Iterative local remeshing for locally-injective deformations

Master Thesis

Antoine Demont

University of Fribourg

May 2024

Abstract

To run simulations or model three-dimensional objects, we need a way to represent them. This is achieved using a mesh, whose vertices collectively form the shape of the object. When subjected to deformations, meshes can develop flat or inverted elements. To address this issue, we perform remeshing on the represented object, which can be done either globally or locally. Global remeshing generates a new mesh that is best suited to the deformed object, whereas local remeshing focuses on identifying and improving the critical elements near the problem areas. In this work, we present an implementation of a local remeshing algorithm for locally-injective deformations, in which each remeshing operation can be reverted. This allows us to restore the mesh to its initial state and topology after any deformation.The remeshing process involves identifying the problematic elements and addressing each through four iterative passes: *topological, contraction, insertion*, and *smoothing* pass. Each pass aims to enhance the mesh quality through multiple techniques such as edge and face removal, edge contraction, vertex insertion, or smoothing.

Prof. Dr. David Bommes, Computer Graphics Group, University of Bern, Supervisor Valentin Nigolian, Computer Graphics Group, University of Bern, Assistant

Contents

1 Introduction **4**

2 Related work **5**
2.1 World space and material space 6
2.2 Types of Meshes 6
2.3 Hill climbing algorithms 7

3 Mesh deformations **9**
3.1 Mesh evaluation 9
3.1.1 Triangular meshes 9
3.1.2 Tetrahedral meshes 10
3.2 Bad elements 10

4 Remeshing **12**
4.1 General remeshing algorithm 12
4.2 Control mesh 14
4.3 Triangular meshes 14
4.3.1 Topological pass 14
4.3.2 Insertion pass 15
4.3.3 Smoothing pass 16
4.4 Tetrahedral meshes 17
4.4.1 Topological pass 17
4.4.2 Insertion pass 20
4.4.2.1 Center of Chebyshev 20
4.4.2.2 Insertion algorithm 20
4.4.3 Smoothing pass 22

5 Results **24**
5.1 Triangular meshes 24
5.1.1 Stretch 24
5.1.2 Compress 25
5.1.3 Finding the minimal quality 26
5.2 Tetrahedral meshes 27
5.2.1 Spin 27
5.2.1.1 Finding the optimal threshold 28
5.2.1.2 Finding the optimal number of timesteps 29
5.2.1.3 Handling greater deformations 31
5.2.2 Stretch 32
5.2.2.1 Finding the optimal threshold 33
5.2.2.2 Finding the optimal number of timesteps 34

5.2.2.3 Handling bigger deformations 37
5.2.3 Impact of the control mesh 38
5.2.4 Reverse experiment 39

6 Conclusion **40**
6.1 Weaknesses and future work 40
6.2 Use of AI tools 41

1 Introduction

At first, the process of making cartoons consisted of drawing successive pictures and sequencing them to create the illusion of movement. Immobile elements could be reused between frames, but any element in motion had to be redrawn for every movement. Once computer-generated images and 3D modeling came forth, following this method would be naive and time-inefficient. In an animated movie, it would be prohibitively expensive for an animator to remodel a character each time the character moves a finger or opens their mouth to speak. Fortunately, unlike static drawings, three-dimensional objects can be altered by manipulating their geometry. This introduces the concept of *mesh deformation*, where we alter the shape of a 3D object by moving the vertices that compose it.

Say we want to perform a physical simulation of an iron bar bending. To represent it, we use a tetrahedral mesh, which will hold data about both the surface and the inside of the object. As the bar bends, the vertex distribution will stray from its initial representation. This could cause some of them to create tetrahedra of bad quality or inverted elements. If we were to use the mesh to compute physical properties, having such irregularities could not translate from the simulation to the real world and would inevitably produce incorrect results. So, we observe that the space of possible deformations is constrained by the mesh used.

To mitigate it, we need to operate on the mesh with the intent of matching more closely to the new shape of the object. This operation is called *remeshing*. In this project, we will present an implementation of a local remeshing algorithm for locally-injective deformations. Meaning it is responsible to, in a first step, identify the conflicting elements of the mesh, then operate on the mesh to either remove them or increase their quality to an acceptable level for each one successively. Lastly, any remeshing operations or deformations on the mesh can be reverted to a previous state.

In Chapter 2, we will first introduce the process to ensure bijection on the operations as well as the types of mesh used in the project. We will explain the concept of hill climbing algorithms and then, in Chapter 3, present how to grade mesh quality during deformation. In Chapter 4 we go into more detail about the implementation of the algorithm in both two and three dimensions. Finally, in Chapter 5, we evaluate its performances on meshes subjects to varying deformations along with discussing the optimal setting of parameters.

2
Related work

In this chapter, we will briefly present the paper we based this project on, as well as discuss the work related to it. This contains the topic of remeshing, the introduction of a different space mapping for meshes, the two types of mesh used for object representation and the concept of hill climbing methods.

This project is based on the work of Wicke et al.[10], who describe the implementation of a dynamic local remeshing algorithm and its use in physics-based simulations of elastoplastic materials.

Remeshing is a wide field, nonetheless, we can identify two main approaches to the problem: global or local remeshing. The first consist of remeshing the entire domain when critical conditions are met. This approach is used in the work of Bargteil et al.[2] as well as Wojtan and Turk[11]. A disadvantage to this way of tackling remeshing is, that regardless of the quality of the "good" elements, the complete mesh must be rebuilt from the ground up every time it is performed, inducing inefficient operations. This inefficiency is particularly evident when only a local area of the mesh is in poor condition while the rest remains in good or acceptable shape. Wicke et al. [10] also state that global remeshing accumulates large numerical errors in simulations, called *artificial diffusion*, due to the need to resample physical properties from an old to a new mesh. By changing completely the mesh at each remeshing step, the diffusion of physical properties, for example, the transmission of forces, can differ from a natural path due to the changes in the mesh topology. [10] show in their simulations that some element of the strain field exaggerated plastic-like behavior of purely elastic objects due to the *artificial diffusion*.

In the second approach, local remeshing, we aim to identify and replace only the low-quality elements from the mesh. By minimizing the changes on the mesh, we reduce the *artificial diffusion* mentioned before. Since most of the mesh is left untouched, its topology is similar for each deformation step, and thus the transmission of physical properties flows through more constantly. Being a sum, having as few changes as possible keeps the *artificial diffusion* to the lowest. Additionally, with local remeshing, coarsening the mesh where it is unnecessarily detailed or inversely adding details in smaller parts of the mesh is automatically done as these regions are more likely to create bad elements. Thus creating a better match between the shape of the object and the mesh.

2.1 World space and material space

To ensure the bijection of all operations, we copy the mesh into a world space representation and a material space representation (fig. 2.1). In the world space will be applied all the deformations of the simulation whereas the material space will remain unchanged. Each remeshing operation is applied to both spaces and has to be valid for each of them. This bijective state guarantees that any operation can be reverted. Making it possible to reset the world space mesh to its initial state, simulate elasticity, or any other temporary deformation.

In this regard, this project deviates from the original implementation by Wicke et al. [10]. The goal of their program was to conduct elastoplastic simulations. An elastic material will retrieve its original shape after being subjected to a deformation. Being able to revert operations was the key point to represent elasticity.

We extended this relation between world space and material space to other deformations. Rather than simulating physical properties, we split the deformation into successive predetermined positions and performed the remeshing based on them. With this, we can extend this method to continuous deformations, where we go through successive positions to reach a final shape. Here, being able to revert operations allows one to navigate freely between the deformation steps performed.

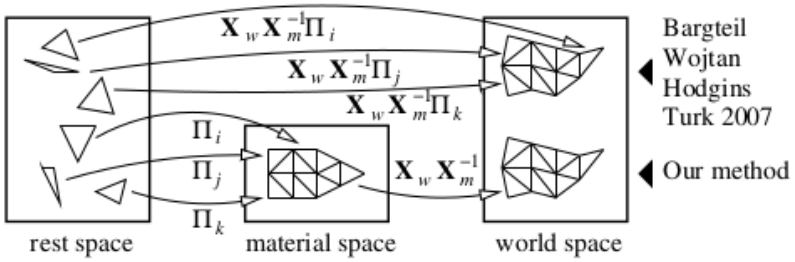


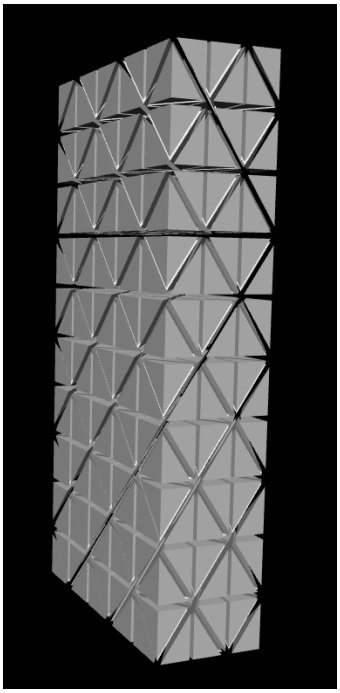
Figure 2.1: Relation between material and world space, taken from [10]. Deformations occur in the world space. Remeshing is performed in both material and world space.

2.2 Types of Meshes

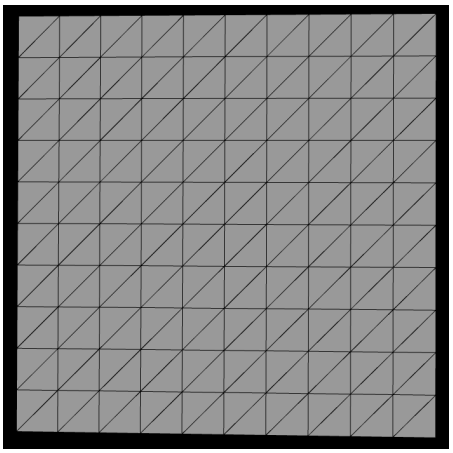
Wicke et al.[10] is using a tetrahedral mesh for the representation of 3-dimensional objects. In this work, we will adapt their algorithm to both 2-dimensional and 3-dimensional meshes.

For the 2-dimensional representation, we work on a flat triangular mesh composed of a plane split into triangles. For 3D meshes, the triangles used in 2D are replaced by tetrahedra. A visual comparison of the two mesh types is shown in figure 2.2.

For simulations, the difference in dimensions means that an object represented using a triangular mesh will only be composed of its surface, making it completely hollow, whereas one using a tetrahedral mesh is filled. Since a 2-dimensional mesh does not have to care about the inside of the object, it is a more lightweight and simple means of representing it. For accurate results, using a 3-dimensional mesh is recommended as it can model different properties between the surface and the inside of the object.



(a) A flat tetrahedral mesh



(b) A flat triangular mesh

Figure 2.2: The two types of meshes used in this project. A flat tetrahedral mesh has a height, width and depth. A flat triangle mesh is only composed of a height and width.

2.3 Hill climbing algorithms

The original paper uses a hill climbing method. It involves starting from a current state and iteratively attempting to achieve the optimal state by consistently taking actions that bring us closer to it. For example, to find the top of a hill we start from its foot and only take the steps allowing us to reach a higher altitude until all the steps we can take lead us downwards, meaning we have reached the summit.

This is a simple way of finding an optimum but suffers from one clear flaw: local optima. When taking the curve in figure 2.3 we see that the clear maximum is situated at point C. But if our only knowledge is the gain we get from our next step, then, starting from A, the first step we take determines whether our efforts are in vain or lead us to the global maximum. By going in the negative direction of the x-axis we will inevitably reach the peak B and falsely set it as the maximum.

This issue can be mitigated by performing multiple attempts. To go back to the literal interpretation, we take 100 hikers with us who will help us reach the highest hill in the region. But if only one hiker reaches the actual maximum, this means that 99 of them did all their work for nothing. This overhead is something to be tuned to match the need between accuracy and performance.

