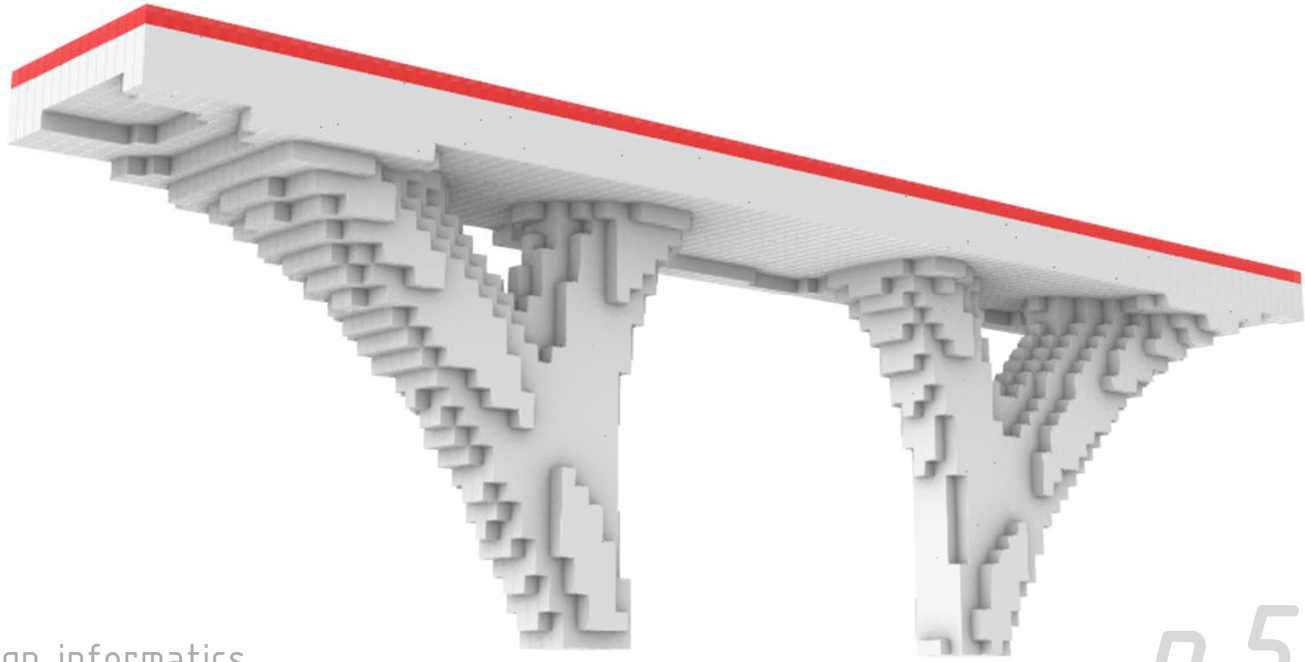
Topology optimization as structural form finding

Rick van Dijk - 4373618



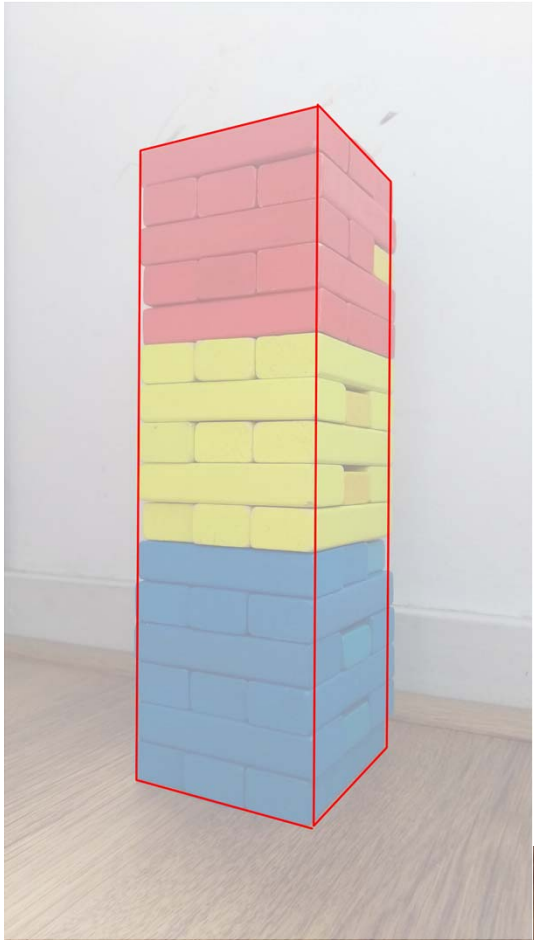
Committee:
Dr. ir. Pirouz Nourian - AE+T / Design informatics
Dr. ir. Matthijs Langelaar - 3ME / Structural optimization and mechanics

p 5

Index

01 Topology Optimization

01 Topology Optimization



Make a tower that:

- Does not fall
- Is 16 blocks high
- Has a rectangular shape

01 Topology Optimization

Make a tower that:

- Does not fall
- Is 16 blocks high
- Has a rectangular shape

48 blocks!

01 Topology Optimization

Make a tower that:

- Does not fall
- Is 16 blocks high
- Has a rectangular shape

32 blocks!

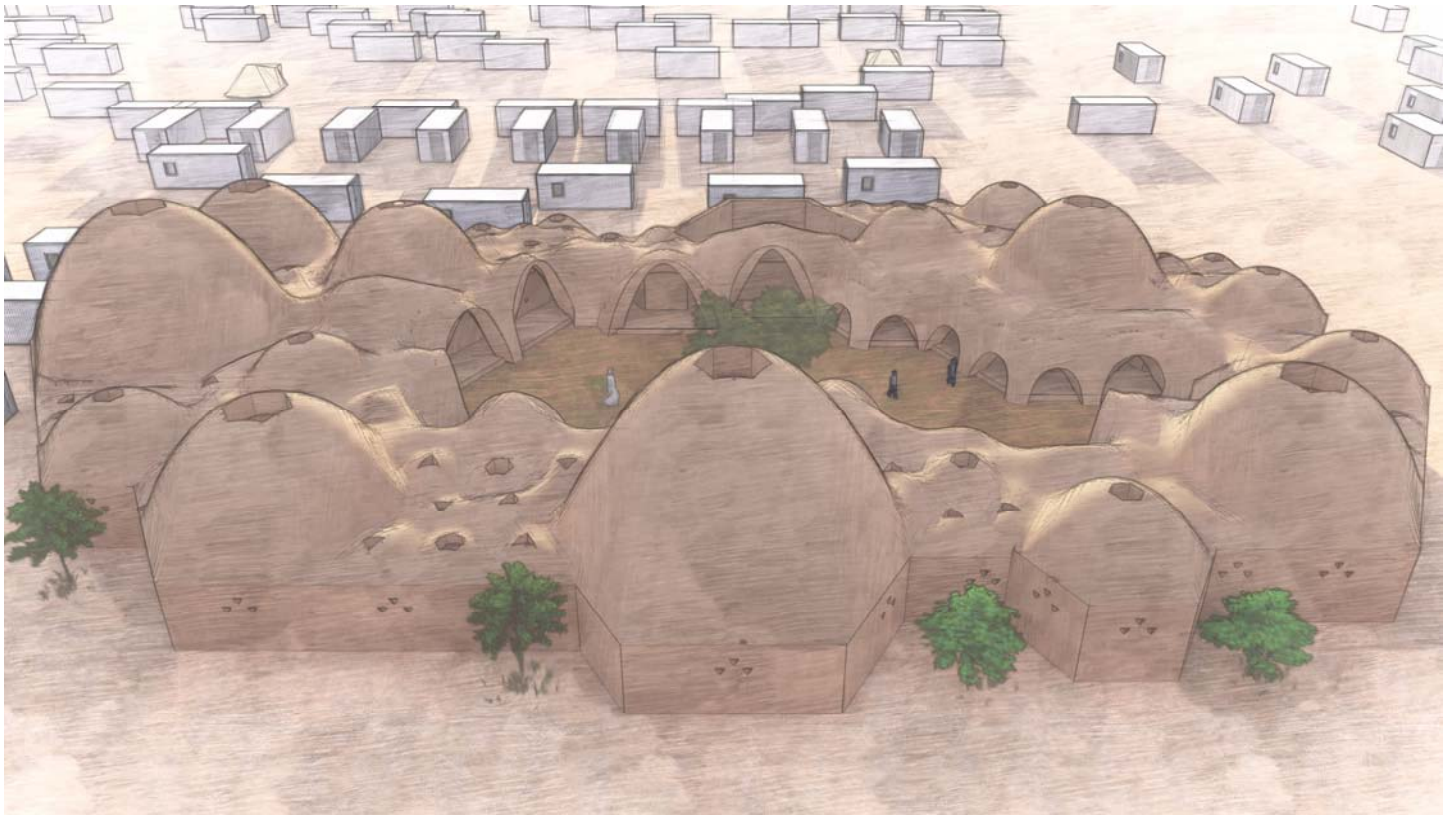
01 Topology Optimization

Make a tower that:

- Does not fall
- Is 16 blocks high
- Has a rectangular shape

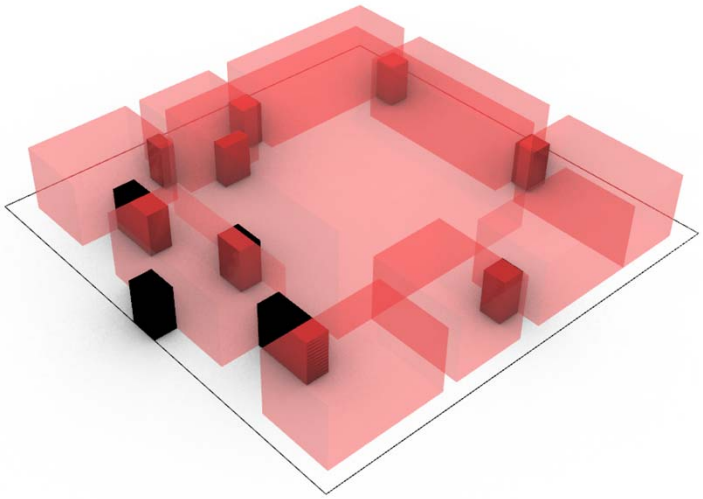
24 blocks!

01Background

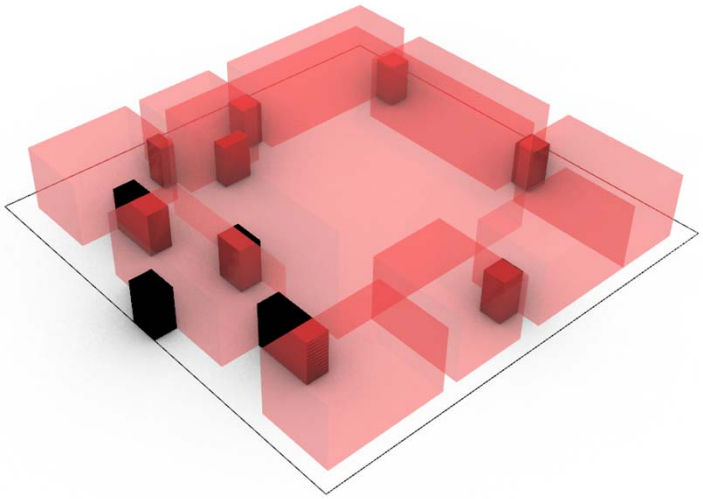


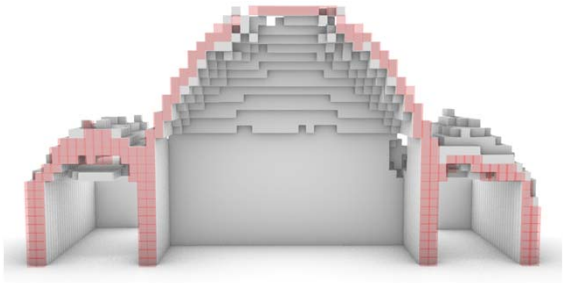
>Earthy result

01 Background

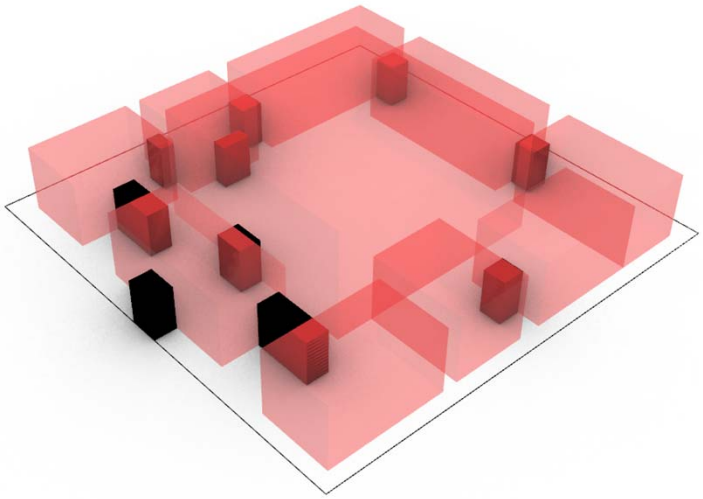


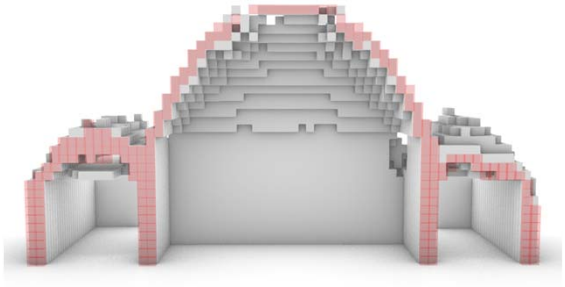
01 Background

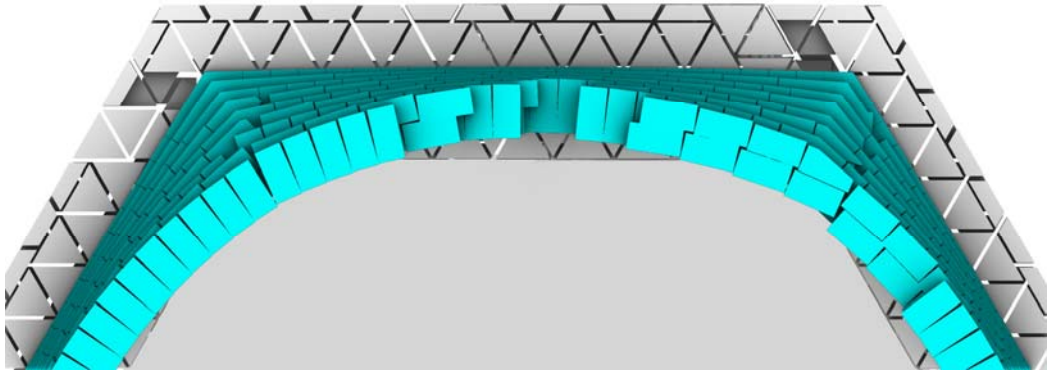




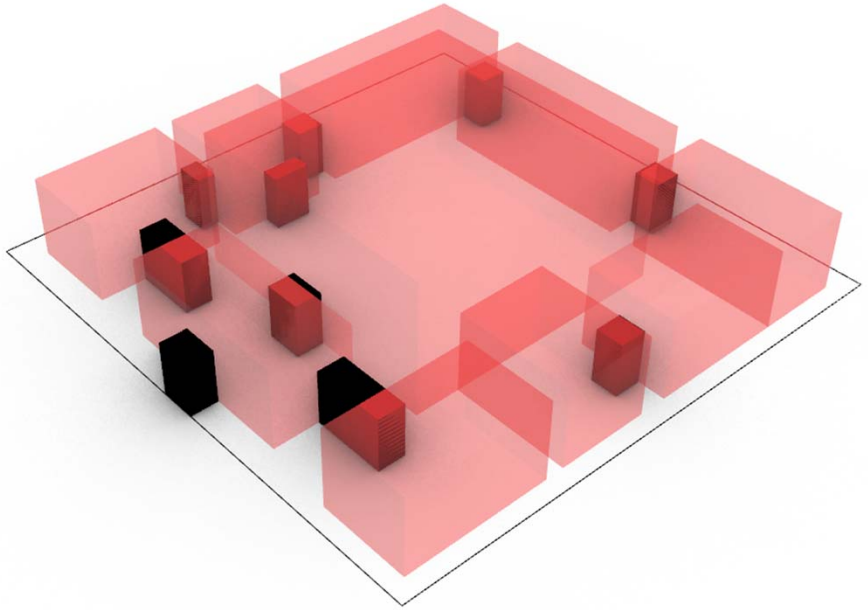
01 Background

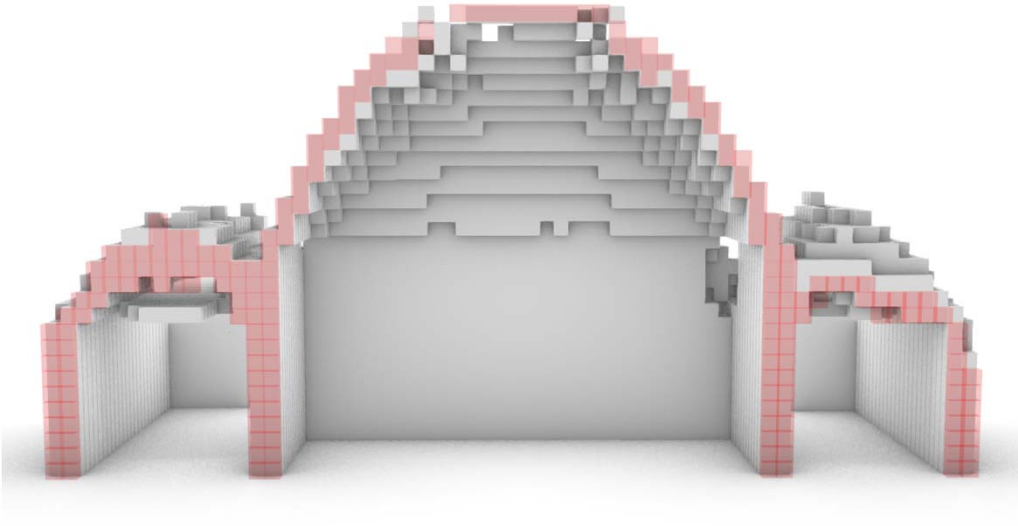






01 Background

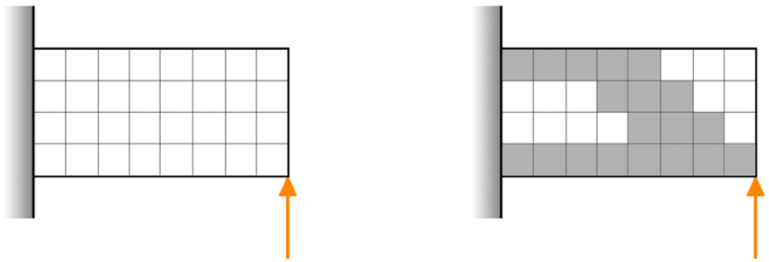


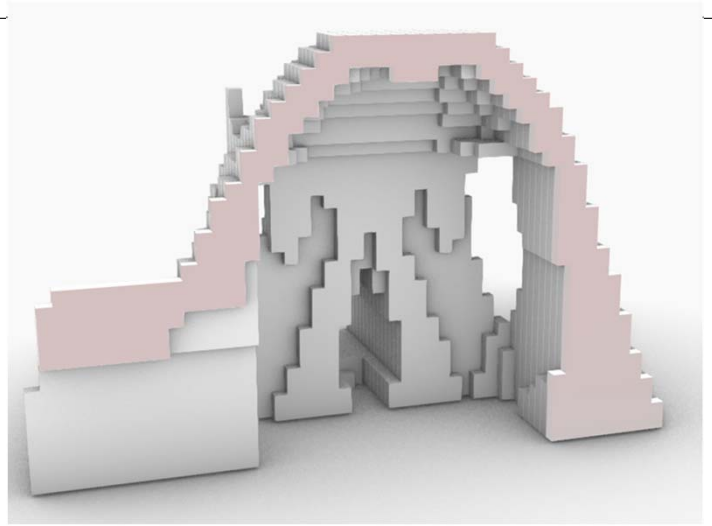


01 The main problem

"Topology optimization is an often-used method to generate complex shapes with a maximized stiffness and little volume. Implementations in (masonry) architecture look promising but requires the implementation of density dependent forces".

01 Objective





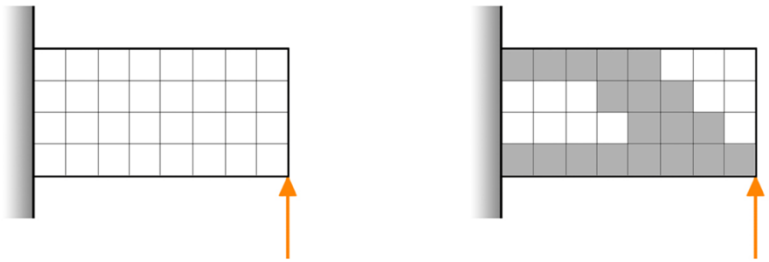
Inputs of topology optimization:

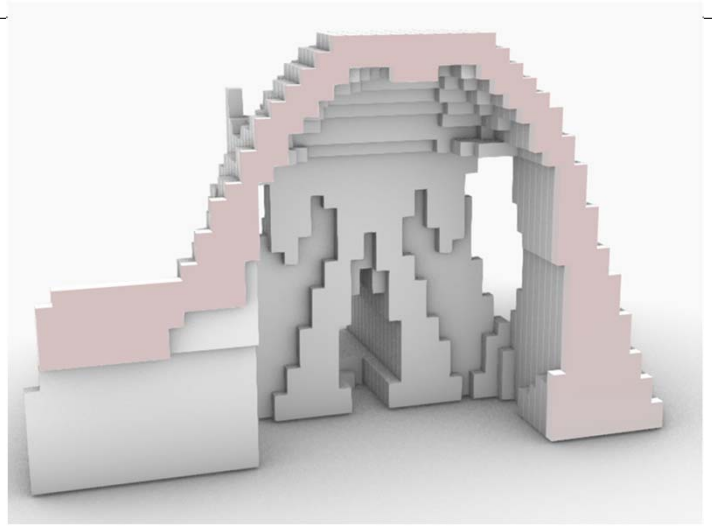
- *Structural*
- *Heat distribution*
- *Water flow*

Topology optimization in buildings:

- *Structural*
- *Preset forces / Area loads*
- *Self weight*
- *Snow loads*
- *Wind loads*

01 Objective





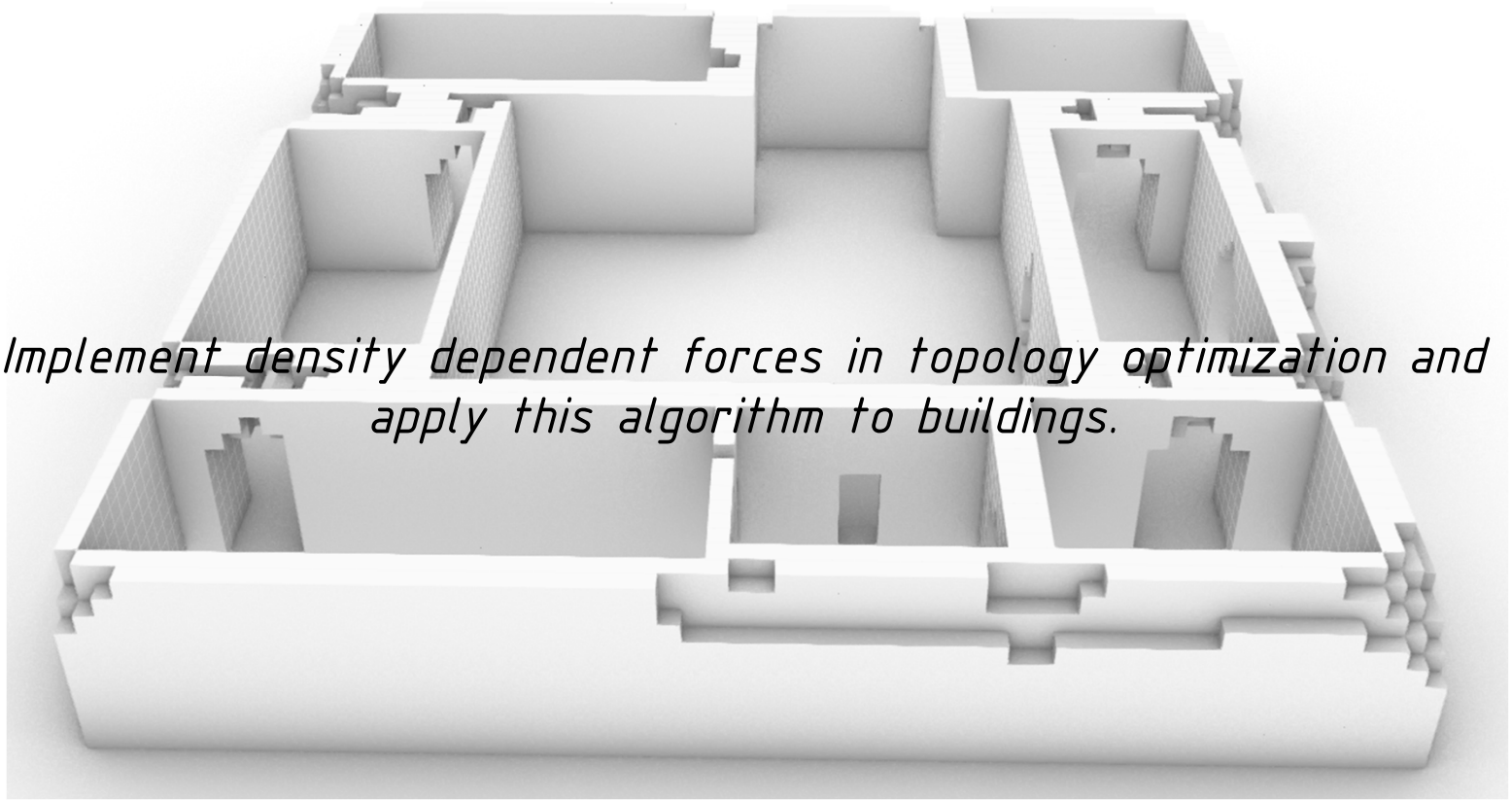
Inputs of topology optimization:

- *Structural*
- *Heat distribution*
- *Water flow*

Topology optimization in buildings:

- *Structural*
- *Preset forces / Area loads*
- *Self weight*
- *Snow loads*
- *Wind loads*

01 Objective



Implement density dependent forces in topology optimization and apply this algorithm to buildings.

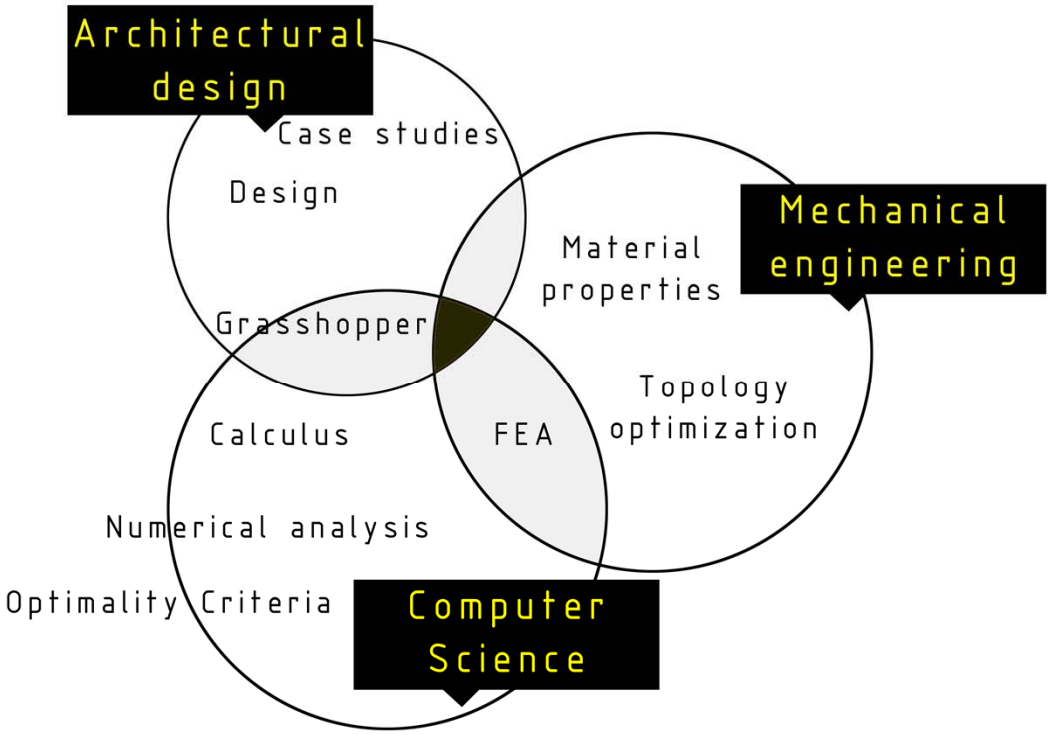
01 Sub objectives

create a working topology optimization methodology and translate this in the form of an algorithm.

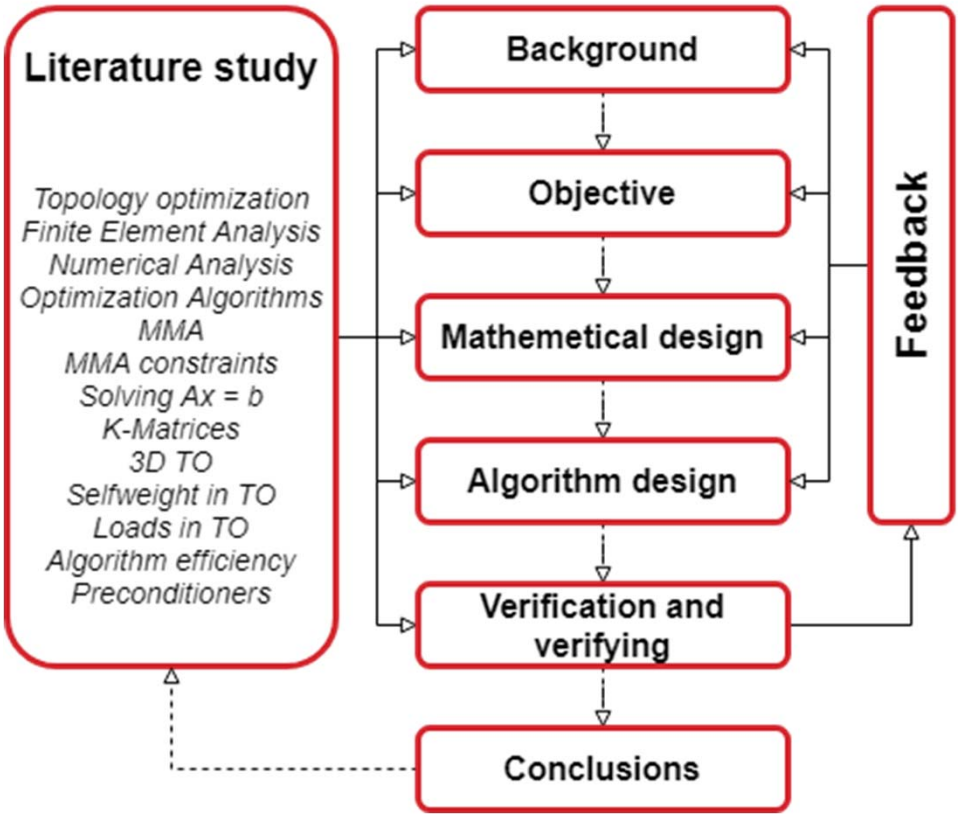
implement density dependent forces in the methodology and in the algorithm.

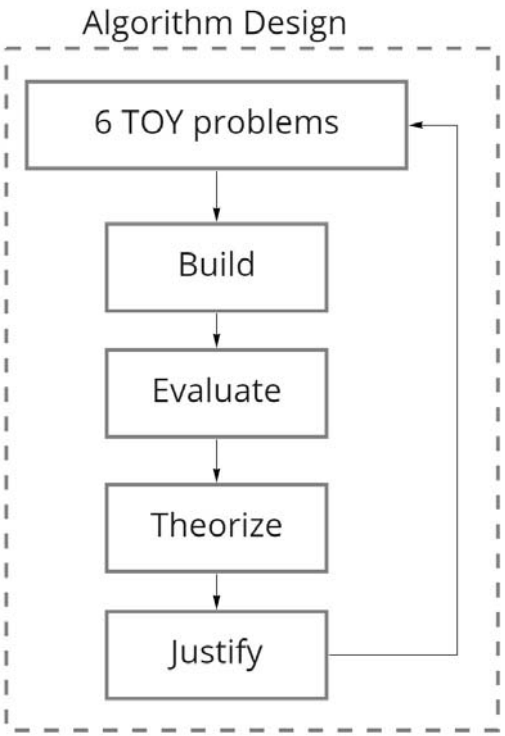
translate the methodology and the algorithm to 3D geometry.

01 Research framework

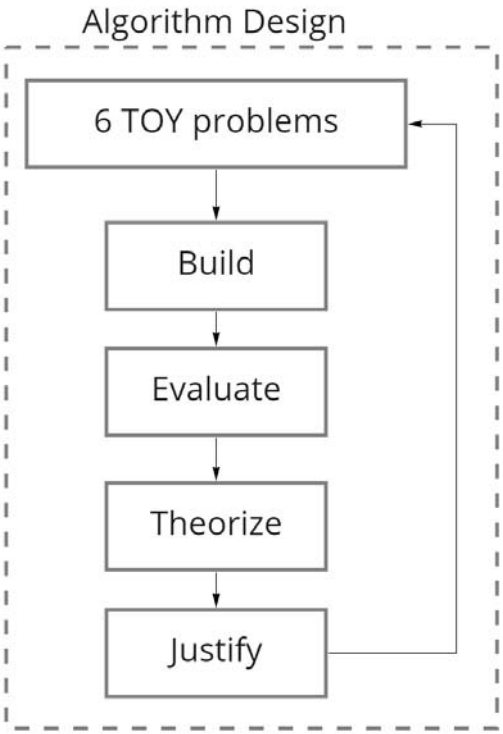


01 Methodology

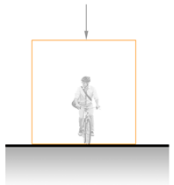




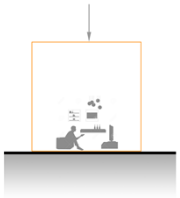
01Methodology

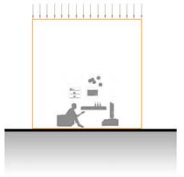


create a working topology optimization methodology and translate this in the form of an algorithm.

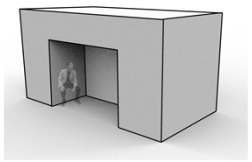


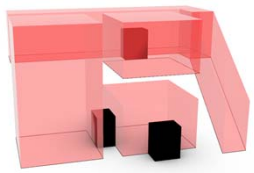
implement density dependent forces in the methodology and in the algorithm



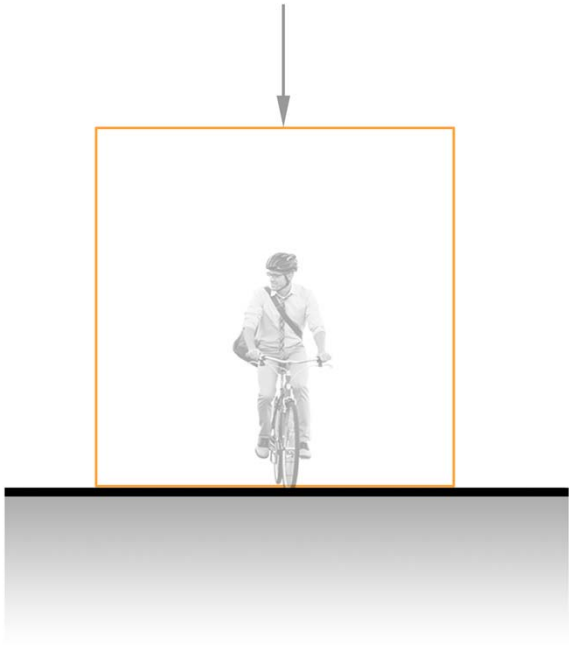


translate the methodology and the algorithm to 3D geometry.





