Topology Optimization for Minimal Compliance using "taichi.h"

1 Code

The homework codebase has only two files (**"taichi.h"** and **"topopt.cpp"**). To complete this homework, you are required to modify the "topopt.cpp" file. You will need to write about 100 lines of C++ code. There is no need to modify "taichi.h". You can use the "Save Image" button to create screenshots (saved under the executable folder) and attach them to your report. **A reference binary (reference/topopt_XXX) is shipped along with the assignment code for you to compare with.**

The code is cross-platform and you can choose either to use the virtual machine or not. Since it has only one file to compile, building and running is easy:

- Linux (recommended) [1]

```
g++ topopt.cpp -std=c++14 -g -lX11 -lpthread -O3 -o topopt
./topopt
```

- Windows with MinGW**64**[2]

```
g++ topopt.cpp -std=c++14 -g -lgdi32 -lpthread -O3 -o topopt
.\topopt
```

- Windows with Visual Studio 2017

  - Create an "Empty Project";

  - Use taichi.h as the only header, and topopt.cpp the only source;

  - Change configuration to "Release" and "x64";

  - Press F5 to compile and run.

- Mac OS X

```
g++ topopt.cpp -std=c++14 -framework Cocoa -lpthread -O3 -o topopt
./topopt
```

[1] In case you miss X11/Xlib.h, execute "sudo apt-get install libx11-dev".

[2] Please make sure you are using the 64-bit version. MinGW32 will not work. gcc version 6.4+ is recommended.

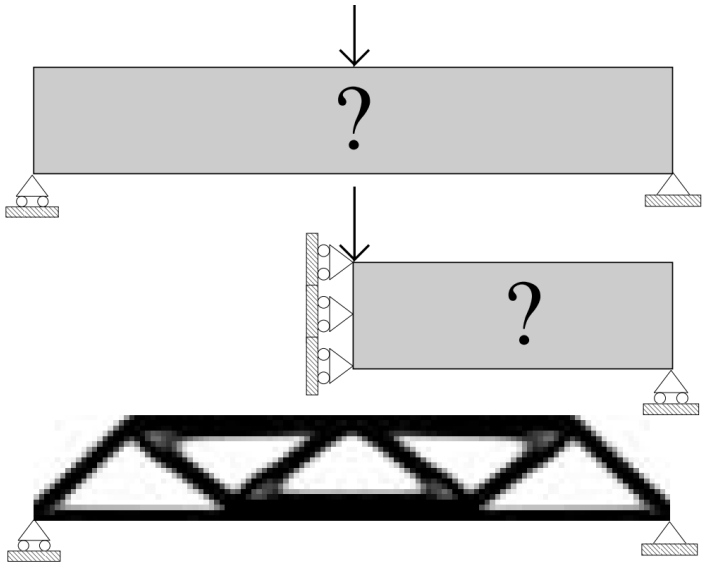

Figure 1: The "MBB beam" topology optimization problem. Figure is from [2]. (Top) The problem: given a completely fixed right anchor and a x-axis-movable anchor, what is the best structure the can resist a downward force? (Middle) According to symmetry we only need to solve the right half. (Bottom) The optimized structure.

2 Recap

From our classic linear FEM setup in homework 4, we have three terms we really care about:

1. $\mathbf{K} \in \mathbb{R}^{n \times n}$, where n is the number of degrees of freedom, is the stiffness matrix, which says something about the elastic forces in the system as the nodes displace;
2. $\mathbf{U} \in \mathbb{R}^n$, the displacement of every node in our mesh;
3. $\mathbf{F} \in \mathbb{R}^n$, the external load forces on the system. Here n is the degrees of freedom of our system. For the purposes of this discussion, we'll assume that all fixed degrees of freedom have already been removed from the system.

3 Assignment

In topology optimization, we seek a distribution of material density throughout a structure to optimize some target objective, given the constraints and load forces. We focus on regular voxels [3] grids in this assignment. The voxels can be either void or solid, but to utilize continuous optimization tools, during optimization we allow intermediate possibilities as well. SIMP (see below) is used to force each voxel to a either void or solid in the end. An example is in Fig. 1.

