Technische Universiteit Delft
Faculteit Elektrotechniek, Wiskunde en Informatica
Delft Institute of Applied Mathematics

Meshfitting for the Level-set Method

(Nederlandse titel: Grid-aanpassing voor de Level-set Methode)

Verslag ten behoeve van het
Delft Institute of Applied Mathematics
als onderdeel ter verkrijging
van de graad van

BACHELOR OF SCIENCE
in
TECHNISCHE WISKUNDE

door

T.E. Wortelboer

Delft, Nederland
September 2018

BSc verslag TECHNISCHE WISKUNDE

"Meshfitting for the Level-set Method"

(Nederlandse titel: "Grid-aanpassing voor de Level-set Methode")

T.E. Wortelboer

Technische Universiteit Delft

Begeleider

Dr.ir. D. den Ouden-van der Horst

Overige commissieleden

Dr. D.C. Gijswijt

Dr. B. van den Dries

September, 2018

Delft

Meshfitting for the Level-set Method

Timo Wortelboer

September 27, 2018

Abstract

In applying the level-set method in the context of a P1 finite-element method, errors can be minimized by adjusting the mesh (usually made by a Delaunay triangulation) to the shape of the zero level-set curve, at the boundary between the two phases defined by the sign of the level-set function. The size of the different types of errors that occur depend on the goodness of fit to the zero level-set curve, on the skewness of the triangles and on the size of the triangles in the mesh. To fit the mesh some basic methods where designed by Javierre [4], Den Ouden [3] and Verbeek [2]. These methods all adjust the mesh only locally and add many edges to get the job done. The aim of this research was to find some better ways in which to adjust the mesh, hopefully without having to add new edges. First some quality measures were set up, namely the maximum skewness, average skewness, standard deviation of skewnesses, maximum size, minimum size, standard deviation of sizes, where the sizes and skewnesses respectively are taken over all the triangles in the (two-dimensional) mesh. Then the existing 'cut' method was analysed in depth and an extension called the 'flip' method was added which improved the quality already by not having to add new edges, improving the average skewness. Then a new approach was found, using shortest path algorithms to decide which points to move, and orthogonal projection to the zero level-set curve to move them. A second method was designed which used a physical model, modelling every non-fitted non-boundary edge as a spring, in order to relax the rest of the mesh. These last two methods combined improved the quality of the created meshes greatly, with its only downside being its more limited applicability (in terms of zero level-set curve topologies).

Contents

Chapter 1

Introduction

The level-set method is used for a large variety of problems, from the melting of ice to the modelling of different phases of steel, basically anywhere where different phases of a substance are capable of changing to the other phase during the process that one is investigating. To apply this method to a two-dimensional problem (using a P1 finite-element method), a triangular grid is used, which preferably fits the interface between the different phases well, since this results in smaller errors during the application of the level-set method. The subject of this thesis is the adjustment of a triangular mesh on a two-dimensional square to a random phase-interface curve.

The subject of adjusting a triangular mesh to a curve has previously been investigated by Javierre [4], who did not actually adjust the mesh but instead chose to change the curve itself slightly when possible. Then Den Ouden [3] moved some points to the interface line, when the distance to be moved was below some threshold, otherwise adding new edges. Lastly Verbeek [2] improved the way that edges were added and also did a statistical analysis to find the optimal threshold for moving a point instead of adding edges.

This research has focussed solely on the mesh adjustment itself. The existing techniques are analysed and compared using 5 different test-cases, and using 6 different quality measures. Then a new technique was designed using a shortest-path method along with a separate mesh-relaxation method and compared to the existing techniques.

Chapter 2 introduces some background material for this research, including terminology that will be used throughout this report. Chapter 3 sets up some measures and methods to assess the quality of a meshfitting method. Chapter 4 analyses some existing methods and simple extensions of those. Chapters 5 and 6 together introduce a new method of adjusting the mesh, where Chapter 5 uses a shortest-path algorithm to fit the mesh to a curve, and Chapter 6 introduces a mesh-relaxation technique to restore the quality of the mesh afterwards. Chapter 7 assesses the quality of the different methods and provides results, which are discussed further in Chapter 8 where also some ideas for further research are presented. In the appendices some pictures of meshes are presented showing the workings and results of the different methods in the 5 test-cases.

Chapter 2

Background material

This chapter provides some background material on the techniques for which the meshes are used. It is assumed that in the end the mesh is used in an application of the finite-element method, which is introduced in the first section. The second section introduces the level-set method, which can be used in combination with the finite-element method, to be able to cope with interfaces between different phases of a material. The last section introduces some much-needed terminology describing mesh characteristics.

2.1 The finite-element method

The finite-element method is the method for which the mesh will in the end be used. It is a numerical method that approximates solutions to complicated (partial) differential equations by describing the solution function as a linear combination of certain basis functions. The solution can then be found by solving a system of linear equations instead of the more complicated (and often unsolvable) differential equations.

In this research we are interested in two-dimensional problems, where the solution is a function of two (usually spatial) variables. The method uses a regular triangular mesh, where the solution function will be approximated by its values on the mesh points. A P1 finite-element method is assumed, meaning that the basis functions are chosen to be the piecewise-linear functions that equal one on one point of the mesh and zero in all others, see Figure 2.1. It is then straightforward to express the approximation to the solution function as a linear combination of these basis functions.

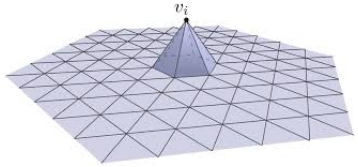

Figure 2.1: A typical basis function for the P1 finite-element method.
Retrieved from [14].

The important part to notice here is that the accuracy of the finite-element method depends on the regularity of the mesh on which it is applied. This means that the triangles of the mesh should be of roughly equal size, and should not be very skewed. This is why the quality measures to be described in Sections 3.1 and 3.2 are focussed on size and skewness.

2.2 The level-set method

The level-set method is a method which can be used in combination with the finite-element method, extending its usability to (more-dimensional) situations where there are different phases of a substance to model, with different partial differential equations in each phase. This means that there has to be some way to track which part of the mesh belongs to which phase. In order to do this, the level-set function $f(x, y)$ was introduced by Sethian [1]. It is a function that differentiates between the phases by having a different sign in each phase, so $\Omega_1 = \{(x, y) : f(x, y) < 0\}$ describes the first phase, and $D \setminus \Omega_1 = \Omega_2 = \{(x, y) : f(x, y) \geq 0\}$ then describes the second phase within the full domain D. Extensions to more than two phases are possible, see [2] for example, but will not be considered here. Figure 2.2 shows the situation.

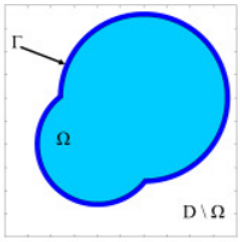

Figure 2.2: Figure showing an example of the different regions of a level-set model. Retrieved from [13].

Of particular interest is the interface between the two phases, given by $\Gamma = \{(x, y) : f(x, y) = 0\}$, since along this boundary the phases change. This means that it is much preferred that there are no triangles in the mesh that lie in both regions Ω_1 and Ω_2 within the finite-element method, because this would induce some difficulties in finding the solution. (The differential equations are different for each phase.)

2.3 Some terminology and basics

Some new terminology will be needed to be able to talk more easily about the mesh in relation to the level-set function $f(x, y)$.

Firstly the mesh consists of **points** p, **edges** e and **elements** Δ, where the elements are the triangles of the mesh. Each element thus consists of three points and three edges, and each edge of two points. Edges of the mesh will sometimes be called **mesh edges** to emphasize the difference between them and other types of edges. The set of all points in the mesh is denoted with P, the

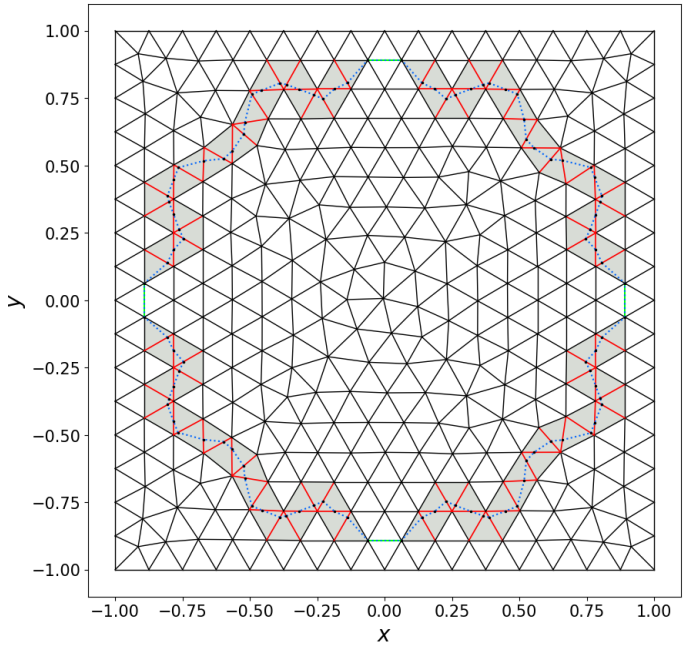

Figure 2.3: A mesh before adjustment showing the level-set edges of the zero level-set curve approximation in dotted blue, zero edges in red, zero elements in grey, interpolated zero level-set points in black, and (mesh) edges that are also level-set edges in green.

set of all edges with E and the set of all elements with T (where T stands for triangle).

For the following terms see Figure 2.3 for a depiction.

A **boundary edge** will be an edge which is contained in just one element, and the **boundary points** are the points contained in some boundary edge. If two neighbouring boundary edges are not in the same direction, then their common point will be called a **corner point** as well (in addition to being a boundary point).

In relation to the mesh we will always talk about the **zero level-set curve** C_z instead of the level-set function. The zero level-set curve is the curve where the level-set function equals zero, given by $\{(x, y) : f(x, y) = 0\}$. (When the method is applied to modelling different phases of a material, this is the interface curve of the different phases.)

When we talk about the level-set function we will almost always implicitly mean the piecewise-linear approximation to the level-set function induced by the mesh, see also the figures in Section 3.4. This is the function that equals the level-set function on the points of the mesh, and equals in each element the linear interpolation between the values in the points of this element.

The edges of the (also piecewise-linear) zero level-set curve corresponding to this approximation are called **level-set edges**. Before adjusting the mesh, each of these edges does not cross the boundary of an element, and touches the boundary in at least two places, as can be seen in Figure 2.3.

When the zero level-set curve C_z crosses a mesh edge e in its interior, so

when $\{C_z \cap \mathring{e}\} \neq \emptyset$, $f(p_1) \neq 0$ and $f(p_2) \neq 0$ with $p_1, p_2 \in e$, this edge is called a **zero edge** e_z. This is the case when $f(p_1)f(p_2) < 0$, so exactly when the level-set function changes sign across this edge. When the zero level-set curve C_z crosses[1] a point p, so when $p \in C_z$, this point is called a **zero point** p_z. The set of zero edges will be denoted by E_z. When the zero level-set curve crosses two points of an edge, such that this edge is correctly fitted to the zero level-set curve, then this edge is called a level-set edge as well as a (mesh) edge.

The piecewise-linear zero level-set curve C_z crosses the zero-edges e_z of the mesh in points that are called **interpolated zero level-set points** p_{interp}, so $p_{\text{interp}} = C_z \cap e_z$ for some e_z, since these can be found by linear interpolation from the values of the level-set function on the two endpoints of the zero edge on which the interpolated zero level-set point lies, which are of different sign.

Lastly there are the **zero elements**, which are those triangles which have a zero edge as one of its edges.

[1] Or rather when it comes closer to this point than some pre-defined small threshold.

Chapter 3

Quality of the mesh

To be able to compare different methods to fit the mesh it is necessary to have a number of ways to quantify the quality of the mesh. Since the mesh is meant to be used with a finite-element method, the quality can be defined along the lines of the errors that occur in the finite-element method, meaning that it can be based on the element skewnesses and sizes.
Since at each step of the finite-element method the level-set functions changes, the mesh needs to be adjusted at each step as well to the new zero level-set curve. Because of this the mesh-fitting process is repeated often, making the computational intensity of some interest as well, although it is not the focus of this research project.
To keep the skewness small, it is important to try to keep the angles of the triangles of the mesh as close to $\frac{\pi}{3}$ ($= 60°$) as possible. This will be quantified in Section 3.1.
Secondly it is also important to maintain similar sizes of the triangles. A measure that quantifies this is described in Section 3.2.
Thirdly also the goodness of the fit of the new mesh to the level-set curve is important to consider, which is briefly touched upon in Section 3.3.
The last section of this chapter describes the functions that have been used as test cases for the meshfitting methods.

3.1 Mesh skewness

The skewness of a triangle roughly corresponds with how much it differs from an equilateral triangle [2][1]. This can be quantified by looking how much the angles of the triangle differ from the $\frac{\pi}{3}$ ($= 60°$) angles of the equilateral triangle, see Figure 3.1 [5]. A good place to start seems to be to look at the values of the largest and the smallest angle of the triangle.

One would like to have a number between 0 and 1 for the skewness of a triangle, where 0 represents the best possible triangle and 1 represents the worst possible triangle. To achieve this, the difference between the smallest angle and the ideal value of $\frac{\pi}{3}$ is first normalized and then compared to the corresponding value of the largest angle. The largest value of these two is then a good measure

[1] The discussion on mesh skewness can be found in [2, Technical Details, p. 29].

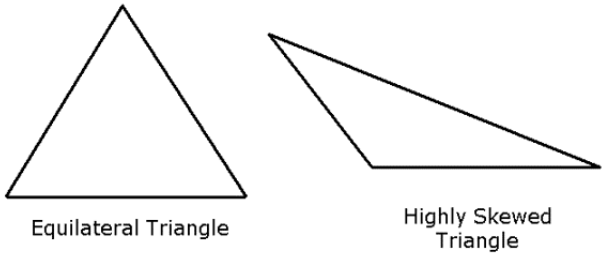

Figure 3.1: Skewness of a triangle, retrieved from [5].

of skewness. This gives the following formula for the skewness of a triangle Δ:

$$\text{Skew}(\Delta) = \max\left\{\frac{\theta_{\max} - \frac{\pi}{3}}{\pi - \frac{\pi}{3}}, \frac{\frac{\pi}{3} - \theta_{\min}}{\frac{\pi}{3}}\right\}, \tag{3.1}$$

where $\theta_{\min}$ is the smallest angle of the triangle, and $\theta_{\max}$ is the biggest angle. This is called the normalized equiangular skewness, and can be generalized to also provide a skewness measure of other types of meshes. For a mesh with quadrilateral elements, for example, one only has to change each $\frac{\pi}{3}$ to $\frac{\pi}{4}$ (assuming rectangles are preferred). This skewness measure returns a number between 0 and 1, where 0 corresponds to an equilateral triangle and 1 corresponds to a 'triangle' with angles (in radians) equal to $(\pi, 0, 0)$.

The following observation simplifies the measure of the skewness of a triangle.

Theorem 3.1

For triangles the skewness depends only on the minimum angle; more explicitly the skewness of a triangle Δ equals

$$\text{Skew}(\Delta) = \frac{\frac{\pi}{3} - \theta_{\min}}{\frac{\pi}{3}} = 1 - \frac{3\theta_{\min}}{\pi},$$

where $\theta_{\min}$ is the smallest angle of the triangle.

Proof 3.1

Since the sum of the angles of any triangle equals π, we have that $2\theta_{\min} + \theta_{\max} \leq \pi$ and

$$\theta_{\min} \leq \frac{\pi}{2} - \frac{\theta_{\max}}{2}.$$

Substituting this in the skewness measure $\frac{\frac{\pi}{3} - \theta_{\min}}{\frac{\pi}{3}}$ we have

$$\frac{\frac{\pi}{3} - \theta_{\min}}{\frac{\pi}{3}} \geq \frac{\frac{\pi}{3} - (\frac{\pi}{2} - \frac{\theta_{\max}}{2})}{\frac{\pi}{3}} = \frac{\frac{\theta_{\max}}{2} - \frac{\pi}{6}}{\frac{\pi}{3}} = \frac{\theta_{\max} - \frac{\pi}{3}}{\pi - \frac{\pi}{3}},$$

which shows that $\frac{\frac{\pi}{3}-\theta_{\min}}{\frac{\pi}{3}}$ is always equal to or bigger than $\frac{\theta_{\max}-\frac{\pi}{3}}{\pi-\frac{\pi}{3}}$ and hence

$$\text{Skew}(\Delta) = \max\left\{\frac{\theta_{\max}-\frac{\pi}{3}}{\pi-\frac{\pi}{3}}, \frac{\frac{\pi}{3}-\theta_{\min}}{\frac{\pi}{3}}\right\} = \frac{\frac{\pi}{3}-\theta_{\min}}{\frac{\pi}{3}} = 1 - \frac{3\theta_{\min}}{\pi}.$$

■

To obtain a measure of the quality of the whole mesh, one can take the maximum, the average and the standard deviation of the skewnesses of the individual elements. Then one must still have a way to calculate the angles of the triangles of a mesh. Of a given mesh one usually just has the coördinates of the points available to calculate with, along with the triples of points that constitute a triangle. From these points the length of the sides can be easily calculated, and then the angles can be determined from the law of cosines[2].

Since the number of triangles in the mesh is equal to $\#T < \frac{2}{3}\#E$, these calculations give a complexity of $\mathcal{O}(\#E)$.[3] Apparently these calculations are not very intensive, such that one could in principle also use the skewness in the algorithm for adjusting the mesh.

In conclusion there are three measures of skewness for the whole mesh (with T being its set of triangles):

1. The maximum of the skewnesses of the individual triangles:

$$\text{Maxskew}(T) = \max\left\{\text{Skew}(\Delta) : \Delta \in T\right\}.$$

   For this measure smaller is better, $\text{Maxskew}(S) = 0$ would imply that none of the triangles is skewed and would be the ideal case.

2. The average of the skewnesses of the individual triangles:

$$\text{Avgskew}(T) = \frac{1}{\#T}\sum_{\Delta\in T}\text{Skew}(\Delta).$$

   In the same reasoning as with $\text{Maxskew}(T)$, $\text{Avgskew}(T) = 0$ would be the ideal value and smaller is again better.

3. The standard deviation[4] of the skewnesses of the individual triangles:

$$\text{Stdskew}(T) = \sqrt{\frac{1}{\#T}\sum_{\Delta\in T}\left|\,\text{Skew}(\Delta) - \text{Avgskew}(T)\,\right|^2}.$$

[2]For a triangle with sides a, b, c and angle γ opposite to edge c, the law of cosines states that $c^2 = a^2 + b^2 - 2ab\cos\gamma$.

[3]Note that each edge is part of at most two triangles, and each triangle has three edges as its perimeter.

[4]This is of course the exact value of the standard deviation, since all the $\text{Skew}(\Delta), \Delta \in T$ are known. If it is considered as a sample, then a finite population correction would be necessary to provide an unbiased estimator of the standard deviation. Luckily, the exactness of the calculation makes things a lot simpler. See for example Section 7.3 in [9].
As a sidenote: perhaps one could think of the five test-functions as a kind of sample from all possible level-set functions. In that case an estimate could be made of the expected standard deviation over all possible level-set functions. This presents difficulties, however, since the total amount of possible level-set functions is infinite, so the sample size may never be appropriate, and the chosen sample functions may never be truly randomly chosen.

The ideal value would be $\text{Stdskew}(T) = 0$ since this would imply that all the skewnesses are the same, implying that the mesh is very regular. So for this measure it again holds that smaller is better.

3.2 Triangle sizes

The second measure of quality to consider is the size of the triangles of the mesh. First one needs a way to easily calculate the area of a triangle Δ when one knows only the lengths of the edges, given by a, b, c respectively. The simplest method is to use Heron's formula, given by

$$\text{Size}(\Delta) = |\Delta| = \sqrt{p(p-a)(p-b)(p-c)}, \tag{3.2}$$

where p is equal to half the perimeter of the triangle, $p = (a+b+c)/2$. [6]

The overall measure of quality of the mesh can now be defined as the minimum, the maximum, and the standard deviation of the sizes of the individual triangles. Just as with the skewness measure in Section 3.1, the calculation of the area is also straightforward and not very computationally intensive, and with the same reasoning one gets a complexity of $\mathcal{O}(\#E)$, so the area too can in principle be used in the algorithm that adjusts the mesh.

In conclusion there are three additional measures of the quality of the whole mesh (with $\#T$ being its set of triangles):

1. The maximum of the sizes of the individual triangles:

   $$\text{Maxsize}(T) = \max\{\text{Size}(\Delta) : \Delta \in T\}.$$

   For this measure smaller is better since one wants to have the triangles be as equal in size as possible.

