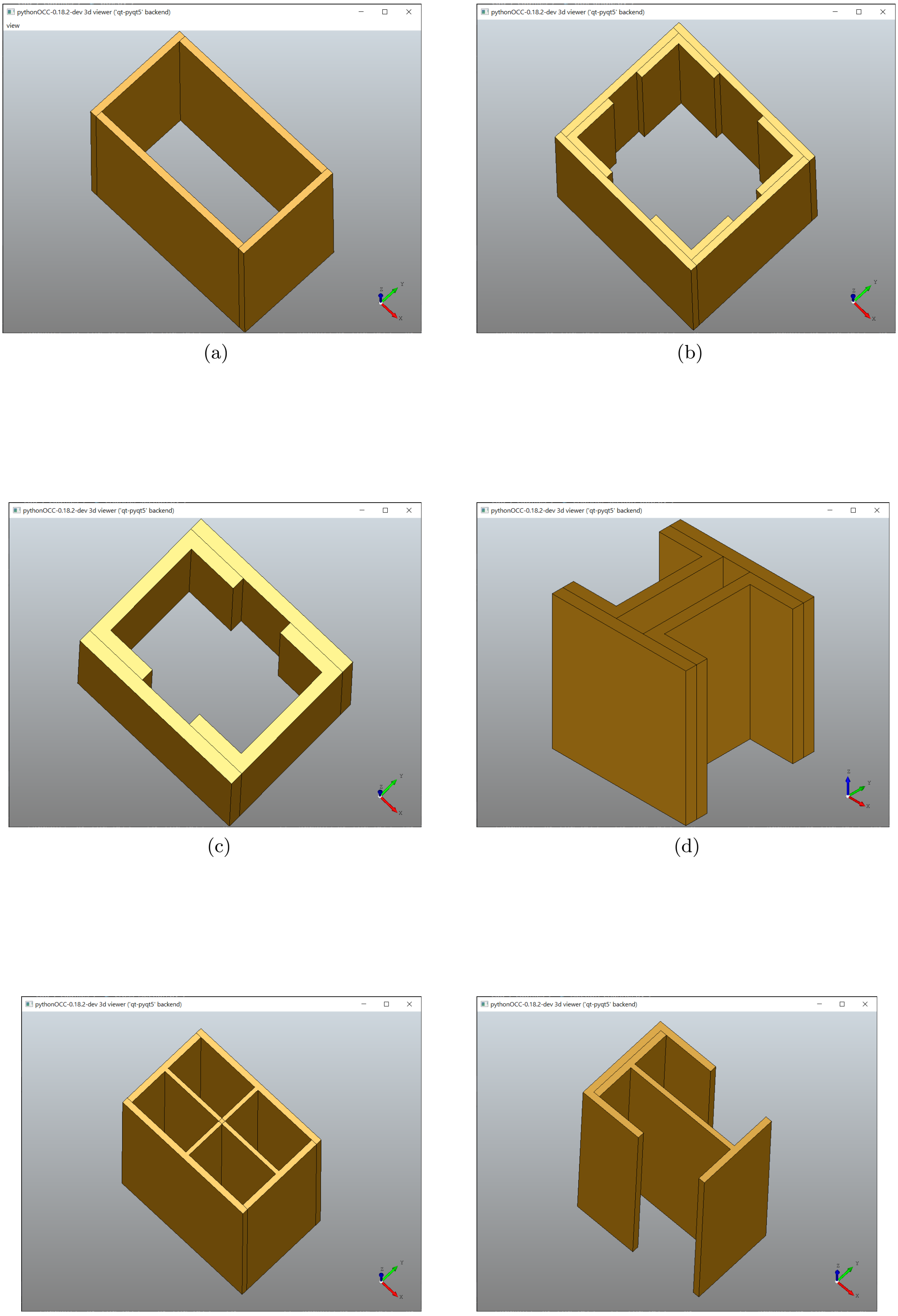
Chapter 2

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

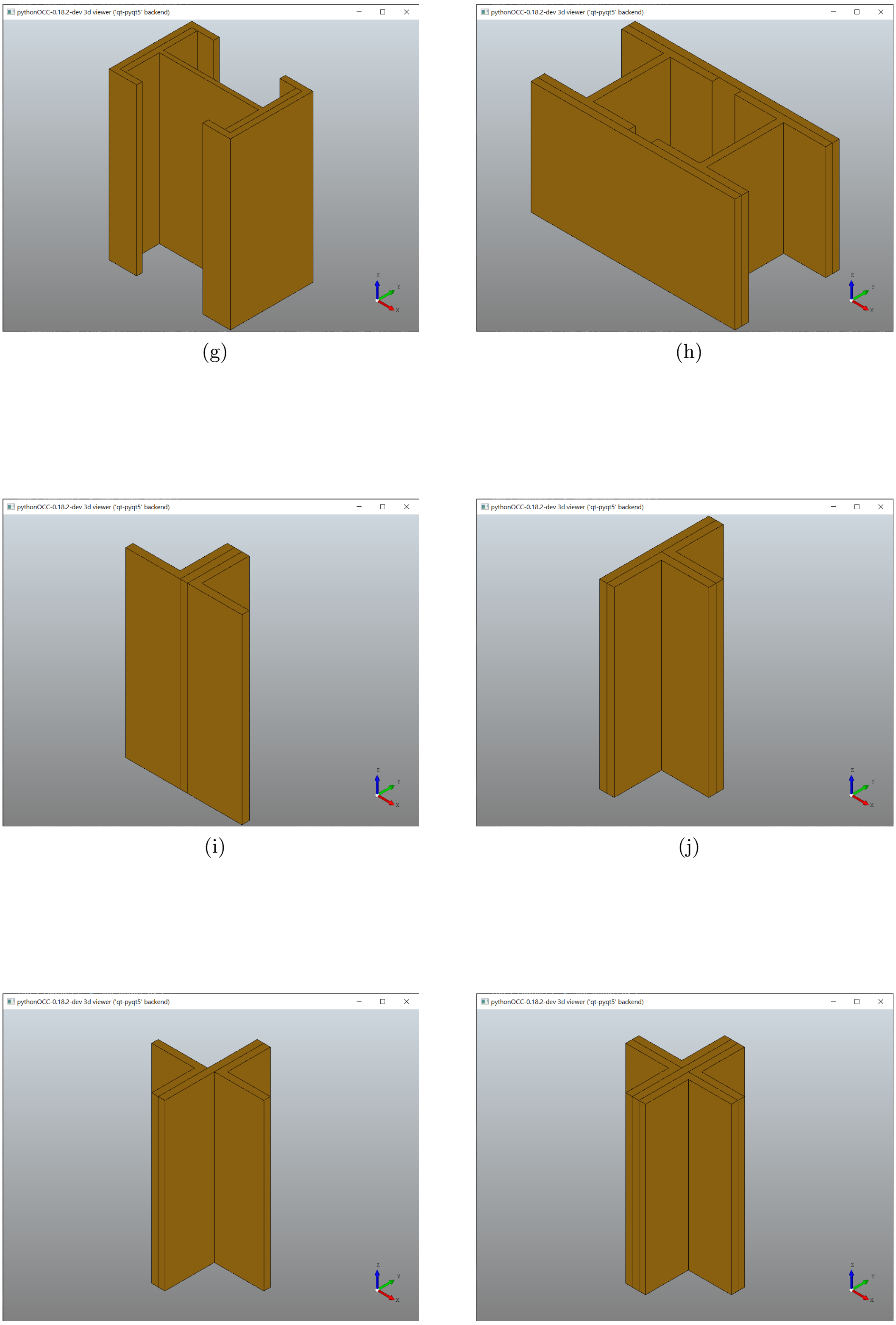
There are already some basic CAD modules are available in CAD.items folder like Isection, channel, angle, plate, nut etc. I have used those premade modules to create now modules for section modeller. There are around twelve CAD modules are created for section modeller which includes channel section (side-side & 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.



(a) (b)

(c) (d)

(e) (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.

