LiU-ITN-TEK-A-14/040-SE

Optimal Coherent Reconstruction of Unstructured Mesh Sequences with Evolving Topology

Christopher Birger

2014-09-22

Department of Science and Technology
Linköping University
SE-601 74 Norrköping, Sweden

Institutionen för teknik och naturvetenskap
Linköpings universitet
601 74 Norrköping

LiU-ITN-TEK-A-14/040-SE

Optimal Coherent Reconstruction of Unstructured Mesh Sequences with Evolving Topology

Examensarbete utfört i Medieteknik
vid Tekniska högskolan vid
Linköpings universitet

Christopher Birger

Handledare Gunnar Läthén
Examinator Jonas Unger

Norrköping 2014-09-22

Linkoping University

Master Thesis

Optimal Coherent Reconstruction of Unstructured Mesh Sequences with Evolving Topology

Christopher Birger*

September 28, 2014

*e-mail: chrbi049@student.liu.se

Abstract

This thesis work will investigate and implement a method for reconstructing an unstructured mesh sequence with evolving topology. The goal of the method is to increase frame-to-frame coherency of the triangulation. The motivation of the method is that many of current state-of-the-art mesh compression and decimation algorithms for mesh sequences are based on static connectivity. The method investigated in this thesis is mainly based on the work by presented in *Tracking surfaces with evolving topology* by Wojtan, Li and Bojsen-Hansen [1]. They use non-rigid alignment to deform a dynamic mesh to track a target mesh sequence. Topological changes such as splitting and merging are handled in two steps. First the mesh is converted into a signed distance field, discretized on a regular grid. Complex cell tests are used to detect any topological events. Finally, all the complex cells are cut out of the meshed and re-meshed using the distance field information. To handle robustness issues with cutting algorithm, we present a novel approach for grid based mesh cutting which does not use numerics to decide how to cut the mesh. We conclude that even though the method is appealing in theory and works well for simple cases. The method is, in the current state, not a valid approach for a production grade system where the input data can be arbitrarily complex.

Acknowledgements

I would like to thank my supervisors Jan Schmid, and Torbjörn Söderman for many great discussion and valuable input. I also would like to thank my examiner Jonas Unger and my supervisor Gunnar Läthen at Linköping University. Finally and most importantly, I would like to thank my wife Anna for the support, encouragement and patience during all the late nights and weekends.

Contents

List of Figures

1. Introduction

Animated three dimensional geometry has become a natural part in many areas over the last decade. Industries such as games and special effects have pushed the creative tools and ways processing such content. Animations are typically defined by a series of static meshes where each mesh, or frame, represents a certain time in the animation. The connectivity is most often static and the geometry is deformed over time by "key-framing" the vertices i.e., each frame lists the position of each vertex. With the increasing amount of complexity and detail it has become a great challenge to effectively store, and transmit such data. Therefore, some kind of compression has to be applied. A great deal of research has been made in this area which stems from static mesh compression and mesh simplification, where each frame is treated separately. There are also delta compression methods, called dynamic mesh compression. However, these methods assumes static connectivity and topology i.e., animations where connectivity is coherent between all frames and that the animation does not split or merge. One kind of animated meshes that does not meet these requirements are unstructured mesh sequences which does not have any correspondence information. The connectivity, or triangulation, can be completely different between consecutive frames. They also allow for topology changes such as splitting and merging. There are several examples of processes that generate such sequences e.g., fluid simulation and morphing. A mesh sequence from a fluid simulation is rarely coherent, it can fold over itself, small droplets can quickly appear and disappear and different regions can merge together or split apart. Fluids are often simulated using a grid structure, with particles or both. The fluid interface is then converted into a signed distance field where there are many methods of extracting a triangulated surface. However, small numerical differences in the simulation or distance field calculation can completely change the triangulation between frames. Fully coherent connectivity means that it is possible to track any vertex, $\mathbf{v}$ in all frames, $F_n \in [n..1]$ and that the neighbouring vertices also stay coherent. The importance of this property is highly favourable since most compression research is dependent on it or at least benefit from it [2]. Mesh simplification of animated meshes needs some information about how parts of the mesh deforms in order to construct heuristics that take deformation into account to avoid flickering and popping [3, 4, 5, 6, 7].

1.1. Problem Description

To be able to optimize mesh compression and use some of today's state-of-the-art dynamic mesh simplification schemes on non-coherent animations, it is of high importance to calculate at least an optimal coherent mesh sequence. Unfortunately, this is a not a trivial problem to solve. To the best of my knowledge, only the work of Wojtan, Li and Bojsen-Hansen [1] addresses the problem of optimizing coherency in unstructured mesh sequences. Their proposed method combines research in the fields of dynamic shape registration and explicit surface deformation with topology changes. Their approach is to track a triangulated surface by evolving a mesh via registration. To handle sequences with topology changes, they incorporate a step where the evolving mesh is matched against the current target frame. If needed their framework will cut out parts of the mesh which is not matching and then generate and stitch the non-matching parts.

1.2. Objectives

The objectives of this thesis is to research ways of optimizing temporal coherency of connectivity in unstructured animation data with evolving topology. With this research, implement a robust tool that can handle any closed triangulated unstructured mesh sequence. The focus is not performance and for reasons of limited time, we use approximation and simpler algorithms for some parts.

1.3. Structure of the thesis

The structure of this thesis is outlined below,

- Chapter 2 introduces the subject of surface tracking.
- Chapter 3 describes in detail the implementation together with theory on the work that is presented.
- Chapter 4 Shows result from the implementation with some discussion.
- Chapter 5 concludes the report and a further discussion of the result is presented. We also discuss improvement and future work.

2. Background

This chapter introduces the concepts and methods for surface tracking which is used in this thesis. Some of the previous work in this field is also presented and we touch on some of the challenges.

2.1. 3D Meshes

A polygonal model or a mesh is defined by a set of vertices, edges and polygons (or faces). Each edge connects two points and each polygon is bounded by a set of edges. The most common type of meshes are triangle meshes. A triangle is the lowest order of a polygon which is composed of three vertices and three edges. The shape of the mesh is defined by the placement of the vertices and the topology. The topology of the mesh describes the connectivity of the elements i.e., which faces are adjacent to each other and how edges connect the vertices. Even though two meshes have vertices with exactly the same positions, different topology will completely change the shape of the surface which is illustrated in figure 2.1

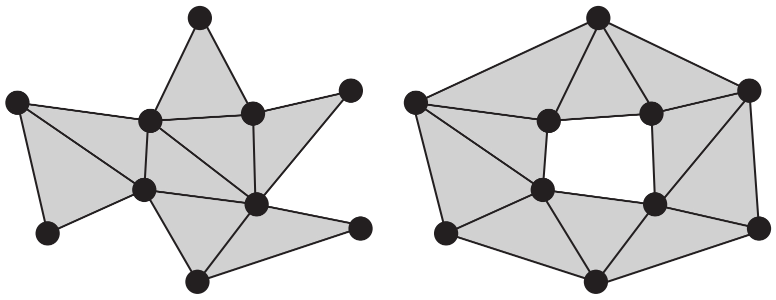
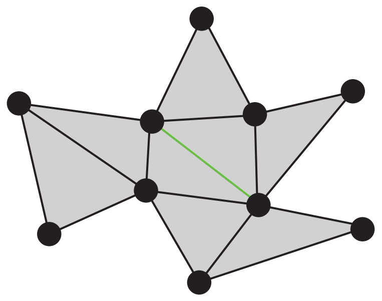


Figure 2.1.: Comparison of shapes with same positions but different topology.

