Politecnico di Milano

Advanced Programming for Scientific Computing

Topology optimization for 3d printing technology

Author:
Luca Ratti

Supervisors:
Prof. Nicola Parolini
Prof. Marco Verani

September 2015

Introduction

The aim of this project is to produce a C++ code to solve topology optimization problems involved with 3d printing technology. In particular, the main task is to tackle the problem of placing the additive supports needed in the process. When performing a 3d printing, indeed, it is sometimes required to add material in order to guarantee the stability of the object: even if the final shape of the object were in stable equilibrium, during the operation its partial shape could be instable. The objective of the project is to find the best way to sustain the printing object during the whole time of the procedure, involving tecniques of Topological Optimization theory. The starting point of the work is the C++ library TopOpt developed by N. Aage, E. Andreassen, B. Lazarov [AL14], which solves minimum compliance problems in three dimensions according to the algorithm formulated in [BS03] and others.
The structure of the report is as follows. The first chapter, 1, is devoted to the description of the problem, introducing the basic aspects of Topological Optimization theory and its application to minimum compliance problems. The solution algorithm proposed in [BS03] is briefly presented, and it is extended to the problems of interest, formulating a numerical procedure for the simulation. The second chapter, 2, focuses on the description of the C++ code, introducing the original features of the library TopOpt and focusing on the changes and extensions developed. The last chapter, 3, presents some applications on benchmark problems, in order to test the efficacy and the versatility of the implemented code.

Chapter 1

Problem Description

The 3d printing technology, introduced in the 1970s and widely developed in the last decades, is based on the procedure of constructing an object by sovrapposition of small layers of material following different techniques to obtain a solid shape (extrusion, photopolymerization, lamination), according to the properties of the employed material. In order to guarantee the stability of the object printed throughout the whole process, additional material is laid with the original shape, tipically creating small pillars in critical regions. The task of optimizing the support structure consists in finding the shape that ensures maximum stability with the least quantity of additional material.
In order to model properly such a problem, an interesting approach is represented by the topological optimization, which allows to find the shape a domain must assume (within a set of admissible shapes) to minimize a functional depending on it. A brief formulation of this theory is outlined in 1.1 together with its application to structural design problems. Moreover, an algorithm to solve numerically this kind of problems is described in 1.2, referring to the procedure shown in [BS03]. Finally, the section 1.3 depicts how to exploit such an approach for the purposes of tackling 3d-printing problem.

1.1 Topological optimization for structural design

1.1.1 Shape and topology optimization

The shape optimization theory can be set in the frame of PDE-constrained optimization problem. The main features of the latter are basically:

- a differential problem, called **state problem**, given by one or more PDEs and related boundary conditions. Its solution is called **state variable**.
- a **control variable**, which can be chosen within a set of admissible controls. It is a function or a parameter the state problem depends on, hence yielding a dependence on the state variable as well.
- a cost functional, depending on both the state and the control, or simply on the control thanks to the previous point.

The aim is to find the control variable which minimizes the cost within the set of admissible controls. In the case of shape optimization problems the *domain* itself on which the differential problem is defined is considered as control variable. In astract terms a shape optimization problem reads as follows:

Given a (Hilbert) functional space X, the differential problem on the domain Ω

$$?u \in X: \quad a(u,v;\Omega) = \ell(v;\Omega) \quad \forall v \in X$$

and the cost functional $J : \mathcal{O}_{ad} \to \mathbb{R}$, being $\mathcal{O}_{ad}$ the space of admissible domains, find

$$\Omega_{opt} \in \mathcal{O} \text{ s.t. } J(\Omega_{opt}) \leq J(\Omega) \quad \forall \Omega \in \mathcal{O}$$

The choice of the admissible domains is performed by prescribing the possible modifications of an initial domain Ω_0. As described in [CM00], this leads to three different philosophies:

1. **parametric optimization**, which supposes the initial domain to be described by some geometric parameters (such as sizing parameters, angles, ratios) and allows only to modify their values
2. **geometric optimization**, which allows each deformation of the initial shape except for the changes in the topology
3. **topological optimization**, which includes topological modifications such as introducing or removing holes, glueing edges, etc.

As in the case of any PDE-constrained optimization problem, we can rely on two different approaches for the solution of a topological optimization problem: *optimize-then-discretize* or *discretize-then-optimize*. The former one, presented for example in [CM00], requires to analitically describe the shapes and the deformations, introducing the possibility to derive the cost functional with respect to the domain and leading to the tools of the geometric and topological gradient of the functional (numerically approximated via FE spaces). The latter requires instead to first disctretize the shape of the domain by a discrete density function and than look for the optimal one. This is the approach followed in this work, and it will be described with more details applied to the problem of interest.