2. The minimum of the sizes of the individual triangles:

   $$\text{Minsize}(T) = \min\{\text{Size}(\Delta) : \Delta \in T\}.$$

   For this measure bigger is better following the same reasoning as for the maximum of the sizes.

3. The standard deviation[5] of the sizes of the individual triangles:

   $$\text{Stdsize}(T) = \sqrt{\frac{1}{\#T}\sum_{\Delta\in T} \big|\, \text{Size}(\Delta) - \text{Avgsize}(T) \,\big|^2},$$

   where $\text{Avgsize}(T)$ is defined as

   $$\text{Avgsize}(T) = \frac{1}{\#T}\sum_{\Delta\in T} \text{Size}(\Delta).$$

   For this measure the value 0 would be the best possible value, since it would imply that all the triangles are equal in size, so smaller is better.

[5] See also the footnote in Section 3.1 at the description of the $\text{Stdskew}(S)$ measure.

3.3 Goodness of fit to the level-set curve

The goal of this study is to adjust the mesh in such a way that it fits the level-set curve more closely (see also Section 2.2), so it is also a requirement that the method to adjust the mesh actually improves this. This can also be quantified, see for example [2]. In this research it has been chosen (for simplicity) to assess the goodness of fit by eye.

3.4 Test-functions

To test the performance of the meshfitting methods, five test-functions have been used, each with their own peculiar characteristics that could present difficulties for the meshfitting methods. In the next subsections graphs of each test-function are shown and the peculiarities are discussed (from the perspective of having the objective to design a method that only moves points and does not create new points). For all the functions a triangular mesh is assumed, on the square $\{(x, y) : x \in [-1, 1] \cap y \in [-1, 1]\}$, although the methods are in principle also suited for different mesh topologies. (This has not been tested.)

Some technical information: The mesh is generated using Delaunay triangulation (see [8]) with some fruitfully chosen points, such that the minimal angle is maximized in the grid (which corresponds to a grid of minimal skewness, see Section 3.1). This mesh will be used as the starting mesh for all the test-functions and meshfitting methods. An average edge length of $\frac{1}{16}$ was chosen, except for some figures which were more readable with an average edge length of $\frac{1}{8}$.

3.4.1 Cosine-times-sine function

The cosine-times-sine level-set test-function is defined as

$$f(x, y) = \cos\left(\frac{3\pi x}{2}\right) \cdot \sin\left(\frac{\pi y}{2}\right).$$

In Figure 3.2 the approximation f^* of f on the triangular mesh is shown. The zero level-set curve is visible in the contour plot in Figure 3.3, as the solid horizontal and vertical lines, which is the set $\{(x, y) : f^*(x, y) = 0\}$, which should closely resemble $\{(x, y) : f(x, y) = 0\}$.

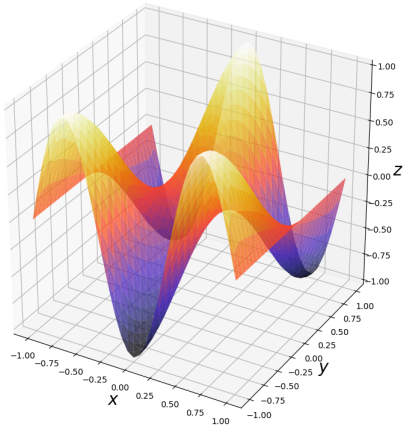

Figure 3.2: A 3d plot of the approximation f^* to the cosine-times-sine test-function.

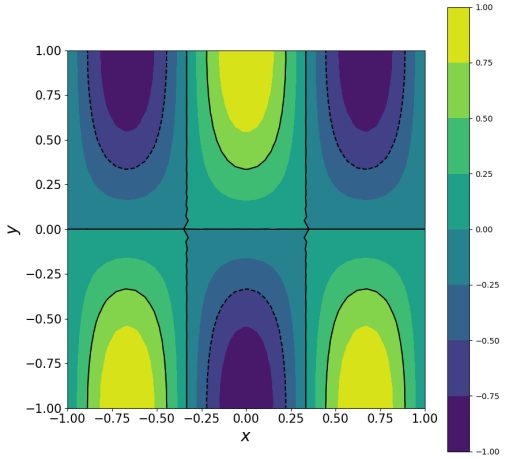

Figure 3.3: A contour plot of the approximation f^* to the cosine-times-sine test-function.

The difficult parts of fitting this level-set function will be detecting and handling the points of intersection where multiple zero level-set curves come together in the interior of the mesh, as well as the points where the zero level-set curves touch the boundary of the mesh. The contour plot in Figure 3.3 also shows a resolution problem near the points of intersection in the interior, where some wiggling about the vertical lines is visible. This shows how the errors in the approximation f^* to f can lead to errors in the calculated position of the zero level-set curve, since the zero level-set curve of f should be exactly vertical at these locations. (Because these are the sets of coordinates (x, y) for which $\cos(3\pi x/2) = 0$.)

3.4.2 Linear function

The linear level-set test-function is defined as

$$f(x, y) = 3x + 2y + 1.$$

In Figure 3.4 the approximation f^* of f on the triangular mesh is shown. The zero level-set curve is visible in the contour plot in Figure 3.5, as the solid line that goes through the point $(-1, 1)$, which is the set $\{(x, y) : f^*(x, y) = 0\}$, which now perfectly coincides with $\{(x, y) : f(x, y) = 0\}$ since f^* is a piecewise-linear approximation to f, which is already linear.

The only difficult parts of fitting this level-set function is in detecting that one of the intersections with the boundary is in a corner of the mesh, such that no other point should be allowed to move towards this intersection, and secondly the detection and handling of the other intersection with the boundary which includes choosing which boundary point of the mesh can best be moved towards the intersection.

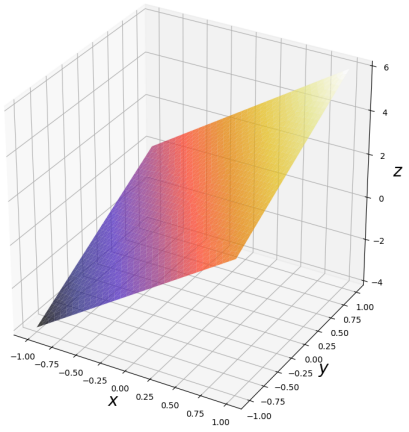

Figure 3.4: A 3d plot of the approximation f^* to the linear test-function.

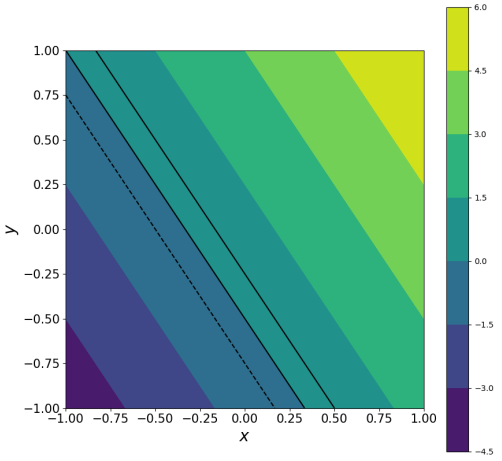

Figure 3.5: A contour plot of the approximation f^* to the linear test-function.

3.4.3 Circular function

The circular level-set test-function is defined as

$$f(x,y) = \left(\frac{x}{1.1}\right)^2 + \left(\frac{y}{1.3}\right)^2 - 1.$$

(This is actually an ellipsoidal, but it is almost circular so for simplicity it will still be referred to as the circular test-function.) In Figure 3.6 the approximation f^* of f on the triangular mesh is shown. The zero level-set curve is visible in the contour plot in Figure 3.7, as the four solid lines, which is the set $\{(x,y) : f^*(x,y) = 0\}$, providing a good approximation to $\{(x,y) : f(x,y) = 0\}$.

The difficult part of fitting this level-set function will be to handle the 8 intersections with the boundary and to make sure that the mesh is fitted to all four zero level-set curve parts. Aside from this, the angle which these lines make with the boundary can result in some difficulties in maintaining the quality of the mesh whilst fitting.

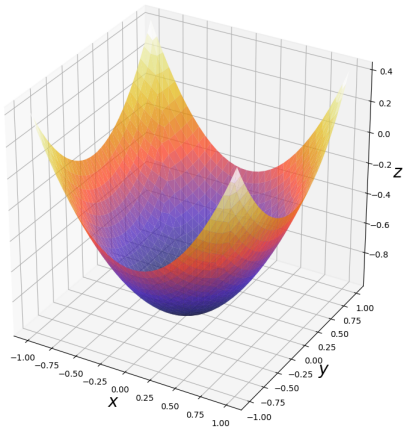

Figure 3.6: A 3d plot of the approximation f^* to the circular test-function.

3.4.4 Star function

The star level-set test-function is defined as[6]

$$f(x,y) = \frac{3}{5} + \frac{1}{4}\cos\left(6\arctan\left(\frac{y}{x}\right)\right)^2 - (x^2+y^2).$$

In Figure 3.8 the approximation f^* of f on the triangular mesh is shown. The zero level-set curve is visible in the contour plot in Figure 3.9, as the outer solid line, which is the set $\{(x,y) : f^*(x,y) = 0\}$, providing a good approximation to $\{(x,y) : f(x,y) = 0\}$.

The difficult parts of fitting this level-set function will be in fitting correctly the details of the oscillation of the cosine function, shown as the 'points' of

[6]Of course actually the continuous extension of this function is meant, where the function $\arctan(y/x)$ is made to return $\pi/2$ or $-\pi/2$ as appropriate, when $x = 0$.

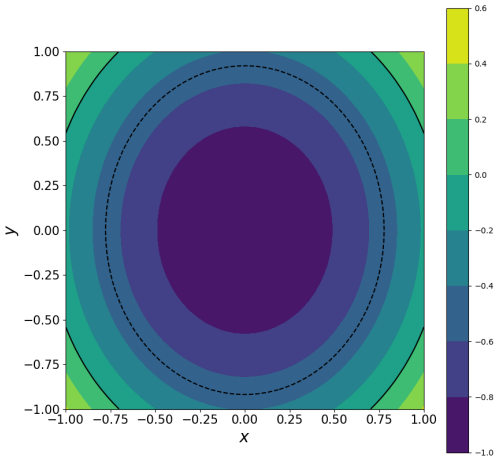

Figure 3.7: A contour plot of the approximation f^* to the circular test-function.

the star in the contour plot, making sure that the fitted line does not degrade towards a circle too much. Because this would change the length of the perimeter of the star-shaped area a lot, this could cause large errors in the solution of the level-set method depending on the application. (Particularly when the equations that are being solved depend on the perimeter length of course.)

Secondly, the star test-function provides us with a closed zero level-set curve that does not cross the boundary, meaning that its fitting method will be different than for the curves that do cross the boundary.

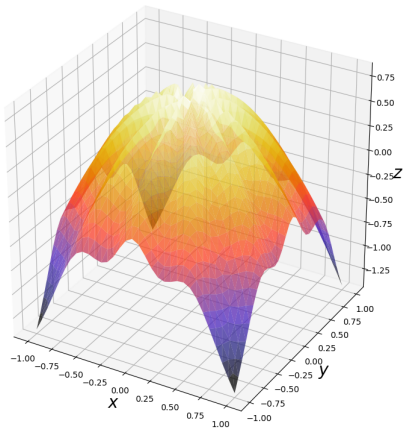

Figure 3.8: A 3d plot of the approximation f^* to the star test-function.

3.4.5 Dumbbell function

The dumbbell level-set test-function is defined as

$$f(\mathbf{x}) = \max\left\{\frac{3}{10} - \left\|\mathbf{x} - \left(-\frac{1}{2}, 0\right)\right\|_2, \quad \frac{1}{4} - \left\|\mathbf{x} - \left(\frac{1}{2}, 0\right)\right\|_2, \quad \frac{1}{10} - |y|\right\},$$

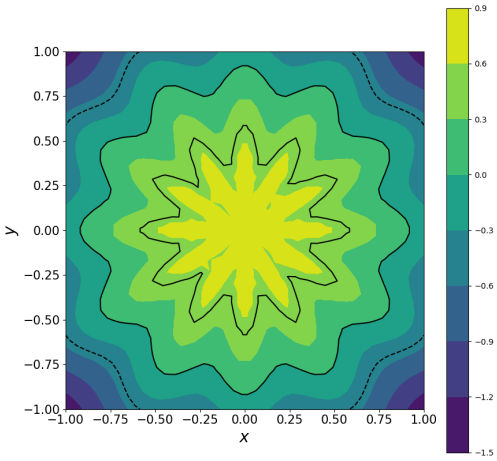

Figure 3.9: A contour plot of the approximation f^* to the star test-function.

where $\mathbf{x} = (x, y)$ and $\|\cdot\|_2$ is the Euclidean norm. In Figure 3.10 the approximation f^* of f on the triangular mesh is shown. The zero level-set curve is visible in the contour plot in Figure 3.11, as the solid line, which is the set $\{(x, y) : f^*(x, y) = 0\}$, providing a good approximation to $\{(x, y) : f(x, y) = 0\}$.

The difficult parts of fitting this level-set function will be to handle the middle area where the horizontal parts of the zero level-set curve can be quite close together. The difficulty here would be to choose when they should remain separate and when they should meld together and disappear, such that the topology changes to two separate regions.

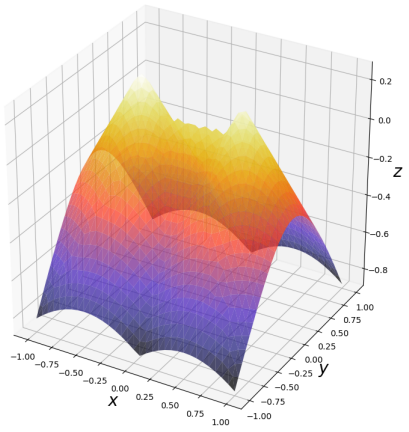

Figure 3.10: A 3d plot of the approximation f^* to the dumbbell test-function.

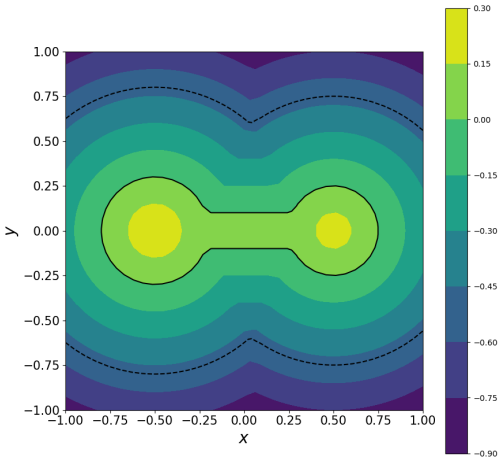

Figure 3.11: A contour plot of the approximation f^* to the dumbbell test-function.

Chapter 4

Some simple meshfitting methods

This chapter describes and analyses the existing method of the combination of moving and adding points, and analyses an improvement on this method which allows for edges to be 'flipped', and another improvement which orthogonally projects points to the zero level-set curve instead of moving them simply towards a nearby zero point p_z.

4.1 Moving points to interpolated zero-point on zero-edge

The simplest possible method to adjust the mesh is to just move points towards nearby intersections of the grid with the zero level-set curve whenever these points are closer to the respective intersections than some given threshold. Let h be the edge length of the triangles of the mesh before adjustment. For this case Verbeek [2] has analysed what threshold works best, which has resulted in an optimal threshold of around $0.3h$.
The best way to implement this method is then to choose for each zero edge e'_z the point p that's closest to the interpolated point $p_{\text{interp}} \in e_z$ and list for this point p, all the distances to the interpolations corresponding to its incident zero edges E_{incident} (if there are more than one), and to chose the interpolation with the least distance towards the point under consideration;

$$p_{\text{interp}} \text{ such that } \left\|p - p_{\text{interp}}\right\|_2 = \min\left\{\left\|p - p_{\text{interp}}\right\|_2 : p_{\text{interp}} \in e_z \in E_{\text{incident}}\right\}.$$

Then this is the interpolation to which the point will be moved. This is depicted in Figure 4.1.

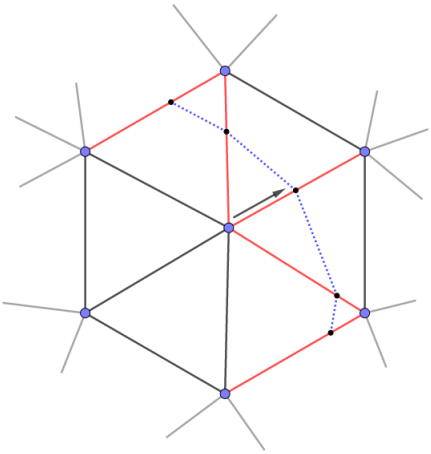

Figure 4.1: Part of the mesh with zero-edges in red and intersections with the blue zero level-set curve in black, where the arrow depicts the movement that would result from the algorithm (ignoring the threshold).

4.2 Orthogonally project to the level-set curve

A slight improvement on this method is to use orthogonal projections to project the point p towards the nearby level-set edges and then choosing the projected point or or interpolated zero level-set point that is the closest. The upside to this method is that points are not moved as far as otherwise would happen, distorting neighbouring triangles less, as well as having less chance of moved points being very close together. The result of this process for one point is depicted in Figure 4.2. Some more detailed technicalities of this projection will be described in Section 5.3.

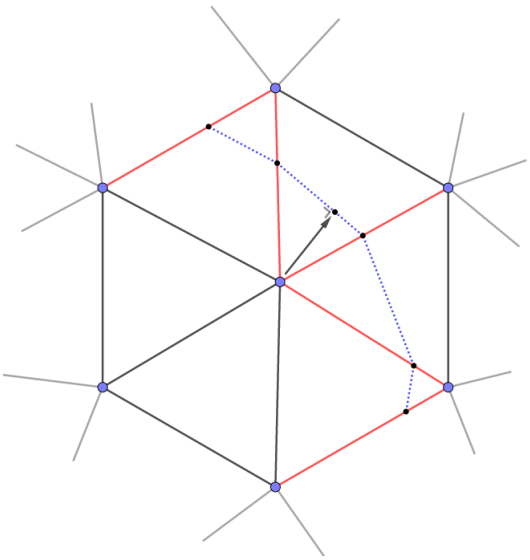

Figure 4.2: Part of the mesh with zero-edges in red and intersections with the blue zero level-set curve in black, where the arrow depicts the movement that would result from the algorithm (ignoring theshold).

4.3 Adding edges

The third option is to add some edges to the mesh to incorporate the level-set edges. The idea is to look at those edges which are not incorporated yet after applying the first method described in the previous sections, where grid-points were moved when they were closer than $0.3h$ to a zero point. (Here h is the edge length of the triangles in the mesh before adjustment.) That method leaves untouched the cases where moving a grid-point would deform the triangles too much, so it is presumed that adding more edges will keep the skewness of the triangles within more reasonable bounds, even though the triangle sizes will of course be affected.

When the zero level-set curve intersects only one edge of a triangle Δ, then we simply add the line from this intersection $p_{\text{interp}} \in e_{\text{z}} \in \Delta$ to the point opposite of said edge e_{z}. In the rest of this section we will concern ourselves with the cases where the level-set curve intersects two boundaries of a triangle, with both intersections at more than $0.3h$ from the grid-points. (Below this threshold point(s) would be moved resulting in the case with one or zero intersections with an edge.)

The situation is depicted in Figure 4.3. Referring to the figure, it seems more than reasonable to add the line e_{CD} connecting both intersections p_{C} and p_{D} to the mesh, after which there are two choices to add one final line to make the new grid triangular again, e_{BC} and e_{AD}. Let T_1 be the set of triangles of the mesh after applying option 1 (adding edge e_{AD}) as shown in Figure 4.3, and T_2 the set of triangles after applying option 2 (adding edge e_{BC}). The question is then what choice to make, i.e. whether $\text{Maxskew}(T_1) < \text{Maxskew}(T_2)$, $\text{Avgskew}(T_1) < \text{Avgskew}(T_2)$ and $\text{Stdskew}(T_1) < \text{Stdskew}(T_2)$. In the same way we ask whether $\text{Maxsize}(T_1) < \text{Maxsize}(T_2)$, $\text{Minsize}(T_1) > \text{Minsize}(T_2)$ and $\text{Stdsize}(T_1) < \text{Stdsize}(T_2)$. (These are all the quality measures discussed in Sections 3.1 and 3.2.)