The topology of a mesh can refer to either the triangulation i.e, the connectivity or the actual surface that the mesh represents. The field of topology is vast and we will not go into any long discussion in this thesis. However, there are some notions that are used throughout the thesis. A mesh is a manifold, more specifically it is a 2-manifold. A 2-manifold is a surface in three dimensions which at every point on the surface defines a 2-dimensional euclidean space [8]. Basically, at every point on the surface there is a well defined euclidean coordinate system.

A non-manifold mesh is a mesh that has edges that connect to multiple faces or faces that only connect through a single vertex. We want to avoid this because most mesh operations can not handle non-manifolds. For instance, queries such as "find the opposite face of this edge" are not well defined. A manifold can also be defined as being open or closed. A closed manifold does not have a boundary. In the case of meshes this means that all edges on the mesh connects two faces. This is important because an open mesh does not have a well defined inside and outside.

The topology of a mesh can also refer to the connectivity of the mesh elements. For instance figure 2.2 illustrates a mesh where one of the edges has been flipped. The two meshes are topologically equal, or homeomorphic, in terms of the surface the mesh describes. However the topology in terms of connectivity is not equal.



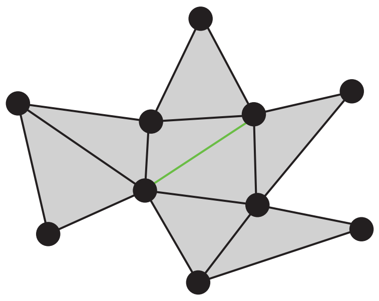


Figure 2.2.: Geometrically equal topology with different topological connectivity.

In this thesis the input data is an unstructured mesh sequence with evolving topology. A mesh sequence is a set of meshes where each mesh represents the sequence at some point in time, each mesh can be considered to be one frame in an animation. An unstructured mesh is a mesh where the connectivity those not follow a set pattern. For example, every vertex can have different valence i.e., the number of connecting edges is not constant. Evolving topology means that the topology can change from one frame to another. A simple example is a sphere deforming into a donut. There exist no continuous mapping from a sphere into a donut thus they are not homeomorphic and can't be represented by the same connectivity.

2.2. Surface Tracking

Tracking surfaces with evolving topology is a fairly new research area introduced by Wojtan, Li and Bojsen-Hansen [1]. It was developed to alleviate the problems of surfaces that alters topology over time, especially how points on the surface correspond to each other between frames. Their mesh tracking framework is based on research in the field of dynamic geometry processing and the problem of finding coherency between frames in mesh sequences [9]. This field of geometry processing has been used for tracking and reconstructing scanned 3d data. Scanned 3d data has no correspondence information, it often contains holes and it is usually noisy. In [10] and later [11], Li et.al., applies registration techniques to reconstruct and complete partial 3d scans with and without the use of templates. There method is based upon a non-rigid registration model which is able to optimally deform a source mesh into a target mesh. Surface tracking is also used within the field of physical simulations, especially for liquids. Liquid simulations requires full knowledge of its liquid-air interface in order to correctly discretize the Navier-Stokes equations. A widely used method is the Fluid Implicit Particle Method (FLIP) [12, 13]. The FLIP scheme uses particles for advecting the fluid interface but solves the pressure equations on a grid by interpolating velocities from the particles. By solving pressure on a grid the detail of the interface is bound to the resolution of the grid which leads to a greater errors in areas with thin features. There is also the problem of reconstructing a surface from the particles. As discussed in chapter 1, the common approach is to convert the grid or particles into a levelset and then use methods such as Marching Cubes [14]. Even if the particle representation of the interface can be highly detailed, the levelset must be discretized with some set resolution. The resolution of the surface reconstruction will always be bound by the resolution of the grid.

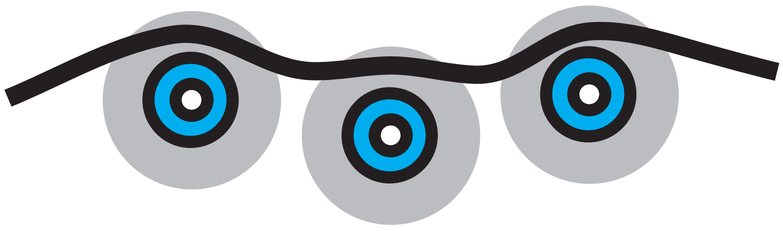


Figure 2.3.: Surface reconstructed using particle to levelset conversion.

Particle mesh reconstruction also suffers from noise due to the particle to levelset conversion which is the standard approach, see Figure 2.3. To improve the detail in both the simulation and surface reconstruction there has been research on how to use an explicit triangle mesh directly [15]. However, deforming meshes that undergoes extreme deformations and topological changes pose a great deal of challenges. As the mesh is advected, triangles can start to have poor geometrical quality. Triangles can become very large which leads to visual artefacts and poor detail preservation. Triangles can also become infinitesimally small which leads to greater memory needs and computational time, see [16] for more information about mesh quality. Jiao et.al., describes three mesh improvement operations that deals with poor quality triangles [17]. These are *Edge flip*, *Edge Collapse* and *Edge Split*, see section 3.2 for details. Mesh improvement is quite straightforward and fairly well researched. However, dynamically dealing with changing topology and is quite tricky. Cases which must be handled are splitting and merging. Wojtan et.al converts the mesh into a fixed resolution signed distance field and uses specifically designed tests to determine topological events [18]. Each cell is essentially checked if it can locally represent the surface with a marching cubes template, it is otherwise marked as *complex*. For instance, if a cell edge intersects multiple surface each surrounding cell is complex. The complex regions are then cut out and replaced by the marching cubes template of the cell. Their approach automatically handles self-intersection due to the levelset based re-meshing stage. However, the cutting stage can sometimes fail and the scheme is limited to closed non-manifold meshes. Brochu et.al., handles open surfaces and robust topological changes [19]. Merging is handled by checking for edges that are close to each other. When a separated edge pair is "near", the two edges and their adjacent faces are removed. This creates two edge loops with four vertices each. The two holes are then stitched together using eight new triangles. To allow the mesh to split the mesh is allowed to be temporarily non-manifold. In their mesh improvement stage, the edge flip and edge collapse passes are allowed to create triangles that shares all three vertices and triangles with repeated vertices. All such triangles are removed. Then they sweep the mesh for triangles that are only connected by a single vertex. These vertices are removed and for each triangle connected to the removed vertex, a new vertex is created per triangle. Compared to Wojtan et.al., Brochu's approach does not handle self-intersection. Instead they use a continuous collision detection scheme while they advect the vertices thus enforcing self-intersection throughout the deformation [19]. Note that Brochu's approach requires knowledge about how the surface evolves between each frame. Therefore, it is suited for simulation based surface tracking where the surface is advected through a velocity- or forcefield.

3. Method and Implementation

This chapter describes the details of the chosen method and implementation. It will cover the different data structures, schemes and algorithms that were used.