1.1.2 Minimum compliance problem

The application of topological optimization techniques to structure design is explored e.g in [BS03], and reveals useful for the purposes of this project. The benchmark problem involves an initial domain, where a linear elasticity is set (with prescribed loads and constraint) and requires to find the optimal distribution of a fixed fraction of the initial volume in order to maximize the stiffness of the object. The starting point is the formulation of the **continuum problem**:

Given the initial domain Ω_0, find the optimal stiffness tensor $E_{ijkl}(x) : \Omega_0 \to R$ s.t.

$$\text{it realizises the } \min_{\tilde{E} \in E_{AD}} l(u(\tilde{E})), \tag{1.1}$$

being $u(E)$ the solution of

$$?u \in X: \quad a(u,v) = \ell(v) \quad \forall v \in X$$

$$\text{with } a(u,v) = \int_{\Omega_0} E_{ijkl}(x)\varepsilon_{ij}(u)\varepsilon_{kl}(v)dx, \quad \varepsilon_{ij}(u) = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)$$

$\ell(v) = \int \Omega f v dx + \int \Gamma_N t u dx, \quad$ *(f = distributed loads, t = traction on the surface Γ_N)*

The choice of the space of admissible tensor is as follows:

$$E_{AD} = \left\{ \tilde{E} : \Omega_0 \to \mathbb{R} \text{ s.t } \exists \Omega \subset \Omega_0 : \tilde{E} = 0 \text{ in } \Omega_0 \setminus \Omega, \quad \int_{\Omega^{mat}} 1 dx \leq V, \, V \text{ fixed} \right\}, \tag{1.2}$$

that is, we are looking for a domain Ω within the bulk Ω_0, whose volume is below a fixed value and which minimizes the work of the external forces imposed to the object (compliance). Another typical constraint is to prescribe some fixed regions of void or of full space within the domain Ω_0, which are called *passive regions*.

According to the *discretize-then-optimize* approach, a mesh is introduced to approximate Ω_0, as well as a 0-1 density function $\rho(x)$, piecewise constant on each element: if an element has density 0 it will be considered void, if it has density 1 it will be full. In other words,

$$\rho(x) = \chi_\Omega(x), \quad \Omega \subset \Omega_0$$

The *Solid Isotropic Material* method consists in considering a material with fixed isotropic properties, and then reducing the set of admissible tensors to:

$$E_{AD} = \{ E_{ijkl}(x) = E^0_{ijkl} \chi_\Omega(x), \quad |\Omega| \leq V \text{ fixed} \} \tag{1.3}$$

Nevertheless, this highly discontinuous approach causes many problems with the uniqueness of the solution and convergence of possible solution algorithm. A very useful alternative is the *Solid Isotropic Material with Penalization* method (SIMP), which consists in considering the density $\rho(x)$ piecewise constant on the element but with intermediate values in the interval $[0, 1]$. Moreover, as the presence of fractional density has no physical meaning, intermediate values are penalized by choosing

$$E_{AD} = \left\{ E_{ijkl}(x) = E^0_{ijkl} \rho(x)^p, \quad \int_\Omega \rho \leq V \text{ fixed} \right\} \tag{1.4}$$

This means that intermediate values of density interpolate the stiffness property between 0 and E^0, but are unfavourable as the stiffness obtained is small compared to the "cost" of introducing volume. The choice of the penalization power has been object of studies, see for instance [BS99], in order to minimize the drawback of non integer densities on the physical model. It has shown to be related to the Poisson ratio ν^0 (depending on E^0), by the relationtip (in 3d geometries):

$$p \geq max \left\{ 15 \frac{1 - \nu^0}{7 - 5\nu^0}, 1.5 \frac{1 - \nu^0}{1 - 2\nu^0} \right\} \tag{1.5}$$

1.2 An algorithm for minimum compliance problem

Exploiting the described discretization procedure of the continuum problem with the SIMP method, it is possible to outline an iterative algorithm to get an approximated solution of the continuum problem (1.1).

1.2.1 Optimality condition

In order to obtain a set of first order necessary optimality condition we must introduce the following Lagrangian functional:

$$\begin{aligned} \mathcal{L}(u, \bar{u}, \rho, \Lambda, \lambda^+, \lambda^-) = & \ell(u) - \{ a(u, \bar{u}) - \ell(\bar{u}) + \Lambda \left(\int_{\Omega_0} \rho - V \right) + \\ & + \int_{\Omega_0} (\lambda^+(x)(\rho(x) - 1) + \int_{\Omega_0} (\lambda^-(x)(\rho_{min} - \rho(x)) \end{aligned} \tag{1.6}$$

where $\bar{u}$ denotes the adjoint state, Λ is the Lagrange multiplier related to the volume constraint, λ^+ and λ^- the Lagrange multipliers related to the upper and lower bound imposed on ρ (typically ρ_{min} is set slighty bigger than 0 to avoid singularity). The optimality conditions are obtained deriving the Lagrangian with respect to each variable, respectively:

1. $u = \bar{u}$ (meaning that the compliance problem is self-adjoint)
2. $a(u, v) = \ell(v) \quad \forall v \in X$ (imposing the solution of the state problem)
3. $$\frac{\partial E_{ijkl}}{\partial \rho} \varepsilon_{ij}(u)\varepsilon_{kl}(u) = \Lambda + \lambda^+ - \lambda^-$$
4. switching conditions: $\lambda^+ \geq 0, \quad \lambda^- \geq 0, \quad \lambda^-(\rho_{min} - \rho) = 0, \quad \lambda^+(\rho_{min} - 1) = 0.$

