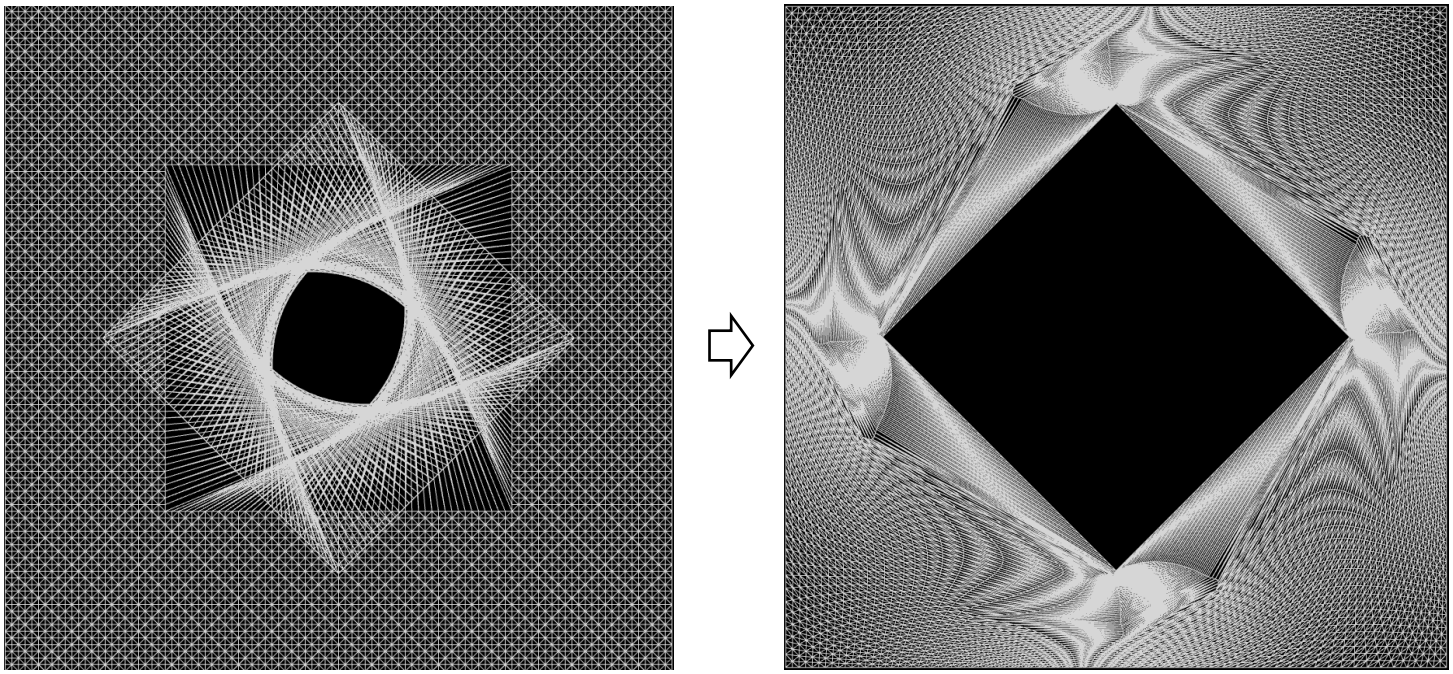
b
UNIVERSITÄT
BERN

Foldover-Free Maps: An Evaluation



Maksim Fomin
CGG 2021
Universität Bern

Acknowledgements

I would like to thank Prof. David Bommes for introducing me to the initial idea for this thesis. Furthermore, I want to express my deepest gratitude to his assistant Valentin Nigolian for supervising the whole process of my work and giving essential inputs at every weekly meeting. In times of technical complications, his invaluable guidance offered great motivation and majorly helped me to progress further.

Abstract

The state of local injectivity occurs when there are no inverted elements left on the mapping of a mesh. In many applications of the mesh mapping from texture mapping to shape deformation, local injectivity can potentially improve the quality of results. This work deals with a recently proposed approach to solve the local injectivity problem in the context of 2D triangular meshes, i.e. to restore the local injectivity for 2D meshes with inverted triangles present. This approach stems from the field of computational physics and adapts the formulations to work with meshes with foldovers. The present work reproduces this approach, evaluates it with numerous respects and goes further by proposing one more variation of the solving method.

Table of Contents

1. Introduction

In the field of geometry processing, mapping of meshes to 2D resp. 3D spaces is among the most important problems. Whenever for any two distinct elements of the initial domain there are exactly two distinct elements in the mapping range, we talk about local injectivity. In the case of 2D triangular meshes, the local injectivity is achieved once there are no more inverted triangles and no two faces overlap (Fig. 1). Such overlapping will be referred to as a mesh foldover. The following thesis deals with a recent mapping approach [Garanzha 2021] intended to recover local injectivity for maps with foldovers present, i.e. to untangle an input mesh, which is one of the main mapping challenges.

The practical importance of such mapping cannot be overstated. In most applications ranging from remeshing and texture mapping to volume deformations, it is generally much easier to work with map spaces instead of having to work directly with an object itself. Notably, the approach proposed in the reference paper claims to allow for optimization of the map distortion, proposing a customizable tradeoff between area and angle preservation.

It should be noted that, although the method is reported to work for 3D volumes and surfaces, this report focuses on mapping of 2D meshes. Moreover, my implementation is based on triangular meshes while the reference paper is primarily based on quad meshes, even though the functionality with quad meshes was confirmed on early prototyping stage.

The motivation of the actual research is to implement this algorithm in C++ and to evaluate initial claims about its efficiency and robustness. Furthermore, I will make advantage of additional libraries and implement the algorithm with three different approaches. These approaches will be put in comparison against one another and listing cases where one implementation is a better choice than other two. Before that, the theoretical background needs to be presented to the reader as well as a short mention of instruments used for the actual implementation.

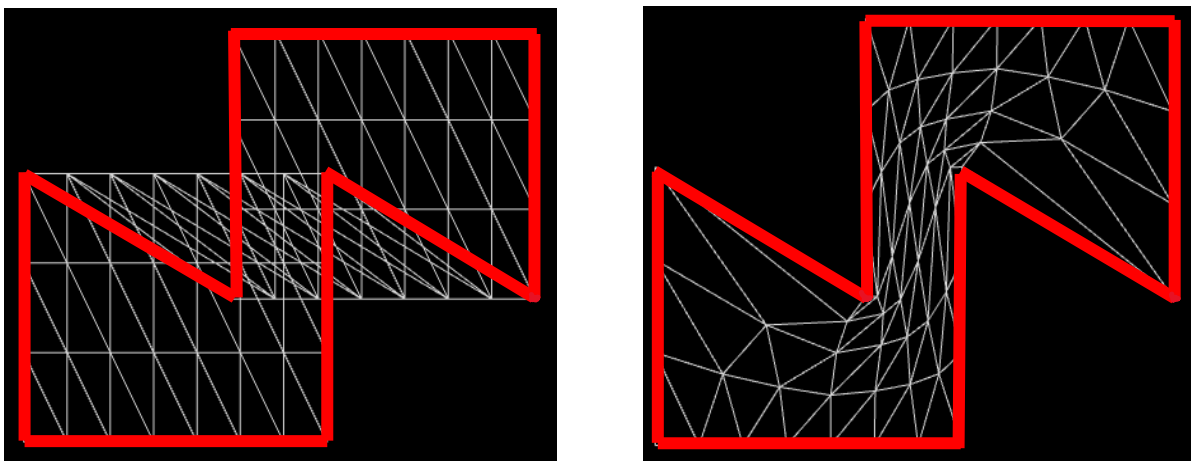


Figure 1. Input 2D-mesh (left-hand side) and the untangled output (right-hand side). The boundary constraints are marked with red.

2. Theoretical Background

This section covers the theoretical concepts used for the untangling approach. To achieve the local injectivity, i.e. to unfold a mesh, we need a proper formalization for the energy of the mesh, and then we will need to minimize that energy. The actual approach at the energy formalization expands upon a well-known general variational problem that would otherwise fail for our conditions, which will be justified later on. After covering the formulas used to compute the energy, I'll present the untangler algorithm which minimizes it and the individual solving approaches.

2.1. Jacobian

For all calculations, we need to define a proper Jacobian matrix for the 2D-case and triangular meshes. For each triangular face with vertices located at (x0, y0), (x1, y1), (x2, y2), its Jacobian is composed as follows:

$$J = \begin{pmatrix} x1 - x0 & x2 - x0 \\ y1 - y0 & y2 - y0 \end{pmatrix} * S^{-1},$$

where S^{-1} is the inverse of the target shape matrix

$$S = \begin{pmatrix} 1 & \frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} \end{pmatrix},$$

with the targeted shape for each face of the mesh being an equilateral triangle in our case. The columns of the shape matrix are basically the x- and y-coordinates of the second and third vertices respectively, given the first vertex at (0,0).

2.2. Energy Formulation

The untangling method discussed in this paper is based on the following variational problem, derived from the theory of finite hyperelasticity introduced by J.Ball [Ball 1976] and intended for non-untangling operations:

$$\arg\min_{\vec{u}} \int_{\Omega} (f(J) + \lambda g(J))\ dx,$$

with the functions f and g having two concurring goals and λ serving as a trade-off parameter between those. The expression inside the integral is the energy. Note that $\vec{u}$ in our case denotes a 2D map to a parametric domain $\vec{u}(x,y) = (u(x,y), v(x,y))$.

The function f preserves the angle/shape of a face to that of a targeted equilateral angle,

$$f(J) := \begin{cases} \frac{\operatorname{tr} J^{\top} J}{(\det J)^{\frac{2}{d}}}, & \det J > 0 \\ +\infty, & \det J \leq 0 \end{cases}$$

with the parameter d indicating the dimension we work in, i.e. d=2 for this paper.

The function g preserves the area of a face

$$g(J) := \begin{cases} \det J + \frac{1}{\det J}, & \det J > 0 \\ +\infty, & \det J \leq 0 \end{cases}$$

As it is, the above formulation can improve the quality of a map without any foldovers. On its own, however, this formulation is practically useless when working with a mesh *with* foldovers present: in the location of a foldover, the rearward face is flipped, i.e. it faces the opposite direction, therefore having a reversed orientation of its vertices leads us to negative values for its Jacobian determinant and hence infinite values for the energy.

To get around this problem, a modification to the energy formulation was proposed in the reference paper. The idea is to introduce a regularization function χ

$$\chi(D, \varepsilon) := \frac{D + \sqrt{\varepsilon^2 + D^2}}{2}$$

and then substitute the Jacobian determinants in the denominators of the f and g functions with χ with D := det J,

$$f(J) := \begin{cases} \frac{\operatorname{tr} J^\top J}{(\det J)^{\frac{2}{d}}}, & \det J > 0 \\ +\infty, & \det J \leq 0 \end{cases} \quad \Longrightarrow \quad f_\varepsilon(J) := \frac{\operatorname{tr} J^\top J}{(\chi(\det J, \varepsilon))^{\frac{2}{d}}}$$

$$g(J) := \begin{cases} \det J + \frac{1}{\det J}, & \det J > 0 \\ +\infty, & \det J \leq 0 \end{cases} \quad \Longrightarrow \quad g_\varepsilon(J) := \frac{\det^2 J + 1}{\chi(\det J, \varepsilon)}$$

This achieves a regularized version of the variational problem which now enables us to work with finite energy values:

$$\lim_{\varepsilon \to 0^+} \arg\min_{\vec{u}} \int_\Omega \left(f_\varepsilon(J) + \lambda g_\varepsilon(J)\right) \, dx$$

Now, our goal will be to iteratively decrease this energy and for that we should update the epsilon at each iteration.