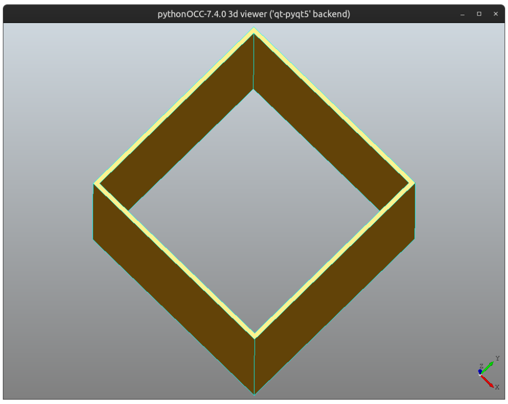


Figure 2.-3: square hollow section

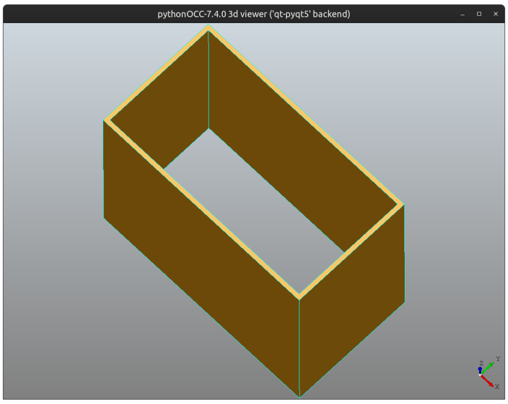


Figure 2.-2: rectangle hollow section

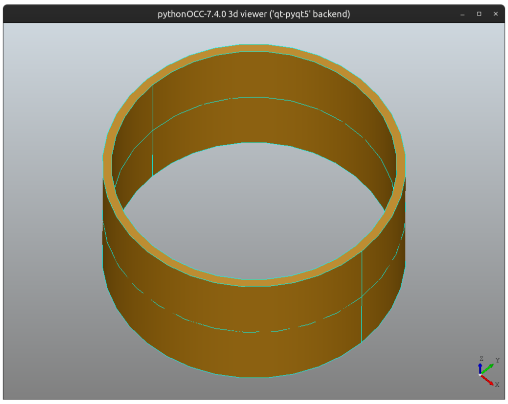
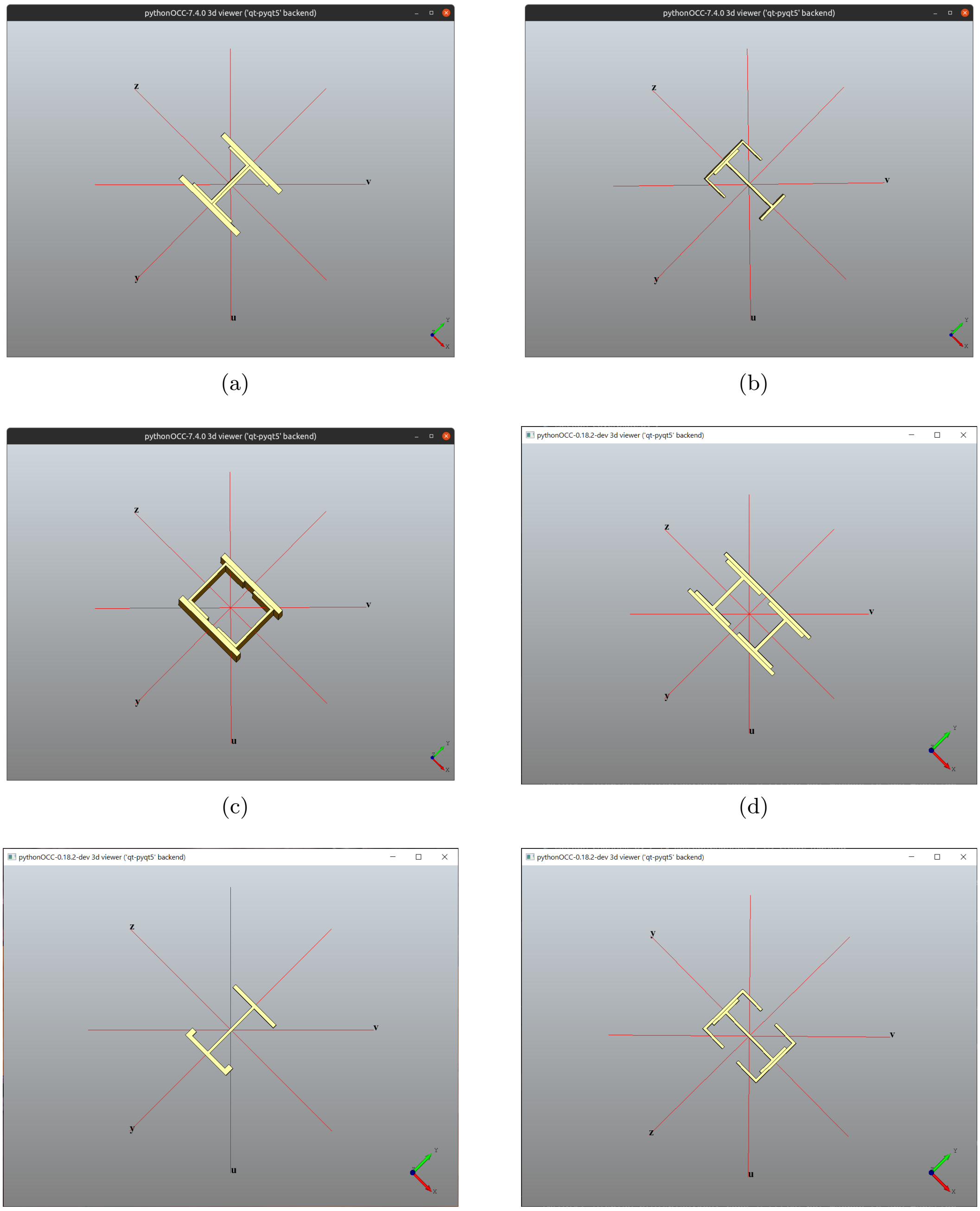