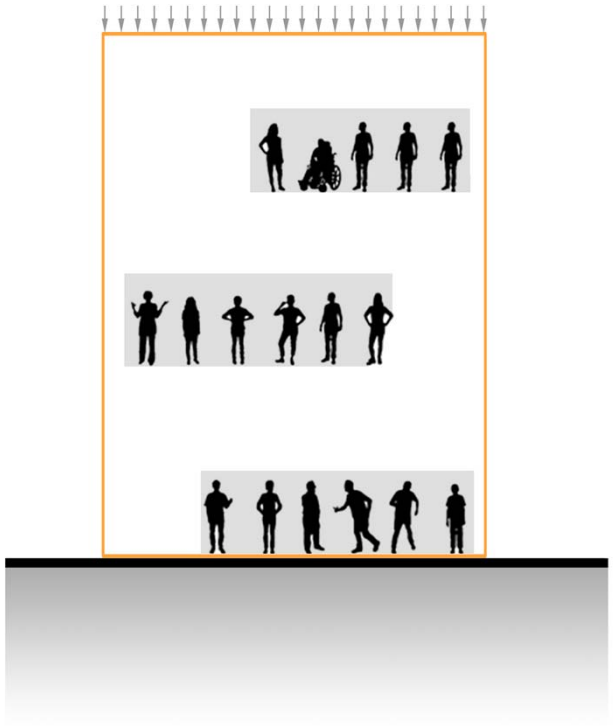
"create a working topology optimization methodology and translate this in the form of an algorithm."



TOY1

- Get TopOp to work in Rhino
- User-Inputs
  - Forces
  - Supports
  - Number of elements
- Implement the existence of voids
- User-based input of voids

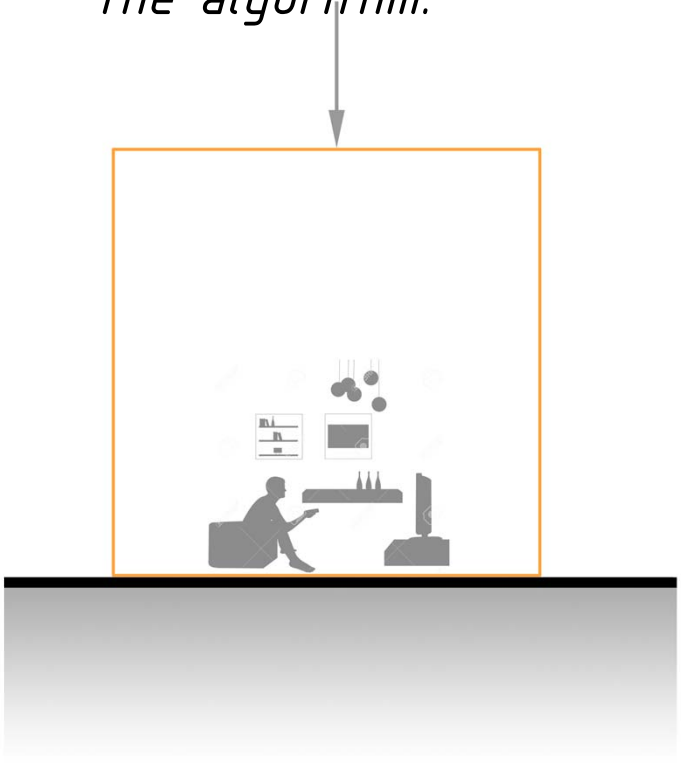
"create a working topology optimization methodology and translate this in the form of an algorithm."



TOY2

- Implement area loads
- A void indexing system
  - Multiple voids
  - Complex design spaces
- Forces inside the design space
- User-input for these "rooms"

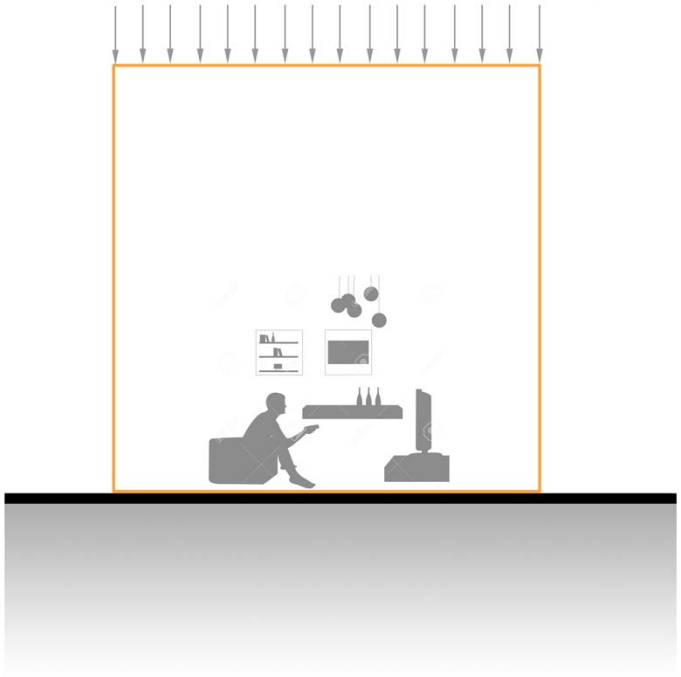
"implement density dependent forces in the methodology and in the algorithm."



TOY3

- Implement self weight in the algorithm
- Define sizes of self weight

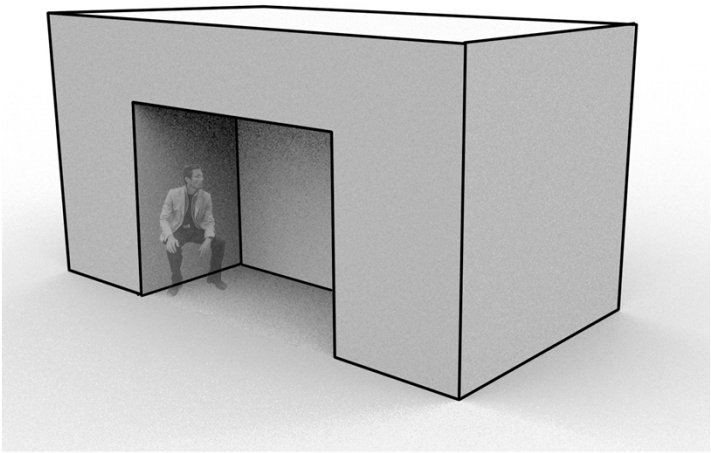
"implement density dependent forces in the methodology and in the algorithm."



TOY4

- Apply area loads dependent on the roof shape
- Implement a roofing constraint

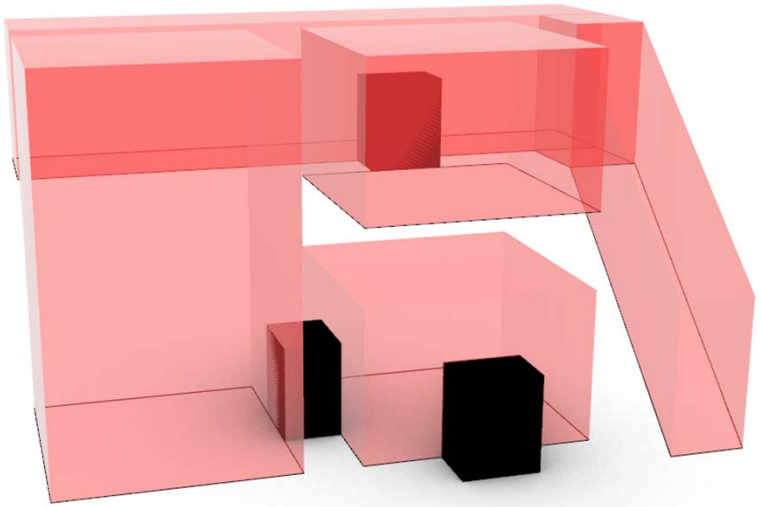
"translate the methodology and the algorithm to 3D geometry.



TOY5

- Implement an indexing system
- Handle inputs, including voids
- Algorithm optimization

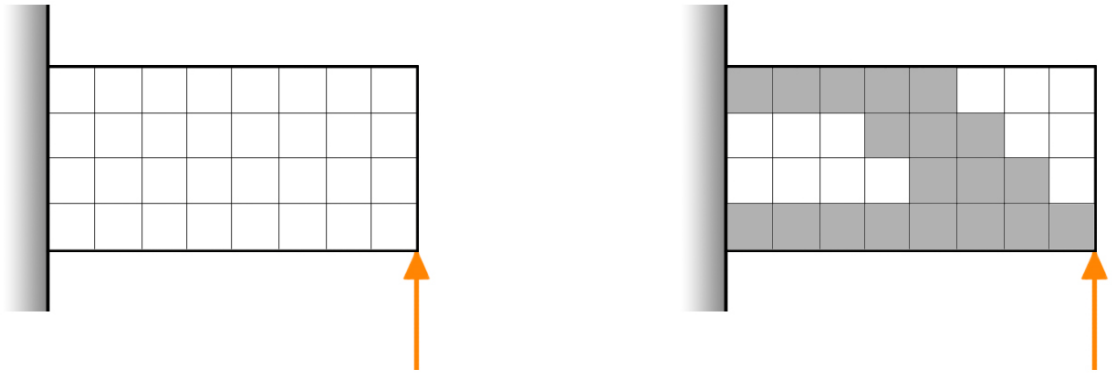
"translate the methodology and the algorithm to 3D geometry.



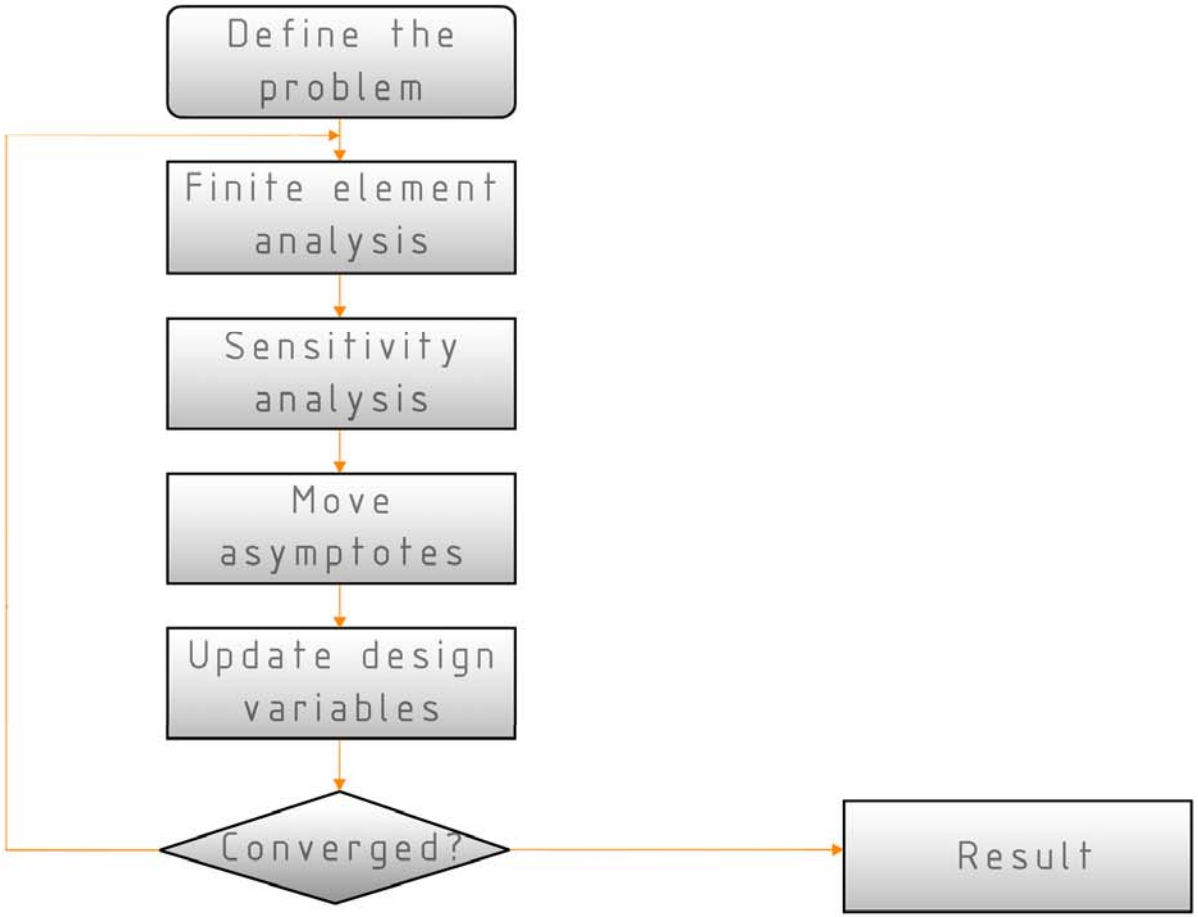
TOY6

- Implement density dependent forces
- Implement roof constraint
- Explore possibilities in architecture

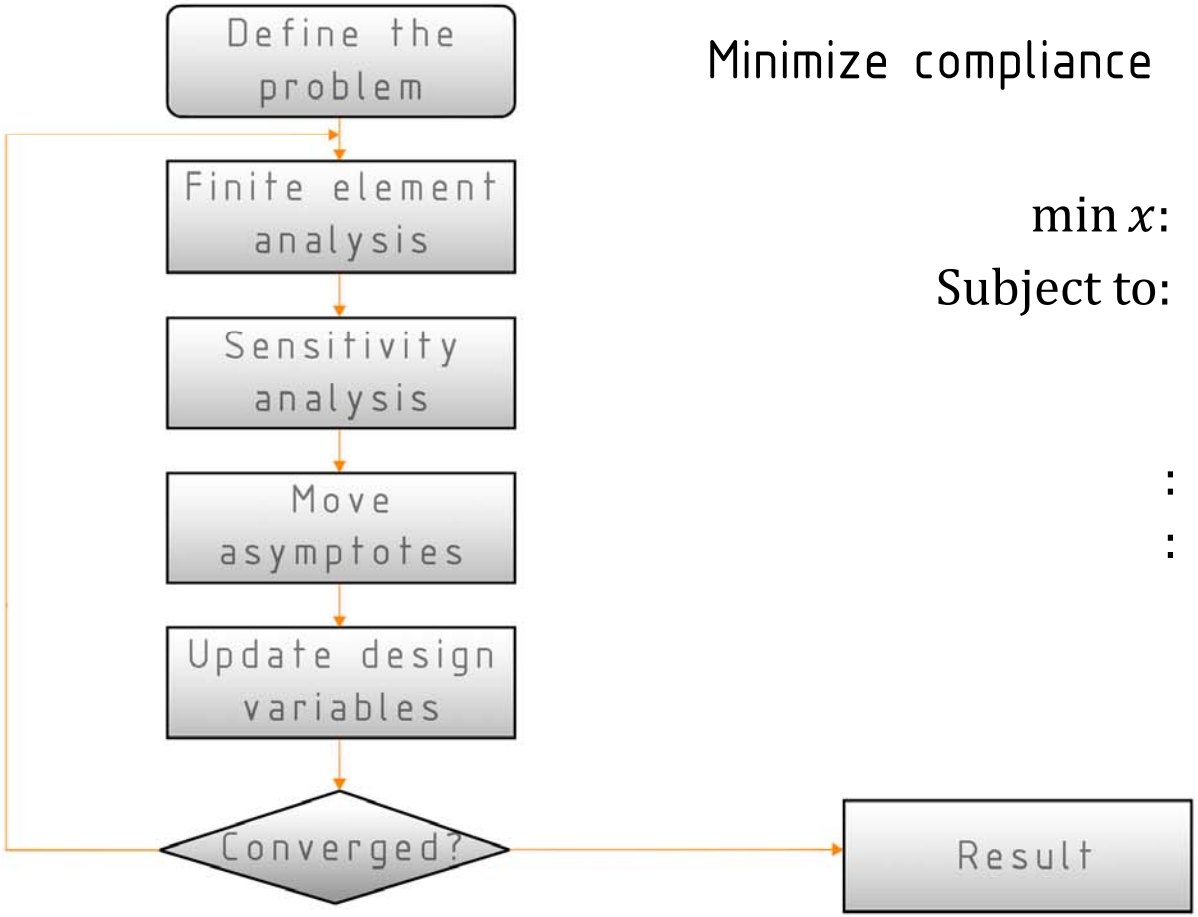
02 What is topology optimization?



(Bensoe & Sigmund, 2003)



(Sigmund, 1999)



Minimize compliance

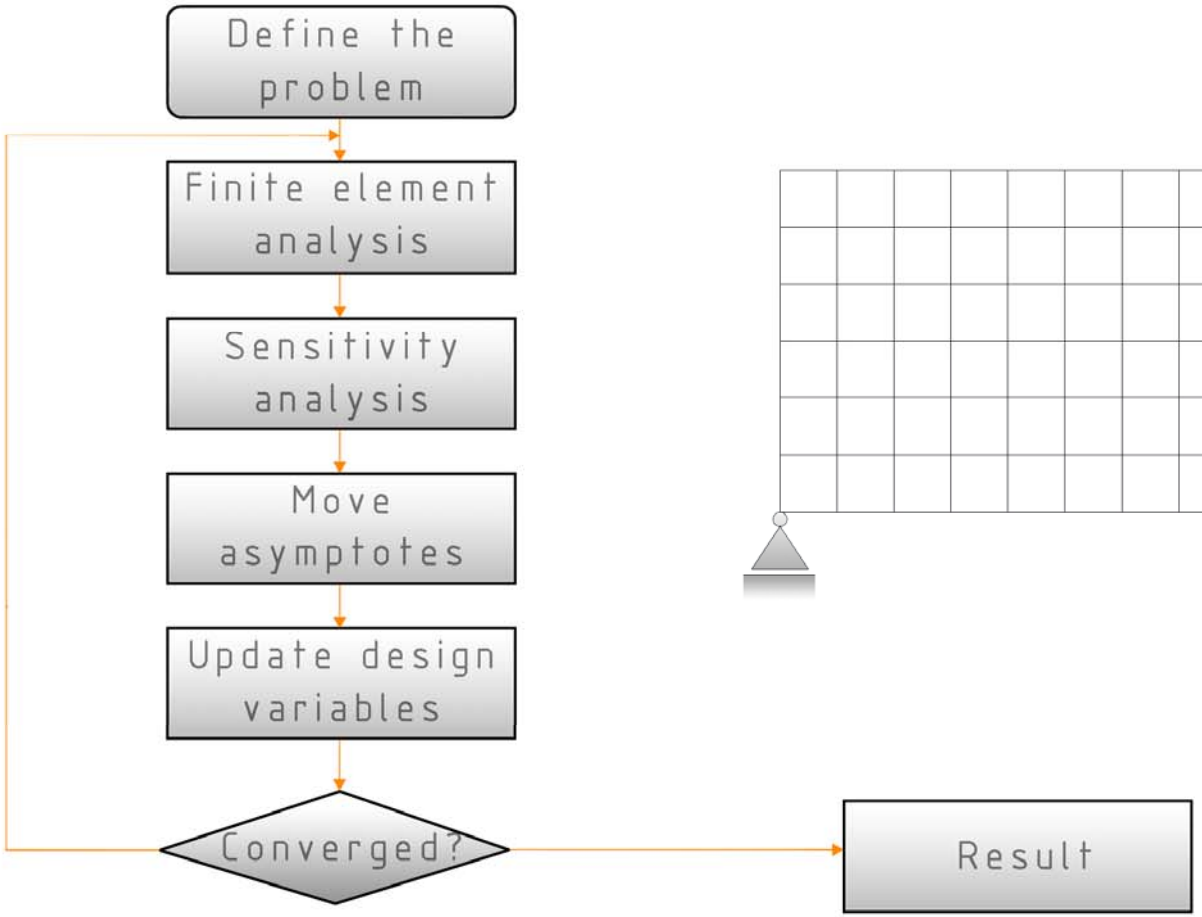
$$\min x: \quad c(x) = U^T K U$$

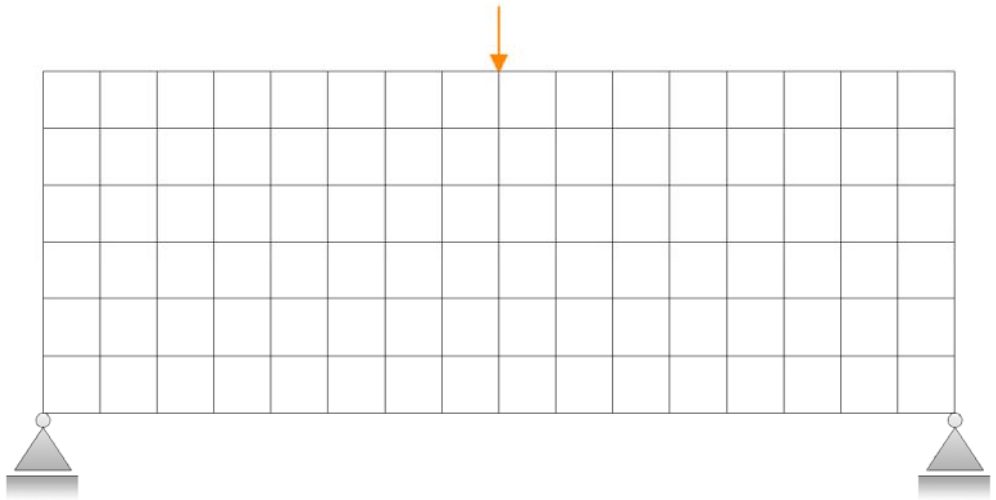
$$\text{Subject to:} \quad \frac{V(x)}{V_0} = volfrac$$

$$: \quad KU = F$$

$$: \quad 0 < x_{min} \leq x \leq 1$$

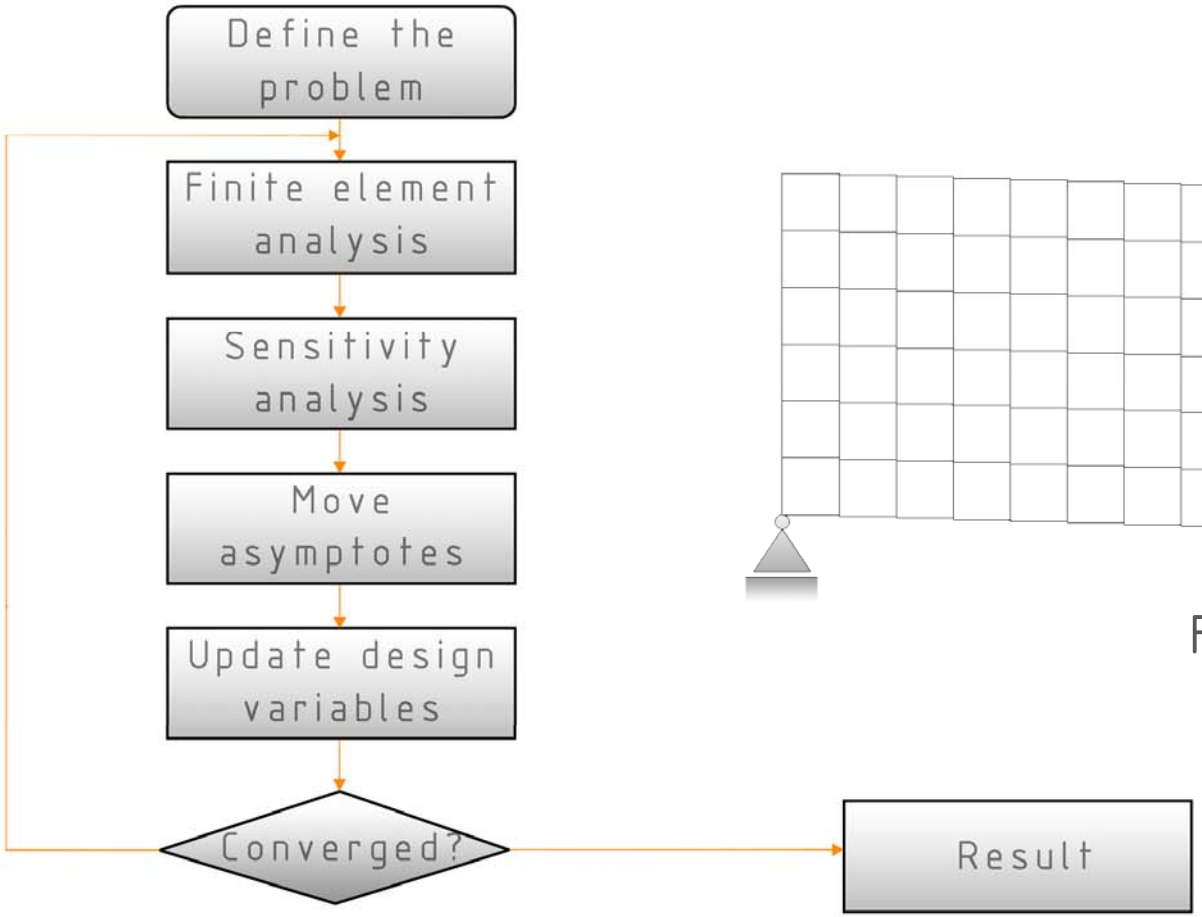
(Bensoe & Sigmund, 2003)

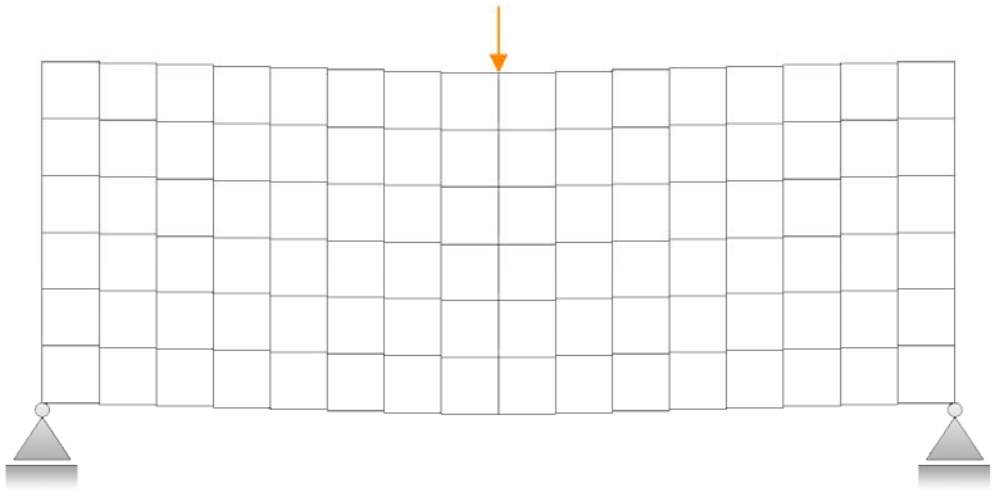




Define the problem

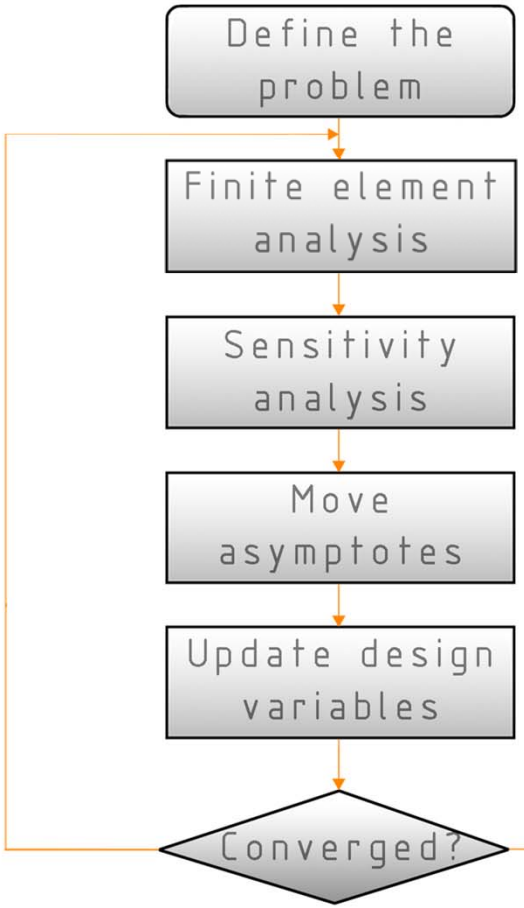
(Bensoe & Sigmund, 2003)

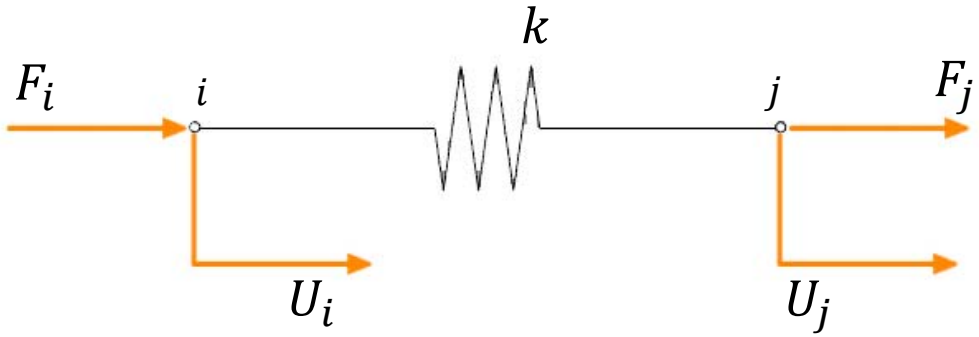




Finite element analysis

(Bathe, 2006)



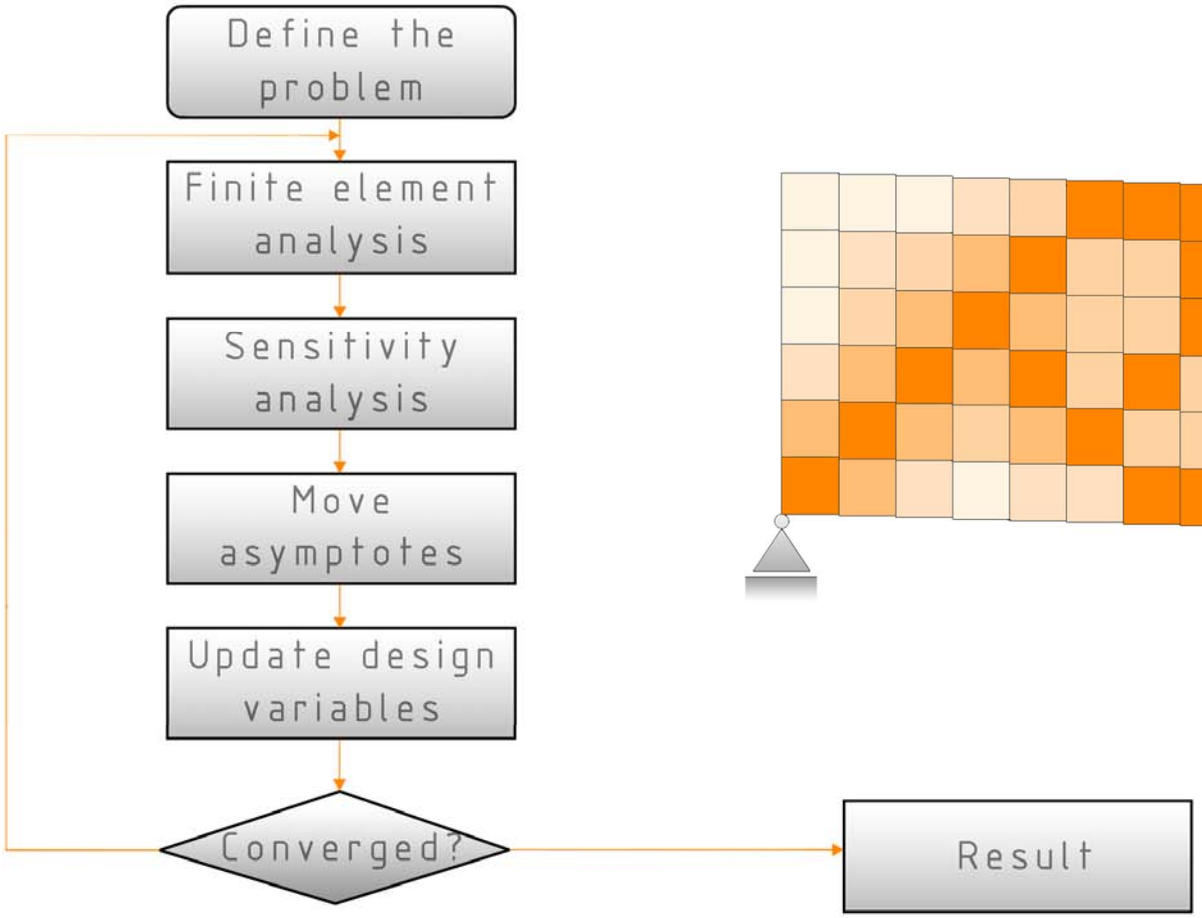


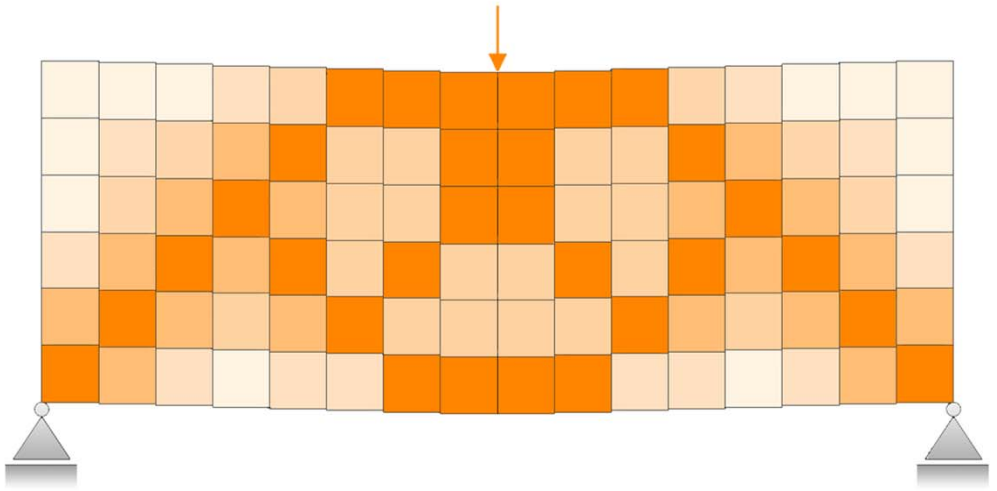
$$KU = F$$

$$K = \begin{bmatrix} k & -k \\ -k & k \end{bmatrix}$$

$$\begin{bmatrix} k & -k \\ -k & k \end{bmatrix} \begin{bmatrix} U_i \\ U_j \end{bmatrix} = \begin{bmatrix} F_i \\ F_j \end{bmatrix}$$

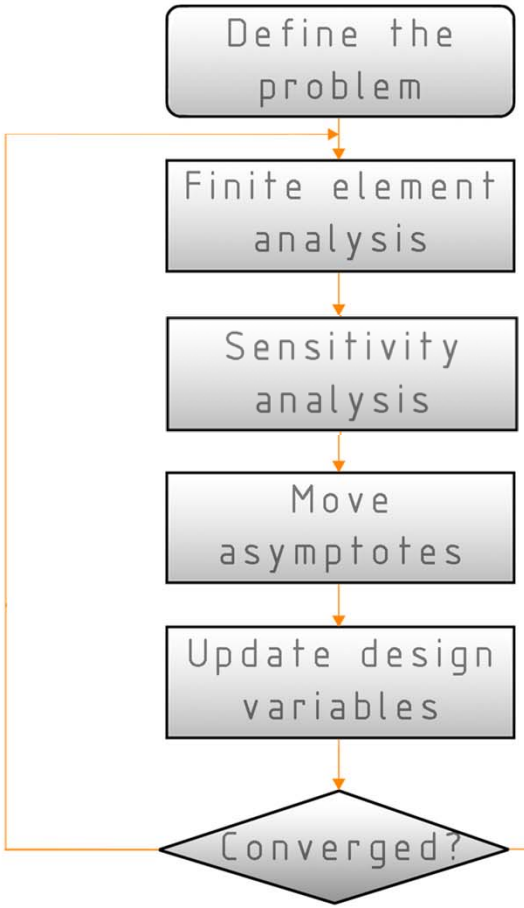
(Bensoe & Sigmund, 2003)