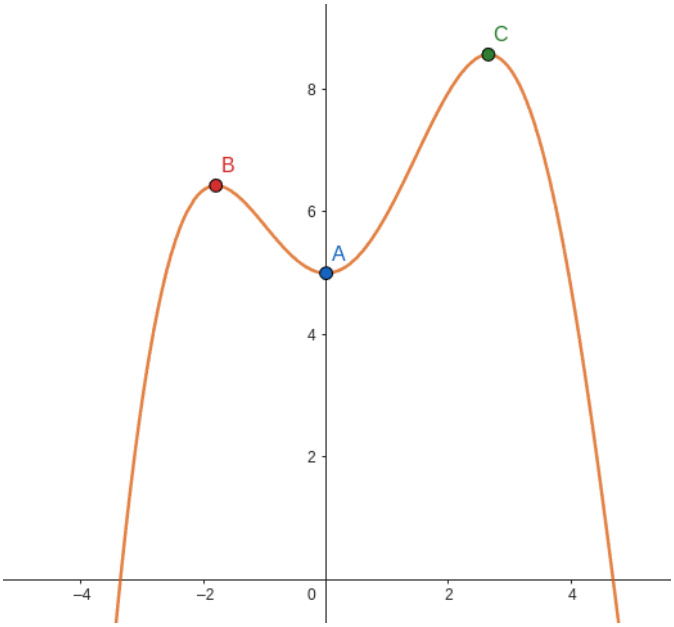


Figure 2.3: Function with a local (B) and global (C) maximum. During remeshing, with mesh quality A, we can improve it by performing an operation o_1 leading us to B, a local maximum, or an operation o_2 leading us to C, the actual maximum. Without knowledge of the future state of the mesh, both operations are treated as the best solution.

3
Mesh deformations

This chapter presents the different metrics used for evaluating the quality of a mesh and its elements, both in two and three dimensions. It also illustrates the occurrence of bad elements in a mesh as well as a first insight into how to treat them.

3.1 Mesh evaluation

For our mesh, we want to have all its elements as close as possible to a reference shape. An element too different from it is considered to be bad. To quantify the distance distance between them we use the following methods: For triangular meshes, we adapt the formula from Wicke et al.[10] to two dimensions. For tetrahedral meshes, we introduce another metric for quality: the *symmetric dirichlet energy*.

3.1.1 Triangular meshes

In [10], the quality of a tetrahedron is equal to:

$$6\sqrt{2}V\frac{\ell_{harm}}{\ell_{rms}^4} \tag{3.1}$$

with V the volume, ℓ_{harm} the *harmonic mean* of the tetrahedron's edge lengths and ℓ_{rms} the *root-mean-squared* edge length. [10] say this measure is bounded in $[0, 1]$, we can assume that negative volumes are treated as having a quality of 0. A quality of 1 is attributed to an equilateral tetrahedron and 0 to a degenerate one, where all the vertices are coplanar.
We find a similar approach for triangles in [4] by Freitag. The difference of an element with a reference equilateral triangle of side length 1 is measured by:

$$4\sqrt{3}A\frac{1}{\sum_{i=1}^{3} l_i^2} \tag{3.2}$$

with, similar to eq. 3.1, A the area and l_i the ith edge of the triangle. Equivalent to eq. 3.1, a quality of 1 means an equilateral triangle and 0 for a degenerate triangle. To detect inverted triangles, we compute the signed area of the triangle such that any value of $A < 0$ indicates an inverted triangle.

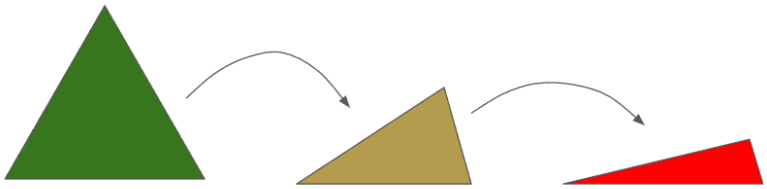


Figure 3.1: Triangle losing quality. As we deform it, the difference with the reference (in green) increases, resulting in a lower *quality*.

3.1.2 Tetrahedral meshes

For tetrahedral meshes, we chose a different approach to measuring the quality of the elements constituting the mesh. For any tetrahedron t, the *symmetric dirichlet energy* measures how far t is from a reference shape. In [9], the dirichlet energy is presented as a metric that evaluates the intrinsic stretching between a "source" M_0 and a "target" M volume in $\mathbb{R}^3$. We adapt this approach to tetrahedra by taking a tetrahedron from the mesh and computing the dirichlet energy with a perfect regular tetrahedron.

In mathematical terms, this translates to

$$E_{dirichlet}(t) = \|\mathcal{J}_t\|^2 + \|\mathcal{J}_t^{-1}\|^2 - 6 \tag{3.3}$$

with $\mathcal{J}_t$ the *jacobian matrix* of the tetrahedron t scaled by a coefficient of the volume, to reduce the penalty on smaller regular tetrahedra.

Note that the reference tetrahedron can be adapted. For example, one could set the reference tetrahedron's value to the best of the mesh before any deformation is applied, thus measuring more closely the distortion between the current and the initial state of the mesh.

With this method, a perfect element, meaning an element that has no stretching with respect to the reference, has an energy of 0. For any other element, we invert the positive value given by the *dirichlet energy*. With this, the bigger the energy, the worse the quality of the element will be.

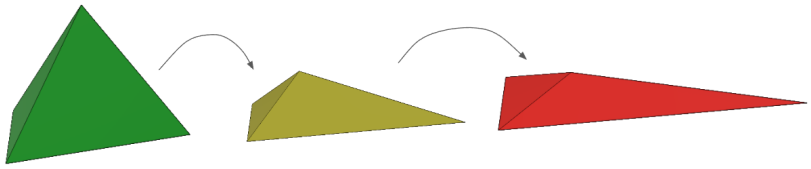


Figure 3.2: Tetrahedron gaining energy. As we deform it, the difference with the reference (in green) increases, resulting in a higher *dirichlet energy*.

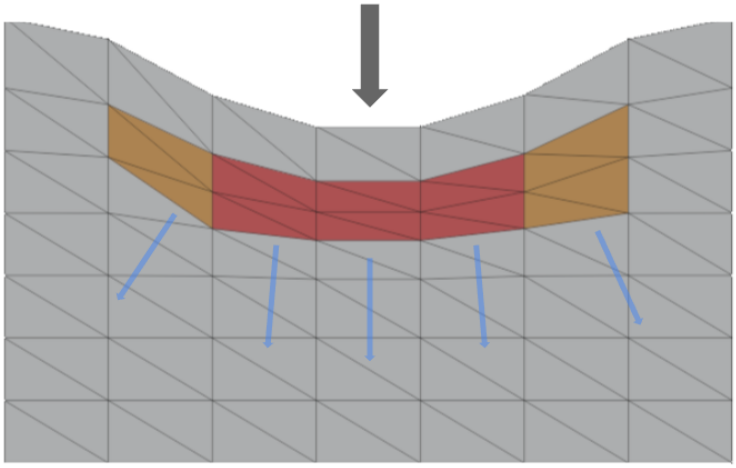
3.2 Bad elements

Now that we can identify bad elements in a mesh, we will discuss when and how those can appear. In figure 3.3a, we see that as we apply a force to the mesh, some of the vertices are getting closer to each

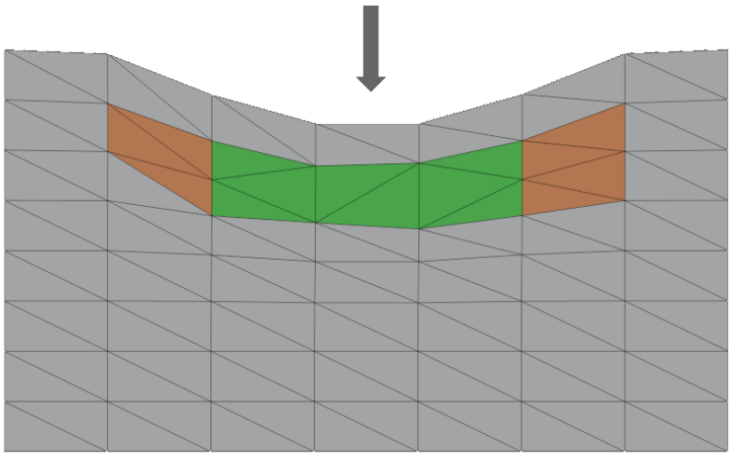
other. Continuing this deformation without any remeshing and restriction will inevitably lead to some inversions in the geometry, producing visual artifacts during render or inaccurate simulations. Since we move elements of the mesh without constraints, the vertices of some elements can move past their neighbors, first creating degenerate elements and then inverted ones.

By performing remeshing operations, we aim to keep the shape of the object but change the topology of the bad elements to stay within acceptable quality values.

Another option would be to limit the vertices' movements to avoid creating low-quality elements but this approach would mean that some deformations could not be applied to the object. Since some would inevitably create low-quality elements on the mesh.



(a) Mesh with bad elements. An external force (in gray) is applied to the mesh, compressing the top boundary and its neighboring triangles.



(b) Same mesh after local remeshing on the red elements.

Figure 3.3: Removal of low-quality elements induced by a force applied to the top boundary of the mesh (gray arrow). Elements of lower quality are highlighted in red and orange.

4
Remeshing

Chapter 4 covers the remeshing algorithm as well as its implementation in two and three dimensions.

4.1 General remeshing algorithm

The overall operation of the algorithm, presented in alg.1, goes as such: We begin by assessing the mesh quality using formulas from Chapter 3 to analyze each element. If an element's quality falls below an empirically set threshold q_{min}, we add it to a list of candidates for remeshing. After identifying all critical elements, we iteratively improve each one. This operation is described in alg. 2. After each mesh operation, we evaluate the quality of the new or altered elements. Based on it, we can decide whether to accept or revert the operation in case the quality of the resulting mesh is lower than of the one before.

Algorithm 1 Loop: The overall remeshing algorithm

Require: M a mesh to improve
  $B \leftarrow \{\}$
  for all $t \in M$ **do** ▷ Find bad elements
    $q \leftarrow ComputeQuality(t)$
    if $q < q_{min}$ **then**
      $B \leftarrow B \cup t$
    end if
  end for
  for all $b \in B$ **do** ▷ Improve mesh
    $ImproveElement(b)$
  end for

The remeshing is composed of three sub-parts, in order the *Topological*, *Contraction*, *Insertion* and *Smoothing* passes. We also see a list called A, sorted based on its elements' quality. It will reduce and expand at each pass. A contains at first only the element of bad quality we want to get rid of but will change in size after each mesh operation. Since by changing the mesh we also change the neighbors to the

starting element, an improvement can be required on those too. Note: A may not only contain elements of quality below q_{min}, Wicke et al.[10] mention that their experiences showed some bad elements can only be improved by also altering their neighbors.

Algorithm 2 ImproveElement: Mesh improvement algorithm

Require: t an element of bad quality, N the number of remeshing iterations
  function IMPROVEELEMENT(t)
    $A \leftarrow \{t\}$
    for $i \leftarrow 0$ to N **do**
      do
        $A \leftarrow TopologicalPass(A)$
        if $Quality(A) \geq q_{min}$ **then** ▷ $Quality(A)$ returns the worst element of A
          return
        end if
      while $TopologicalPass(A)$ changes the mesh
      $A \leftarrow ContractionPass(A)$
      if $Quality(A) \geq q_{min}$ **then**
        return
      end if
      $A \leftarrow InsertionPass(A)$
      if $Quality(A) \geq q_{min}$ **then**
        return
      end if
      $A \leftarrow SmoothingPass(A)$
      if $Quality(A) \geq q_{min}$ **then**
        return
      end if
    end for
  end function

  function CONTRACTIONPASS(A)
    $E \leftarrow$ set of all edges of elements in A
    for all $e \in E$ **do**
      if e still exists **then**
        attempt to contract e, smooth resulting vertex
      end if
    end for
    return the surviving elements of A and the mesh elements modified by edge contraction
  end function

To improve an element t of the mesh, alg. 2 uses an iterative process where a list of elements goes through a succession of passes to locally improve quality. Note that [10] uses a value of $N = 10$ but it was adapted in this implementation due to performance. Since their implementation differs based on the dimensionality of the mesh, the details of the *Topological*, *Insertion*, *Smoothing* passes will respectively be discussed in sections 4.3 and 4.4. As for the *Contraction pass*, the idea is to try to contract each element's edges to try to get rid of those that are close to being degenerated and its implementation is similar regardless of the dimensionality of the mesh.