2.3. Epsilon and its Update Process

The regularization parameter ε used by χ should be updated based on the current state of the Jacobian determinants of all faces. To be more precise, the minimum of the current Jacobian determinants will be used in the computation of the current epsilon at iteration k

$$\varepsilon^k := \sqrt{10^{-12} + 4 \cdot 10^{-2} \cdot \left[\min(0, \min_{t \in 1 \ldots \#T} \det J_t^k)\right]^2}$$

Once the minimum Jacobian determinant becomes positive, i.e. once the mesh is untangled, epsilon will take the value 10^{-6} until the energy reaches convergence. Notably, the above formula is just one way to compute epsilon, while two other update rules were presented between two versions of the reference paper. For the actual report, only this, empiracally chosen update rule, was implemented. As was claimed in the early version of

the reference paper, the formula at hand provides a much better performance than a more conservative update rule. The published version of the paper provides a third epsilon update rule while fully omitting the first two, which would definetely be worth looking into in future work.

2.4. Energy Minimization

Having made all the preparations, we can now minimize the regularized energy of a map with foldovers with the help of a backtracking linesearch algorithm. For this, there were two approaches proposed in the reference paper – either use a Quasi-Newton solver such as L-BFGS, or a Newton solver. For this report, both approaches were implemented.

ALGORITHM 1: Computation of a locally injective map

Input: U^0; // *initial guess (vector of size $\#V \times d$)*
Input: *useQuasiNewton*; // *boolean to choose the optimization scheme*
Output: U; // *final locally injective map (vector of size $\#V \times d$)*

$k \leftarrow 0$;
repeat
  compute ε^k ; // *regularization parameter, Eq. (6)*
  if *useQuasiNewton* **then**
    $U^{k+1} \leftarrow$ L-BFGS (U^k, ε^k); // *inner L-BFGS loop*
  else
    compute a modified Hessian matrix $H^+(U^k, \varepsilon^k)$;
    $\Delta U^k \leftarrow (H^+)^{-1} \nabla F(U^k, \varepsilon^k)$; // *conjugate gradients*
    $U^{k+1} \leftarrow \arg\min_{\tau} F(U^k + \tau \Delta U^k, \varepsilon^k)$; // *line search*
  end
  $k \leftarrow k + 1$;
until $\min_{t \in 1 \ldots \#T} \det J_t^k > 0$ ***and*** $F(U^k, \varepsilon^k) > (1 - 10^{-3}) F(U^{k-1}, \varepsilon^{k-1})$;
$U \leftarrow U^k$;

Figure 2. Pseudo-code of the actual algorithm provided in the reference paper.

At the start of each iteration, epsilon gets updated. Then, if we go for a Quasi-Newton solver, we delegate the energy minimization to the solver. Otherwise, we use the Newton method. This approach requires all triangles to have a positive semi definite Hessian, but the energy formulation doesn't guarantee to satisfy this requirement. That's why we modify the Hessian matrix to make it positive semidefinite. Then, a Newton step is computed and the minimization process is thus delegated to the Newton method. This algorithm shall proceed until no triangles are flipped over *and* until the energy is converged. (Fig.2)

2.4.1. L-BFGS

The L-BFGS method proved to be a good choice in the early prototyping stage, mostly so because it doesn't require second order derivatives, unlike the Newton solver. Instead, it uses an estimation of the inverse Hessian matrix. Notably, this estimation is implicit, i.e.

for a problem with n variables, not the whole dense $n \times n$ approximation is stored, but only m recent pairs of vectors (set to 10 in the actual implementation). This guaranteess a much better performance with larger problems, than otherwise with an explicit estimation, such as the case with the original BFGS.

To get the search direction, the estimation of the second-order derivative is multiplied with the gradient.

The main drawback of this choice for a solver are potentially slower convergence rates for larger problem sizes, since we're talking about linear convergence. Whether this will stay true in practice, will be discussed in the evaluation section.

2.4.2. Newton's Method

Considering its quadratic convergence, Newton's Method has a promising potential to an improved performance over L-BFGS, especially when working with larger problems. The search direction, also called Newton step, is calculated similarly to the one for L-BFGS, except that now it's the real Hessian matrix, and not just an approximation of it, involved:

$$\Delta x_{nt} = -\nabla^2 f(x)^{-1} \nabla f(x)$$

However, as it is now, we can't be sure whether the Hessian for all triangles will be positive semidefinite, which is the method's requirement. To achieve positive semi definiteness, we can either modify the Hessian by projecting it or use a modification recipe provided in the reference paper. For the implementation, both approaches were taken. While the Hessian projection was provided with the used solver library as a method, the Hessian modification was implemented in accordance with provided formulas. The basic idea of this Hessian modification is to remove the terms which prevent the Hessian from being positive semidefinite, and thus the computation cost for such modification is reduced. From now on, I'll refer to the first approach as the projected Hessian and to the second approach simply as the modified Hessian, even though technically they are both Hessian modification approaches.

2.4.3. Stopping Criteria

At each iteration, the epsilon and then the vertices' positions get updated, which results in the energy of the map getting updated, too. Therefore, we need some checks at the end of each iteration which should stop the algorithm in case of their positivity.

First, we can say that a mesh is untangled whenever no more flipped triangles remain, i.e. when each triangle has a positive Jacobian determinant. In the context of practical implementation, this is the same as having a positive signed area for every triangle, which will be important in the next section.

Second, there is a test for convergence at the end of each iteration, which is performed by comparing the current energy with its previous value. We consider the energy converged once the relative difference between the actual energy and the previously updated energy is smaller than 10^{-5}. Convergence only comes after the mesh is untangled, and in scenarios with bigger meshes it may take noticeably more time than the untangling itself while converging immediately after the untangling for the smaller scenarios.

3. Methodology

In the first paragraph of this section, the evaluation methodology as well as the benchmark data are going to be presented, while in the second paragraph, I'll summarize the architecture of the actual implementation with a mention of tools used.

3.1. Evaluation Method

To evaluate the untangler with its three solver-based variations, we let it process numerous input problems of variable size and complexity, i.e. the foldover rate in our case – a number of inverted triangles per total number of triangles. At the end of the process, we expect the unfolded map output and, whenever we put one variation against another, what we care about the most are the performance times, the final energy values and the visual result.

What concerns us about the final energy is its relative difference to the final energy of the same mesh unfolded with another variation. The best result is the relative final energy

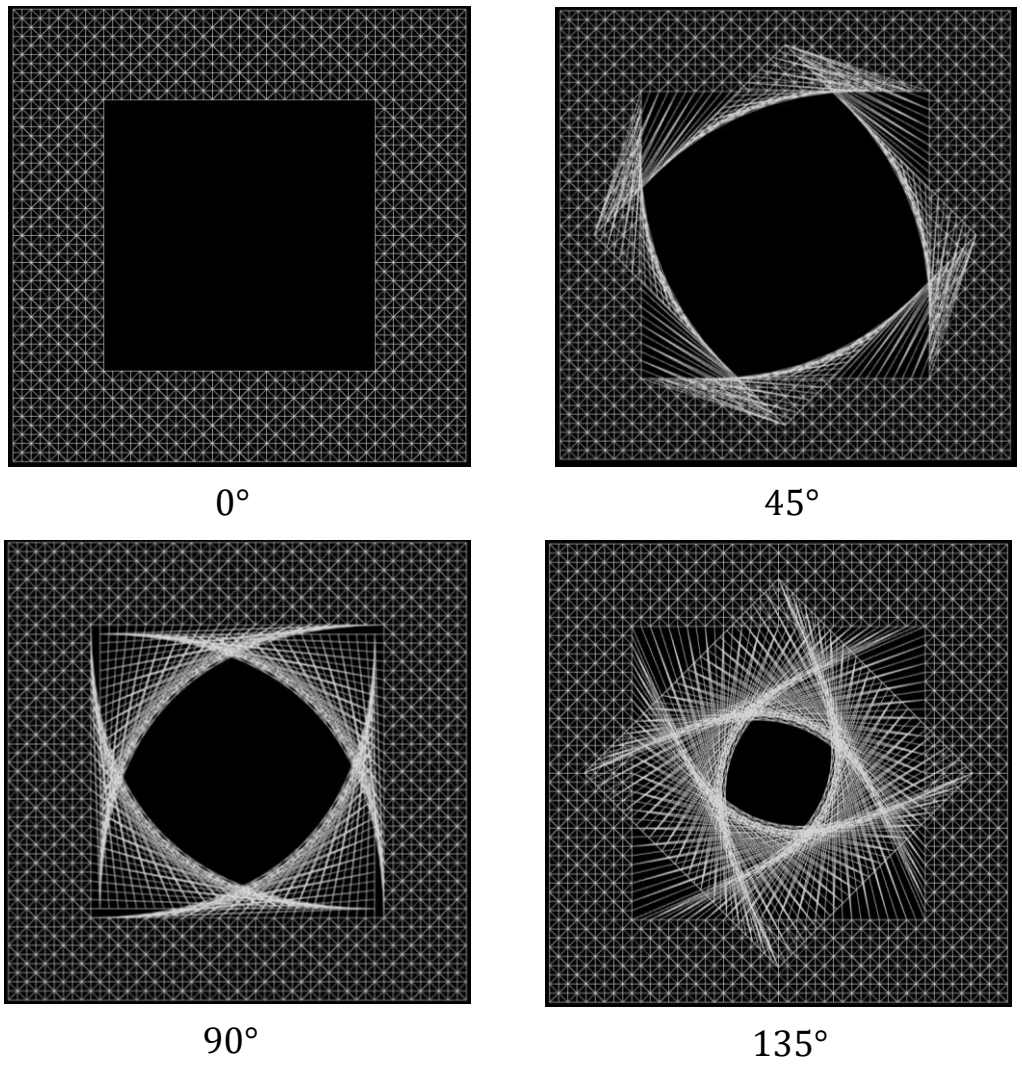


Figure 3: A 50 by 50 grid mesh with a square hole with a variable rotation applied to

difference being as small as possible, i.e. variations converging to roughly the same minimum. The tolerance value lies somewhere by 0.5% while the values above may be alarming for us.

The procedure to generate an input problem to solve is as follows: let us generate a square grid mesh and put a square hole of a variable size inside of it. Let us then apply a variable rotation to this hole to cause the foldovers. (Fig. 3)

The grid size, the hole size and the rotation angle, all three factors, dedicate to the complexity of the input problem. If we increase just the square hole, then the problem will become more complex for the untangler due to a higher foldover rate, despite the mesh itself having less vertices in the center. For the benchmark, three square grid mesh sizes were used – 20, 50 and 100 vertices wide, three square hole sizes – 1/3, 1/2 and 3/5 of the mesh's width respectively (rounded up to the next integer), and three rotation angles - 45°, 90° and 135°.