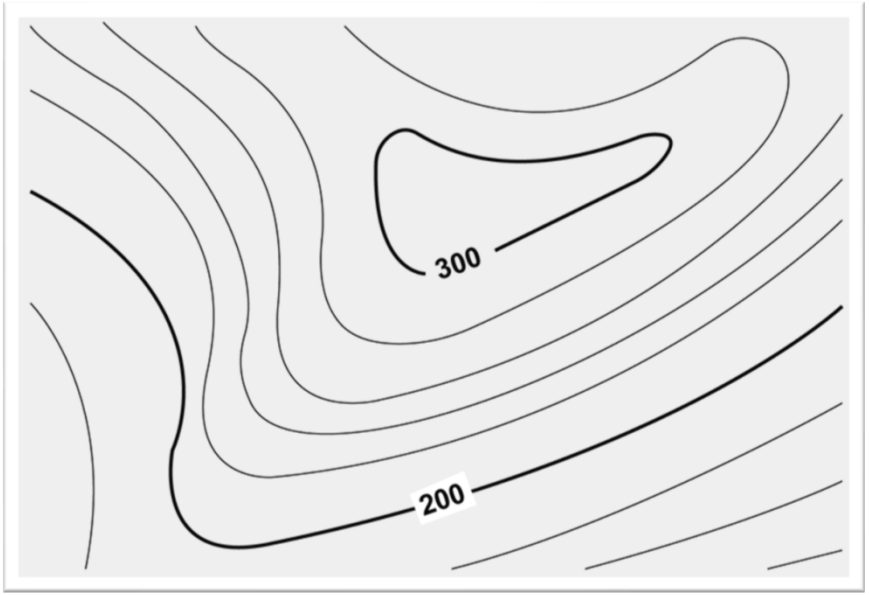


Sensitivity analysis

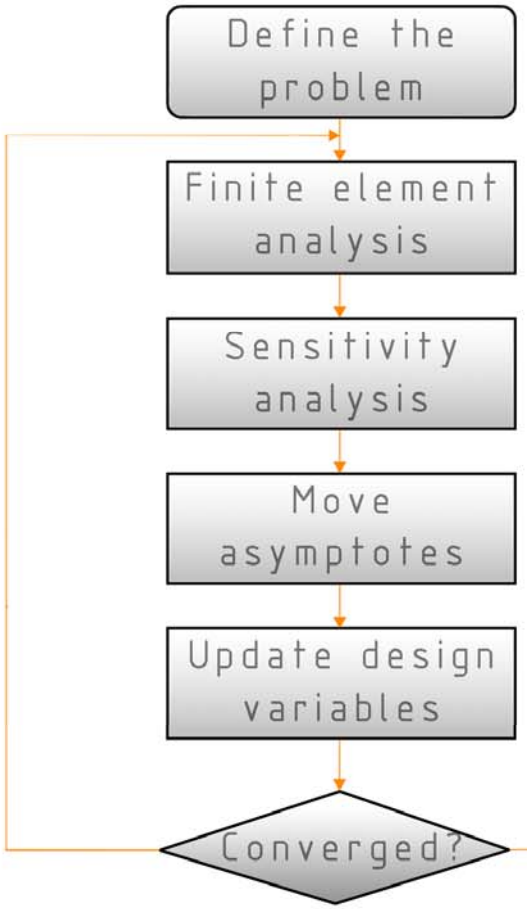
(Sigmund, 1999)



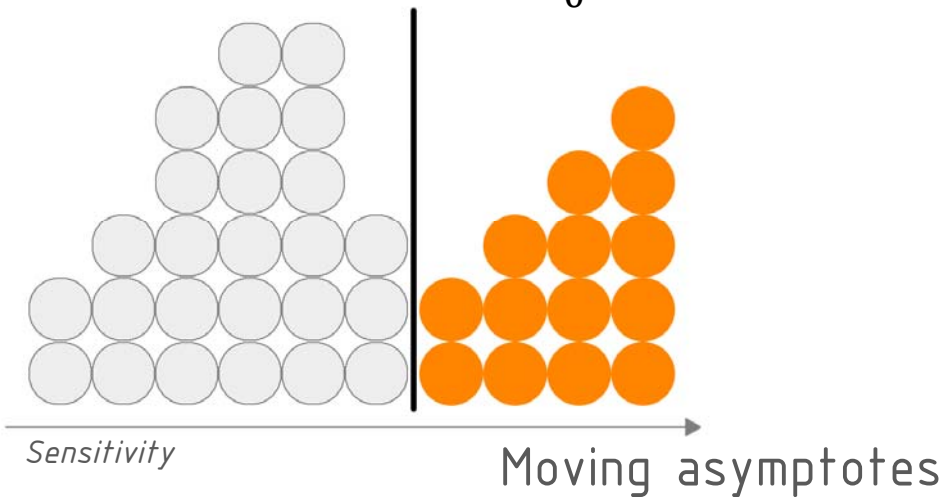
$$\frac{\partial C}{\partial x_e} = -p\,(x_e)^{p-1} U_e^T k_0 U_e$$



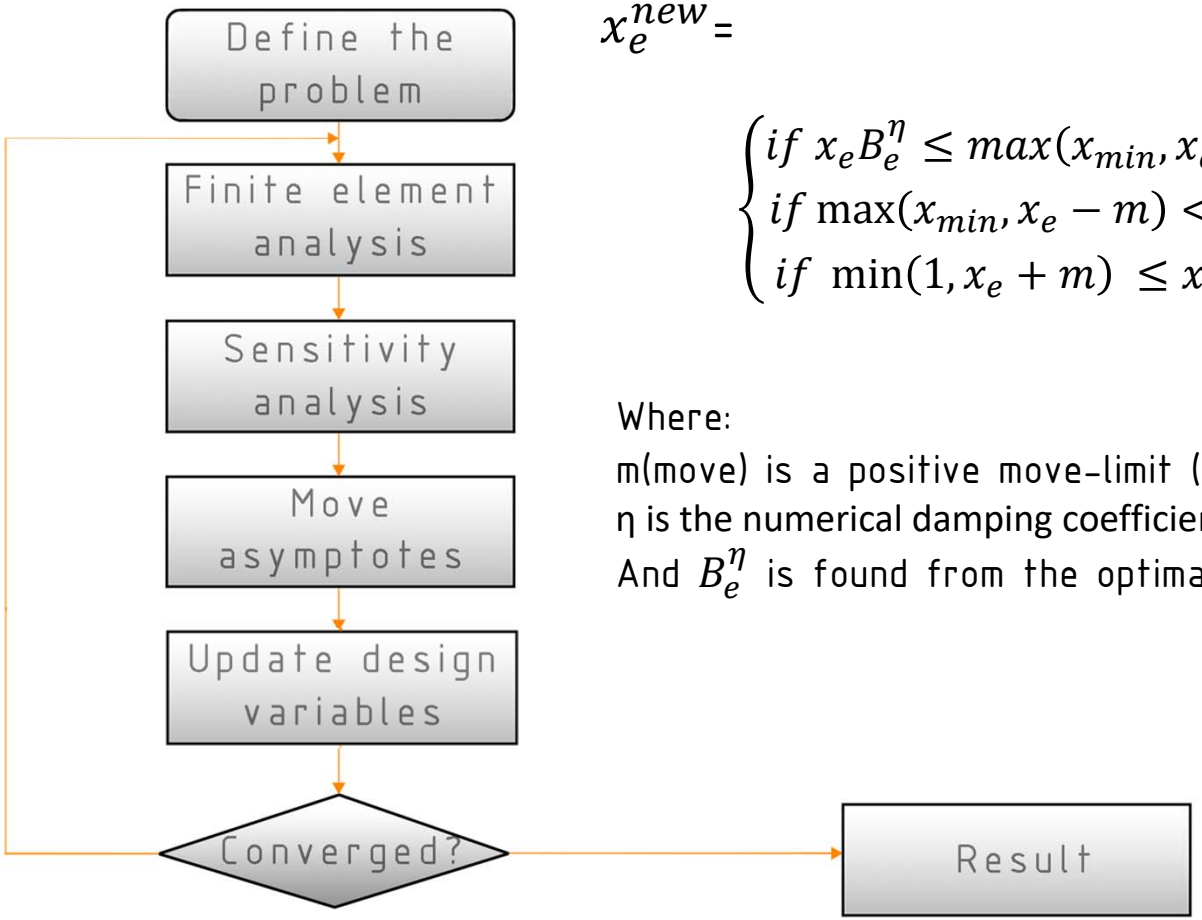
(Bensoe & Sigmund, 2003)



$$\frac{V(x)}{V_0} = volfrac$$



(Bendsoe, 1995)



$x_e^{new} =$

$$0 < x_{min} \leq x \leq 1$$

$$\begin{cases} if\ x_e B_e^{\eta} \leq max(x_{min}, x_e - m): & \mathbf{max(x_{min}, x_e - m)} \\ if\ \max(x_{min}, x_e - m) < x_e B_e^{\eta} < \min(1, x_e + m): & \boldsymbol{x_e B_e^{\eta}} \\ if\ \min(1, x_e + m) \leq x_e B_e^{\eta}: & \boldsymbol{min(1, x_e + m)} \end{cases}$$

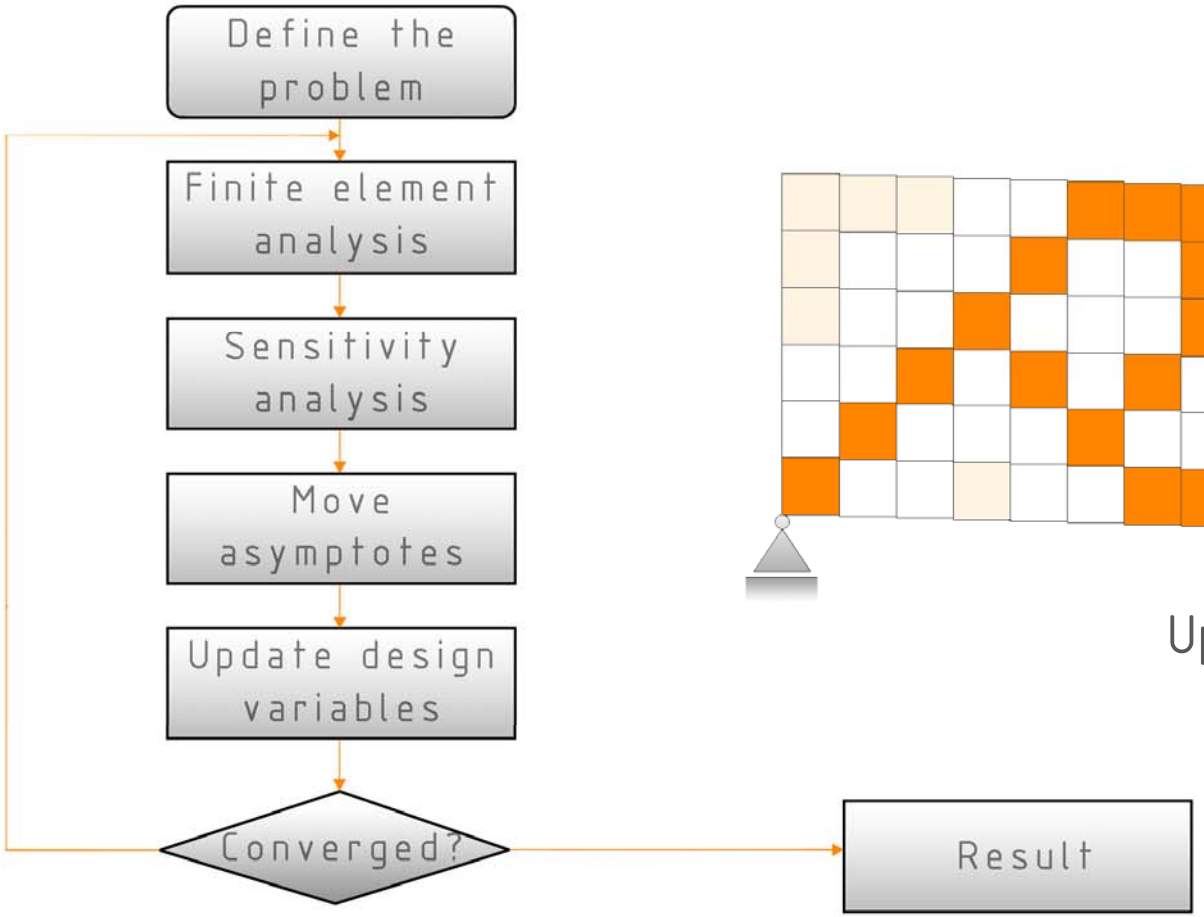
Where:

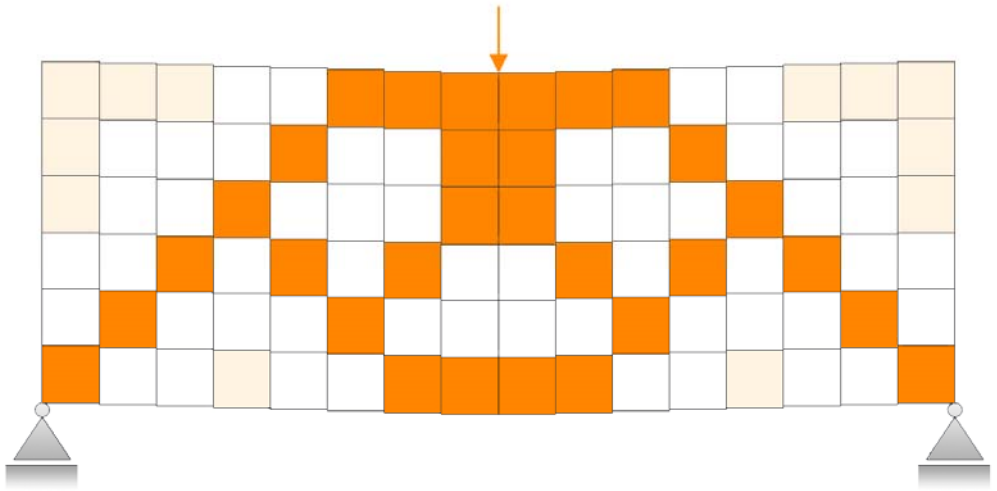
m(move) is a positive move-limit (= 0.2)

η is the numerical damping coefficient (= 1/2)

And B_e^{η} is found from the optimality condition

(Bensoe & Sigmund, 2003)



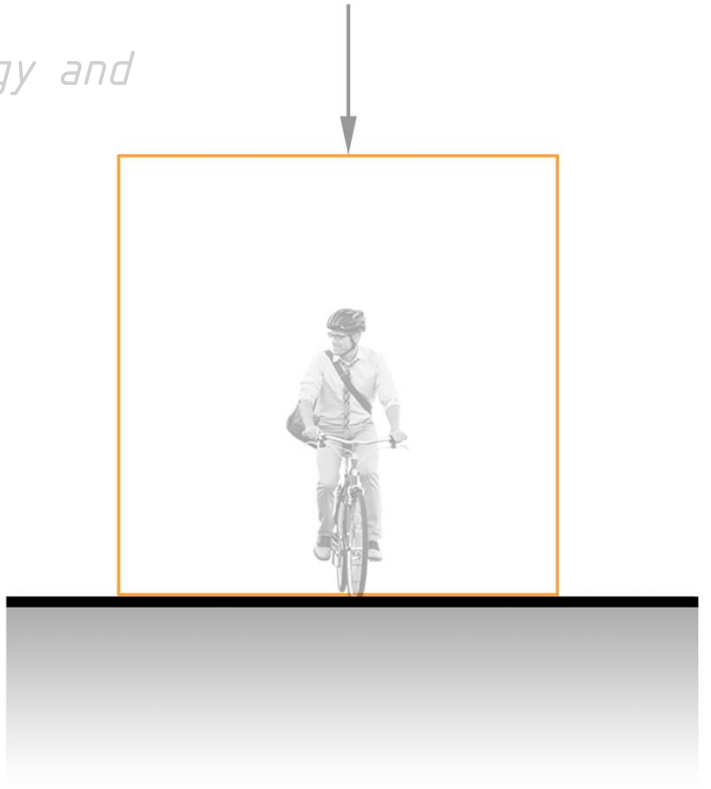


Update design variables

03 Toy problems

TOY1

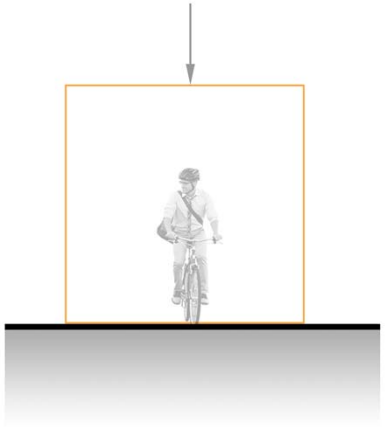
"create a working topology optimization methodology and translate this in the form of an algorithm."

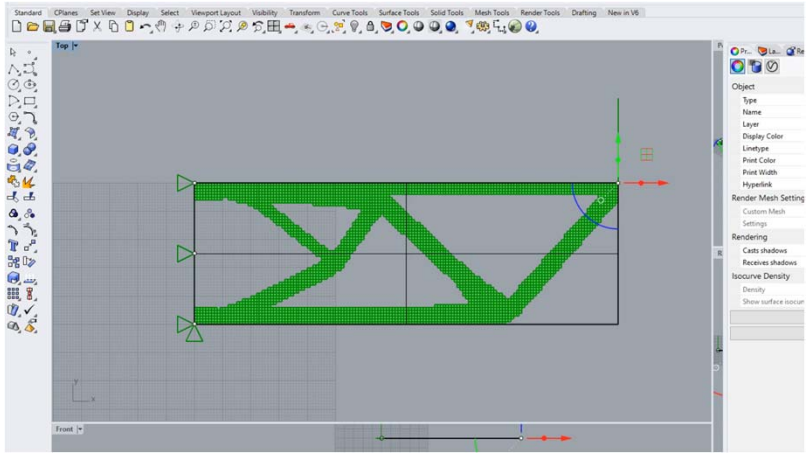


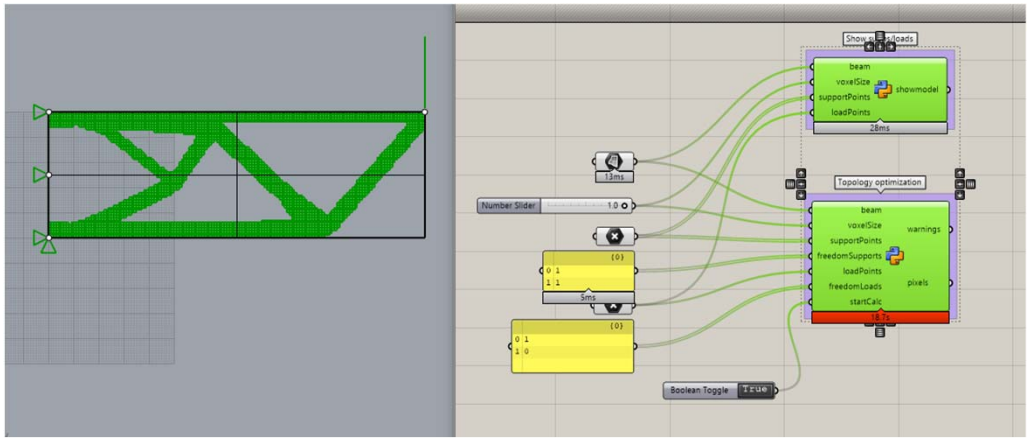
03 Toy problems

TOY1

- Get TopOp to work in Rhino
- User-Inputs
  - Forces
  - Supports
  - Number of elements
- Implement the existence of voids



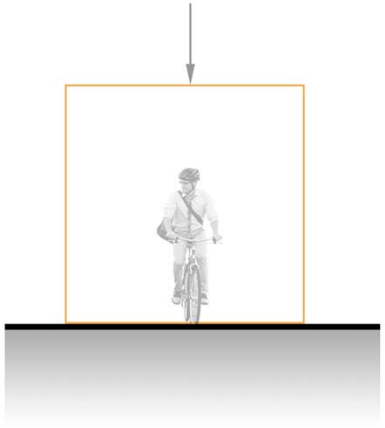


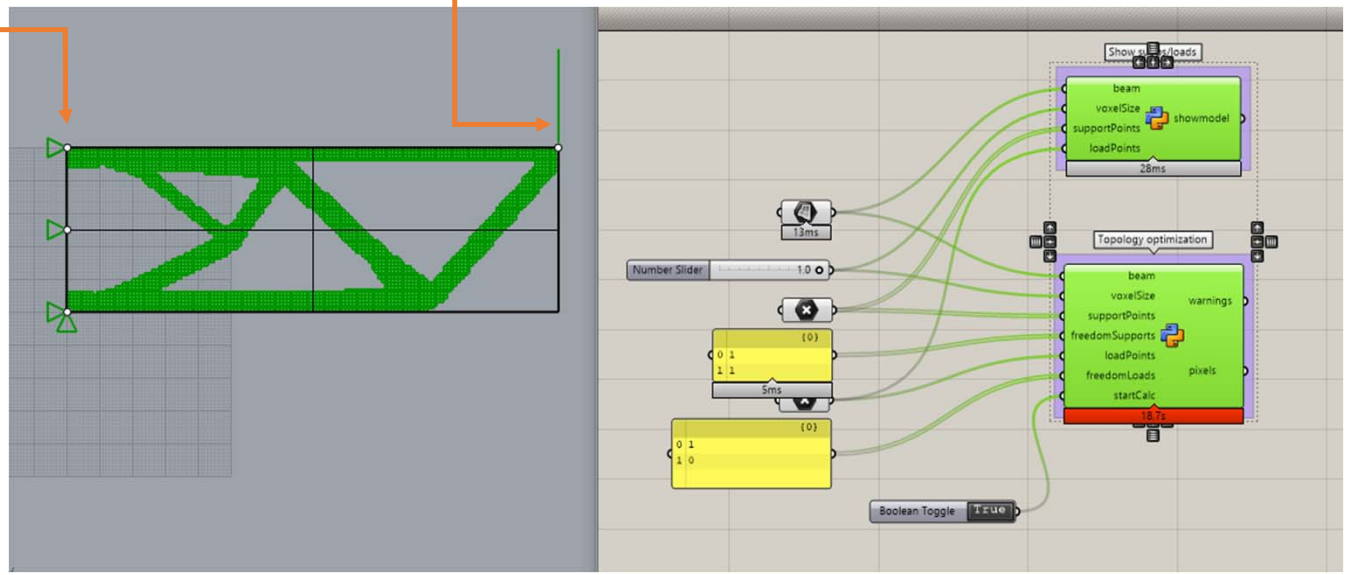


03 Toy problems

TOY1

- Get TopOp to work in Rhino
- User-Inputs
  - Forces
  - Supports
  - Number of elements
- Implement the existence of voids

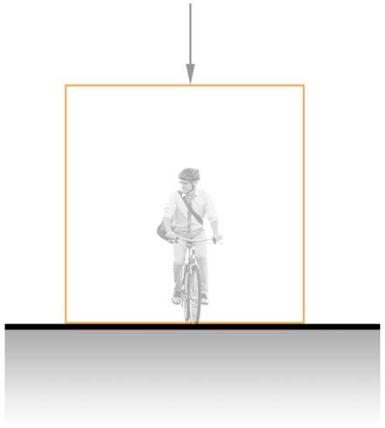




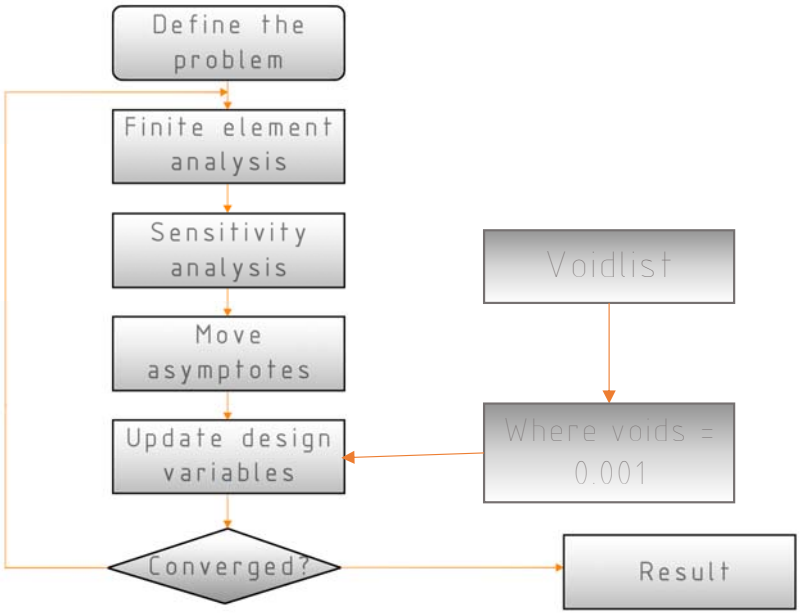
03 Toy problems

TOY1

- Get TopOp to work in Rhino
- User-Inputs
  - Forces
  - Supports
  - Number of elements
- Implement the existence of voids

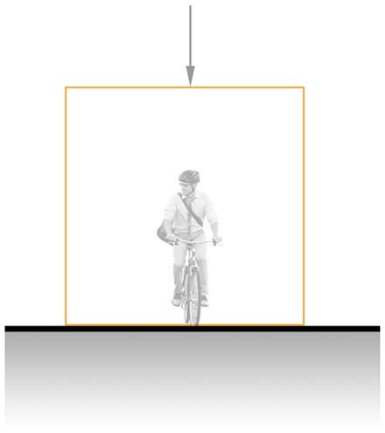


voids[*voidslist*]=1
where **voids** = 1, x = 0.001
else: x = x

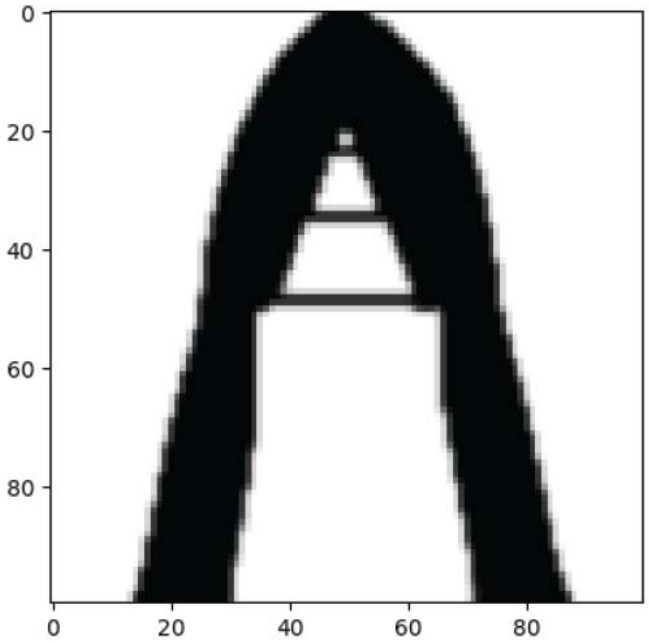


TOY1

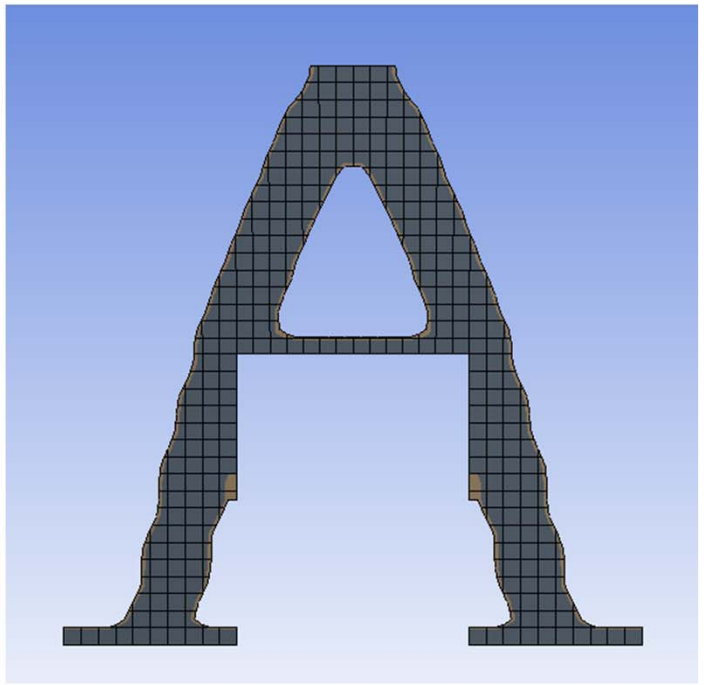
- Verification



Result



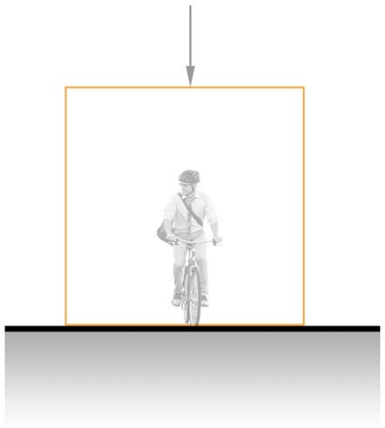
Ansys:

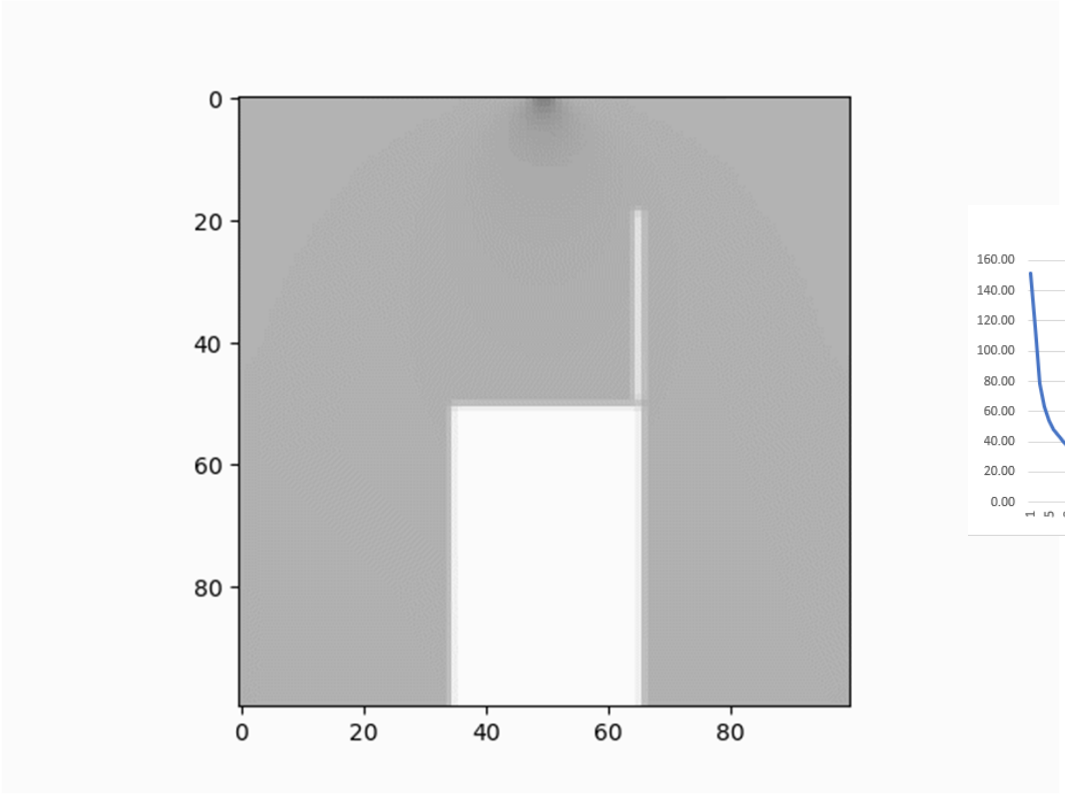


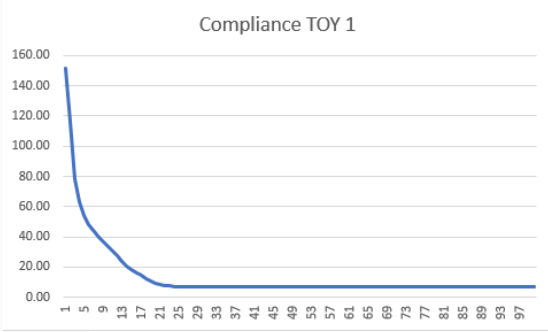
03 Toy problems

TOY1

- Verification

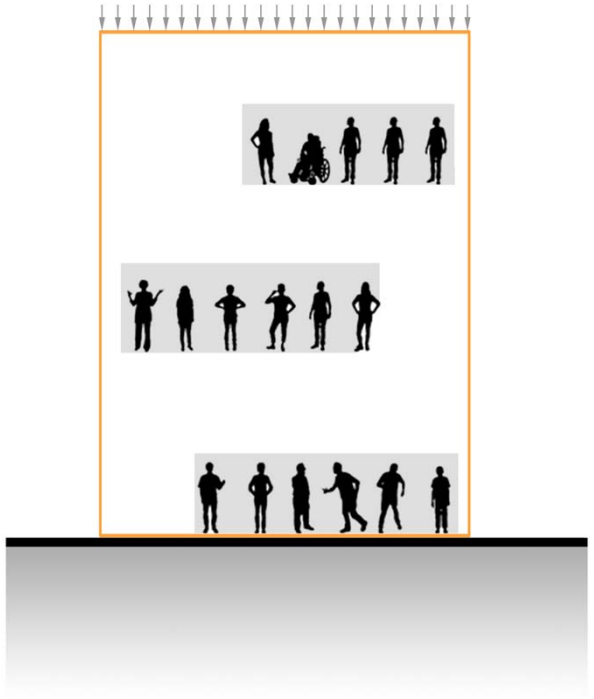






TOY2

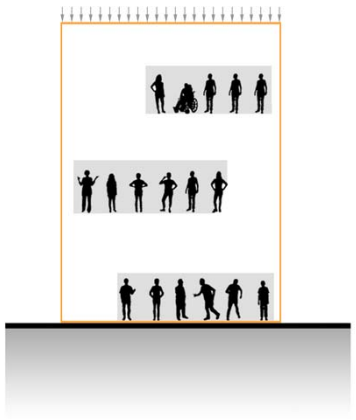
Configure more complex buildings with multiple voids and forces

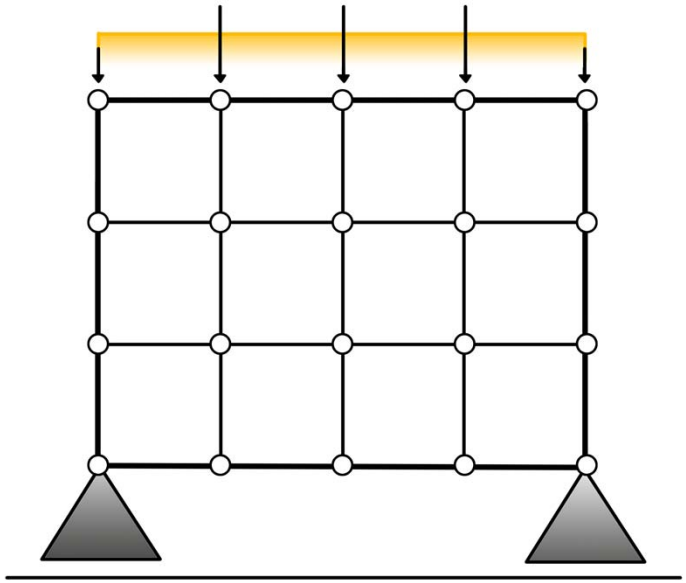


03 Toy problems

TOY2

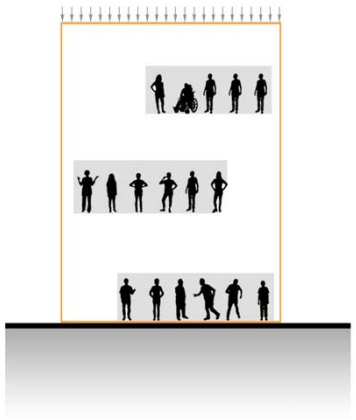
- Implement area loads
- A void indexing system
  - Multiple voids
  - Complex design spaces
- Forces inside the design space
- User-input for these "rooms"

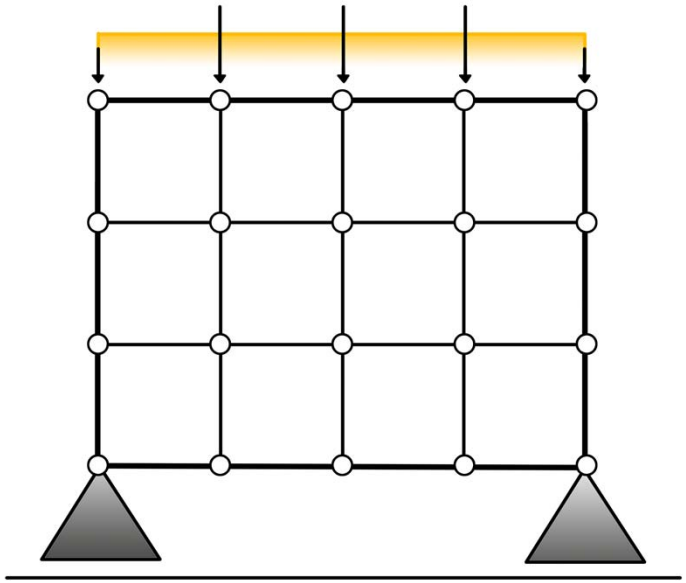




TOY2

- Implement area loads
- A void indexing system
  - Multiple voids
  - Complex design spaces
- Forces inside the design space
- User-input for these "rooms"