4.2 Control mesh

As presented in chapter 2, we represent the object in two distinct spaces called world and material space. In its representation in world space are applied all the deformation whereas in material space only the topological changes are applied. This method ensures bijection on all mesh operations.

When we perform a remeshing step on the world mesh, the operation needs to be validated on the control mesh in the material space. As illustrated in fig.4.1, after each operation is performed we check whether it creates a bad element in the control mesh. In this case, we reset the material space mesh to its previous stable state or update the control mesh if all elements are of sufficient quality.

With this, each modification made on the mesh has an opposite operation able to revert it to its original state.

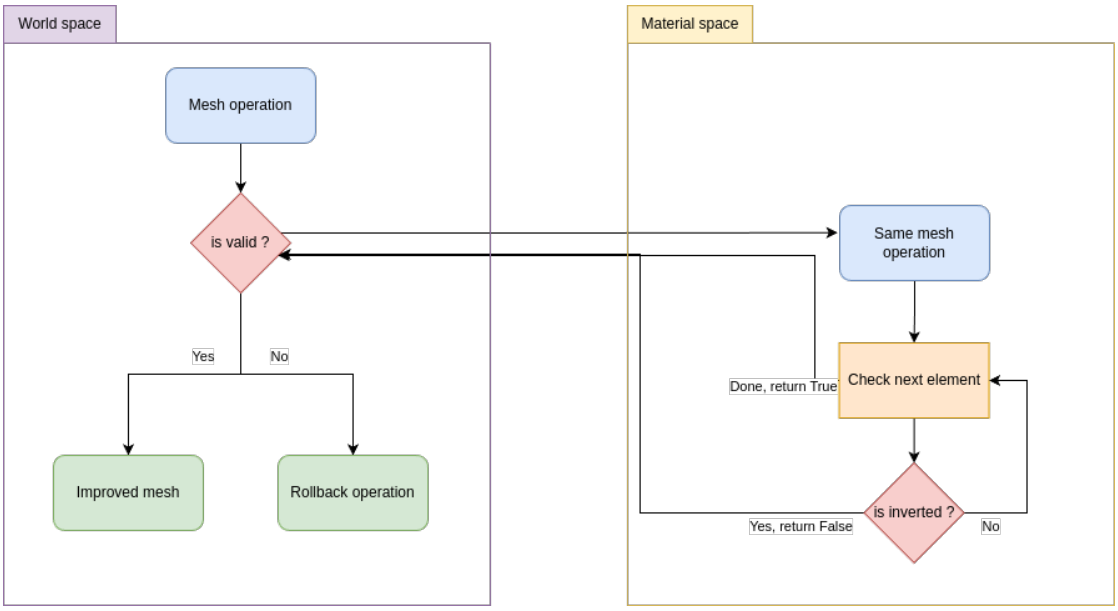


Figure 4.1: Flow diagram for control mesh validation. Each operation in world space has to be validated in the material space before being applied to the mesh.

4.3 Triangular meshes

This section describes the implementation of the topological, insertion and smoothing passes with two-dimensional meshes.

4.3.1 Topological pass

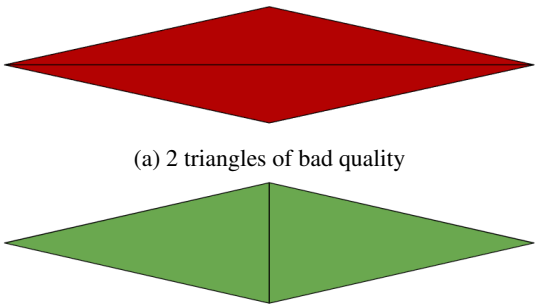
In three-dimensional meshes, Wicke et al.[10] use this pass to remove edges and faces of bad quality. We adapted it to two-dimensional meshes while keeping this idea of low-quality element removal. All without being redundant with the contraction pass presented in alg. 2. To do so we used the edge flip.

Shown in fig. 4.2, we see that by flipping an edge both of the bad-quality faces disappear to leave only elements of acceptable quality. Now that we have a mesh operation similar to the one in [10], the algorithm (alg. 3) for the *topological pass* goes as such:

We take the edges of all elements we want to improve quality. For each of them, a flip is performed and we check whether the quality of the neighboring triangle improves. Note that boundary edges cannot be flipped as it would modify the shape of the object and not only its topology. For all triangles whose quality improved, we add them to a set of new triangles, even if their quality is above the quality threshold. This set can then be processed in another pass of alg. 2.

Algorithm 3 Topological pass on triangular meshes

Require: A a set of triangles of the mesh
  function TOPOLOGICALPASS(A)
    $changed \leftarrow false$
    $E \leftarrow$ set of all edges of triangles in A
    $A_{new} \leftarrow \{\}$
    for all $e \in E$ **do**
      if e still exists **then**
        attempt to flip e ▷ t_1, t_2 created by flipping edge e
        $A_{new} \leftarrow A_{new} \cup t_1, t_2$
        $changed \leftarrow true$
        if $Quality(t_1, t_2)$ did not improve **then** ▷ Ensure mesh improvement
          revert operation
        end if
      end if
    end for
    return the surviving elements of $A \cup A_{new}$ and $changed$
  end function



(a) 2 triangles of bad quality

(b) 2 better triangles after edge flip

Figure 4.2: Improving topology by edge flip

4.3.2 Insertion pass

For the two-dimensional insertion pass, presented in alg. 4, the idea is similar to the original three-dimensional implementation. We first attempt to dig a cavity, which is in turn filled with new elements of better quality.

The key part of this algorithm is cavity digging, as filling it up simply consists of connecting the new point p located at the centroid of the triangles of A to the edges of the previously dug cavity. To find which elements belong to a cavity, the $FindFaceWithP$ function goes as follows: We start from the first element of the cavity, t, and mark it as visited. Then, we iterate through its neighbors and check whether,

one they never have been visited, and two their circumcircle contains the new point p. If this is the case, we add this neighbor to the cavity and recursively treat its own neighbors as candidates for the cavity. This process is illustrated in fig. 4.3.

Once completed, the result of the smoothed-out vertex p added to the mesh can be seen in fig. 4.4.

Algorithm 4 Insertion pass on triangular meshes

Require: A, a set of triangles of the mesh
  function INSERTIONPASS(A)
    $A_{new} \leftarrow \{\}$
    for all $t \in A$ **do**
      $p \leftarrow$ centroid of t
      $D \leftarrow FindFaceWithP(D, t, p)$
      for all $f \in D$ **do**
        $delete f$ ▷ creates a cavity
      end for
      for all $e \in$ cavity's edges **do**
        add face f' with p, e
        $A_{new} \leftarrow A_{new} \cup f'$
      end for
      $Smooth(p)$
    end for
    return the surviving elements of $A \cup A_{new}$
  end function

  function FINDFACEWITHP(D, f, p)
    if $visited(f)$ or !$contains(p)$ **then**
      return
    end if
    $visited(f) \leftarrow True$
    $D \leftarrow f$ ▷ we add f to the cavity
    for all $adj \in$ adjacent faces **do**
      $FindFaceWithP(D, adj, p)$
    end for
  end function

4.3.3 Smoothing pass

The last pass to discuss is the *smoothing*. In this part of the program, we attempt to regularize the point distribution through the mesh. For the triangular mesh implementation, we kept it simple and made it so that any point p to be smoothed is moved to the average position of its neighbors.

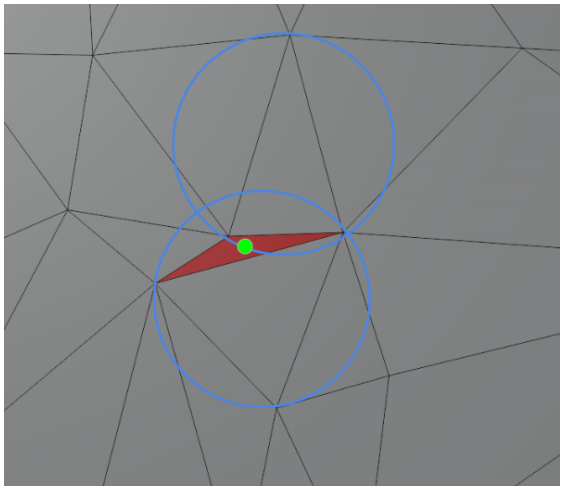


Figure 4.3: 2D Cavity building with successive circumcircles. In green the new vertex added to the mesh to remove the bad triangle in red.

(a) Center vertex offset from its neighbor's average

(b) Same mesh after smoothing

Figure 4.5: Vertex smoothing on 2D meshes

4.4 Tetrahedral meshes

This section describes the implementation of the topological, insertion and smoothing passes with tetrahedral meshes. For clarity, all the material space checks presented in section 4.2 are omitted. In reality, for each mesh operation presented below, a validity check on the control mesh is performed to ensure bijection.

4.4.1 Topological pass

In the topological pass, the aim is to perform flips such that we can transform edges into faces or vice versa. These operations, illustrated in fig. 4.6, can be described as follow:

- **2-3 flip:** Transforms a face between *2* tetrahedra into an edge between *3* tetrahedra.
- **3-2 flip:** Reverse operation to the 2-3 flip, transforms an edge between *3* tetrahedra into a face between *2* tetrahedra.
- **Multi-face removal:** Generalization of the 2-3 flip, transforms n faces into an edge between $n + 2$ tetrahedra.

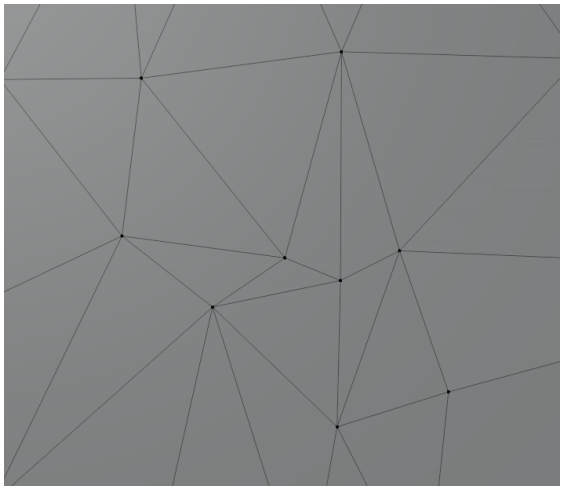


Figure 4.4: Result of insertion pass. The bad triangle of fig. 4.3 has been removed and the new vertex smoothed to its new position.

- **Edge removal:** Generalization of the 3-2 flip and reverse operation to the multi-face removal, transforms an edge between n tetrahedra into $n-2$ faces between $2n-4$ tetrahedra.
- 2-2 and 4-4 flips: Switches the orientation of a group of tetrahedra, this operation is used in [10] but not in this implementation.

A pass, described in alg. 5, tries to remove each edge and face of a tetrahedron.

To remove an edge e, it goes through the steps of alg. 6. We first compute the *one ring* around e. It is composed of the vertices adjacent to e without being connected to the edge. Shewchuk [8] mentions that with a ring size greater than 7 vertices, the resulting quality will rarely improve and is not worth the processing cost. After splitting e, we must find the collapse direction resulting in the smallest local energy. To do so, we collapse the newly added vertex in each direction and compare the results, storing the best halfedge to collapse. In the end, we ensure that the energy of the mesh around e has improved. If that is not the case, the operation is rolled back.

As for the face removal, we compare the resulting energy between a simple 2-3 flip and the multi-face removal. If neither operation improves the mesh energy, none is chosen. The multi-face removal, described in more detail in [8], consists of finding the faces that are *sandwiched* between two vertices a and b. From [8], "[...] a triangular face f is *sandwiched* between a and b if the two tetrahedra that include f are conv($f \cup a$) and conv($f \cup b$)". Once we have found all the sandwiched faces, they are removed via a combination of 2-3 and 3-2 flips.