There are multiple relations for which we'll perform evaluations. First and foremost, we would like to compare in terms of performance and final result the approaches taken by the untangler – L-BFGS, Newton's Method with projected Hessian and Newton's Method with modified Hessian. Hereinafter, I'll refer to both Newton Method's variations as N+pH and N+mH respectively.

Then, we will compare the three variations with the lambda parameter set to different values. For the previous evaluation, the lambda was set to zero and thus the scenario of shape preservation was chosen, also known as the angle preservation. This time, we would also like to set the lambda to 10'000 and 1 for the scenarios of the area preservation and the angle/area-tradeoff, respectively. For the visual presentation of the evaluation of lambda variation, I'll also use a 40 by 40 grid mesh with a 10 by 10 hole rotated by 135° to reduce the level of visual artifacts which could otherwise distract the reader from some important observations.

It was claimed in the reference paper that the discussed algorithm is initialization-independent in terms of final energy, unlike some previous works in this field, such as

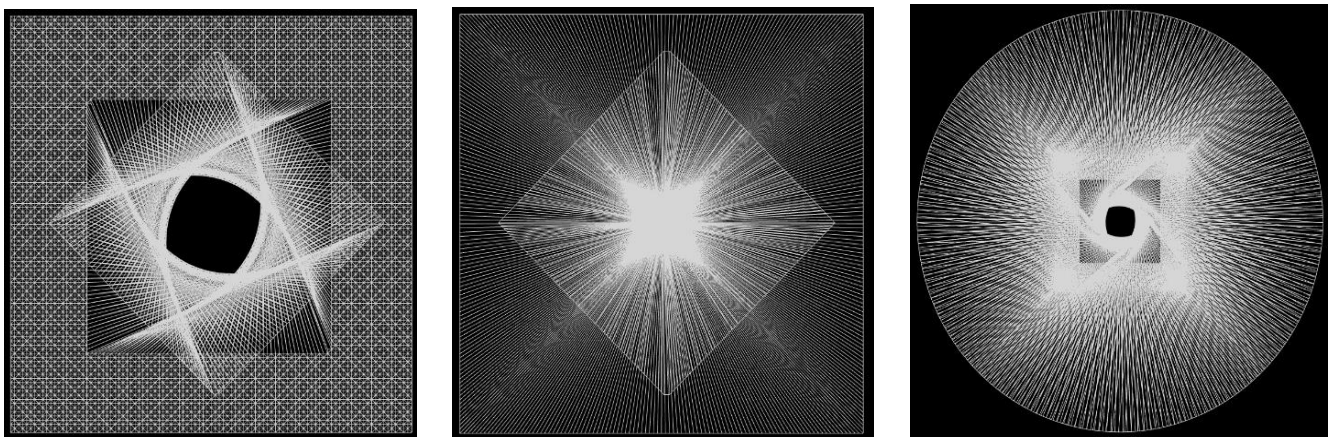


Figure 4: A 100 by 100 grid mesh with a square hole rotated by 135°, generated with three different initializations: the grid initialization to the left, the center initialization in the middle and the outer circle initialization to the right. Between the three cases, the only thing they have in common is the boundary of the mesh.

Total Lifted Content [Du et al. 2020], which provided [Garanzha 2021] quite different results based on the initialization. This property would mean that the energy converged to largely the same minimum independent from an initialization. To check this claim, let us use three different initializations of the same mesh – the grid initialization used in all the previous evaluations, the center initialization and the outer circle initialization (Fig.4). Notably, we check this claim for the final result, based on both the final energy relative difference and the visual output, and so the performance differences are tolerable.

3.2. Implementation

The mesh untangler for this thesis was implemented with C++ as a plugin for the ***OpenFlipper*** [Möbius 2019] application. The central class is *FOFMappingPlugin* (with the abbreviation *FOF* standing for "foldover-free", the desired quality of the processed mesh). It connects the logic of the methods to UI, initializes the untangler and parses the input mesh data to it, delegates the generation of problem inputs to the dedicated class *GridMeshGenerator*, and overall provides a connection to ***OpenFlipper***. The architecture of the implementation is briefly summarized in diagram on Fig. 5.

The calculations are mostly performed within two classes – *MeshUntangler* and a dedicated class for the energy computation – *FOFEnergy*. The energy computation is performed differently for L-BFGS and the Newton method. To minimize the energy with L-BFGS, a header-only library ***L-BFGS++*** [Qiu 2021] was used, and the evaluation of the required gradient, i.e. the derivative of the energy with respect to the Jacobian, was implemented manually. For the Newton method, a header-only solver library ***CoMISo*** was used, and not only did it offer to us the tools to minimize the energy, it also provided the methods to evaluate the Hessian matrix and project it to meet the requirement of positive semi definiteness. The method to modify the Hessian in another way was implemented according to the formulas provided in the reference paper. For both approaches of the Newton method, the required gradient was set-up automatically with the help of the library ***TinyAD***. This library allows for automatic differentiation and has not been published yet.

To find out whether we could benefit from the TinyAD gradient when using L-BFGS, the performance of both gradients was compared. For that, the class *FOFEnergyTinyADWrapper* was used as a counterpart to *FOFEnergy*, which adapted the energy computed with the help of automatic differentiation to be compatible with L-BFGS. The comparison of both gradients takes place in *MeshUntangler* and we compare the gradients performance both when combined with L-BFGS and just their initialization times. The evaluation of this comparison will come in the next section.

MeshUntangler represents the main logic of the algorithm. The checks on the stopping conditions are performed at the end of each iteration. As mentioned in the previous section, we first check for a positivity of minimum Jacobian determinant among faces, and then for a convergence of the energy. It may happen, however, that the first check delivers a false positive or even a false negative when working with a large number of tiny triangles, due to a potential numerical imprecision. In other words, a flipped-over triangle may wrongfully have a positive Jacobian determinant or a non-flipped triangle may in the

same manner have a negative Jacobian determinant. In our evaluation, we'll often witness such cases. Thus, we need some high-precision tools to bypass this problem. Therefore, it was decided to also check that each triangle has a positive signed area, which is a distinctive sign of a non-flipped-over triangle. For this check, the library ***CGAL*** [The CGAL Project 2021] was used due to its reliability with precision-sensitive math operations.

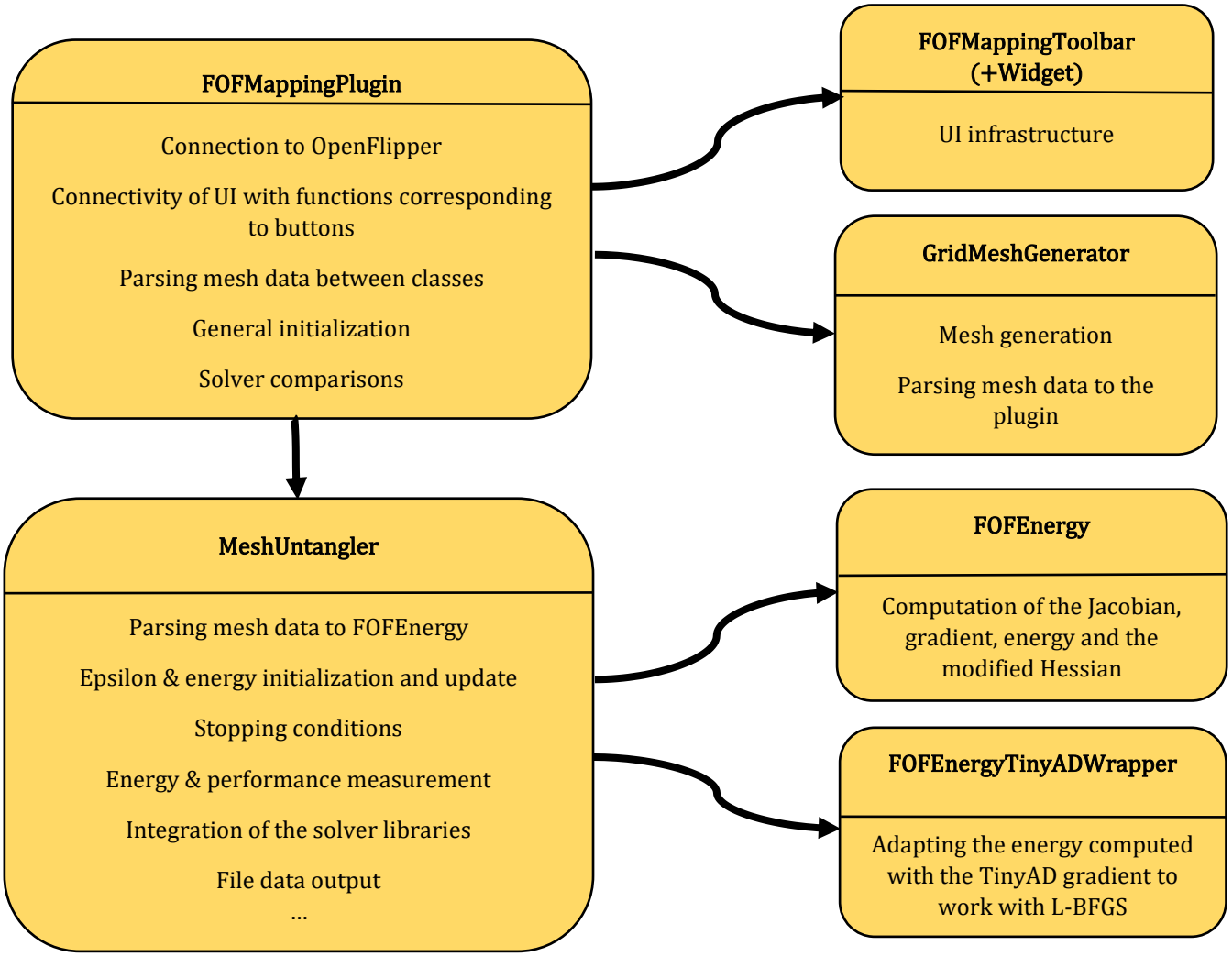


Figure 5: A brief summary of the architecture of the plugin with the respective roles of classes.

4. Evaluation

In this section, the untangler will be evaluated as follows. First, L-BFGS will be compared to N+pH. Then, both Hessians will be compared to show just how the modified Hessian is a worthy upgrade over the projected Hessian. After that, all three methods will be evaluated based on the lambda parameter. Afterwards, the very important claim from the reference paper about the untangler's independence from mesh initialization of the discussed algorithm will be put to test and confirmed. Lastly, a brief evaluation of two gradient evaluation approaches will be presented in the context of using with L-BFGS.

4.1. Comparison between L-BFGS and the Newton Method

The general intuition for this comparison is for L-BFGS to become slower than the Newton solver with increasing the problem size and/or foldover rate (i.e. the number of inverted triangles per total number of triangles), but how far should we go to confirm it? For this comparison, let us set the lambda to 0.