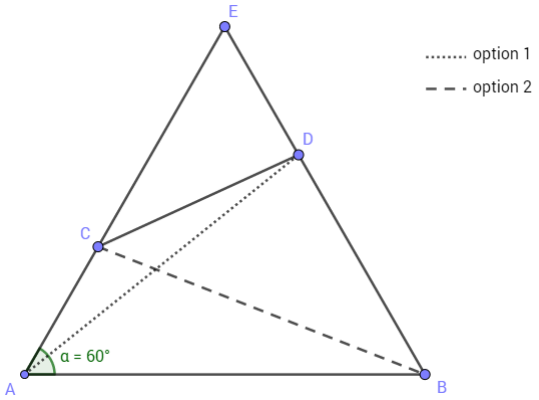

Figure 4.3: The two options of adding a line to make the grid triangular again.

Looking at Figure 4.3, our intuition tells us that option 2 is going to be the better option, both in the average skewness measure and in the standard deviation in triangle size measure. The next theorem proves that option 2 is indeed better in the skewness measure, assuming the whole triangle ΔABE to be equilateral. Note that the triangle ΔCDE is not considered since it gives the same contribution with both options.

Theorem 4.1

Applying the second option in Figure 4.3 results in smaller average skewness than applying the first option;

$$\text{Avgskew}(T_2) \leq \text{Avgskew}(T_1),$$

or more specifically, after multiplying by $\#T_1 = \#T_2$ and ignoring terms that are equal,

$$\text{Skew}(\Delta\text{ABC}) + \text{Skew}(\Delta\text{BCD}) \leq \text{Skew}(\Delta\text{ABD}) + \text{Skew}(\Delta\text{ACD}).$$

Proof 4.1

We have already seen in Theorem 3.1 that only the smallest angle $\theta_{\min}$ needs to be considered when calculating the skewness of a triangle. Recall that $\text{Skew}(\Delta) = \frac{\frac{\pi}{3} - \theta_{\min}}{\frac{\pi}{3}}$, where $\theta_{\min}$ is the smallest angle of the triangle Δ. Figure 4.4 sketches the situation. Without loss of generality we can assume that points A and B are at the same height, and also that point D lies higher than point C, and thus $\delta > 0$.

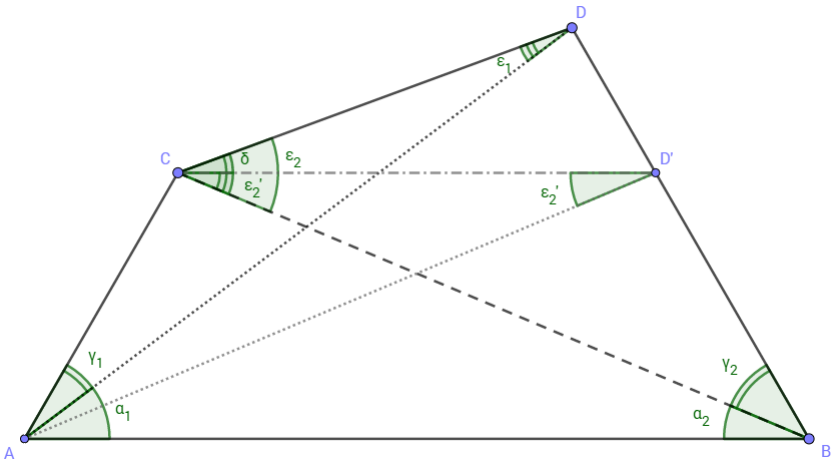

Figure 4.4: The angles used in the proof of Theorem 4.1 .

Viewing Figure 4.4, we see that the minimal angle $\theta_{\min}$ of ΔABC is

$$\theta_{\min}(\Delta\text{ABC}) = \alpha_2 = \frac{\pi}{3} - \gamma_2,$$

since angle $\angle$ACB is more than $\frac{\pi}{3}$, and angle $\angle BAC = \frac{\pi}{3}$, and then the second equality follows since angle $\angle ABD = \frac{\pi}{3} = \alpha_2 + \gamma_2$. Using the same reasoning we have that

$$\theta_{\min}(\Delta\text{ABD}) = \alpha_1 = \frac{\pi}{3} - \gamma_1.$$

Now consider triangle ΔBCD. Since $\angle\text{BDC} > \frac{\pi}{3}$, this cannot be $\theta_{\min}(\Delta\text{BCD})$, so $\theta_{\min}(\Delta\text{BCD})$ is either ε_2 or γ_2.

First consider the case when $\theta_{\min}(\Delta\text{BCD}) = \gamma_2$. For triangle ΔACD we have

$$\theta_{\min}(\Delta\text{ACD}) \leq \gamma_1.$$

Now noting that the skewness of a triangle is a function of the negative of the smallest angle $\theta_{\min}$, we can see that the smallest average skewness corresponds with the largest sum of the smallest angles of the triangles. Summing all these smallest angles in S_1 we have

$$\sum_{\Delta \in T_1} \theta_{\min}(\Delta) = \theta_{\min}(\Delta\text{ABD}) + \theta_{\min}(\Delta\text{ACD}) \leq \frac{\pi}{3} - \gamma_1 + \gamma_1 = \frac{\pi}{3}.$$

Summing all the smallest angles in S_2 we have

$$\sum_{\Delta \in T_2} \theta_{\min}(\Delta) = \theta_{\min}(\Delta\text{ABC}) + \theta_{\min}(\Delta\text{BCD}) = \frac{\pi}{3} - \gamma_2 + \gamma_2 = \frac{\pi}{3}.$$

From this it follows that the sums of the smallest angles in option 1 is always smaller or equal to the sum of the smallest angles in option 2, and since the skewnesses are a function of the negative of the smallest angles, we then have that $\text{Avgskew}(T_2) \leq \text{Avgskew}(T_1)$, where equality holds when points C and D are at the same height above the base of ΔABE.

Now we need to consider the other possibility, namely that $\theta_{\min}(\Delta\text{BCD}) = \varepsilon_2$. For triangle ΔACD we have that

$$\theta_{\min}(\Delta\text{ACD}) \leq \varepsilon_1.$$

Now summing over the smallest angles as before, we have

$$\sum_{\Delta \in T_1} \theta_{\min}(\Delta) = \theta_{\min}(\Delta\text{ABD}) + \theta_{\min}(\Delta\text{ACD}) \leq \alpha_1 + \varepsilon_1.$$

Likewise for option 2 we have, since $\varepsilon_2 = \varepsilon_2' + \delta$ (see Figure 4.4), that

$$\sum_{\Delta \in T_2} \theta_{\min}(\Delta) = \theta_{\min}(\Delta\text{ABC}) + \theta_{\min}(\Delta\text{BCD}) = \alpha_2 + \varepsilon_2 = \alpha_2 + \varepsilon_2' + \delta.$$

The remaining portion of the proof now consists of two parts, namely proving $\alpha_2 - \alpha_1 > -\delta$ and proving $\varepsilon_2' > \varepsilon_1$. For when these two inequalities hold, we have

$$\sum_{\Delta \in T_2} \theta_{\min}(\Delta) - \sum_{\Delta \in T_1} \theta_{\min}(\Delta) = \alpha_2 - \alpha_1 + \delta + \varepsilon_2' - \varepsilon_1 > 0. \tag{4.1}$$

So let's now look at the first of these two inequalities, $\alpha_2 - \alpha_1 > -\delta$, or equivalently $\alpha_1 - \alpha_2 < \delta$. Notice that because of symmetry, $\angle\text{BAD}' = \alpha_2$ and thus $\alpha_1 - \alpha_2 = \angle\text{DAD}'$. Calling this last angle α', we now have the situation sketched in Figure 4.5, and we would like to prove $\alpha' < \delta$.

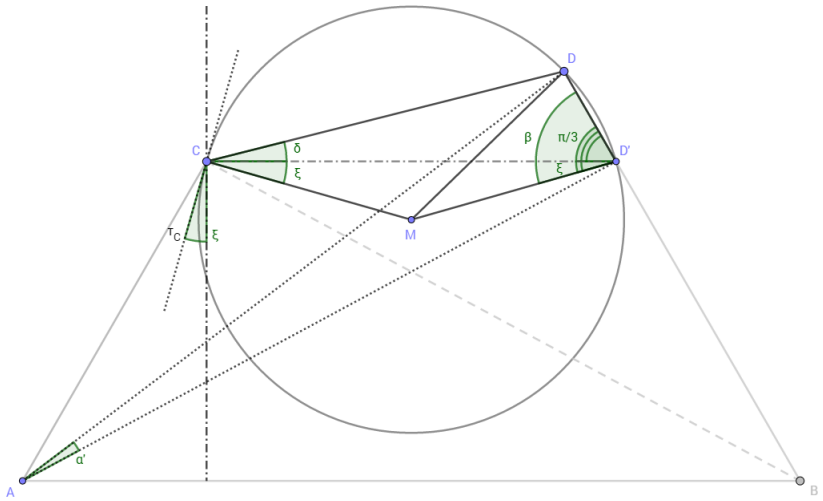

Figure 4.5: Figure showing that $\alpha' < \delta$.

To do this, we first construct a circle through the points C, D and D′ as shown in Figure 4.5. The significance of this circle is that if we move point C around this circle, keeping D and D′ in place, the angle δ remains the same (unless we move C all the way into the arc DD′ of course). In fact, if we move C to some point inside the circle, then the angle δ will be bigger. If we move it instead to some point outside the circle, but not crossing the extension of AC, then the angle δ will be smaller than its present value. We will use this observation to prove that δ is bigger than α' by proving that α' lies outside the circle.
First construct the tangent line τ_C to the circle touching the circle at point C. We will prove that this tangent line makes an angle ξ with the vertical line that is less than $\frac{\pi}{6}$. If this is the case, then moving point A from C towards its present location will take it outside the circle, thus proving that $\alpha' < \delta$, since both α' and δ are two angles corresponding two the same chord of the circle DD′.

Note that from the previous figures we have that $\angle \mathrm{CD'D} = \frac{\pi}{3}$. Because MC is perpendicular to the line τ_C, we have that $\angle MCD' = \xi$, and because the triangle ΔCDD′ is isosceles we have also that $\angle \mathrm{CD'M} = \xi$. Lastly because the angles of the triangle ΔMDD′ must add to π, the angle β cannot be more than $\frac{\pi}{2}$ (since ΔMDD′ is also isosceles). This means that

$$\xi = \beta - \frac{\pi}{3} < \frac{\pi}{2} - \frac{\pi}{3} = \frac{\pi}{6}.$$

And because now $\xi < \frac{\pi}{6}$ we must have that point A falls outside of the circle, and thus in the region that implicates that α' is smaller that δ. The only case where α' and δ can be equal is when C coincides with A, which would imply a different kind of cut in the mesh.

Then last but not least the second inequality, $\varepsilon_2' > \varepsilon_1$, needs to be proven. To do this, we first construct a circle through the points A, C and D′ as shown in Figure 4.6.

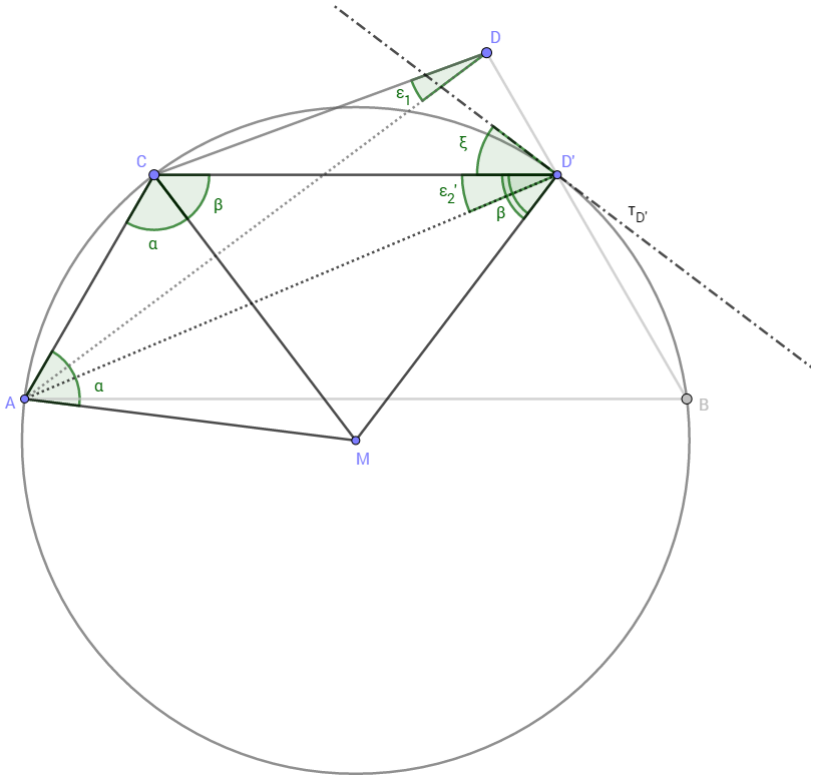

Figure 4.6: Figure showing that $\varepsilon_2' > \varepsilon_1$.

We will use a similar technique as the one applied in the proof of the previous inequality. This circle implies that if we move point D′ around this circle, while keeping all the other points stationary, the angle ε_2' does not change (unless we move into the arc AC of course), and if we move D′ to some point inside the circle, then the angle ϵ_2' will be bigger. If we move it instead to some point outside the circle, but not crossing the extension of AC, then the angle ε_2' will be smaller than its present value. We will use this observation to prove that ε_1 is smaller than ε_2' by proving that ε_1 lies outside the circle.
First construct the tangent line to the circle touching the circle at point D′. We will prove that this tangent line makes an angle ξ with the line CD′ that is smaller than $\frac{\pi}{3}$. If this is the case, then moving point D from D′ towards its present location will take it outside the circle, thus proving that $\varepsilon_1 < \varepsilon_2'$, since both angles ε_2' and ε_1 are angles corresponding to the same arc AC of the circle. From the previous figures we know that $\angle \mathrm{ACD}' = \frac{2\pi}{3}$. Considering triangle ΔAMC we can easily see that $2\alpha < \pi$ and thus $\alpha < \frac{\pi}{2}$. This means that

$$\beta = \frac{2\pi}{3} - \alpha > \frac{2\pi}{3} - \frac{\pi}{2} = \frac{\pi}{6}.$$

Since line MD′ is perpendicular to the tangent line $\tau_{D'}$, we now have that

$$\xi = \frac{\pi}{4} - \beta < \frac{\pi}{4} - \frac{\pi}{6} = \frac{\pi}{12}.$$

We have proved that $\xi < \frac{\pi}{12} < \frac{\pi}{6}$ and as such the line DD′ lies outside of the circle through points A, C and D′, and because of that the angle $\varepsilon_1 < \varepsilon_2'$. This proved the last bit of Equation 4.1 such that the sums of the smallest angles is smaller when option 1 is applied. As stated earlier we then have that the sums (and thus the averages) of the skewnesses of the triangles is bigger when option 1 is applied. ■

Thus we have shown that $\text{Avgskew}(T_2) \leq \text{Avgskew}(T_1)$, and indirectly (following the remarks in Section 3.1) also that $\text{Stdskew}(T_2) \leq \text{Stdskew}(T_1)$.

The next quality measure to check is the $\text{Maxskew}(T)$ measure, the following theorem proves that option 2 performs better in terms of this measure as well.

Theorem 4.2

Let T_1 be the triangles after applying option 1, and T_2 likewise for option 2. Then the following holds:

$$\text{Maxskew}(T_1) \geq \text{Maxskew}(T_2),$$

or more specifically:

$$\theta_{\min}(T_1) \leq \theta_{\min}(T_2),$$

where $\theta_{\min}(T)$ is simply the smallest angle of any corner in any of the triangles in the set T.

Proof 4.2

The proposition of this theorem is equivalent to: for every angle of a triangle in T_2, there exists an angle of a triangle in T_1 that is at least as small. More specifically:

$$\forall \theta_2 \in T_2,\ \exists \theta_1 \in T_1 : \theta_1 \leq \theta_2,$$

where with $\theta \in T$ we mean $\theta \in \Delta$ for some $\Delta \in T$, and each triangle Δ 'contains' three angles θ corresponding to its three corners. Looking at Figure 4.7, we can see that

$$\{\theta \in T_1\} = \{\gamma_1, \varepsilon_1, \delta + \varepsilon_2 + \beta, \kappa, \alpha_1, \varepsilon_2'' + \gamma_2\}$$

and

$$\{\theta \in T_2\} = \{\alpha_1 + \gamma_1, \beta, \varepsilon_2'', \gamma_2, \varepsilon_2 + \delta, \varepsilon_1 + \kappa\}.$$

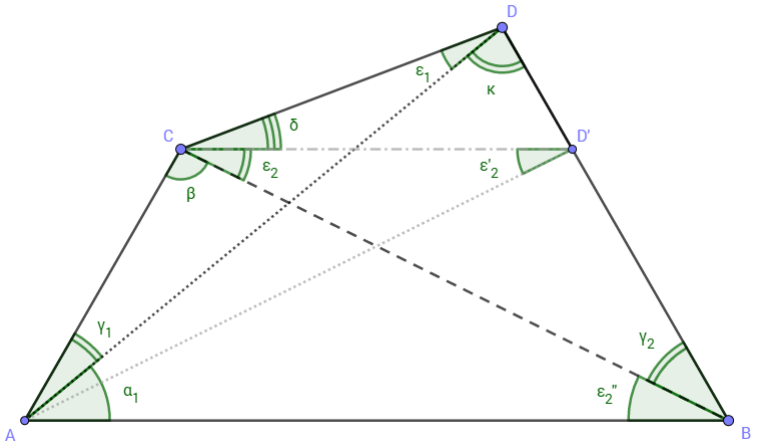

Figure 4.7: Figure showing the angles for the proof of Theorem 4.2.

Now we only need to find for every $\theta_2 \in T_2$ some $\theta_1 \in T_1$ such that $\theta_1 \leq \theta_2$. First consider $\alpha_1 + \gamma_1 \in T_2$, this is clearly bigger than $\gamma_1 \in T_1$. Secondly $\beta \in T_2$ is larger than $\kappa \in T_1$ since $\alpha_1 > \varepsilon_2''$ and $\alpha_1 + \gamma_1 = \varepsilon_2'' + \gamma_2 = \frac{\pi}{3}$, and for both triangles (ΔABD and ΔABC) the angles need to sum to π. For angle $\varepsilon_2'' \in T_2$ notice that

lines AB and CD′ are parallel, AB‖CD′, such that $\varepsilon_2'' = \varepsilon_2 = \varepsilon_2'$. From the figure it is easily seen that $\varepsilon_2' \geq \varepsilon_1$ such that $\varepsilon_2'' \geq \varepsilon_1 \in T_1$. Now consider $\gamma_2 \in T_2$. Since $\alpha_1 + \gamma_1 = \varepsilon_2'' + \gamma_2 = \frac{\pi}{3}$, and $\alpha_1 \geq \varepsilon_2''$, we immediately see that $\gamma_2 \geq \gamma_1 \in T_1$. Now for $\varepsilon_2 + \delta \in T_2$ we have $\varepsilon_2 + \delta \geq \varepsilon_2 = \varepsilon_2'' \geq \varepsilon_1 \in T_1$. For the last angle $\varepsilon_1 + \kappa \in T_2$ we simply have $\varepsilon_1 + \kappa > \varepsilon_1 \in T_1$.

This proves that for every $\theta_2 \in T_2$ some $\theta_1 \in T_1$ can be found such that $\theta_1 \leq \theta_2$ from which it follows that $\theta_{\min}(T_1) \leq \theta_{\min}(T_2)$ and thus $\text{Maxskew}(T_1) \geq \text{Maxskew}(T_2)$. ■

It is reasonable to assume that when option 2 (see Figure 4.3) is better for the total skewness, that then the standard deviation of triangle sizes will be better (smaller) with option 2 as well. We will denote the size of a triangle Δ by $|\Delta|$, and the standard deviation of the sizes of a set T of triangles by $\sigma\,(\,\{|\Delta| : \Delta \in T\})$. The next theorem proves the assertion. Note that we can again ignore the existence of triangle ΔCDE, since it is the same size for both options.

