BESO Topology Optimization

Albert Li

Albert Li

www.albertlidesign.com

天津大学 环境艺术设计 本 -> RMIT 土木工程 博

一个热爱艺术设计和图形学的土木工程博士生。

What is Topology Optimization?

Input:

A given design domain

Loads

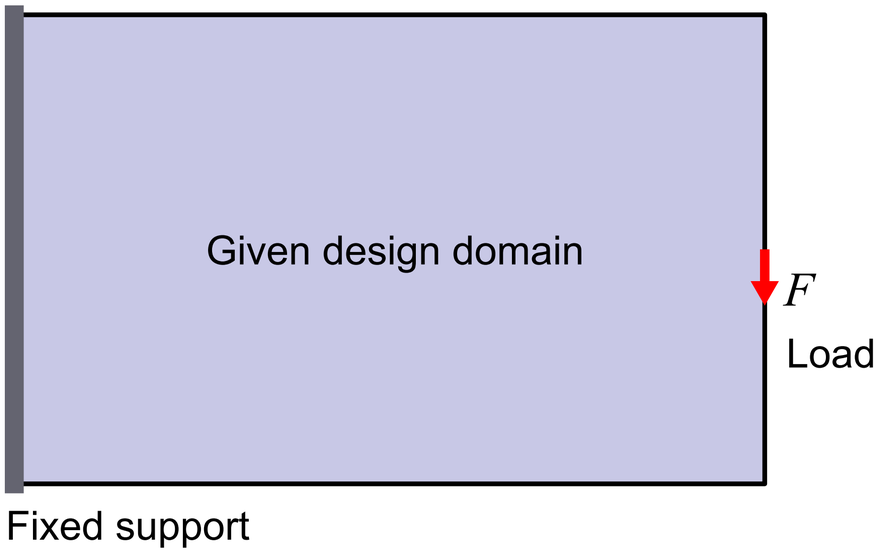
Boundary conditions

Constraints

Output:

An optimal design

(e.g. minimizing compliance)



What is BESO?

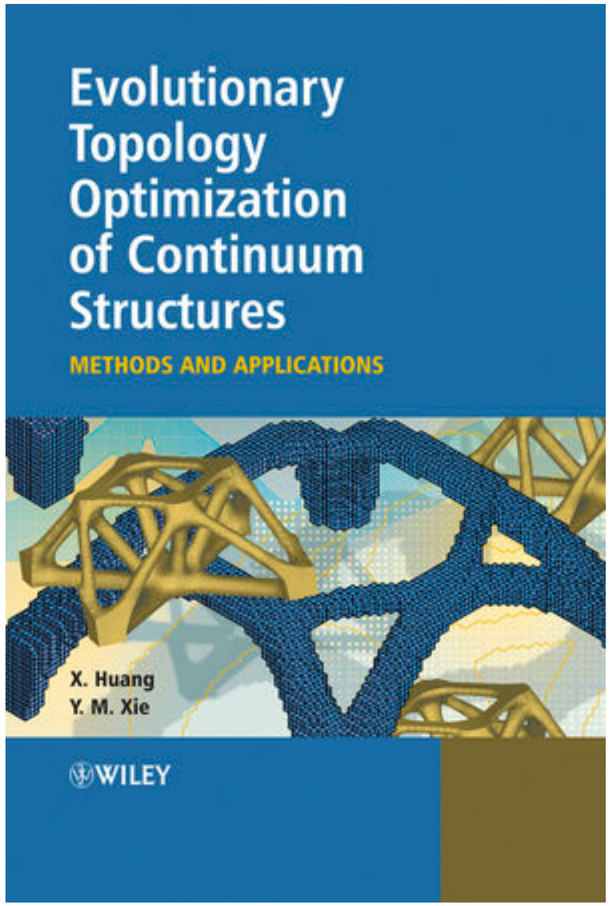
BESO (Bi-directional Evolutionary Structural Optimization) can remove inefficient material and add material to needed areas, simultaneously.