Under this scheme, our $\mathbf{K}$ matrix is parameterized by some vector $\boldsymbol{\nu} \in \mathbb{R}^m$, representing how "solid" each voxel is, where m is the number of voxels of our system. $\nu_i = \nu_{\min}$ means

[3] In fact, for simplicity we use a 2D "pixel" grid in this assignment; we will use voxel interchangeably.

voxel i is completely void, $\nu_i = 1$ means that voxel i is completely solid. Note that we disallow ν_i to be zero to avoid singularity in $\mathbf{K}(\boldsymbol{\nu})$. The most common topology optimization problem is **minimal compliance**:

$$\begin{aligned} \min_{\boldsymbol{\nu}} \quad & L(\boldsymbol{\nu}) = \mathbf{F}^T\mathbf{U} = \mathbf{U}^T\mathbf{K}(\boldsymbol{\nu})^{-1}\mathbf{U} \\ \text{subject to} \quad & \mathbf{K}(\boldsymbol{\nu})\mathbf{U} = \mathbf{F}, \\ & \sum_{i=1}^{m} \nu_i = cV, \\ & \nu_{\min} \leq \nu_i \leq 1, \end{aligned}$$

where $0 < c \leq 1$ is the target volume fraction, i.e. the ratio between solid material volume and the whole design space volume V. Typical values for $\boldsymbol{\nu}_{\min}$ is 10^{-2} or 10^{-3}. We use 10^{-2} in this assignment. Intuitively, we are asked to fill only fraction c of all voxels to be solid, and the resulted structure should **deform least** (as measured by work) under external load $\mathbf{F}$.

Note that there is no guarantee that this optimization leads to a binary result. Many voxels may have intermediate values such as 0.5, which is in practice not manufacturable. To solve this problem, we use the so-called "SIMP" (Solid Isotropic Material with Penalization) method, where the Young's modulus of a voxel i (E_i) is raised to the p-th power: $E_i = E_0\nu_i^p$, where E_0 is the Young's modulus of the solid material. In the end most ν_i will be either 0 or 1 so $\nu_i^p = \nu_i$. p is the **penalization** parameter, typically 3. Specifically,

$$\mathbf{K}(\boldsymbol{\nu}) = \sum_{i=1}^{m} \mathbf{K}_{e,i}(\nu_i) = \sum_{i=1}^{m} \nu_i^p \mathbf{K}_0,$$

where $\mathbf{K}_{e,i}(\nu_i)$ is the element stiffness matrix for voxel i and $\mathbf{K}_0$ is the element stiffness matrix for a completely solid voxel. Note that $p = 1$ corresponds to the case in our last assignment.

But how should we properly assign material to voxels? The general idea is to assign material to "important" voxels. Suppose you are given ε amount of material, which voxel would you like to enhance? Our solution is to greedily use the material in the most **sensitive** ones, i.e. those with largest **sensitivities** (i.e. gradients):

$$s_i = -\frac{\partial L(\boldsymbol{\nu})}{\partial \nu_i} = -p\nu_i^{p-1}\mathbf{U}_i^T\mathbf{K}_0\mathbf{U}_i,$$

where $\mathbf{U}_i^T$ are the displacements of voxel i. We have a negative sign here since we are doing minimization. We have implemented the FEM solver and the computation of s_i for you.

Then we can use gradient-based constrained optimizers given the sensitivity. Popular choices include **optimality criteria** [1] (OC) and the method of moving asymptotes [3] (MMA). We will implement the former in this homework.

Sensitivity filtering, another step which averages neighboring voxels, helps us get rid of checkerboard artifact. We will implement this technique as well.

The pseudocode for topology optimization is in Algorithm 1.