Theorem 4.3

Let T_1 be the set of triangles after applying option 1, and T_2 likewise for option 2. Then the following holds:

$$\text{Stdsize}(T_1) \geq \text{Stdsize}(T_2),$$

or more specifically:

$$\sigma\,(\,\{|\Delta| : \Delta \in T_1\}\,) \;\geq\; \sigma\,(\,\{|\Delta| : \Delta \in T_2\}\,). \tag{4.2}$$

Proof 4.3

First note that $T_1 = \{\Delta\text{ACD}, \Delta\text{ABD}\}$ and $T_2 = \{\Delta\text{ABC}, \Delta\text{BCD}\}$. The standard deviation $\sigma(\,\{|\Delta| : \Delta \in T\}\,)$ is defined as

$$\sigma(\,\{|\Delta| : \Delta \in T\}\,) \;=\; \sqrt{\frac{1}{\#T}\sum_{\Delta\in T}(|\Delta| - \mu)^2},$$

where $\#T$ denotes the number of triangles involved, and $\mu = \frac{1}{\#T}\sum_{\Delta\in T}(|\Delta|)$ is the average of the triangle sizes $|\Delta|$. This can be written as

$$\#S \cdot \sigma^2(\,\{|\Delta| : \Delta \in T\}\,) \;=\; \sum_{\Delta\in T}(|\Delta| - \mu)^2 = \sum_{\Delta\in T}|\Delta|^2 - 2\mu|\Delta| + \mu^2. \tag{4.3}$$

Since $\#T_1 = \#T_2$, and $\sigma \geq 0$, we have that the standard deviation of triangles sizes $\sigma(|\Delta_{\Delta\in T}|)$ is minimized whenever the sum on the right hand side of Equation (4.3) is minimized. Furthermore, when comparing the standard deviations for the two options, the last term in the sum can be ignored, since it has the same value for both options (since the total area of the mesh does not change):

$$\mu_1 = \frac{1}{\#T_1}\sum_{\Delta\in T_1}(|\Delta|) = \frac{1}{\#T_2}\sum_{\Delta\in T_2}(|\Delta|) = \mu_2 = \mu. \tag{4.4}$$

We now have

$$\sum_{\Delta\in T_1}|\Delta|^2 - 2\mu|\Delta| = |\Delta\text{ACD}|^2 + |\Delta\text{ABD}|^2 - 2\mu(|\Delta\text{ACD}| + |\Delta\text{ABD}|),$$

$$\sum_{\Delta \in T_2} |\Delta|^2 - 2\mu|\Delta| = |\Delta\text{ABC}|^2 + |\Delta\text{BCD}|^2 - 2\mu(|\Delta\text{ABC}| + |\Delta\text{BCD}|),$$

from which the last term cancels when comparing the two options, since

$$|\Delta\text{ABC}| + |\Delta\text{BCD}| = |\Delta\text{ACD}| + |\Delta\text{ABD}|,$$

which follows from the definition of the two options and also indirectly from Equation (4.4). We have now boiled the problem down to determining whether the following holds:

$$|\Delta\text{ACD}|^2 + |\Delta\text{ABD}|^2 \geq |\Delta\text{ABC}|^2 + |\Delta\text{BCD}|^2. \tag{4.5}$$

Claim: $|\Delta\text{ABD}| \geq |\Delta\text{ACD}|$ and $|\Delta\text{ABD}| - |\Delta\text{ACD}| \geq \Big||\Delta\text{BCD}| - |\Delta\text{ABC}|\Big|$.

For the moment we take this claim for granted. Since the average μ of two sizes is exactly halfway in between the values of the two sizes, we can write

$$|\Delta\text{ABD}| - |\Delta\text{ACD}| = \mu + \delta - (\mu - \delta) = 2\delta \geq 2\varepsilon = \mu + \varepsilon - (\mu - \varepsilon) = \Big||\Delta\text{BCD}| - |\Delta\text{ABC}|\Big|,$$

for some $\delta, \varepsilon > 0$, and $\delta \geq \varepsilon$. Writing the sizes in Equation (4.5) in this way we obtain

$$\begin{aligned} |\Delta\text{ABD}|^2 + |\Delta\text{ACD}|^2 &= (\mu + \delta)^2 + (\mu - \delta)^2 = 2\mu^2 + 2\delta^2 \\ \geq 2\mu^2 + 2\varepsilon^2 &= (\mu + \varepsilon)^2 + (\mu - \varepsilon)^2 = |\Delta\text{BCD}|^2 + |\Delta\text{ABC}|^2. \end{aligned}$$

Through Equation (4.5) we have now proved Equation (4.2) and as such the theorem.

Proof of claim: Looking at Figure 4.8,

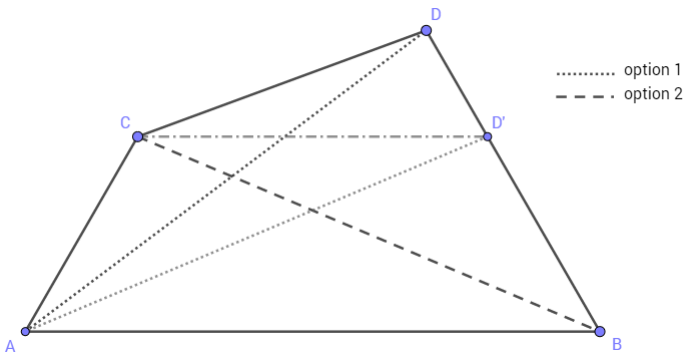

Figure 4.8: Figure showing the two situations considered in comparing the standard deviations of the triangle sizes for the two options.

we see that when D is moved to D′, that $|\Delta\text{ABC}| = |\Delta\text{ABD}'|$ and $|\Delta\text{BCD}'| = |\Delta\text{ACD}'|$, such that we have equality in the inequality of the claim. If we then move point D in the direction of line BD′ back to its original location, then triangle ΔCDD′ is added.

Considering option 1 we see that the whole area $|\Delta\text{CDD}'|$ is added to triangle ΔBCD′ to make the new triangle ΔBCD. In option 2 we see that ΔABD′ has grown an area of $|\Delta\text{ADD}'|$ to form ΔABD. We will prove that the area $|\Delta\text{ADD}'|$ is larger than $|\Delta\text{CDD}'|$. Then we also implicitly have that $|\Delta\text{ABD}| \geq |\Delta\text{ACD}|$, since ΔACD′ grows an area strictly less then $|\Delta\text{CDD}'|$ when D′ is moved to D, making the area that was added to ΔACD less than that which was added to ΔABD. More explicitly, we then have

$$|\Delta\text{ABD}| \geq |\Delta\text{ACD}|.$$

The last step is to prove that indeed $|\Delta ADD'| > |\Delta CDD'|$. Notice that $|AD'| > |CD'|$, and that $\angle CD'D = \frac{\pi}{3}$ while $\frac{\pi}{3} < \angle AD'D < \frac{2\pi}{3}$ such that $\sin(\angle AD'D) > \sin(\angle CD'D)$. Now calculating the sizes of the triangles we have

$$|\Delta ADD'| = \frac{1}{2}|AD'| \cdot \sin(\angle AD'D) > \frac{1}{2}|CD'| \sin(\angle CD'D) = |\Delta CDD'|,$$

so now we have $|\Delta ADD'| > |\Delta CDD'|$ from which it follows that

$$|\Delta ABD| - |\Delta ACD| \geq |\Delta ADD'| \geq |\Delta CDD'| \geq \Big||\Delta BCD| - |\Delta ABC|\Big|,$$

where also the facts that $|\Delta ACD'| < |\Delta ABD'|$ and $|\Delta BCD'| < |\Delta ABC|$ were used in the first and last inequalities, respectively.

This proves the claim. ■

So we now have $\text{Stdsize}(T_2) \leq \text{Stdsize}(T_1)$. Since the total area of the triangles in T_1 and T_2 respectively is the same, this also indirectly proves that $\text{Maxsize}(T_2) \leq \text{Maxsize}(T_1)$ and $\text{Minsize}(T_2) \geq \text{Minsize}(T_1)$.

So for all quality measures we have now proved that option 2 performs better than option 1, proving that option 2 is indeed the best option when adding edges is necessary and the zero level-set curve crosses two edges of the triangle.

4.4 Flipping edges

When the zero level-set curve crosses one corner of a triangle a good option is to add the edge from the corner to the interpolated intersection of the zero level-set curve with the opposite edge. When the other triangle to which that edge belongs is of the same type, however, it is often better to use a flip mechanism as shown in Figure 4.9.

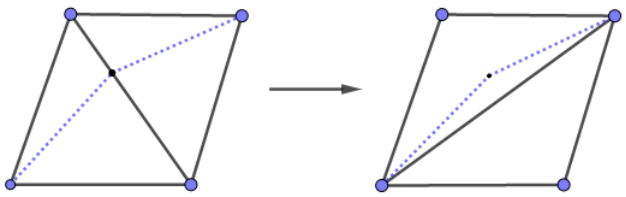

Figure 4.9: The flip mechanism, the zero level-set curve is depicted as the blue dotted line.

It is clear from the picture that, using the flip mechanism, the maximum skewness will be better since the smallest angle of the four triangles in the left part of the figure is smaller than those of the two triangles in the right part. In the same way it follows that the average skewness and the standard deviation of the skewness will both be better, and having just two instead of four smaller triangles causes the standard deviation of sizes to be better as well. The maximum and minimum size quality measures might deteriorate slightly from applying the flip mechanism instead of the cut mechanism, but only in special circumstances which normally won't be present in a well constructed starting mesh.

Aside from this, the flip method results in a fit to the zero level-set curve that is less accurate, since only one edge is used, where in the cut method two edges would be used for the same piece of the zero level-set curve. From this it follows that the flip mechanism should be used whenever possible, where the goodness of fit to the zero level-set curve is not that important.[1] (It will be assumed that this is always the case since the starting-mesh edge-length was already chosen as to provide sufficient detail.)

[1] The goodness of fit is particularly important in equations where the evolution of the zero level-set curve in time is dependent on the boundary length of the domains.

Chapter 5

Shortest path method

The methods of the previous chapter all chose the points to move by selecting the points that where closer to an interpolated zero level-set point or projected point than some given threshold. This can give very distorted meshes in some circumstances however, which is why a new way of choosing which points to move is needed. The idea behind the shortest path algorithm is that the path along the mesh edges that is fitted to the zero level-set curve has to be as 'smooth' as possible with some fruitful definition of 'smooth'. This is translated to a 'short' path when the edges of the mesh are given weights depending on their orientation with respect to nearby level-set edges.

In the first section the choice of edge weights is described and explained, after which in the second section workings of the algorithm are explained. The third section then gives the method by which the chosen points are moved towards the level-set curve and are redistributed along this line. The next chapter will show how to redistribute the internal (non-boundary and non-fitted) points of the mesh.

5.1 Edge weights

The edge weights represent how suitable the edge is to include in the path that will be fitted to the zero level-set curve. The weight measure should naturally include the angle that the edge e makes with a level-set edge e_{lvset}, denoted by $\Delta\alpha(e, e_{\text{lvset}})$,[1] as well as its distance from this level-set edge, chosen as $\|\Delta\mathbf{m}(e, e_{\text{lvset}})\|_2$, where $\Delta\mathbf{m}(e, e_{\text{lvset}}) = \mathbf{m}(e) - \mathbf{m}(e_{\text{lvset}})$ is the vector between the midpoints of the edges e and e_{lvset}. In order to calculate this measure, it has been chosen to just consider the edge weight with respect to the closest (in Euclidean distance) level-set edge. Experiments with also taking into account the properties with respect to some other close-by level-set edges showed no improvement in performance. (Instead it sometimes gave some unexpected results

[1] Calculating this can be a bit tricky, since one has to take into account the cases where the edges are near vertical. So calculating the angle α_e of edge e can best be done using $\alpha_e = \arctan(\Delta y/\Delta x)$ except when $\Delta x = 0$ (in which case we simply return $\frac{\pi}{2}$ of course). Here Δx is the difference between the x-coordinates of the two endpoints and Δy for the y-coordinates respectively. The difference in angle, $\Delta\alpha$, between edges e_1 and e_2 can then be calculated simply as $\min\{|\alpha_{e_1} - \alpha_{e_2}|, |\alpha_{e_1} - \alpha_{e_2} + \pi|, |\alpha_{e_1} - \alpha_{e_2} - \pi|\}$.

with the 'star' test-function described in Section 3.4.4, making the chosen path look more circle-like than star-like.)

The chosen measure for the edge weight $w(e)$ of edge e is defined as

$$w(e) = (1 + \Delta\alpha(e, e_{\text{lvset}})) \cdot (1 + \|\Delta\mathbf{m}(e, e_{\text{lvset}})\|_2) - 1 \tag{5.1}$$

where $e_{\text{lvset}} \in C_{\text{z}}$ is chosen such that $\|\Delta\mathbf{m}(e, e_{\text{lvset}})\|_2$ is minimized, C_{z} being the zero level-set curve which consists of the set of all level-set edges.

Note that this measure has the bonus that the weight of an ideally positioned edge e (when $\Delta\alpha(e, e_{\text{lvset}}) = \|\Delta\mathbf{m}(e, e_{\text{lvset}})\|_2 = 0$ for some e_{lvset}) equals zero, which is an important property for some more complicated zero level-set curves like the 'dumbbell' test-function described in Section 3.4.5.

In order to be able to setup these edge distance properties quickly, a branching algorithm has been used. This algorithm loops over all level-set edges, and branches along the neighbouring edges, saving for these edges the values of $\|\Delta\mathbf{m}\|_2$, $\Delta\alpha$ and $w(e)$. When these already exist it checks whether the $\|\Delta\mathbf{m}\|_2$ is smaller then the old one, otherwise it cuts off the branch. If the new $\|\Delta\mathbf{m}\|_2$ property is smaller it updates the value of $w(e)$. This process is only allowed to go a specified depth along the neighbour-edges branches. (A depth of two edges has been found to be sufficient.)

Since no point is connected to more than, say, 8 edges, this algorithm proves to be polynomial in the total number of edges, with an amount of edges to calculate on the order of $\mathcal{O}(\#E)$,[2] meaning that this algorithm is suited for any size of mesh. The algorithm is also suited for larger dimensional meshes, given that the maximum number of incident edges per point does not grow too quickly.

5.2 Application of the shortest path algorithm

Now that the edge weight properties are calculated one still has to find some way of applying this algorithm to given meshes and zero level-set curves. In order to do this, two cases have been distinguished, allowing for the simplest topologies of zero level-set curves to be fitted to. The first and simplest case is when the zero level-set curve crosses the boundary, and the second case is when it does not, for example when the zero level-set curve is a simply closed curve. The present algorithm allows for (in theory, depending on the details) one simple closed curve or any combination of boundary-crossing curves, possibly intersecting. First the case of boundary intersecting curves will be handled after which the additions necessary for application on simple closed curves are dealt with.

[2] For the algorithm does not go deeper than 2 edges along the branches, we have in total no more 8^2 edges per point of the first edge, so we have that the total number of edges calculated when no branches are cut is no more than $2 \cdot 8^2$ per starting edge, and the number of level-set edges from which each branching algorithm springs is less than $\#E$, the total number of edges in the mesh, revealing the upper bound of $2 \cdot 8^2 \cdot \#E$ for the whole algorithm.

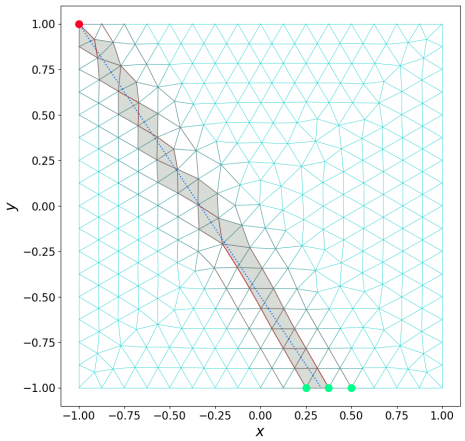

Figure 5.1: A mesh for the linear test-function showing the choices of start-point sets, each set is shown in a different color, and the redder the edges the lower their assigned weights are.

5.2.1 Choosing start and endpoints

In order to be able to apply a shortest path algorithm to the problem, one first needs to find some suitable start points and end points. In the case of boundary-crossing zero level-set curves it is obvious that at least at each crossing with the boundary at least one start point must be defined. Because it is not certain which of the close-by boundary-points will be the optimal choice for the shortest path, instead sets of three close-by boundary-points are defined as 'start-point sets' for each crossing.[3] For this the closest boundary point and its two neighbouring boundary points are chosen, excluding any corner points. When the crossing is exactly in a corner point (so no movement is allowed for this point), then just that one corner point is added as a start-point set.

These choices for the linear test-function are shown in Figure 5.1, where the red point is a corner point so that set consists only of that point, while the other set consists of three neighbouring boundary points.

The result is a set S consisting of two start-point sets $S = \{s_1, s_2\}$ where the first set s_1 equals just the corner point, $s_1 = \{p_{\text{corner}}\}$ shown in red in Figure 5.1. The second set equals the three points shown in green, $s_2 = \{p_1, p_2, p_3\}$.

When there are intersections of the zero level-set curve within the interior of the mesh, then more points need to be added to make sure all the parts of the zero level-set curve are fitted to. Figure 5.2 shows the choices for the cosine-times-sine test-function, where the two intersections in the internal part of the mesh are automatically detected and each added four times as start-point sets, because they each are the start point of four paths. (If no point on the mesh

[3]The number of boundary points to consider here was chosen arbitrarily, although it can be argued that at least the two closest should be included since these are the most likely to be chosen as start point of the path. The third point might be chosen in some special circumstances.

Also note that corner points are not to be considered here, since these are not allowed to move. If there are no points to consider because all of the three are corner points, the algorithm will fail to find a shortest path and the mesh should be made more accurate.

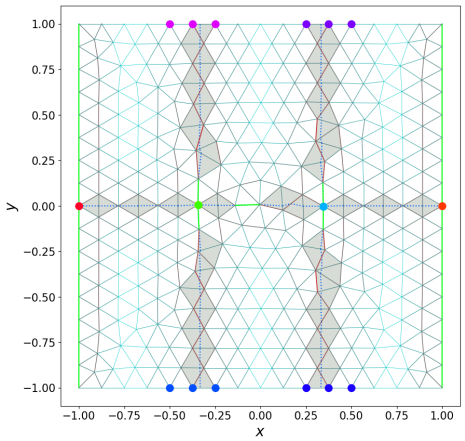

Figure 5.2: A mesh for the cosine-times-sine test-function showing the choices of start-point sets, each set is shown in a different color. The two intersections in the interior of the mesh are added four times as start-point sets since these are start points of four different paths. The redder the edges the lower their assigned weights are.

coincides with the point of intersection, then the closest point is moved to the point of intersection.)

The result is a set S of stary-point sets which consists of four sets of three boundary points, two sets of one boundary point (corresponding to the red and the orange points in the figure), and eight sets of one internal point, four for the green point in the figure and four for the blue point, so S looks like

$$S = \Big\{\{p_{\text{red}}\}, \{p_{\text{orange}}\}, s_{\text{pink}}, s_{\text{purple}}, s_{\text{blue}}, s_{\text{dark blue}}, p_{\text{light blue}}\}, \{p_{\text{light blue}}\}, \\ \{\{p_{\text{light blue}}\}, \{p_{\text{light blue}}\}, \{p_{\text{green}}\}, \{p_{\text{green}}\}, \{p_{\text{green}}\}, \{p_{\text{green}}\}\Big\}.$$

Here the s_{color} sets consist of three points each. Note that the red and orange points being intersections as well is ignored here, since no shortest path algorithm is needed to 'fit' to the level-set edges that are already on the boundary.

Now S is the set of start-point sets where each start-point set occurs a number of times according to its multiplicity as a start point of a segment of the zero level-set path, and the start-point sets of intersections consist of only one point in order to make sure that no further complications occur at these points, since the shortest-path algorithm of the different segments could otherwise end in different points of the start-point set, after which further decisions would have to be made about how to connect these edges.

5.2.2 Running the algorithm itself

After building the start-point sets, one can apply Dijkstra's algorithm[4] [10],[12] to find the shortest-path solutions in the following way. Remember that Dijkstra's algorithm works by maintaining a list of paths, and at each point choosing the path with least total weight to update, and continuing in this way until the

[4]Here Dijkstra's algorithm is applied in a way that precludes the algorithm to choose any edges that have been previously chosen for other segments of the zero level-set curve.

end point of the path has been reached, and the least-weight path in the list of paths weighs more than the shortest path to the endpoint. For a more thorough explanation of Dijkstra's algorithm see for example [11].

Application of the algorithm starts with choosing one start-point set $s_1 \in S$ at random (but preferably one with only one point in it), and this start-point set is removed from the set S. From each point p_1 in this start-point set, the shortest path algorithm is applied to find the least-weight path towards some other point in some other start-point set. Once some other point p_2 in some other start-point set s_2 is reached[5], the algorithm continues until all the points in s_2 have been reached by the algorithm, and then the path with least weight is chosen from these paths to the end points in s_2. This gives a shortest path from each $p_1 \in s_1$ towards some $p_2 \in s_2$, and from these the shortest one is chosen as the resulting path. When this is done, the set s_2 is deleted from S as well, after which the algorithm continues if the set S is not yet empty. Written out in steps, this is

1. Choose a startpoint set $s_1 \in S$, remove s_1 from S.
2. For each $p_1 \in s_1$, apply Dijkstra's algorithm to find the shortest path to some $p_2 \in s_2$ for some $s_2 \in S$.
3. Then continue until all the $p_2 \in s_2$ are reached, and choose the shortest of the paths as the shortest path for p_1.
4. From the list of shortest paths for each $p_1 \in s_1$, choose the shortest one as the resulting shortest path.
5. Delete s_2 from S as well.

