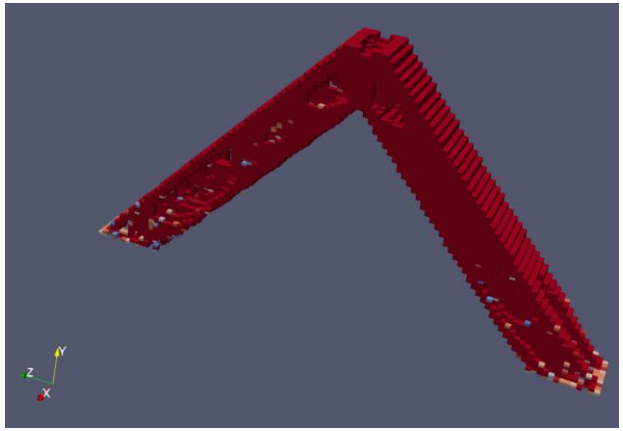
Optimisation Project

REVIEW OF THE STRUCTURAL OPTIMISATION PROJECT

TOPIC: **3D COAT HANGER**



RISHYAVANDHAN VENKATESAN

Github: https://github.com/RISHYAVANDHAN/Structural-Optimisation

Immatrikulation Nummer: 23356687

Computational Engineering

Short Summary of the Project:

Objective:

- The study aimed to develop a 3D topology optimization model to study the topology of the 3D coat hanger using OpenCFS.
- I aimed to obtain a structurally optimized 3D coat hanger by testing different options available.

Optimization Methods:

- **Solid Isotropic Material with Penalization (SIMP):** A density-based method for material distribution optimization.

Process Overview:

- Mesh the 3D coat hanger.
- Set up simulation using OpenCFS in xml (3dhanger.xml) and view it in Paraview.
- Analyze the results and try out different options available in hand.
- Try to reach an optimal design (if there´s one)

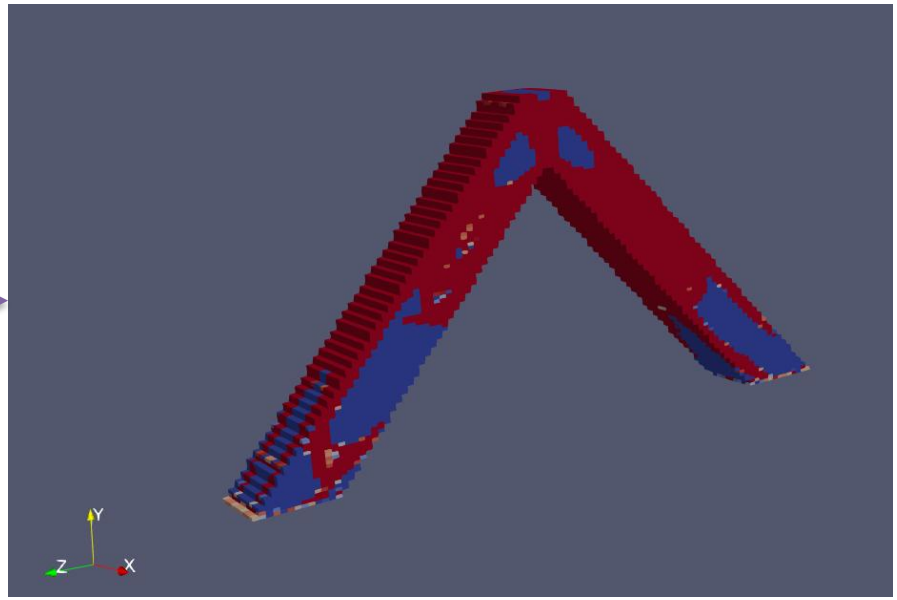
Results:

- Optimised shape is obtained. (spoiler alert!!)