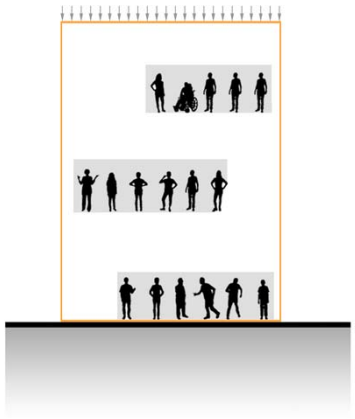


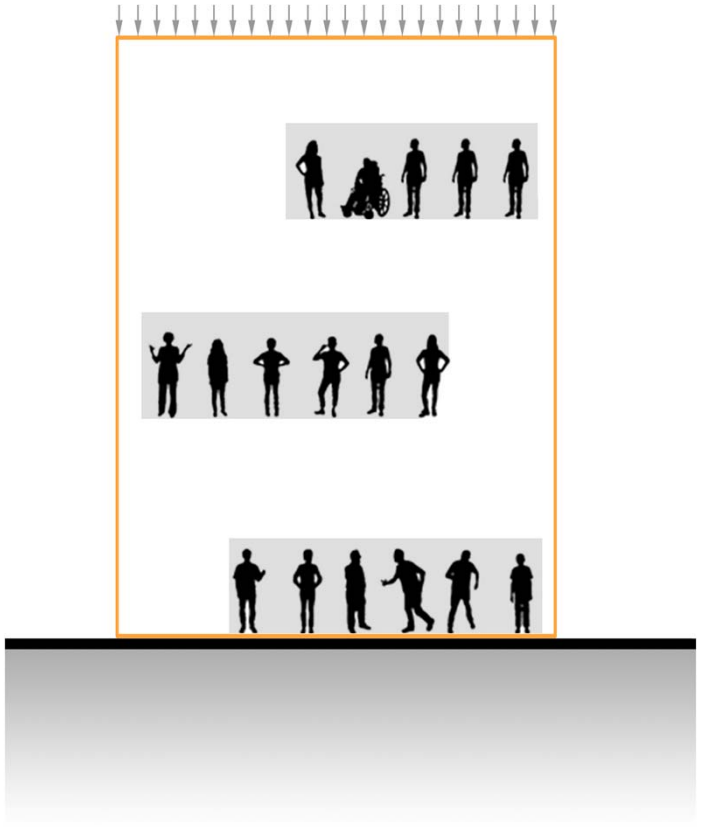
```
forceValue = 0.01
for roofelement in range(nelx + 1):
      nodeLeft = roofelement * (2* (nely+1)) + 1
      nodeRight = (roofelement+1) * (2* (nely+1))+1
      f[nodeLeft, nodeRight] = forceValue / 2
```

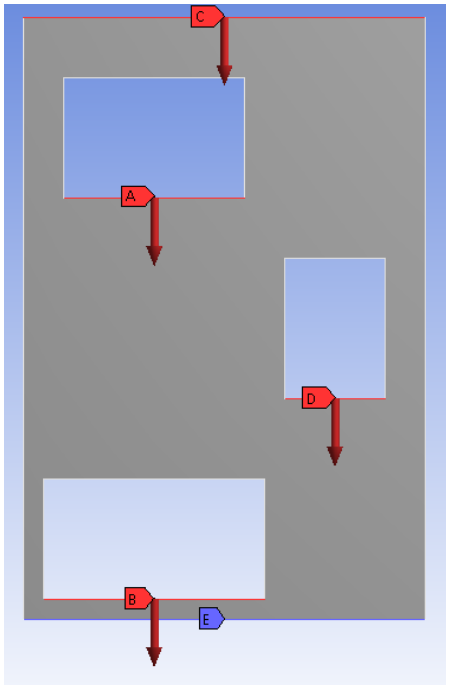
03 Toy problems

TOY2

- Implement area loads
- A void indexing system
  - Multiple voids
  - Complex design spaces
- Forces inside the design space
- User-input for these "rooms"



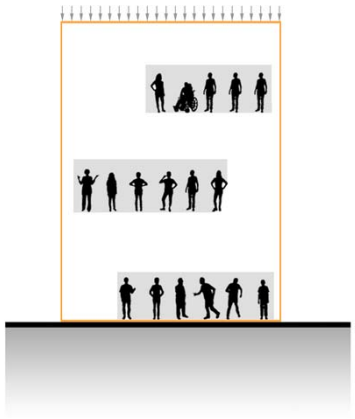


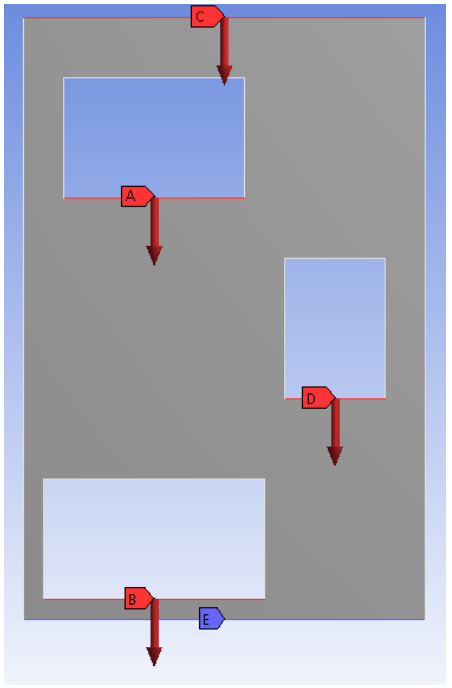


03 Toy problems

TOY2

- Implement area loads
- A void indexing system
  - Multiple voids
  - Complex design spaces
- Forces inside the design space
- User-input for these "rooms"

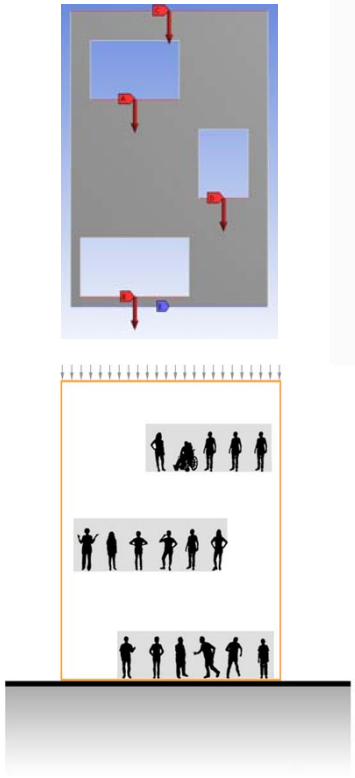


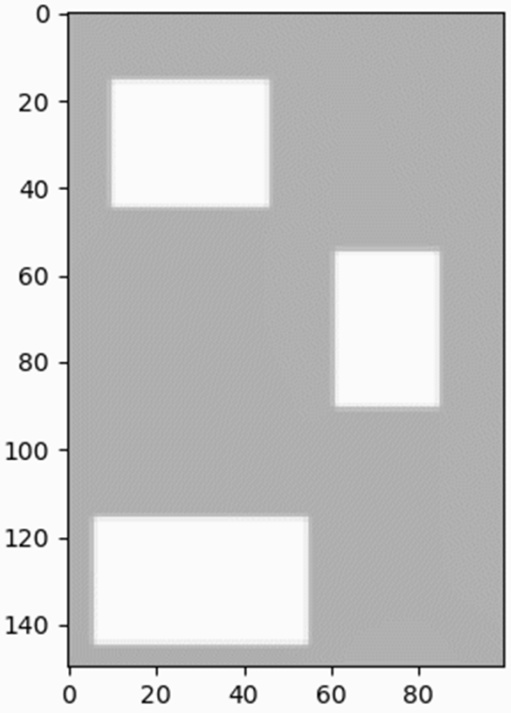


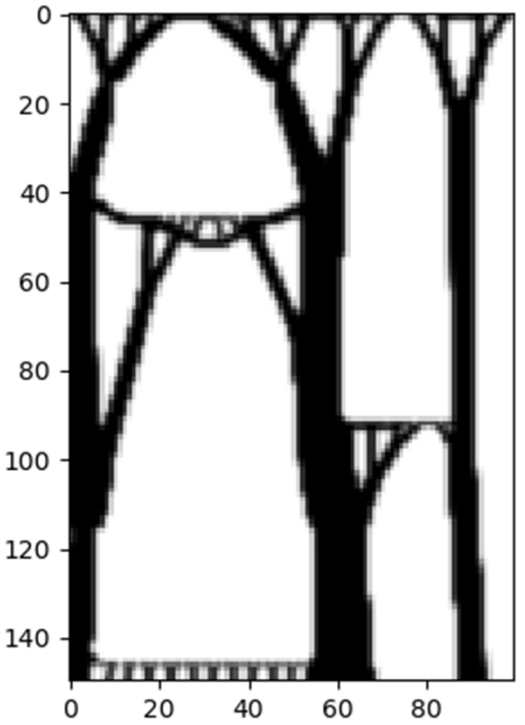

```
#Voidlist contains indexes of the voids
forceValue = 0.01
for void in voidslist:
    if void + 1 is not in voidlist:
        nodetopleft = xvalue * 2*(nely+1)
                              + 2 * yvalue +1
        nodetopright = (xvalue+1) * 2*(nely+1)
                              + 2 * yvalue + 1
        f[nodetopleft,nodetopright]= forceValue/ 2
```

TOY2

- Force size

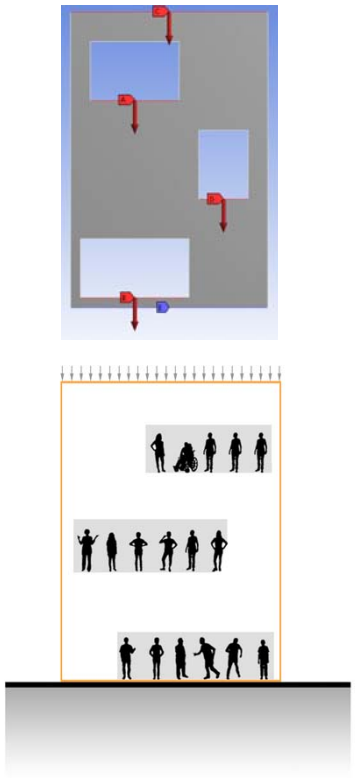


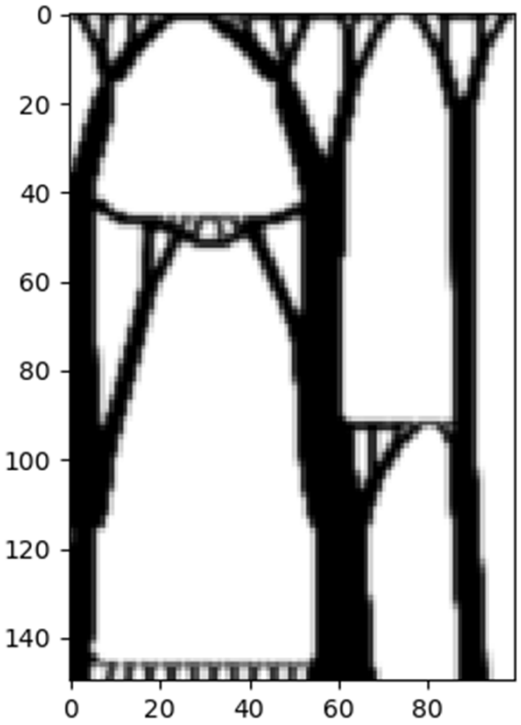


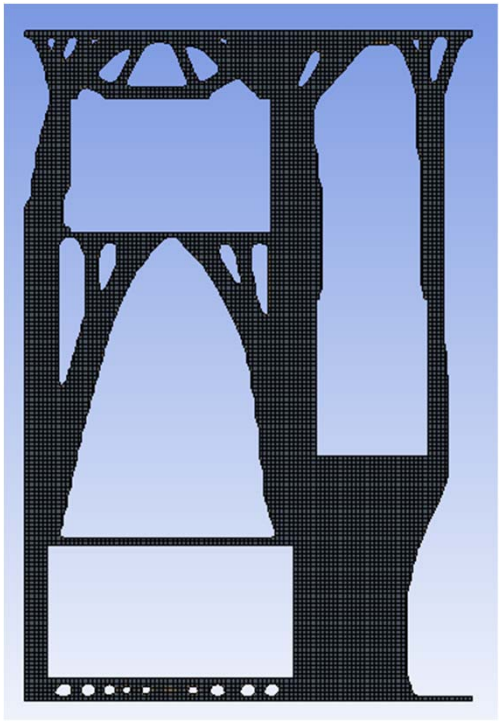


TOY2

- Force size

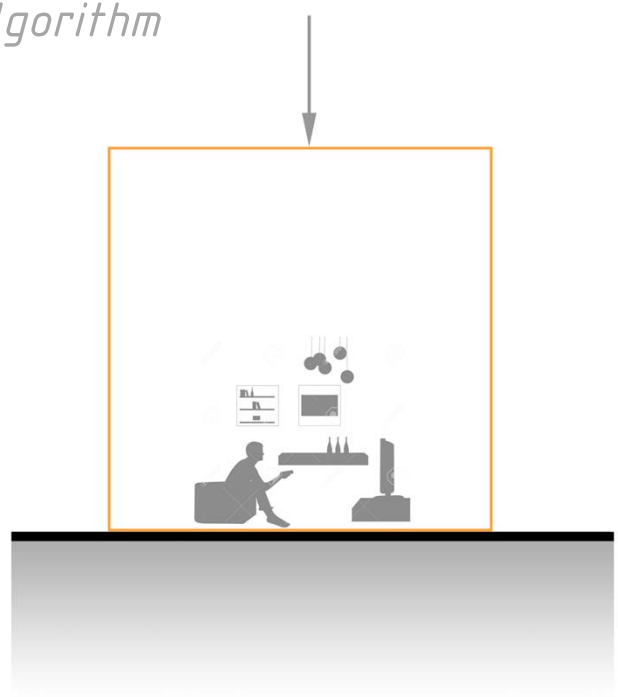






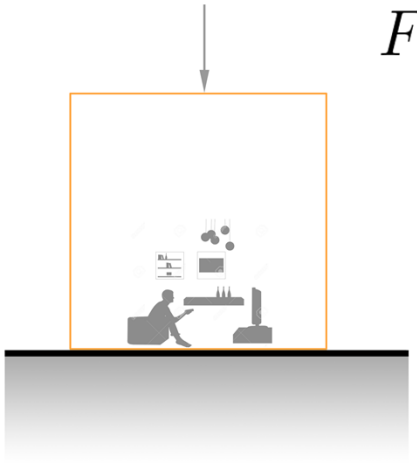
TOY3

implement self-weight in the methodology and in the algorithm

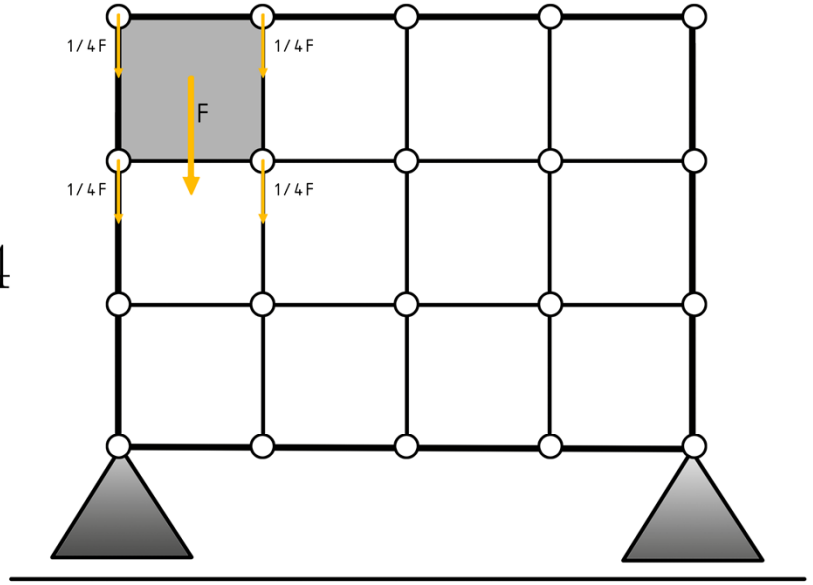


TOY3

- Implement self-weight in the algorithm
- Define sizes of self-weight

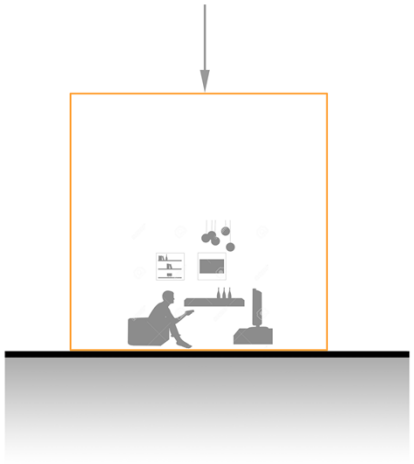


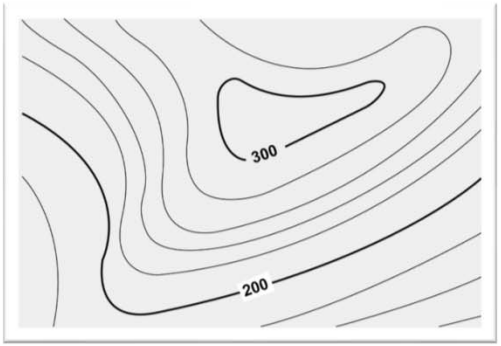
$$F_e = x_e \cdot selfweight/4$$

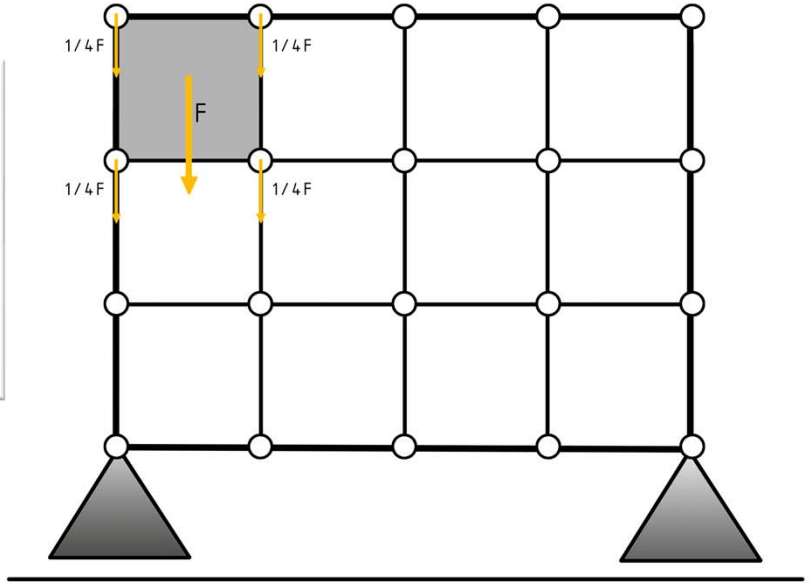


TOY3

- Implement self-weight in the algorithm
- Define sizes of self-weight

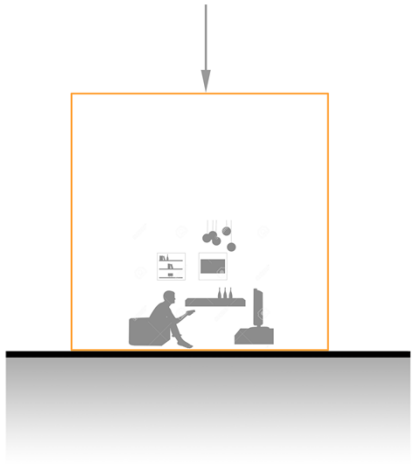


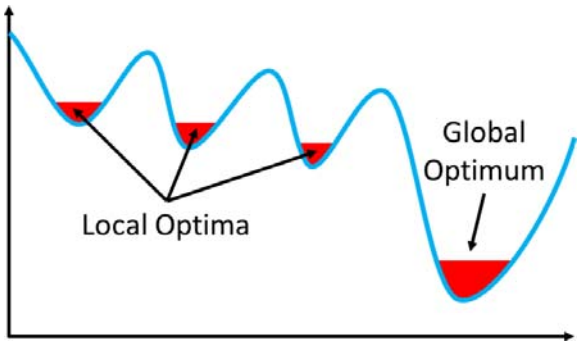


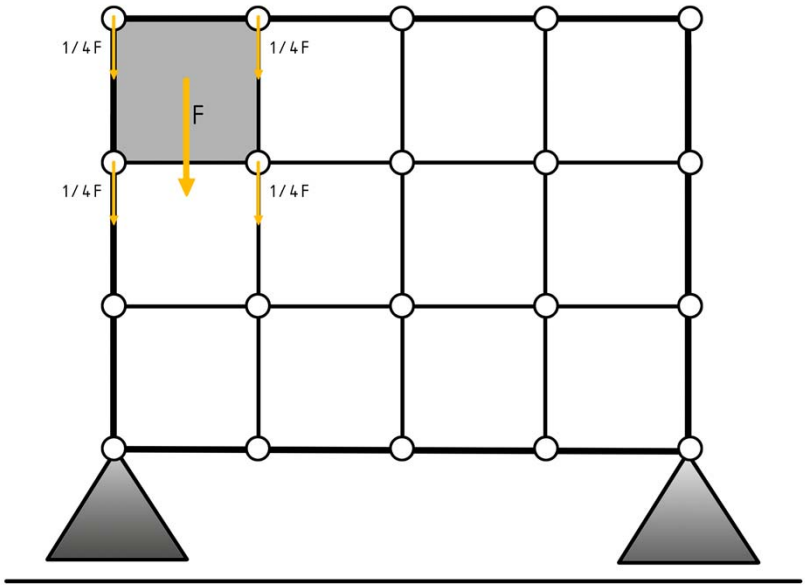


TOY3

- Implement self-weight in the algorithm
- Define sizes of self-weight

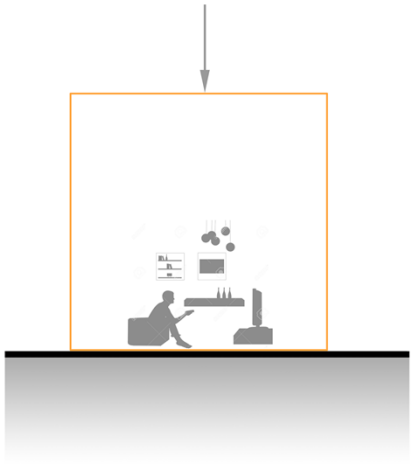


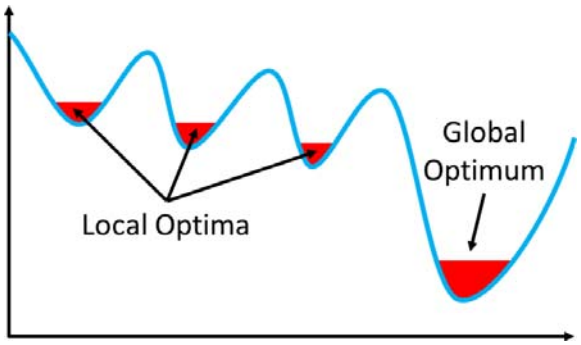




TOY3

- Implement self-weight in the algorithm
- Define sizes of self-weight





$$C(x_e) = F^T U + \lambda(KU - F)$$

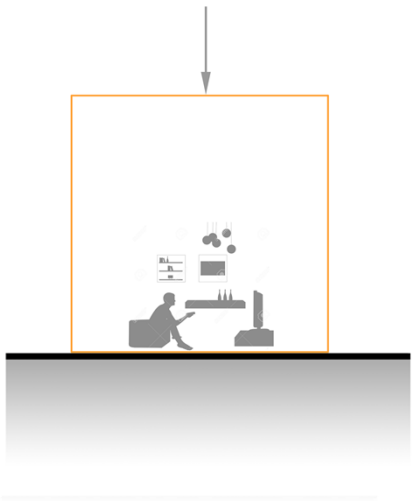
$$\frac{C_e}{x_e} = -px_e^{p-1} - U_e^T \frac{\delta Ke}{\delta x_e} U_e - 2U_e^T \frac{\delta Fe}{\delta x_e}$$

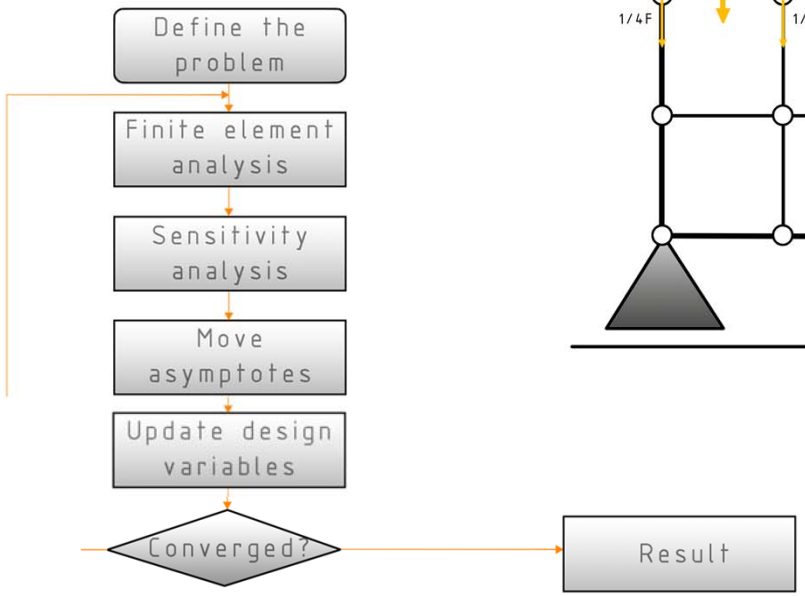
$$F_e = x_e \cdot selfweight/4$$

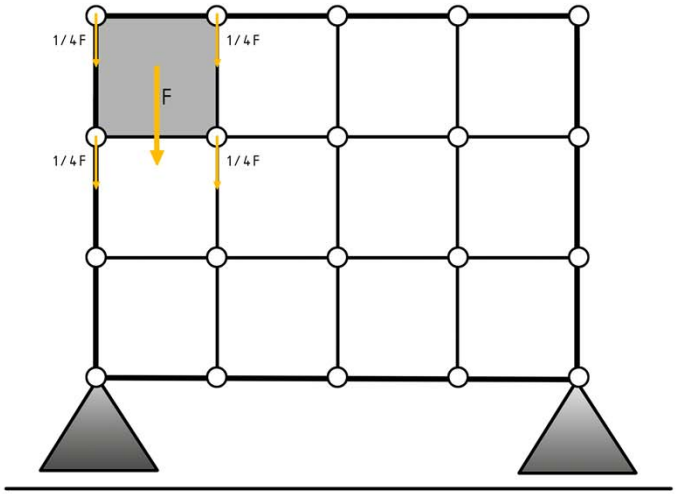
$$\frac{\delta F_e}{\delta x_e} = selfweight/4$$

TOY3

- Implement self-weight in the algorithm
- Define sizes of self-weight

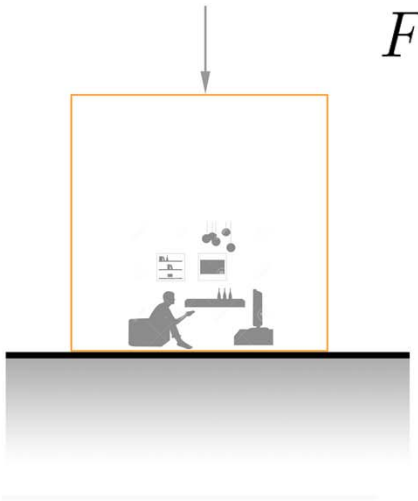




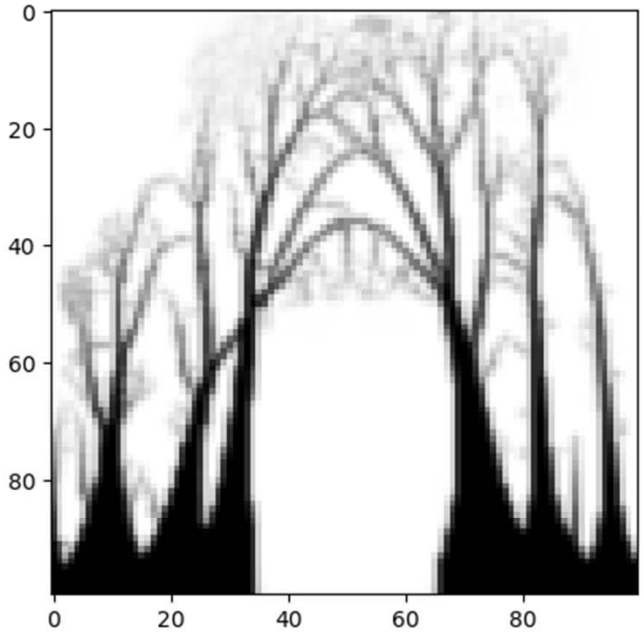


TOY3

- Implement self-weight in the algorithm
- Define sizes of self-weight

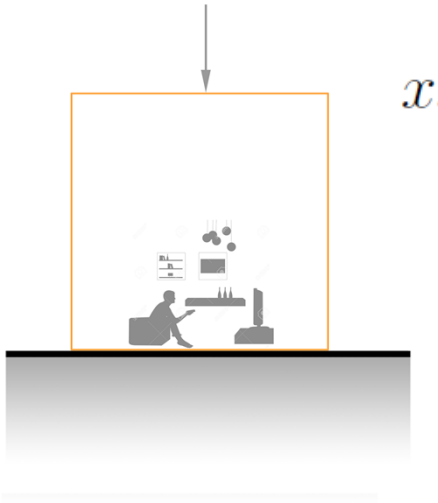


$$F_e = x_e \cdot selfweight/4$$



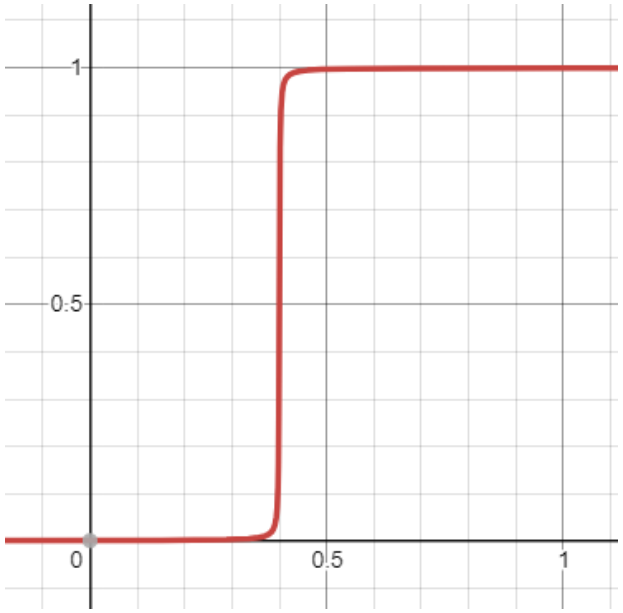
TOY3

- Implement self-weight in the algorithm
- Define sizes of self-weight



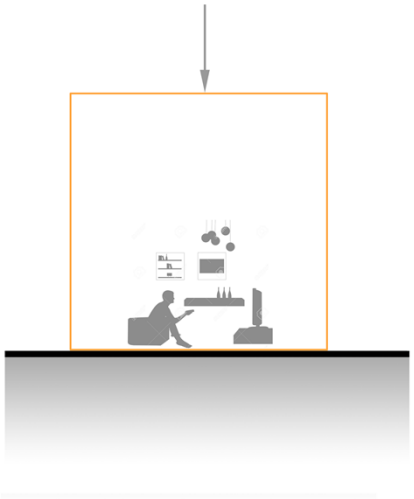
$$F_e = x_e \cdot selfweight/4$$

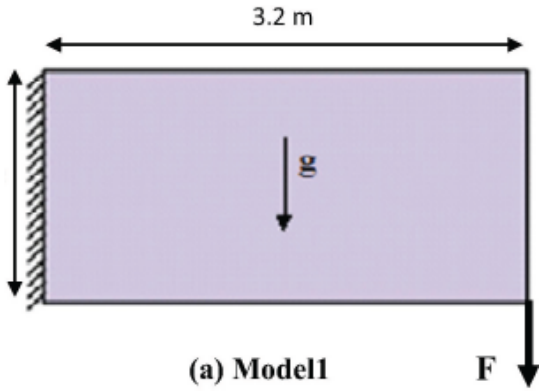
$$x_{rounded} = \frac{1}{2}\frac{1}{\pi} arctan\left(\frac{x-a}{s}\right)$$