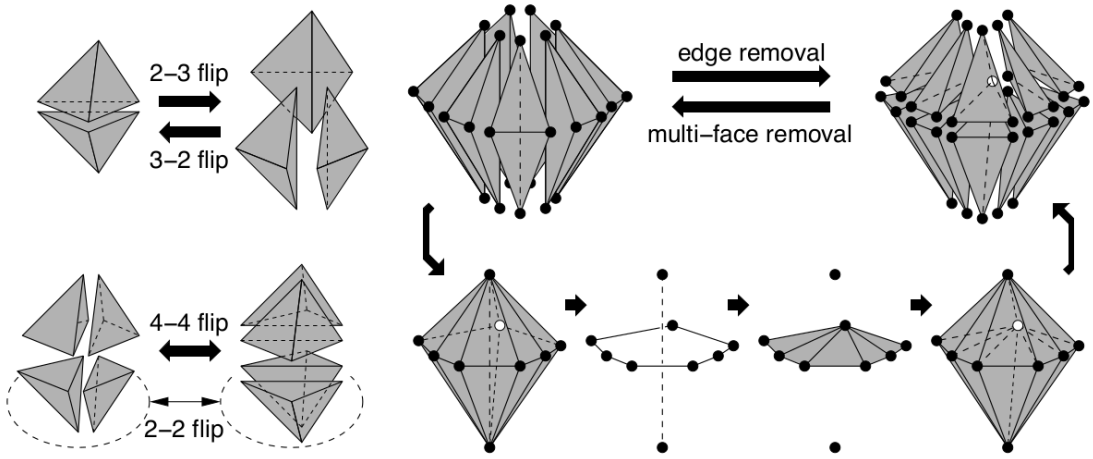


Figure 4.6: Mesh operations used, taken from [10]

Algorithm 5 Topological pass on tetrahedral meshes

Require: A, a set of tets of the mesh
  function TOPOLOGICALPASS(A)
    $A_{new} \leftarrow \{\}$
    for all $t \in A$ **do**
      $E \leftarrow$ edges of t
      $F \leftarrow$ faces of t
      for all $e \in E$ **do**
        if e exists **then**
          $changed \leftarrow RemoveEdge(A_{new}, e)$
        end if
      end for
      if $changed$ **then**
        continue
      end if
      for all $f \in F$ **do**
        if f exists **then**
          $changed \leftarrow RemoveFace(A_{new}, f)$
        end if
      end for
    end for
    return the surviving elements of $A \cup A_{new}$
  end function

Algorithm 6 Edge removal on tetrahedral meshes

Require: $added$, a set of tets, e, the edge to remove
  function REMOVEEDGE($added$)
    $m \leftarrow OneRing(e)$
    $q_{old} \leftarrow Quality(mesh)$
    if $m > 7$ **then**
      return ▷ From [8], after $m > 7$ the mesh rarely improves
    end if
    $v \leftarrow SplitEdge(e)$
    $h \leftarrow FindCollapseDirection(v)$ ▷ Find the halfedge to collapse giving the best quality
    $Smooth(v)$
    if $Quality(mesh) < q_{old}$ **then**
      rollback operation
      return False
    end if
    $added \leftarrow$ tets around v
    return True
  end function

4.4.2 Insertion pass

As described above, the insertion pass consists of digging a cavity and filling it up anew with elements of lower energy. This allows us to break from local optima that could be reached in the other passes. By adding new tetrahedra we create a different topology opening a new set of remeshing opportunities.

To dig the cavity, we adapt the concept of *galaxies* presented by Hinderink and Campen[5]. Starting from a list of tetrahedra of high energy we construct a set of *stars* united in a common *galaxy*. To construct these, we use their *center of chebyshev*.

4.4.2.1 Center of Chebyshev

The *center of chebyshev* [3] of a star is the center of its largest inscribed sphere. In fig. 4.7, we can see its use to determine whether a cavity is star-shaped. We take the boundary of the cavity and find the inscribed sphere. If the radius r of the sphere is greater than a threshold s_{min} then the cavity is star-shaped. In the field of remeshing, we can increase the value of s_{min} to ensure that a cavity is not too small, where it would then be filled with tetrahedra close to degenerate, or of high energy.

4.4.2.2 Insertion algorithm

The insertion algorithm is composed of three substeps: the cavity digging, the cavity filling, and a topological pass.

For the first step (alg. 7), from each tetrahedron in A we start growing a star. This means that at first, each star has a size of one. To add to it, we want to add the neighboring tetrahedron that is the most likely to maintain star-shapedness. [5] use the following equation (eq. 4.1): Among the faces of the tetrahedra from a star S, we search the face whose incident tetrahedron is the least likely to break the star conditions, with n_f the normal of a triangle f, a_f a point on its image and x_0 the center of chebyshev. The candidates f^* are presented as a sorted list, where we drop any element already contained in a star. If the end of the list is reached, then no additional element is joined to the star and the procedure ends.

$$f^* = \arg\max_f n_f^T x_0 - n_f^T a_f \tag{4.1}$$

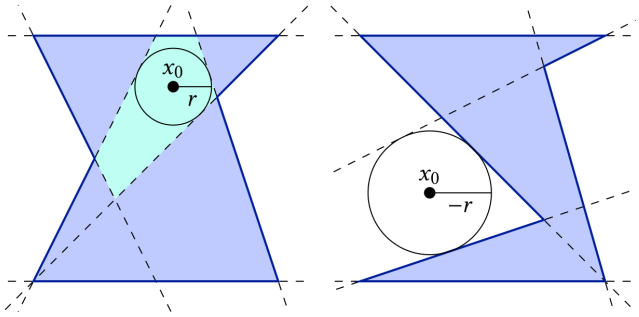


Figure 4.7: Star-shapedness tested with the center of chebyshev, taken from [5]. A cavity is star-shaped if it holds an inscribed sphere of radius $r > 0$ with center of chebyshev x_0.

Once the galaxy is fully computed, we remove all the tetrahedra contained in the stars, hollowing out the mesh.

Algorithm 7 Cavity digging on tetrahedral meshes

Require: A, a set of tets from the mesh
  function CAVITYDIG(A)
    $N \leftarrow 5$
    $G \leftarrow \{\}$ ▷ Empirically set, controls the size of the stars
    for all $t \in A$ **do** ▷ create galaxy
      if $t \notin S, \forall S \in G$ **then**
        $S' \leftarrow \{t\}$ ▷ Create new star
        do
          $t' \leftarrow FindNext(t)$
          $S' \leftarrow S' \cup \{t'\}$
        while $StarShaped(S')$ and $S' < N$
        $G \leftarrow G \cup S'$
      end if
    end for
    for all $S \in G$ **do** ▷ dig cavity
      for all $t \in S$ **do**
        $DeleteTet(t)$
      end for
    end for
  end function

Once the cavities are hollowed, for each star of the galaxy we add a new vertex at its center of chebyshev and connect the cavity to this new element. Once the cavity is filled up, we smooth the created center to set an acceptable starting point for the last step of the insertion pass: a topological pass on the tetrahedra used to fill the cavity. Finally, if this pass did not improve the mesh energy, the operation is rolled back to its previous state.

Algorithm 8 Cavity Filling on tetrahedral meshes

Require: G, a galaxy
  function CAVITYFILL(G)
    for all $S \in G$ **do**
      $c \leftarrow$ center of chebyshev of S
      $A' \leftarrow \{\}$
      $AddVertex(c)$
      for all $b \in$ boundary faces of S **do**
        $t' \leftarrow AddTet(c, b)$
        $A' \leftarrow A' \cup \{t'\}$
      end for
      $Smooth(c)$
      $i \leftarrow 2$
      do
        $changed \leftarrow TopologicalPass(A')$
        $i \leftarrow i - 1$
      while $i > 0$ and $changed$
    end for
    if $Quality(G) < q_{old}$ **then** ▷ q_{old} is the quality of the galaxy before the pass
      Rollback operation
    end if
  end function

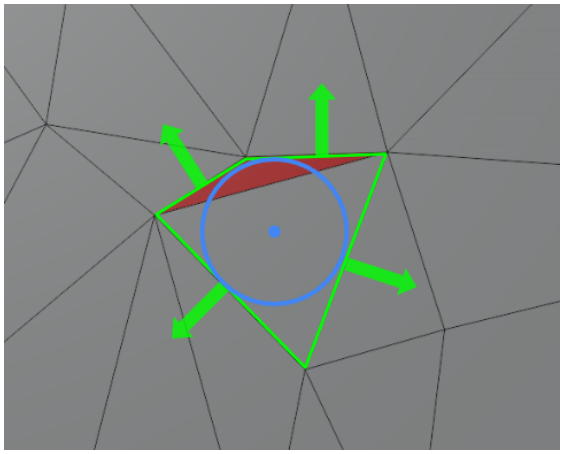


Figure 4.8: Boundary of a cavity (green) with its inscribed sphere and center of chebyshev (blue).

4.4.3 Smoothing pass

For the last pass of the algorithm, as presented in alg. 9, we go through all the remaining tetrahedra from A. A is the list of elements that went through all the previous passes. Here, the goal is to smooth out the vertices of the list. To do so, we use the *center of chebyshev* presented in sec. 4.4.2.1. We treat the neighboring tetrahedra to the vertex v we want to smooth as a cavity similar to the insertion pass (see fig. 4.9) and move v to the position of the center of chebyshev.

This pass is the only one that does not guarantee global improvement of the energy. Even though local improvement is checked during execution, the work done on a vertex can be undone by the smoothing of the following one. To avoid this, a more global approach should have been taken rather than the iterative way of doing remeshing used in this implementation. Empirically, we saw that sometimes this issue even is a benefit as it allows some remeshing operations to be performed by passing the energy of some elements above the threshold where it would not have happened otherwise. So an increase in energy at a timestep t might result in a decrease at a later stage $t + n$.

This pass is the last to be applied to the list A as it expends it exponentially. For each vertex smoothed, we add its neighbors to A. Even after removing duplicates, this is by far the operation that changes the list the most and thus impacts performances.

Algorithm 9 Smoothing pass on tetrahedral meshes

Require: A, a set of tets of the mesh
  function SMOOTHINGPASS(A)
    $A' \leftarrow \{\}$
    for all $t \in A$ **do**
      $V \leftarrow$ vertices of t
      for all $v \in V$ **do**
        $N \leftarrow$ neighboring tets of v
        $q_{old} \leftarrow Quality(N)$ ▷ Calculate energy of tets from N
        $Smooth(v)$
        if $Quality(v) < q_{old}$ **then**
          Rollback operation
          continue
        end if
        $A' \leftarrow A' \cup N$
      end for
    end for
    return the surviving elements of $A \cup A'$
  end function

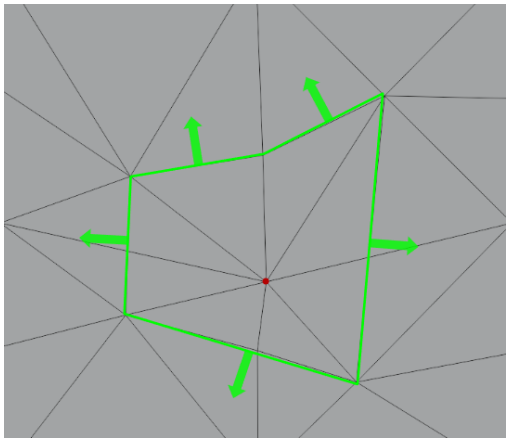


Figure 4.9: Boundary (green) used to find the center of chebyshev during smoothing pass on a vertex (red).

5 Results

This chapter presents the experiments run with the algorithm, on two- and three-dimensional meshes, as well as discusses the results obtained.

5.1 Triangular meshes

In this section, I'll describe the two experiments made in 2D and show the search for the optimal value of minimal quality. Mentioning that 2D was the stepping stone into the project and not many experiments were run. The implementation of the algorithm for triangle meshes served as a stepping stone into the more complex field of tetrahedral meshes. As such, not much experimentation was run with this implementation since it served more as a proof of concept. Two experiments were led with the two-dimensional remeshing algorithm: stretching the mesh and compressing it. In both cases, the goal was to empirically determine an acceptable value for the remeshing threshold q_{min}.

Each experiment is run through iterative timesteps, defining the harshness of each deformation step. At each time step, first, the deformation is applied to the mesh, and after this comes a remeshing pass. This composition can be seen in figure 5.1.