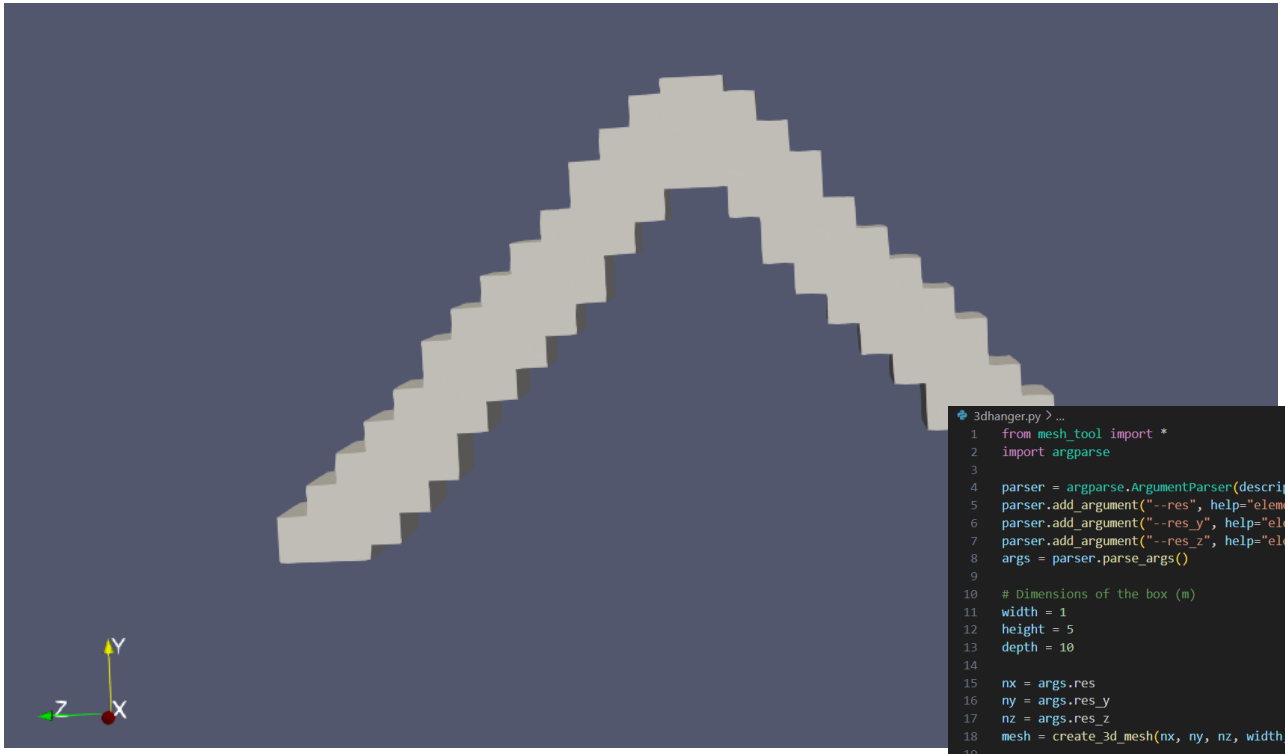
BASIC OUTLINE OF THE PROJECT

OBJECTIVE

RESULT



3D Hanger Meshing



Slope: ½ of depth

Top surface thickness = 0,1 * depth

3dhanger.py > ...

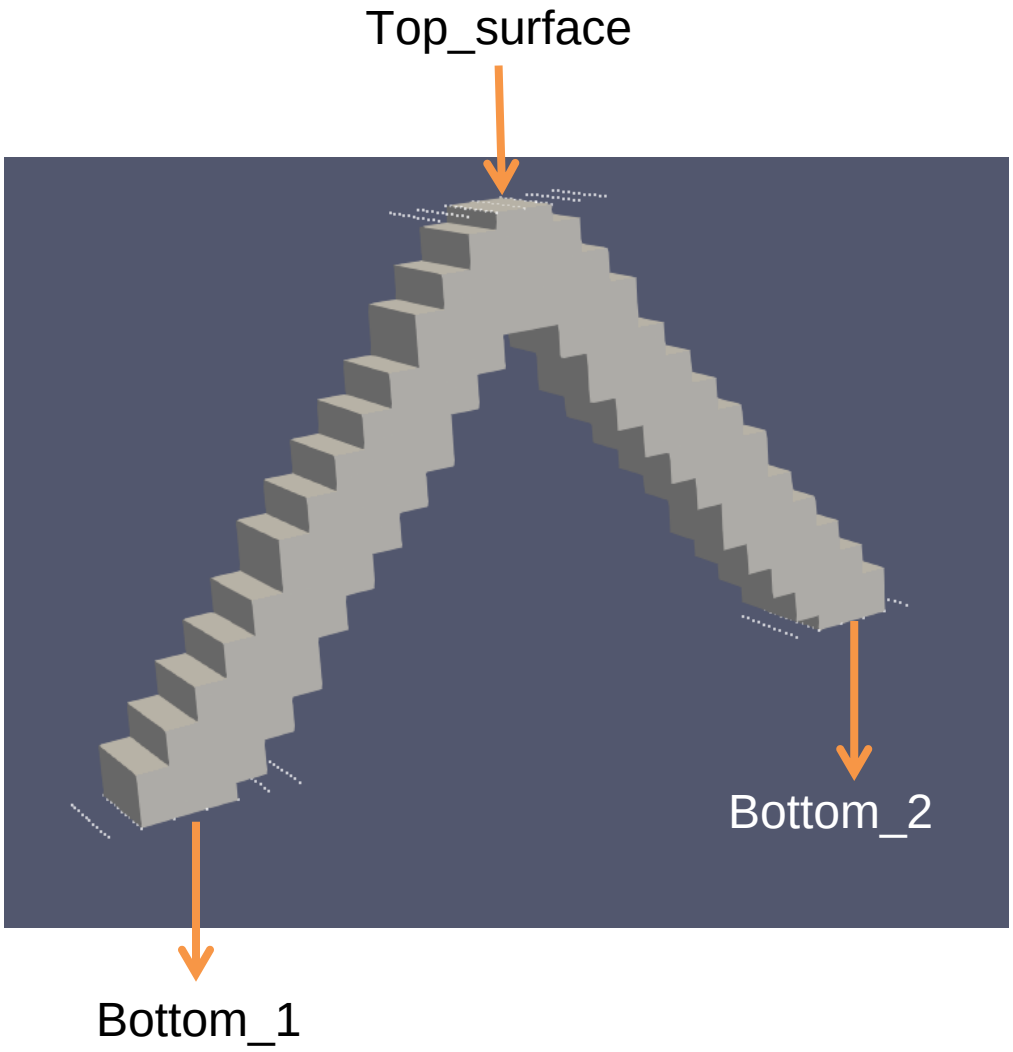
```python
from mesh_tool import *
import argparse

parser = argparse.ArgumentParser(description="generate a basic mesh for a 3D coat hanger.")
parser.add_argument("--res", help="elements in x-direction", type=int, required = True )
parser.add_argument("--res_y", help="elements in x-direction", type=int, required = True )
parser.add_argument("--res_z", help="elements in x-direction", type=int, required = True )
args = parser.parse_args()

# Dimensions of the box (m)
width = 1
height = 5
depth = 10

nx = args.res
ny = args.res_y
nz = args.res_z
mesh = create_3d_mesh(nx, ny, nz, width, height, depth)

y_slope = height / (depth / 2) + 0.1 * width
solid_gap = 0.025 * height
mech_gap = 0.3 * height

for e in mesh.elements:
    x, y, z = mesh.calc_barycenter(e)

    if z <= depth / 2:
        diagonal_y = z * y_slope
    else:
        diagonal_y = (depth - z) * y_slope

    if (y < (diagonal_y - mech_gap)):
        e.region = 'solid'
    elif diagonal_y - mech_gap <= y < diagonal_y - solid_gap:
        e.region = 'mech'
    else:
        e.region = 'void'

f = '3dhanger_' + str(args.res) + "_" + str(args.res_y) + "_" + str(args.res_z) +'.mesh'
write_ansys_mesh(mesh, f)
print('Mesh file created:', f)
```

Basic Outline:
1m x 5m x 10m box

“Mech” region representing a “A” shape.