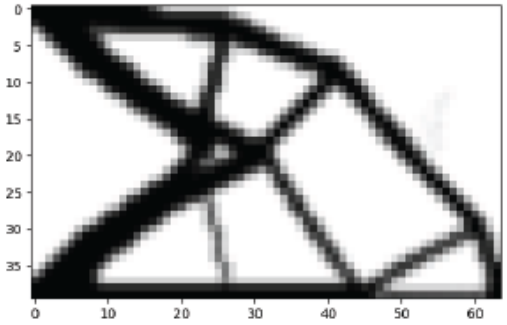


TOY3

- Implement self-weight in the algorithm
- Define sizes of self-weight

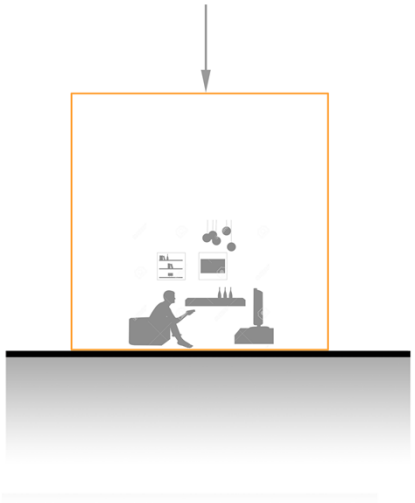


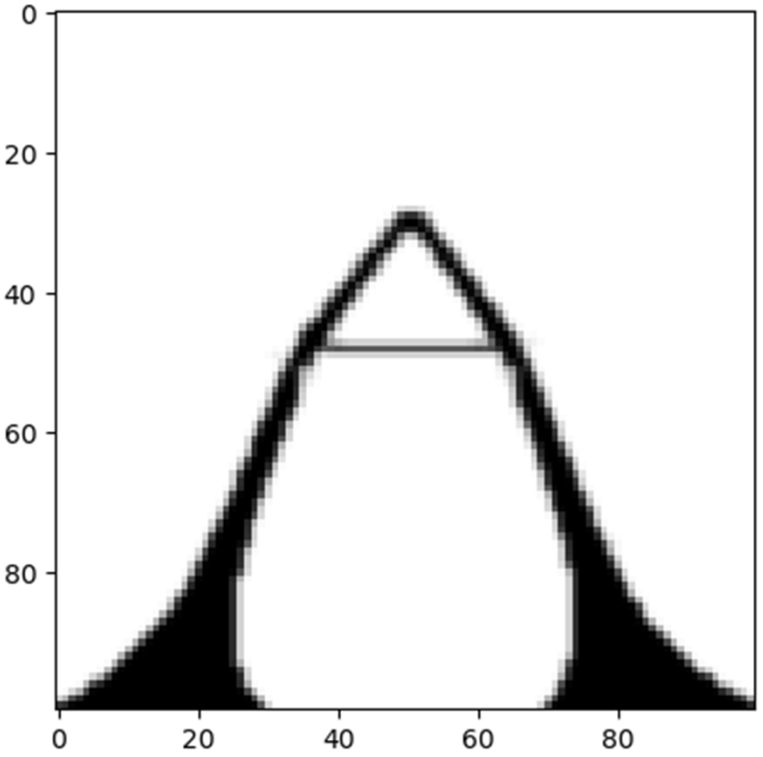




TOY3

Verification

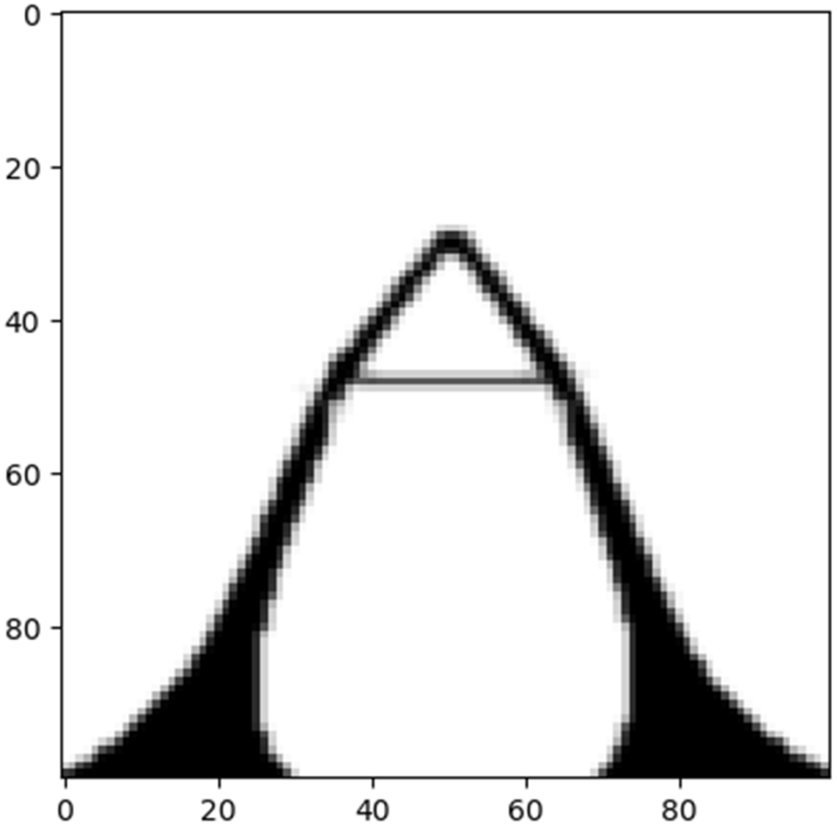


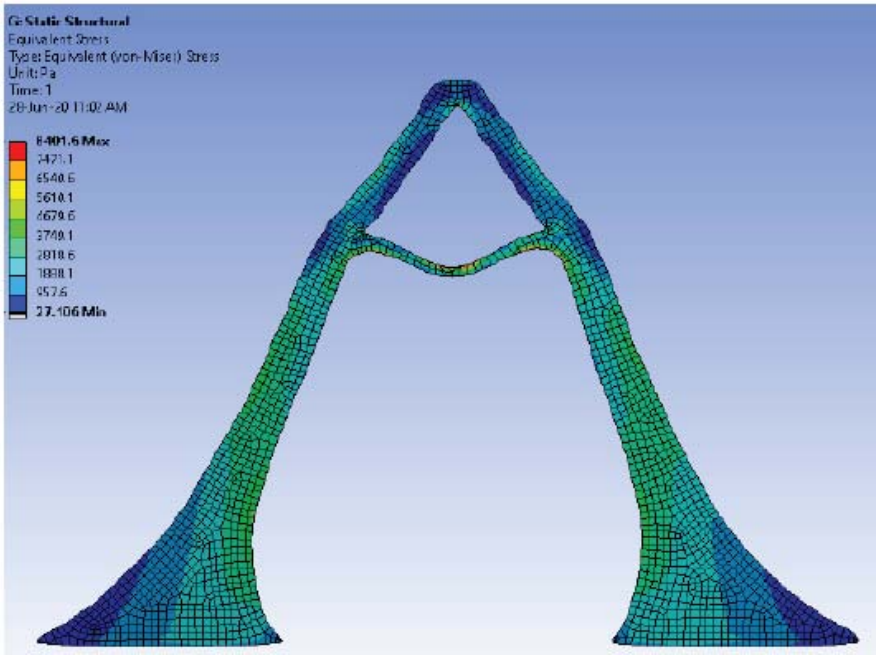


03 Toy problems

TOY3

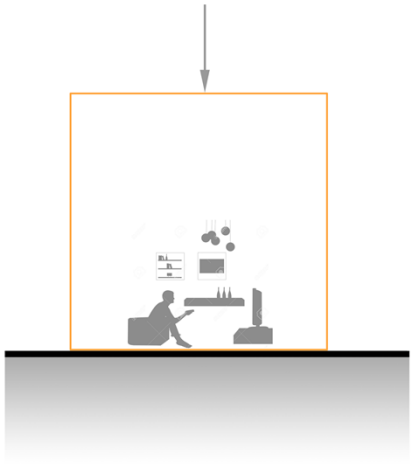
Verification

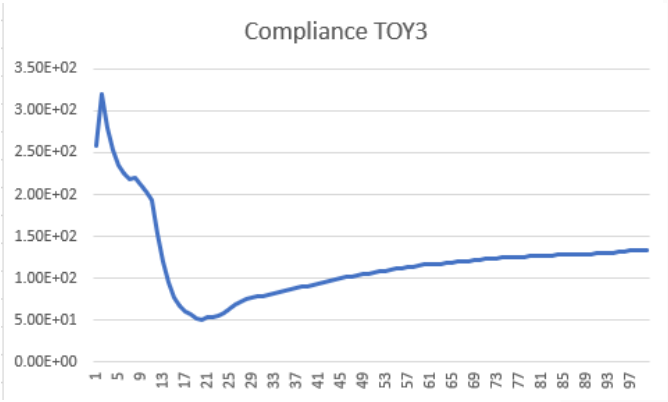


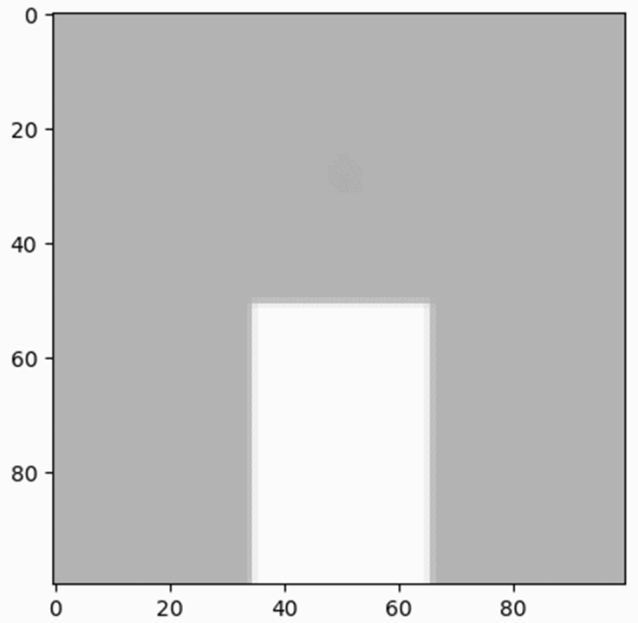


TOY3

Verification

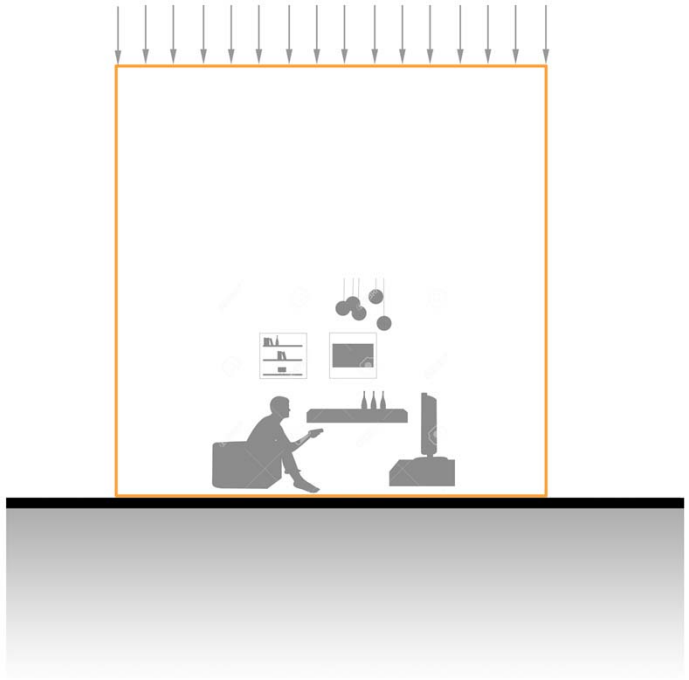






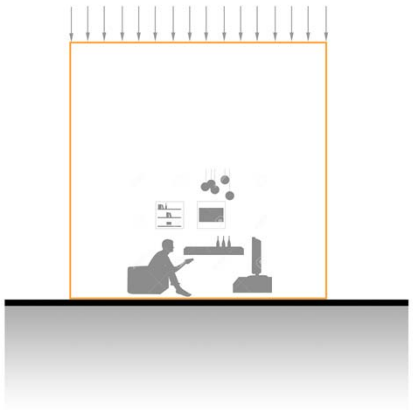
TOY4

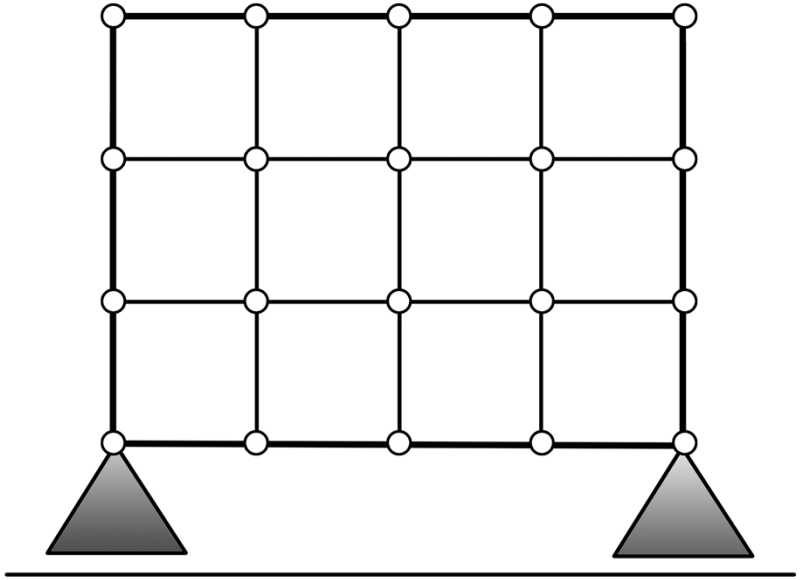
- Implement a roofing constraint
- Add area loads dependent on roof shape



TOY4

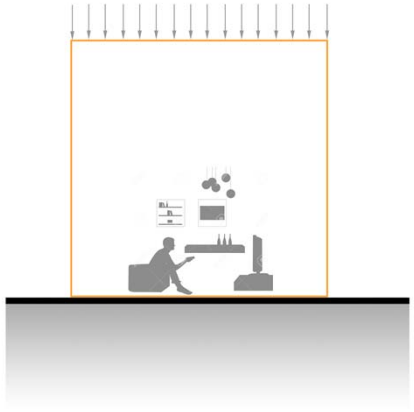
- Implement a roofing constraint
- Add area loads dependent on roof shape

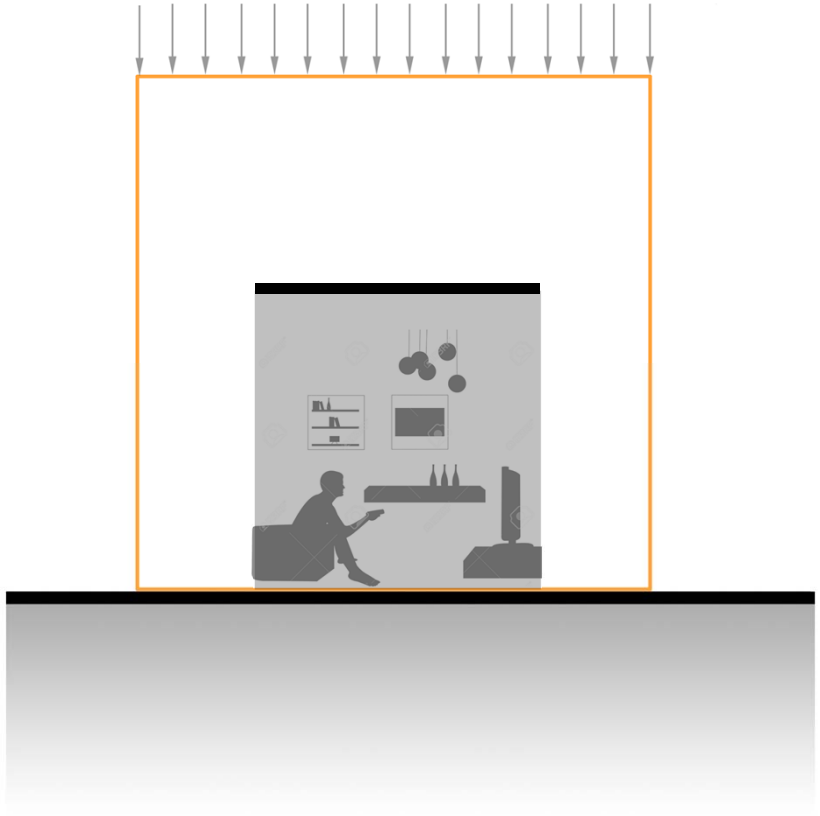




TOY4

- Implement a roofing constraint
- Add area loads dependent on roof shape

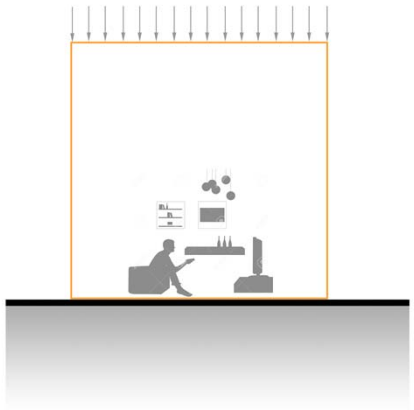


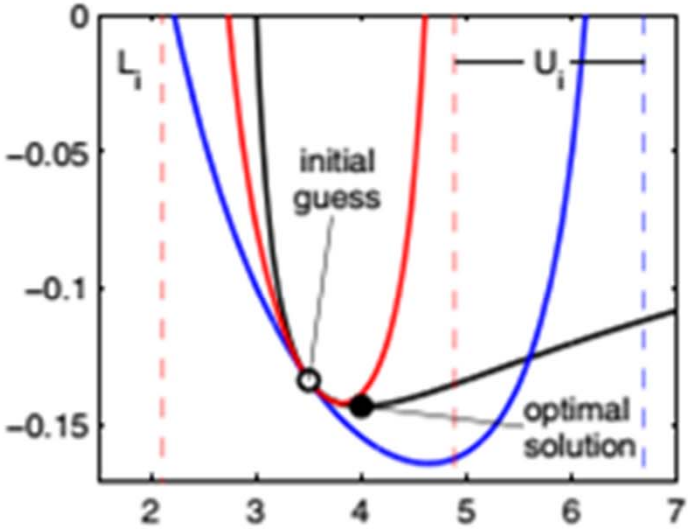


(Blackman & Miller, 2016)

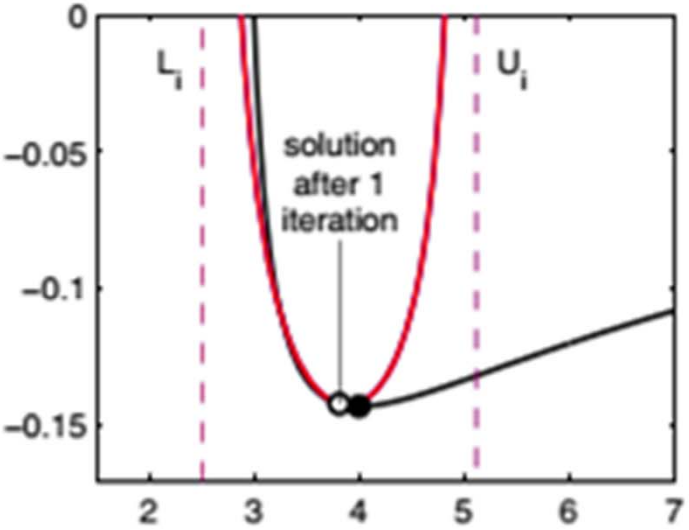
TOY4

- Implement a roofing constraint
- Add area loads dependent on roof shape





(a) MMA, 1st iteration



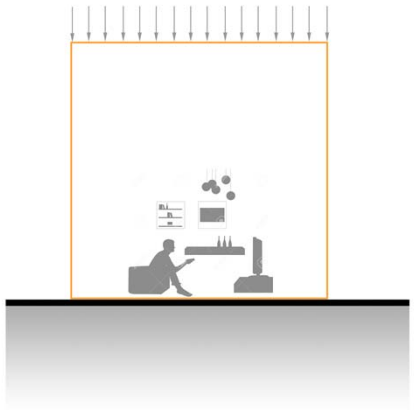
(b) MMA, 2nd iteration

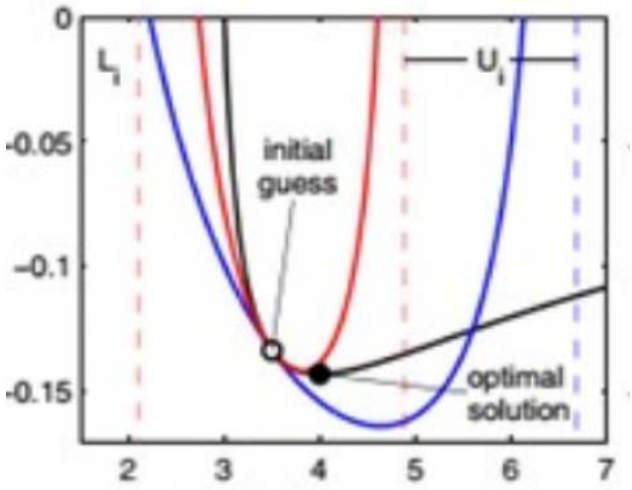
(Blackman & Miller, 2016)

TOY4

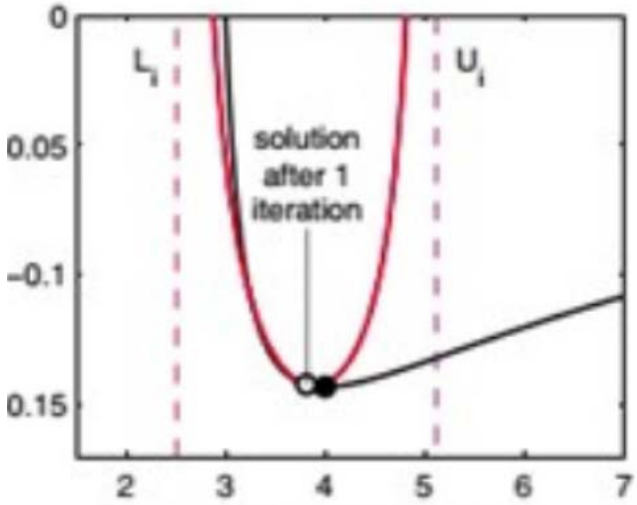
- Implement a roofing constraint
- Add area loads dependent on roof shape

$$g(x) \leq 0$$





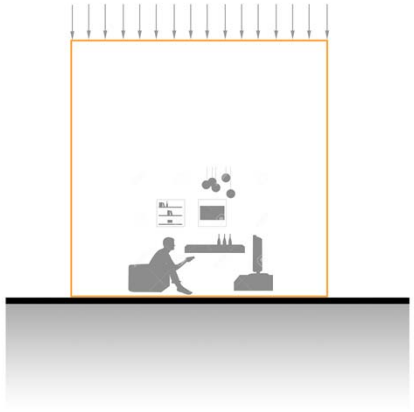
(a) MMA, 1st iteration

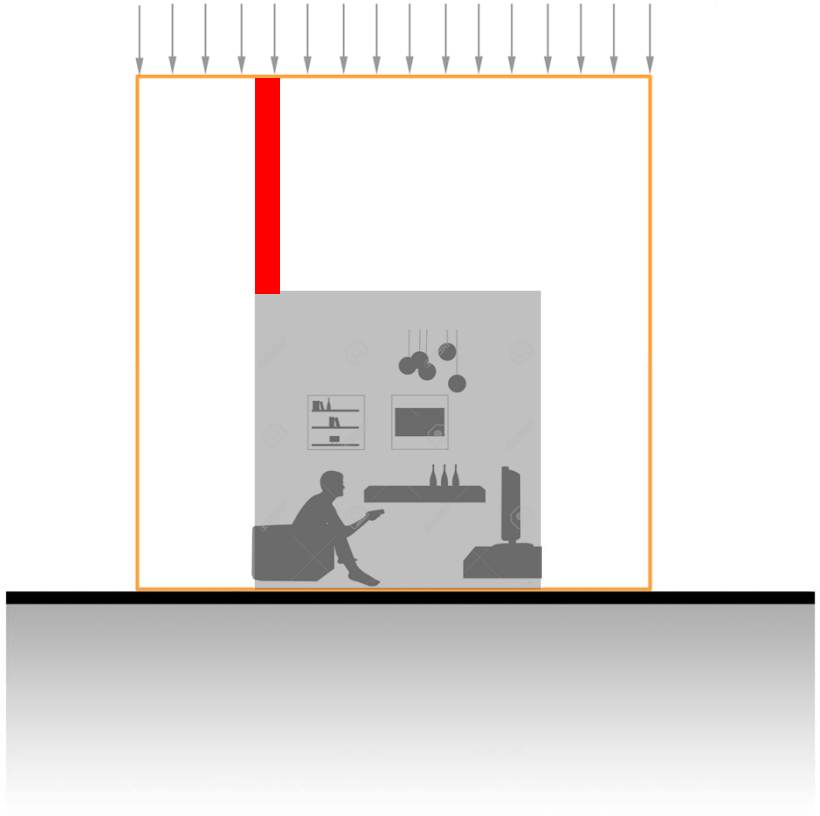


(b) MMA, 2nd iteration

TOY4

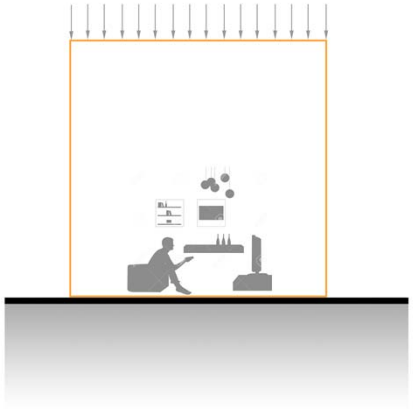
- Implement a roofing constraint
- Add area loads dependent on roof shape

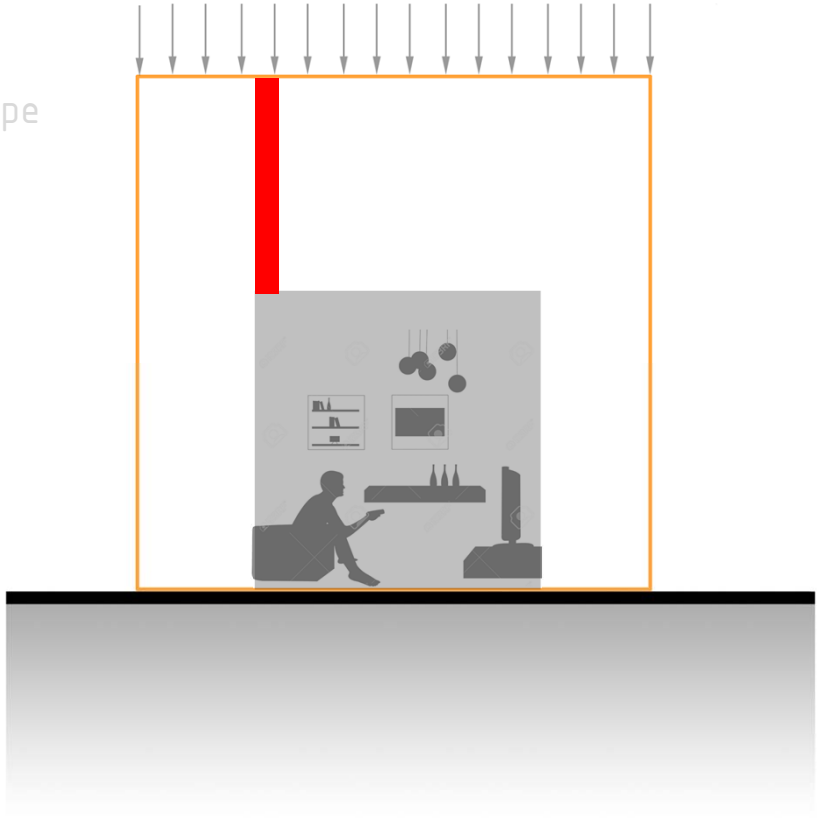




TOY4

- Implement a roofing constraint
- Add area loads dependent on roof shape

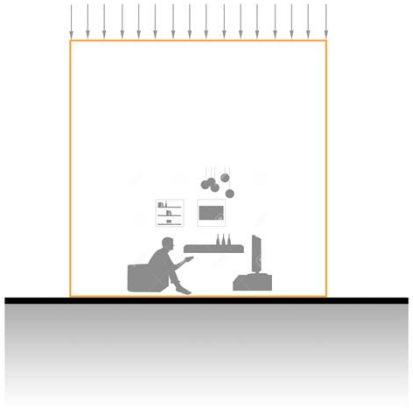


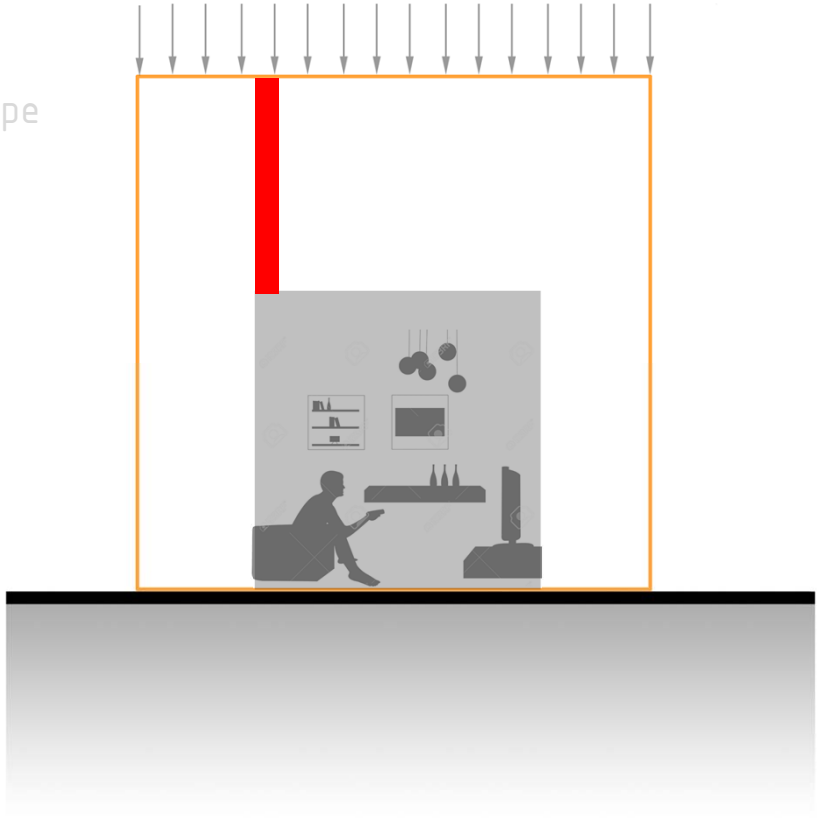


$$\sum(x_{k,i}) \geq 1$$

TOY4

- Implement a roofing constraint
- Add area loads dependent on roof shape



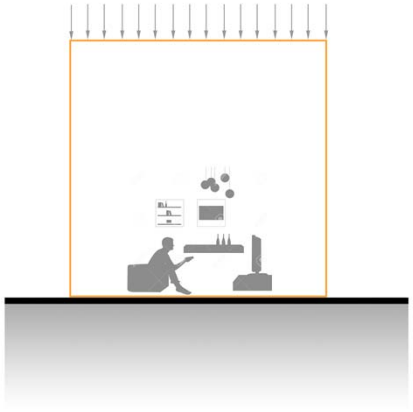


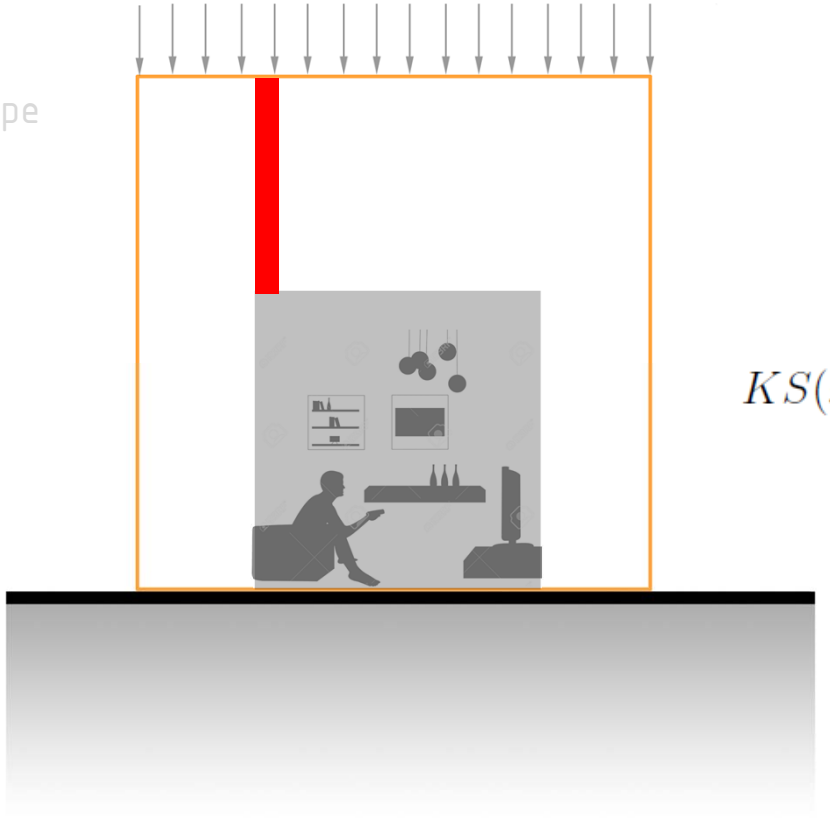
$$\sum(x_{k,i}) \geq 1$$

$$g_k = 1 - \sum_{column_k} x_{k,i}$$

TOY4

- Implement a roofing constraint
- Add area loads dependent on roof shape





$$\sum(x_{k,i}) \geq 1$$

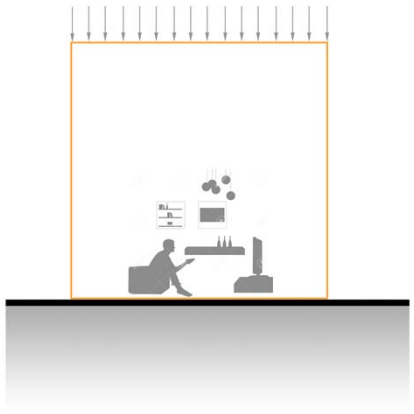
$$g_k = 1 - \sum_{column_k} x_{k,i}$$

$$KS(x) = \frac{1}{P} ln(\sum_{k \in K} e^{P g_k}) \leq 0$$

TOY4

- Implement a roofing constraint
- Add area loads dependent on roof shape

$$\frac{\delta \widetilde{g}}{\delta x_{k,i}} = \frac{\delta \widetilde{g}}{\delta g_k}\frac{\delta g_k}{\delta x_{k,i}} = -\frac{e^{Pg_k}}{\sum_{k\in K} e^{Pg_k}}$$

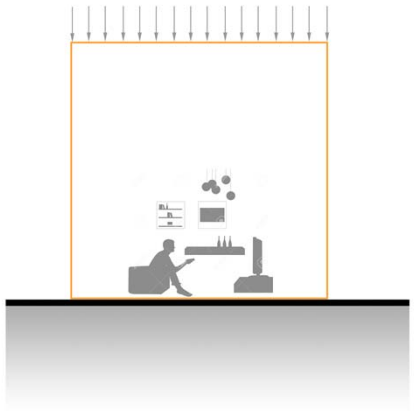


$$\frac{\delta \widetilde{g}}{\delta x_{k,i}} = \begin{cases} \textbf{if } \mathbf{k} \in \mathbf{K} : -\frac{e^{Pg_k}}{\sum_{k\in K} e^{Pg_k}} \\ \textbf{if } \mathbf{k} \notin \mathbf{K} : 0 \end{cases}$$

TOY4

- Implement a roofing constraint
- Add area loads dependent on roof shape

$$\frac{\delta \tilde{g}}{\delta x_{k,i}} = \frac{\delta \tilde{g}}{\delta g_k}\frac{\delta g_k}{\delta x_{k,i}} = -\frac{e^{Pg_k}}{\sum_{k \in K} e^{Pg_k}}$$

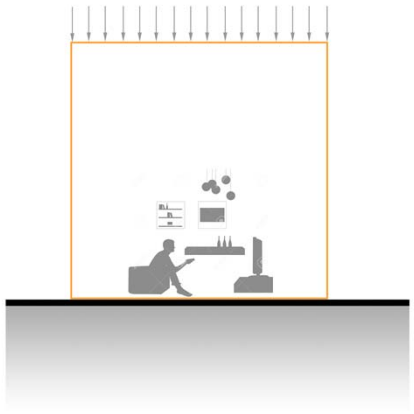


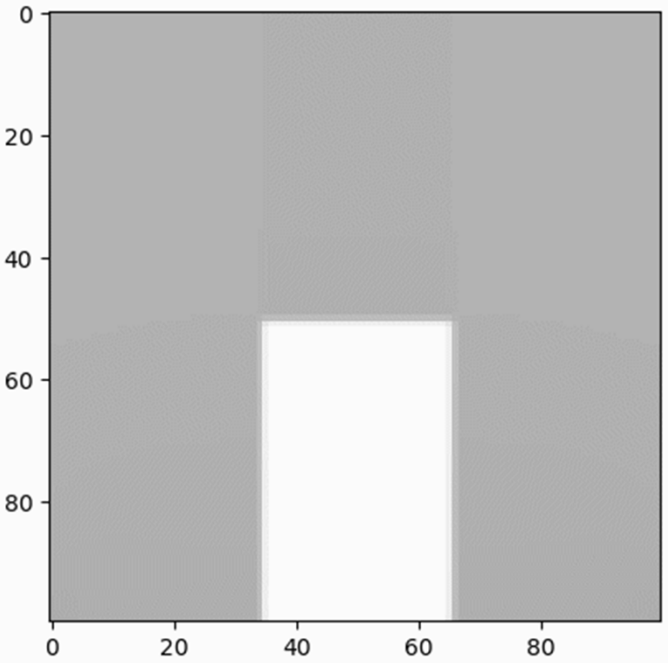
$$\frac{\delta \tilde{g}}{\delta x_{k,i}} = \begin{cases} \text{if } \mathbf{k} \in \mathbf{K} : -\frac{e^{Pg_k}}{\sum_{k \in K} e^{Pg_k}} \\ \text{if } \mathbf{k} \notin \mathbf{K} : 0 \end{cases}$$

```
voidcolumns = columnindex(sum(voidsMatrix)>1)
invertedarea = 1 - voids[voidscolumns]

epgk = e ^ (10 * sumofcolumns(x ^ p))
gcolumns = np.log(sum(epgk))/10

dcroof = epgk / gcolumns
dcrc[voidcolumns] = tile(dcroof, nely) * invertedarea
```

TOY4

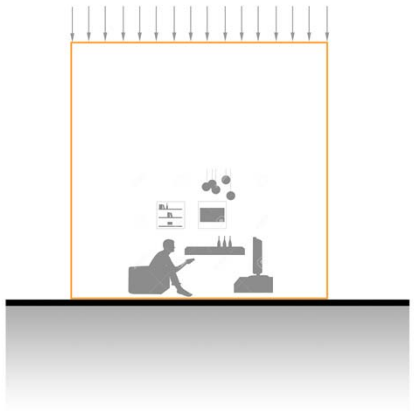
- Implement a roofing constraint
- Add area loads dependent on roof shape

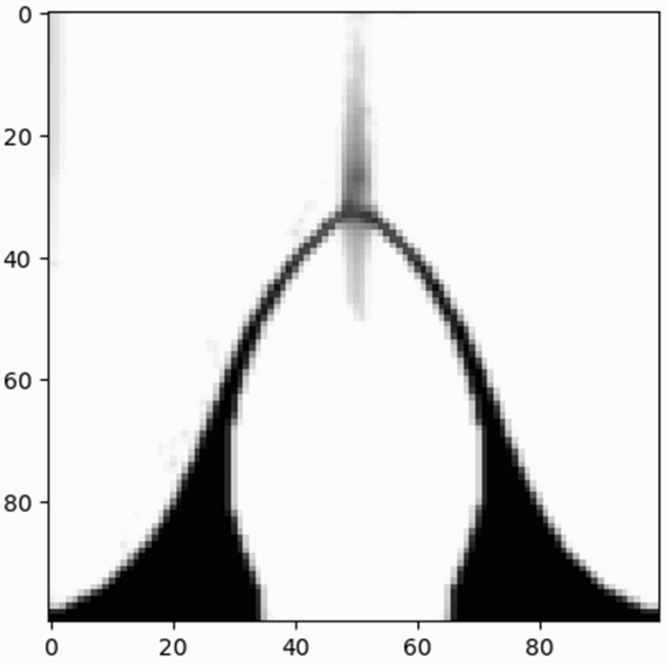




TOY4

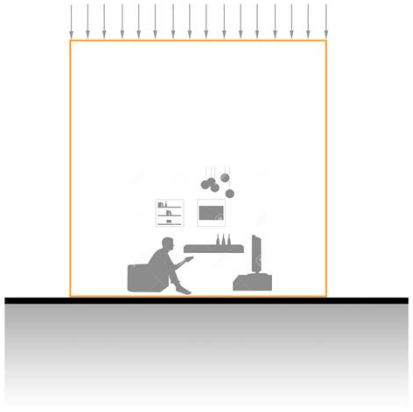
- Implement a roofing constraint
- Add area loads dependent on roof shape