It follows that, in elements with intermediate values of density, $\lambda^+ = \lambda^- = 0$, and the third point states:

$$\Lambda^{-1} p \rho^{p-1} E^0_{ijkl} \varepsilon_{ij}(u)\varepsilon_{kl}(u) = const.$$

Hence, a possible fix-point scheme for updating the density (introducing also two tuning parameters to emprove convergence) reads as follows: starting from an initial guess ρ_0, iterate (until convergence)

$$\rho_{k+1}(x) = \begin{cases} max\{(1-\zeta)\rho_K, \rho_{min}\} \text{ if } \rho_K B_K^\eta \leq max\{(1-\zeta)\rho_K, \rho_{min}\} \\ min\{(1+\zeta)\rho_K, 1\} \text{ if } \rho_K B_K^\eta \geq min\{(1+\zeta)\rho_K, 1\} \\ \rho_K B_K^\eta \text{ otherwise} \end{cases}, \tag{1.7}$$

$$B_K^\eta = \Lambda_K^{-1} p \rho \rho_K(x)^{p-1} E^0_{ijkl} \varepsilon_{ij}(u_K)\varepsilon_{kl}(u_K)$$

The parameter ζ is called *moving limit*. The main remark on the formula is that it is possible to update each element indipendently except for the computation of Λ_K, which must enforce the global volume constraint. Moreover, each iteration of the algorithm requires to solve the differential problem with the current density function in order to compute B_K. Note that the problem is self adjoint, thus only the state problem has to be solved at each iteration.

1.2.2 Sensitivity analysis

Another solving technique frequently used is the *sensitivity analysis*, which consists in considering the cost functional not as a function of u but of the density ρ_e of each element. In this way, the computation of the derivatives of the cost with respect to them leads to:

$$c(\rho) := \ell(u(\rho)) \quad \Rightarrow \quad \frac{\partial c}{\partial \rho_e} = -p\rho_e^{p-1} E^(e)^0_{ijkl} \varepsilon_{ij}(u)\varepsilon_{kl}(u).$$

Hence, a gradient-like method can be implemented in order to update the density ρ_K element-wise by means of the sensitivity of the functional. A particular method based on this procedure is the *Method of Moving Asymptote* (MMA), which introduce a parameter whose behaviour is similar to the ζ introduced before. More details about MMA can be found in [Sva02]. Thanks to the faster convergence of this algorithm, MMA has been used as update scheme in the present work.

1.2.3 Algorithm

The solver algorithm, as proposed in [BS03], is:

Algorithm 1. (general minimum compliance problem)

1. Preprocessing: define the domain Ω_0 and introduce a computational mesh
2. Preprocessing: define the physics (loads, boundary conditions, passivity) and set up FE space solvers
3. Optimization: choose an initial design (respecting the volume constraint)
4. Optimization: iterate (until a stopping criterion is fullfilled)
   A) compute displacement field via FE
   B) evaluate compliance
   C) evaluate (and filter) sensitivities
   D) update density (MMA)
   E) check stopping cryterion
5. Post Processing: extract optimal shape

In particular, we build a staggered mesh, made of equal parallelepides or cubes. As shown in the description, the solution of the state equation is performed through the Galerkin Finite Element Method: the costruction of the stiffness matrix for 3d staggered mesh is detailed, for example, in [LT14] [ZT00]. The initial guess about the design typically consists in spreading the volume fraction homogeneously on the whole initial domain, ensuring to fulfill the constraint. The stopping criterion can be chosen whether on the relative decrease of compliance or on the change of the density between iterations, computed as:

$$\frac{1}{|\Omega_0|}\|\rho_{k+1} - \rho_K\|_{L_\infty \Omega}$$

The filtering tecniques, instead, are meant to regularize the data coming from the sensitivity analysis, tipically with a uniform filter of selected radius.
The presented algorithm is the core of the C++ library TopOpt that will be described later. It reveals very versatile and efficient, but shows some limitations. The most relevant one is the impossibility to fix passive regions within the domain; this new feature has been implemented in the extension of the library developed within this project.

1.3 An algorithm for the 3d-printing problem

The problem of optimal position and shape of the supports in 3d printing can be approached with the strategy sketched in the previous section. First of all, it is important to correctly state the problem, giving a mathematical interpretation to the engineering issue:

Given an initial domain $\Omega_0 = [x_0, x_1] \times [y_0, y_1] \times [z_0, z_1]$, containing a subdomain Ω_{print} (representing an object to print), define $\Omega(t) \subset \Omega_0 : \Omega(t) = [x_0, x_1] \times [y_0, y_1] \times [z_0, z_0 + vt]$, where $t \in [0, T]$ denotes a time instant and v is the mean velocity of the deposition of layers during the printing. Let f be the function describing the load distributed all over Ω_{print}, and suppose the material satisfies the laws of linear elasticity, with an homogeneous stiffness tensor E^0_{ijkl}. Consider a Dirichlet boundary condition on the lower boundary, $[x_0, x_1] \times [y_0, y_1] \times \{z_0\}$, and free stress conditions on the others. Denote the linear elasticity problem in the domain