Nodes and other setups:

```
<nodeList>
  <nodes name="top_surface">
    <list>
      <freeCoord comp="x" start="0.0" stop="1.0" inc="0.01" />
      <freeCoord comp="z" start="4.0" stop="6.0" inc="0.01" />
      <fixedCoord comp="y" value="5" />
    </list>
  </nodes>
  <nodes name="bottom_1">
    <list>
      <freeCoord comp="x" start="0.0" stop="1.0" inc="0.01" />
      <freeCoord comp="z" start="0.0" stop="1.5" inc="0.01" />
      <fixedCoord comp="y" value="0.0" />
    </list>
  </nodes>
  <nodes name="bottom_2">
    <list>
      <freeCoord comp="x" start="0.0" stop="1.0" inc="0.01" />
      <freeCoord comp="z" start="8.0" stop="10.0" inc="0.01" />
      <fixedCoord comp="y" value="0.0" />
    </list>
  </nodes>
</nodeList>
<bcsAndLoads>
  <fix name="top_surface">
    <comp dof="x" />
    <comp dof="y" />
    <comp dof="z" />
  </fix>
  <force name="bottom_1">
    <comp dof="y" value="-1" />
  </force>
  <force name="bottom_2">
    <comp dof="y" value="-1" />
  </force>
</bcsAndLoads>
```

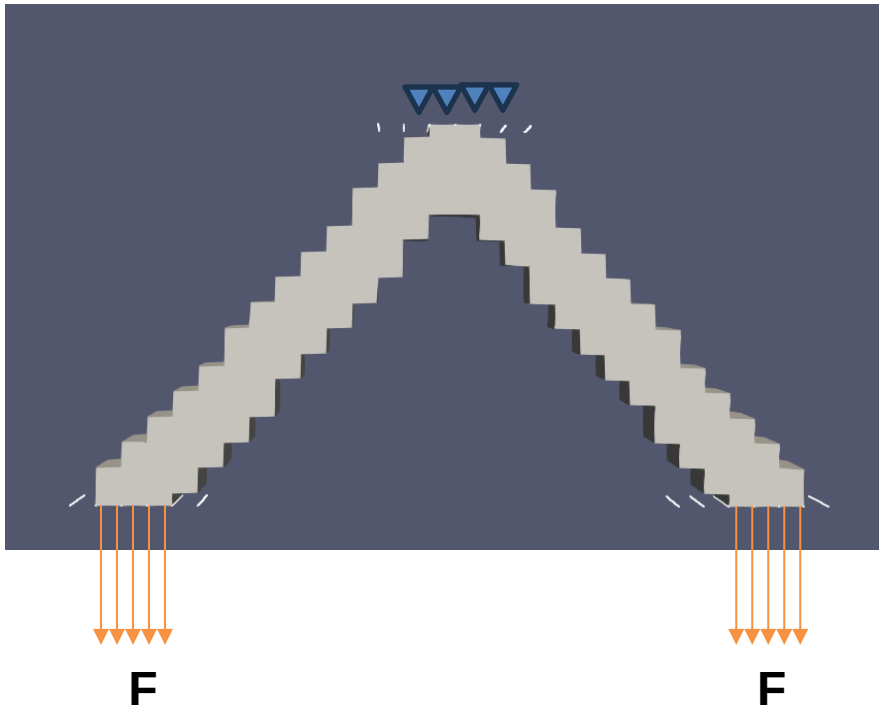


Load cases:

- Fixing – top_surface
  Load -- bottom_1 and bottom_2 in (-y) direction.

- Load – top_surface
  Fixing – parts of top_surface ; but this configuration didn't yield any result worth mentioning, why?

```
<bcsAndLoads>
  <fix name="top_surface">
    <comp dof="x" />
    <comp dof="y" />
    <comp dof="z" />
  </fix>
  <force name="bottom_1">
    <comp dof="y" value="-1" />
  </force>
  <force name="bottom_2">
    <comp dof="y" value="-1" />
  </force>
</bcsAndLoads>
```



Process Walkthrough:

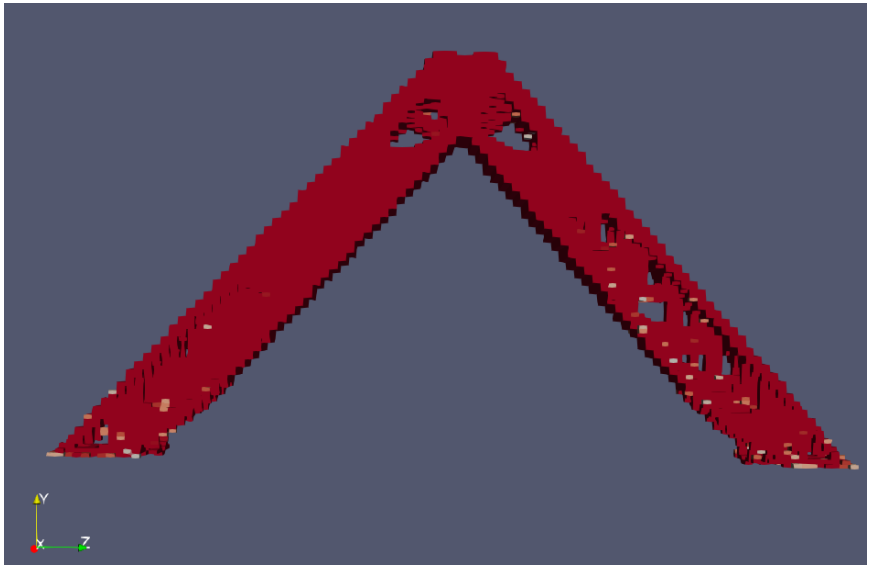
- Run test to benchmark – checking up initial setups like load and boundary conditions
- Improve resolution to improve results
- Experimenting with FEM solvers, every solver do not solve every problem, figuring out a solver is very important.
- Main ingredient of optimization == Optimizers, switch and check which one yields the least compliance;
  least compliance ➔ better stiffness
- Increasing active and inactive constraints to observe and analyze different properties that influence the structure.
- Experimenting the volume constraint (the primary one) to fix the amount of material to be used.
- Setting the density filter value by experimenting the filter neighborhood.
- Once this is done, change load case (Started with first, then move onto the second one).

```
<Starting cfsSimulation>
  <domain geometryType="3d">
    <regionList>
      <Define regions that exist everywhere/>
    </regionList>
    <nodeList>
      <Define Nodes>
    </nodeList>
  </domain>

    <pdeList>
      <Setting up the regions involved in simulation
        <bcsAndLoads>
          <Define all the Boundary conditions and loads
        </bcsAndLoads>

        <storeResults>
          <Store different results you wish to analyze
        </storeResults>
      </mechanic>
    </pdeList>

    <linearSystems>
        <Solving the linear system using a FEM solver
    </linearSystems>
  </sequenceStep>

  <optimization>
    <Setting up the important functions, constraints, optimizers
       and filters available to experiment>
  </optimization>

</cfsSimulation>
```