$$Minimize: \quad C = \frac{1}{2}\mathbf{U}^T\mathbf{K}\mathbf{U}$$

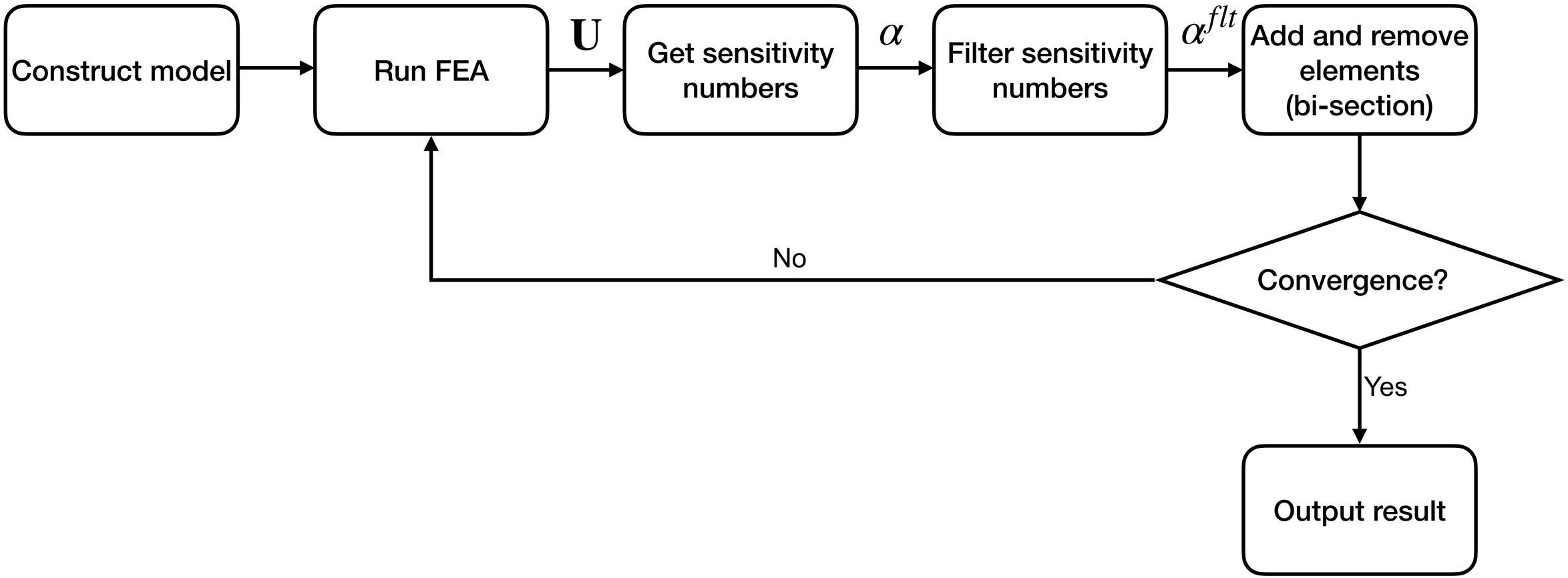
$$Subject\ to: \quad \mathbf{V}^* - \sum_{i=1}^{N} \mathbf{V}_i x_i = 0$$

$$x_i = x_{min}\ or\ 1$$

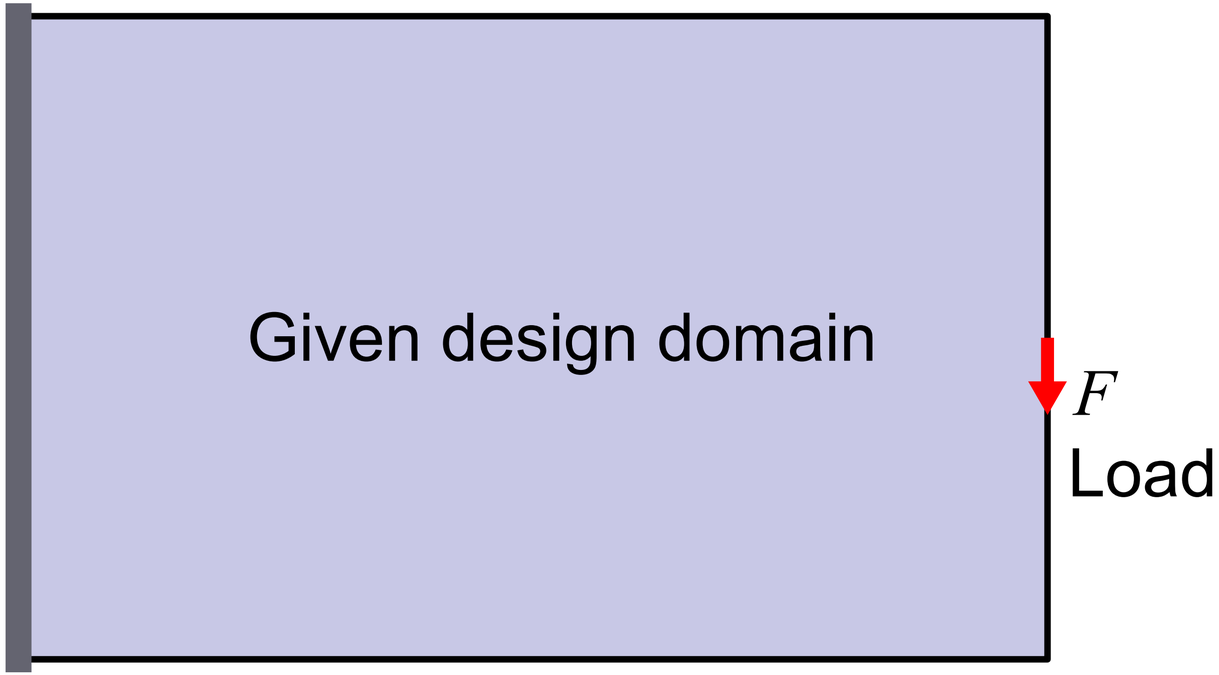
C	Compliance	V^*	Target volume
$\mathbf{U}$	Displacement	V_i	i th elemental volume
$\mathbf{K}$	Global stiffness matrix	x_i	i th design variable