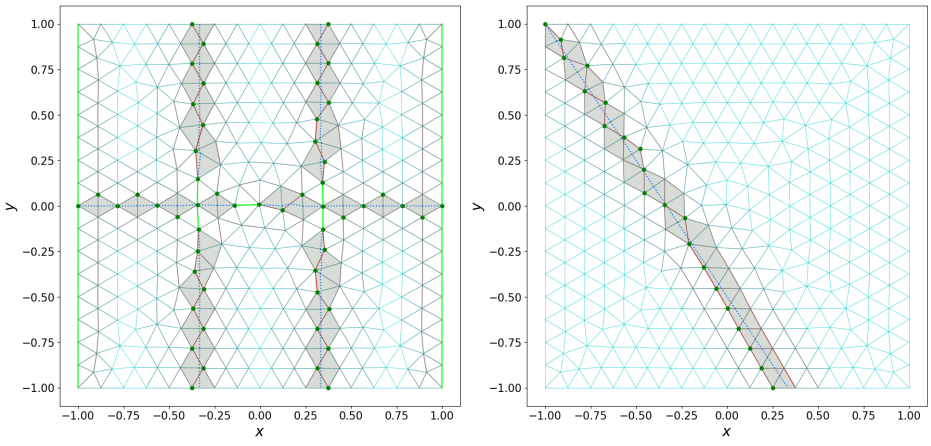

Figure 5.3: The result of the shortest-path algorithm applied to the linear and the cos-times-sin test-functions shown in green.

[5]The word 'reached' actually does not fully describe the condition that is tested for, since actually a point in s_2 is only 'reached' when not only there is some path with some path weight which ends in this point, but when this path weight is also less than the weight of any other path still to be fully calculated through by the algorithm. This is because only when every other path can only result in a path weight higher than the path weight for some p_2, does one know for certain that the current path is actually the shortest one to p_2.

After applying this algorithm, one should end up with a set of paths which together constitute the shortest-path approximation to the best points to move towards the zero level-set curve. Figure 5.3 shows the found paths for the linear and the cos-times-sin test-functions.

5.2.3 Dealing with low angles of incidence with the boundary

A problem occurs when the angle of incidence with the boundary is low. This is the case near the red start-points in Figure 5.5, which shows a part of the mesh of the circle test-function also showing the chosen start-points. Note in particular that the red chosen start-points are strictly *below* the point of intersection of the zero level-set curve with the boundary. If one chooses points above the intersection to move towards the intersection and use as a start point, then the triangle above it would get very skewed, and would remain about the same size. From error considerations in the finite-element method, it would be preferable to have a much smaller very skewed triangle there and some extra less skewed triangles to make up the rest of the space, where the smaller size of the triangles allows for a greater resolution in this complicated area as well. This difference is shown in Figure 5.4, where it can be seen that a large part of the original triangle is now less skewed, and also that more triangles occupy the same region of space.

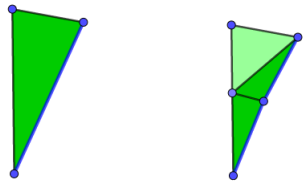

Figure 5.4: Schematic visualization of the difference between moving a point from the small angle of incidence side and from the large angle side. The amount of green of the triangles approximates the skewness of the triangle.

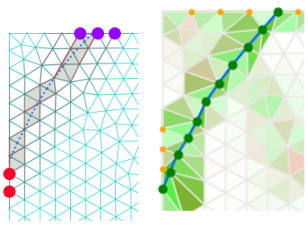

Figure 5.5: The left plot shows a detail of the mesh for the circle test-function, also showing the chosen start-points in big colored dots. The zero level-set curve is shown in dotted blue. The right plot shows the situation when the shortest-path algorithm has been applied, including movement of the points. The amount of green and red depict the skewness of each triangle and the size mismatch, respectively. The yellow points depict points that have been moved using the redistribution algorithm to be described in Section 5.3.

In order to do this, the situation is detected by comparing the orientation of the zero level-set edges incident with the boundary with the orientation of

the boundary edge itself, checking if the angle between them is smaller than $\frac{\pi}{4}$ ($= 45°$). If this is the case, the start points on the small angle of incidence side of the intersection are removed from the start-point set such that only the start points on the large-angle side of the intersection remain, which are the red points in Figure 5.5. The figure also shows that the triangles near the intersection are not too skewed.

5.2.4 Dealing with simple closed zero level-set curves

An important class of zero level-set curves consists of simple closed curves, and for these a modified method is needed to be able to fit the mesh to this curve. The problem is mainly determining the two start points to use, since then these points can be considered as 'intersections' and the previously described methods of applying the shortest path algorithm can be used.

Finding the start points

To find an appropriate path, the start-points need to found first. Ideally these points are zero-points, such that these points themselves do not have to be moved towards the zero level-set curve. So first one tries to find such zero points, and if there are more than one, one tries to find the furthest apart pair among these points. The simple approximation to the solution to this problem that is used here is to choose the first point at random, and than select as the second point the point that is the furthest away from the first point. After this the first point is swapped with the point that is the furthest away from the second point. These simple steps should provide with points that are far enough away, which can be checked against some threshold[6].

When there are not enough zero points or when they are too close together, the search for the second is done among the interpolated zero level-set points $p_{\text{interp}} \in e_{\text{z}}$ of the zero edges e_{z} of the mesh, and the first point is not changed afterwards. When there is no zero point at all, a random interpolated zero level-set point of a zero edge is chosen at random for the first point, and the rest of the algorithm proceeds as with zero points but now with interpolated zero level-set points of zero-edges. Figure 5.6 shows the two points that were found for the dumbbell test-function, showing that they are indeed quite far apart such that the algorithm is likely to operate successfully.

Applying the shortest path algorithm

In the application of the algorithm another modification has to be made to let the algorithm operate smoothly. The problem here is that the path in one direction towards the other start point along the zero level-set curve may be much shorter than the path in the other direction. In this case the algorithm

[6] In the application of the fitting algorithm to the test-functions presented here, a threshold of $5h$ was found to be sufficient, where h is the edge length of the starting mesh. A too low threshold can cause problems where the algorithm can choose a path that skips the larger part of the zero level-set curve and just makes a small circle-like fit, while a too large threshold can cause smaller zero level-set curves to not be able to be fitted to, since no pair of start points would be found.

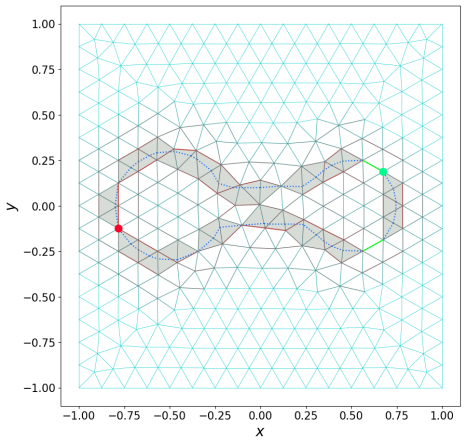

Figure 5.6: The two points found for the dumbbell test-function as start points. These points are both zero points.

may prefer to go in the same direction for both paths that are fitted to the zero level-set curve and ignore the other part of the zero level-set curve. To make sure this does not happen, all the edges in the neighbourhood of a path that has been chosen are given a very large weight, along with its neighbouring edges. Now the algorithm has extra obstacles for the direction already fitted to, and is likely to always choose the right unfitted direction.

To prevent some problems from occurring near the start point, this is only done in the middle third of the already fitted path such that the edge weights near the start points remain unchanged. Figure 5.7 shows the path that has been found in this way, and also shows the modified weights for the top path which block the algorithm from going that direction again.

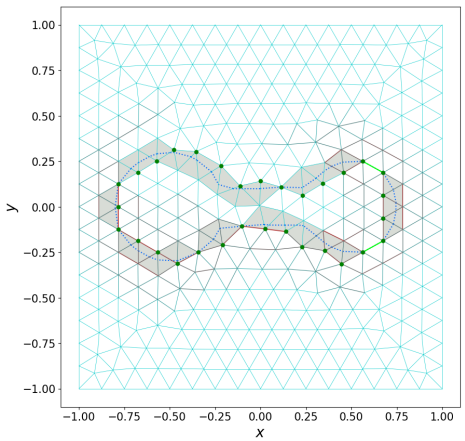

Figure 5.7: The shortest path algorithm applied to a the dumbbell test-function, a simple closed zero level-set curve. The edge weights are shown as the amount of redness in the edges. Around the top part of the zero level-set curve it can be seen that the edge weights are increased to a large value after the top path was found, in order to force the algorithm to go back along the bottom path.

5.3 Orthogonal projection and redistribution

Now that the shortest paths are determined, the path still has to be fitted to the zero level-set curve by moving the points in the path. To determine where to move these points, orthogonal projection has been used, much like in Section 4.2.

Orthogonal projection

The points in the path are now orthogonally projected to the zero level-set curve. This is done by finding the nearest level-set edges, and then orthogonally projecting to all of them. The projected point closest to the original point is then chosen. When this point is on the edge itself, the point is moved to this projected point, otherwise it is moved to the nearest zero point. This makes sure that the resulting point is actually on the path, and not on some extension of a piece of the path. The situation after orthogonally projecting is shown in Figure 5.8, for some details of the meshes for the circle and dumbbell test-functions.

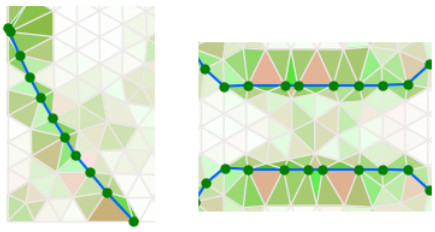

Figure 5.8: Details of the meshes of the circle and dumbbell test-functions after applying the orthogonal projection. The amount of green and red depict the skewness of each triangle and the size mismatch, respectively.

5.4 Redistributing projected points

The locations that the points have been projected to are still not ideal, as can be seen from Figure 5.8. The points are still too close together sometimes, not quite evenly distributed. To overcome this, Each point is placed exactly in the middle of the next and previous points in the paths that the shortest path algorithm found. After this they are immediately projected to the zero level-set curve again. The end points of each segment (the start points in the application of the shortest path algorithm) are left where they are. This method has the benefit that it is very quick and does not move points too far away from their first projection, but it also gives very evenly distributed points without needing a lot of iterations. (One loop over all the points in all the paths that the shortest path algorithm found.) When the largest distance moved in one iteration is less than $0.01h$, the iteration is stopped. Figure 5.9 shows the situation after the points have been redistributed. (Compare this with Figure 5.8.)

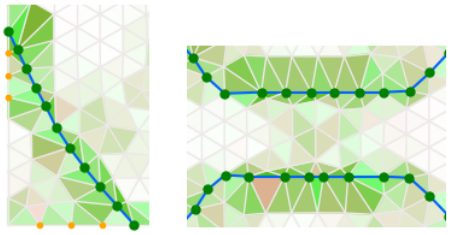

Figure 5.9: Details of the meshes of the circle and dumbbell test-functions after applying the redistribution of the projected points. The amount of green and red depict the skewness of each triangle and the size mismatch, respectively.

The same technique has been applied to the boundary points surrounding an intersection of the zero level-set curve with the boundary. This redistributes these boundary points evenly around the boundary point that has been moved to the intersection with the zero level-set curve. This can in principle also be done using the technique developed in Chapter 6, if one restricts the movement of the boundary points such that they are forced to remain on the boundary line.

Chapter 6

Spring model for mesh relaxation

After moving the shortest path points towards the zero level-set curve, the mesh may contain a lot of triangles that are skewed and small or large compared to the original mesh. To correct this, a physical model is introduced which models each edge as a spring. The lowest-energy state of the springs then corresponds to the ideal mesh configuration. The first section of this chapter describes the spring model in detail, and the second section describes the numerical iteration used to approximate the lowest energy state.

6.1 The spring model

To convert the problem of skewed and wrongly sized triangles into a physical problem, each edge is modelled as a spring with spring constant w_k. For the rest position of the springs two options will be considered. The first option is to have all springs exert zero force when their extension is zero, so when the edge length is zero. An explanation will be given for why this does not lead to good results, after which the second option is described where the rest position is chosen to be at an extension equal to h. (Where h is the edge length at the initialization of the mesh before any adjustments.)

Rest-extension equals zero

A simple approach to adjusting the points of the grid is to loop over all points of interest a couple of times and just adjusting those points one at a time. The question is then to find a simple way of determining where the equilibrium point of a mesh point is when all the surrounding points are held fixed, see Figure 6.1. We consider therefore the potential energy PE of the spring system surrounding the point. We have a nice result from Apostol and Mamikon [7] stating

$$\mathrm{PE} = \frac{1}{2}\sum_{k=1}^{n} w_k \|\mathbf{z} - \mathbf{r}_k\|_2^2 = \frac{1}{2}\sum_{k=1}^{n} w_k \|\mathbf{c} - \mathbf{r}_k\|_2^2 + \frac{1}{2} W \|\mathbf{c} - \mathbf{z}\|_2^2, \tag{6.1}$$

where $\{w_k : k \in 1, \ldots, n\}$ are the spring constants for the n springs, $\mathbf{c}$ is the weighted centroid of the surrounding points with weights equal to the spring

constants w_k, $\mathbf{z}$ is the position of the point itself and $\{\mathbf{r}_k : k \in 1, \ldots, n\}$ the position vectors of the surrounding points, and $W = \sum_{k=1}^{n} w_k$.

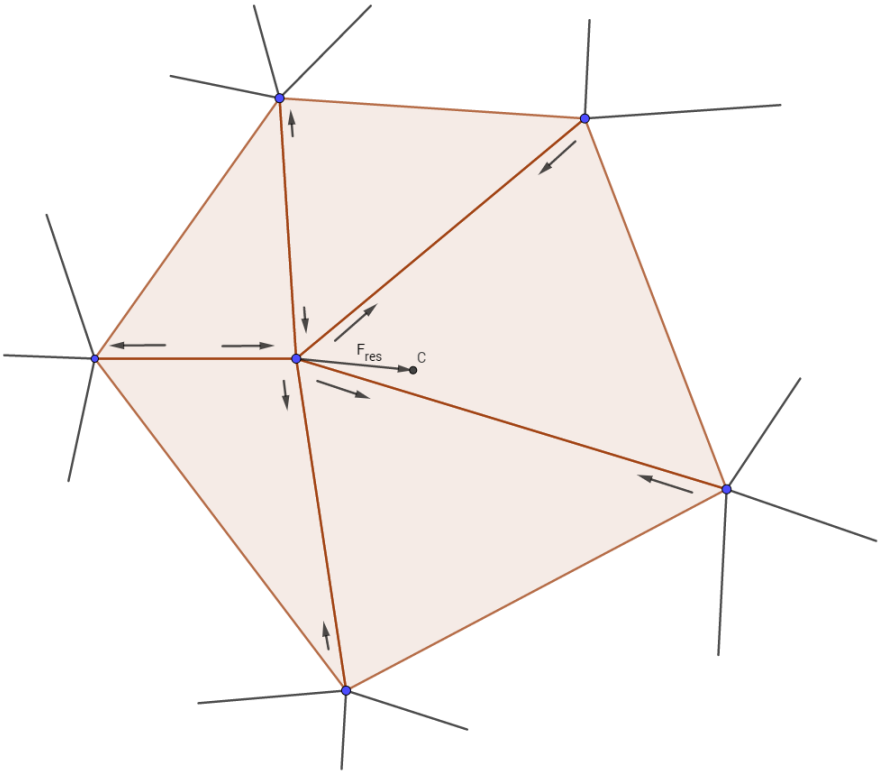

Figure 6.1: Figure showing the force system on the elements surrounding a point of the mesh. This rest extension of the springs in this figure is equal to h, at a rest extenion of 0 all the springs would be in tension proportional to the extention. The point c depicts the equilibrium point, which is in the weighted centroid in the case of a rest extension of 0.

From this result it can easily be seen that the minimum of potential energy lies at the weighted centroid, since only the term $\frac{1}{2}W\|\mathbf{c}-\mathbf{z}\|_2^2$ depends on $\mathbf{z}$ in the right-hand side of Equation (6.1), and this term has its minimum at $\mathbf{z} = \mathbf{c}$.

When for all spring constants w_k one has $w_k = 1$, this simple algorithm thus consists of moving all the grid-points of interest towards the centroid of the surrounding points until the changes thus made are under a certain threshold. In addition one could decide to only start adjusting the points when the resultant force F_{res} on the point reaches a certain threshold, because then we don't have to move as many points. The force is then calculated as

$$\mathbf{F}_{\text{res}}(\mathbf{z}) = \sum_{k=1}^{n} \mathbf{F}_k = -\sum_{k=1}^{n} (\mathbf{z} - \mathbf{r}_k). \tag{6.2}$$

Since we only need the magnitude of the force, we have

$$\|\mathbf{F}_{\text{res}}(\mathbf{z})\|_2 = \left\| \sum_{k=1}^{n} \mathbf{z} - \mathbf{r}_k \right\|_2 .$$

This algorithm does not give good results, however, since the higher concentration of points in area's where points have been moved to the zero level-set curve now means that these area's attract more points towards them, since the respective centroids will be biased towards these highly concentrated area's. Adjusting the spring constants to counteract this problem is also not a good idea since this would change the spring constants at each step, because of which a lot more iterations would be needed before converging to the solution.

This means that the only choice to make sure that concentration of points does not have any influence is to choose the more complicated system of equations corresponding to the case where the rest extension is equal to h.

Rest-extension equals h

For all the spring constants w_k, let $w_k = 1$. The set of equations to solve is now

$$\mathbf{F}_{\text{res}}(\mathbf{z}) = \sum_{k=1}^{n} \mathbf{F}_k = -\sum_{k=1}^{n} (\mathbf{z} - \mathbf{r}_k) - h\frac{\mathbf{z} - \mathbf{r}_k}{\|\mathbf{z} - \mathbf{r}_k\|_2} = 0, \tag{6.3}$$

for each point with location $\mathbf{z}$ and n surrounding points $\mathbf{r}_k$. This can be written as

$$\mathbf{F}_{\text{res}}(\mathbf{z}) = -\sum_{k=1}^{n} (\mathbf{z} - \mathbf{r}_k)\left(1 - \frac{h}{\|\mathbf{z} - \mathbf{r}_k\|_2}\right) = 0, \tag{6.4}$$

implying that one might linearise this by setting $w_k = 1 - h/\|\mathbf{z} - \mathbf{r}_k\|_2$, and applying an iterative method to the non-linear set of equations. Unfortunately the behaviour of any fixed-point iteration-method to solve this system is very erratic because the $\|\mathbf{z} - \mathbf{r}_k\|_2$ can be very small, causing the equilibrium point for the equations of one point in one iteration to possibly be far outside of the surrounding points. The next section describes an Euler-forward inspired method that performs better for solving for the equilibrium situation.

6.2 Euler-forward inspired approximation-method

In this section the equations for each point are modified in order to be able to solve them more easily. The equations become

$$\mathbf{v}_p(\mathbf{z}) = -\sum_{k=1}^{n} (\mathbf{z} - \mathbf{r}_k)\left(1 - \frac{h}{\|\mathbf{z} - \mathbf{r}_k\|_2}\right) = 0, \tag{6.5}$$

where $\mathbf{v}_p$ is the velocity vector of the point p. Thus each point moves with a certain velocity towards the equilibrium point, since $\mathbf{v}_p$ is now in the same direction that $\mathbf{F}_{\text{res}}$ used to be, and as long as the method can be forced to converge towards $\mathbf{v}_p = 0$ for all points p, then it must have the same solution since $\mathbf{v}_p = 0$ corresponds to the situation where $\mathbf{F}_{\text{res}} = 0$ in the unmodified system of equations.

The Euler-forward integration-method could then be applied to the system, such that for a point with coordinates $\mathbf{x}_p(t)$ at time t, the coordinates at

time $t + \Delta t$ will be

$$\mathbf{x}_p(t + \Delta t) = \mathbf{x}_p(t) + \Delta t\ \mathbf{v}_p(\mathbf{x}_p(t)).$$

But convergence towards the equilibrium solution is not guaranteed using such a method, since a spring system without friction can easily oscillate instead of reaching an equilibrium.