Initial Setup simulation

Benchmark:

Resolution	– (10, 100, 100) = (x, y, z)
FEM Solver	– Cholmod (Cholskey factorisation)
Optimizer	– OCM
Volume	– 0.5
Filter neighborhood	– 1.3

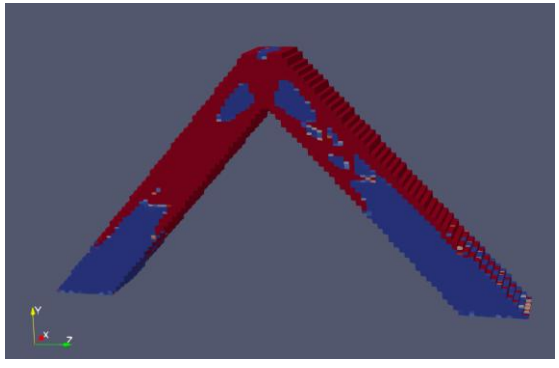



```
C:\Users\srini\test\Project>postproc benchmark.plot.dat
#  (1)        (2)      (3)          (4)           (5)              (6)              (7)    (8)          (9)      (10)
# iter compliance duration plain_volume plain_greyness physical_greyness physical_volume lambda lambda_iters   problem
 100.0    1056.07    1.339     0.499995        0.01643          0.241925        0.379107 2211.0         12.0 benchmark
```

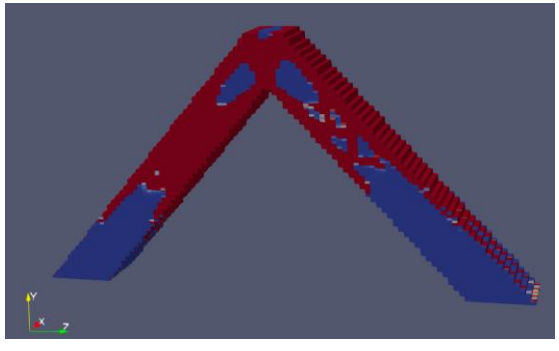
Improving the resolution

Idea: Find a resolution that has less computational load and gives a clearer picture.

(20, 100, 100)



(40, 100, 100)

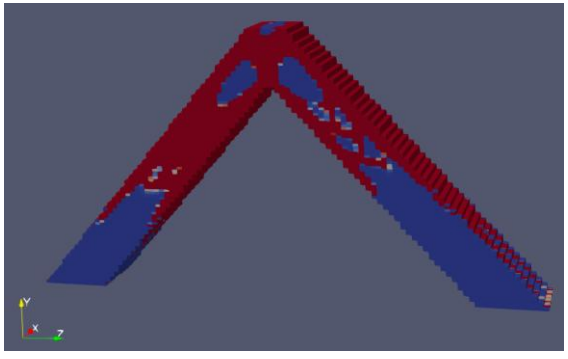


Comparing the different quantities/attributes, It can be observed that (60,100,100) is a better resolution with less greyness, more volume and the least compliance, it takes more time, which is the only drawback, we get equivalent result with (40,100,100) with the second least greyness, higher volume and 1/3rd of the duration.

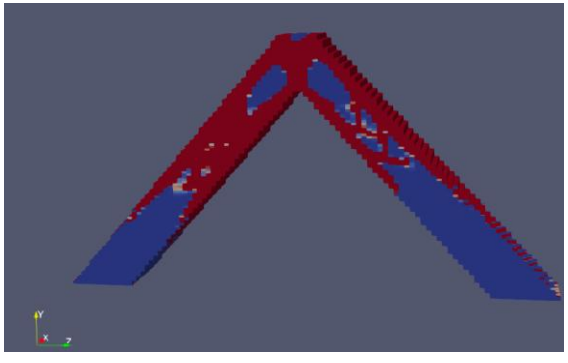
- **Compliance (1074.34)**: This has the lowest compliance value, indicating the stiffest structure among the four.
- **Plain Volume (0.500006)** and **Physical Volume (0.369049)**: The volume is very close to the other cases, with no significant loss in material efficiency.
- **Greyness Values**: The plain greyness (0.0192406) and physical greyness (0.246119) are low, suggesting a highly optimized material distribution.

So, strikes the best balance between material efficiency and stiffness without compromising computational cost significantly

(60, 100, 100)



(80, 100, 100)



```
C:\Users\srini\test\Project>postproc mesh_*.plot.dat
#  (1)       (2)      (3)         (4)           (5)              (6)              (7)     (8)         (9)                                  (10) (11)    (12)  (13)
# iter compliance duration plain_volume plain_greyness physical_greyness physical_volume  lambda lambda_iters                          problem mesh dhanger     _
 100.0    1079.66    3.194     0.499992       0.0208932         0.247967         0.366966 2390.29         12.0 mesh_3dhanger_20_100_100.mesh    3      20 100.0
 100.0    1074.34    7.907     0.500006       0.0192406         0.246119         0.369049 2346.04         12.0 mesh_3dhanger_40_100_100.mesh    3      40 100.0
 100.0    1072.83    22.08          0.5       0.0186324         0.245748         0.369629 2333.83          1.0 mesh_3dhanger_60_100_100.mesh    3      60 100.0
 100.0    1073.83   22.422     0.500002        0.018461         0.246906         0.369152  2340.7         12.0 mesh_3dhanger_80_100_100.mesh    3      80 100.0
```