Workflow


Construct model
Run FEA
U
Get sensitivity numbers
α
Filter sensitivity numbers
α^{flt}
Add and remove elements (bi-section)
Convergence?
No
Yes
Output result

Construct model




```
# Model parameters
nely = 50
nelx = 80
n_node = (nelx+1) * (nely+1)
ndof = 2 * n_node
```

```
# Define boundary conditions
fixed_dofs = list(range(0, 2 * (nely + 1)))# fixed dof
all_dofs = list(range(0, 2 * (nelx + 1) * (nely + 1)))
free_dofs = np.array(list(set(all_dofs) - set(fixed_dofs)))# free dof
n_free_dof = len(free_dofs)
```

```
# Define load vector
F[2*(nelx+1)*(nely+1)-nely-1] = -1.
```

Run FEA

1. Define K, F, U, Ke

```
# FEM variables
K = ti.field(ti.f32, shape=(ndof, ndof))
F = ti.field(ti.f32, shape=ndof)
U = ti.field(ti.f32, shape=ndof)
Ke = ti.field(ti.f32, shape=(8, 8))
free_dofs_vec = ti.field(ti.i32, shape=n_free_dof)
F_freedof = ti.field(dtype=ti.f32, shape=n_free_dof)
U_freedof = ti.field(dtype=ti.f32, shape=n_free_dof)
```

2. Construct elemental stiffness matrix

```
# Get elemental stiffness matrix
def get_ke():
    k = np.array(
        [1 / 2 - nu / 6, 1 / 8 + nu / 8, -1 / 4 - nu / 12, -1 / 8 + 3 * nu / 8, -1 / 4 + nu / 12, -1 / 8 - nu / 8,
         nu / 6, 1 / 8 - 3 * nu / 8])

    Ke_ = E / (1. - nu ** 2) * np.array([[k[0], k[1], k[2], k[3], k[4], k[5], k[6], k[7]],
                                         [k[1], k[0], k[7], k[6], k[5], k[4], k[3], k[2]],
                                         [k[2], k[7], k[0], k[5], k[6], k[3], k[4], k[1]],
                                         [k[3], k[6], k[5], k[0], k[7], k[2], k[1], k[4]],
                                         [k[4], k[5], k[6], k[7], k[0], k[1], k[2], k[3]],
                                         [k[5], k[4], k[3], k[2], k[1], k[0], k[7], k[6]],
                                         [k[6], k[3], k[4], k[1], k[2], k[7], k[0], k[5]],
                                         [k[7], k[2], k[1], k[4], k[3], k[6], k[5], k[0]]])

    Ke.from_numpy(Ke_)
```

3. Assemble global stiffness matrix

```
@ti.kernel
def assemble_k():
    for I in ti.grouped(K):
        K[I] = 0.

    # 1. Assemble Stiffness Matrix
    for ely, elx in ti.ndrange(nely, nelx):
        n1 = (nely + 1) * elx + ely + 1
        n2 = (nely + 1) * (elx + 1) + ely + 1
        edof = ti.Vector([2*n1 -2, 2*n1 -1, 2*n2 -2, 2*n2 -1, 2*n2, 2*n2+1, 2*n1, 2*n1+1])

        for i, j in ti.static(ti.ndrange(8, 8)):
            K[edof[i], edof[j]] += xe[ely, elx] ** penalty * Ke[i, j]


@ti.kernel
def assemble_k_fd(A: ti.sparse_matrix_builder()):
    # 2. Get K_freedof
    for i, j in ti.ndrange(n_free_dof,n_free_dof):
        A[i, j] += K[free_dofs_vec[i], free_dofs_vec[j]]
```