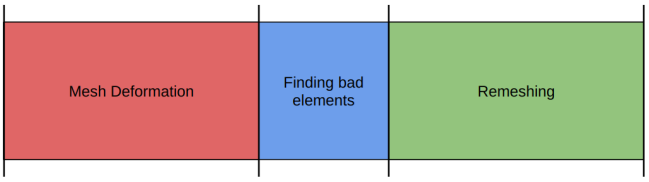


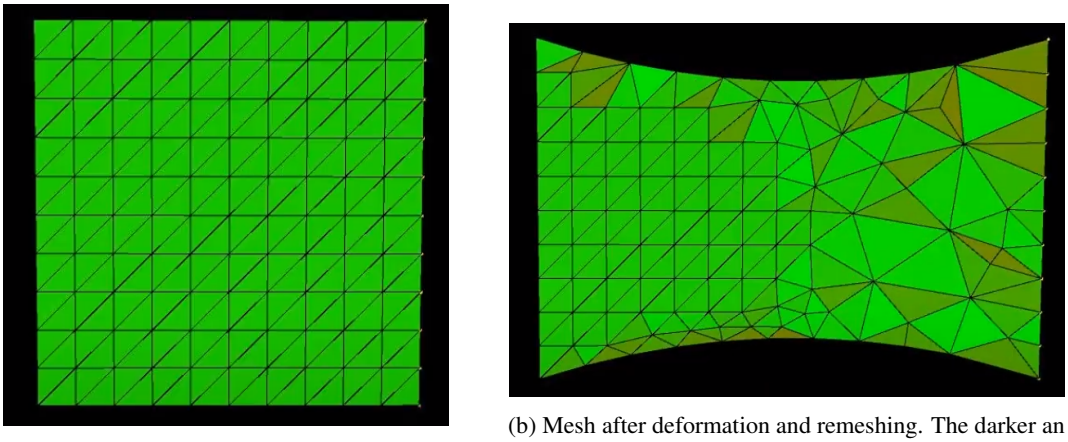
Figure 5.1: Timestep composition

5.1.1 Stretch

In this first experiment, the rectangular base mesh (fig. 5.2a) is stretched horizontally by a factor of 1.5. All of the stretching comes by moving the right boundary of the mesh. A real-life representation could

be thought of as the left boundary being anchored and the right being pulled. To approximate physical properties, the boundaries perpendicular to the stretching direction are bent into a concave parabola as it would in a plastic deformation.

We indeed see in figure 5.2b that most of the low-quality elements and thus remeshing performed are situated near the deformed parts of the mesh: the top, bottom, and right boundaries. We can also note some flips as well as additional geometry, meaning that both the topological and insertion pass influenced the remeshing.



(a) Mesh before deformation

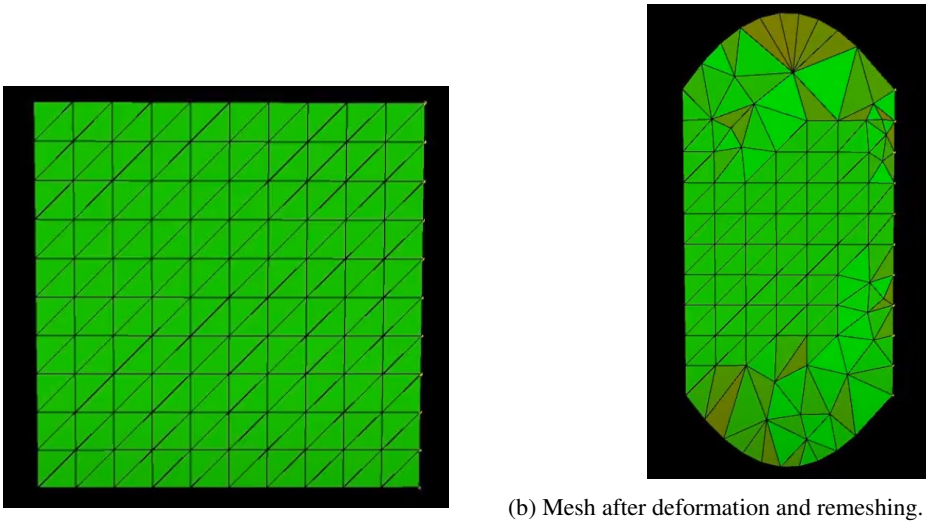
(b) Mesh after deformation and remeshing. The darker an element, the lower its quality

Figure 5.2: 2D stretch experiment

5.1.2 Compress

In this experiment, opposite to the stretching done before, we compress the mesh by moving the rightmost boundary to 66% of the initial mesh length. Similar to before, the left boundary stays immobile whereas the top and bottom boundary are bent to a parabola. For this situation, to simulate the material being pushed out by the compression, the parabola takes a convex shape.

Similar to the stretch experiment, most of the work is done near the boundaries. We also have a clear show of the work of the contraction pass. When counting the number of triangles for a horizontal line going through the middle of the mesh, the number decreases from 20 to 14. This means that the triangles compressed by the moving edge have been removed by contracting one of their edges during the contraction pass. We can also note that the result of the remeshing differs significantly between the top and bottom boundaries. This highlights a point, which will also appear later for the tetrahedral meshes, of the remeshing results depending on initial conditions and some operations unlocking new possibilities at later timesteps that could be missed without them.



(a) Mesh before deformation

(b) Mesh after deformation and remeshing. The darker an element, the lower its quality

Figure 5.3: 2D compress experiment

5.1.3 Finding the minimal quality

This last experiment on triangle meshes was run with the compress setting described above. The goal for it is to find the best value for q_{min}, the threshold responsible for adding elements to the list of elements to be processed in the remeshing pass. We need to fine-tune between a value too low, not allowing enough remeshing thus not improving mesh quality, and a value too high not realistically achievable triggering useless remeshing operations without actually improving topology. As presented in section 3.2, the quality of a triangle ranges between 0 and 1. The compressing experiment was run with a value of q_{min} from 0.1 to 0.7 with the same number of timesteps in each run. Higher values were not deemed interesting as they exceeded or were close to the initial mesh quality, thus not being realistically reachable.

Observing the results presented in figure 5.4, we can split them into three categories:

- **Close to degeneracy**: With q_{min} equal to 0.1 or 0.2 (first row of fig. 5.4), the threshold is so low that it almost accept degenerate element into the mesh. This can lead to the situation where performing a remeshing worsens the situation as it induces inverted elements faster than the mesh deformation.
- **Optimal threshold**: For values ranging between 0.3 and 0.55 (middle rows of fig. 5.4) we see that the remeshing is still capable of maintaining the desired mesh quality even with the deformation at its maximum. Note that with $q_{min} = 0.4$, the threshold is not reached at later stages but still can be reached for higher values. An explanation can be found in the hill climbing principle. A remeshing operation could at a time t be seen as the best choice, only to be at a later time $t + n$ restricting a more efficient operation. That is why 0.4 still can be included in this category as the threshold is neither too restrictive nor tolerant but only depends on initial mesh configuration.
- **Unreachable threshold**: Values ranging from 0.6 and above (last row of fig. 5.4), although the desire behind them is to maintain high mesh quality, show that there exists a point where this desire is simply too much and cannot be fulfilled. They, while performing more operations than the other categories, produce worse results than lower thresholds situated in the optimal category. This is due to the program trying its best to reach an acceptable mesh quality, performing as many remeshing steps as possible and diverting from the regular initial mesh. This irregular mesh is more likely to create even worse irremovable elements after the next deformation step.

From the optimal threshold category, we can determine that a value for q_{min} near 0.5 is the best choice. We want the best possible quality with the best performances. With $q_{min} = 0.55$, we get closer to the upper limit and might risk losing the capability to reach it after deformation.

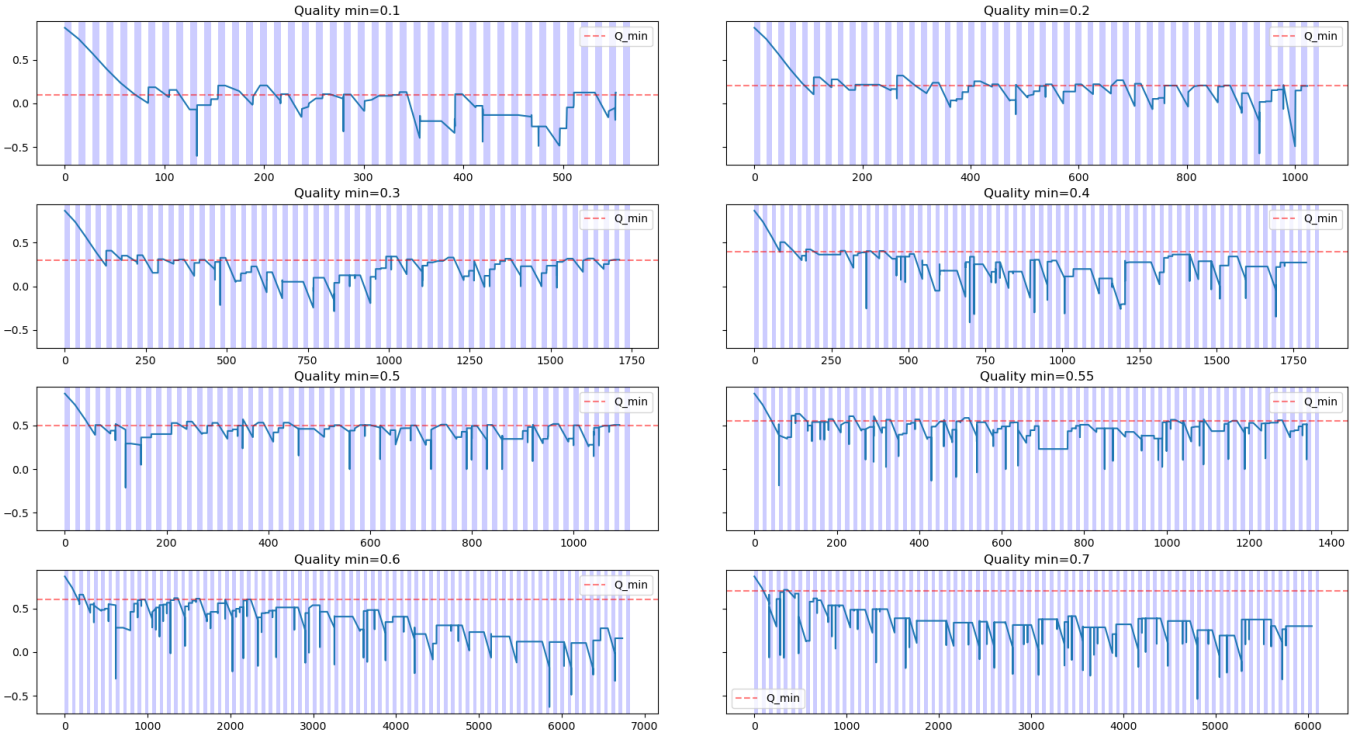


Figure 5.4: Evolution of the quality for different thresholds. The y-axis shows mesh quality, the x-axis shows the number of mesh operations performed

5.2 Tetrahedral meshes

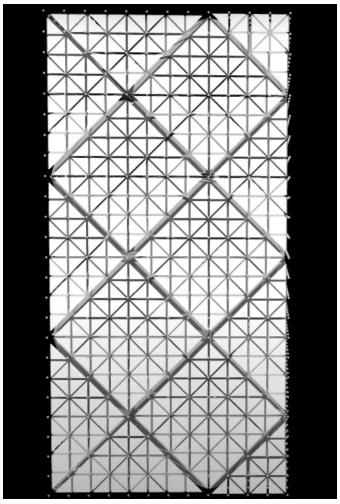
For 3D meshes, each deformation applied to the mesh is either a spin or a stretch but the observed points are more diverse than for triangle meshes. Mainly, our goal through these experiments is to find an optimal threshold for the energy, evaluating the impact of the timestep size and deformation intensity as well as the impact of the control mesh.