Experimenting with solvers:

Solvers checked:

- Cholmod (**CHOL**esky)
- Pardiso (**PAR**allel **DI**rect **SO**lver)
- CG (**C**onjugate **G**radient)

In terms of compliance and greyness there´s no difference between the solvers, but the time taken decides the choice, by that we can go with cg, but cg got stopped multiple times and pushed the ocm limit a bit more often than the other solvers.
So, we are going to proceed with Cholmod, the default solver used in every simulation, but at the end we will also compare it with pardiso for some cases to draw parallels and intercept any valuable information possible.

```
C:\Users\srini\test\Project>postproc solver_*.plot.dat
#  (1)        (2)      (3)          (4)            (5)              (6)              (7)      (8)          (9)           (10)
# iter compliance duration plain_volume plain_greyness physical_greyness physical_volume  lambda lambda_iters        problem
 100.0    1074.34    8.033     0.500006      0.0192406         0.246119         0.369049 2346.04         12.0      solver_cg
 100.0    1074.34    8.318     0.500006      0.0192406         0.246119         0.369049 2346.04         12.0 solver_cholmod
 100.0    1074.34    9.817     0.500006      0.0192406         0.246119         0.369049 2346.04         12.0 solver_pardiso
```

Experimenting with Optimizers:

Optimizers used:

- OCM - **O**ptimal **C**ontrol **M**ethods
- SNOPT - **S**parse **N**onlinear **OPT**imizer
- IPOPT - **I**nterior **P**oint **OPT**imizer
- MMA - **M**ethod of **M**oving **A**symptotes
- SCPIP - **S**equential **C**onvex **P**rogramming **I**nterior-**P**oint

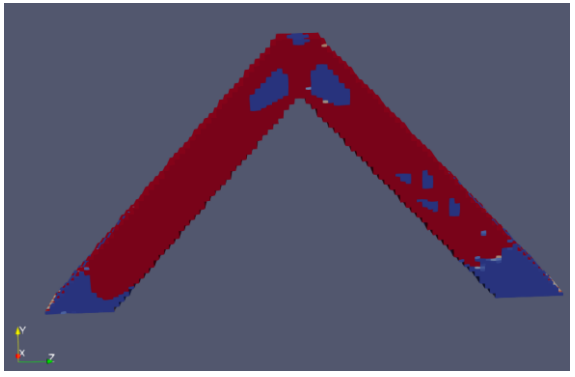
Insights:

- **Efficiency**: optimizer_mma achieves good results but at higher computational cost (duration and subproblem iterations).
- **Smoothness:** MMA seems to favor smoother designs (lower greyness).Penalty
- **Handling:** Higher lambda values for MMA might make it more aggressive in satisfying constraints.

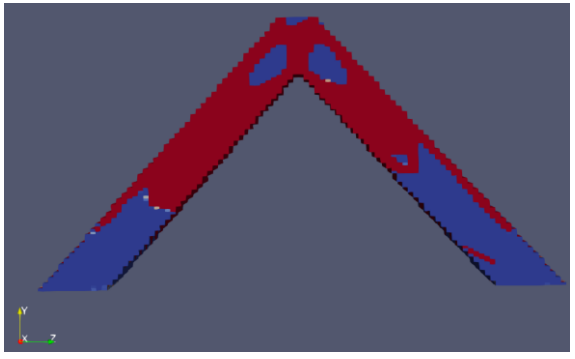
```
C:\Users\srini\test\Project>postproc optimizer_*.plot.dat
#  (1)       (2)     (3)          (4)           (5)              (6)             (7)   (8)           (9)       (10)        (11)        (12)        (13)        (14)                (15)    (16
)        (17)
# iter compliance duration plain_volume plain_greyness physical_greyness physical_volume  nObj        problem sub_prb_itr neg_asym_min neg_asym_max pos_asym_min pos_asym_max lambda_plain_volume  lambd
a lambda_iters
 100.0    1106.79    9.099         0.5       0.190015         0.280407        0.356312 107.0 optimizer_ipopt          0           0           0           0           0                   0
0           0
 100.0    1084.22   10.514    0.493261     0.00803949         0.241671        0.364398     0   optimizer_mma       20.0    -1151.25    0.993839   0.00477568      1153.25             2551.95
0           0
 100.0    1074.34    7.963    0.500006      0.0192406         0.246119        0.369049     0   optimizer_ocm          0           0           0           0           0                   0 2346.0
4        12.0
 100.0    1058.62    8.169    0.499989     0.00717342         0.238659        0.377743     0 optimizer_scpip          0           0           0           0           0                   0
0           0
 100.0    1074.34     7.89    0.500006      0.0192406         0.246119        0.369049     0 optimizer_snopt          0           0           0           0           0                   0 2346.0
4        12.0
```

Experimentation with volume constraints:

V = 0.3

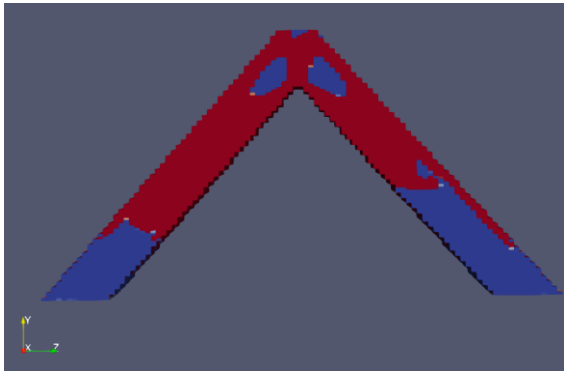


V = 0.4

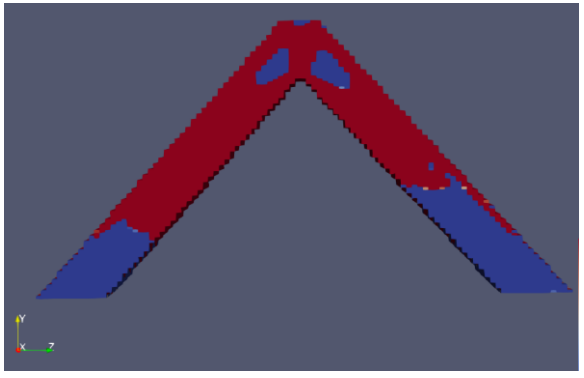


- **volconstr_0.3_mma_** uses less material but has the highest compliance (1185.08), making it less stiff.
- **volconstr_0.4_mma_** and **volconstr_0.6_mma_** offer moderate performance but still result in higher compliance than **volconstr_0.7_mma_** (1088.21 and 1036.55, respectively).
- **volconstr_0.7_mma_** achieves a **significantly stiffer** structure without excessive material use, while having **faster convergence times** (8.119 seconds) than the lower volume constraints.