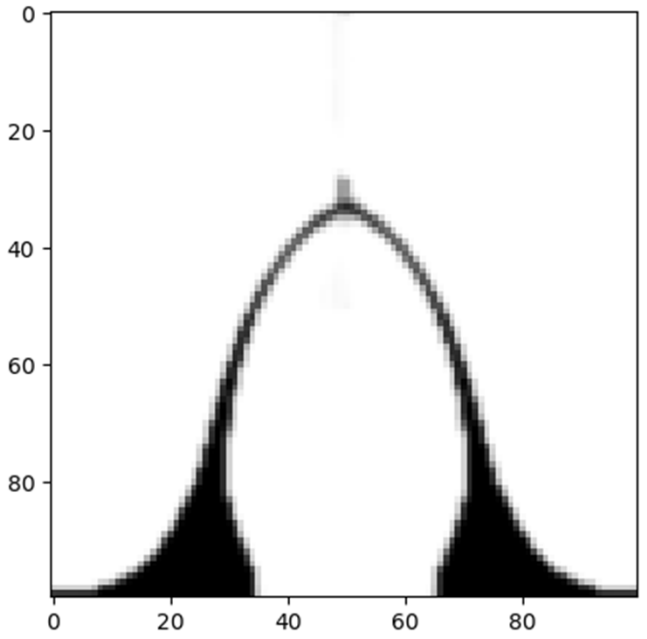




TOY4

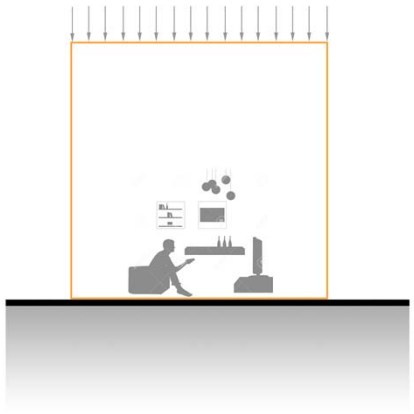
- Implement a roofing constraint
- Add area loads dependent on roof shape

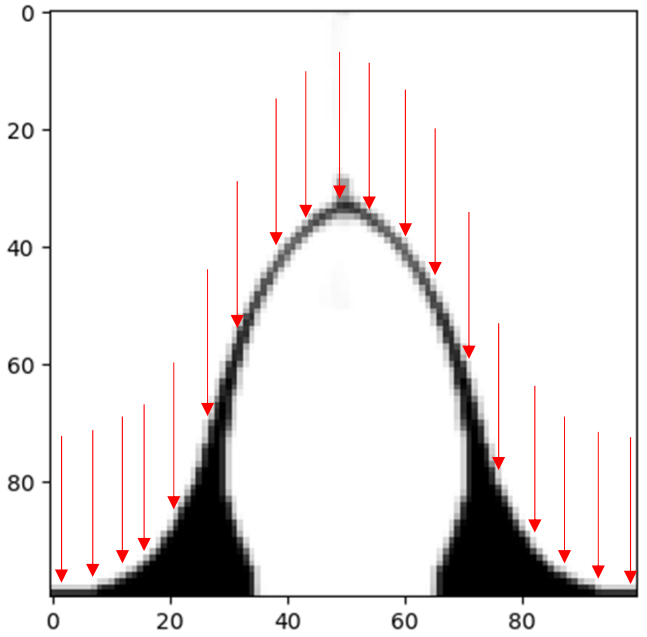




TOY4

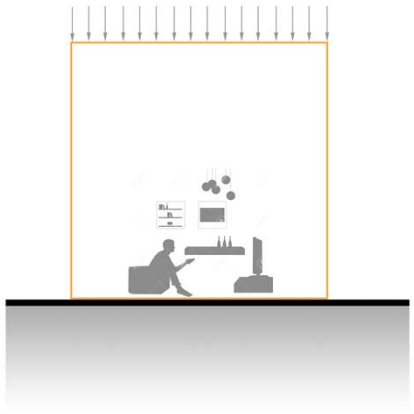
- Implement a roofing constraint
- Add area loads dependent on roof shape

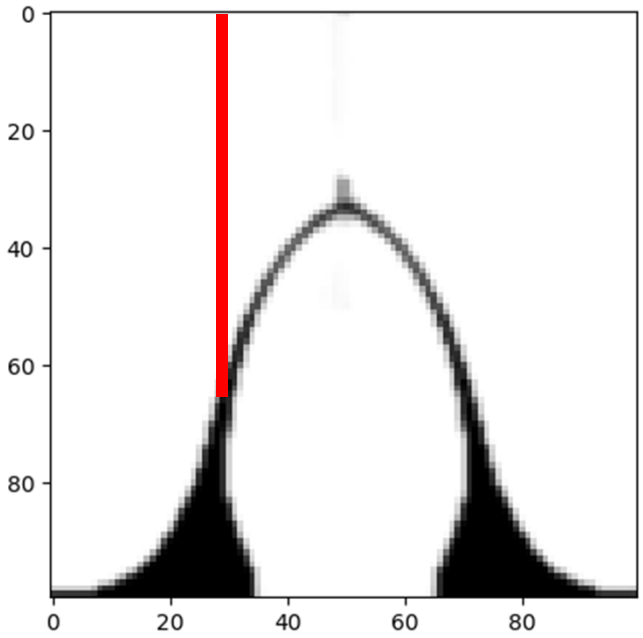




TOY4

- Implement a roofing constraint
- Add area loads dependent on roof shape

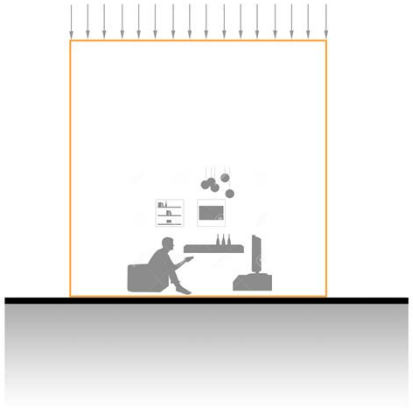


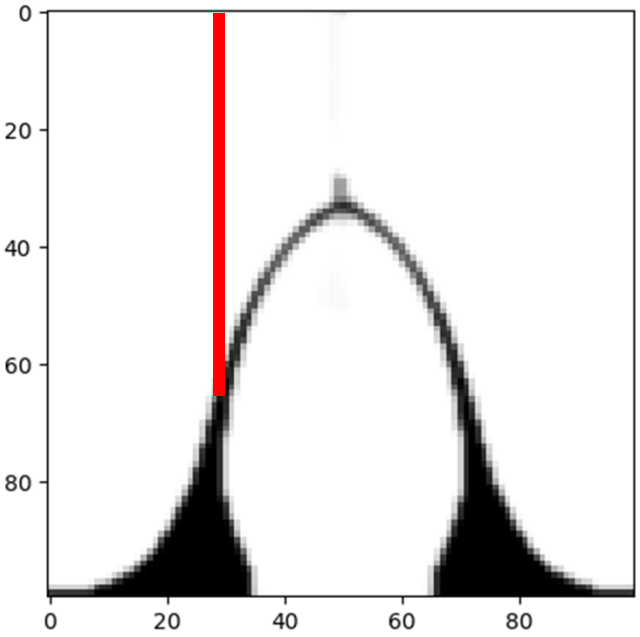


$$y_{i,j} = \sum_{k=j}^{nele} x_{[i,k]}$$

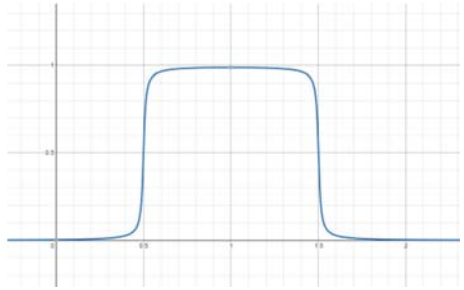
TOY4

- Implement a roofing constraint
- Add area loads dependent on roof shape





$$y_{i,j} = \sum_{k=j}^{nele} x_{[i,k]}$$



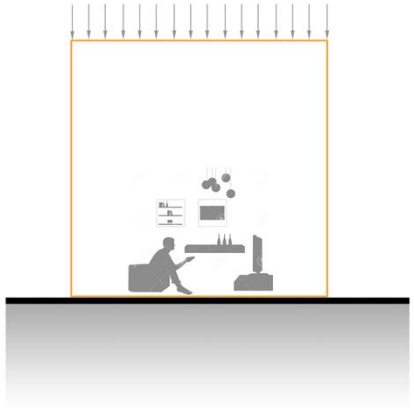
$$F_{nodesof[i,j]} = \frac{snowFactor}{4} f(y_{[i,j]})$$

TOY4

- Implement a roofing constraint
- Add area loads dependent on roof shape

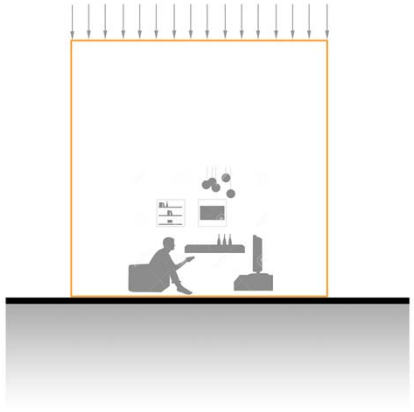
$$\frac{C_e}{x_e} = -px_e^{p-1} - U_e^T \frac{\delta Ke}{\delta x_e} U_e - 2U_e^T \frac{\delta Fe}{\delta x_e}$$

$$F_e = F_{preset} + F_{self} + F_{snow}$$



TOY4

- Implement a roofing constraint
- Add area loads dependent on roof shape



$$\frac{C_e}{x_e} = -px_e^{p-1} - U_e^T \frac{\delta Ke}{\delta x_e} U_e - 2U_e^T \frac{\delta Fe}{\delta x_e}$$

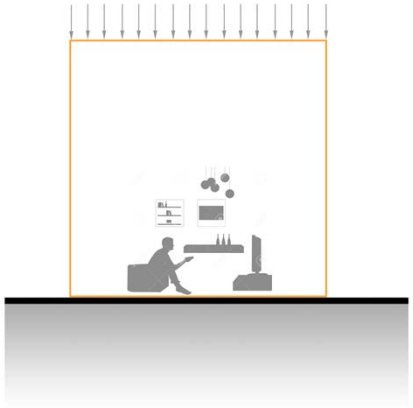
$$F_e = F_{preset} + F_{self} + F_{snow}$$

$$\frac{\delta F_{e[i,j]}}{\delta x_{[i,j]}} = \frac{snowFactor}{4} \frac{\delta f}{\delta y_{[i,j]}} \frac{\delta y_{[i,j]}}{\delta x_{[i,k]}}$$

$$\frac{\delta y_{[i,j]}}{\delta x_{[i,k]}} \begin{cases} 1 & k \geq j \\ 0 & k < j \end{cases}$$

TOY4

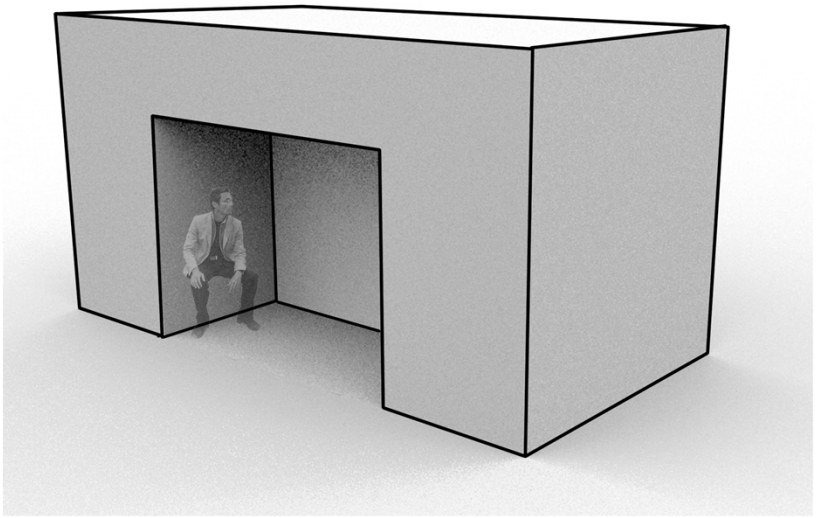
- Implement a roofing constraint
- Add area loads dependent on roof shape



03 Toy problems

TOY5

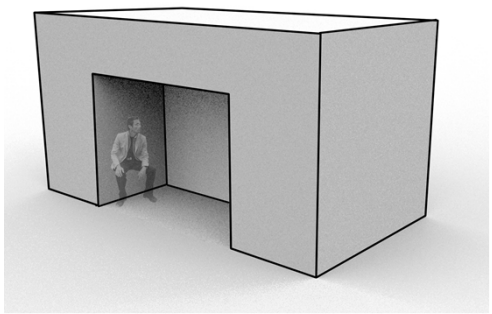
- Implement an indexing system
- Handle inputs, including voids
- Algorithm optimization



(Liu & Tovar, 2014)

TOY5

- Implement an indexing system
- Handle inputs, including voids
- Algorithm optimization



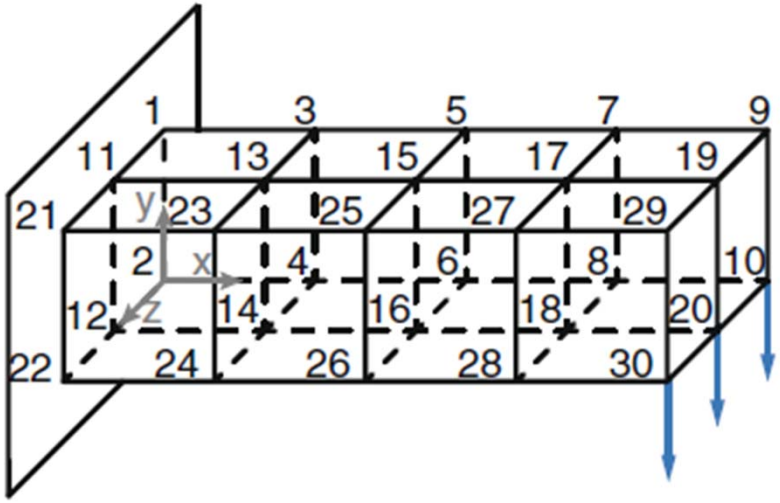


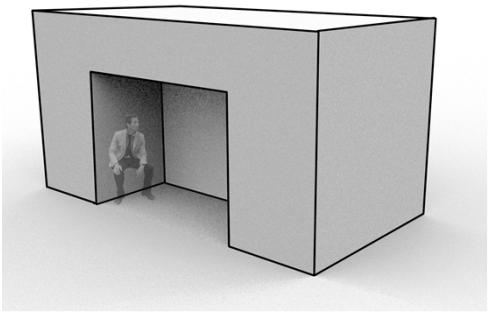
Fig. 2 Global node IDs in a prismatic structure composed of 8 elements

03 Toy problems

(QNCC, 2019)

TOY5

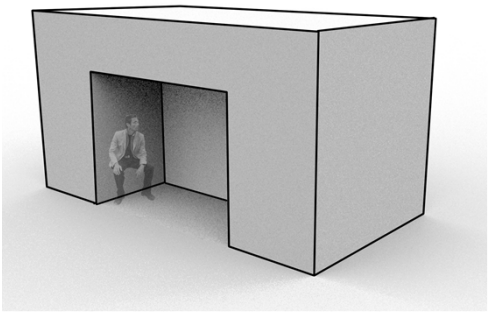
- Implement an indexing system
- Handle inputs, including voids
- Algorithm optimization

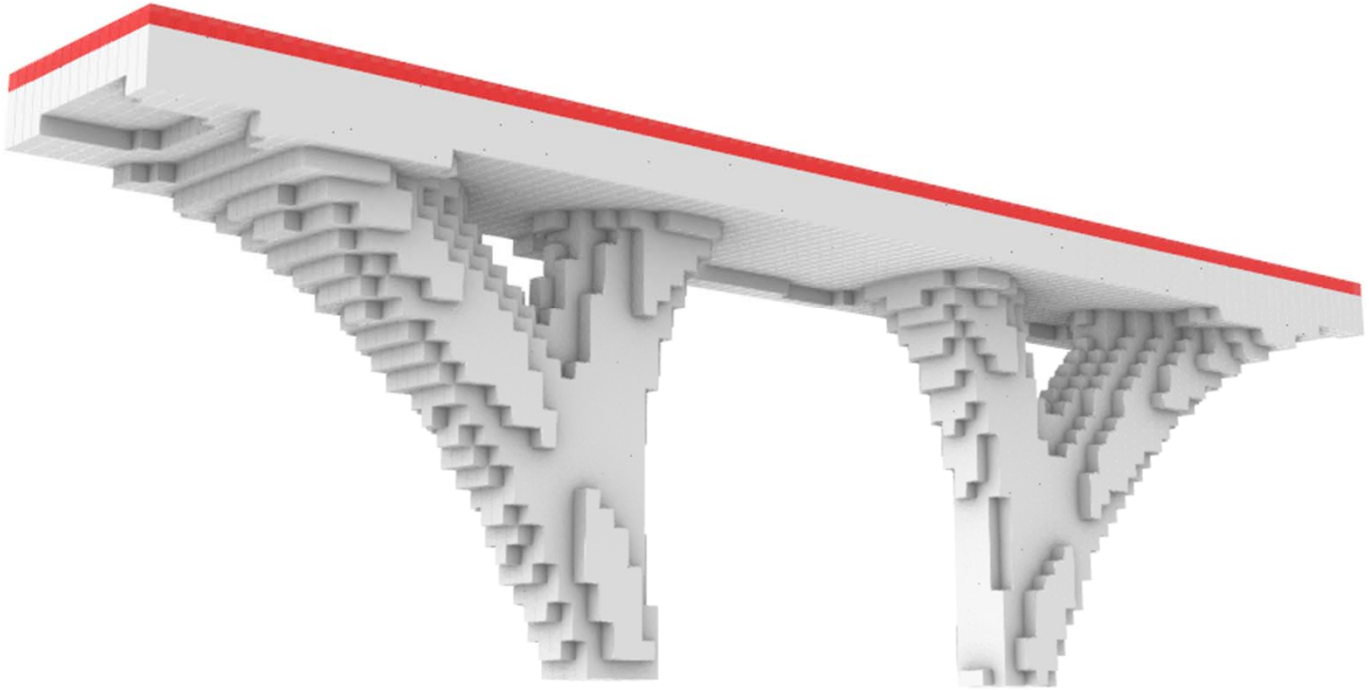


03 Toy problems

TOY5

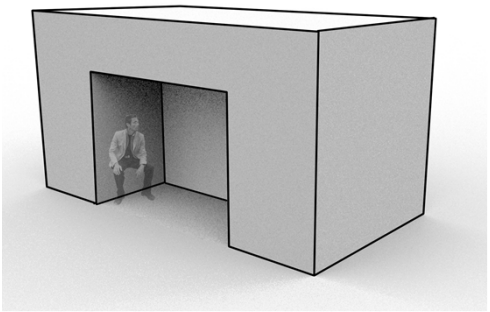
- Implement an indexing system
- Handle inputs, including voids
- Algorithm optimization

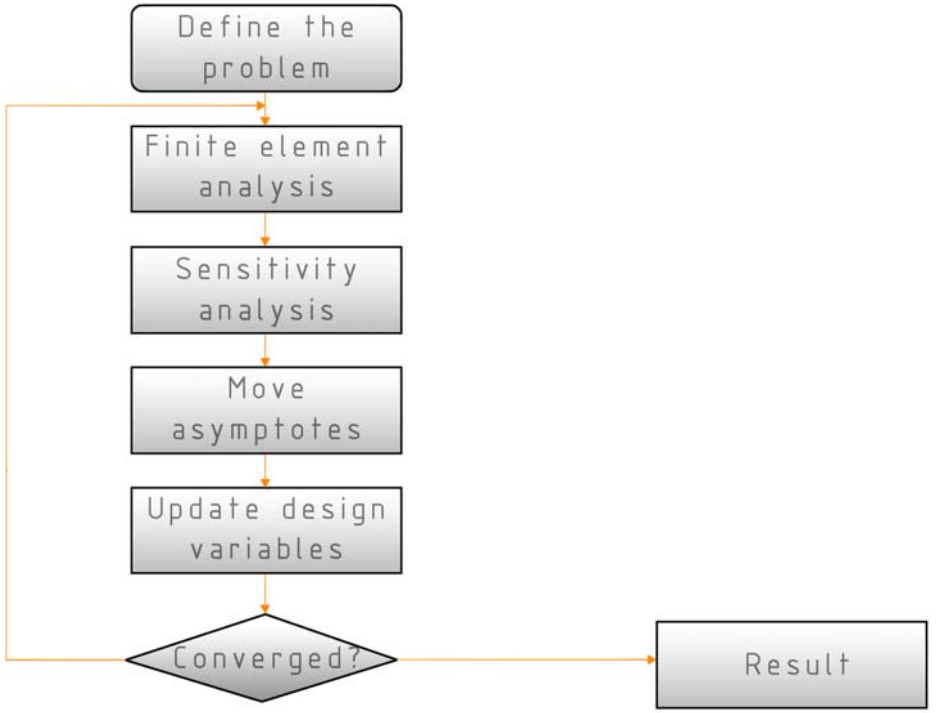




TOY5

- Implement an indexing system
- Handle inputs, including voids
- Algorithm optimization

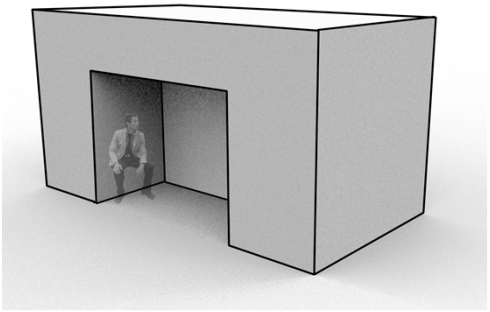




TOY5

- Implement an indexing system
- Handle inputs, including voids
- Algorithm optimization

```
voidList = [ ]
for each element in the designspace:
    isPointInside = void.contains(element)
    nodeID = yvalue*((nelx)*(nely)) + xvalue*(nelz)
                              + (-nelz + nely)-1
    if isPointInside == True:
          voidList.addtoList(nodeID)
```

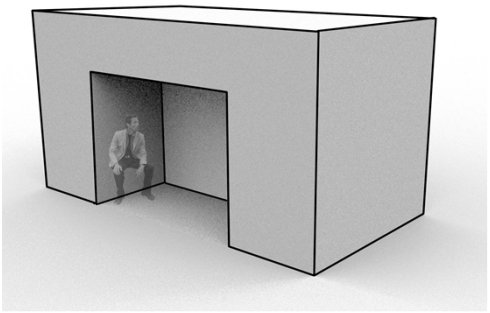


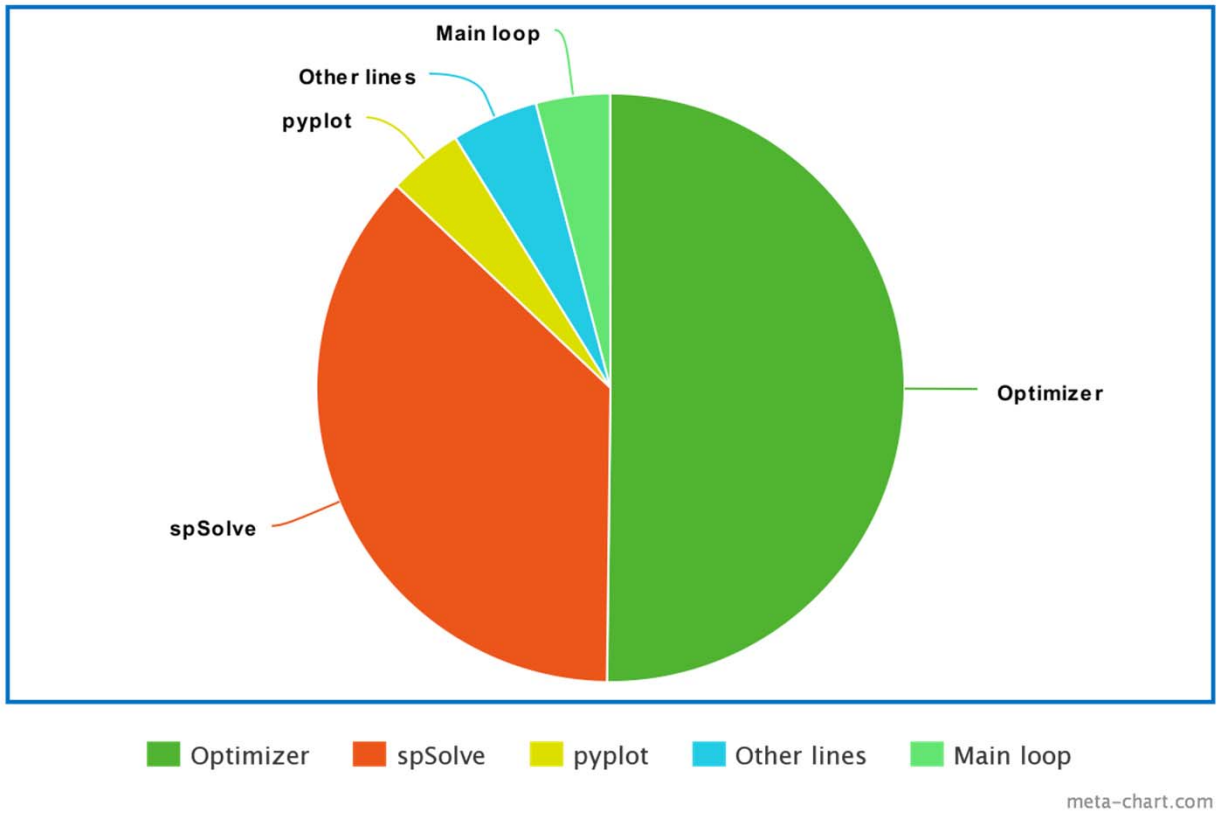
```
voids[voidslist]=1
where voids = 1, x = 0.001
else: x = x
```

03 Toy problems

TOY5

- Implement an indexing system
- Handle inputs, including voids
- Algorithm optimization



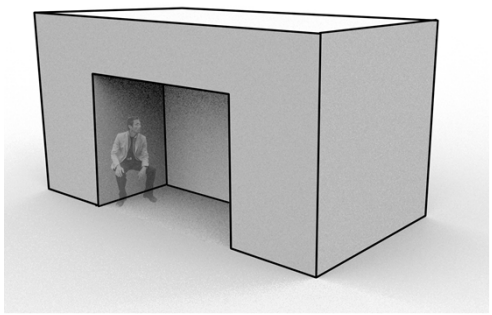


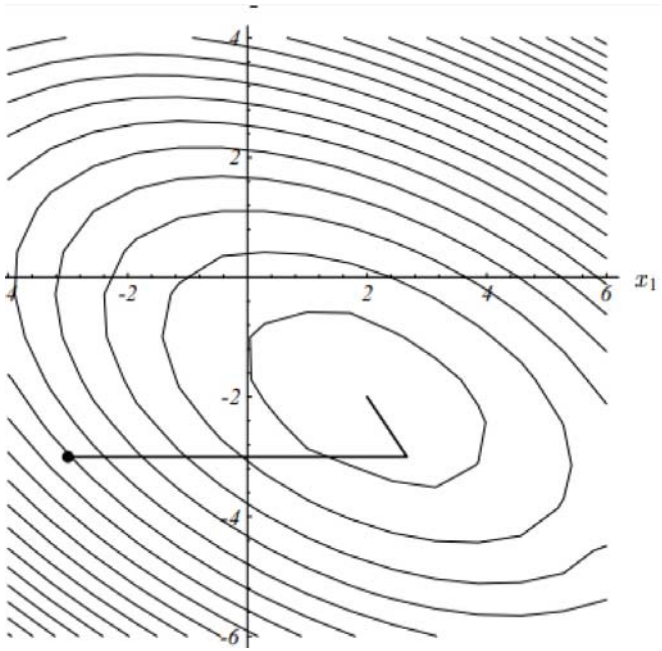
03 Toy problems

(Shewchuck, 1997)

TOY5

- Implement an indexing system
- Handle inputs, including voids
- Algorithm optimization



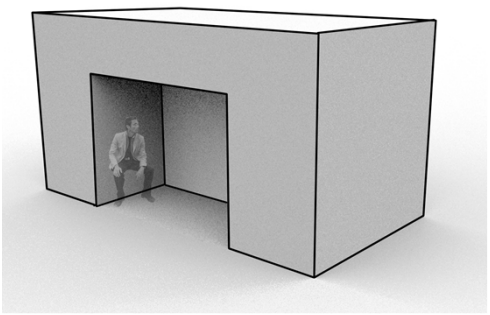


TOY5

- Implement an indexing system
- Handle inputs, including voids
- Algorithm optimization

$$M^{-1}Ax = M^{-1}b$$

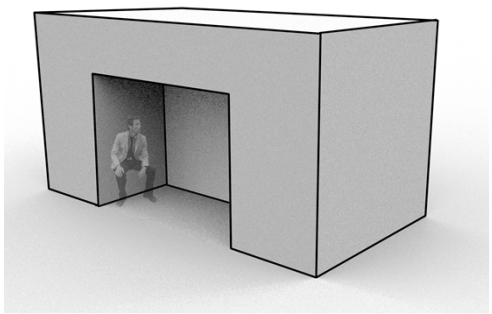
$$M = diag(A)$$

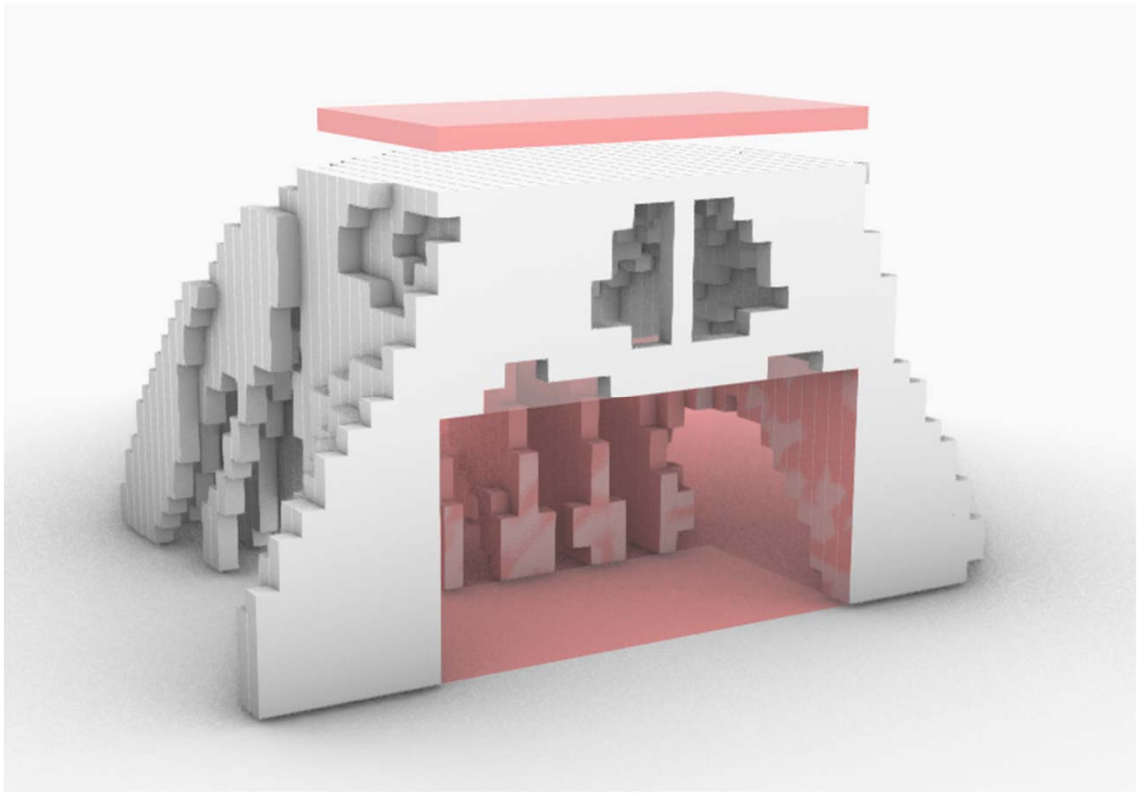


spsolve(K,f)	gmres(K,f)	cg(K,f,M=1/Jac)	cg(K,f,M=Jac)
Total(sec)			
547.44	805.96	189.43	302.07

TOY5

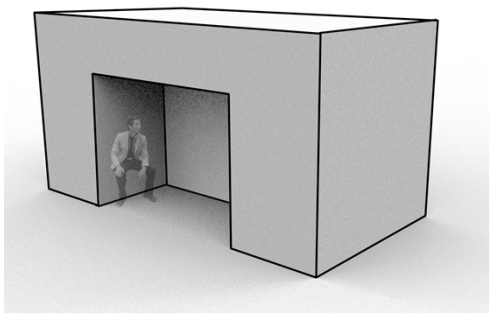
- Result

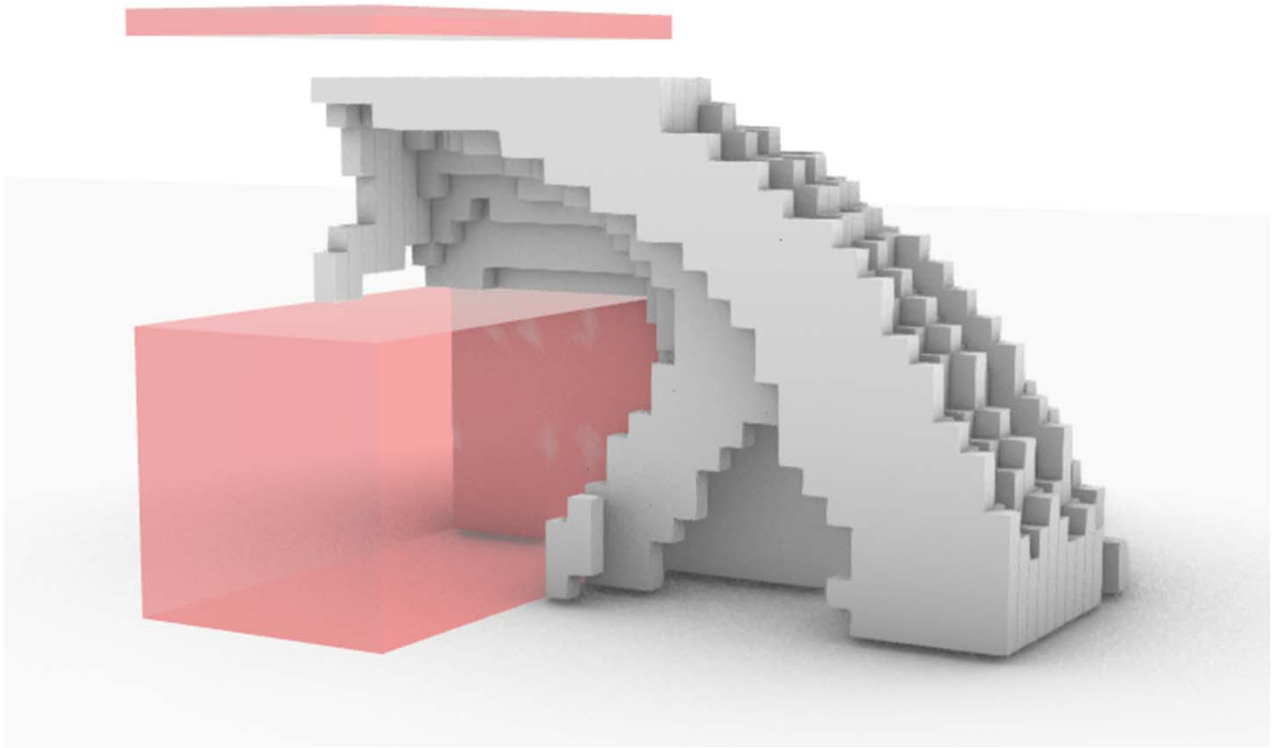




TOY5

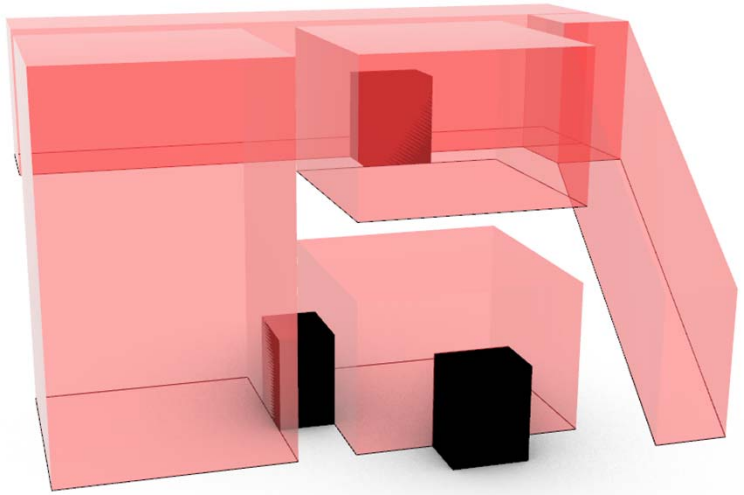
- Result





TOY6

- More complex geometry
- Implement density dependent forces
- Implement roof constraint