3.1. Overview

The input of the system is an arbitrary unstructured sequence of manifold and closed triangular meshes. The goal of the system is to output a sequence of optimally structured manifold and closed triangular meshes. Optimal in the sense that, if the input is a sequence of randomly triangulated non deforming meshes the output should be perfectly structured. However, if the sequence undergoes extreme deformation there are some limitations. If the topology changes, it is impossible to keep the same connectivity. Nevertheless, it should only be necessary to alter the areas of the mesh where the topology change occurred. Since the mesh tracking deforms the mesh, some triangles might become small or stretched. To preserve "good" triangles, mesh improvement have to be performed. Note that, mesh improvement changes the connectivity. Therefore, we want to keep that operation ta a minimum. The only published method that has all of these properties is the work by Wojtan et.al., [1]. Though, the numerical issues in the their cutting stage can sometimes fail which is not acceptable in a production system. Therefore, a new approach to cutting and re-meshing is proposed which only uses numerics as a guide, see section 3.4.5. Our approach closely follows the method proposed by Wojtan et.al., except from the topology change step. It is comprised of three operations which is mesh improvement 3.2, surface registration 3.3 and topology change 3.4. The input sequence of meshes is defined as $T_i, i \in [0, 1, ..., n]$, where n is the number of frames in the sequence. We denote the input with T since this is the target of the output. The output sequence is denoted with $M_i, i \in [0, 1, ..., n]$. The different steps of the method which will be described in this chapter is outlined below.

1. Perform mesh improvement on the current output frame M_i which yields a temporary output $\hat{M}_{i+1}$.
2. Deform $\hat{M}_{i+1}$ to fit the next target frame T_{i+1} using surface registration.
3. Detect topological and mismatching parts of $\hat{M}_{i+1}$ compared to T_{i+1}.

4. Cut out the faulty geometry from $\hat{M}_{i+1}$.
5. Re-triangulate the cut out holes from $\hat{M}_{i+1}$ using surface information from T_{i+1}. This gives the final result, M_{i+1}.

3.2. Mesh Improvement

As mentioned in 2.2, the geometrical quality of the triangles can become poor during deformation and stitching. Optimally we would like to have equilateral triangles which are also regular i.e, all sides are of same length and all angles measure 60 °C. Regular triangles are beneficial in e.g., parallelogram prediction since the error vector becomes smaller [20]. The mesh improvement is comprised of three different operations which are *edge flip*, *edge collapse* and *edge split*.

3.2.1. Edge Flip

Edge flipping is useful to get rid of thin stretched triangles. The *Edge Flip* flips an edge so that the edge connects over the two opposing vertices. This operation can't produce non-manifolds. However, it can change the geometry and create faces with flipped normals as seen in figure 3.1.

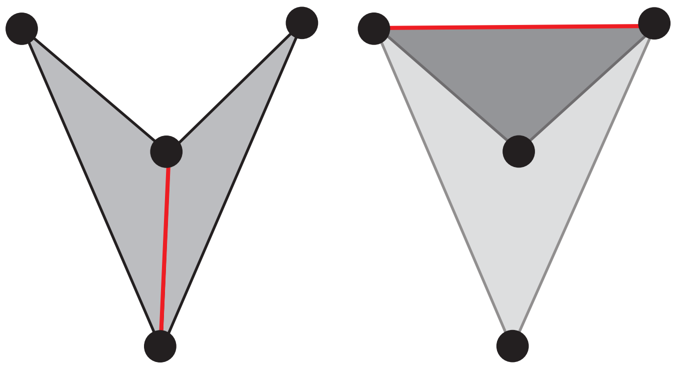


Figure 3.1.: Edge flip generating flipped normal.

To check if the *Edge Flip* will flip any triangle faces, the sign of the dot product between the two old face normals must be the same as the dot product of the new normals of the new faces. An *Edge Flip* can change the geometry quite drastically if the angle between the two edge-adjacent faces is small. This can be prevented by comparing the volume of the tetrahedron defined by the four points making up the two faces, before and after the edge split [19]. If the change in volume is greater than some set threshold the *Edge Flip* is not valid.

3.2.2. Edge Collapse

The *Edge Collapse* removes an edge in the mesh and merges the connecting vertices resulting in that the two faces adjacent to the edge is removed. To choose the position of the resulting single vertex many different schemes can be used. Brochu et.al., propose a butterfly subdivision scheme to choose the location of the vertex **P** [19]. The equation is given below with accompanying figure 3.2.

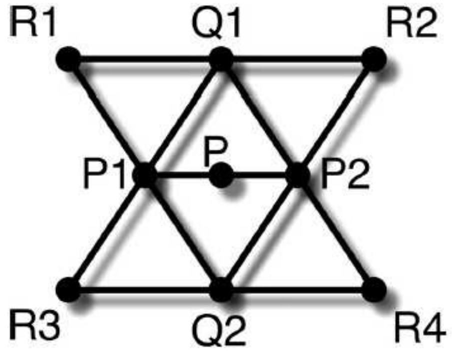


Figure 3.2.: Butterfly subdivision.

$$P_{new} = \frac{1}{16}(8(P_1 + P_2) + 2(Q_1 + Q_2) - (R_1 + R_2 + R_3 + R_4)) \tag{3.1}$$

The edge collapse can however introduce some problems in certain cases. When collapsing edges that lie on the ring around a valence 3 vertex, the opposite triangle faces will "fold" over themselves. This produces a non-manifold connectivity which is often a property that many algorithms and data structures can't handle, such a case is depicted in figure 3.3. When the red edge is collapsed, triangle A and B will share all its vertices and edges.

3.2.3. Edge Split

The *Edge Split* takes as input a given edge and its two adjacent faces. A vertex is inserted somewhere along the edge producing two new edges splitting the faces, thus creating a total of four new faces, see figure 3.4. As long as the new point is inserted on the original edge, the *Edge Split* does not change the geometry.

3.2.4. Summary

Mesh improvement is a separate step in our mesh reconstruction pipeline. It is driven by a set of parameters that states the longest and shortest allowed edge, face area and triangles angles. We apply edge- flip, collapse and split on the whole

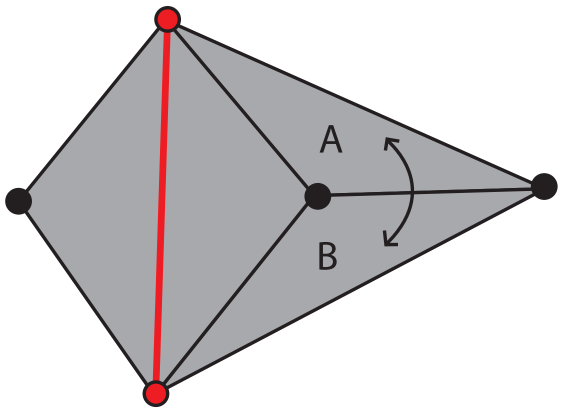


Figure 3.3.: Edge Collapse leading to non-manifold mesh.