V = 0.6



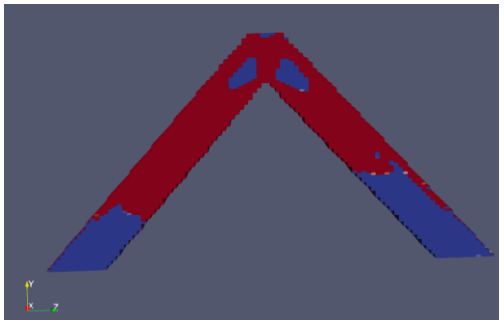
V = 0.7



# (1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)	(15)
# iter	compliance	duration	plain_volume	plain_greyness	physical_greyness	physical_volume	sub_prb_itr	neg_asym_min	neg_asym_max	pos_asym_min	pos_asym_max	lambda_plain_volume	problem	volconstr
100.0	1185.08	10.497	0.449025	0.0092833	0.221602	0.32794	20.0	-3362.69	0.997878	0.0056699	3362.7	3165.79	volconstr_0.3_mma_	0.3
100.0	1088.21	8.076	0.492918	0.0102116	0.244423	0.362787	20.0	-1650.15	0.996935	0.00514768	1651.88	2579.28	volconstr_0.4_mma_	0.4
100.0	1036.55	8.291	0.511925	0.00849551	0.239698	0.386014	20.0	-1641.79	0.998521	0.00535925	1643.79	2222.76	volconstr_0.6_mma_	0.6
100.0	995.782	8.119	0.530872	0.00796251	0.238172	0.407388	20.0	-2343.49	0.995678	0.00573588	2345.49	1942.38	volconstr_0.7_mma_	0.7

Experimenting with filter neighborhood:

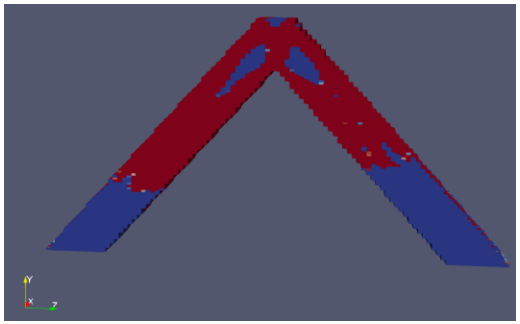
Maxedge = 1.3



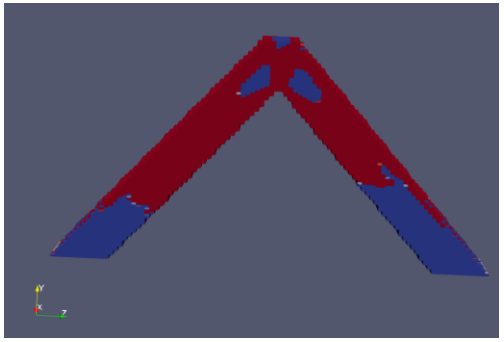
Considering all key factors (compliance, volume, greyness, and duration), **denfltr_maxedge_1.9** is the best choice because:

- Lowest compliance (984.338) → Best stiffness
- Least material usage (0.417955) → Most efficient
- Acceptable physical greyness (0.313216) → Manufacturable
- Solving time (8.224s) is competitive

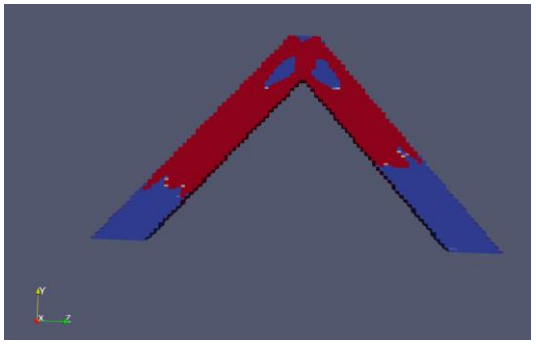
Maxedge = 1.9



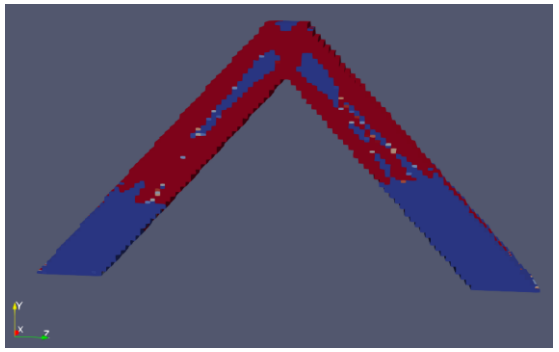
Maxedge = 1.5



Maxedge = 1.7



Maxedge = 2.1



```
C:\Users\srini\test\Project>postproc denfltr*.plot.dat
#  (1)       (2)      (3)      (4)          (5)            (6)              (7)             (8)          (9)          (10)         (11)          (12)          (13)                (14)                  (15)
# iter compliance duration plain_volume plain_greyness physical_greyness physical_volume sub_prb_itr neg_asym_min neg_asym_max pos_asym_min pos_asym_max lambda_plain_volume           problem maxedge
 100.0    995.782    8.119     0.530872     0.00796251         0.238172        0.407388         20.0     -2343.49     0.995678   0.00573588       2345.49              1942.38 denfltr_maxedge_1.3     1.3
 100.0    1005.59    8.139     0.543615      0.0113113         0.268061        0.402452         20.0     -3365.06     0.996981   0.00515154       3367.06               2050.0 denfltr_maxedge_1.5     1.5
 100.0     990.37    9.459     0.565204      0.0123762         0.293091        0.413046         20.0     -2372.78     0.993876   0.00566682       2372.79              1997.99 denfltr_maxedge_1.7     1.7
 100.0    984.338    8.224     0.578373      0.0150223         0.313216        0.417955         20.0     -2369.41     0.998968   0.00514395       2371.41              1986.39 denfltr_maxedge_1.9     1.9
 100.0    988.112    8.424     0.583491      0.0173541         0.329844         0.42075         20.0     -2363.31     0.987514   0.00764257       2363.32              2048.16 denfltr_maxedge_2.1     2.1
```