Figure 2.-1: circular hollow section

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.


pythonOCC-7.4.0 3d viewer ('qt-pyqt5' backend)
pythonOCC-0.18.2-dev 3d viewer ('qt-pyqt5' backend)
z
v
y
u

(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.

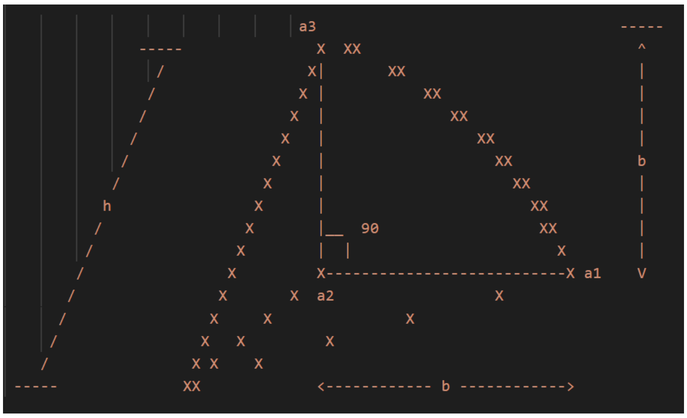


Figure 2.1: ASCII diagram of quarter cone module

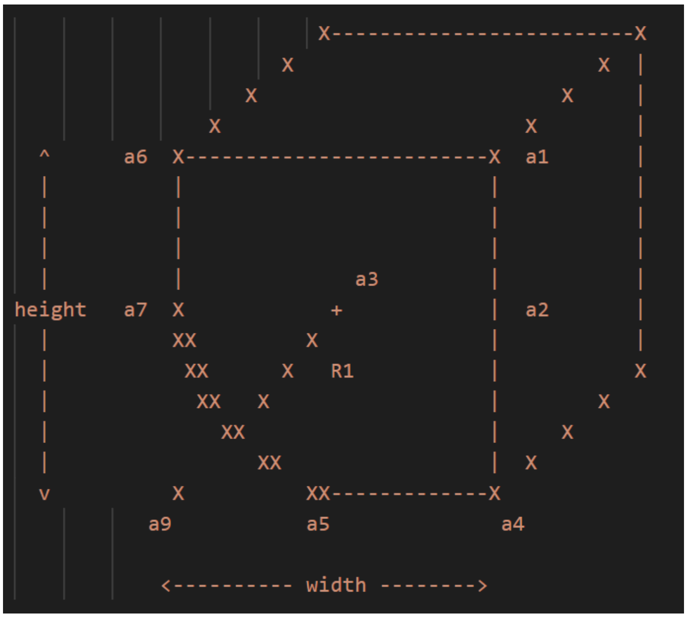


Figure 2.2: ASCII diagram of notch module

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.