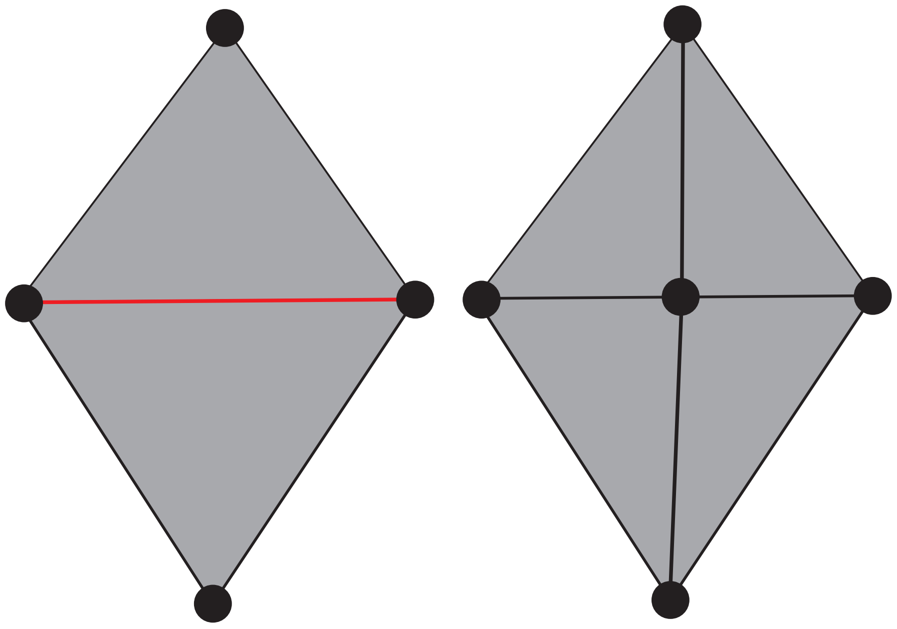


Figure 3.4.: Examples of Non-Manifold Surface (Left), Manifold Surface (Right).

mesh in three separate passes. Each of the operations are also allowed to run a set number of passes on the elements of the mesh. For each pass, each element is tested. If it does not meet the requirements the operation is carried out.

3.3. Surface Registration

Surface registration tries to solve the problem of finding a deformation, $\mathbf{f} \to \mathbb{R}^3$ such that $M_{i+1} \approx \mathbf{f}(M_i, T_{i+1}) \approx T_{i+1}$. The problem is depicted in figure 3.5.

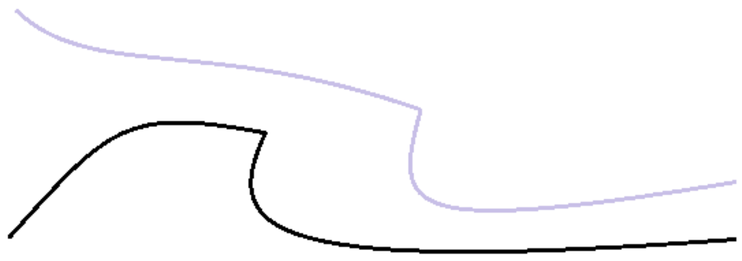


Figure 3.5.: Surface registration.

We use the methods proposed by Li et.al in *Robust single-view geometry and motion reconstruction* [10]. Their method uses a deformation graph which was first used by Sumner et.al in *Embedded deformation for shape manipulation* [21]. Each node in the graph corresponds to a vertex on the mesh. Correspondence vectors are calculated for each graph node which is the vector given by connecting each graph vertex with the best estimated corresponding point on the target mesh. A coarse non-rigid alignment is applied using the deformation graph. The final step is a detail registration step where all vertices of M_i are used to align it with T_{i+1}.

3.3.1. Deformation graph

The deformation graph, $\mathbf{D}_g$, is a coarse representation of the mesh where the nodes of the graph are chosen from the vertices of the underlying mesh. To build the graph, the mesh is regularly sampled with approximately four times the average edge length between graph nodes. This is done by starting at a random vertex and growing a region that is approximately four times the average edge length by walking through mesh edges. At the boundry of the region we choose a new vertex and repeat the process until all vertices has been visited. Each node of the graph is then assigned an affine transformation $\mathbf{R}$ and a translation vector $\mathbf{t}$. The deformation graph affects a vertex on the mesh, $\mathbf{v}$, to $\widetilde{\mathbf{v}}$ with the following equation [21].

$$\widetilde{\mathbf{v}_j} = \sum_{i \in N(\mathbf{v}_j)} \hat{\omega}_i(\mathbf{v}_j)(\mathbf{R}_i(\mathbf{v}_j - \mathbf{g}_i) + \mathbf{g}_i + \mathbf{t}_i) \tag{3.2}$$

Where j denotes the index of a vertex in M_i and i denotes the i:th deformation graph node. $\mathbf{g}_i$ is the position of the i:th node. The set, $N(\mathbf{v}_j)$ is all nodes with $\omega_i(\mathbf{v}_j) > 0$. The weight $\omega_i(\mathbf{v}_j)$ is a blending factor which sums to 1 for the set, $N(\mathbf{v}_j)$, of nearest graph nodes, 4 was used in the tests. The weights are calculated per vertex and is given by equation 3.3 and are normalized as in equation 3.4 [21].

$$\omega_i(\mathbf{v}_j) = (1 - \|\mathbf{v}_j - \mathbf{g}_j|/d_{max})^2 \tag{3.3}$$

$$\hat{\omega}_i(\mathbf{v}_j) = \frac{\omega_i(\mathbf{v}_j)}{\sum_{k}^{k_{max}} \omega_k} \tag{3.4}$$

Where d_{max} is the distance to the $k+1$:th nearest neighbouring node.

3.3.2. Correspondence Vectors

Correspondence vectors are what drives both the coarse and detail registration. They are simply specifying the vector connecting each node or vertex to a point on the target mesh. There are several methods of finding correspondences between two input meshes. Sumner et.al., [21] uses a KD-tree to find the k nearest nodes. Wojtan et.al., [1] proposes to instead project points onto to target mesh in the direction of normal. We use the latter of the two. The two different approaches are illustrated in figures 3.7 and 3.6.

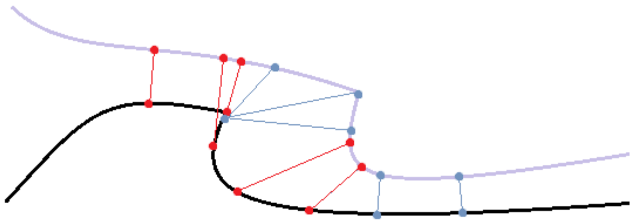


Figure 3.6.: Correspondence vectors by closest point. Black line is source with red correspondence vectors. Blue line is target with blue correspondence vectors.

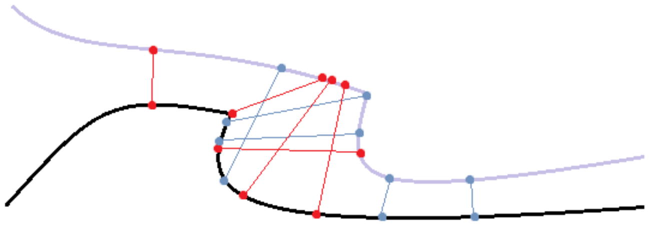


Figure 3.7.: Correspondence vectors by projection in normal direction. Black line is source with red correspondence vectors. Blue line is target with blue correspondence vectors.