Introducing Heavyside projection filter:

Heaviside Filter in Topology Optimization. The Heaviside filter sharpens material distribution by pushing intermediate densities towards 0 (void) or 1 (solid), enhancing manufacturability. The parameter β (beta) controls the severe binarization impact.

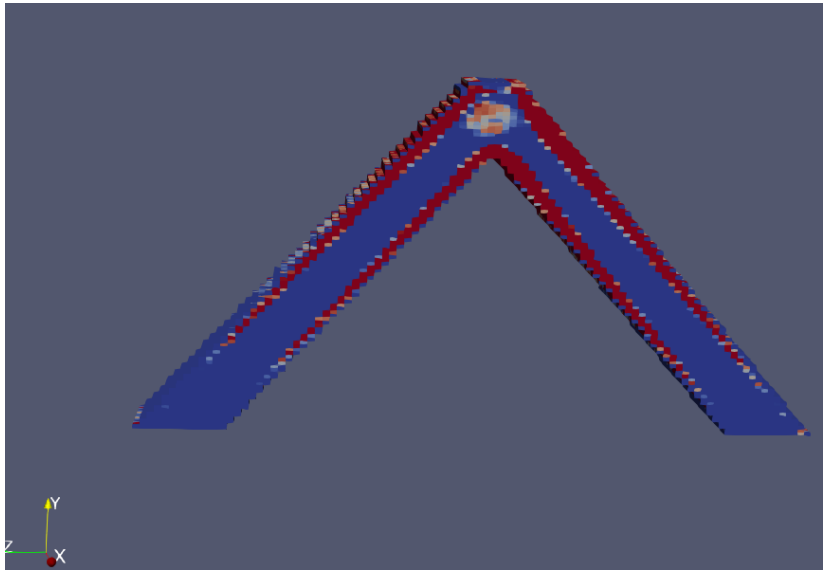
Low β indicates smoother transitions and more intermediate material.
High β indicates strong binarization, but may cause numerical instability.

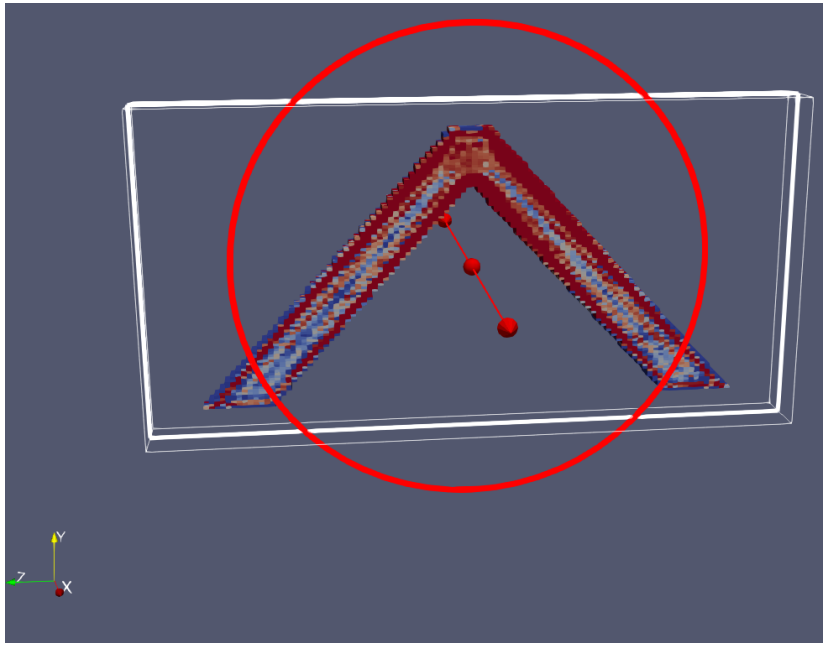
- β = 4.0 offers a balanced trade-off between compliance, greyness, and material volume.
- β = 8.0 gives lowest compliance but retains some intermediate densities.
- β = 16.0+ is too aggressive, causing collapse or excessive binarization.

```
C:\Users\srini\test\Project>postproc 3dhanger-beta*.plot.dat
#  (1)          (2)      (3)          (4)           (5)              (6)              (7)          (8)          (9)          (10)          (11)          (12)              (13)                (14) (15) (1
6)
# iter   compliance duration plain_volume plain_greyness physical_greyness physical_volume sub_prb_itr neg_asym_min neg_asym_max pos_asym_min pos_asym_max lambda_plain_volume           problem      be
ta
 100.0      1047.71   16.082      0.52048      0.0217604         0.352574          0.3801         20.0     -1663.27     0.993857   0.00499359      1665.27           2313.6  3dhanger-beta_1.0    3  1
.0
  50.0      703.784   12.764     0.675848       0.539315        0.00137212        0.999656          3.0     -46.6325     0.997901    0.0688774      46.6418         0.205886 3dhanger-beta_16.0    3 16
.0
  50.0      764.635   12.964     0.698678      0.0459076         0.393564        0.645005          5.0      -566.86     0.993924    0.0223549       566.87           487.738  3dhanger-beta_2.0    3  2
.0
  50.0 7037630000.0    0.793   0.00464159            0.0      -2.38447e-21            1e-07          1.0     -5.36284    -0.145919    0.155202      5.37212             1e+20 3dhanger-beta_32.0    3 32
.0
  50.0      730.337    12.69     0.688247       0.226197         0.422067         0.77613          4.0      -565.163     0.991294   0.0107233       565.172           205.329  3dhanger-beta_4.0    3  4
.0
  50.0      7961.05   12.689   0.00874822      0.0156329         0.136619        0.0584876         20.0       -3.633     0.149727   0.00475735      4.26398         3910540.0 3dhanger-beta_64.0    3 64
.0
  50.0      705.659   13.007     0.699825       0.269119        0.0996435         0.972593          2.0      -278.315     0.998979    0.0170409      278.324           19.2581  3dhanger-beta_8.0    3  8
.0
```