5.2.1 Spin

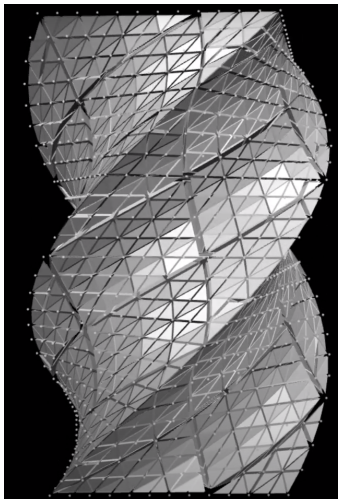
This mesh deformation consists of rotating vertices along the z-axis, giving a spiraling look to the base mesh (figs. 5.5a and 5.5b). Two variants of the deformation were applied. First, a basic one in which the whole mesh rotates along the axis. Second, one where only the surface vertices were allowed to move, thus triggering more remeshing operations as the difference between the surface and inside increases. The first option was used in most experiments as it is more realistic from a physical standpoint. If in a section nothing about the deformation is mentioned, assume all the vertices were free of movement.

For performance reasons, the surface of the mesh has been subdivided (fig. 5.5c) to allow more remeshing options while keeping a lower number of tetrahedra. As this is the part subject to the most

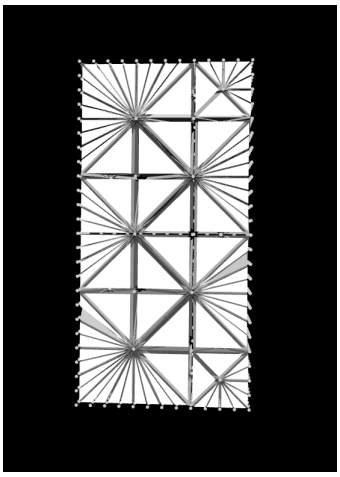
deformation, being the furthest away from the rotation axis.



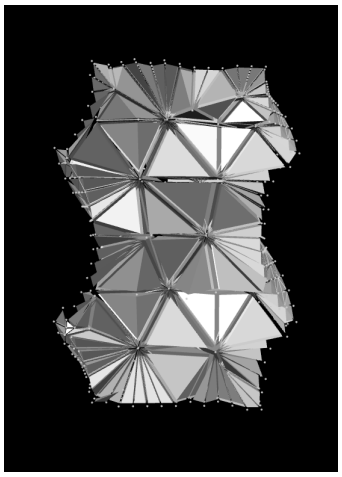
(a) Mesh before deformation



(b) Mesh after deformation and remeshing



(c) Mid section of the mesh before deformation



(d) Mid section of the mesh after deformation and remeshing

Figure 5.5: 3D spin experiment

5.2.1.1 Finding the optimal threshold

In this experiment, we aim to find the optimal value for the maximal energy threshold d_{max}. To do so, we applied the spin deformation to boundary vertices with increasing leniency towards deformation. Moving only the surface vertices allows for more observations of the remeshing being performed. They rotated 180 around the axis in 360 timesteps, giving a 0.5 increment per timestep.

In figure 5.6, we observe three categories of results:

1. **Unsuccessful remeshing**: With $d_{max} = 35$, the desired energy is too low and no remeshing step allows the mesh to go back to this state once the limit is crossed by many elements. In figure 5.6 top

left, we see that the first crossing of the threshold can be recovered like any other value for d_{max} but as soon as multiple timesteps are unable to lower the energy it continuously increases in average.

2. **Partially successful remeshing**: With $d_{max} = 40$, we seem to be near the limit of energy the algorithm can recover from. In the first part, the deformation is salvageable but it reaches a state from which it is not able to come back, slipping into the first category of unsuccessful remeshing.

3. **Successful remeshing**: With $d_{max} \geq 45$, each crossing of the threshold is recovered from. With no sign of slipping into an endless loop of increasing energy.

These results show that a threshold of ~ 45 is the best choice in this situation. This value is in no way absolute as it must take into account the severity of the deformation. For more intense deformations, where greater energies are unavoidable, a d_{max} of 45 might give results of the first or second category.

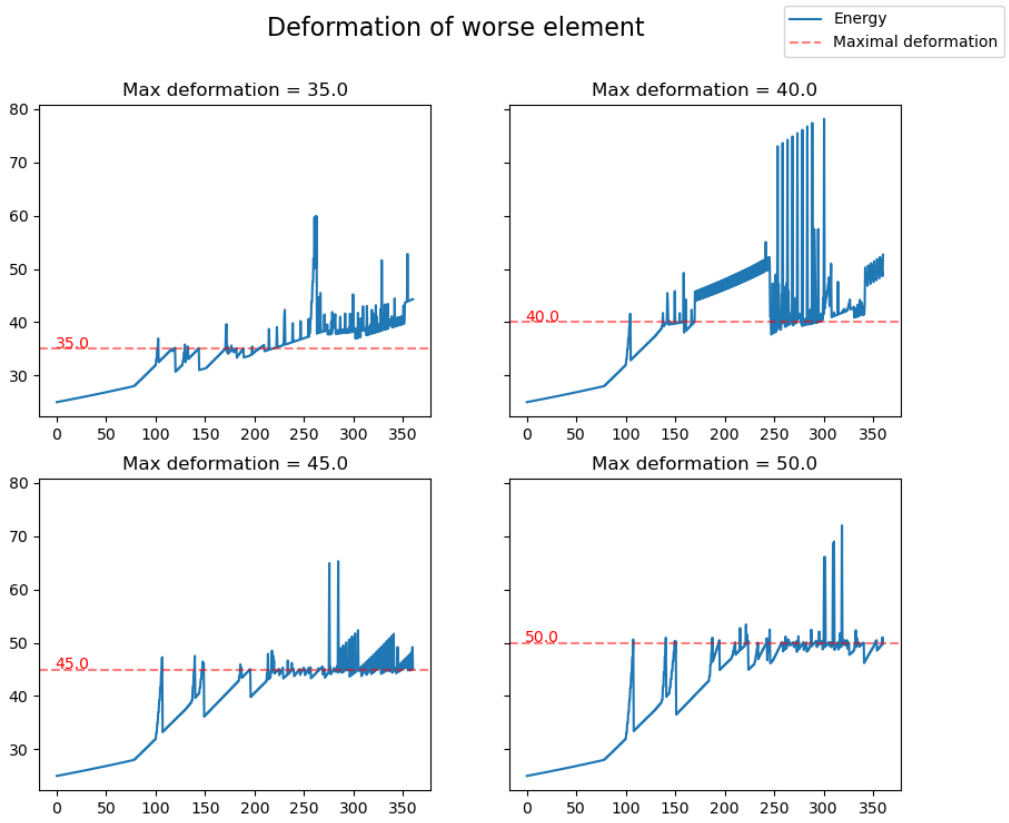


Figure 5.6: Comparison of energy for different threshold values during the spin. The x-axis represents the timestep progression, the y-axis shows the deformation energy of the worst element of the mesh

5.2.1.2 Finding the optimal number of timesteps

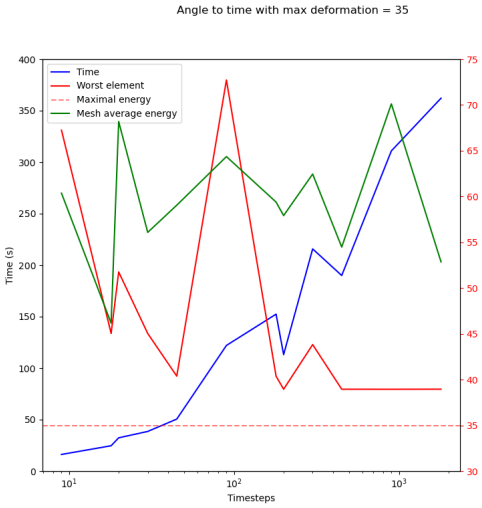
In this second experiment, the goal is to evaluate the impact on performance and quality of the remeshing. To do so, the spin deformation is applied with different parameters shown in figure 5.7. For the half turns (rotation of 180), the number of timesteps ranges from 9 to 1800 and is doubled for the full turn. This

brings us to a rotation angle of 0.1 to 20 per timestep.

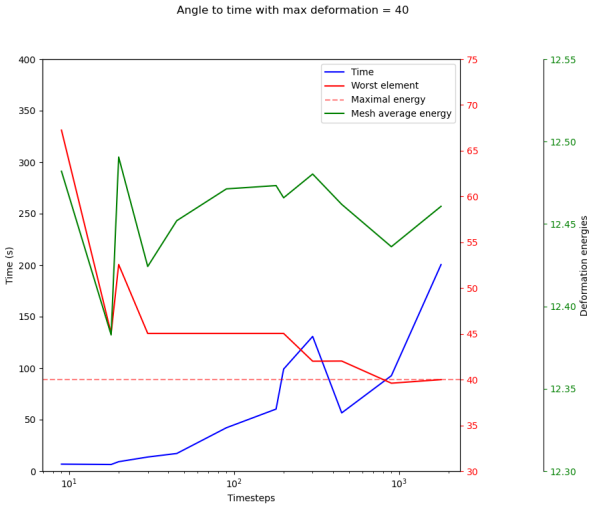
For experiments with a low remeshing threshold or high deformation (figs. 5.7a and 5.7d), we get results of the unsuccessful remeshing category presented in the previous experiment where no matter the number of timesteps it is unable to recover from the high energy. We see that increasing the number of timesteps does not equate with better quality. This shows the limitation induced by local optima in hill-climbing methods. Based on the remeshing choices performed during previous iterations, some largely better options cannot be performed but are available to the ones who skipped some remeshing steps.

In figure 5.7b is shown that the solution with d_{max} presented as a partially successful remeshing could be tipped into any other category based on the number of timesteps, fortifying its position as the limit between them. By increasing the number of timesteps it now falls into the successful category whereas by decreasing it worsens its results. This highlights the trade-off between performance and quality as this shift in category comes at a great price in terms of execution time.

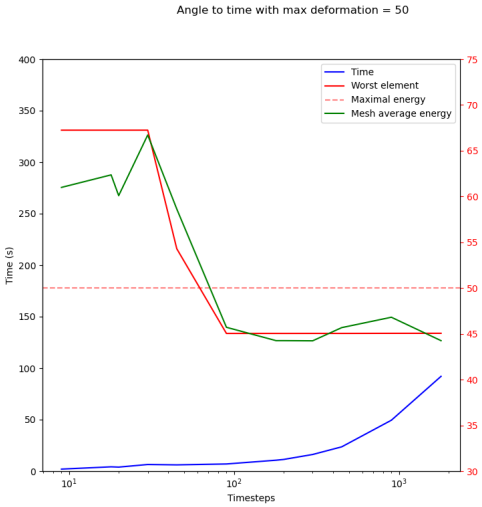
The more timesteps the better can be used as a good approximation tool but is not to be treated as a golden rule. In the experiment with $d_{max} = 50$ (fig. 5.7c), it appears that the energy of the worst element, as well as the mesh, seems to plateau below the threshold with an exponential growth in time required. In this situation, using more timesteps is unnecessary as we could use the additional time freed by performing more passes of the algorithm, maybe giving an even better result. This plateau might be interpreted as the last acceptable energy for a tetrahedron. Any higher energy would trigger the remeshing after the next deformation step, thus bringing its quality below the threshold again.



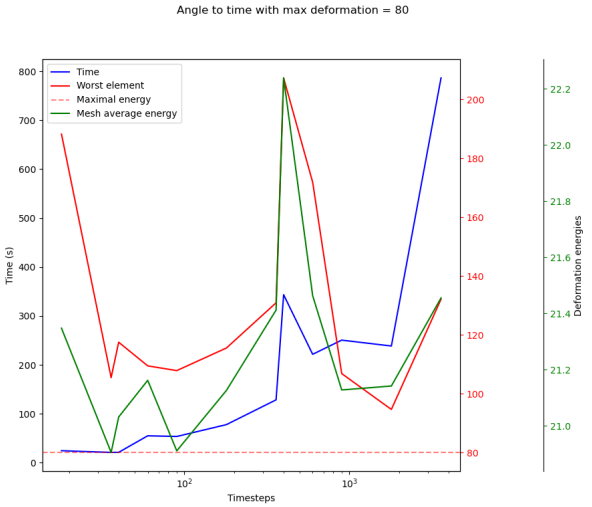
(a) Half turn with maximal deformation of 35



(b) Half turn with maximal deformation of 40



(c) Half turn with maximal deformation of 50



(d) Full turn with maximal deformation of 80

Figure 5.7: Comparison of time (blue), worst element energy (red) and average mesh energy (green) for varying timestep numbers. The x-axis shows the number of timesteps, the left y-axis the time in seconds, and the right y-axis the deformation energies

5.2.1.3 Handling greater deformations

For this last experiment with the spin deformation, we observe the impact of larger deformations on the remeshing algorithm. We took the lenient $d_{max} = 50$ from previous experiments and 0.5 increment timesteps. We see in figure 5.8 that larger deformations explode in terms of both energy and time, especially for the full turn. We see here the limitations of this implementation as the only way to make bigger deformations work would be to set a high remeshing threshold that would result in visibly bad tetrahedra, rendering the remeshing almost useless given that the increase in mesh quality would not be

worth the resources used.