Finding "good" correspondences is extremely difficult and both these methods are quite naive. If we were to just move the points to their respective point on the target surface we would get self-intersections. However, with the normal direction approach we also compare the direction of the source and target normal for each correspondence. By doing this we can prune away correspondences where the normals do not agree, as illustrated in figure 3.8.

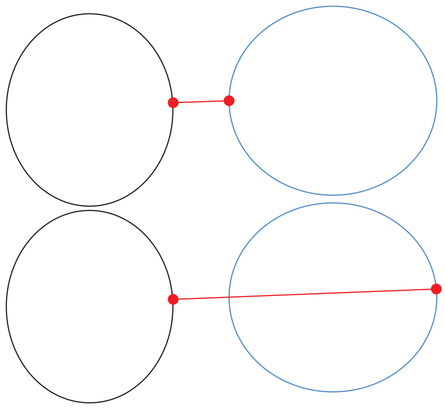


Figure 3.8.: Pruning Correspondence vectors by comparing surface normal of the source and target. Top) Correspondence is faulty since normals disagree. Bottom) Correspondence is chosen since surface normals agree.

3.3.3. Coarse Registration

The coarse registration is a non-rigid registration, which means that the surface is allowed to deform. Compared to rigid registration which only uses rotation and translation to deform the surface globally. Given the correspondence points, the optimal affine transformation for each node, is found via minimizing set of energy functions. These are defined in equations 3.5, 3.6 and 3.7 which are defined below [21]. The following energy function measures how far $\mathbf{M}$ is from $\mathbf{T}$ and forces the mesh to move towards the target surface.

$$E_{fit} = \sum_{i \in \nu} \alpha_{point} \|\mathbf{g}_i + \mathbf{t}_i - \mathbf{c}_i\|^2 + \alpha_{plane} \langle \mathbf{n}_i, \mathbf{g}_i + \mathbf{t}_i - \mathbf{c}_i \rangle^2 \tag{3.5}$$

α_{point} and α_{plane} are set to 0.1 and 1.0 respectively which lets the source slide along the target surface. The second term, E_{rigid}, minimizes distortion and scaling of the matrix $\mathbf{R}_i$ such that is a close to a rotation matrix. This maximizes the rigidity of the transformation [21].

$$E_{rigid} = \sum_{i \in \nu} \langle \mathbf{r_{i1}}, \mathbf{r_{i2}} \rangle^2 + \langle \mathbf{r_{i1}}, \mathbf{r_{i3}} \rangle^2 + \langle \mathbf{r_{i2}}, \mathbf{r_{i3}} \rangle^2 + (1 - \|\mathbf{r_{i1}}\|)^2 + (1 - \|\mathbf{r_{i2}}\|)^2 + (1 - \|\mathbf{r_{i3}}\|)^2 \tag{3.6}$$

Where $\mathbf{r_{ij}}$ is the j:th column vector of the i:th graph node matrix $\mathbf{R}_i$. The last term, E_{smooth}. ensures that connected graph nodes are transformed smoothly in relative to each other.

$$E_{smooth} = \sum_{i \in \nu} \sum_{j \in N(i)} \|\mathbf{R}_i(\mathbf{g}_j - \mathbf{g}_i) + \mathbf{g}_i + \mathbf{t}_i - (\mathbf{g}_j + \mathbf{t}_j)\|^2 \tag{3.7}$$

Li et.al [10], propose weighting the influence of E_{smooth} between each pair of graph nodes to handle non-uniformly sampled deformation graphs. However, as we sample the mesh approximately uniformly we set this weight to one for all pairs. This is suggested by Sumner et.al [21] and is reported to not cause any noticeable artefacts. The complete energy function is minimized using a standard Gauss-Newton solver based on Cholesky decomposition. That is, we want to minimize the following.

$$\begin{aligned} \underset{\mathbf{R}_0,\mathbf{t}_0,\ldots,\mathbf{R}_i,\mathbf{t}_i}{min} \; & \alpha_{fit} E_{fit} + \alpha_{reg}(E_{rigid} + 0.1 E_{smooth}) \\ & \text{subject to } \mathbf{R}_i = \mathbf{I}, \mathbf{t}_i = \mathbf{0}, i \in \nu \end{aligned} \tag{3.8}$$

Li et.al [10], proposes a relaxation scheme to prevent the solver getting stuck in local minima. α_{reg} and α_{fit} are initialized to 100 and 0.1 respectively. When the relative change $(E_{tot}^{n+1} - E_{tot}^{n})/E_{tot}^{n}$ falls below a certain threshold $\sigma = 0.005$, α_{reg} is divided by 10. The relaxation is carried out until $\alpha_{reg} > 0.1$, we then calculate new correspondence points and restart the solver with the original weights. The total minimization scheme is said to converge when $(E_{tot}^{n+1} - E_{tot}^{n})/E_{tot}^{n} < 1e-5$ or $abs((E_{tot}^{n+1} - E_{tot}^{n}) < 1e-7$.

3.3.4. Detail Registration

The coarse registration only registers the basic shape. To obtain a greater level of detail we use a detail registration step as describe by Li et.al in [10]. We try to find a displacement, d, along the normal from each vertex while maximizing the smoothness of d between connected vertices.

$$E_{detail} = \sum_{i \in \mathbf{M}_\nu} \|\mathbf{v}_i + d_i \mathbf{n}_i - \mathbf{c}_i\|^2 + \sum_{(i,j) \in \mathbf{M}_\xi} |d_i - d_j|^2 \tag{3.9}$$

$\mathbf{M}_\nu$ is the set of vertex indices in the mesh and $\mathbf{M}_\xi$ is referring to the mesh edges.

3.3.5. Summary

The surface registration can be summarized with the following steps,

1. Build a Deformation Graph by sampling the mesh vertices uniformly.
2. For each vertex in $\mathbf{M}$ find the four geodesically closest graph nodes and calculate the weight of influence, $\omega(\mathbf{v})$.
3. Use coarse registration to fit the basic shape of $\mathbf{M}$ onto the target mesh.
4. Improve detail of the coarse registration by solving the detail registration.

The next section will describe how it is possible to fix cases when the registration fails to find a good solution.

3.4. Grid Based Topology Changes

It is impossible to find a deformation that deforms M_i into T_{i+1} if the two meshes has different topology. Therefore, we need a way to identify and fix these errors at parts of the mesh that could not be matched in the registration stage. There can also be cases where the registration simply fails even though the topology is the same. The method used was first proposed by Wojtan et.al., [18] which we covered a high level in section 2.2. We use the same approach in this thesis. However, we propose a new method of cutting and stitching the mesh that is based on mesh rasterization. The details of this method and the motivation is covered in section 3.4.5. The outline of the topology change step is

1. Rasterize M_i and T_{i+1} into a cell grid, see section 3.4.2
2. Classify inside and outside, see section 3.4.3.
3. Classify complex cells, see section 3.4.4.
4. Cut out geometry from M_i that is inside complex cells, see section 3.4.5.
5. Re-mesh the cut-out parts of M_i using the implicit representation of T_{i+1}, see section 3.4.7.

Note that this algorithm relies highly on the implicit representation of the input and output mesh. Therefore we begin by explaining the surface representation that is used.