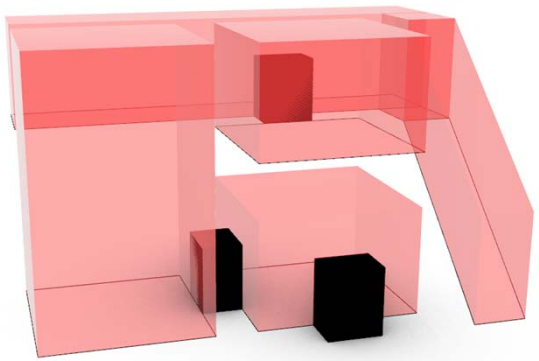


TOY6

- Implement density dependent forces
- Implement roof constraint
- More complex geometry

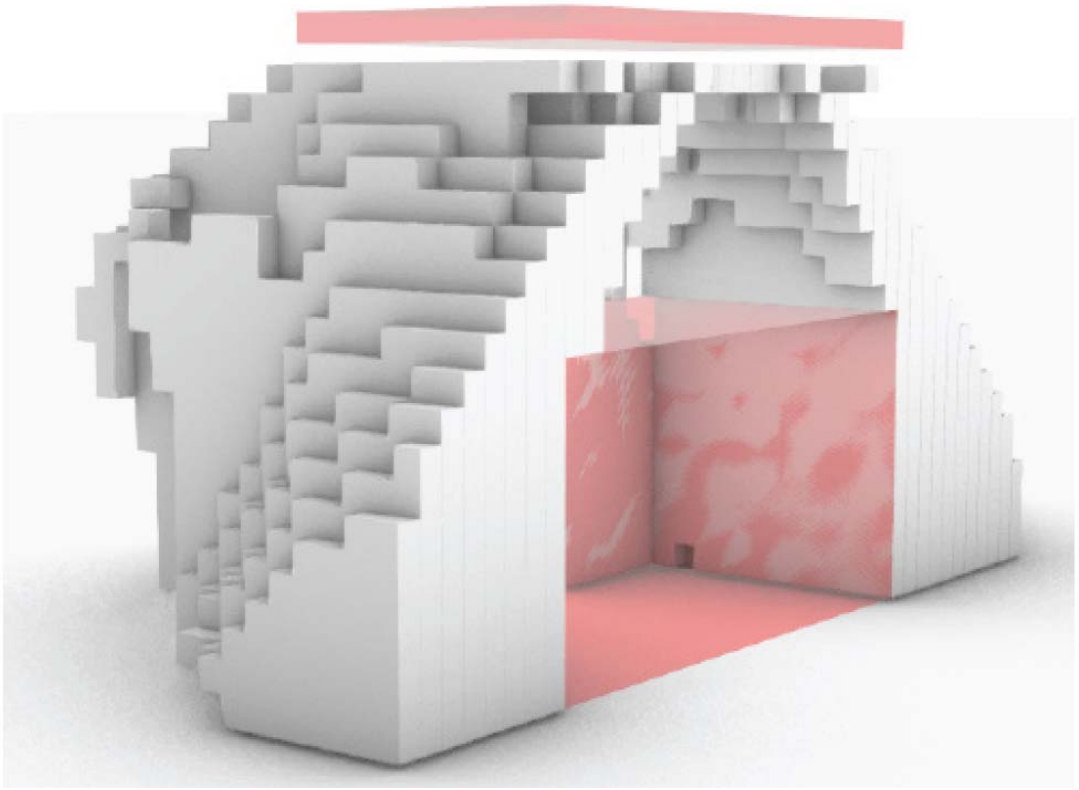
$$\frac{C_e}{x_e} = -px_e^{p-1} - U_e^T \frac{\delta Ke}{\delta x_e} U_e - 2U_e^T \frac{\delta Fe}{\delta x_e}$$

$$F_e = F_{preset} + F_{self} + F_{snow}$$



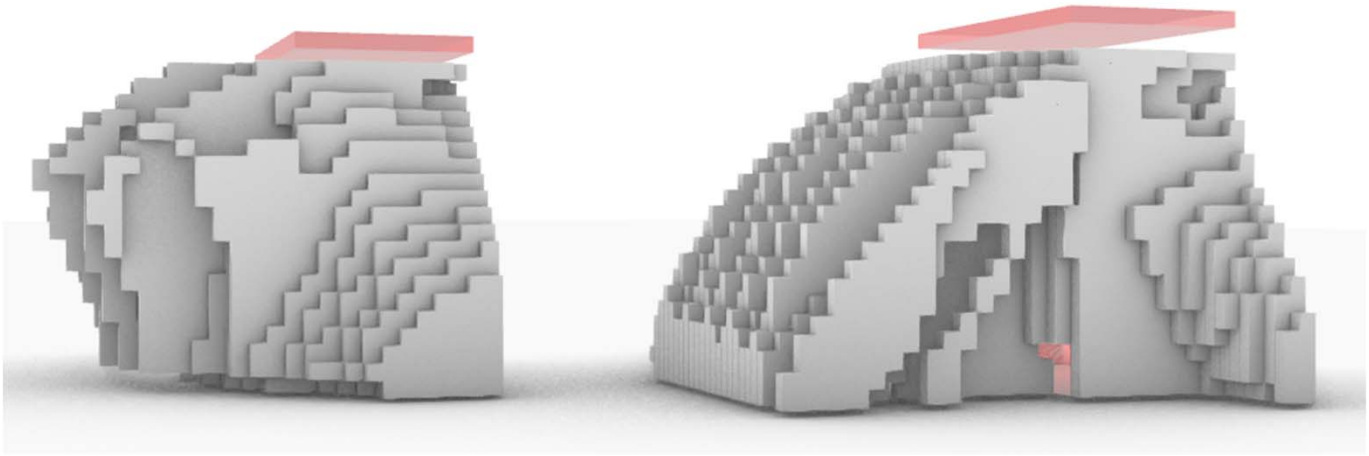
TOY6

- Implement density dependent forces
- Implement roof constraint
- More complex geometry



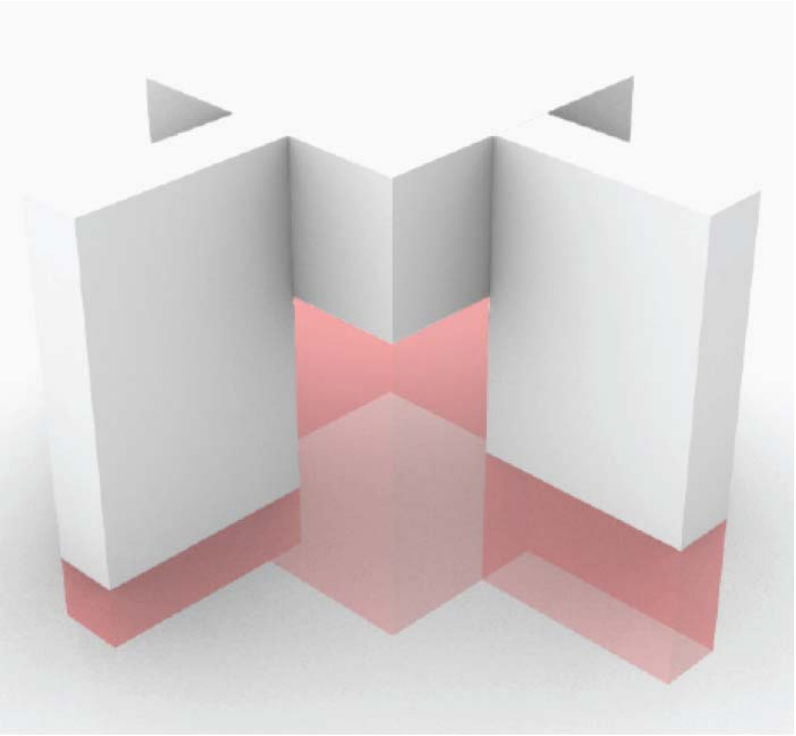
TOY6

- Implement density dependent forces
- Implement roof constraint
- More complex geometry



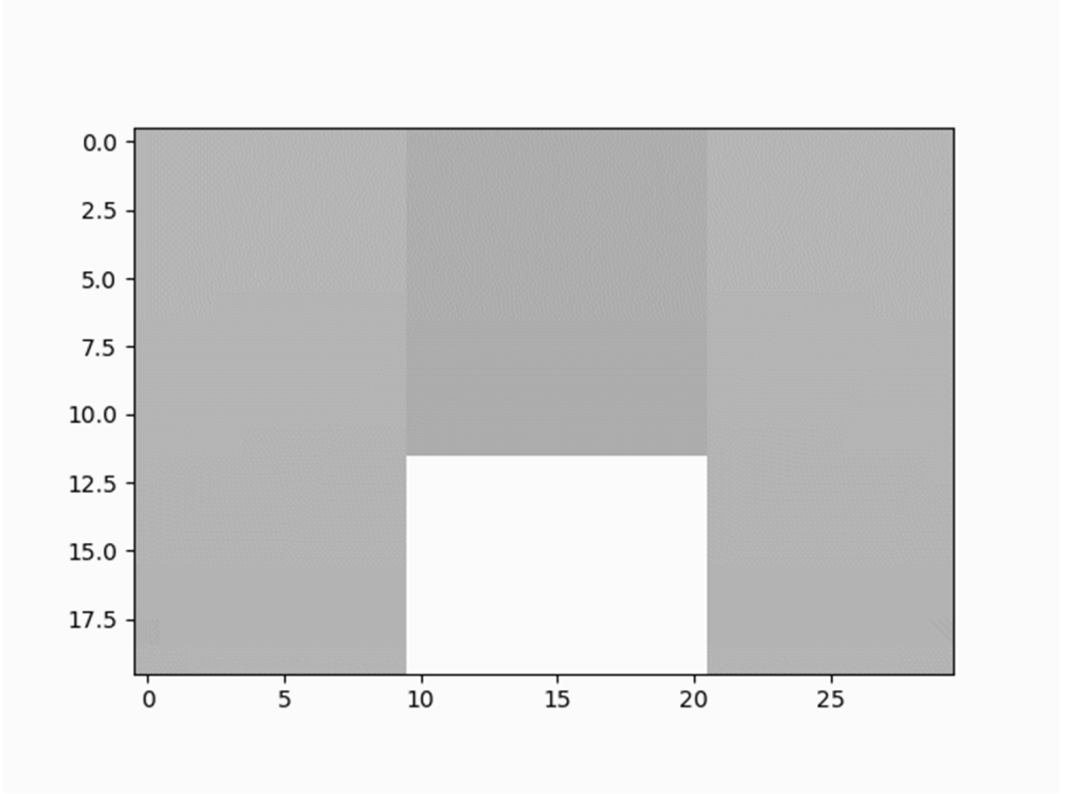
TOY6

- Implement density dependent forces
- Implement roof constraint
- More complex geometry



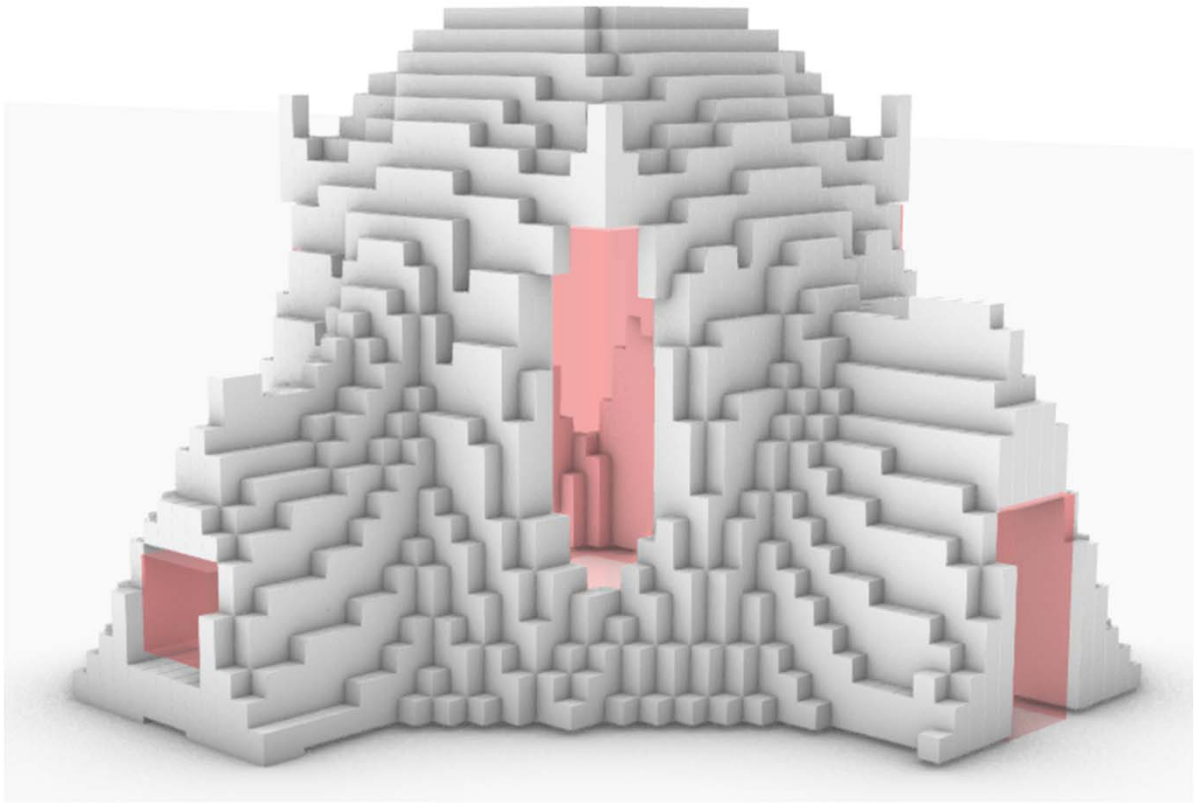
TOY6

- Implement density dependent forces
- Implement roof constraint
- More complex geometry



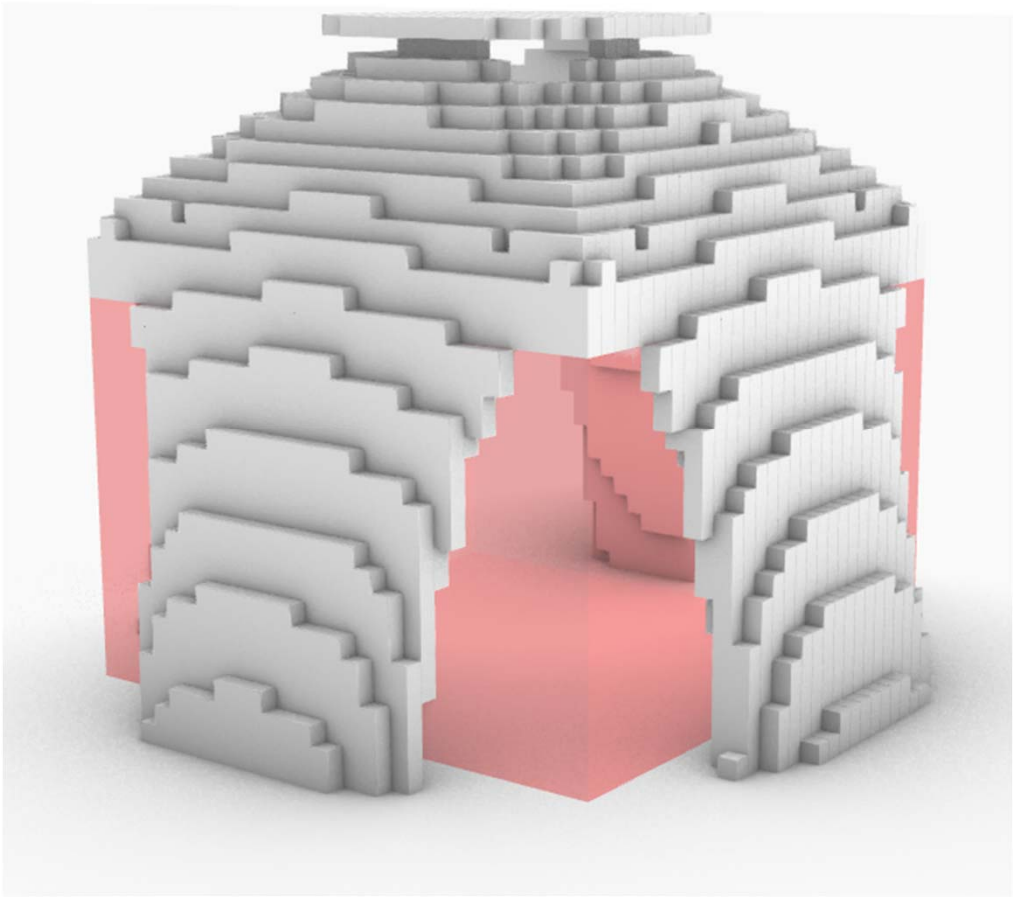
TOY6

- Implement density dependent forces
- Implement roof constraint
- More complex geometry



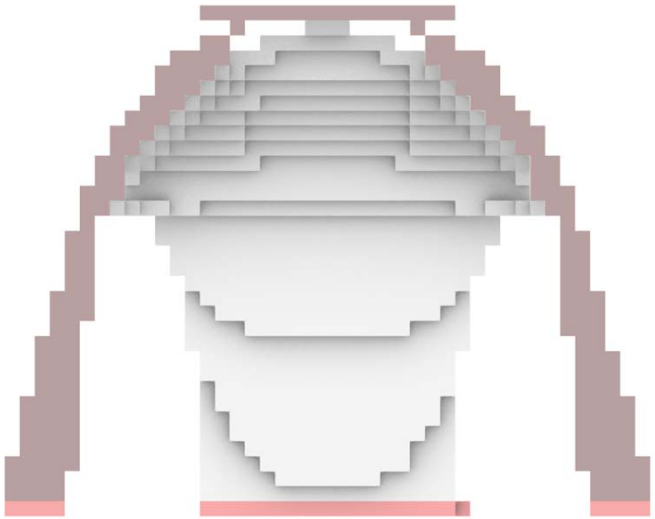
TOY6

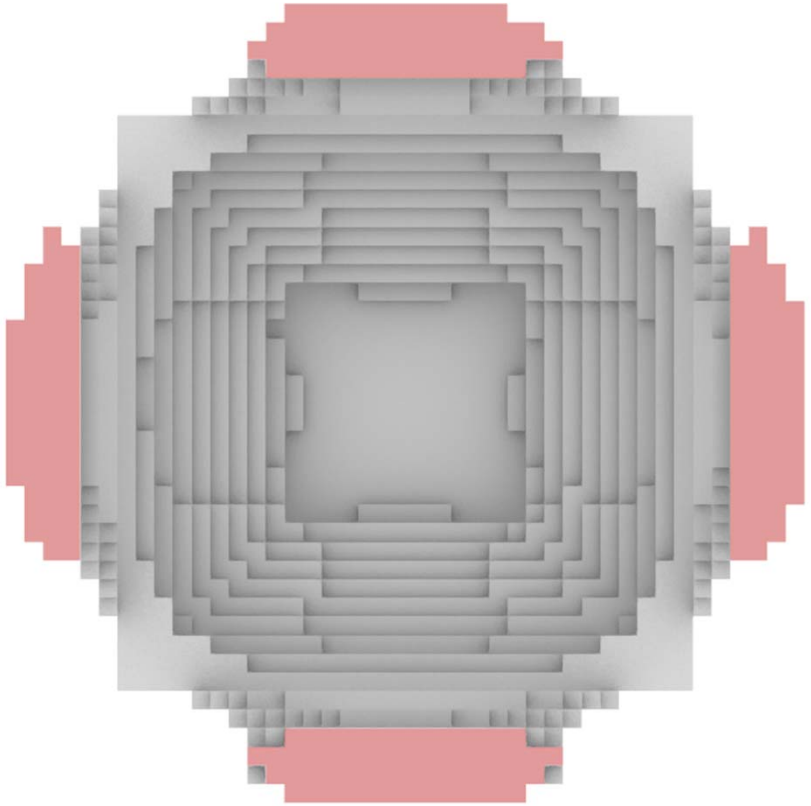
- Implement density dependent forces
- Implement roof constraint
- More complex geometry



TOY6

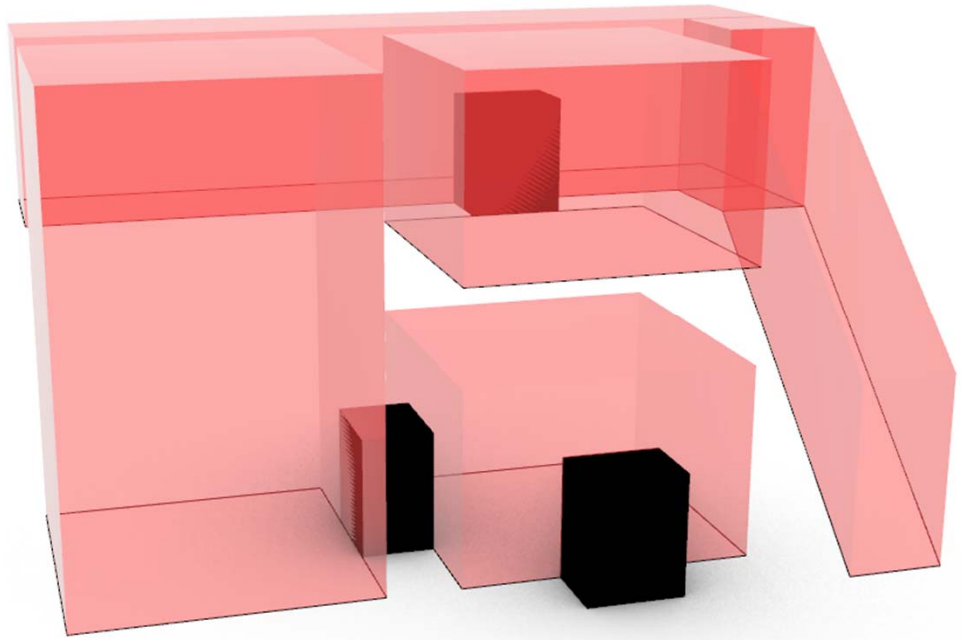
- Implement density dependent forces
- Implement roof constraint
- More complex geometry





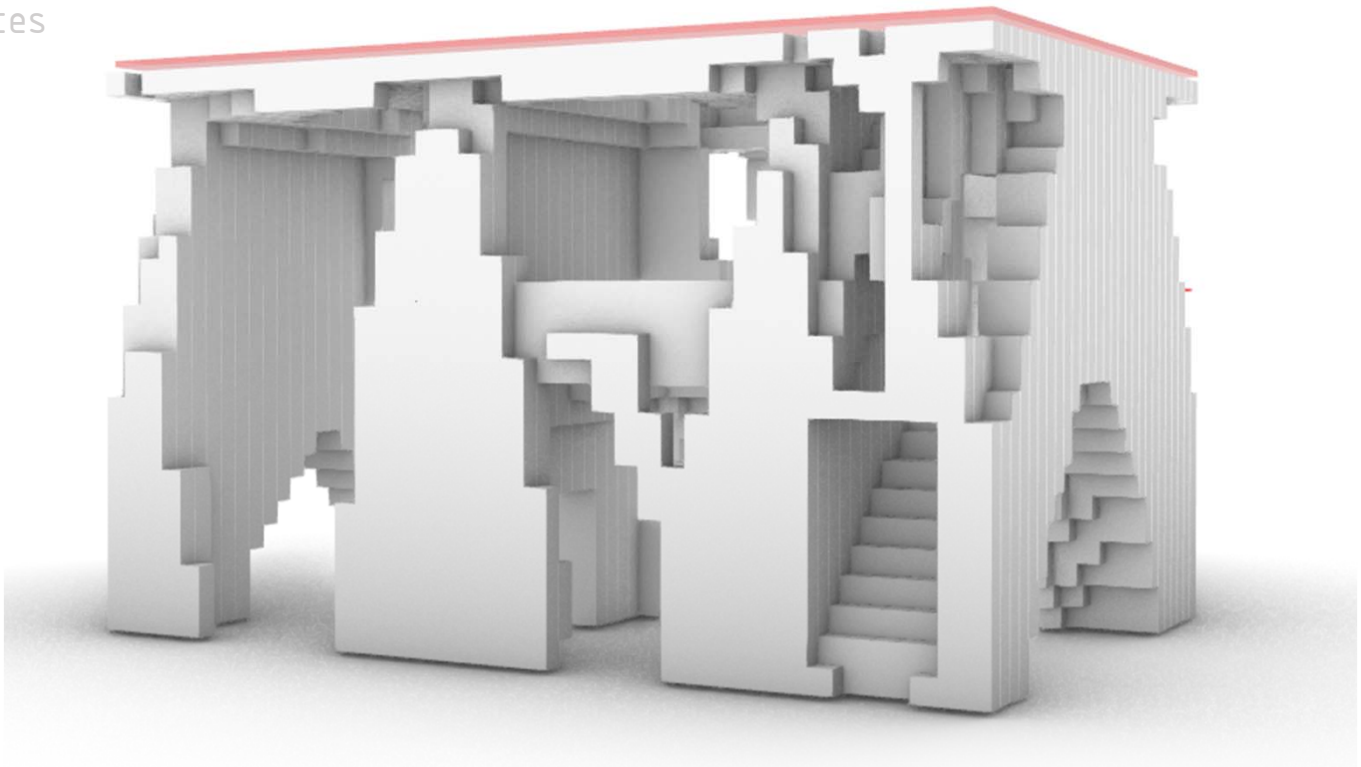
TOY6

- Implement density dependent forces
- Implement roof constraint
- More complex geometry



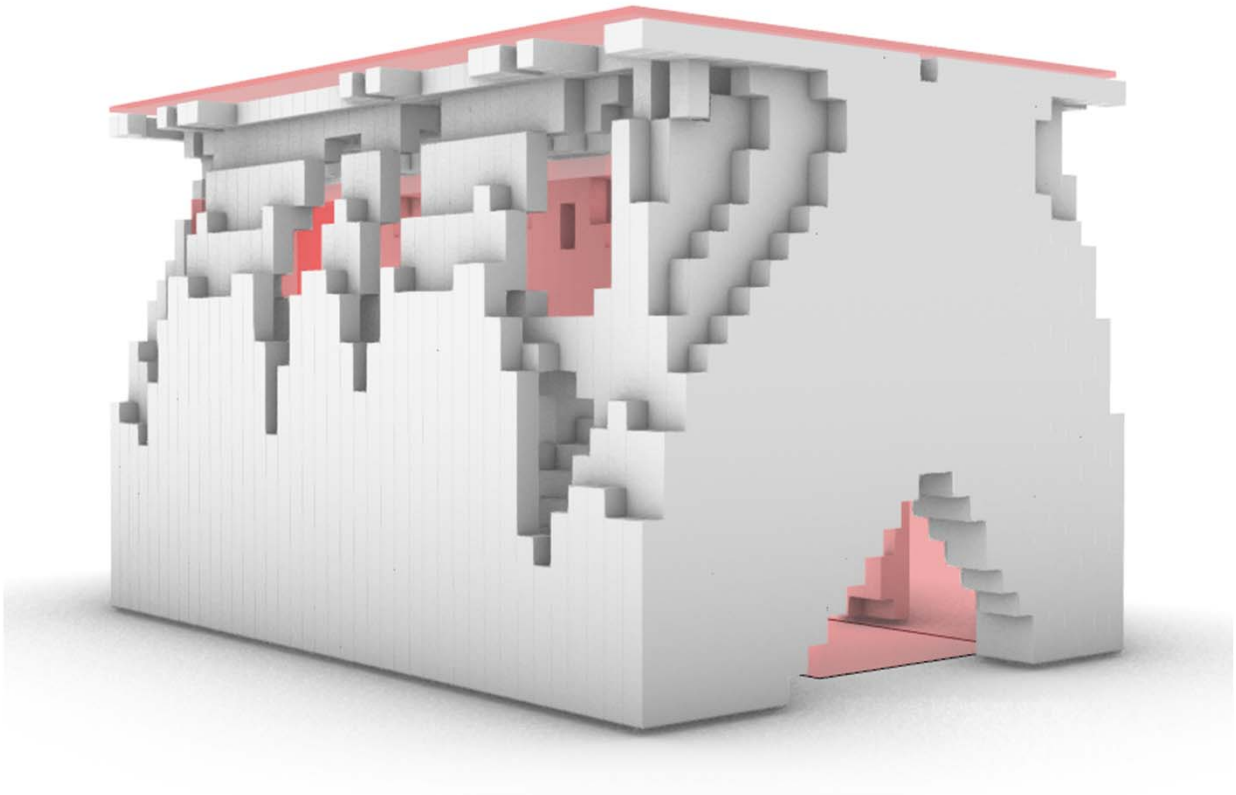
TOY6

- Implement density dependent forces
- Implement roof constraint
- More complex geometry



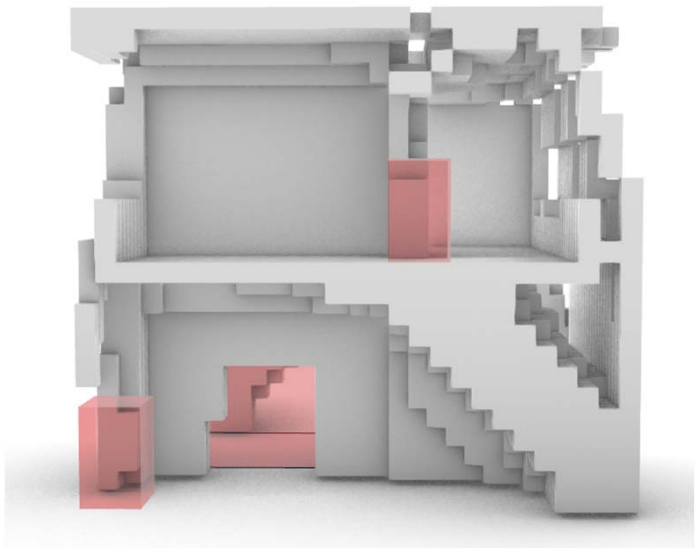
TOY6

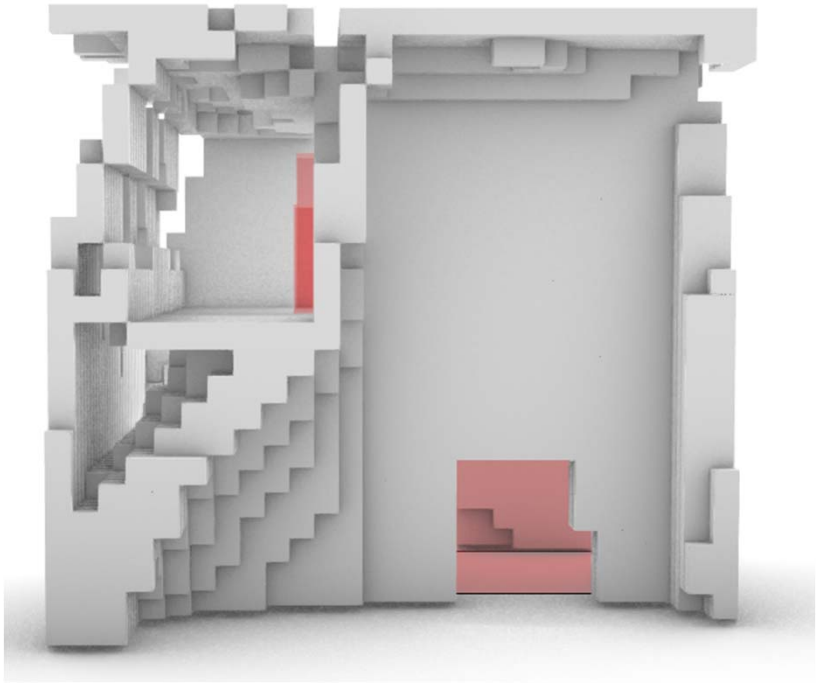
- Implement density dependent forces
- Implement roof constraint
- More complex geometry



TOY6

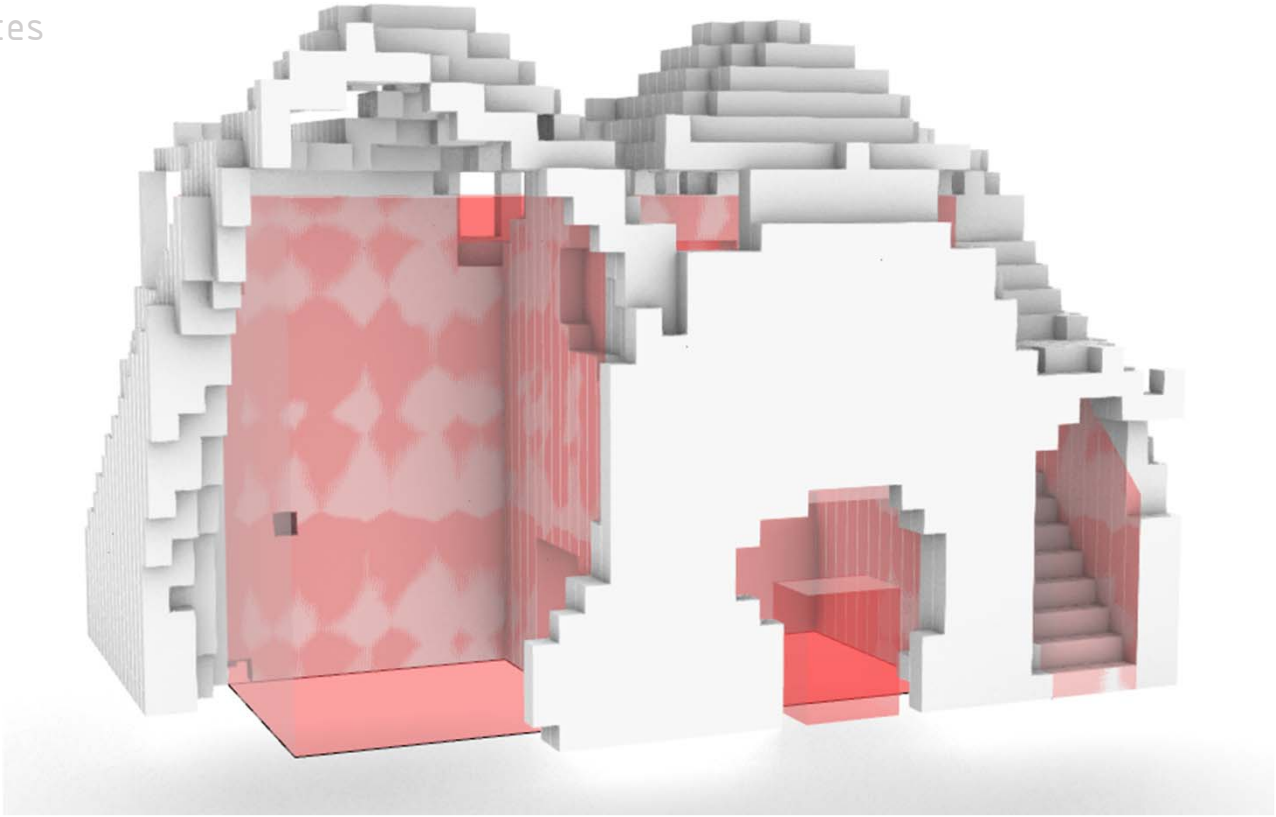
- Implement density dependent forces
- Implement roof constraint
- More complex geometry