Let us first take a rather small problem from the benchmark (Fig. 6). After untangling a 20 by 20 grid mesh with a 11 by 11 hole and its rotation by 45°, we see L-BFGS performing faster than the N+pH in both finding a valid solution *and* converging. Let us then increase the foldover rate by applying a bigger rotation. For the rotation by 90°, L-BFGS finds a valid solution in 0.029 seconds and converges in 0.060 seconds and N+pH finds a valid solution in 0.147 seconds and converges in 0.255 seconds, which is a rather big difference in favor of L-BFGS. But once we apply the rotation by 135°, we can see that, while L-BFGS is still faster at finding a valid solution, its convergence becomes slower than the Newton Solver's convergence. Of course, we're still talking a mere second difference between the two. But the trend will become problematic with an increased mesh size.

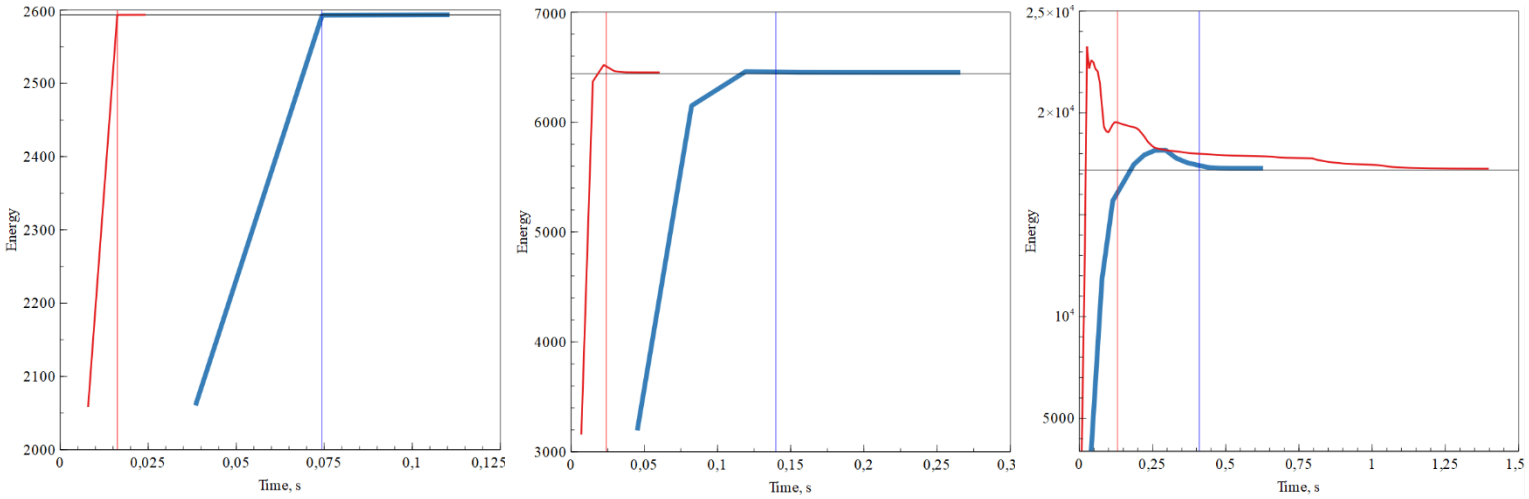


Figure 6: Comparison of L-BFGS and Newton with projected Hessian for a 20 by 20 mesh with a 11 by 11 hole rotated by 45°, 90° and 135° respectively. A colored vertical axis indicates where a solver has finished untangling, while a black horizontal axis indicates the point of convergence.

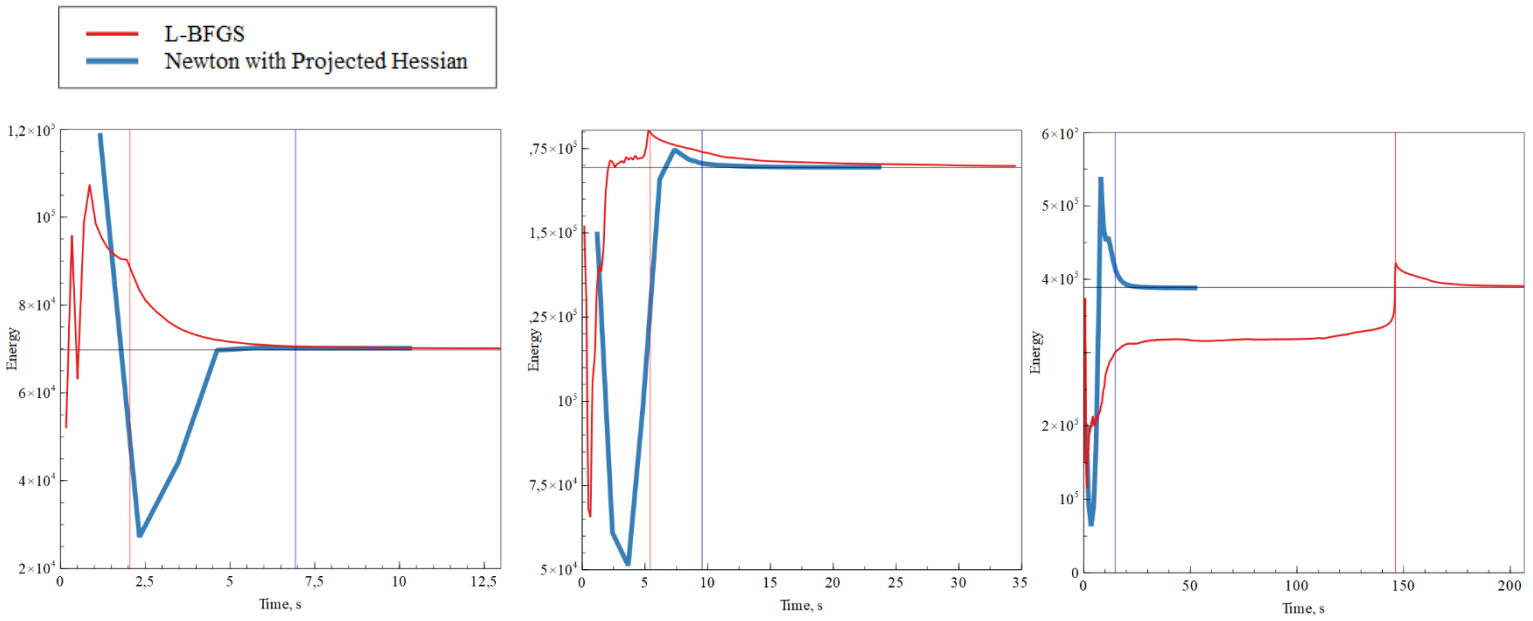


Figure 7: Comparison of L-BFGS and Newton with projected Hessian for a 100 by 100 mesh with a 59 by 59 hole rotated by 45° ,90° and 135° respectively.

For the grid mesh size 100 by 100, the performance downside of L-BFGS becomes apparent. (Fig. 7) Already with the rotation angle of 45°, it converges 2.5 seconds slower than the Newton solver, albeit it still is 5 seconds faster at finding a valid solution. The trend grows for the rotation by 90° - L-BFGS untangles 3.5 seconds faster but converges 12 seconds slower. With the rotation angle of 135°, even the search for a valid solution delivers a staggering difference of some 130 seconds when compared to the N+pH, not even counting the added minute for the energy convergence.

The reader may observe considerable energy spikes. Indeed, the untangler usually starts with a smaller energy which is starting to grow rapidly. The rapid growth is connected to the initial rapid decrease of the epsilon value. The epsilon, in its turn, decreases as much as the minimum Jacobian determinant has a bigger, that is, a closer to zero value at each next iteration (see Paragraph 2.3). Figure 8 illustrates a close-up on the energy and the epsilon behavior.

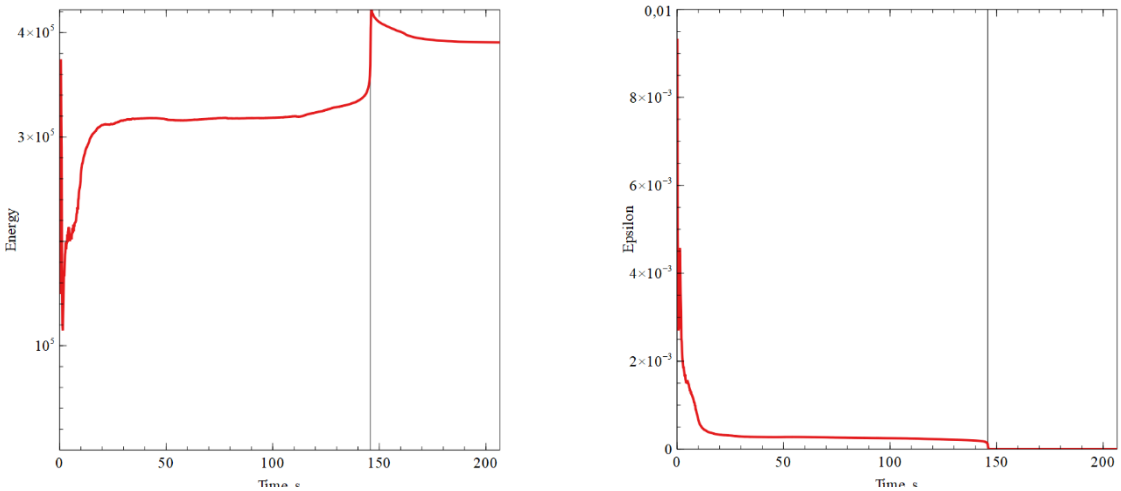


Figure 8: Two plots illustrating the dependency of the energy (left) from the epsilon value (right). The reference mesh is 100 by 100 with a 59 by 59 hole rotated by 135° and untangled with L-BFGS.

Notably, for the whole benchmark there was no significant final energy difference between the L-BFGS and the Newton method. Some other testing scenarios are listed on Figure 9.

Just as expected, the untangler based on L-BFGS is not as reliable in terms of performance as the Newton solver for scenarios with big meshes and is overall not an efficient option for problems with bigger meshes and/or higher foldover rate.

Mesh			Time to untangle, s		Time to converge, s	
Mesh Width, vertices	Hole Width, vertices	Rotation angle	L-BFGS	N+pH	L-BFGS	N+pH
20	5	135°	0.028	0.15	0.028	0.15
	9		0.04	0.20	0.15	0.55
	11		0.13	0.40	1.41	0.62
50	15	45°	0.11	0.73	0.11	0.73
	24	90°	0.24	1.15	1.00	1.15
	29	135°	18.36	2.38	24.54	12.51
100	59	45°	2.10	6.90	12.84	10.30
		90°	5.55	9.19	33.99	22.80
		135°	144.27	14.99	202.85	50.31

Figure 9: A table presenting nine scenarios of comparisons of L-BFGS and Newton's Method with projected Hessian in terms of finding a valid solution and convergence.