$\Omega(t)$ as:

$$a(u, v; t) = \ell(v; t) \quad \forall v \in X(t)$$

which means that all the integrals are computed not in the whole Ω but only in $\Omega(t)$. Fixed a prescribed amount of volume V, find the density function ρ which minimizes the functional

$$J(u(\rho)) = \max_{t \in [0,T]} \ell(u(\rho); t).$$

In order to solve numerically the problem, some approximations are required. First of all we can introduce a computational mesh on the domain, and follow the already presented *discretize-then-optimize* to minimize compliance. Moreover, the time discretization can be achieved by selecting some time instants $\{t_0 = 0, t_1, \dots t_N = T\}$ and considering N steady problems on the domains $\Omega(t_n)$.
Two crucial aspects are investigated in the following subsections.

1.3.1 Passive regions and effective volume

In the 3d printing problem, it is important to set the domain Ω_{print} always full, otherwise the optimization will try to reduce the density of the region which enforces the loads. This can be done by fixing the MMA update scheme, for example. Moreover, the elements which do not show any full element above themselves can be considered as void, defining a region $\Omega_{eff} \subset (\Omega_0 \setminus \Omega_{print})$ of effective free volume. In this project, the fixed amount of volume V is prescribed through the variable $volfrac$, the maximum fraction of effective volume we would like to fill. This is a slighty difference with the traditional minimum compliance algorithm, where one prescribes a fraction of the entire volume Ω_0.

1.3.2 Timesteps

When solving the problem on the domain $\Omega(t_n)$, in order to discretize the continuum problem correctly, one must consider some relationships with the previuos timesteps. This involves the inheritance of the supports created in the solution at previous steps and a correct division of the volume fraction between them. The first issue is fulfilled by allowing an *inprinting* of each timestep from the previous one: fixed a threshold value for the density, we select those elements that are above it, and consider them as passive full regions for the following timesteps. A clarification is in order: the new passive regions do not have any load, because this is done throughout the process of minimization of the compliance (otherwise, each step would require the creation of different FE matrices). This can have a physic interpretation considering the existing technique of using two different materials for printing the object and the support structure. The second task, related with the correct distribution of material between timesteps, is performed as follows: among the effective free volume of the timestep, $\Omega_{eff}(t)$, we define an *actual* effective free volume considering only those free elements that have a full element somewhere above themselves but within the domain $\Omega(t)$. Each timestep will then be given as maximum volume the prescribed fraction times its actual free volume, even if the other effective free regions will not be considered passive, as they could be usefully filled.

1.3.3 Algorithm

Collecting all the information from previous sections, the following procedure is proposed:

Algorithm 2. (3d printing problem)

- Global preprocessing: set up the entire domain (geometry, physics, passivity,...)
- Iterate, on each timestep:
  1. Preprocessing: define the subdomain $\Omega(t)$
  2. Preprocessing: define the physics, paying attention to passivity and imprinting from previous steps
  3. Optimization: choose an initial design (dividing the volume fraction as explained)
  4. Optimization: iterate (until a stopping cryterion is fulfilled)
     A) compute displacement field via FE
     B) compute compliance
     C) compute (and filter) sensitivities
     D) update density (MMA, with corrections)
  5. Post Processing: prepare the inprinting for next step

Chapter 2

Code Description

In this chapter we present the code produced to implement the proposed algorithm. It has been developed exploiting the existing C++ library TopOpt, designed by N. Aage, E. Andreassen, B. Lazarov and described in [AL14]. A first description of the structure and the main classes of the library is reported in section 2.1. The description of the new implemented code is the object of 2.2, whereas in section 2.3 we show the links with other tools and libraries that have been performed the improve it.

2.1 Starting point: TopOpt library

As explained in the paper [AL14], the main task of the C++ library TopOpt is to develop an easy-to-use and easy-to-extend basis to implement the depicted topology optimization framework on large scale, various physical problems. The code is based on PETSc, a freely available library for high performance scientific computing [al.13], which consists in a collection of parallelized libraries containing several features required for the purpose: sparse matrices, vectors, iterative linear solvers and non-linear solvers. Hence, mesh generation, partitioning in processes, linear algebraic computation and solvers are delegated to already existing, well tested functions, and the focus is put on the physics and optimization aspects.
The TopOpt library consists in a main program and five C++ classes, dividing the different issues in separated units. In particular:

- **Topopt** contains all the information about the geometry, the discretization and optimization parameters of the problems.
- **LinearElasticity** is devoted to the physical aspects of the problem, containing information about loads and boundary conditions. It also provides methods to set and solve the state problem, together with the computation of the cost function and of the sensitivities.
- **Filter** and **PDEFilter** provide different schemes to filter density and sensitivity, according to [Bou01]
- **MMA** contains the (parallel) implementation of the method of moving asymptotes to update the density.
- **MPIIO** is responsible for the output of the iteration and of the final result in a binary file.