TOY6

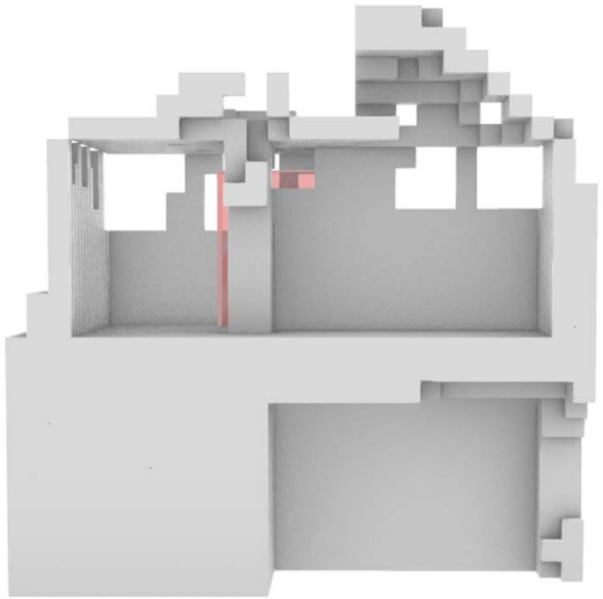
- Implement density dependent forces
- Implement roof constraint
- More complex geometry

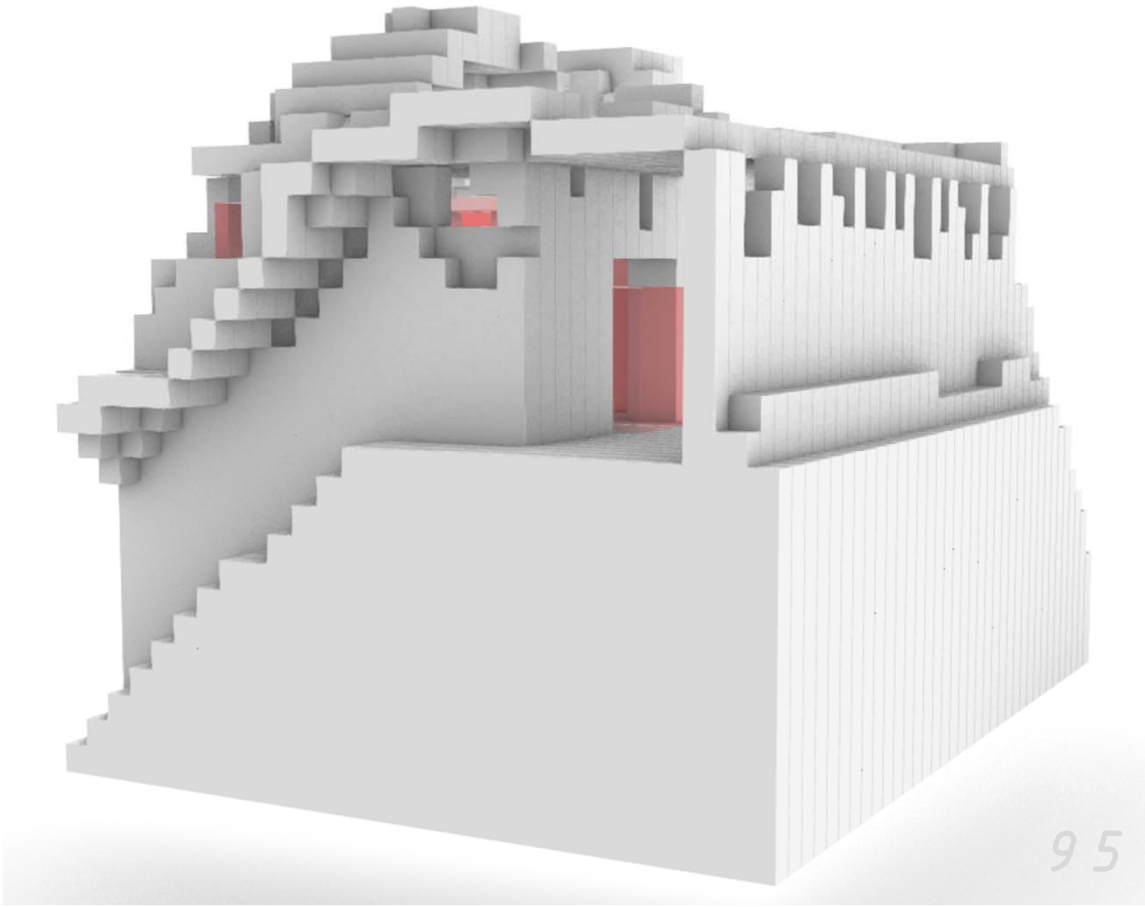


03 Toy problems

TOY6

- Implement density dependent forces
- Implement roof constraint
- More complex geometry

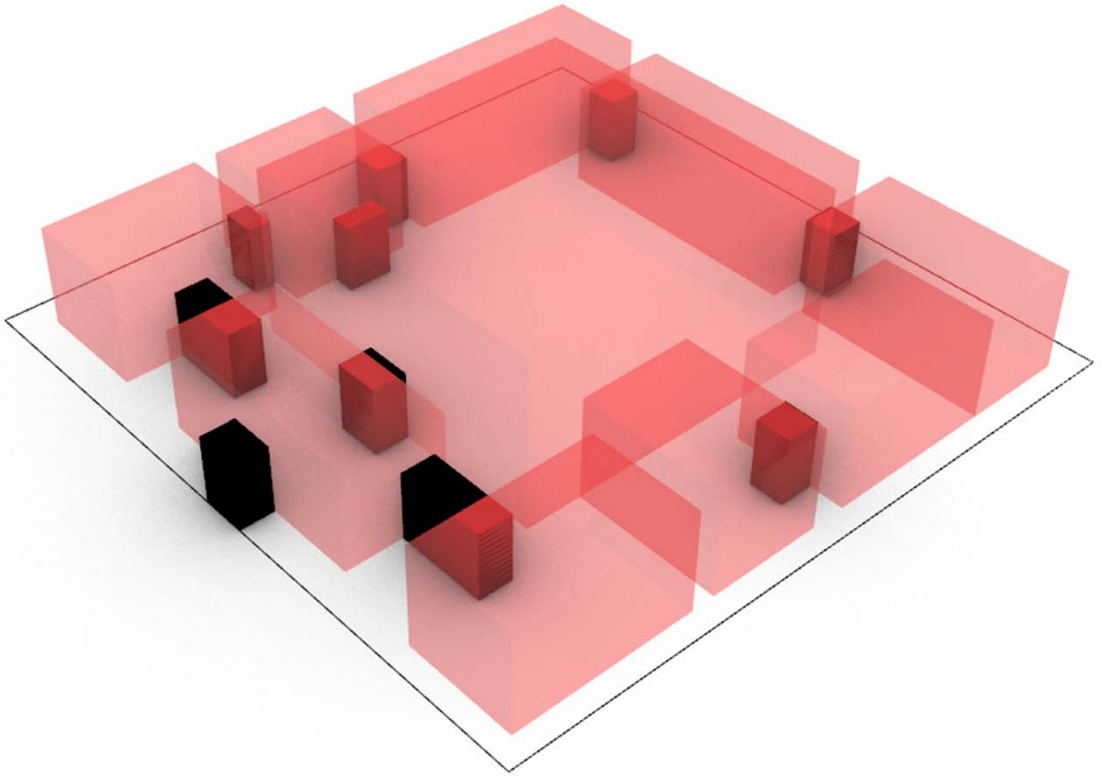




03 Toy problems

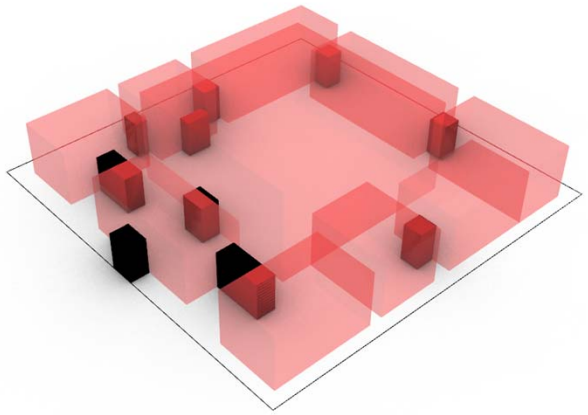
TOY6

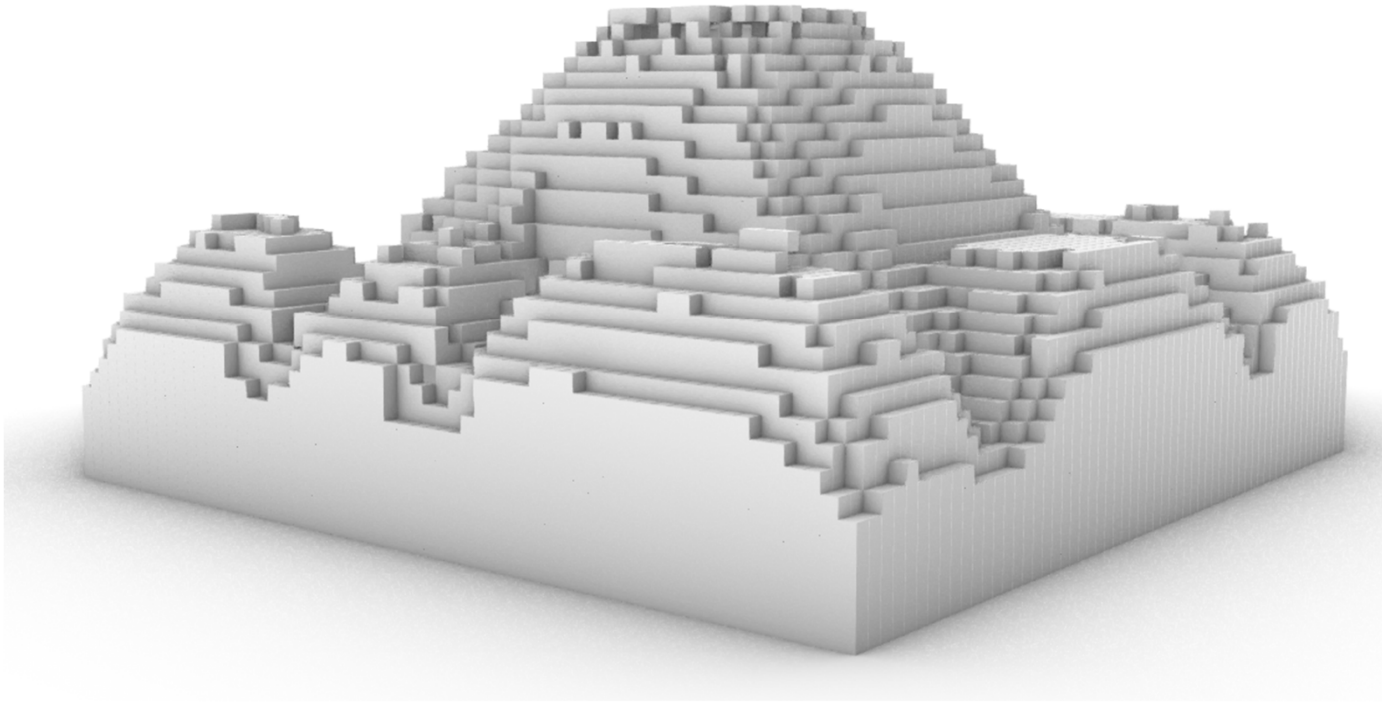
- Implement density dependent forces
- Implement roof constraint
- More complex geometry



TOY6

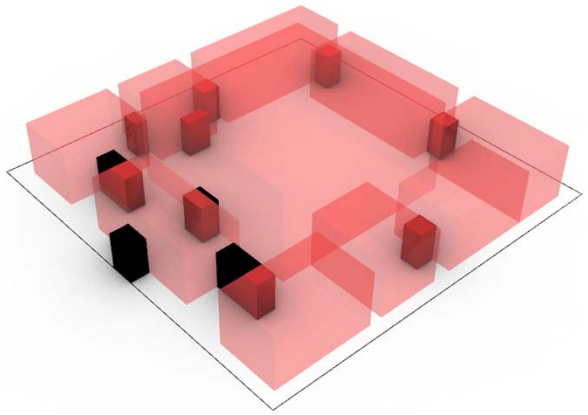
- Implement density dependent forces
- Implement roof constraint
- More complex geometry

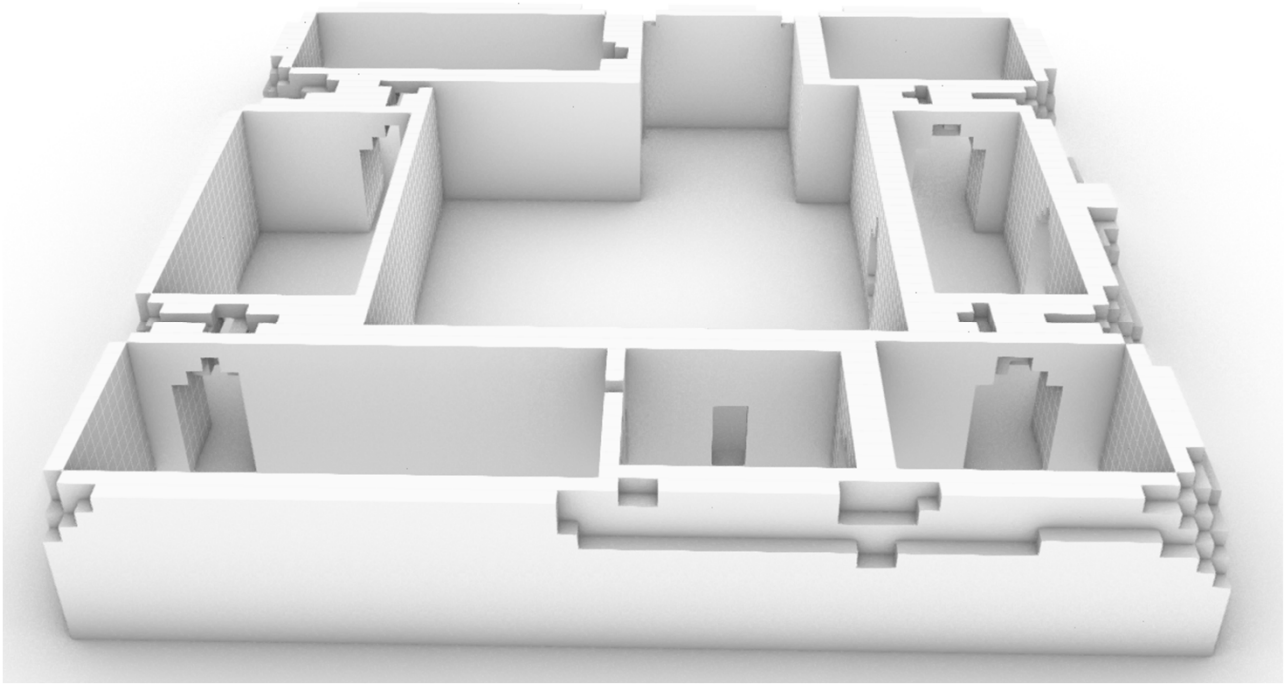




TOY6

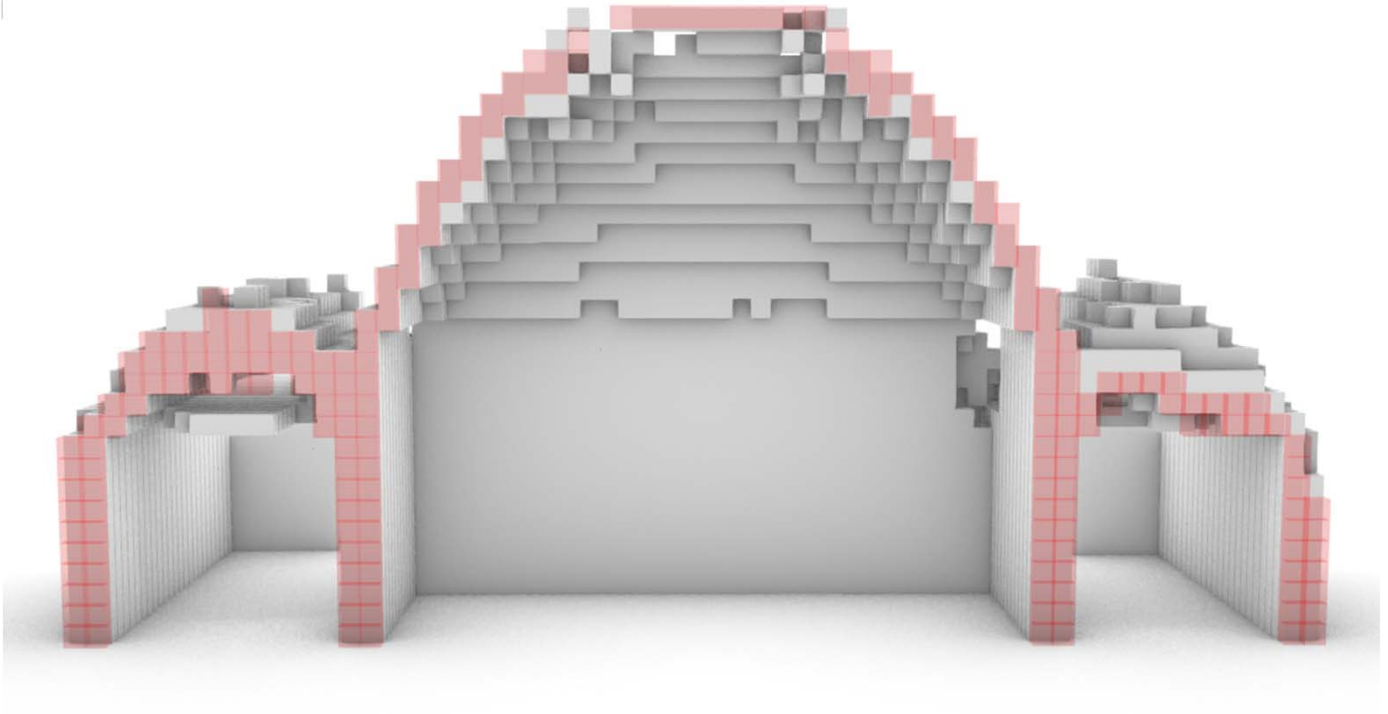
- Implement density dependent forces
- Implement roof constraint
- More complex geometry





TOY6

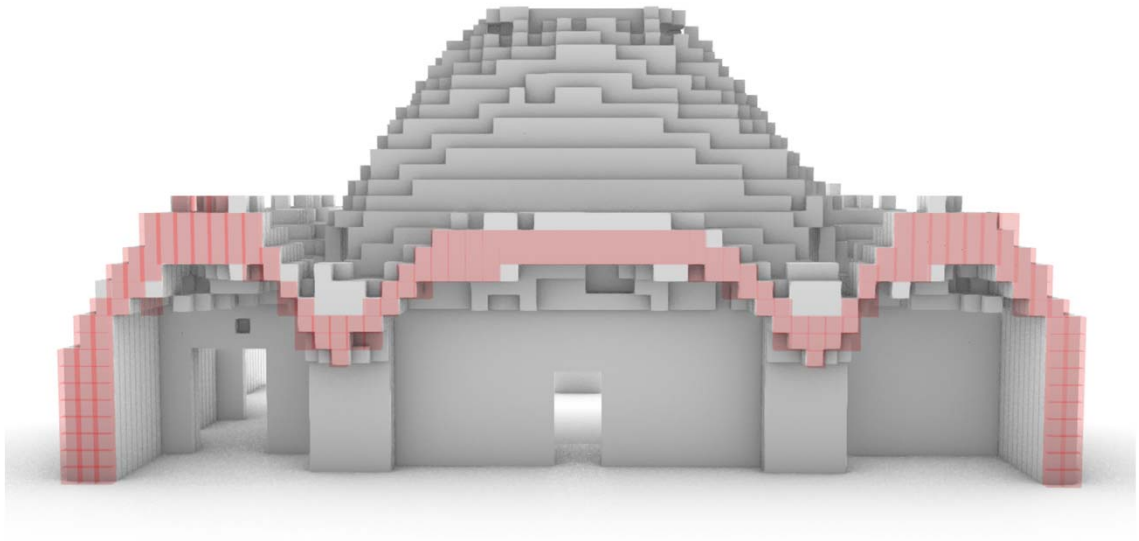
- Implement density dependent fo
- Implement roof constraint
- More complex geometry



03 Toy problems

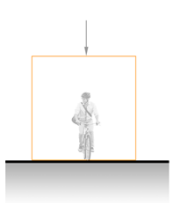
TOY6

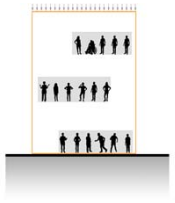
- Implement density dependent forces
- Implement roof constraint
- More complex geometry



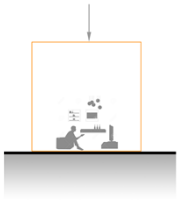
04 Conclusions

create a working topology optimization methodology and translate this in the form of an algorithm.

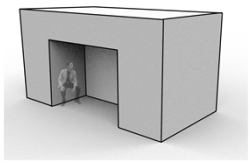


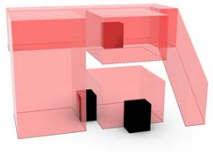


implement density dependent forces in the methodology and in the algorithm.



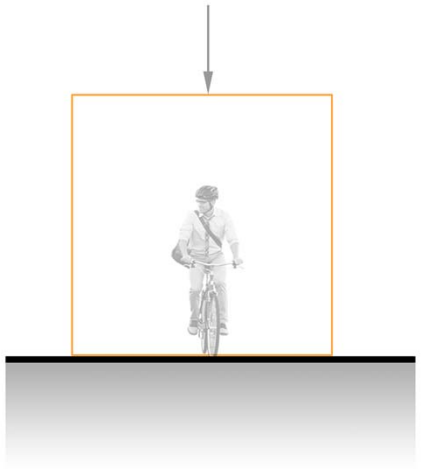
translate the methodology and the algorithm to 3D geometry.

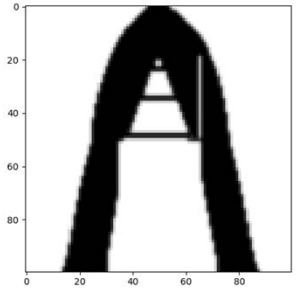


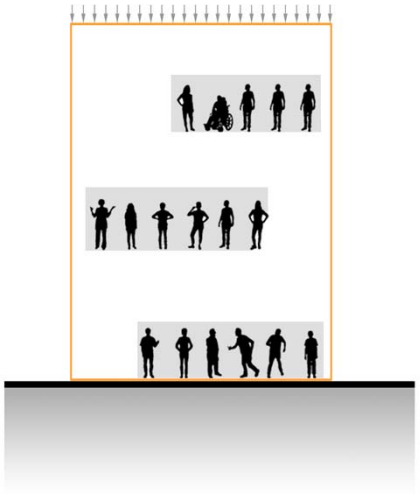


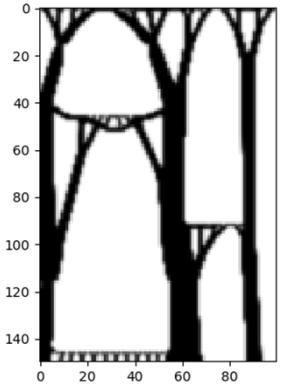
04 Conclusions

create a working topology optimization methodology and translate this in the form of an algorithm.



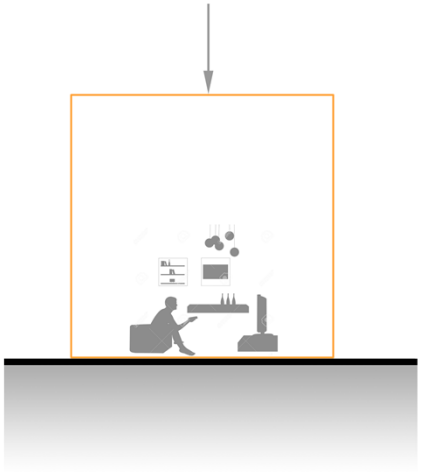


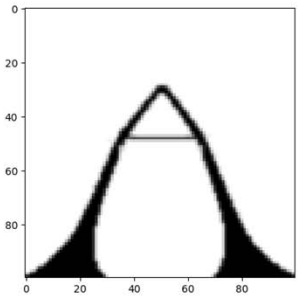


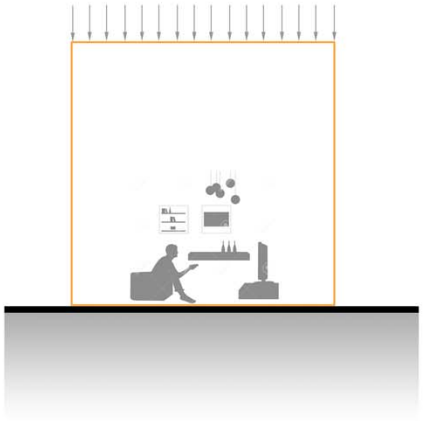


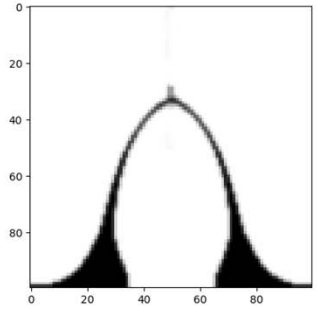
04 Conclusions

implement self-weight in the methodology and in the algorithm.



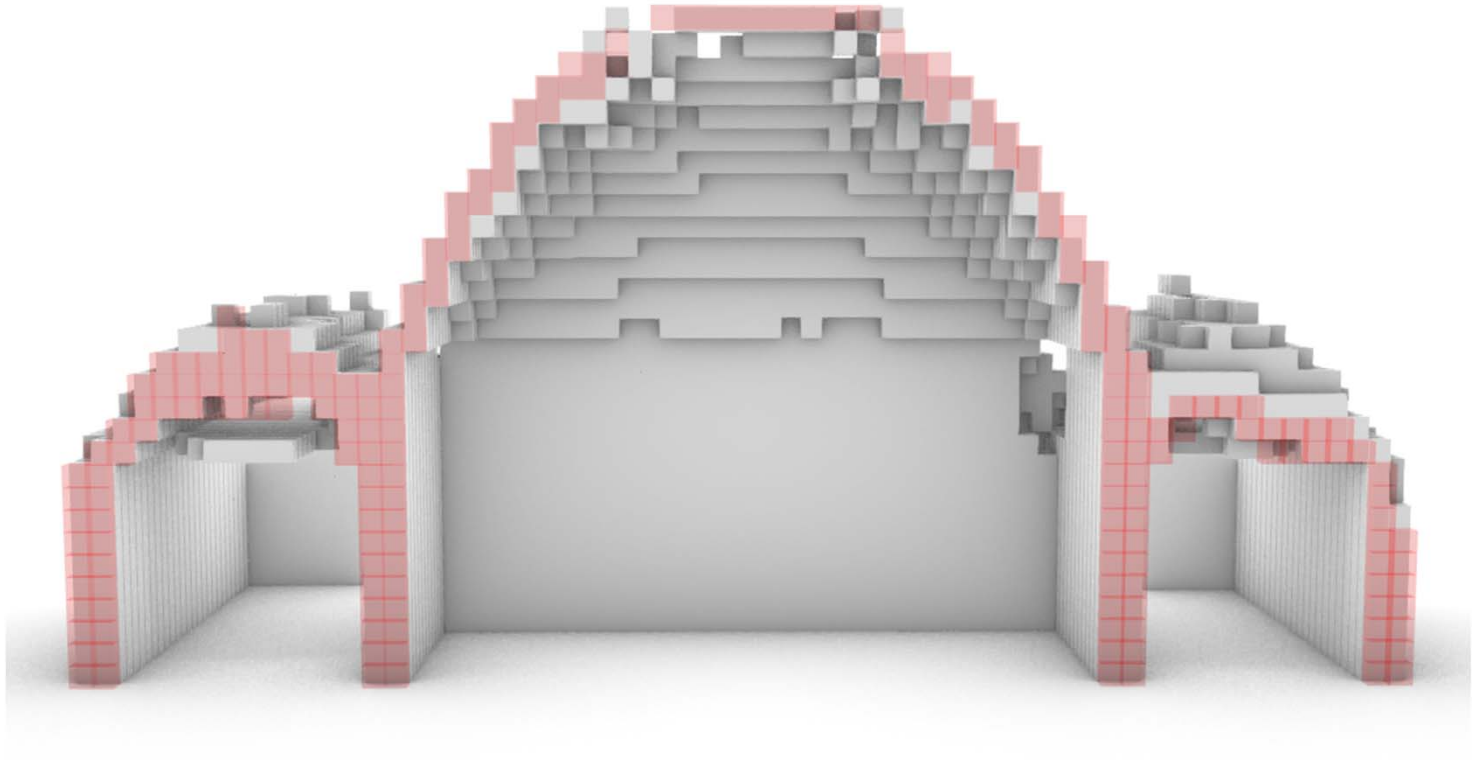




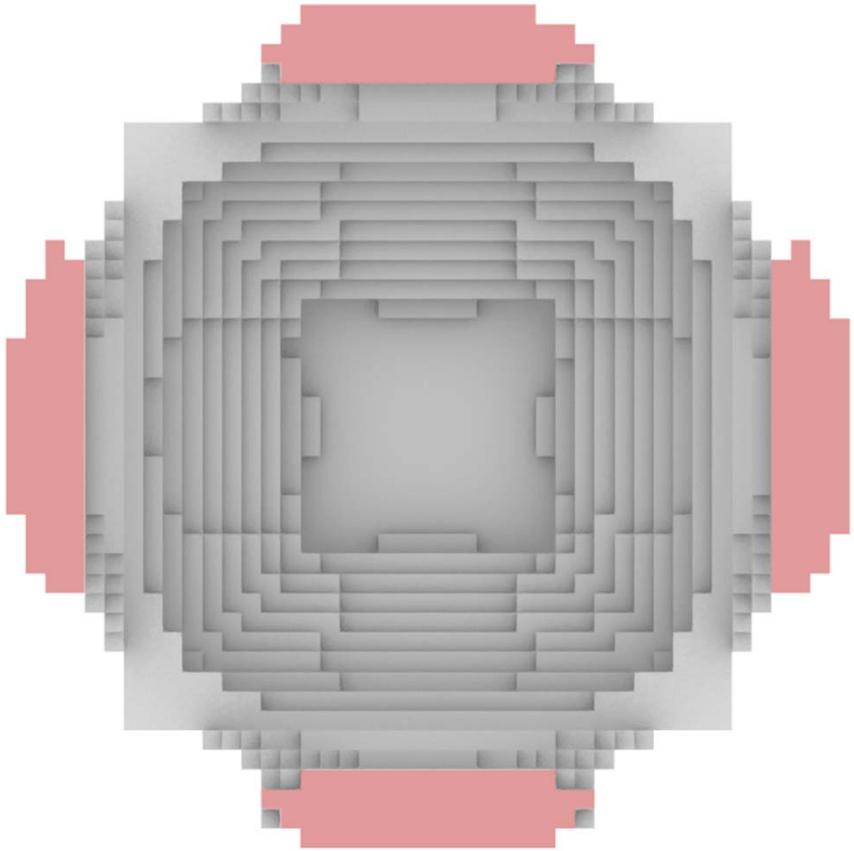


04 Conclusions

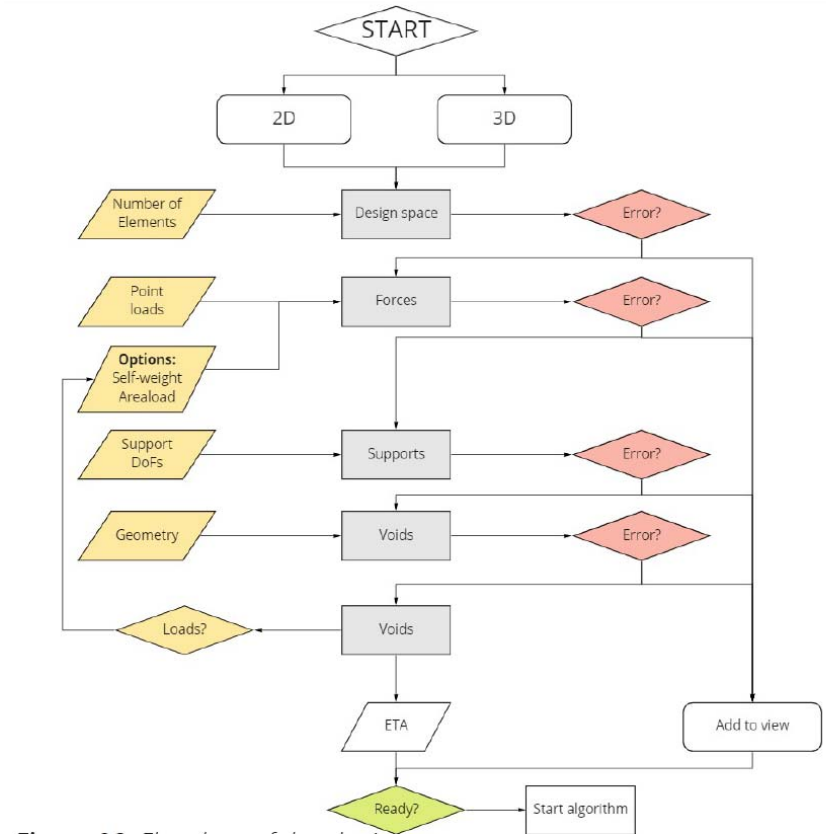
translate the methodology and the algorithm to 3D geometry.

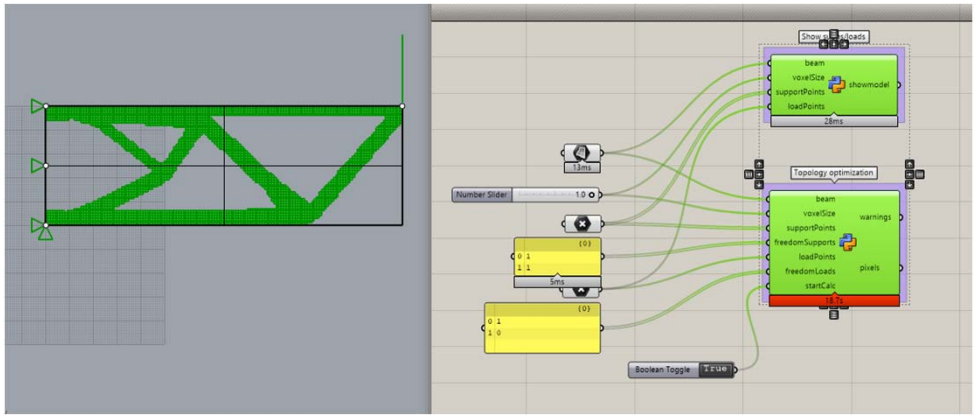


04 Conclusions

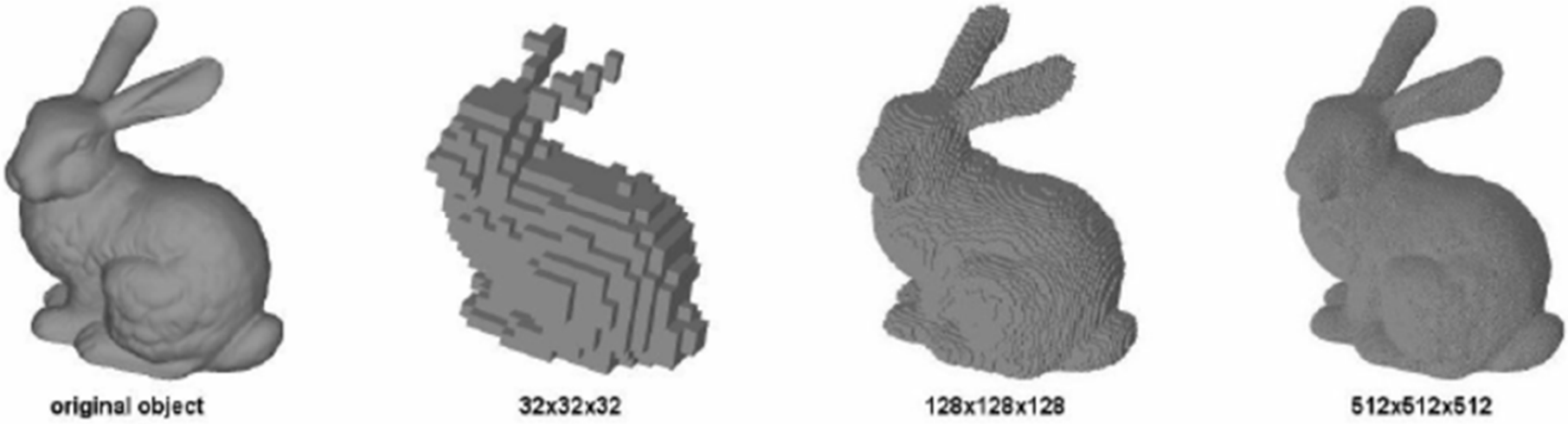


04 Conclusions





(Kakadiaris, 2007)



Thank you