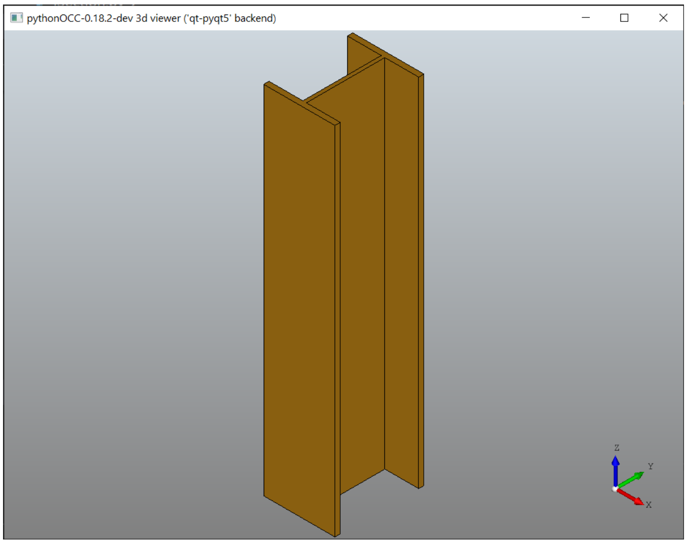


Figure 3.1: Isection 3D module

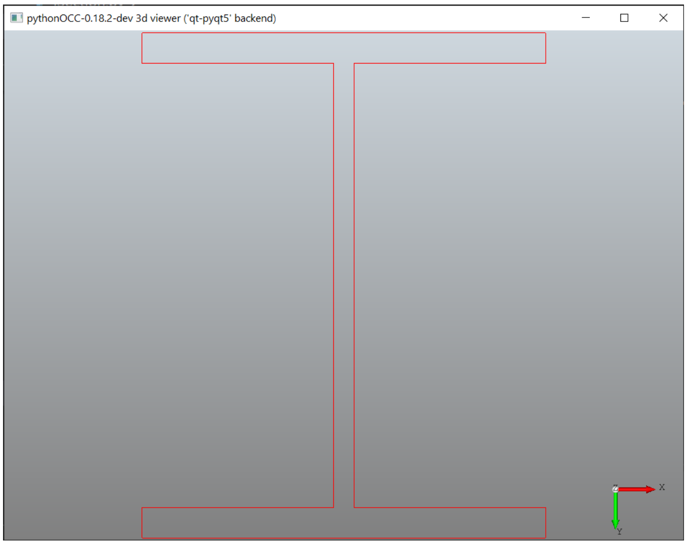


Figure 3.2: Top view of Isection module

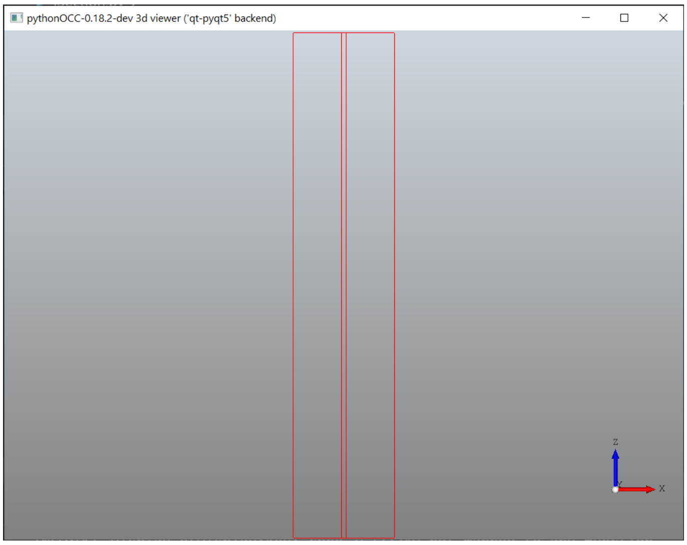


Figure 3.3: Front view of Isection module

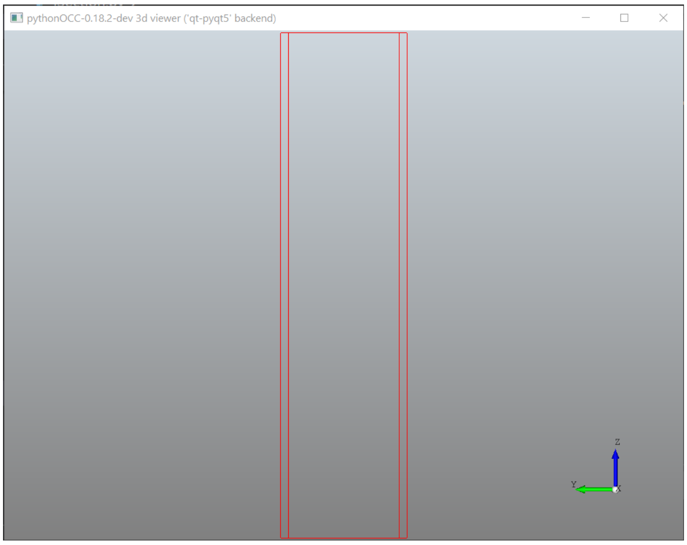


Figure 3.4: side view of Isection module

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

    def compute_params(self):
        self.plate1.compute_params()
        self.plate2.compute_params()
        self.plate3.compute_params()
        self.plate4.compute_params()

    def create_model(self):
        prism1 = self.plate1.create_model()
        prism2 = self.plate2.create_model()
        prism3 = self.plate3.create_model()
        prism4 = self.plate4.create_model()