Finally, the Phyton scripts **makevtu** and **bin2vtu** provide a conversion from binary output to vtu format, allowing post processing and visualization with graphic tools as ParaView. The main program implements an algorithm similar to the one described in **Algorithm 1**, for the problem of minimum compliance. The whole library is quite versatile within this class of problem (just changing the geometry in TopOpt and the physics in LinearElasticity), whereas some modifications must be implemented in order to use it for 3d-printing problems

2.2 Changes and extensions: ToPrinter library

The aim in extending the TopOpt library is to introduce the possibility of solving the problem of interest preserving the original structure as far as possible. Particular attention has been paid to the correct implementation of the features explained in 1.3 and to guarantee versatility within the class of problems, making easy to change the parameters and the geometry. The strategy by which this has been performed is to collect all the main information of the problem (geometry, physics, passivity) within the class TopOpt and then to create a derived class, ToPrinter, in order to manage the specificy of the problem.

2.2.1 Modifications to TopOpt

The first purpose of the modifications in the library has been to collect the physics information of the problem within the class TopOpt, adding the attributes **elemToLoad** and **dofsToClamp** which are passed to the class LinearElasticity (thanks to the fact that it was already constructed by passing a pointer to an object TopOpt). This has the effect that the class LinearElasticity does not need to be modified when changing the loads or the boundary condition, and that task is delegated to TopOpt (with the new method **setUpInterface**).
Moreover, some changes are required in the update procedure of the class MMA in order to treat properly the passive regions. The possibility of not-changing regions was not considered in the library, and so two methods have been introduced in TopOpt and in MMA: **setUpPassive** and **FixPassive**. The first one creates a (PETSc) vector containg information about the passivity of each element (whether it is full, void or free). The second one is called inside the Update method of the MMA class, before the evaluation of the Lagrangian multiplier of the volume constraint. FixPassive imposes the prescribed properties to passive elements, enforcing the update scheme to the 0-1 value on that elements and causing the other densities to be rescaled to fit the volume constraint.
With this two modifications, the library is ready to be applied to the class of problems we are interested in, just by extending the faculties of the TopOpt class.

2.2.2 Class description: ToPrinter

ToPrinter is the new class introduced to deal with each single timestep optimization procedure. An extract of declaration of the class is reported:

```
class ToPrinter: public TopOpt{
private:
  // New optimization parameters
  Vec geometry;
  PetscInt timestep;
  std::vector<PetscScalar> inprinting;
  ...
  // Set Up method
  PetscErrorCode SetUpGeometry(std::function<PetscScalar(PetscInt,
      PetscInt,PetscInt)>);
  PetscErrorCode SetUpGeometry(std::vector<PetscScalar>);

public:
```

```
  // Constructors/Destructor
  ToPrinter(PetscInt,std::vector<PetscScalar>)
  ToPrinter(PetscInt,std::vector<PetscScalar>,std::vector<PetscScalar>&,
      PetscInt,PetscInt,PetscInt)
  ...
  // Overriding methods
  PetscErrorCode SetUpInterface() override;
  PetscErrorCode SetUpPassive() override;
  ...
  // New Methods
  PetscErrorCode FixVolFrac();
}
```

The first element we want to focus on is the inheritance from the TopOpt class. This has been done to exploit the structure of the existing code, where the majority of the methods occuring during the main program (even the constructors of the other classes' objects) recieve as an argument a pointer to a TopOpt object. The inheritance allows the use of dynamic binding: a pointer to a derived class is created and assigned to a pointer to the base one. Whenever it is needed, a pointer to the ToPrinter object can be used as a pointer to a TopOpt one. This allows to reduce the changes in the code, although it requires modifications within the TopOpt class, es explained before. In our opinion this is the best choice since it condenses all the new features in a single class, making easier to understand and use it.

Another important aspect is the attribute **geometry**, which is the core of the class. This is a PETSc vector of dimension equal to the number of element of the mesh, which contains information about all the important features of each element. In particular, it has value 1 on elements that are forced to by full, -1 on the ones forced to be empty, 0 on the free ones and 2 one the ones that have been filled be the previous timesteps (and must be preserved full). This attribute becomes the centre of the method **setUpInterface**, which reads the information from geometry and initializes the loads and the passivities to be passed to the other classes. This procedure is new with respect to the class TopOpt, and allows to describe a complete problem only by initializing the vector geometry. The method which is devoted to this task is **setUpGeometry**, which has been set private and it is called by the constructor of the object ToPrinter. We implemented two different options to set up the geometry: by aims of function wrappers or of an existing vector. The first one is related to the possibility to describe the shape of the object to print through a function which takes as input the (rescaled) coordinates of each element and gives as output numerical values according to the rules described above. This allows to implement simple benchmark cases by simply writing a lambda function and pass it to the set up method. The initialization by vector responds to the necessity to apply the present code on pre-existing geometries, possibly coming from other computations. The input file is supposed to be a vector of zeros and ones, describing the void and non-void regions of the object. In both cases, the set up method is supposed to cut the geometry up to the height of the current timestep, fixing to void each element above.