3.4.1. Implicit Surface representation

The goal is to replace the parts of the registered mesh M_i with geometry from T_{i+1}. To accomplish this the mesh is converted into an implicit representation. A regular grid is used which is at least the size of the bounding box of the mesh.

For each cell edge that intersects the surface, the point of intersection is stored together with the normal. Figure 3.9 illustrate a simple surface represented in this way.

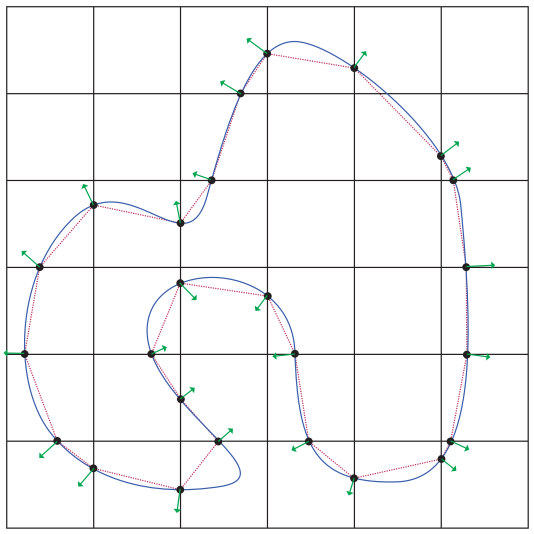


Figure 3.9.: Implicit surface representation. Blue line is original surface. Dashed red line represents the accuracy of the implicit surface. The circles and arrows are intersection points and normals respectively.

We will see how this representation also simplifies several of the test during complex cell classification.

3.4.2. Mesh Rasterization

The mesh cutting step requires all intersection information between the grid and the mesh. A cell contains a reference to each of the 8 corners, 12 edges and 6 faces. Each cell face has a list of mesh edges that intersects it. Each cell edge contains a list of the mesh faces it intersects and finally a cell corner has a flag weather it is inside or not. Triangles also store information about the cells. Each triangle stores a set of cells that overlaps the triangle. Each edge has a list of cells that it goes through. A vertex can be located inside a cell, on a cell face or on a cell edge. Therefore, it has a reference to all three cell elements. For a detailed description of this data structure see appendix A. Since only the surface of the mesh is of interest, only cells that are intersecting the mesh are stored. A hash map is used to store the cells. The hash key of each cell is generated by the three grid coordinates using equation 3.10 which was proposed by Teschner et.al [22].

$$hash(i, j, k) = (i\ 73856093\ xor\ y\ 19349663\ xor\ z\ 83492791)\ mod\ n \tag{3.10}$$

Where the three constants are large prime numbers and n is the size of the hash map. The grid size is given by the bounding box of the mesh and is calculated using equation 3.11.

$$(N_i, N_j, N_k) = \frac{1}{\Delta}(\mathbf{bb}_{Max} - \mathbf{bb}_{Min}) \tag{3.11}$$

Where the two **bb** defines the min and max bounding box points. The resolution or edge length of the grid is defined by Δ which is chosen to be half the average edge length of the mesh. The rasterizer is split into two stages. First all vertices are raserized and stored in the data structure. The cell is given by equation 3.12.

$$\mathbf{ijk} = \lfloor \frac{1}{\Delta}(\mathbf{v} - Grid_{\bar{\mathbf{0}}}) \rfloor \tag{3.12}$$

Where **ijk** is the integer grid coordinates, $Grid_{\bar{\mathbf{0}}}$ is the world space origin of the grid and $\lfloor \cdot \rfloor$ is a floor operation. Each triangle is then rasterized separately. The goal is to find the set of cells that the edges goes through together with the cell faces the edges intersects and finally any cells that only intersects the inside of the triangle. For each face we choose an edge which is defined by the two vertices $\mathbf{v}_{start}$ and $\mathbf{v}_{end}$ which has direction $\mathbf{d} = \mathbf{v}_{end} - \mathbf{v}_{start}$. From the vertex rasterization we know the start and end cell C_{start}, C_{end}. The idea is to walk from, C_{start} to C_{end}, only travelling through cell faces. The problem is then simply how to choose which cell face to walk through. This is done by comparing which centerpoint of the 6 faces is closest to the point intersected by **d**. We can limit the number of intersection tests needed to be done using the fact that the grid is axis aligned. For example, if the x component of **d** is greater than zero, the direction vector can't intersect the cell face left of the current cell. In total it is only required to do at most three intersection tests. These steps are repeated until reaching a cell that is a neighbour to C_{end}. Note that an edge is only rasterized once. When a triangle is adjacent to an already rasterized triangle we can simply query the stored data for the cells containing the shared edge. Figure 3.11 illustrate the final result where the view is projected in one of three axes. The yellow circles are the points where the direction vector **d** intersects each faces and filled cells are the rasterized cells. The final step for each triangle is finding the inner cells which are the cells that intersects the triangle yet does not contain any of the triangles edges. Because a triangle is a plane it is safe to assume that if there are at least two corners of the cell on opposite sides of the triangle. The cell must be intersected by the plane defined by the triangle. However, the inner cells must also be

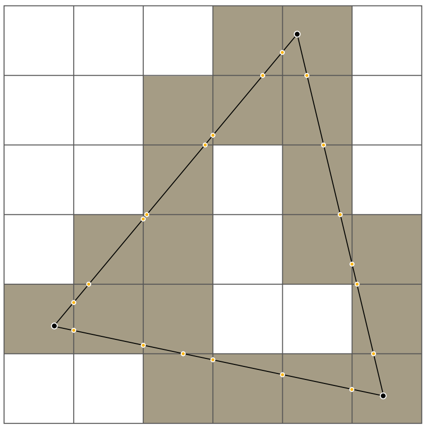


Figure 3.10.: Triangle edge rasterization.

within the boundary of the cells containing the edges. All cells with coordinates $\mathbf{ijk} \in [max(\mathbf{C}_{v_1}, \mathbf{C}_{v_2}, \mathbf{C}_{v_3}), min(\mathbf{C}_{v_1}, \mathbf{C}_{v_2}, \mathbf{C}_{v_3})]$ are considered as possible candidates, where $\mathbf{C}_{v_n}$ is the grid coordinate of vertex n. We begin by marking each corner in the grid bounding box as either *above* or *below*. This is done by simply checking the sign of the dot product between the normal of the plane and a vector defined by a point on the plane and the cell corner. If a cell has at least one corner *above* and one *below*, the cell is checked if it is inside bounds of the rasterized edge cells. This is done by, for each axes, projecting the cell onto the axis and checking if it lies within the max and min coordinates of the edge cells on the other two axes. Figure 3.11 illustrates the end result where the red circles are the points on the cell edges that intersects the triangle. Note that a cell edge that intersects the triangle must have four adjacent cells that also contains the triangle. This is a good validation that is run after the rasterization is complete.