The optimal setting:

Mesh : (40,100,100)
Solver: Cholmod
Optimizer: MMA
Volume constraint: 0.7
Filter neighbborhood: maxedge 1.9
Heavyside filter: beta = 8



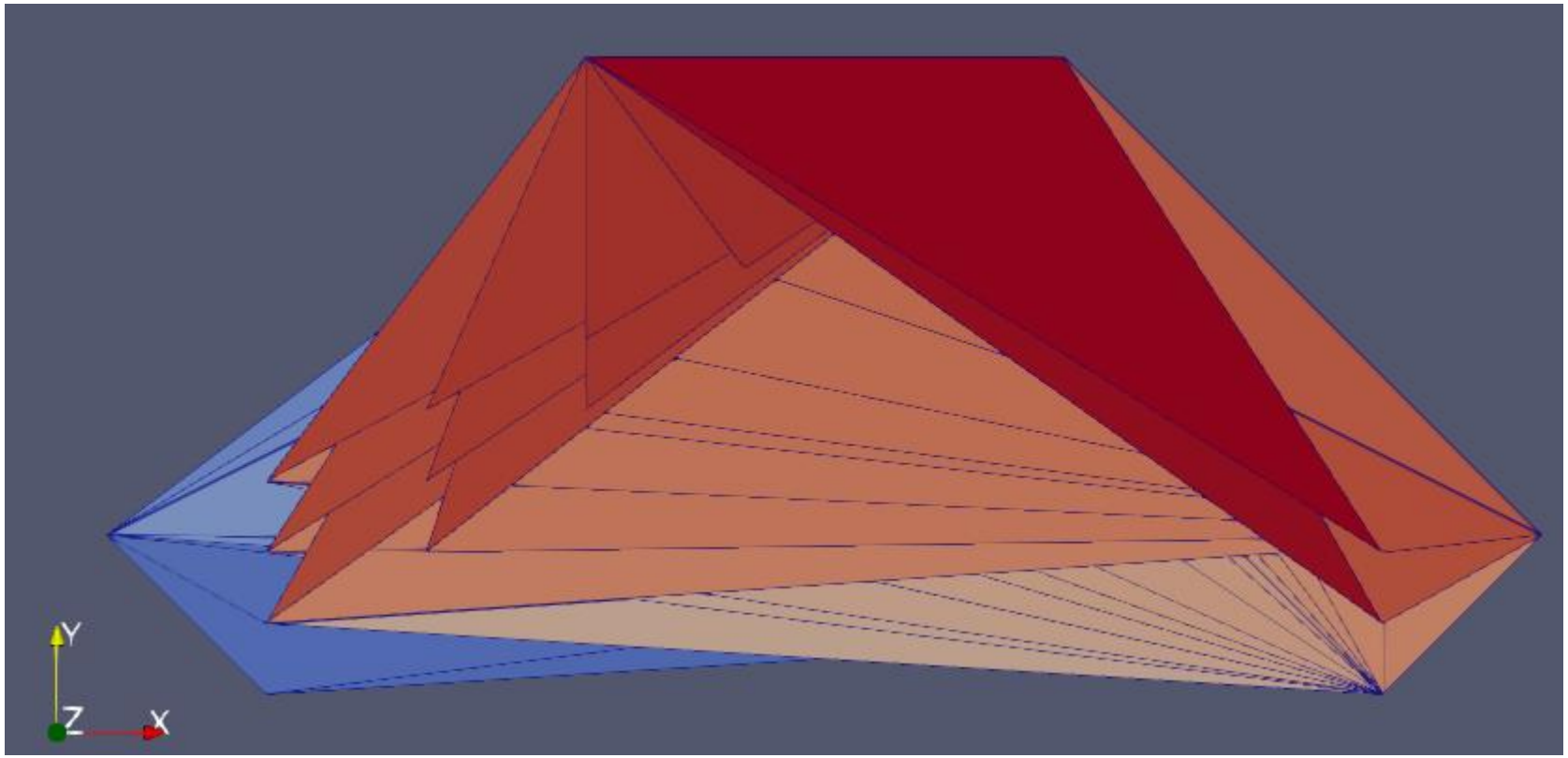


Iter	Compliance	Duration	Plain Volume	Plain Greyness	Physical Greyness	Physical Volume	Sub-Prb Iter	Neg Asym Min	Neg Asym Max	Pos Asym Min	Pos Asym Max	Lambda Plain Volume	Problem	β (Beta)
50.0	705.659	13.007	0.699825	0.269119	0.0996435	0.972593	2.0	-278.315	0.998979	0.0170409	278.324	19.2581	3dhanger-beta_8.0	8.0

Problems faced:

DO NOT TRUST CHATGPT FOR MESHING!!!!!!

It might be pretty, but it definitely won´t simulate and if it does (IT DID) it won´t be correct. Beware and test it or be dumb like me and ask the professor why numbers don´t number!



DO TRUST CHATGPT FOR MESHING !!!

I was able to mesh my hanger shape using ChatGPT (obviously) as it was a simple shape which has just 2 slopes and flat surface on top, on the other hand the mesh that failed had many edges and vertices (3 of my weekends too), some of which were improperly created, resulting in a corrupted mesh. This, in turn, led to terrible FEM results and made the mesh unsuitable for optimization

ANY QUESTIONS ?

THANK YOU