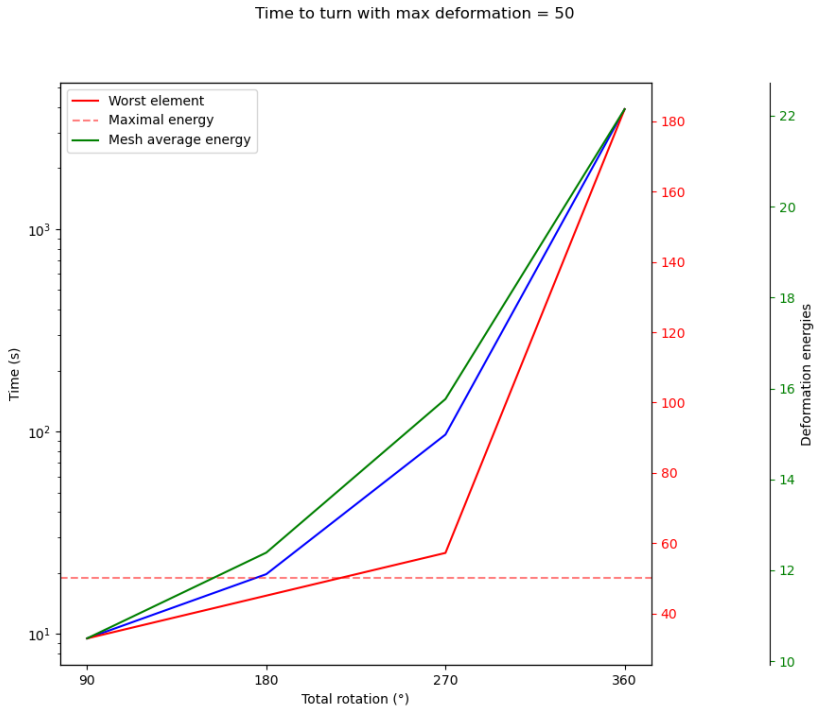
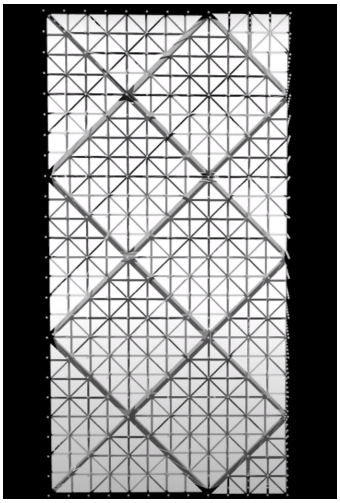


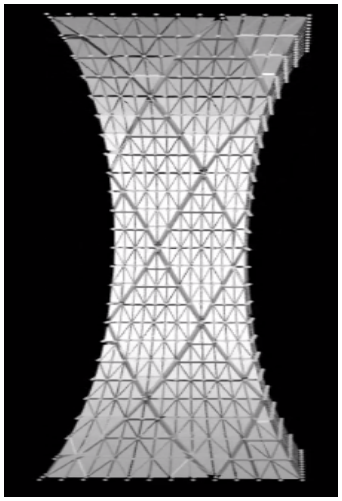
Figure 5.8: Evolution of performance with deformations ranging from 90° turns to 360°, in 90° increments.

5.2.2 Stretch

Similar to the two-dimensional implementation of section 5.1.1, for this mesh deformation the bottom boundary is pulled down while the sides are bent. This allows to emulate a plastic or elastic deformation of the object.



(a) Mesh before deformation



(b) Mesh after deformation and remeshing

Figure 5.9: 3D spin experiment

5.2.2.1 Finding the optimal threshold

In this experiment, the mesh is stretched to 1.7 times its initial length with remeshing thresholds equal to $30, 35$ and 40. All the runs were performed in 300 timesteps.

Similarly to the spin's results, we see again in the figure 5.10 the three categories set in the previous section (sec. 5.2.1.1): *unsuccessful, partially successful and successful remeshing*.

With a threshold value too low like $d_{max} = 35$, the remeshing is not able to keep control of the deformation, and the energy increases without showing signs of getting back to an acceptable level. When nearing the optimal value, the energy curve follows the threshold for a while before taking off.

With the results of this second experiment, we can generalize the behavior of the energy curve to identify whether an arbitrary value for d_{max} is optimal based on its curve. The procedure would be to start high and, as long as the curve stays in the 3rd category, lower it until we reach the 2nd category. If we ever overshoot and reach the 1st one, we then know to increase it again, following some kind of *binary search* methodology.

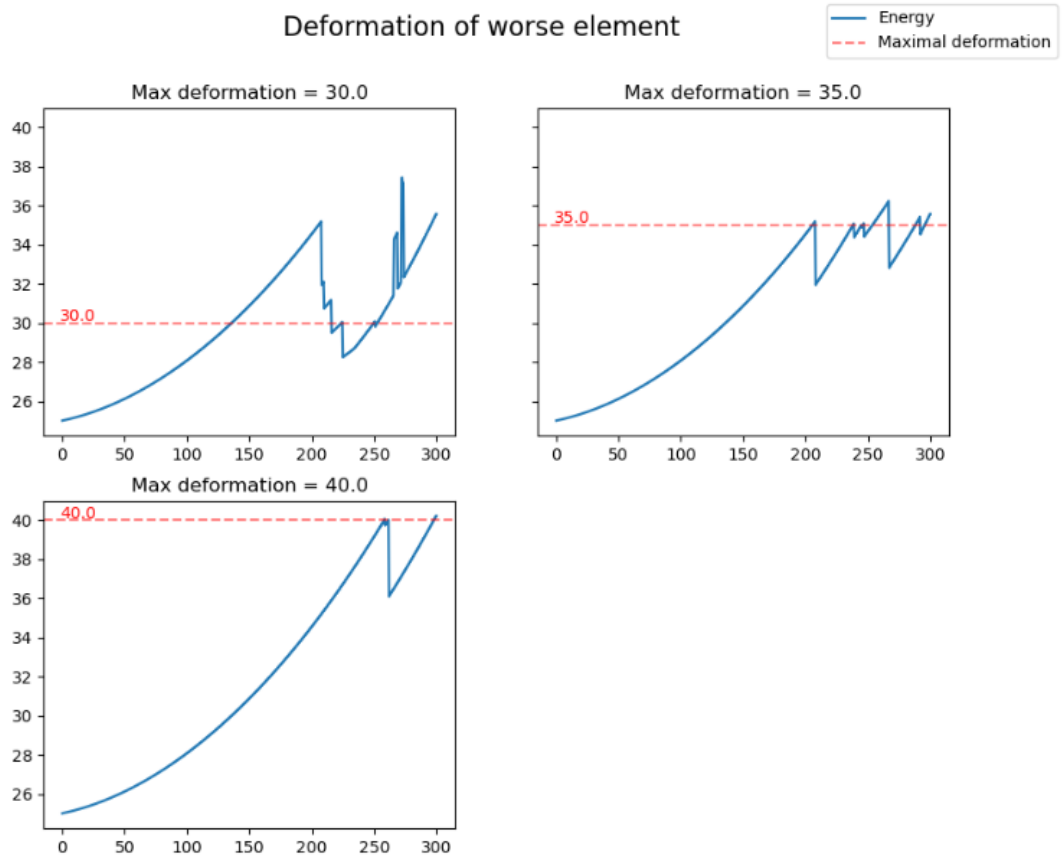


Figure 5.10: Comparison of energy for different threshold values during the stretch. The x-axis represents the timestep progression, the y-axis shows the deformation energy of the worst element of the mesh

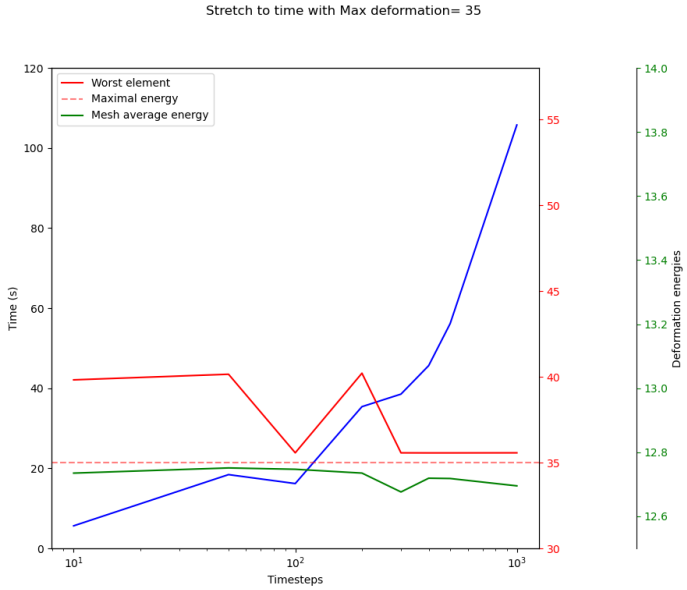
5.2.2.2 Finding the optimal number of timesteps

In this experiment, we tried stretching the mesh by a factor of 1.7 with varying numbers of timesteps, ranging from 10 to 1000. In figure 5.11, we see the results with two different thresholds. The first one (fig. 5.11a) takes a voluntarily low threshold to see if we can recover from the deformation with finer timesteps and on the second (fig. 5.11b) is the value we found to be on the limit of acceptability in the experiment of section 5.2.2.1.

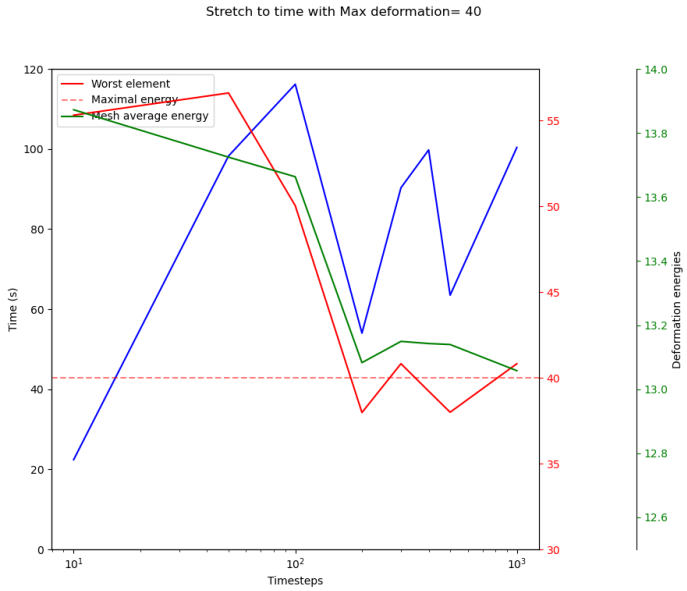
In the first case, we observe that no matter the amount of timesteps, the algorithm is never able to reach an energy below the desired value of the energy. The highest numbers of timesteps are closer and the average mesh energy lower so we can assume that a high number of timesteps is the best choice in this situation but comes at an exponential cost in performance. If the curve's dynamics were to continue with bigger x values, then settling for the smallest time-to-energy ratio would be even more efficient. We can observe a bump in energy at 200 timesteps. This is another example of a hill-climbing algorithm deciding on a local optimum leading to worst solutions in the future. Adjusting the timestep size opens possibilities that smaller or bigger amounts would not offer, thus creating a bump at only $x = 200$.

In the second case, this variety of available solutions is even clearer. Since we are at the limit of the threshold's remeshing trigger, solutions vary greatly from one another based on the timestep size. We see

that lower values yield significantly worse results by crossing too harshly the limit set by the threshold and cannot be recovered from, triggering remeshing on bigger sets of tetrahedra and increasing time consumption. For higher values, we do not see a clear favorite in terms of energy or time. We do note that failing to reach the desired energy results in bigger time requirements. This is due to the implementation of the algorithm, once we reach an acceptable energy or treat all the tetrahedra of high energy the execution stops without going through the remaining passes. Failing to treat even one element would result in additional remeshing passes impacting performance.