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

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

        z_points = [numpy.array([-offset,0.,self.H/2]),
        ↪  numpy.array([offset,0.,self.H/2])]
        line.append(makeEdgesFromPoints(z_points))

        y_points = [numpy.array([0.,-offset,self.H/2]),
        ↪  numpy.array([0,offset,self.H/2])]
        line.append(makeEdgesFromPoints(y_points))

        u_points = [numpy.array([-uvoffset,uvoffset,self.H/2]),
        ↪  numpy.array([uvoffset,-uvoffset,self.H/2])]
        line.append(makeEdgesFromPoints(u_points))

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

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

        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,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 *
```

```python
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)

    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()
        self.plate3.compute_params()
        self.plate4.compute_params()

    def create_model(self):
        prism1 = self.angle1.create_model()
        prism2 = self.angle2.create_model()
        prism3 = self.angle3.create_model()
        prism4 = self.angle4.create_model()


        prism5 = self.plate1.create_model()
        prism6 = self.plate2.create_model()
        prism7 = self.plate3.create_model()
        prism8 = self.plate4.create_model()

        prism = BRepAlgoAPI_Fuse(prism1, prism2).Shape()
        prism = BRepAlgoAPI_Fuse(prism, prism3).Shape()
        prism = BRepAlgoAPI_Fuse(prism, prism4).Shape()
        prism = BRepAlgoAPI_Fuse(prism, prism5).Shape()
        prism = BRepAlgoAPI_Fuse(prism, prism6).Shape()
        prism = BRepAlgoAPI_Fuse(prism, prism7).Shape()
        prism = BRepAlgoAPI_Fuse(prism, prism8).Shape()
        return prism

    def rotate(self, points, x):
        rotated_points = []
        rmatrix = numpy.array([[numpy.cos(x), -numpy.sin(x), 0],
                              [numpy.sin(x), numpy.cos(x), 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)

        z_points = [numpy.array([-offset,0.,self.H/2]),
        ↪   numpy.array([offset,0.,self.H/2])]
        line.append(makeEdgesFromPoints(z_points))

        y_points = [numpy.array([0.,-offset,self.H/2]),
        ↪  numpy.array([0,offset,self.H/2])]
        line.append(makeEdgesFromPoints(y_points))

        u_points = [numpy.array([-uvoffset,uvoffset,self.H/2]),
        ↪  numpy.array([uvoffset,-uvoffset,self.H/2])]
        line.append(makeEdgesFromPoints(u_points))

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

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

        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,message_color=(0,0,0))

    #def view_markings():
     #   display_lines(lines, m_pnt, labels)

    #add_menu('view')
    #add_function_to_menu('view',view_markings)

    l = 40
    l1 = 50
    a = 15
    b = 15
    t = 2
    t1 = 2
    s = l  - 2*t1
    s1 = l1 - 2*t1 - 2*t
    H = 50


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

    box_angle = BoxAngle(a, b, t, l, t1, l1, H, s, s1)
    _place = box_angle.place(origin, uDir, wDir)
    point = box_angle.compute_params()
    prism = box_angle.create_model()
    lines, m_pnt, labels = box_angle.create_marking()
    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()
        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))

        y_points = [numpy.array([0.,-offset+self.D/2,self.H/2]),
        ↪  numpy.array([0,offset+self.D/2,self.H/2])]
        line.append(makeEdgesFromPoints(y_points))

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

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

        middel_pnt =
        ↪  [[-offset,self.D/2,self.H/2],[0,-offset+self.D/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,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))

        y_points = [numpy.array([0.,-offset+self.D/2,self.H/2]),
        ↪   numpy.array([0,offset+self.D/2,self.H/2])]
        line.append(makeEdgesFromPoints(y_points))

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

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

        middel_pnt =
        ↪   [[-offset,self.D/2,self.H/2],[0,-offset+self.D/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))

    B = 20
    T = 4
    D = 40
    t = 4
    t1 = 4
    s = 10
    l = s + 2*B
    H = 50