Other features of the class are related to the timestep managements, such as the new attributes necessary to take into account the current step and their total number. The issue of properly dividing the volume fraction is related with the method **FixVolFrac**, which is called during the construction of the density vector and initializes the free element following the strategy explained in 1.3. At last, the inprinting procedure is managed by the function **OutPrint**, which saves the final density in a vector which is later on passed to the constructor of the new timestep. The

decision not to filter the density with the fixed threshold before passing it to the new timestep has been taken foreseeing a possible future improvement, where the threshold of the previous timestep could be modified inside the FixVolFrac procedure of the following, in order to obtain a more precise attainment to the volume constraint.

2.3 Third-part tools and libraries

The code structure and implementation exploits some third-part tools, following the purpose of general extensibility and versatility. Moreover, as explained before, one of the most important features of the extended library TopOpt was the utilization of the PETSc library to achieve efficient and suitable parallelization.

2.3.1 PETSc

PETSc is the acronym for *Portable and Extendable Toolkit for Scientific Computing*, and consists in a collection of parallelized libraries that contain most of the necessary building blocks needed for the presented topology optimization solver. The benefits apported to the code by exploiting the PETSc library are various: they guarantee efficient computation and well-tested parallelization of the most expensive methods; the library is easy to install and customize, and hides the complexity of the parallelization to the user. The extension apported to the TopOpt is mainly related to adapting the existing algorithm to the problems of interest, and this has required to interface with methods providing creation an management of vectors, matrices and exceptions.

2.3.2 CMake

CMake is a very powerful tool for the configuration and the management of the dipendences. Although the original TopOpt library already has a complete makefile, the large number of scripts and the dipendence by the PETSc external library suggested the use of such a tool. Indeed, CMake provides modules to search the needed packages in the system and set a link them during the compilation. A CMakeList has been produced for the project, which permorms the search of the PETSc library and creates the executable *toprinter* in the build directory. The whole project directory has been rearranged in a more ordered way, creating subfolders for source files (src), headers and other files to be included (include), output of the code (output and img), configuration parameters (config), CMake specific tools (cmake) and the build directory. More details on how to compile the sources and run the executable are given in the last chapter.

2.3.3 GetPot

The GetPot library consists in a single header file which implements the GetPot class, that allows the parsing of options specified by command line or by configuration files at runtime. In particular, in the present project it has been used for the latter task: in order to avoid recompilation at each change of geometric, physical or optimization parameters, a configuration file has been created (**config.pot**, in the config folder), which contains all of them. The GetPot library allows to read their values at runtime, meaning that a change inside the configuration file does not require a new compilation of the project, resulting in a overall advantage in time saving and versatility of the code. A switching parameter has been introduced between the already implemented geometry, hence it is possible to solve completely different problems without modifying and recompiling the sources.

2.3.4 ParaView

ParaView is the application used for the result visualization and post processing. It is an open-source, multi-platform tool which allows to explorate and analyze the data by and interactive graphic 3d user interface. The output file created during the execution of the code is processed by the phyton scripts already given in the original library to obtain vtu files. However, the script **bin2vtu.py** has been largely modified in order to obtain snapshots all over the optimization process. Thanks to ParaView, it is possible to create images and animations of the evolution of the geometry, plotting the displacement field or the density distribution and setting thresholds for the visualizations.

Chapter 3

Examples of application

In this chapter we present some examples of application on particular geometries already implemented in the library. The first benchmark problem (that will be called **A**) is a stair, that is a very simple and effective example of object which needs to be substained during the printing procedure.The second prototype problem denoted with **B** models a realistic tool to be printed. The shapes are shown in 3.1 and 3.2, colouring in red the object to be printed and in pale blue the free effective volume.