So instead of applying the Euler-forward integration-method to this set of equations, a new method is used to make sure it does converge. The idea is to just let $\Delta t = 1$ in the Euler-forward method, such that now the displacement of each point p in the time step form t to $t + \Delta t$ would be equal to the velocity vector $\mathbf{v}_p(t)$. Lets call this displacement $\vec{\Delta}$, so $\vec{\Delta}_t = \vec{\Delta}(t) = \mathbf{v}_p(t)$. To assure convergence, however, the point p is moved instead to

$$\mathbf{x}_p(t + \Delta t) = \mathbf{x}_p(t) + \frac{2h}{\pi} \arctan\left(\|\vec{\Delta}_t\|_2\right) \frac{\vec{\Delta}_t}{\|\vec{\Delta}_t\|_2}. \tag{6.6}$$

To see why Equation (6.6) works, first note that the $\frac{2h}{\pi}$ factor in front of the $\arctan(\cdot)$ moves the horizontal asymptote at $+\infty$ of $\arctan(\cdot)$ from $\frac{\pi}{2}$ towards h, such that a point can only be moved a length of maximally h in each iteration, preventing a lot of the erratic behaviour that the fixed-point iteration described earlier provided.

Secondly note that the derivative of $\arctan(\|\vec{\Delta}\|_2)$ is 1 near $\|\vec{\Delta}\|_2 = 0$ such that for small $\vec{\Delta}_t = \mathbf{v}_p(t)$ we have that the displacement roughly equals $\frac{2h}{\pi}\mathbf{v}_p$, which corresponds to $\frac{2h}{\pi}\mathbf{F}_{\text{res}}$ in the unmodified equations where $\mathbf{F}_{\text{res}}$ is the resultant force on point p (at time t). This means that a threshold on the minimal force before movement is allowed can still be built in by providing a threshold for $\vec{\Delta}$ in the modified equations, equal to $\frac{\pi}{2h}$ times the corresponding threshold for the unmodified equations.

The third property is the property that the new method actually converges (or seems to at least).

Does it converge?

The question of convergence is difficult to answer definitively without a lot of (computer-aided) algebra. This is beyond the scope of the present research project. For the mesh sizes and test functions used in this project it seems to converge, however, and the arguments for this technique already mentioned indicate that it at least will not diverge too quickly if it does. It seems likely that the method converges for similar, reasonably simple, level-set functions as well.

An oscillation could still occur, which can be seen in a one-dimensional case already when there are only three points x_1, x_2, and x_3, and the middle point x_2 is 'relaxed' using the described method. For when the middle point is a distance of Δ from the equilibrium position $\frac{1}{2}(x_1 + x_3)$, and one iteration of the method happens to move the point exactly 2Δ, then it is exactly at the opposite side from the equilibrium point. The method would then make this point oscillate between the two positions, each at a distance of Δ from the equilibrium point. Figure 6.2 shows the process.

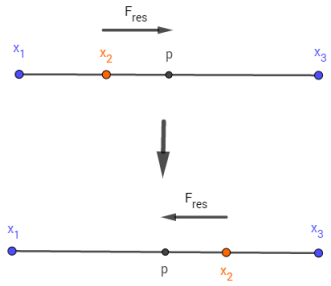

Figure 6.2: Figure showing the oscillatory property of the relaxation method for a simple one-dimensional case.

The more complicated structure of two-dimensional meshes and the many more moving points make it very unlikely that such an oscillation occurs, however. The fact that oscillations have not occurred in the test-cases corroborates this. (Oscillations would have been detected by the absence of convergence, because the maximal movement of any point in one iteration would then not fall below the threshold.)

Chapter 7

Qualitative comparison of all described methods

In order to measure the quality of the different methods, a mesh of edge-length of $\frac{1}{16}$ has been fitted to the five test-functions described in Section 3.4. Then the quality measures described in Sections 3.1, 3.2 and 3.3 are calculated for the resulting meshes. The sections below describe for each quality measure in succession the performance of the five test functions, with explanations of why these performances might be expected considering the characteristics of each test function. It is useful to compare the performance with pictures of actual meshes being fit to the test functions. These pictures are shown in Appendix A for a mesh edge-length of $\frac{1}{8}$.

The shortest path algorithm was unable to fit correctly to the cos-times-sin test-function, due presumably to some technical mistakes in the implementation. Because of this at least one triangle with area 0 and skewness 1 was created, such that the Maxskew and the Minsize measures do not give representative results in the three shortest-path method variants compared in this chapter.

7.1 Maximum skewness

The first measure to consider is the Maxskew measure, of which the results are shown in Figure 7.1. The figure shows that the cut method and the cut-and-flip method perform about the same, although a slight increase in performance is seen in the dumbbell test-function. This is to be expected, since only in very skewed situations can the skewness of flipped triangles be so large as to influence the maximum skewness taken over all the triangles.

The figure also shows that the shortest-path fit can behave quite poorly on its own, for example with the circle test-function. The redistribution method seems to improve this when it is particularly bad, but might also make things worse a bit like with the star test-function. The forward-relaxation method, however, seems to make all these changes worthwhile by improving the performance greatly, after which there is no doubt that the shortest-path approach gives a better performance in terms of Maxskew, except of course for the cos-

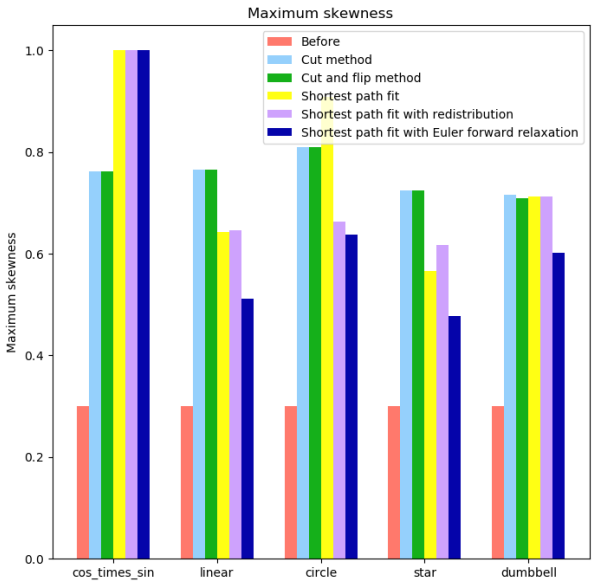

Figure 7.1: The maximum-skewness quality-measure of the initial mesh and of the resulting mesh after application of the different methods applied to each of the test-functions. The mesh edge-length of the starting mesh is $\frac{1}{16}$.

times-sin test-function where the algorithm could not fit correctly.

The Maxskew quality measure actually measures the worst triangle of the mesh, around the details of the zero level-set curve that are the hardest to fit to. These results show that for these hardest details of the mesh, the shortest-path fit with Euler-forward relaxation is the best choice.

7.2 Average skewness

For the Avgskew measure it is a different story altogether, see Figure 7.2. The flip method improves on the cut method quite a bit, which can be explained by the observation that when the cut method is applied, the result is three triangles of which at least two are quite skewed. The flip method, however, results in only two skewed triangles, which are a bit bigger. Therefore the Avgskew measure may not be an appropriate measure to compare these two methods, perhaps the skewness of each triangle should be normalized with respect to the triangle size.

The figure also shows that the shortest-path fit can perform better or worse than both cut methods. The refinement methods of the shortest-path fit, the redistribution and the Euler-forward relaxation, only make things worse in terms of the Avgskew measure. This can be explained by the fact that the shortest-path methods adjust a whole lot more triangles than the cut method and the cut-and-flip method. All these triangles were very close to being optimal before they were adjusted, so any adjustment would cause the skewness of these triangles to deteriorate. It is interesting to note, though, that for a function like

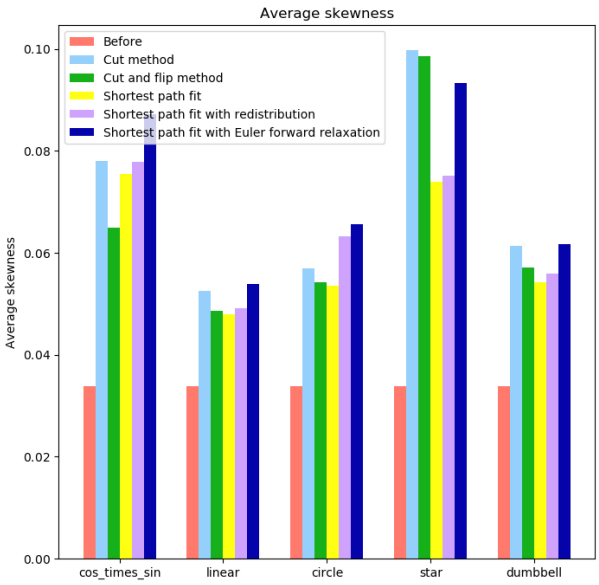

Figure 7.2: The average-skewness quality-measure of the initial mesh and of the resulting mesh after application of the different methods applied to each of the test-functions. The mesh edge-length of the starting mesh is $\frac{1}{16}$.

the star test-function, the shortest-path methods still perform better than the cut methods, which is because in this test-function in particular, a lot of cuts where needed (as opposed to moves or flips) to make the mesh fit.

This quality measure shows that when the average skewness of the triangles is what counts, then either the cut-and-flip fit or the shortest-path fit is the best choice. (Although the horrible performance of the shortest-path methods on the cos-times-sin test-function can be caused by the earlier described error in the fitting process.[1] If this is the case, the shortest-path algorithm without extensions is probably the best choice.)

7.3 Standard deviation of skewnesses

To give a better indication on the distribution of skewnesses than the Avgskew described earlier, the Stdskew measure was introduced. The results are shown in Figure 7.3. The figure shows that the cut method performs especially poorly, which the cut-and-flip method improves upon. Note that adding the flip method sometimes does not do a lot, like with the star test-function. The same argument as in the previous section about the number of skewed triangles contributing to the measure also applies here, explaining why the cut method performs so poorly by adding so many skewed triangles.

The performance of the shortest-path methods seems to be better, with the bare shortest-path method already performing at least as good as the cut

[1] See the introductory paragraph of this chapter.

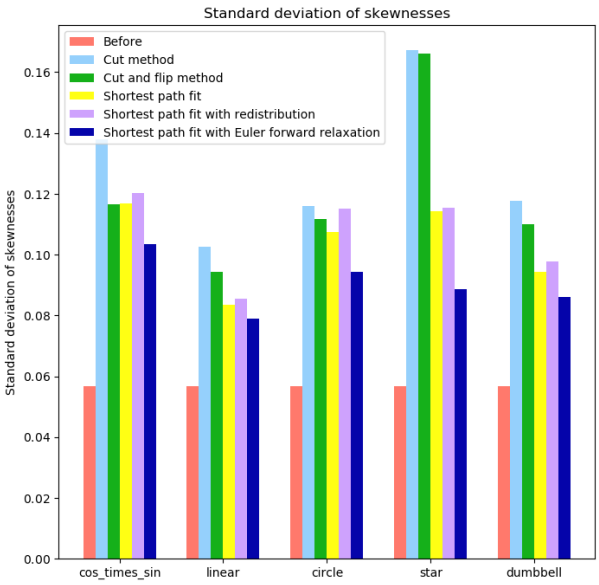

Figure 7.3: The standard-deviation-of-the-skewnesses quality-measure of the initial mesh and of the resulting mesh after application of the different methods applied to each of the test-functions. The mesh edge length of the starting mesh is $\frac{1}{16}$.

methods. The redistribution then comes with a small cost, but the Euler-forward relaxation then performs outstandingly, improving the performance a lot more then the small cost of the redistribution.

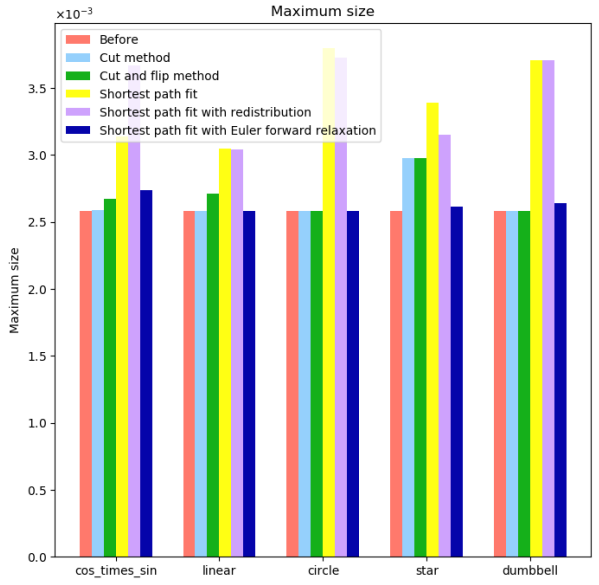

Figure 7.4: The maximum-size quality-measure of the initial mesh and of the resulting mesh after application of the different methods applied to each of the test-functions. The mesh edge-length of the starting mesh is $\frac{1}{16}$.

7.4 Maximum size

Then on to the size measures. The results of the Maxsize measure is shown in Figure 7.4. The figure shows that the cut method performs quite well, except with the star test-function, where it increases the Maxsize a bit. The flip method only seems to make things worse with the first two test functions, meaning that some already skewed triangles were used in the flip method. In the other three test functions the performance is about the same as the cut method.

The shortest-path method initially performs horrible, increasing the maximum size quite a bit with all test-functions. Only after applying the redistribution and in particular the Euler-forward relaxation does the performance improve, but then the performance is much better then all the other methods, and very consistent. In fact the quality of the resulting meshes is almost as good as the original mesh before fitting, which is assumed to be nearly ideal. The shortest-path fit with Euler relaxation seems to be the preferred choice considering the Maxsize measure.

7.5 Minimum size

The results of the Minsize measure is shown in Figure 7.5. The cut method and the cut-and-flip method seem to perform exactly the same. This is due to the fact that the cut method results in much smaller triangles than the original, much smaller than moving points around is likely to do. The cut-and-flip method also cuts a lot of triangles, because of which the worst cut is likely to still occur. (The triangles in which the worst cut occurs can likely not be flipped, since the requirements for a flip to occur will likely prevent these triangles to flip.)

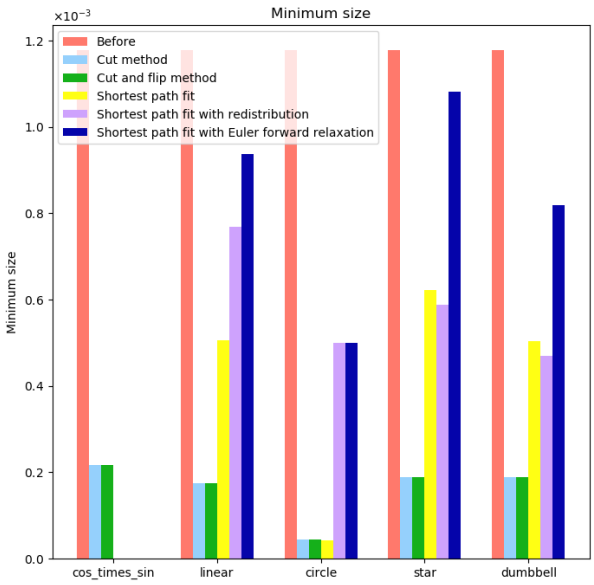

Figure 7.5: The minimum-size quality-measure of the initial mesh and of the resulting mesh after application of the different methods applied to each of the test-functions. The mesh edge-length of the starting mesh is $\frac{1}{16}$.

The values for the shortest-path methods on the cos-times-sin test-function are not representative, leaving the other four test-functions to consider. The bare shortest-path method seems already to perform a lot better than the cut methods. The redistribution can then deteriorate the matter a bit, but the Euler-forward relaxation can then improve matters greatly. On the whole the shortest-path fit with Euler forward relaxation seems to be the best choice.

7.6 Standard deviation of sizes

The results for the Stdsize measure are shown in Figure 7.6. The cut method performs the worst, which the flip method improves a bit. This is probably again for the same reasons already mentioned in the previous two sections about the Maxsize and Minsize quality measures.

The interesting part is how the shortest-path methods perform. The bare shortest-path method already performs better than the cut method, and usually better than the cut-and-flip method as well. The redistribution method then improves on this a bit after which the Euler-forward relaxation method improves the mesh a lot. The Euler-forward relaxation method is thus the best choice considering the Stdsize measure, performing quite a bit better than all other considered methods, even including the improperly fitted cos-times-sin test-function.

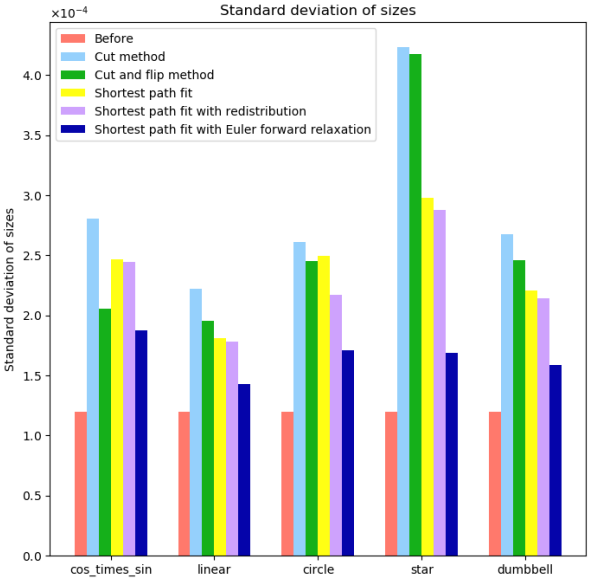

Figure 7.6: The-standard-deviation-of-sizes quality-measure of the initial mesh and of the resulting mesh after application of the different methods applied to each of the test functions. The mesh edge-length of the starting mesh is $\frac{1}{16}$.

7.7 Conclusion

When the shortest-path function works correctly,[2] the shortest-path-fit-with-Euler-relaxation method performs the best in all measures except the Avgskew measure. So unless the Avgskew measure is the most important in the problem to which the mesh is applied, or the shortest-path method cannot yet be used for such a problem, then the use of this method is much preferred. The two refinement methods of the shortest-path method show a large improvement on the base method, such that one should always use the shortest-path method with its refinements, never without.

When a shortest-path method is not an option, the cut-and-flip method is preferred instead of the cut method, since it only performs worse than the cut method in the Maxsize measure.

[2]See the introductory paragraph of this chapter.

Chapter 8

Discussion and future research

This chapter describes the conclusions from this research in Section 8.1, along with discussions and thoughts on further research on mesh adjustment in Sections 8.2 to 8.6.

8.1 Conclusion

This research has analysed the cut method extensively in Chapter 4 after which two other modifications were presented, which were shown in Chapter 7 to improve the performance of the method. This resulted in the conclusion that the cut-and-flip method, when implemented as described in this research, performs better than the other existing methods. (Primarily better than the cut method.)

After this a new method was developed, using a shortest-path algorithm and a spring model. Three variants of this method were developed, of which the third one (including the redistribution method as well as the Euler-forward mesh relaxation method) proved to perform best, most notably much better than the cut-and-flip method. From this the conclusion follows that when possible the shortest-path-with-Euler-forward-relaxation method should be applied. The current implementation of the shortest-path method is limited in its application, however, as described in the next sections.

8.2 Different mesh topologies

Although many methods developed and used in this research are probably compatible with different mesh topologies, this has not been tested and may give some complications. In this research only a simple square topology has been used, with a standard Delauney triangulation. If a different mesh topology is used where many points along the boundary are corners, then the situation where new edges would have to be added to fit the mesh properly to the zero level-set curve could easily occur. The shortest-path method presented in this

report is not able to decide to add new edges in such cases and others, where new edges are needed to fit properly.

8.3 Complicated level-set topologies

The topology of the zero level-set curves is still a problem. The present algorithm cannot deal with complications like intersections of more than two zero level-set curves, or even of two curves as happened using the cos-times-sin test-function. Also the present algorithm cannot detect multiple simple curves not crossing the boundary, it now just assumes that there is only one. Of course combinations of curves crossing the boundary and not crossing the boundary give complications as well.

A possible solution could be to use a different approach to find this 'shortest path' by instead finding the dual graph of the mesh, and in this graph finding the minimal cut between a point in the interior of a region and another point in the exterior. (Where the interior is for example where the level-set function is negative and the exterior where it is positive.) Each line in the dual graph then still represents an edge, but connects the two faces which it neighbours, so each point in the dual graph corresponds to a triangle in the original graph. In this way the edges can still have the same weight as in the original, and a minimal cut algorithm can find which edges to choose.