4. Solve $\mathbf{KU} = \mathbf{F}$

```
# run FEA
assemble_k()
K_freedof = ti.linalg.SparseMatrixBuilder(n_free_dof, n_free_dof, max_num_triplets=10000000)
assemble_k_fd(K_freedof)
A = K_freedof.build()
solver = ti.linalg.SparseSolver(solver_type="LLT")
solver.analyze_pattern(A)
solver.factorize(A)
result = solver.solve(F_freedof)
U_freedof.from_numpy(result)
isSuccess = solver.info()
print(f'isSuccess: {isSuccess}')
print("-------------------------")
backward_map_u()
```

Get sensitivity numbers

```
# BESO parameters
E = 1. # Young modulus
nu = 0.3 # Possion's rate
rmin = 4 # Filter radius
volfrac = 0.5 # Volume fraction
ert = 0.02 # Evolutionary rate
penalty = 3 # Penalty
xmin = 1e-3 # minimal design variable

# BESO variables
xe = ti.field(ti.f32, shape=(nely, nelx))
dc = ti.field(ti.f32, shape=(nely, nelx))  # derivative of compliance
compliance = ti.field(ti.f32, shape=()) # compliance
dc_old = ti.field(ti.f32, shape=(nely, nelx))  # derivative of compliance
```

$$\alpha = -\frac{1}{p}\frac{\partial C}{\partial x_i} = \begin{cases} \frac{1}{2}\mathbf{u}_i^T\mathbf{K}_i^0\mathbf{u}_i & when\ x_i = 1 \\ \frac{x_{min}^{p-1}}{2}\mathbf{u}_i^T\mathbf{K}_i^0\mathbf{u}_i & when\ x_i = x_{min} \end{cases}$$

```
dc = ti.field(ti.f32, shape=(nely, nelx))  # derivative of compliance
```

```
@ti.kernel
def get_dc():
    for ely, elx in ti.ndrange(nely, nelx):
        n1 = (nely + 1) * elx + ely + 1
        n2 = (nely + 1) * (elx + 1) + ely + 1
        Ue = ti.Vector([U[2*n1 -2], U[2*n1 -1], U[2*n2-2], U[2*n2-1], U[2*n2], U[2*n2+1], U[2*n1], U[2*n1+1]])

        t = ti.Vector([0.,0.,0.,0.,0.,0.,0.,0.])
        for i in ti.static(range(8)):
            for j in ti.static(range(8)):
                t[i] += Ke[i, j] * Ue[j]
        d = 0.
        for i in ti.static(range(8)):
            d += 0.5 * Ue[i] * t[i] # d = Ue' * Ke * Ue

        compliance[None] += xe[ely, elx] ** penalty * d

        dc[ely, elx] = xe[ely, elx] ** (penalty - 1) * d
```

Filter sensitivity numbers

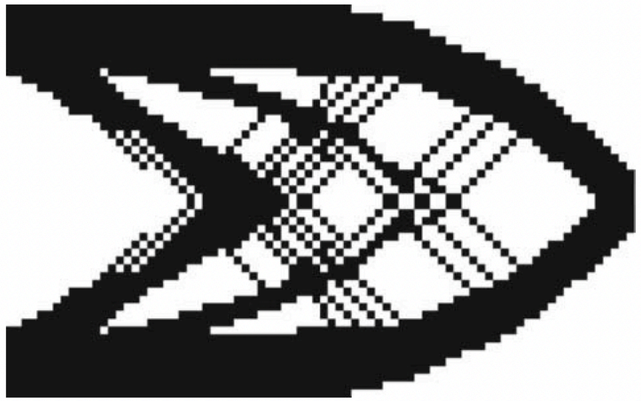
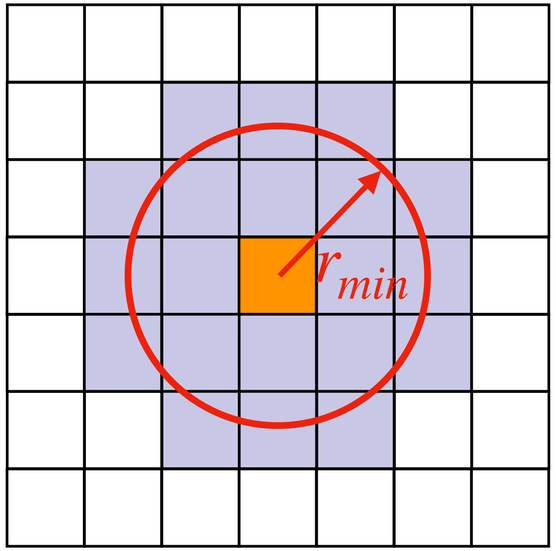