Algorithm 1 The topology optimization algorithm.

```
function TopologyOptimization
    Initialize density field and boundary conditions.
    while not converged do
        FEM Solve
        Sensitivity analysis
        Sensitivity filtering
        Update density field using OC
    end while
end function
```

3.1 Optimality Criteria (40 Points)

We have already implemented the FEM solver and sensitivity computation for you. The OC optimizer redistributes material density according to the sensitivity. More specifically, given an old material assignment $\boldsymbol{\nu}$ whose sum is cV, OC looks for a value λ (sometimes called a "Lagrangian multiplier") and updates $\boldsymbol{\nu}_i$ as

$$\nu_i' \leftarrow \min\left\{\max\left\{\nu_{\min}, \nu_i - M, \sqrt{\frac{s_i}{\lambda}}\nu_i\right\}, \nu_i + M, 1\right\}. \tag{1}$$

Intuitively, we scale ν_i according to a factor $\sqrt{s_i/\lambda}$, which is positively related to s_i. M is the **change limit**, which prevents the change in ν_i' to be bigger than M and stablizes optimization. We use $M = 0.2$ by default. Of course, we do not want ν_i' to be larger than 1 or smaller than $\nu_{\mathbf{min}}$ either.

It can be seen that ν_i' decreases monotonically (or at least does not increase if clamped) when λ increases, so is the sum of all voxels is $\boldsymbol{\nu}'$. We need to make sure the sum of $\boldsymbol{\nu}'$ is still cV, and there should be only one λ that satisfies this requirement. Use a binary search to find the λ value, such that the sum of $\boldsymbol{\nu}'$ is cV.

Now you are ready to implement the OC algorithm. Make sure you are using **double** precision for all arithmetic. The pseudocode is in Algorithm 2. To make sure you have implemented the OC algorithm correctly, compare your result with the reference implementation (set **filtering radius**=0 in the GUI, since you haven't implemented filtering). The result should look like Fig. 2.

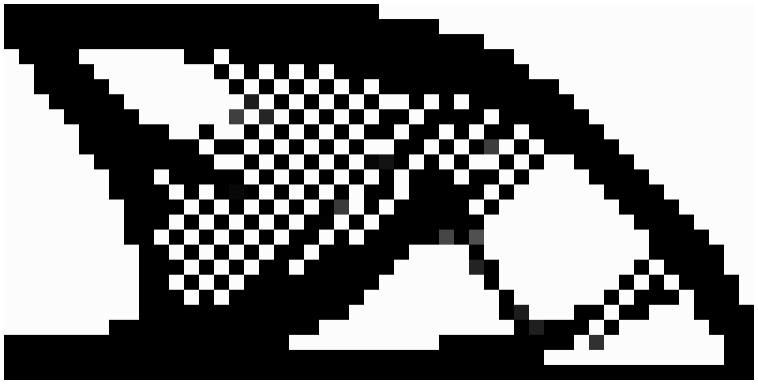

Figure 2: Result you will get after implementing OC. Note the checkboard artifacts, which will be addressed in the next part.

Algorithm 2 The Optimality Criteria (OC) algorithm.

function OPTIMALITY CRITERIA(c: volume fraction; V: total volume, cell_res[0] * cell_res[1] in your code)
    $\lambda_{\text{lower}} \leftarrow 0, \lambda_{\text{higher}} \leftarrow 10^{15}$,
    while $(1 + 10^{-15})\lambda_{\text{lower}} < \lambda_{\text{higher}}$ **do**
        $\lambda = (\lambda_{\text{lower}} + \lambda_{\text{higher}})/2$
        Compute ν_i' for all i using Formula 1.
        $S \leftarrow \sum_i \nu_i'$
        if $S < cV$ **then**
            $\lambda_{\text{higher}} \leftarrow \lambda$
        else
            $\lambda_{\text{lower}} \leftarrow \lambda$
        end if
    end while
    $\boldsymbol{\nu} \leftarrow \boldsymbol{\nu}'$
end function