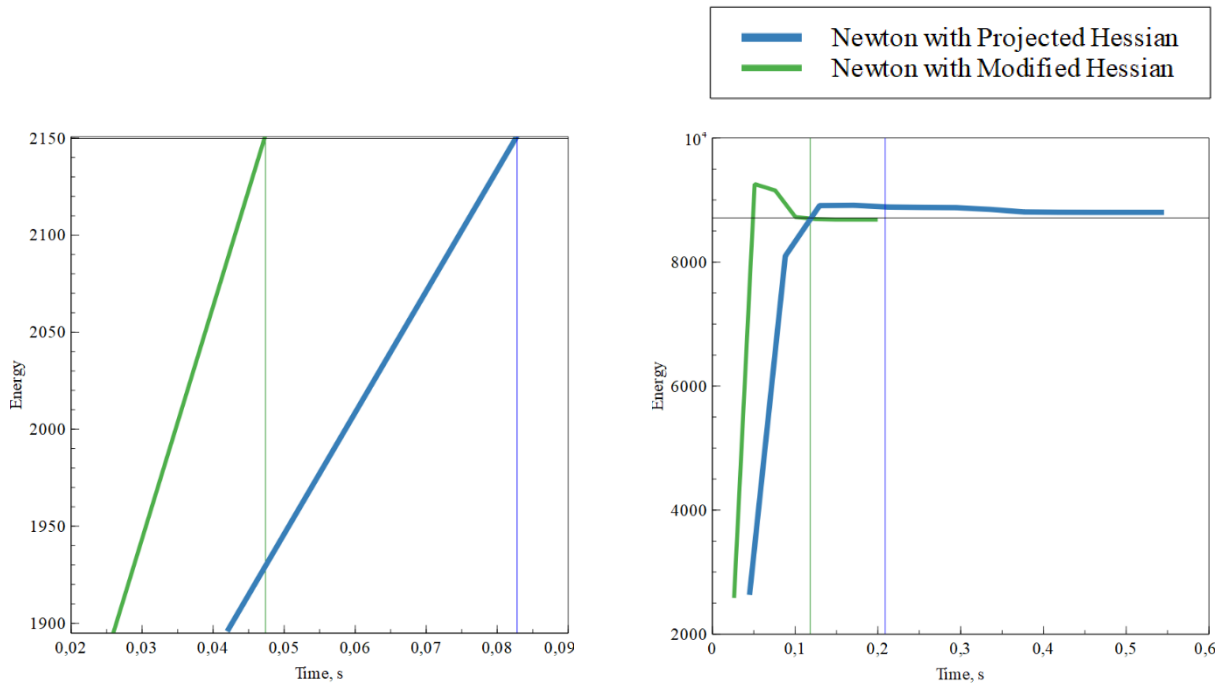


Figure 10: Comparison of Newton with projected and modified Hessian for a 20 by 20 mesh with a 9 by 9 hole rotated by 45° (left) and 135° (right)

4.2. Comparison between the Newton Method with Projected Hessian and the Modified Hessian

Let us now look how Newton solver performs based on the Hessian type.

In terms of the untangling time, both Hessians are practically on par with each other. For all the benchmark meshes, the modified Hessian is just a little bit faster than the projected Hessian, with the only exception being the 100 by 100 grid mesh with a 49 by 49 hole rotated by 135°, and even that 0.5 seconds difference in time is inconsiderable (see Fig. 12).

What's more interesting, though, is the convergence time comparison. For all meshes from the benchmark, the convergence is to some extent faster with the modified Hessian,

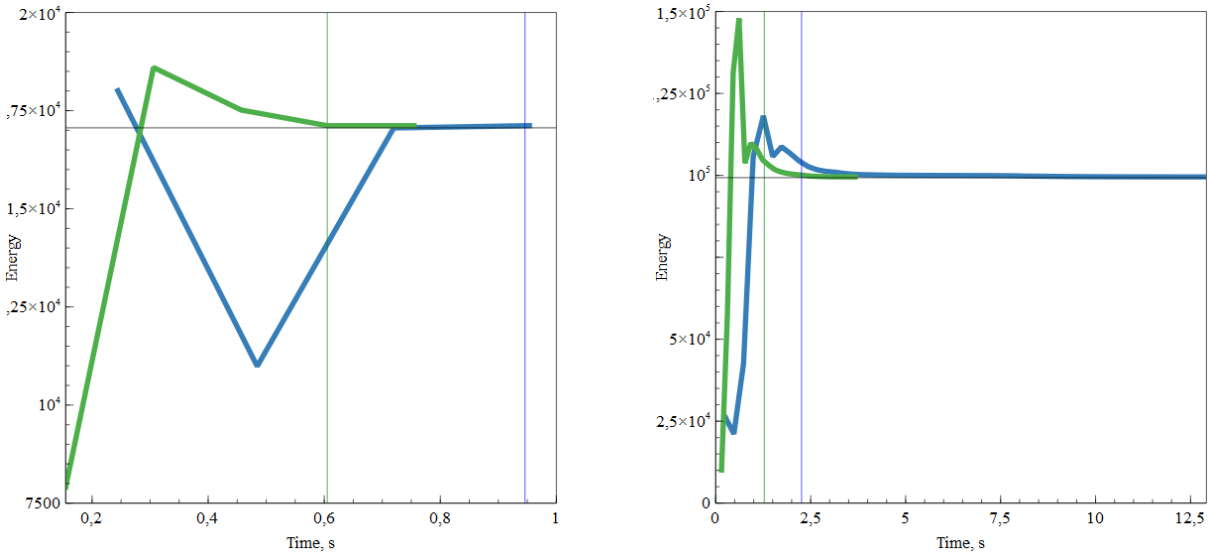


Figure 11: Comparison of Newton with projected and modified Hessian for a 50 by 50 mesh with a 29 by 29 hole rotated by 45° (left) and 135° (right)

Newton with Projected Hessian
Newton with Modified Hessian

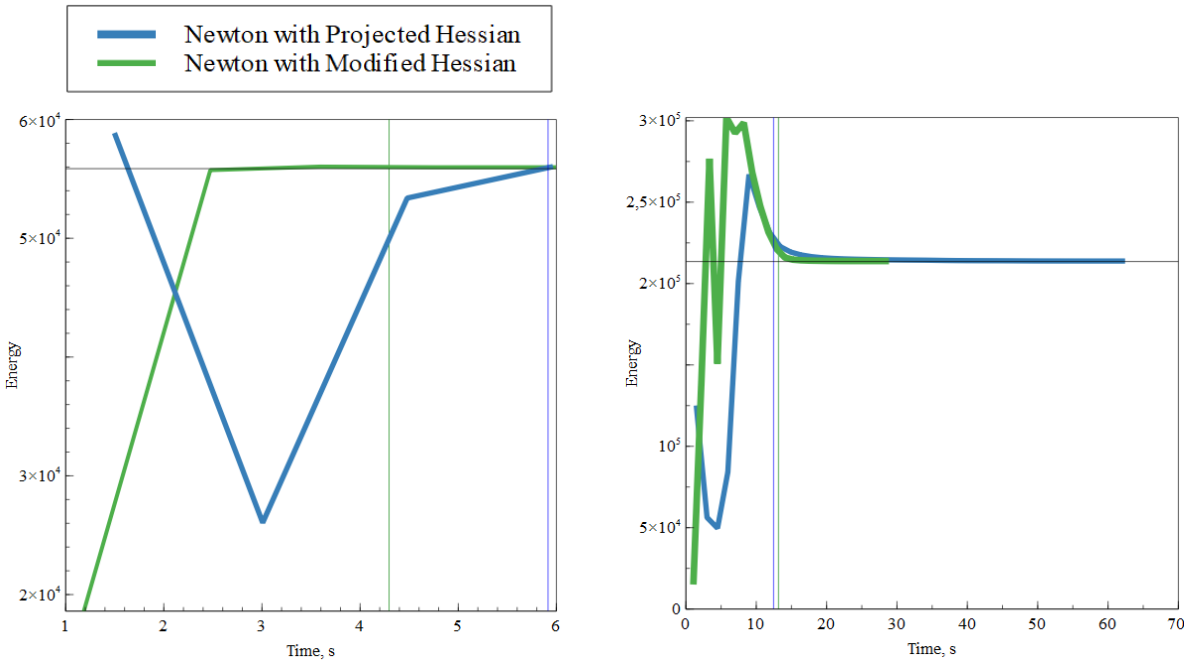


Figure 12: Comparison of Newton with projected and modified Hessian for a 100 by 100 mesh with a 49 by 49 hole rotated by 45° (left) and 135° (right)

but those are mostly slight differences. On the other hand, as we increase the mesh size, the hole size and the rotation angle, considerable differences in terms of performance become apparent with the modified Hessian being drastically faster. Even in the aforementioned scenario where the mesh was untangled faster with the projected Hessian, it took for the modified Hessian 27.229 seconds to converge, while the projected Hessian converged more than twice as long in 62.124 seconds. (Fig.12, right)

The relative final energy differences are fairly small to the extent of inconsiderable in each scenario and grow slightly bigger with more rotation applied, i.e. with a higher foldover rate, and even this growth becomes smaller with increased mesh size.

The only instance where the energy difference is any significant is for a 20 by 20 grid mesh with a 9 by 9 hole rotated by 135, with values 8801.1 and 8683.95 for the projected and modified Hessian respectively, with a relative difference of 1.33%. (Fig.10) Already for a 50 by 50 mesh with a 29 by 29 hole rotated by 135 this difference is just 0.002%. (Fig.11)

Notably, the times for computing the projected Hessian and the modified Hessian are more or less on par, with the calculation of the modified Hessian being just slightly faster, so that's not really something *major* speaking in favor of the latter in the context of the overall performance.

With that being said, the Newton solver with the modified Hessian is the preferred method when working with lambda set to zero, since it outmatches the Newton solver with projected Hessian in terms of convergence in every scenario, and even more so when working with big meshes with a high foldover rate. In the next paragraph, among other we'll look at the circumstances under which the modified Hessian might not be the preferred choice anymore. Some other testing scenarios are listed on Figure 13.

Mesh			Time to untangle, s		Time to converge, s	
Mesh Width, vertices	Hole Width, vertices	Rotation angle	N+mH	N+pH	N+mH	N+pH
20	5	135°	0.11	0.15	0.14	0.15
	9		0.12	0.20	0.19	0.55
	11		0.21	0.40	0.26	0.62
50	15	45°	0.48	0.73	0.48	0.73
	24	90°	0.75	1.15	1.12	1.15
	29	135°	1.53	2.38	3.66	12.51
100	59	45°	3.99	6.90	4.78	10.30
		90°	6.52	9.19	9.75	22.80
		135°	11.89	14.99	30.57	50.31

Figure 13: A table presenting nine scenarios of comparisons of the two Hessian modification approaches in terms of finding a valid solution and convergence.