(a) Stretch of factor 1.7 with maximal deformation of 35



(b) Stretch of factor 1.7 with maximal deformation of 40

Figure 5.11: Comparison of time (blue), worst element energy (red) and average mesh energy (green) for varying timestep numbers. The x-axis shows the number of timesteps, the left y-axis the time in seconds, and the right y-axis the deformation energies

5.2.2.3 Handling bigger deformations

In this last experiment, we evaluate the remeshing's capacity to handle greater stretching. Starting from the default experiment with $d_{max} = 40$ and 300 timesteps with a stretching factor of 1.7, we apply stretching ranging from 1.5 to 2 while adapting the number of timesteps. Failing to do so would be unfair as bigger deformations would be performed in bigger increments, which we have seen to lower the quality of the remeshing. The full conversion can be seen in table 5.1.

Stretching factor	1.5	1.6	1.7	1.8	1.9	2
Timesteps	265	283	300	318	335	353

Table 5.1: Stretching factor to timestep relation

Like with the spin deformation (sec. 5.2.1.3) we see that bigger energies quickly go out of control of the remeshing algorithm, both in terms of quality and performance. An interesting observation is that, although on average higher, the energy of the worst element of the mesh is lower with a factor of 2 than 1.9. We observe the vulnerability of the algorithm to its preceding operations. In this case, a tetrahedron might have been altered to lower its energy at a timestep t but the following deformation step created an even worse element. The inverse interpretation is also possible, by applying a big enough deformation, a problematic tetrahedron and its neighbors that could not be removed reached a state where the remeshing operation is possible and improves the mesh quality.

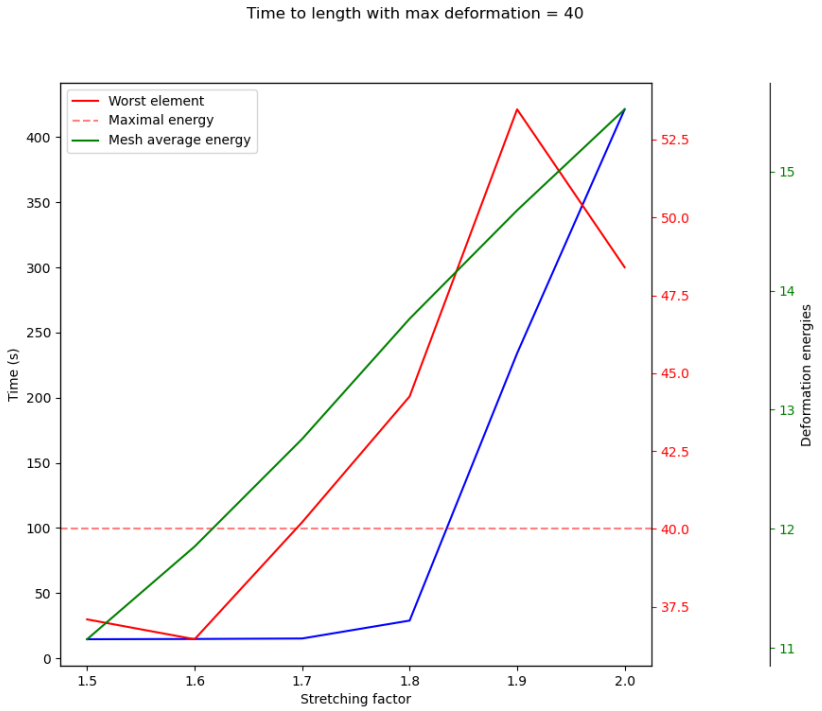


Figure 5.12: Evolution of performance with stretching factors ranging from 1.5 to 2, in 0.1 increments

5.2.3 Impact of the control mesh

Bijection of operation is ensured through the existence of both the material and world spaces. Validating each operation on the material space induces a performance overhead but also limits the range of operations available to the remeshing algorithm. If the best choice is refused in material space does the quality of the remeshing suffer from it or is it able to find alternatives to reach the desired energy?

We see in figure 5.13 that for a spin experiment, the version with the material space (in orange) can reach the same level of energy as the one without it (in blue). Not having this restriction allows for a quicker recovery but eventually this restriction yields similar results while only using revertible operations. This means that keeping a stack of the performed remeshing steps and going through it in reverse order would give us the initial mesh without changes in topology.

This demonstrates that the only additional cost of using the two spaces is resource-related. Specifically, it affects memory consumption, as a representation of the mesh is needed in both spaces, and performance, since each operation must be validated before being applied to the mesh.

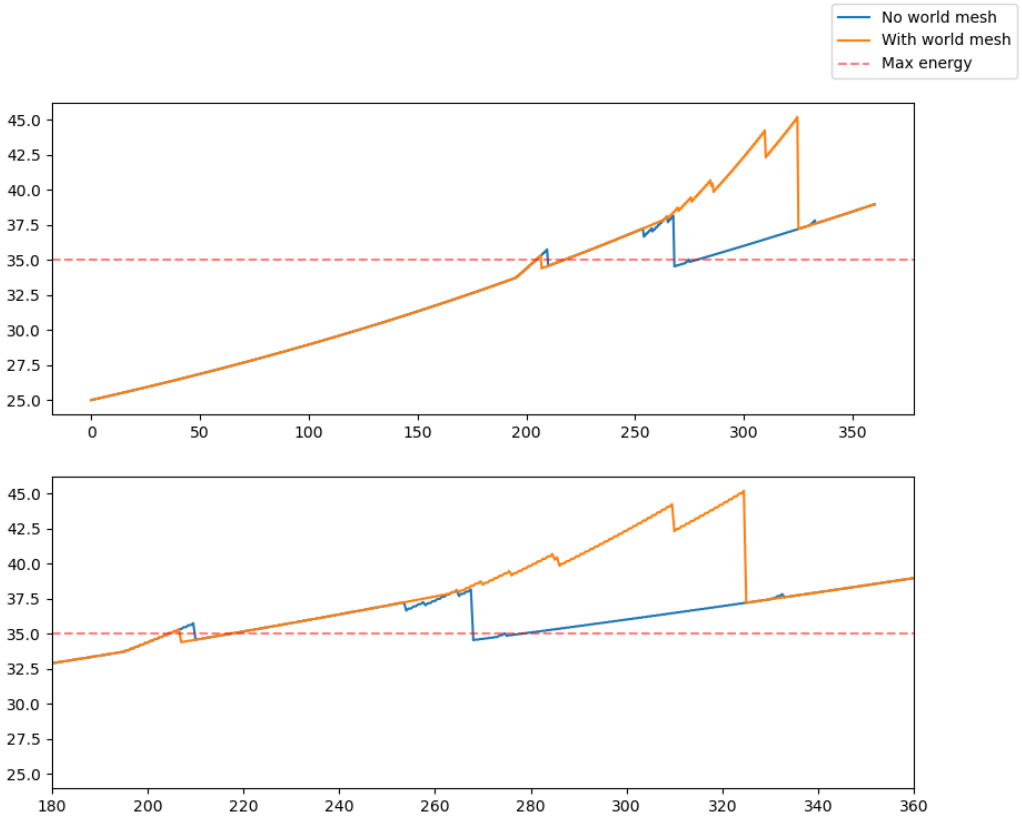


Figure 5.13: Evolution of the quality with and without using material space, for the full experiment (top) and a close view after the trigger of the remeshing (bottom). In orange the algorithm with a control mesh and in blue the algorithm without control mesh

5.2.4 Reverse experiment

The last observation made on these deformations consists of starting from the final mesh state after the completed experiment and applying the inverse deformation. The object returns to its initial shape but does not have its initial topology as the starting energy is lower than the remeshing threshold. Remeshing will only be performed once the threshold is crossed, meaning that the final reverted mesh will have elements of energy closer to the threshold than at its initial state. With better performances, we could try to set the threshold to the initial average mesh energy but in the current state of the implementation, this takes too much time to be worth testing.

6 Conclusion

In this work presented an implementation of a local remeshing algorithm for injective deformations of both two- and three-dimensional meshes. Injection is ensured thanks to the representation of the 3D object in two different spaces, material and world spaces. The first one is responsible for validating remeshing operations applied to the second. The algorithm is a hill-climbing method in which bad elements of the mesh go through four successive passes, each one responsible for tackling a different approach to improve the mesh.

The implementation was evaluated by subjecting the mesh to multiple deformations. Their goal was to identify optimal parameter values as well as evaluate the quality and performance of the remeshing.

6.1 Weaknesses and future work

Through our experiments, we observed situations in which the algorithm's implementation was not optimal, mainly two points: Performance and local optima. We saw that great deformations could not be recovered from, even with small timesteps, and that the algorithm was sensible to its configuration. Where timestep size would give widely different values but not only in increasing quality, meaning that finer deformation steps would sometimes perform worse than bigger ones.

Performance is linked to the concept of hill climbing. Only taking steps in the summit direction means that any other step evaluated was run for nothing. If a hundred possible steps are available but only the last one leads us in the right direction, the 99 others were computed for nothing. And this right step at the time might not even be the best choice if we take into account the step after. In our implementation, a remeshing step a timestep t is not aware of the upcoming deformation step. This means that the next step might result in a worse mesh after remeshing than had we done nothing. This problem links both weaknesses of the project: by going in the wrong direction of local optima, we then have to perform more remeshing steps to recover thus, impacting performances.

A way of fixing this would be to have some kind of temporal knowledge. With it, we would be able to take into account preceding timesteps and approximate the next deformation step to better evaluate our remeshing choices. However, this does not improve the handling of big deformations. Improving performance would be beneficial for solving both the issue of treating big deformations as well as making

wrong remeshing choices. By performing more remeshing passes we would be able to extend the treated area around the bad element and thus regularize it to avoid surprises at the next remeshing step. To take again the hiker image, if more hikers are searching for the top of the hill, the chances of finding the summit increase.

6.2 Use of AI tools

Please note that the LLM ChatGPT was punctually used in the redaction of this report to improve the quality and readability of the text.

Bibliography

[1] *Reviving the Search for Optimal Tetrahedralizations*. Zenodo, February 2020.

[2] Adam W. Bargteil, Chris Wojtan, Jessica K. Hodgins, and Greg Turk. A finite element method for animating large viscoplastic flow. In *ACM SIGGRAPH 2007 Papers*, SIGGRAPH '07, page 16–es, New York, NY, USA, 2007. Association for Computing Machinery.

[3] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[4] Lori Freitag, Mark Jones, and Paul Plassmann. A parallel algorithm for mesh smoothing. *SIAM Journal on Scientific Computing*, 20(6):2023–2040, 1999.

[5] Steffen Hinderink and Marcel Campen. Galaxy maps: Localized foliations for bijective volumetric mapping. *ACM Trans. Graph.*, 42(4), jul 2023.

[6] Bryan Matthew Klingner and Jonathan Richard Shewchuk. Aggressive tetrahedral mesh improvement. In Michael L. Brewer and David Marcum, editors, *Proceedings of the 16th International Meshing Roundtable*, pages 3–23, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[7] A. Rassineux. 3d mesh adaptation. optimization of tetrahedral meshes by advancing front technique. *Computer Methods in Applied Mechanics and Engineering*, 141(3):335–354, 1997.

[8] Jonathan Shewchuk. Two discrete optimization algorithms for the topological improvement of tetrahedral meshes. 01 2002.

[9] Justin Solomon, Leonidas Guibas, and Adrian Butscher. Dirichlet energy for analysis and synthesis of soft maps. *Computer Graphics Forum*, 32(5):197–206, 2013.

[10] Martin Wicke, Daniel Ritchie, Bryan M. Klingner, Sebastian Burke, Jonathan R. Shewchuk, and James F. O'Brien. Dynamic local remeshing for elastoplastic simulation. *ACM Trans. Graph.*, 29(4), jul 2010.

[11] Chris Wojtan and Greg Turk. Fast viscoelastic behavior with thin features. *ACM Trans. Graph.*, 27(3):1–8, aug 2008.