3.2 Sensitivity Filtering (30 Points)

To get rid of the checkerboard artifact, we need sensitivity filtering before using the raw sensitivity in OC. The filtered sensitivity

$$s_i' = \frac{\sum_j w_{ij} s_j \nu_j}{\sum_j w_{ij} \nu_j},$$

where

$$w_{ij} = \max\{0, r - \text{dist}(i, j)\},$$

and $\text{dist}(i, j)$ is the Euclidean distance between the center of voxel i and voxel j, in the unit of edge length, and we use $r = 1.5$ in this assignment. For example, the distance between voxels with coordinates $(3, 4)$ and $(4, 6)$ is $\sqrt{5}$. After correctly implementing filtering, your

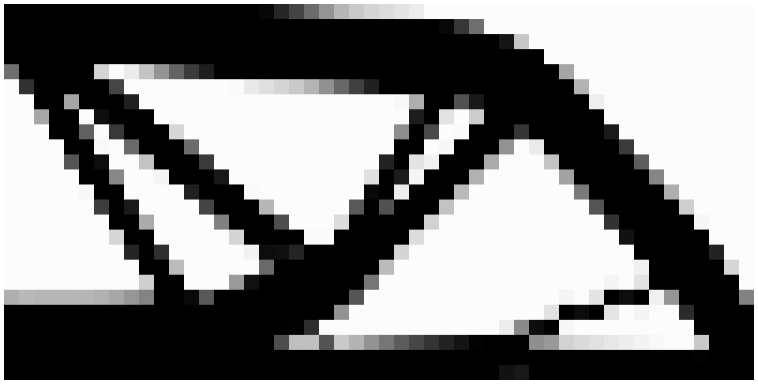

Figure 3: With filtering you can get rid of the artifacts.

implementation should generate the same result as the reference implementation, as shown in Fig. 3.

3.3 Experiments (30 Points)

Now it is time to have fun with what you have coded.

3.3.1 Cantilever Beam and Bridge Design (10 Points)

All the previous tasks are based on the "MBB beam" setting. As an additional example, construct a **cantilever beam**. This can be done by fixing both x and y axis of the lower-left and upper-left node, and applying a downward force to the middle-right node, as shown in Fig. 4.

Another interesting setting would be the **bridge**, where the lower two corners are fixed and all the top nodes are applied a downward force.

Implement these two boundary conditions (Fig. 4) and report the result with your implementation.

3.3.2 Parameter Study (15 Points, 6.839 Only - Extra Credit for 6.807)

Pick your favourite example with resolution = 100 (i.e. 100×50 elements), study the affect of the following parameters:

1. **Volume fraction. (5 Points)** Show results with volume fraction 0.3 and 0.6 respectively. How does this parameter affect the final structure?

2. **Filter radius. (5 Points)** Show results with filter radius 0, 1.5 and 3.0 respectively. How does this parameter affect the final structure? Please also briefly explain why.

3. **SIMP penalty. (5 Points)** Show results with SIMP penalty 1, 2, and 3 respectively. How does this parameter affect the final structure?