4.3. Comparison for different Lambda Values

Let us now look how the untangler's efficiency varies depending on the lambda parameter. For this evaluation, let us focus on a mesh outside of the benchmark. The initial motivation for that was to present to the reader a very clear and self-explanatory result, devoid of any visual artifacts occurring when adapting a zoomed-out 100 by 100 or 50 by 50 mesh. A 20 by 20 mesh would also be insufficient to draw any performance-regarding conclusions due to its relative simplicity. The 40 by 40 mesh with a 10 by 10 hole rotated by 135° was chosen empirically (Fig.14), and already with it a potential problem emerged with lambda set to 10'000, which I'll soon come back to.

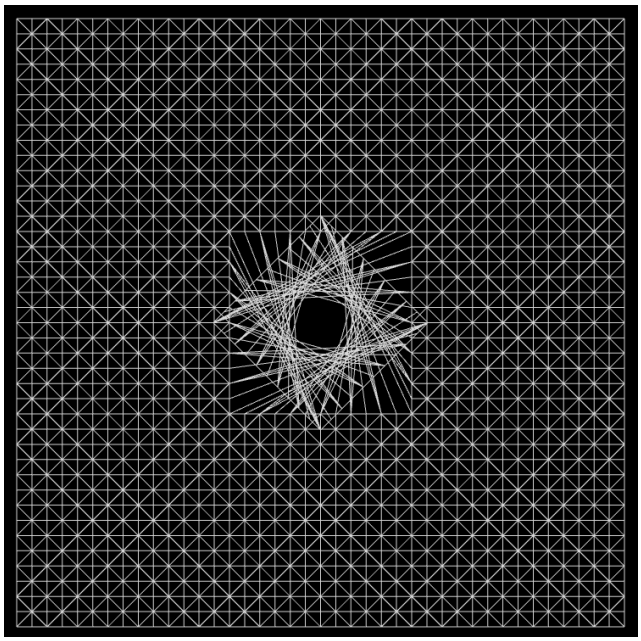


Figure 14: Input problem: a 40 by 40 mesh with a 10 by 10 square hole rotated by 135°

With lambda set to 0, the area-preserving function ***g*** of the energy is neglected, so only the ***f*** function will have any effect on the energy and thus the untangler will try to keep all angles of every face as close to a target shape as possible, which is called angle or shape preservation. With our target shape being an equilateral triangle, all angles will be kept as close to 60° as possible. In previous evaluations, this scenario was chosen to eliminate any impact of lambda on the general performance of the solvers, and so, the same conclusions apply here: L-BFGS is having a compromised performance with a higher foldover rate, while the Newton method offers a vastly superior performance with modified Hessian only adding to that.

For the mesh in focus, however, the performance of L-BFGS has not yet revealed its potential for a slow convergence, and in fact, L-BFGS is still almost a second faster than both Newton-related approaches. Because the final energy difference is neglectable as demonstrated previously, we can safely expect the mesh to untangle identically with all three methods. (Fig. 15)

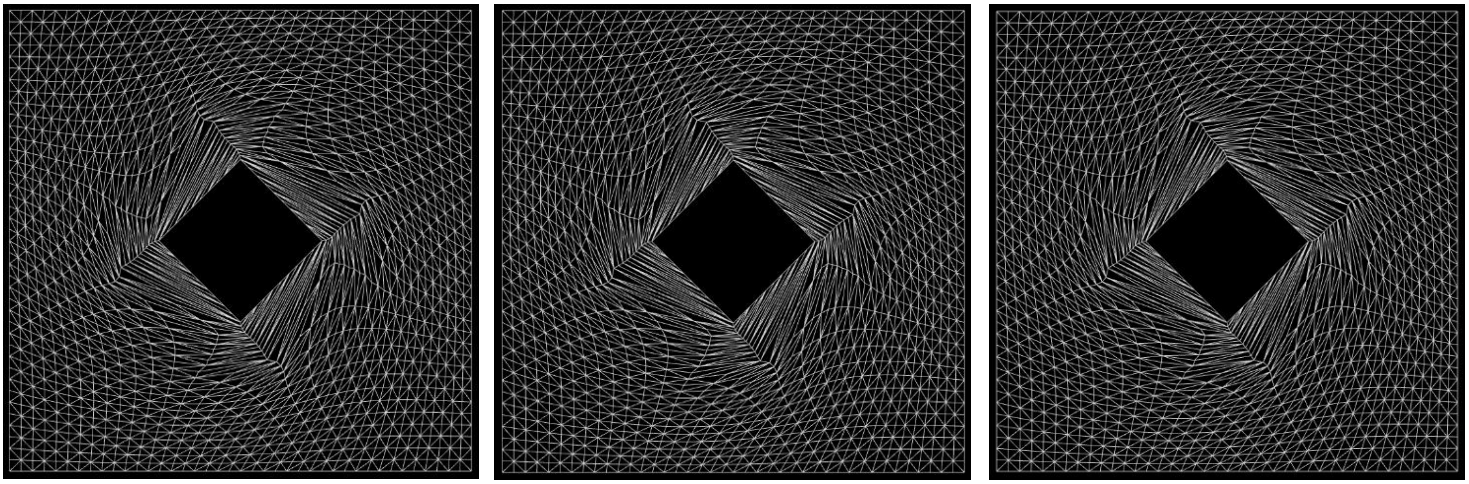


Figure 15: Input problem untangled with L-BFGS, N+pH and N+mH respectively while the lambda is set to 0.

Once we aim for the area preservation, the situation changes drastically. As proposed in the reference paper, we set lambda to 10'000 to neglect the impact of $\boldsymbol{f}$ and expect the performance and the final energy to be different to some extent. For the same mesh, L-BFGS now finds a valid result in 129.39 seconds and converges in 129.70 s as opposed to 0.154 s and 0.351 s respectively with lambda set to 0. This is a huge blow at the performance, but with the Newton method it's significantly better. For N+pH, it takes just 0.95 seconds and only three iterations to untangle and converge. The visual result for both solvers is significantly different, but both show the tendency of the closest to the hole triangles being stretched-out, more so with L-BFGS. (Fig. 16)

When working with N+mH, however, a new problematic trend emerges. The distortion caused by the stretched-out triangles is now all over the place, and, while the performance seems still good and on par with N+pH, the check for the area positivity of all triangles is failed, and therefore we shouldn't even talk about the convergence under these circumstances. (Fig. 16, right)

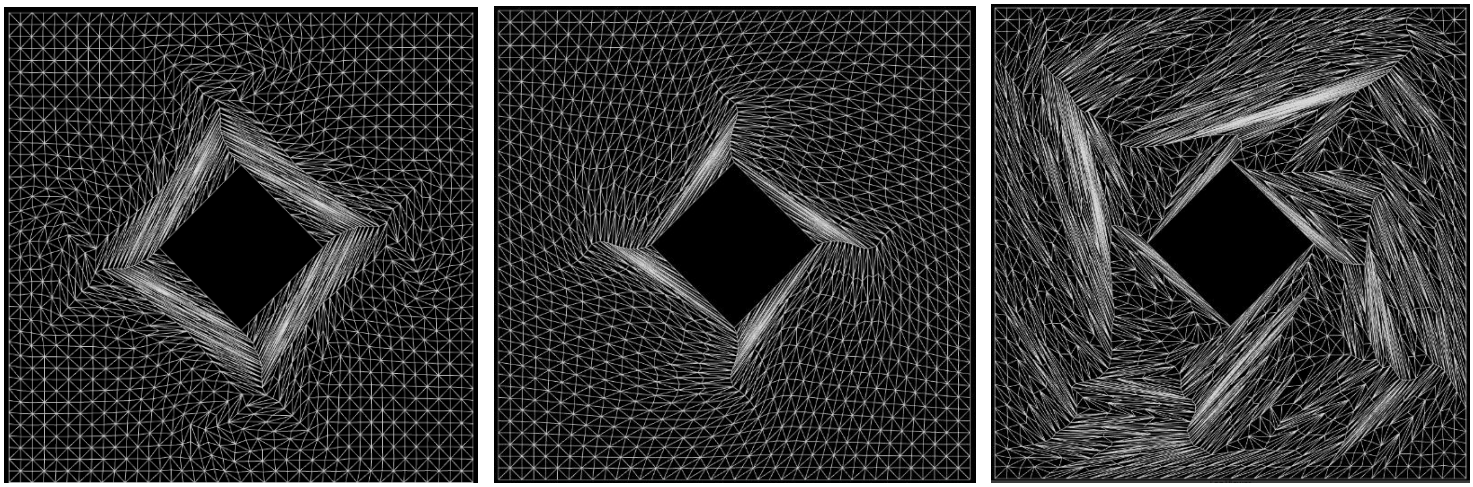


Figure 16: Input problem untangled with L-BFGS, N+pH and N+mH respectively while the lambda is set to 10'000.

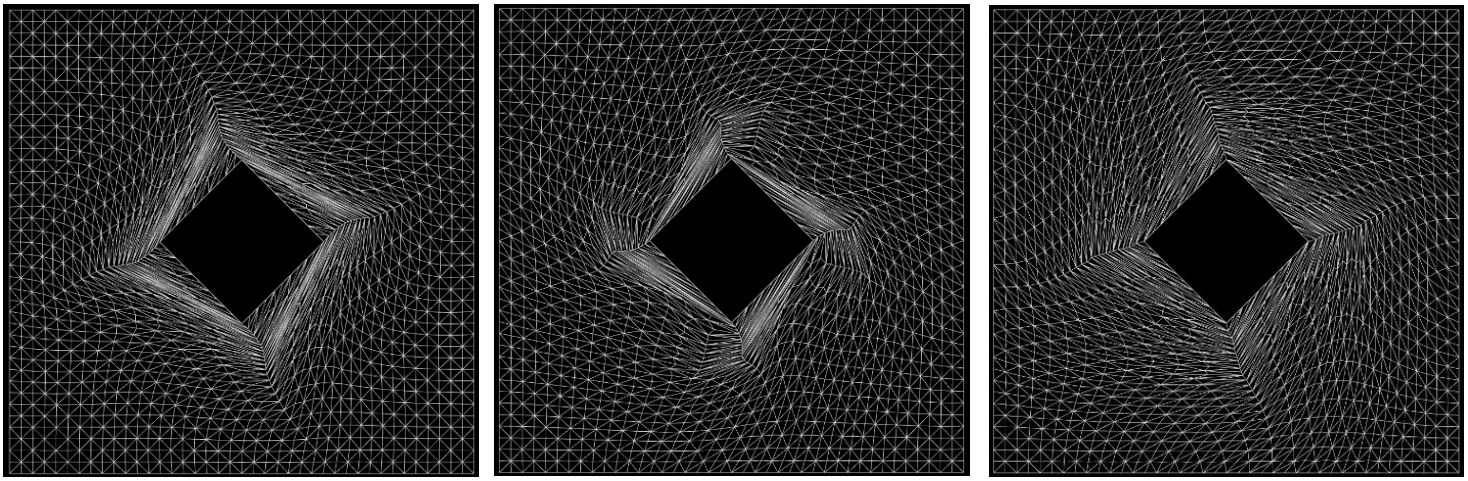


Figure 17: Input problem untangled with L-BFGS, N+pH and N+mH respectively while the lambda is set to 1.