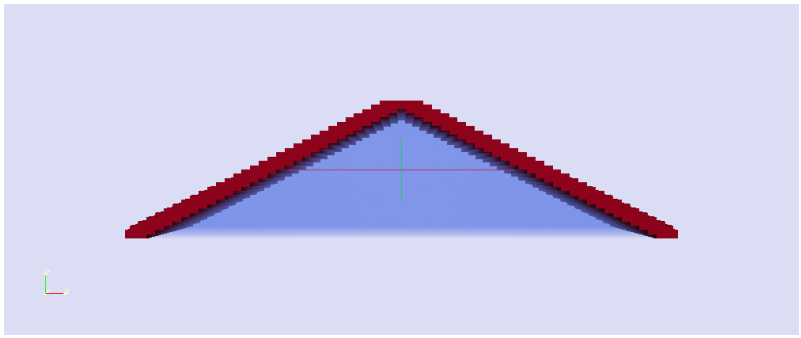


Figure 3.1: Benchmark problem A

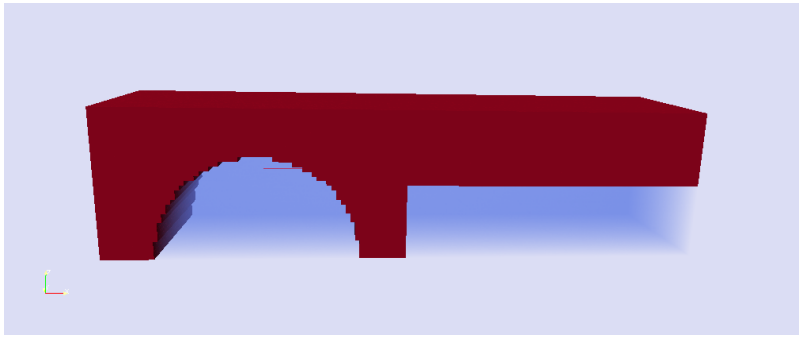


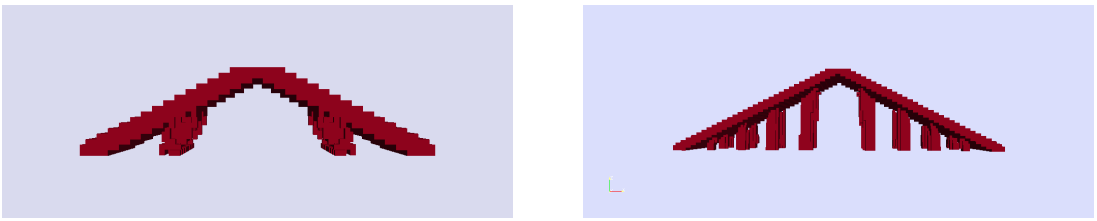
Figure 3.2: Benchmark problem b

They have the same aspect ratios and physic properties, but they can be easily customized:

Dimensions	$[0,4] \times [0,1] \times [0,1]$
Poisson ratio	0.3

3.1 Dependence on the mesh refinement

A first aspect we want to remark is that, due to the involved discretization procedure followed, the obtained results depend on the refinement of the mesh significantly. This means that numerical solutions of the same problem (same shape, same physics, same volume fraction) but on different meshes could be dramatically different. An example is reported in 3.3



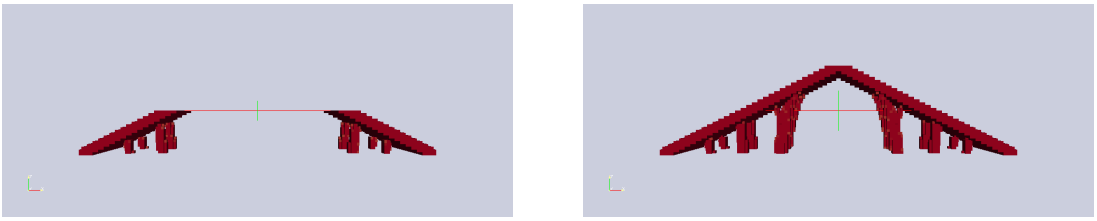
(a) Lower-quality mesh **(b)** Higher-quality mesh

Figure 3.3: Same problem on different meshes

3.2 Results on benchmark problem A

We report two different simulations on the stair problem, in order to show the effect of the different tuning of the parameters to the solution: both volume fraction and timestep subdivision are changed, obtaining sligthly different results.

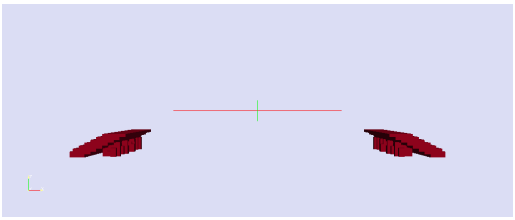
	Problem A01
Number of elements	$128 \times 32 \times 32$
Volume fraction	0.10
Max nr. of optimization it.	200
Timesteps	2



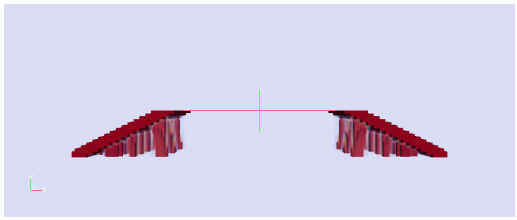
(a) Solution of timestep 1 **(b)** Solution of timestep 2

Figure 3.4: Simulation A01

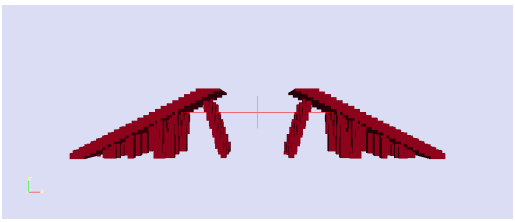
	Problem A02
Number of elements	$128 \times 32 \times 32$
Volume fraction	0.20
Max nr. of optimization it.	200
Timesteps	4