A complication may lie in how to choose the points to separate using the min cut algorithm. For example the dumbbell test-function might be unsuited to a min-cut algorithm since choosing a point in the left part could cause the right part to be excluded from the path by the algorithm. The upside to this method would probably be that it would be easier to make it work for multiple disjunct zero level-set curves, since finding appropriate internal and external points of a region for the min-cut algorithm are probably more easily found than appropriate start points on the edge of the region for the shortest-path algorithm.

8.4 Refining the mesh relaxation method

The mesh-relaxation method can be made more mathematically precise by modelling the spring system with a finite-element method. The downsize may be the calculation time, since this whole finite-element method would have to be modelled until equilibrium for each step where the zero level-set curve changes.

Secondly the model for the mesh relaxation can be changed to a continuous elastic model, where the mesh area would be modelled as a two-dimensional elastic substance, perhaps like rubber, on which the mesh points would be 'drawn'. The surface would then be forced to move its points according to the orthogonal projection in the shortest path method. Then the rest of the point locations would be found by finding the displacements using a solution for a finite-element method for this rubber material.

8.5 Higher dimensions

An obvious way to extend the methods developed in this research is to use more-dimensional problems, such as 3-dimensional ice melting. The problem with this is that the shortest-path algorithm cannot be used to find a least-weight surface in 3 dimensions. (The phase-interface is now always a surface instead of a curve in 2-dimensions.) So a new algorithm has to be developed with a different approach.

If one wants to keep using the equivalent of edge weights (which are triangle weights in 3 dimensions) then perhaps one can use the earlier mentioned minimal-cut approach to the dual graph. (Which is now in 3 dimensions, so dual with respect to the elements, not the faces of the elements. Each line in this dual graph then corresponds with a face between two elements in the original graph.) This can work because a minimal cut does not discriminate between the number of dimensions of the mesh, it just gives the minimal cut in the dual graph whatever this represents in the original graph/mesh.

8.6 Quantifying goodness-of-fit to zero level-set curve

In this research the goodness-of-fit to the zero level-set curve has not been quantified. This has been done by Verbeek [2] and gives a better perspective on the performance of the different methods. The redistribution method from Section 5.4 is likely to have a positive influence on the performance with respect to such a measure. Possibly the orthogonal projection method can be adjusted based on this measure such that redistribution method becomes obsolete.

Bibliography

[1] J.A. Sethian, *Curvature and Evolution of Fronts*, 1985.

[2] Thijs Verbeek, *Modelling and Simulating Three Phases of Steel*, Technische Universiteit Delft, Delft, 2018.

[3] Dennis den Ouden, *Mathematical Modelling of Nucleating and Growing Precipitates: Distributions and Interfaces*, Technische Universiteit Delft, Delft 2015.

[4] E. Javierre, *Numerical methods for vector Stefan models of solid-state alloys*, Technische Universiteit Delft, Delft 2006.

[5] SharcNet, *Skewness*, retrieved on May, 2018 from https://www.sharcnet.ca.

[6] Math Open Reference, *Heron's Formula for the area of a triangle*, retrieved on May, 2018 from https://www.mathopenref.com.

[7] Tom M. Apostol, Mamikon A. Mnatsakian, *New Horizons in Geometry*, Mathematical Association of America, 2012.

[8] Wikipedia, *Delaunay triangulation*, retrieved on August, 2018 from https://en.wikipedia.org/wiki/Delaunay_triangulation.

[9] John A. Rice, *Mathematical Statistics and Data Analysis*, 3rd edition, Brooks/Cole, 2007.

[10] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, *Introduction to Algorithms*, 2nd edition, McGraw-Hill, 2001.

[11] Wikipedia, *Dijkstra's algorithm*, retrieved on September, 2018 from https://en.wikipedia.org/wiki/Dijkstra's_algorithm.

[12] William J. Cook, William H. Cunningham, William R. Pulleyblank, Alexander Schrijver, *Combinatorial Optimization*, Wiley, 1998.

[13] Jiang, Chen, *Parametric structural shape & tolopogy optimization with a variational distance-regularized level-set method*, Computer Methods in Applied Mechanics and Engineering, 2017.

[14] Lu, Tianye, *Lecture 12: Discrete Laplacian*, Stanford University, Stanford. Retrieved on September, 2018 from https://graphics.stanford.edu.

Appendices

Appendix A

Pictures of the methods applied to the test-functions

What follows are pictures of the test functions as they are applied to a triangular grid on a square domain $[-1, 1]^2 \in \mathbb{R}^2$, with an edge-length equal to $h = \frac{1}{8}$. This is somewhat smaller than we would like considering the errors, but it shows the applied methods more clearly.

In the figures of this chapter, the situation before adjusting the mesh is depicted to the left of the situation after the method has been applied. In the figure depicting the situation before the application the red lines depict the zero edges where the zero level-set curve crosses a mesh edge. In blue are shown the lines of the zero level-set line as approximated by linearly interpolating the crossing of the zero level-set curve. These interpolations can be considered to be exact since we usually do not have the actual level-set function present, but only the triangular approximation by its values on the grid points. In this case the (approximated) level-set function is linear on each triangle, such that linear interpolation gives an 'exact' answer.

A.1 Cut method

This method uses only the move and the cut techniques as described in Chapter 4. Figures A.1 and A.2 show this method in action. The green lines represent the (approximated) level-set edges that now coincide with edges of the mesh. The move technique consists of moving a grid point towards an interpolated level-set crossing of a neighbouring edge when the distance is less than $0.3h$.[2][1] The points that have moved are shown in orange.

The cut technique splits a triangle in multiple triangles in such a way that these triangles fit the zero level-set curve better. Depending on the type of cut, a triangle will split up in two or three parts, where the results described in Section 4.3 are used to decide how the triangle is cut. The triangles that have been cut are shown in the following figures in yellow.

[1] The discussion on the optimal threshold is extensively treated in [2, Quality and Accuracy of the Mesh, p. 48].

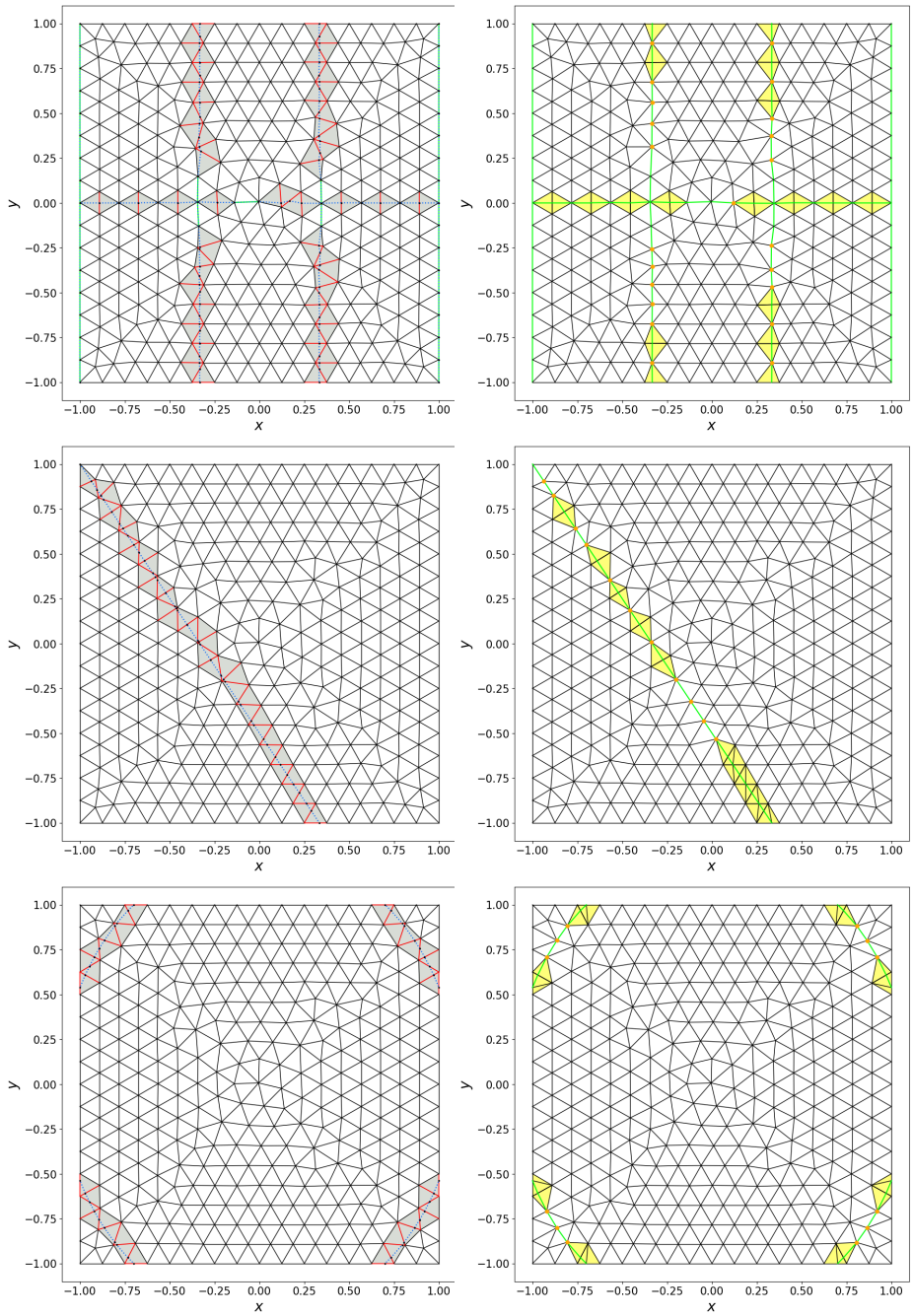

Figure A.1: The meshfitting method applied to the first three test functions. The method applied here is the cut method, which moves points and cuts up triangles to make the mesh fit the zero level-set curve. Orange points depict points that have been moved, while yellow triangles depict triangles that have been cut into pieces.

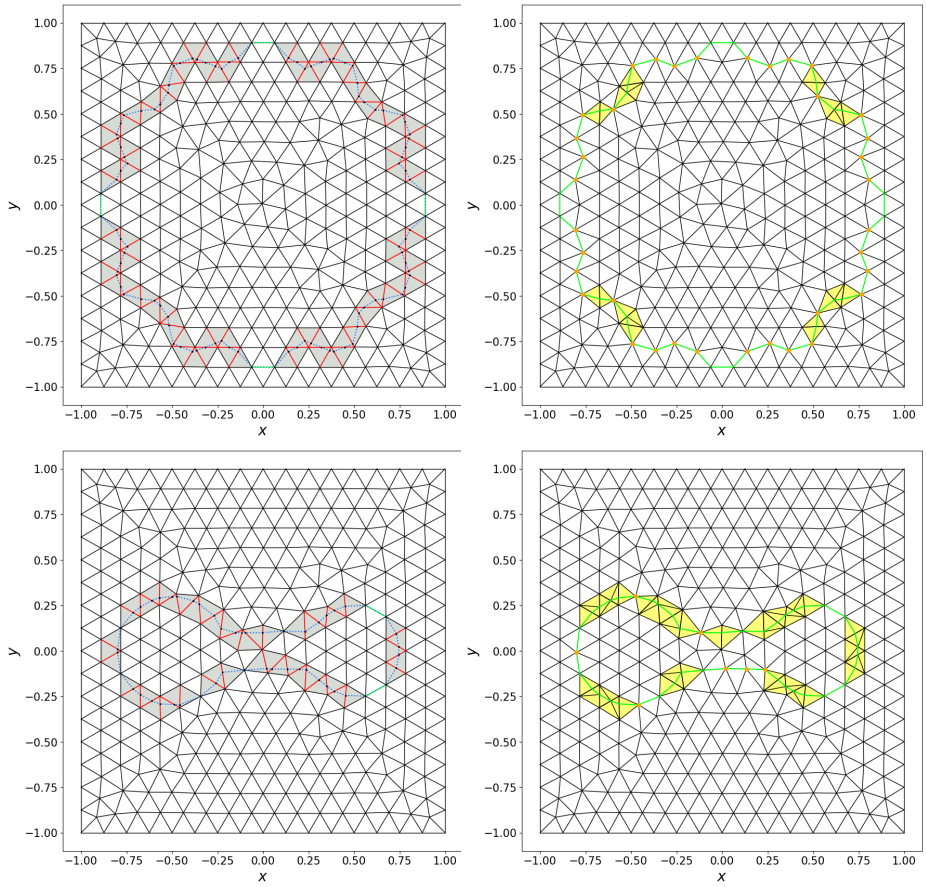

Figure A.2: The meshfitting method applied to the last two test functions. The method applied here is the cut method, which moves points and cuts up triangles to make the mesh fit the zero level-set curve. Orange points depict points that have been moved, while yellow triangles depict triangles that have been cut into pieces.

A.2 Cut-and-flip method

This method uses the move, cut and the flip techniques as described in Chapter 4. Figures A.3 and A.4 show this method in action. The green lines again represent the (approximated) zero level-set curves that now coincide with edges of the mesh. The move technique consists of moving a grid point towards an interpolated level-set crossing of a neighbouring edge when the distance is less than $0.3h$.[2][2] The points that have moved are shown in the figures below in orange.

The cut technique splits a triangle in multiple triangles in such a way that these triangles fit the zero level-set curve better. Depending on the type of cut, a triangle will split up in two or three parts, where the results described in Section 4.3 are used to decide how the triangle is cut. The triangles that have been cut are shown in the following figures in yellow.

Whenever possible, the cut-and-flip method applies a flip instead of a cut, meaning that instead of creating a lot of new triangles, two neighbouring triangles flip their common edge, such that the flipped edge now connects the other two of the four points of the two triangles. Where this technique has been applied, both triangles involved are shown in light blue.

[2]The discussion on the optimal threshold is extensively treated in [2, Quality and Accuracy of the Mesh, p. 48].

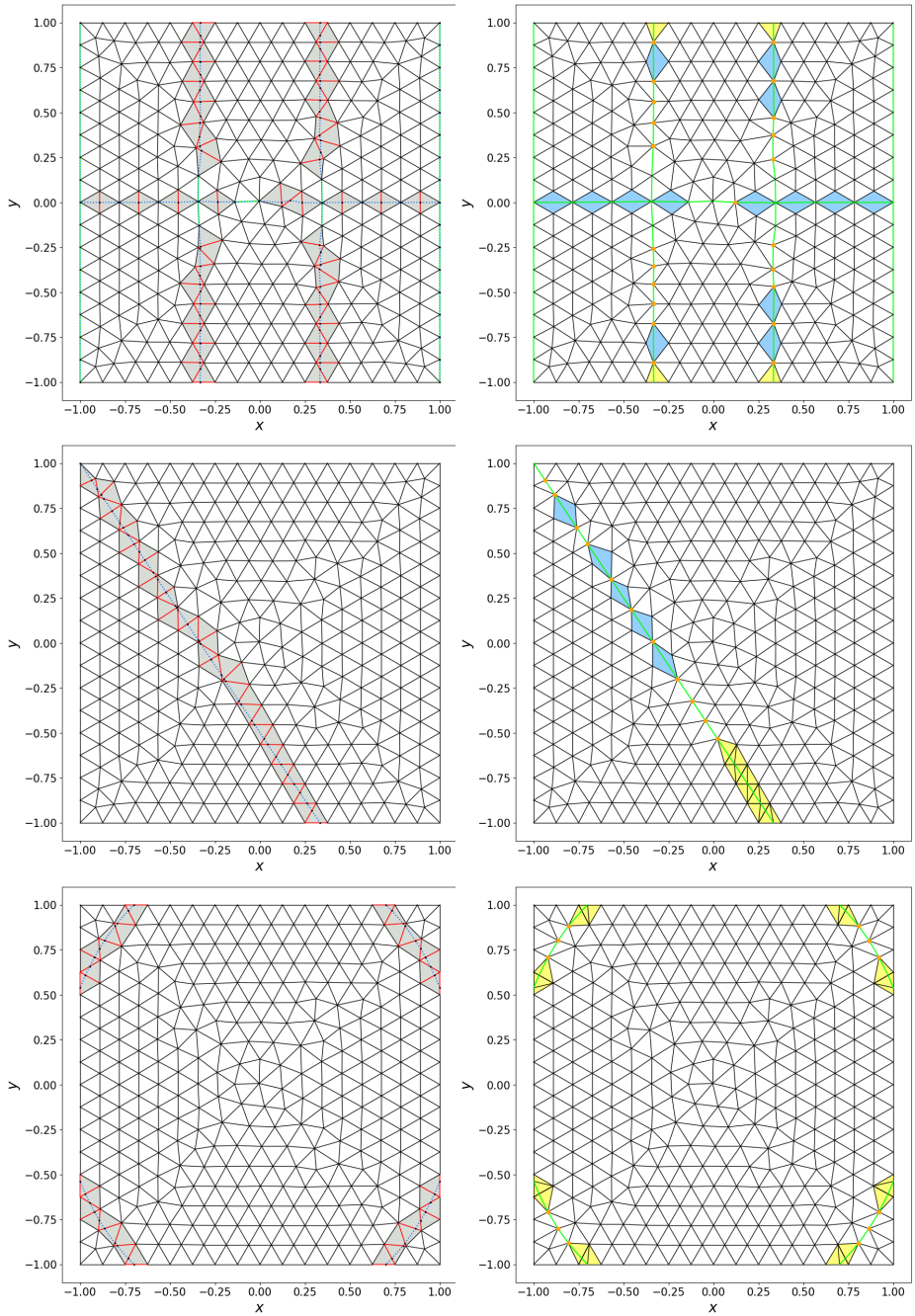

Figure A.3: The meshfitting method applied to the first three test functions. The method applied here is the cut-and-flip method, which moves points, flips edges when possible or else cuts up triangles to make the mesh fit the zero level-set curve. Orange points depict points that have been moved, yellow triangles depict triangles that have been cut into pieces and blue triangles are those whose edges have been flipped.

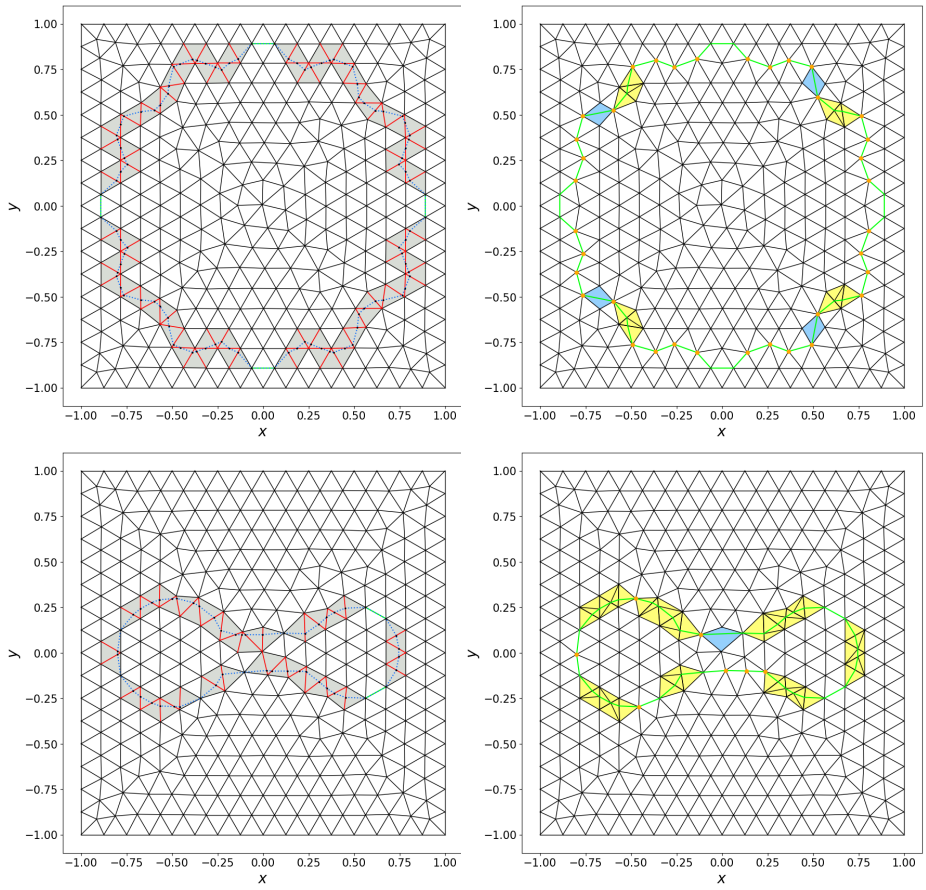

Figure A.4: The meshfitting method applied to the last two test functions. The method applied here is the cut-and-flip method, which moves points, flips edges when possible or else cuts up triangles to make the mesh fit the zero level-set curve. Orange points depict points that have been moved, yellow triangles depict triangles that have been cut into pieces and blue triangles are those whose edges have been flipped.