For the benchmark meshes, the problem then emerges for L-BFGS and N+pH, too. For N+mH, it impacts all 50 by 50 meshes; from now on, it affects all meshes with a rotation bigger than 45° even when using L-BFGS; for N+pH, just a small fraction of bigger meshes with rotations 45° and 90° is untangled properly, the biggest of which is a 100 by 100 mesh with a 32 by 32 hole. For the properly untangled meshes, there is just a minor relative final energy difference, with the only exception being a 20 by 20 grid mesh with a 9 by 9 hole rotated by 135°, with a relative final energy difference of 1.5432% between L-BFGS and N+pH. Notably, for numerous meshes which seemingly properly untangled with N+mH, the minimum Jacobian determinant at the end is always a small negative value, such as for a 20 by 20 grid mesh with a 11 by 11 hole rotated by 90° with min.det. value of -0.13. But given that those meshes passed the triangle areas positivity check, I suspect that those are just the numerical precision issues. The whole benchmark now takes 8404 seconds, in large part due to the L-BFGS struggles.

Let us now evaluate the shape/area-tradeoff scenario with lambda set to 1. In terms of performance, it's slightly faster than with lambda set to 0, and the trend for L-BFGS becoming slower for bigger meshes and staying faster for smaller meshes remains true. Similar to the previous scenario, the triangles closer to the hole are stretched-out, but to a much lesser extent. Not only is N+mH now devoid of previous distortions, its visual result is now the best out of three methods in terms of the stretched-out triangles. (Fig.17)

With lambda set to 1, there are now just two meshes with the problem of failing the triangle areas positivity check, both fail only with L-BFGS – 100 by 100 with a 32 by 32 hole rotated by 90° and 135° respectively, both struggling for 2800 seconds each. N+mH still has the issue of minimum Jacobian determinants having small negative values for all the meshes in the benchmark, but once again I connect it to numerical precision issues given the successful positive triangle area checks. The whole benchmark took 6316 seconds this time, delivering in general a good performance except for just the two aforementioned problems taking 5600 seconds.

Summing up, the untangler is at its best in terms of stability when the lambda is set to 0. By setting it to 1, we still see good results, albeit there's a couple exceptions, and the performance is still satisfactory. By setting the lambda to 10'000, however, we're entering the zone of high instability, be it due to a compromised performance, visually unsatisfactory results or computation failures. Even though it mostly concerns L-BFGS and N+mH, at some point N+pH is affected, too.

4.4. The Property of Independence from Initialization

Let us now evaluate the claim for the initialization-independence, i.e. check whether the energy converges to largely the same minima between the given three initializations of a mesh.

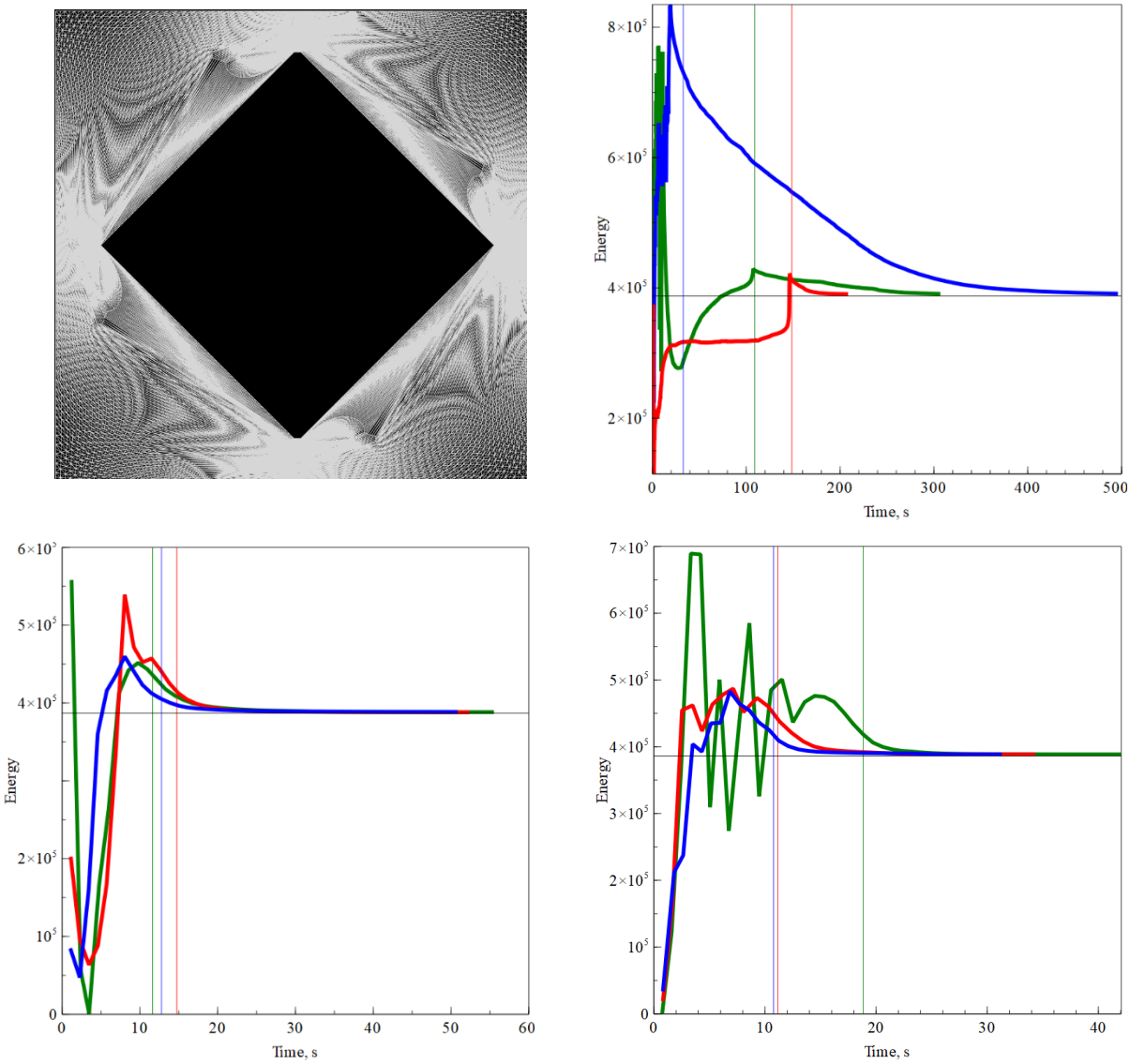


Figure 18: Three initializations of a 100 by 100 mesh (untangled result top left) untangled with all three methods: L-BFGS at the top right, N+pH at the bottom left and N+mH at the bottom right. Horizontal axes indicate the final energy value while the colored vertical axis indicate finding a valid solution, i.e. untangling.

Looking at the biggest problem from our benchmark (Fig. 18), we see how for all three initializations, the energy converges to nearly the same values for each of the untangling methods used. For L-BFGS (top graph), the relative final energy differences to the grid initialization are 0.1671% for the center and 0.0944% for the outer circle initialization respectively - certainly not a considerable difference in this case. For both Newton-based methods, the final energy differences are even smaller: for the projected Hessian, we have 0.1082% relative difference to the grid initialization 0.0546%, and for the modified Hessian, 0.0025% and 0.0195%.

Notably, for a fixed mesh size, the relative energy difference increases with a bigger foldover rate, i.e. with more rotation applied to the hole in our case. Even for a smaller problem, such as a 20 by 20 grid mesh with a 9 by 9 hole and the rotation by 135°, the relative final energy difference between the grid and the center initialization is 1.26% for L-BFGS and 1.33% for the Newton with projected Hessian with no energy differences between all three initializations for the Newton with modified Hessian, as well as no

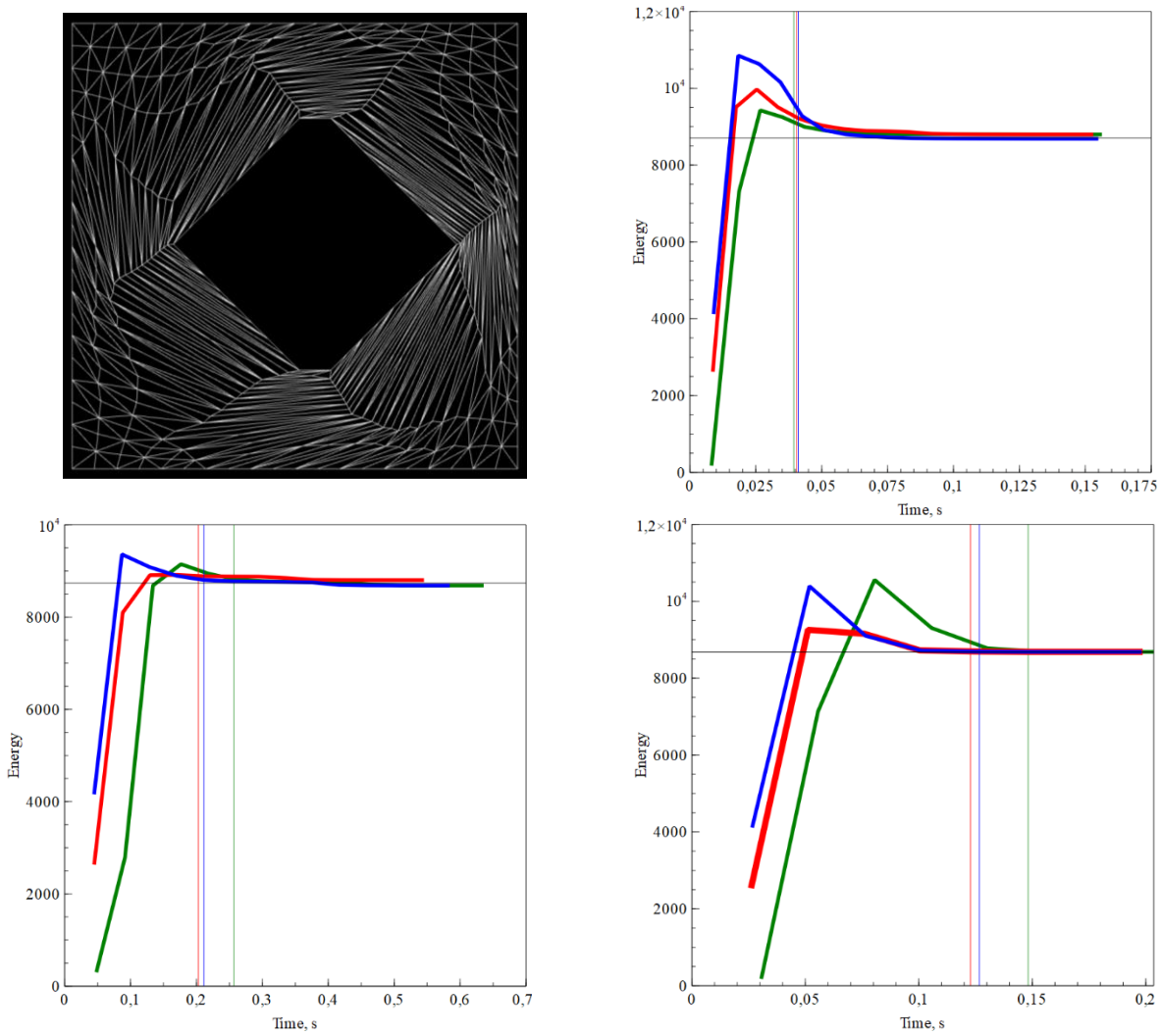


Figure 19: Three initializations of a 20 by 20 mesh (untangled result top left) untangled with all three methods: L-BFGS at the top right, N+pH at the bottom left and N+pH at the bottom right. Horizontal axes indicate the final energy value while the colored vertical axis indicate finding a valid solution, i.e. untangling.

considerable energy differences for this mesh with smaller rotations applied for all three untangling methods. This trend stops at a 50 by 50 mesh with a 29 by 29 hole rotated by 135°, where with L-BFGS the relative final energy difference is 1.7089% between the grid and both other initializations.