3.4.3. Inside Outside Classification

To find all geometrical and topological differences between the source and target mesh we need to define which point on the grid is outside and inside. We use the simple yet robust method proposed in [15]. During the rasterization we store alongside the point of intersection, the normal of the intersected triangle. By comparing the surface normal and the direction of the cell edge it is possible to decide weather the intersection is *entering* or *exiting* the surface. The grid is sweeped in all directions, both positive and negative. By counting the number of times the surface is entered and exited it is possible to determine the sign of each grid corner. For each line of cell edges in the sweeping direction we initialize a counter to zero, see the orange line in figure 3.12. For each intersection on each

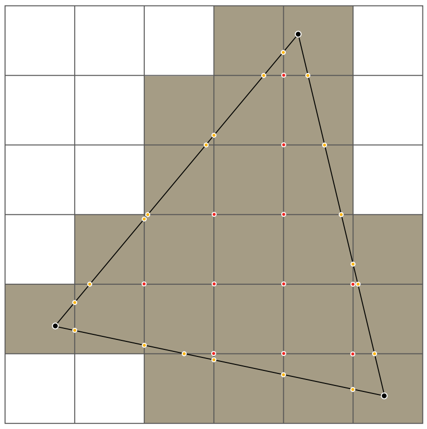


Figure 3.11.: Triangle face rasterization.

edge, the counter is incremented if the surface is entered otherwise decremented. The first cell corner is always set to be outside. After the intersections has been counted for each edge, the furthest away corner in the sweeping direction is set be inside if the counter is positive and outside if the counter is negative.

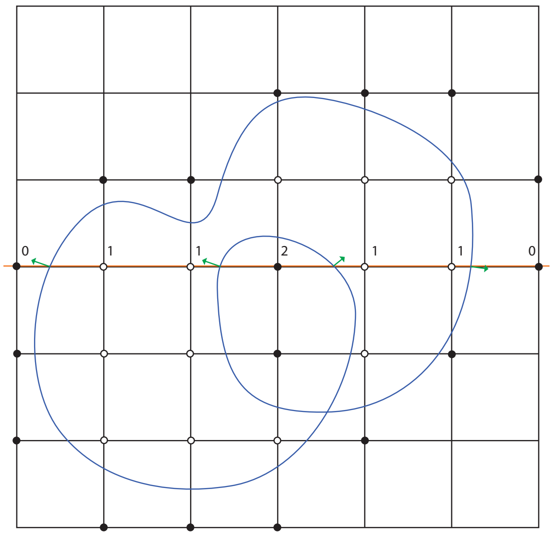


Figure 3.12.: Counting celledge intersections.

There is a degree of uncertainty in this approach due to numerical errors in the intersection tests. If a cell edge goes through an edge or a vertex it might be counted multiple times. Wojtan et.al proposes to clump together intersection points that lie within some threshold [15]. However, there is the possibility that sharp features are missed. This can be prevented by comparing all the triangles each cell edge intersects. If there are any adjacent triangles that are facing in the same direction only one intersection is counted. Self-intersection are also of interest. If the counter is greater than one or less than zero the corner is marked as *self-intersecting.*

3.4.4. Cell Classification

The cell classification step identifies topological differences between the input and the target mesh. We use complex cell tests introduced by Varadhan et.al [23] and later modified by Wojtan et.el to minimize surface re-sampling [18]. Cell tests and re-meshing are interconnected. As mentioned, re-meshing is solely based on the implicit representation of the input and output mesh. This means that new triangles are created per cell. The configuration of inside and outside cell corners determines the surface that is meshed in each cell. The same tests that Wojtan et.al proposes are used and we include them here for clarity,

- A *complex edge* is a cell edge in the grid that intersects the mesh more than once.
- A *complex face* is a cell face that intersects the mesh in the form of a closed loop or touches a *complex edge.*
- A *complex cell* is a cell that contains triangles and with corners of the same sign. It is also complex if any of the above is true.

These three tests are quite simple to check with the data structure explained in section 3.4.2. Therefore, with our data structure the rules are,

- A *complex edge* references more than one triangle.
- A *complex face* references more than three triangles edges.
- A *complex cell* only contains triangle faces and vertices i.e, no cell face references a triangle edge.

The target surface can in theory be any surface. Therefore, there might be boundary cells in the grid representation of the target mesh which does not exist in the source mesh. Therefore, cells that does not have the same inside/outside configuration as the target cell, are also marked as complex. Cells are also marked as complex if any of the corners of the cell has been marked as *self-intersting* in the inside-outside classification. One problem with these tests are that subtle variations in the surface can produce many complex cells which increases re-sampling. As mentioned, Wojtan et.al proposed a solution for this [1]. Their approach is that only variations greater than a cell causes re-sampling. They define a *deep cell*, which is a complex cell at least one cell away from the boundary of the surface. The final step in the classification is making sure that the regions of complex cells define a *simple* interface which is possible to stitch together with geometry generated from the target implicit surface. We do not directly use marching cubes templates as Wojtan et.al describes [18]. However, we must ensure the same condition. Therefore, the complex regions defined by the *deep cells* are grown until the boundary of the regions are non-complex.

3.4.5. Mesh Cutting

After the complex regions of cells has been classified, the triangles inside of the region has to be removed. To remove the triangles, the mesh is cut where the complex region intersects the mesh. In section 3.4.7 we desribe how to re-mesh the region which is cut. The used method is based on the work proposed by Wojtan et.al in *Deforming meshes that split and merge* [18]. For comparative reason we include a brief description of their method before describing our approach. The algorithm is illustrated in the four images below.

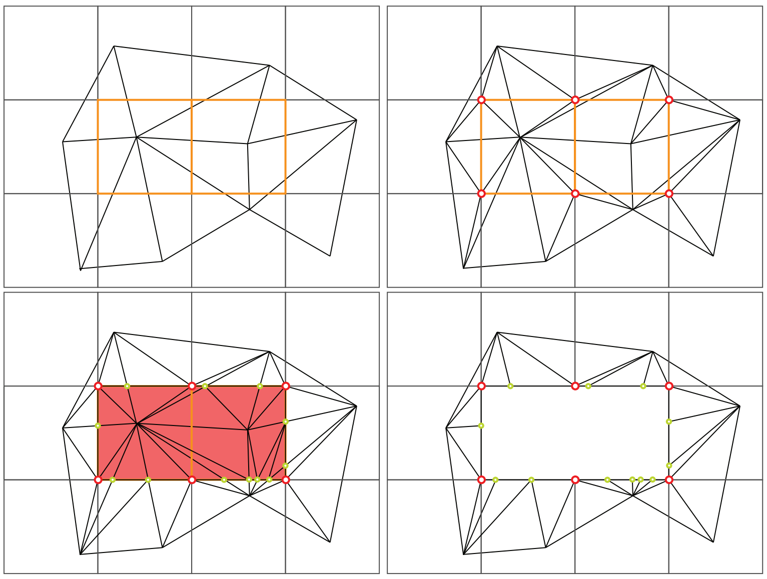


Figure 3.13.: Cutting out a complex region. *top left*: Complex region is colored with orange borders. *top right*: Find cell edge intersections on the border of the comlex region and perform *face split*. *lower left*: Find the intersecting between the mesh edges and the cell faces and perform edge splits. *lower right*: Remove the triangles inside the complex region.