Figure 4.1 ‘Optimal’ design from the soft-kill BESO method without a filter.




```
def filt(dc):
    rminf = math.floor(rmin)
    dcf = np.zeros((nely, nelx))

    for i in range(nelx):
        for j in range(nely):
            sum_ = 0.
            for k in range(max(i - rminf, 0), min(i + rminf + 1, nelx)):
                for l in range(max(j - rminf, 0), min(j + rminf + 1, nely)):
                    fac = rmin - math.sqrt((i - k) ** 2. + (j - l) ** 2.)
                    sum_ += max(0., fac)
                    dcf[j, i] = dcf[j, i] + max(0., fac) * dc[l, k]
            dcf[j, i] = dcf[j, i] / sum_
    return dcf
```

$$\alpha_p^{flt} = \frac{\sum_{q=1}^{N_n} (r_{min} - r_{pq})\alpha_q}{\sum_{q=1}^{N_n} (r_{min} - r_{pq})}$$

Add and remove elements (bi-section)

```
def beso(crtvol):
    dc_np = dc.to_numpy()
    l1 = dc_np.min()
    l2 = dc_np.max()
    tarvol = crtvol * nely * nelx
    x = xe.to_numpy()
    while l2 - l1 > 1e-5:
        lmid = (l2 + l1) / 2.
        for ely in range(0, nely):
            for elx in range(0,nelx):
                x[ely, elx] = 1 if dc_np[ely, elx] > lmid else xmin
        if (sum(sum(x)) - tarvol) > 0:
            l1 = lmid
        else:
            l2 = lmid
    return x
```

Convergence

$$\Delta = \frac{\left| \sum_{i=1}^{N} C_{k-i+1} - \sum_{i=1}^{N} C_{k-N-i+1} \right|}{\sum_{i=1}^{N} C_{k-i+1}}$$

```
while change > 1e-3:
```

```
# check convergence
if iter > 10:
    change = abs((sum(history_C[iter - 5:iter]) - sum(history_C[iter - 10:iter - 5])) / sum(history_C[iter - 5:iter]))
```

BESO

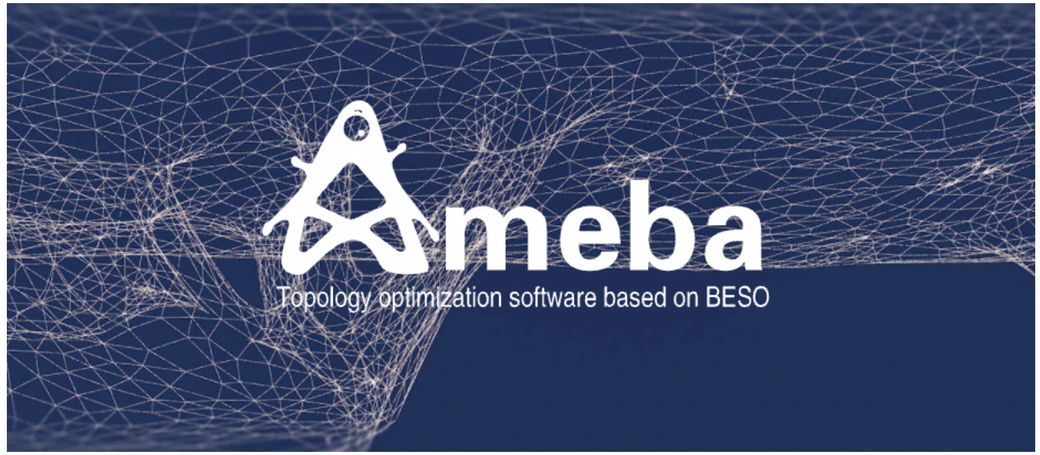
Code

我和助教李喆昊同学致力于基于taichi开发一个高效、易用、轻量的GPU并行拓扑优化框架。

期待大牛与我们一起开发。

https://github.com/Ricahrd-Li/taichi-TopOpt

Applications



基于Rhinoceros的拓扑优化软件Ameba

拓扑优化椅——Jue Chair

References

1. https://github.com/ToddyXuTao/BESO-for-2D
2. https://www.cism.org.au/tools
3. Zuo, Z.H. and Xie, Y.M., 2015. A simple and compact Python code for complex 3D topology optimization. Advances in Engineering Software, 85, pp.1-11.
4. Huang, X. and Xie, M., 2010. Evolutionary topology optimization of continuum structures: methods and applications. John Wiley & Sons.
5. Sigmund, O., 2001. A 99 line topology optimization code written in Matlab. Structural and multidisciplinary optimization, 21(2), 120-127.