For a 20 by 20 grid mesh with a 11 by 11 hole and the rotation by 135°, we get a rather exceptional case: while the relative energy difference is neglectable for L-BFGS and the Newton with projected Hessian, for the Newton with modified Hessian there is an alarming difference of 3.4292% between the grid and both other initializations. This shows not only the biggest relative difference of the final energy throughout the whole benchmark set, but also the only instance when a more considerable relative difference occurs only for the Newton with modified Hessian, while being utterly neglectable for the other two solving methods. Judging from Fig. 20, we even see the considerable visual difference in the untangling result between the grid and the outer circle initializations. It currently is by far the most considerable final energy difference throughout the whole evaluation and the only instance where the independence from initialization doesn't really hold true.

The few scenarios presented above demonstrate the only instances where the final energy differs by more than one percent between different initializations. For the rest of the cases, only performance varies significantly, mostly so with L-BFGS and to a lesser extent with both Newton-based methods. Other than the aforementioned exception, Newton with modified Hessian provides the least variable result in most cases.

Summing up, for the majority of the benchmark data, there isn't a considerable final energy difference between all three initializations. Even though the energy is potentially more variable when applying a bigger rotation, we see neglectable energy differences once we go to the biggest 100 by 100 meshes of our benchmark. In future work, it would

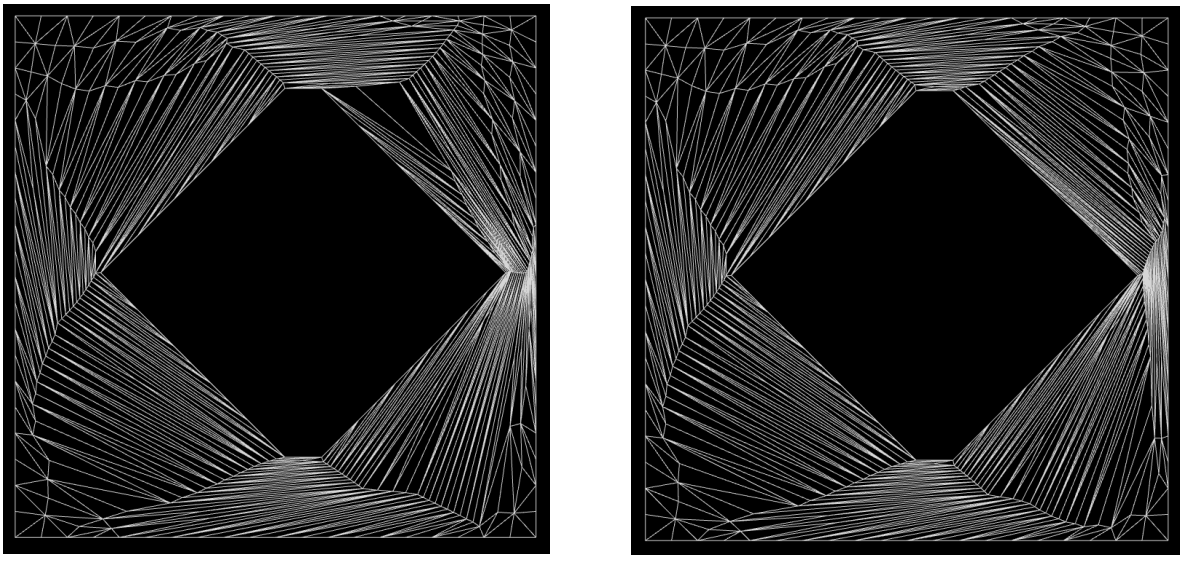


Figure 20: A 20 by 20 mesh with a 11 by 11 hole rotated by 135° untangled with Newton with modified Hessian. On the left is the untangled grid initialization, and on the right – the outer circle initialization. The relative final energy difference is about 3.4%, and the visual difference is apparent in the upper right quadrant of the mapping.

be worth checking the relative energy difference for some more small meshes with high foldover rate. As it is now, the property of independence from initialization claimed in the reference paper is confirmed with the exception of the aforementioned case.

4.5. TinyAD vs. Manual Gradient

If the TinyAD gradient was any better than the manually implemented gradient, it would be very interesting to see whether we could improve the lacking performance of L-BFGS with bigger meshes to any extent. However, that's not the case.

Starting with smaller problems, L-BFGS with the TinyAD gradient untangles and converges about two times slower than with the manually implemented gradient. (Fig.21) For a 50 by 50 grid mesh with a 29 by 29 hole rotated by 135°, the trend softens up significantly, but the convergence with the automatic gradient is still 2 seconds slower, even though a valid solution was found 7 seconds faster. The second exception is a 100 by 100 grid mesh with a 59 by 59 hole rotated by 135°, although L-BFGS is still a whole minute faster at convergence.

When looking only at the gradient initialization outside of the usage with L-BFGS, the manual gradient also delivers a performance two times faster than the automatic gradient for all the meshes in the benchmark. (Fig.21)

Summing up, in no scenario did the automatic gradient deliver a better performance when used with L-BFGS, therefore we should stick to the manual gradient when using L-BFGS.

Mesh			Time to untangle, s		Time to converge, s		Time to initialize, s	
Mesh Width, vertices	Hole Width, vertices	Rotation angle	Manual	TinyAD	Manual	TinyAD	Manual	TinyAD
20	5	135°	0.028	0.055	0.028	0.055	7.8e-05	0.000174
	9		0.04	0.078	0.15	0.29	8e-05	0.00016
	11		0.13	0.25	1.41	3.00	5.8e-05	0.000137
50	15	45°	0.11	0.22	0.11	0.22	0.000475	0.001049
	24	90°	0.24	0.37	1.00	1.87	0.000402	0.000884
	29	135°	18.36	10.46	24.54	25.26	0.000347	0.00076
100	59	45°	2.10	4.14	12.84	23.71	0.0013	0.0030
		90°	5.55	10.57	33.99	64.21		
		135°	144.27	155.56	202.85	271.94		

Figure 21: A table presenting nine scenarios of comparisons of the two gradients in terms of initialization, finding a valid solution and convergence.

5. Discussion

For this thesis, I reproduced the approach at solving the local injectivity problem and evaluated it with numerous respects. The focus of this implementation was on 2D triangular meshes.

[Garanzha 2021] proposed two variations for a solver method, and additionally to that I implemented a third variation – the Newton method with projected Hessian. Depending on a scenario, the preference changes from one variation to another.

Going for the shape preservation or settling for a shape/area tradeoff, the situations are similar. For smaller meshes with a small foldover rate, L-BFGS provides a slightly better performance, but with increasing the mesh size and/or the foldover rate, its convergence becomes its weak point, and for the biggest meshes, even finding a valid solution becomes problematically slow. Both variations of the Newton solver, however, demonstrate a much better performance for bigger meshes without showing any considerable weaknesses for smaller meshes. For all meshes from the benchmark, N+mH provided a better performance than N+pH, more so for the bigger meshes. Therefore, the Newton method with modified Hessian is definitely a preferred solver under these conditions.

However, going for the area preservation, the untangler isn't quite as stable anymore. L-BFGS suffers even more from a compromised performance, and the Newton method with modified Hessian starts failing to untangle early on, in addition to applying high levels of distortion to the mapping result. Newton with projected Hessian is significantly more reliable in that aspect, but even it fails to untangle properly for numerous meshes from the benchmark.

The attempt to explore the opportunity of providing the TinyAD gradient for L-BFGS didn't deliver a positive result, since it only made the solver slower than when used with the manual gradient, and two times slower at that.

The claim for the initialization independence is confirmed by the example of three different initializations with which the solvers mostly converge to almost identical values for the majority of the benchmark meshes, however an exception was found with an alarming final energy difference.

6. Future Work

As a potential for future work, I see it interesting to use another epsilon update technique, proposed in the published version of the reference paper. While the early version proposed two other techniques, one of which was implemented for this report, both of those techniques were omitted in the subsequent version of the paper. The general intuition is that the latest technique may provide an even better performance.

The instability of the untangler with the lambda set to high values is truly problematic, and it would be worthwhile to dive deeper into the issues with the Newton method with modified Hessian. My initial intuition is that the precision of the triangle area positivity check is not enough anymore. For L-BFGS, however, I don't find it worthwhile to look into its compromised performance, since that part was expected due to increased energy values.

Although the initialization independence was mostly confirmed, some disquieting scenarios were found and we therefore should be careful not to overlook some problematic scenarios where the final energy difference might be considerable. It would be important to explore the state of this property for a more varied mesh benchmark set, especially for numerous small meshes.

Given that the manual gradient was much faster for L-BFGS, I wonder whether the Newton method could also at least slightly benefit from a manual gradient in terms of performance.

Finally, I would consider this a top-priority to perform a thorough evaluation in the same manner, but for the 3D case and/or with the quad-meshes.

References

- Vladimir Garanzha, Igor Kaporin, Liudmila Kudryavtseva, François Protais, Nicolas Ray, et al.. Foldover-free maps in 50 lines of code. ACM Transactions on Graphics, Association for Computing Machinery, 2021, Volume 40 (issue 4), Article No.102, pp 1-16. 10.1145/3450626.3459847. hal-03127350v2
- John M Ball. 1976. Convexity conditions and existence theorems in nonlinear elasticity. *Archive for rational mechanics and Analysis* 63, 4 (1976), 337–403.
- Xingyi Du, Noam Aigerman, Qingnan Zhou, Shahar Z. Kovalsky, Yajie Yan, Danny M. Kaufman, and Tao Ju. 2020. Lifting Simplices to Find Injectivity. ACM Trans. Graph. 39, 4, Article 120 (July 2020), 17 pages. https://doi.org/10.1145/3386569.3392484
- Yixuan Qiu, L-BFGS++, 2021. https://github.com/yixuan/LBFGSpp/
- Computer Graphics Group, RWTH Aachen University, CoMISo - Constrained Mixed-Integer Solver, 2010. https://www.graphics.rwth-aachen.de/software/comiso/
- The CGAL Project, The Computational Geometry Algorithms Library, 2021. https://www.cgal.org
- Jan Möbius, RWTH Aachen University, OpenFlipper 4.1, 2019. https://www.graphics.rwth-aachen.de/software/openflipper/