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

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

cross isection

```
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 = []
        labels = ["z","y","u","v"]
        offset = 100
        uvoffset = offset/numpy.sqrt(2)

        z_points = [numpy.array([-offset,0.,self.H/2]),
         ↪  numpy.array([offset,0.,self.H/2])]
        line.append(makeEdgesFromPoints(z_points))

        y_points = [numpy.array([0.,-offset,self.H/2]),
         ↪  numpy.array([0,offset,self.H/2])]
        line.append(makeEdgesFromPoints(y_points))

        u_points = [numpy.array([-uvoffset,uvoffset,self.H/2]),
         ↪  numpy.array([uvoffset,-uvoffset,self.H/2])]
        line.append(makeEdgesFromPoints(u_points))

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

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

        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,message_color=(0,0,0))

    B = 50
    T = 3
    D = 70
    t = 2
    H = 100
    d = (B - 2*T - t)/2
    s = (D - t)/2

    CrossISec = cross_isection(D, B, T, t, H, s, d)

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

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

```python
            return prism

    def rotateZ(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)

        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))

        y_points = [numpy.array([0.,-offset+self.d/2,self.H/2]),
        ↪  numpy.array([0,offset+self.d/2,self.H/2])]
        line.append(makeEdgesFromPoints(y_points))

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

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

        middel_pnt =
        ↪  [[-offset,self.d/2,self.H/2],[0,-offset+self.d/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))

    B = 20
    T = 2
    D = 40
    t = 1.5
    T1 = 2
    t1 = 2
    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)
        origin2 = numpy.array([0.,(self.D+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.Isection1.compute_params()
        self.Isection2.compute_params()
        self.Plate1.compute_params()
        self.Plate2.compute_params()

    def create_model(self):

        prism1 = self.Isection1.create_model()
        prism2 = self.Isection2.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 create_marking(self):
        middel_pnt = []
        line = []
        labels = ["z","y","u","v"]
        offset = 100
        uvoffset = offset/numpy.sqrt(2)

        #b = self.B/2+self.s/2
        z_points = [numpy.array([-offset,0.,self.H/2]),
        ↪  numpy.array([offset,0.,self.H/2])]
        line.append(makeEdgesFromPoints(z_points))

        y_points = [numpy.array([0,-offset,self.H/2]),
        ↪  numpy.array([0,offset,self.H/2])]
        line.append(makeEdgesFromPoints(y_points))

        u_points = [numpy.array([-uvoffset,uvoffset,self.H/2]),
        ↪  numpy.array([uvoffset,-uvoffset,self.H/2])]
        line.append(makeEdgesFromPoints(u_points))

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

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

        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))

    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

        self.angle1 = Angle(H, a, b, t, 0, 0)
        self.angle2 = Angle(H, a, b, 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
        origin1 = numpy.array([self.t1/2, 0., 0.])
        self.angle1.place(origin1, self.uDir, self.wDir)
        origin2 = numpy.array([self.t1/2, 0., 0.])
        self.angle2.place(origin2, 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)
        self.plate1.compute_params()
        #self.plate2.compute_params()

    def create_model(self):
        prism1 = self.angle1.create_model()
        prism2 = self.angle2.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 rotate(self, points, x):
        rotated_points = []
        rmatrix = numpy.array([[numpy.cos(x), -numpy.sin(x), 0],
                               [numpy.sin(x), numpy.cos(x), 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)

        z_points = [numpy.array([-offset,0.,self.H/2]),
        ↪   numpy.array([offset,0.,self.H/2])]
        line.append(makeEdgesFromPoints(z_points))

        y_points = [numpy.array([0.,-offset,self.H/2]),
        ↪   numpy.array([0,offset,self.H/2])]
        line.append(makeEdgesFromPoints(y_points))

        u_points = [numpy.array([-uvoffset,uvoffset,self.H/2]),
        ↪   numpy.array([uvoffset,-uvoffset,self.H/2])]
        line.append(makeEdgesFromPoints(u_points))

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

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

        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 = 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 = StarAngle2(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 with four 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 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()
        prism2 = self.angle2.create_model()
        prism3 = self.angle3.create_model()
        prism4 = self.angle4.create_model()


        prism5 = self.plate1.create_model()
        #prism6 = self.plate2.create_model()

        prism = BRepAlgoAPI_Fuse(prism1, prism2).Shape()
        prism = BRepAlgoAPI_Fuse(prism, prism3).Shape()
        prism = BRepAlgoAPI_Fuse(prism, prism4).Shape()
        prism = BRepAlgoAPI_Fuse(prism, prism5).Shape()
        #prism = BRepAlgoAPI_Fuse(prism, prism6).Shape()
        return prism


    def rotate(self, points, x):
        rotated_points = []
        rmatrix = numpy.array([[numpy.cos(x), -numpy.sin(x), 0],
                              [numpy.sin(x), numpy.cos(x), 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)

        z_points = [numpy.array([-offset,0.,self.H/2]),
        ↪   numpy.array([offset,0.,self.H/2])]
        line.append(makeEdgesFromPoints(z_points))

        y_points = [numpy.array([0.,-offset,self.H/2]),
        ↪   numpy.array([0,offset,self.H/2])]
        line.append(makeEdgesFromPoints(y_points))

        u_points = [numpy.array([-uvoffset,uvoffset,self.H/2]),
        ↪   numpy.array([uvoffset,-uvoffset,self.H/2])]
        line.append(makeEdgesFromPoints(u_points))