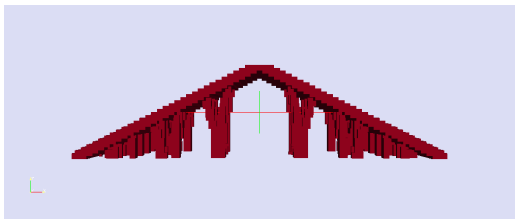
(a) Solution of timestep 1



(b) Solution of timestep 2



(c) Solution of timestep 3



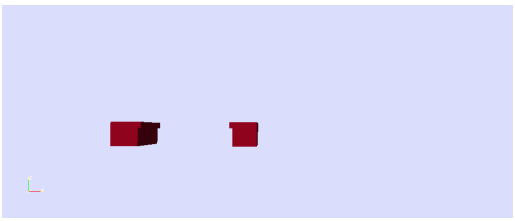
(d) Solution of timestep 4

Figure 3.5: Simulation A02

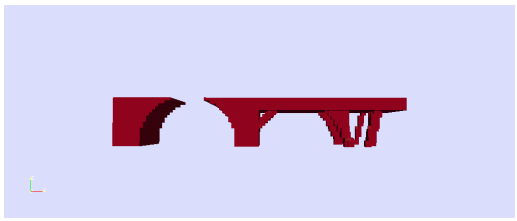
3.3 Results on benchmark problem B

The second problem was tested with other settings, to show some good properties of the algorithm produced. As in the first step the structure is stable, no supports are added, then the second step is standard and finally the last step's geometry causes a strengthening of the existing pillars (see 3.6)

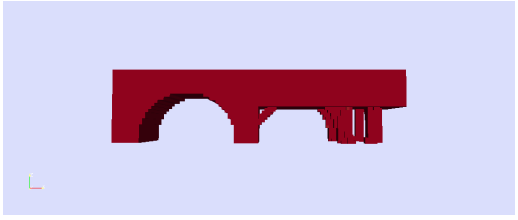
	Problem B01
Number of elements	$96 \times 24 \times 24$
Volume fraction	0.20
Timesteps	3



(a) Solution of timestep 1



(b) Solution of timestep 2



(c) Solution of timestep 3

Figure 3.6: Simulation B01

Conclusion

3.4 Code utilization

The ToPrinter library can be easily configured and compiled using the commands collected in the bash file **compile_and_run.sh**. Setting the local directory where the PETSc library is installed is required, and then the CMake file is called with the needed specification for the compiler. After the compilation, the executable produced is launched with the command **mpiexec**, specifying the number of threads to use and possibly the maximum number of iteration of the optimization loop. Later, the python scripts are invoked, specifying the number of optimization steps and timesteps too process. The output files can be found in the **output** folder, including the files needed to restart the optimization loop after the end of the execution and the files containing the evolution of the cost functional throughout the steps. The .vtu files are contained inside the **img** folder, where they can be opened with ParaView.

3.5 Possible extensions

The purpose followed in the present project has been to extend the TopOpt library implementing a suitable algorithm for the 3d-printing problem. In particular, specific attention was given to the preservation of the original structure and to the versatility of the code. At present, the ToPrinter can solve a large amount of problems by simply changing the parameters in the configuration file or implementing new geometries through analytic description or passing a vector. An important extension would be to create an interface with the files in **.stl**, the standard format for 3d printing projects, allowing the possibility to create the geometry starting from such files. Moreover, it would be interesting in terms of efficiency to try different linear solvers for the state problem, taking into account the presence of void regions within the computational domain. Finally, the introduction of other constraints to the problem (such as a perimeter constraint for the support structure) could also be taken into account, which would require modifications in the physics and optimization classes.

Bibliography

[AL14] E. Aage N. Andreassen and B.S. Lazarov. "Topology optimization using PETSc: An easy-to-use, fully parallel, open source topology optimization framework". In: *Structural and Multidisciplinar Optimization* 51 (2014), 565–572.

[BS03] M.P. Bendsoe and O. Sigmund. *Topology Optimization: Theory, Methods and Applications*. Springer-Verlag, 2003.

[BS99] M.P. Bendsoe and O. Sigmund. "Material interpolation schemes in topology optimization". In: *Archives of Applied Mechanics* 69(9-10) (1999), pp. 635–654.

[Bou01] B. Bourdin. "Filters in topology optimization". In: *International Journal for Numerical Methods in Engineering* 50(9) (2001), 2143–2158.

[CM00] P. Céa J. Guillaume and M. Masmoudi. "The shape and topological optimizations connection". In: *Comput. Methods Appl. Mech. Engrg* 188 (2000), pp. 713–726.

[LT14] K. Liu and A. Tovar. "An efficient 3D topology optimization code written in Matlab". In: *Structural and Multidisciplinar Optimization* 50 (2014), pp. 1175–1196.

[Sva02] K. Svanberg. "A class of globally convergent optimization methods based on conservative convex separable approximations". In: *Siam Journal of Optimization* 12(2) (2002), pp. 555–573.

[ZT00] O.C. Zienkiewicz and R.L. Taylor. *The Finite Element Method (Parts 1 - 3)*. Butterworth Heinemann, 2000.

[al.13] S. Balay et al. *PETSc users manual. Tech. Rep. ANL-95/11 - Revision 3.4*. Argonne National Laboratory, 2013.