A.3 Shortest path method with orthogonal fit

This method uses the a shortest-path algorithm (using Dijkstra's algorithm to calculate the shortest path), where the edge weights are determined by their position and orientation relative to the closest level-set edges. Figures A.5, A.6 and A.7 show this method in action. When these paths are found they are projected orthogonally towards the closest level-set edges. Then they are redistributed by placing each point in the average position of its two neighbouring points within the chosen path, and afterwards are again projected to the nearest zero level-set edge.

The plots also show the choices for start points and start-point sets used in the shortest-path algorithm, as well as the edge weights, using the colouring of the points and edges respectively. The plots with coloured triangles also show the skewness and size mismatch in the amount of green and red in the triangle colour respectively.

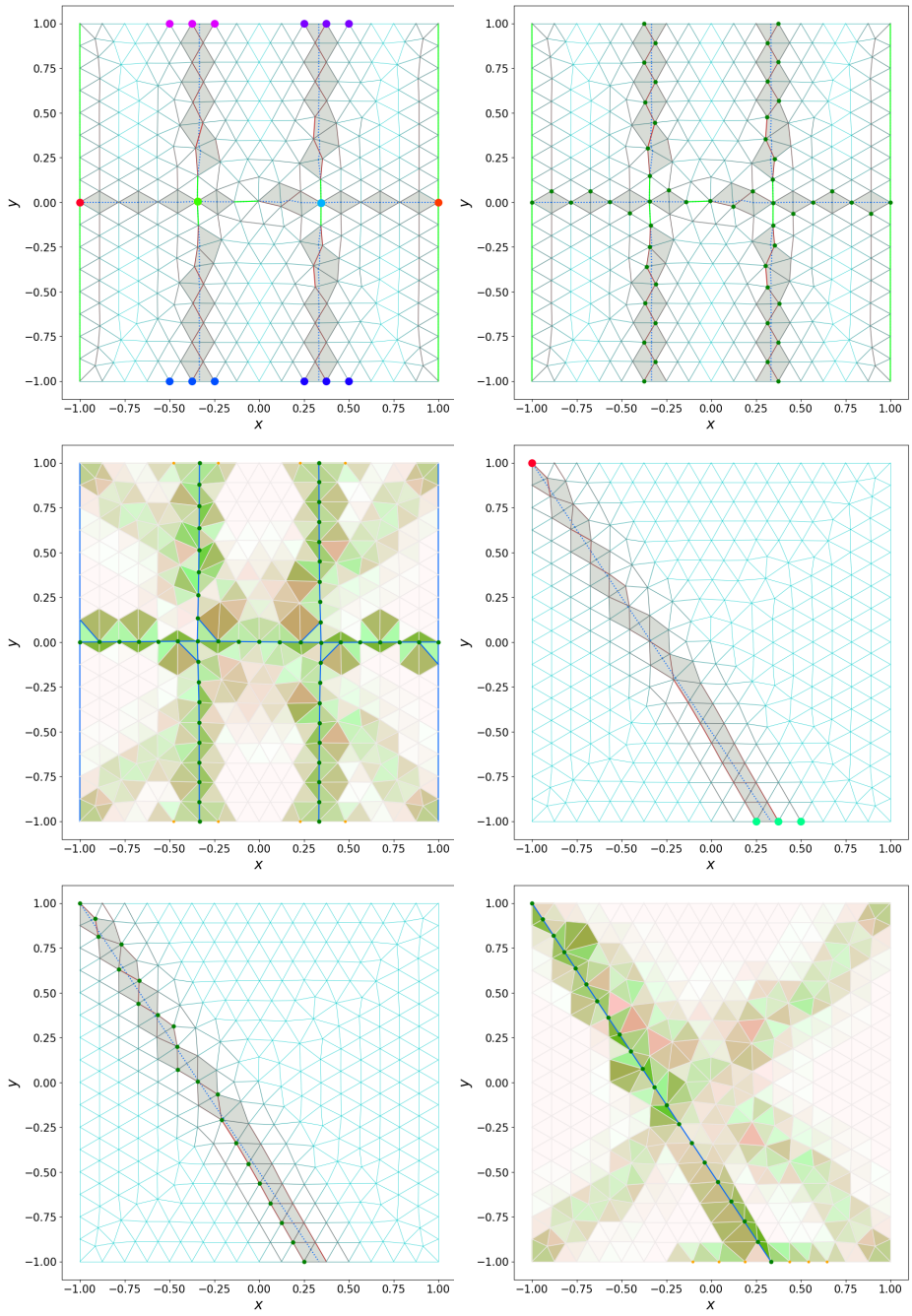

Figure A.5: The shortest path method applied to the first two test functions. The shortest-path method first decides which points will be moved towards the zero level-set curve and then moves them with an orthogonal projection method. The colored points in the first plot of each three depict the start-point sets from which the shortest path starts or and to calculate. The green points in the second plot of each three shows the selected points based on the shortest-path algorithm, which are in plot three of the three moved orthogonally towards the zero level-set curve. In this last plot the redness of each triangle depicts the skewness of the triangle.

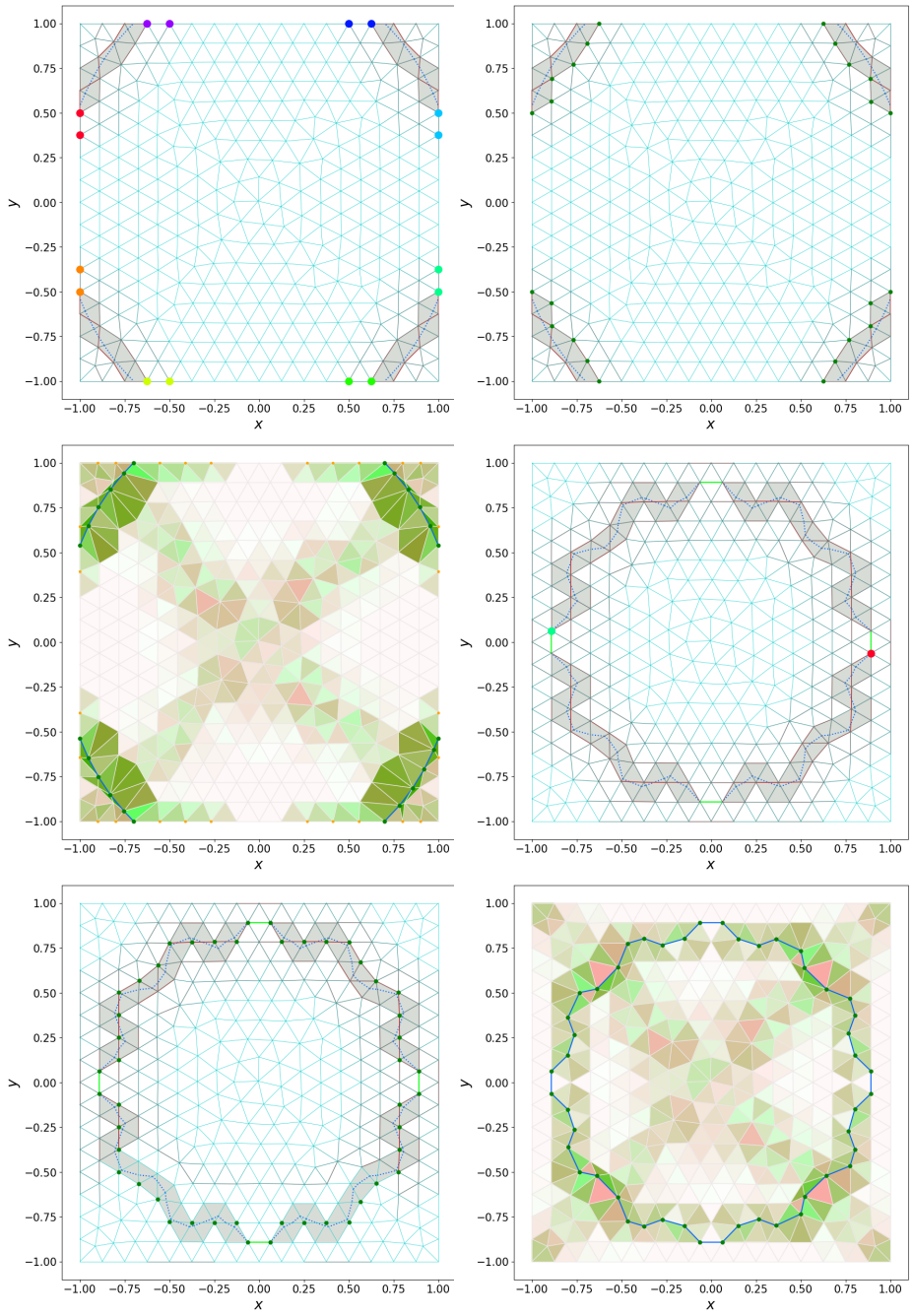

Figure A.6: The shortest-path method applied to the third and fourth test functions. The shortest-path method first decides which points will be moved towards the zero level-set curve and then moves them with an orthogonal-projection method. The colored points in the first plot of each three depict the start-point sets from which the shortest path starts or and to calculate. The green points in the second plot of each three shows the selected points based on the shortest-path algorithm, which are in plot three of the three moved orthogonally towards the zero level-set curve. In this last plot the redness of each triangle depicts the skewness of the triangle.

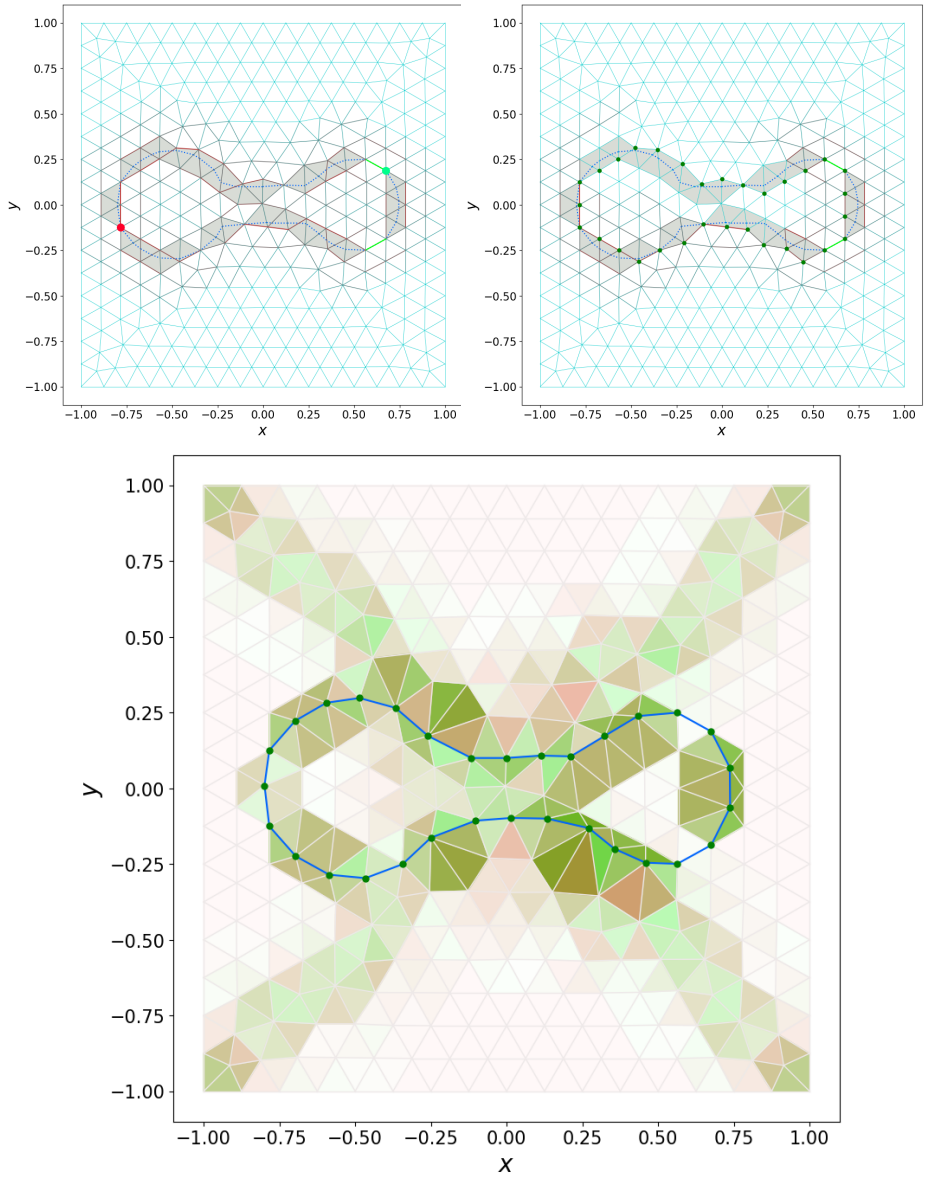

Figure A.7: The shortest-path method applied to the fifth test function. The shortest-path method first decides which points will be moved towards the zero level-set curve and then moves them with an orthogonal-projection method. The colored points in the first plot of each three depict the start-point sets from which the shortest path starts or and to calculate. The green points in the second plot of each three shows the selected points based on the shortest-path algorithm, which are in plot three of the three moved orthogonally towards the zero level-set curve. In this last plot the redness of each triangle depicts the skewness of the triangle.

A.4 Shortest path method with orthogonal fit and mesh relaxation

The last method shown here is simply the shortest-path method with orthogonal fit described in the previous section, after which the mesh-relaxation method was applied. Figures A.8 and A.9 show this method in action.

The mesh-relaxation method uses a spring model to 'relax' the mesh-points into better positions, such that the mesh quality measures are improved greatly. The plot show the situations before and after application of the mesh relaxation method, where the triangle skewness and size mismatch is depicted as the amount of green and red, respectively, in the colour of the triangle.

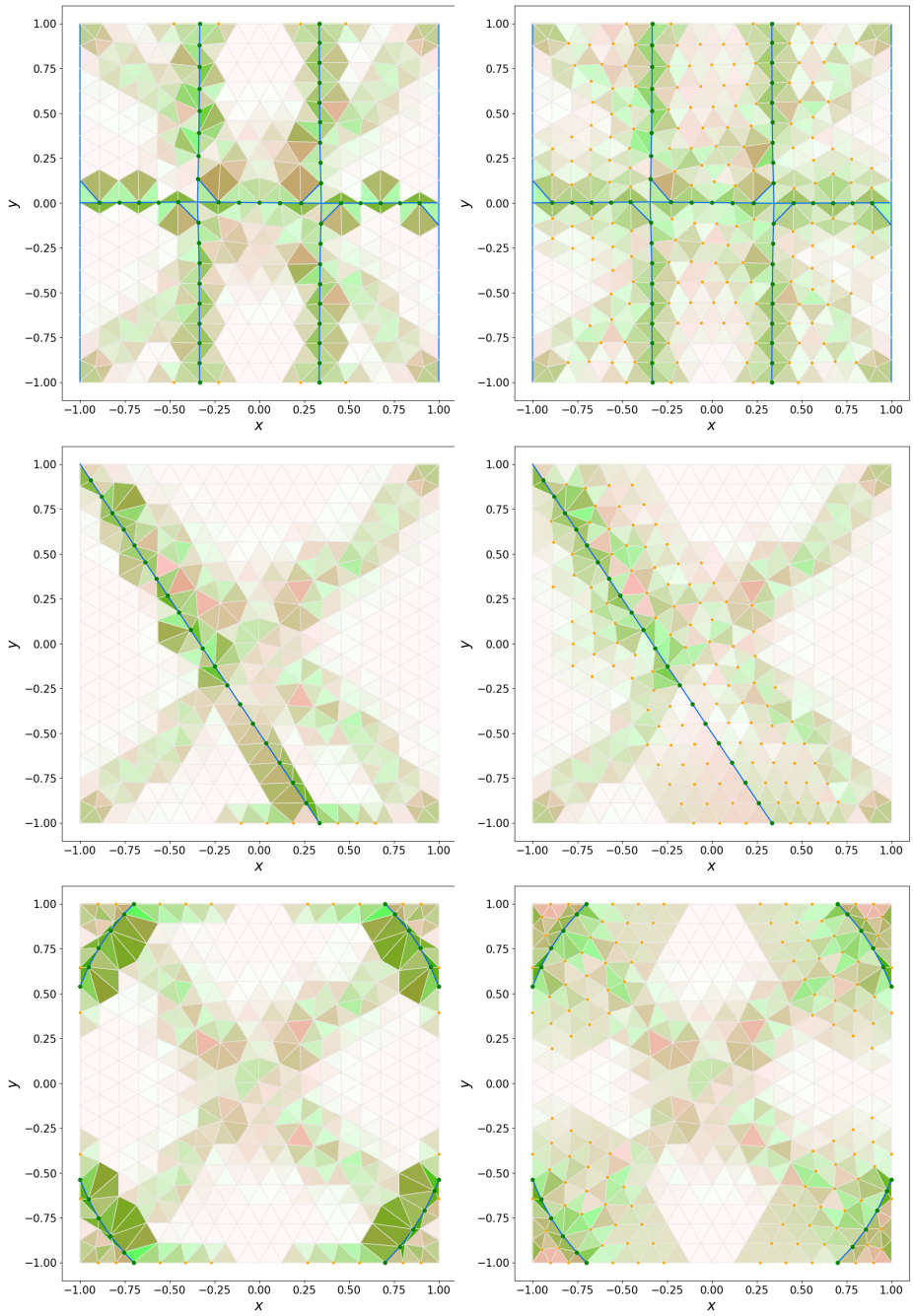

Figure A.8: Before (left) and after (right) mesh relaxation using the spring model for the first three test functions. The amount of green and red depict the amount of skewness and mismatch in size, respectively.

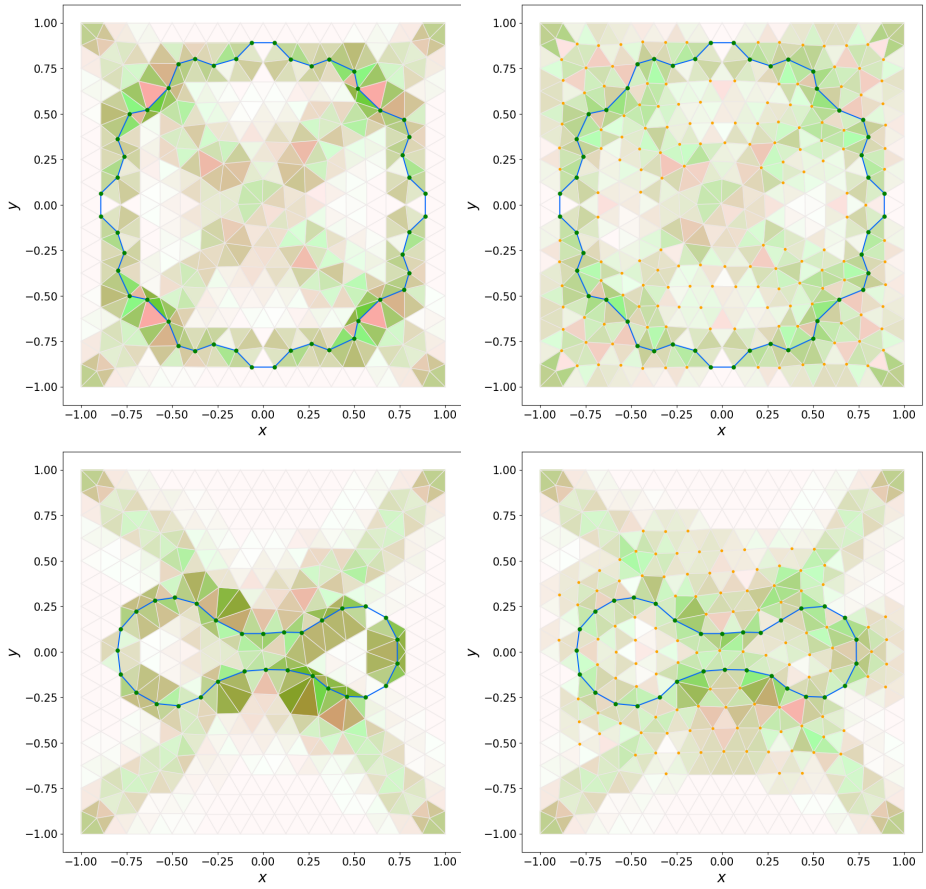

Figure A.9: Before (left) and after (right) mesh relaxation using the spring model for the last two test functions. The amount of green and red depict the amount of skewness and mismatch in size, respectively.