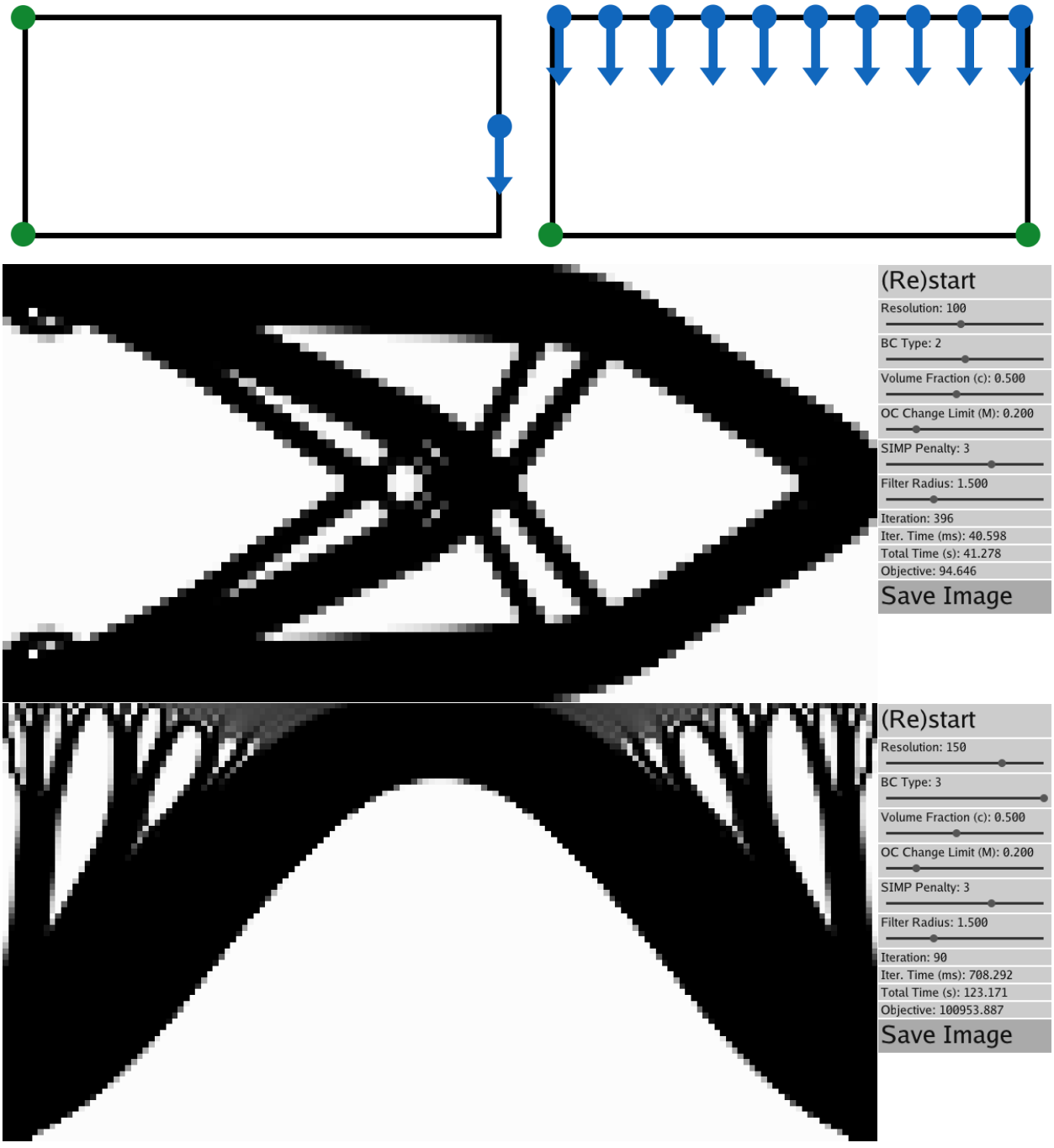

Figure 4: Top: The cantilever beam (left) and bridge (right) boundary conditions. The green dots indicates nodes to be completely fixed. Middle: The optimized cantilever beam. Bottom: The optimized bridge.

3.4 How does SIMP work? (5 Points)

You probably know how the SIMP penalty term affects the result. It seems to work magically, but how? What is the intuition behind? Briefly explain your reasoning.

In your report, please include the resulted images for the above experiments. You can make use of the "Save Image" button.

3.5 Extra Credit: Structural Symmetry Constraints (20 points)

Your optimized cantilever beam problem is probably quite symmetric in the y (top-down) direction. However, a structure that is also symmetric in the x (left-right) direction can be very interesting. Is there a way to ensure such symmetry in the result? Implement your approach, show the result, and explain your approach.

References

[1] Martin P Bendsøe and Ole Sigmund. *Optimization of structural topology, shape, and material*, volume 414. Springer, 1995.

[2] Ole Sigmund. A 99 line topology optimization code written in matlab. *Structural and multidisciplinary optimization*, 21(2):120–127, 2001.

[3] Krister Svanberg. The method of moving asymptotes—a new method for structural optimization. *International journal for numerical methods in engineering*, 24(2):359–373, 1987.