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

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

        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 = 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)
        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)
        origin3 = numpy.array([0., self.a/2, 0.])
        self.plate1.place(origin3, self.uDir, self.wDir)

    def compute_params(self):
        self.angle1.computeParams()
        self.angle2.computeParams()
        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)

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

        y_points = [numpy.array([0.,-offset+self.t/2,self.H/2]),
        ↪   numpy.array([0,offset+self.t/2,self.H/2])]
        line.append(makeEdgesFromPoints(y_points))

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

        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)

        z_points = [numpy.array([-offset,0.,self.H/2]),
        ↪  numpy.array([offset,0.,self.H/2])]
        line.append(makeEdgesFromPoints(z_points))

        y_points = [numpy.array([0.,-offset,self.H/2]),
        ↪  numpy.array([0,offset,self.H/2])]
        line.append(makeEdgesFromPoints(y_points))

        u_points = [numpy.array([-uvoffset,uvoffset,self.H/2]),
        ↪  numpy.array([uvoffset,-uvoffset,self.H/2])]
        line.append(makeEdgesFromPoints(u_points))

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

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

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

bulid up section

```python
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
        self.d6 = self.sec_origin + ((self.B / 2.0) - self.b) * self.uDir +
        ↪  -((self.D / 2.0) - self.T) * self.vDir
        self.d4 = self.sec_origin + ((self.B / 2.0) - self.b) * self.uDir +
        ↪  -((self.D / 2.0) -self.T - self.d) * self.vDir
        self.b4 = self.sec_origin + (self.B / 2.0) * self.uDir + -((self.D /
        ↪  2.0) - self.T - self.d) * self.vDir
        self.c4 = self.sec_origin + (self.B / 2.0) * self.uDir + -(self.D /
        ↪  2.0) * self.vDir




        self.points = [self.a1, self.b1, self.c1,
                       self.c2, self.b2, self.a2,
                       self.a3, self.d7, self.d5,
                       self.b3, self.c3, self.c4,
                       self.b4, self.d4, self.d6,
                       self.a4]
        print(self.d4)

    def create_model(self):

        edges = makeEdgesFromPoints(self.points)
        wire = makeWireFromEdges(edges)
        aFace = makeFaceFromWire(wire)
        extrudeDir = self.length * self.wDir  # extrudeDir is a numpy array
        prism = makePrismFromFace(aFace, extrudeDir)

        return prism

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

        z_points = [numpy.array([-offset,0.,self.length/2]),
        ↪  numpy.array([offset,0.,self.length/2])]
        line.append(makeEdgesFromPoints(z_points))

        y_points = [numpy.array([0.,-offset,self.length/2]),
        ↪  numpy.array([0,offset,self.length/2])]
        line.append(makeEdgesFromPoints(y_points))

        u_points = [numpy.array([-uvoffset,uvoffset,self.length/2]),
        ↪  numpy.array([uvoffset,-uvoffset,self.length/2])]
        line.append(makeEdgesFromPoints(u_points))

        v_points = [numpy.array([-uvoffset,-uvoffset,self.length/2]),
        ↪  numpy.array([uvoffset,uvoffset,self.length/2])]
        line.append(makeEdgesFromPoints(v_points))

        middel_pnt =
        ↪  [[-offset,0,self.length/2],[0,-offset,self.length/2],[uvoffset,-uvoffset,self.length

```

```python
        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))

    B = 40
    T = 3
    D = 50
    t = 2
    P = 8
    Q = 4
    H = 100

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

    TISec = TISection(D, B, T, t, P, Q, H)
    _place = TISec.place(origin, uDir, shaftDir)
    point = TISec.compute_params()
    prism = TISec.create_model()
    lines, m_pnt, labels = TISec.create_marking()
    display.DisplayShape(prism, update=True)
    display_lines(lines, m_pnt, labels)
    display.DisableAntiAliasing()
    start_display()
```