The cells are processed one after another since the operation on one cell could affect other complex cells. The only cells that are processed are the cells on the boundary of the complex regions. First each intersection is found between the cell edges and the mesh. This is done by line to triangle intersection. A point that lies on a cell edge boundary is called a *type 1* vertex. The triangle is then split into three new triangles which can be seen in the upper right image in figure 3.13. The red circles shows all the *type 1* vertices. The second step is to split each mesh edge that crosses the complex cell region. These points are calculated by simply finding the intersection between the mesh edge and the cell face, see the green circles in the lower right image in figure 3.13. These vertices always lie on cell faces and are called *type 2* vertices. This scheme creates a loop of edges on the boundary of the complex region. Because Wojtan et.al uses marching cubes templates to generate the triangles inside the complex region there can not be any *type 2* vertices on the boundary. Therefore, they collapse all edges that has at least one *type 2* vertex. The resulting cut out region is shown in figure 3.14.

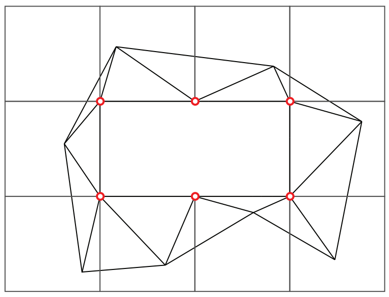


Figure 3.14.: Result after collapsing all boundary edges with at least one *type 2* vertex.

This approach is quite easy to implement. However, there are a lot of edge cases where the algorithm fails. The obvious reason is that the edge collapse operation can produce non-manifold geometry as we discussed in section 3.2.2. Wojtan et.al., states this case and their solution is to add the surrounding cells to the list of complex cells and repeat the flood fill [18]. Unfortunately this can, in theory, cause all cells to become complex. The second problem is that the algorithm assumes that the opposite mesh edges of a *type 1* vertex, must intersect the four surrounding cell faces. However, if a cell edge intersects a triangle edge this can't be guaranteed due to numerical instability. Figure 3.15 tries to show this but for illustrative purposes the dotted triangle edge has been moved. Imagine that the dotted edge is in fact going through the cell edge. Lets assume that the intersections between the triangle edges and the grid, that we can observe in the figure, is what the intersection tests would generate. Except for the dotted edge which we will discuss further.

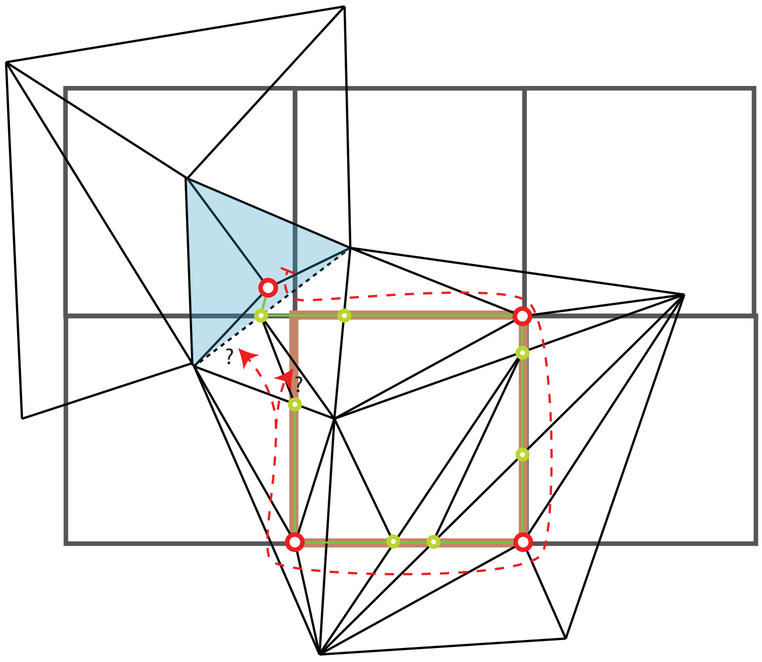


Figure 3.15.: Illustration of numerical issues in the intersection tests.

The first cell edge intersection test yields the blue tinted triangle. To connect the first two *type 1* vertices in the direction of the red dotted line, two mesh edges have to be split. That is, two *type 2* vertices must be created which is shown in the figure. However, since the dotted line is in fact going through the cell edge, in the top left corner of the complex cell, its not certain the intersection test agrees. If it does not, it is not possible to connect the first two *type 1* vertices walking through *type 2* vertices. Therefore, the end result will have an incomplete edge loop around the complex region. Lets assume that the intersection test say that the dotted edge intersects the cell face. If we follow the steps of the algorithm there are no problems until reaching the last *type 2* vertex shown in the figure. The question is which edges should be split to create a full edge loop around the complex cell? In fact, there is only one valid choice, which is splitting the dotted line. Choosing otherwise would result in an edge loop that intersects itself which can not be triangulated. But again, it is not possible to ensure that intersection test yields the correct result.

3.4.6. Grid based cutting

Several methods were attempted to alleviate issues discussed in section 3.4.5. However, we concluded that any approach that uses numerics to decide which faces and edges to split in each step can never be robust since one choice implies constraints on subsequent decisions. This is where our quite complicated rasterization data structure becomes useful. Remember that we stated that if a cell edge intersects the surface the four surrounding cells must contain the triangle.

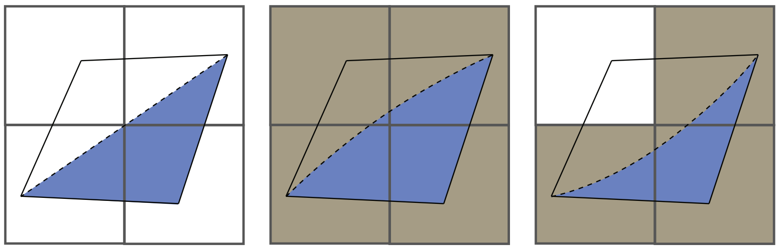


Figure 3.16.: Triangle and its two possible rasterized representations.

Figure 3.16 shows a triangle in blue shade, where the dotted line goes directly through a cell edge. The rasterization scheme describes in section 3.4.2 can have two possible outcomes shown in the two left images. The outcome is affected by the same inaccuracy as the intersection tests. However, the difference is that this decision is only made once. After the rasterization we know exactly which faces and edges to split, since these are stored. The grid based scheme is conceptually the same. The only difference is that the face and edge split are performed using the rasterization data. Because edge and face split can affect later operation the new triangles must also be rasterized. One important observation is used. Since the face and edge split operation does not change the surface neither will the voxel representation. Therefore, rasterizing new mesh elements is a matter of distributing the original voxels and figuring out which cell faces the edges intersects.

Grid Based Face-Split

For a complex boundary cell edge we already know which face to cut and where the new vertex should be positioned. Note that a cell edge on the boundary of the region can only intersect one triangle, otherwise it would be complex. The triangle, $\triangle$, defined by three vertices, $\mathbf{v}_1\mathbf{v}_2\mathbf{v}_3$ is contained by a set of cells $\zeta(\triangle_{\mathbf{v}_1\mathbf{v}_2\mathbf{v}_3})$. The three triangle edges are contained by the set of cells $\xi_{\mathbf{v}_1\mathbf{v}_2}$, $\xi_{\mathbf{v}_1\mathbf{v}_3}$ and $\xi_{\mathbf{v}_2\mathbf{v}_3}$. Figure 3.17 illustrates a triangle where the orange bordered cell is to be cut out. As before, the face is split into three new triangles where the new edges connect to the *type 1* vertex on the cell edge. The difference here is that the *type 1* vertex is stored in the cell edge.

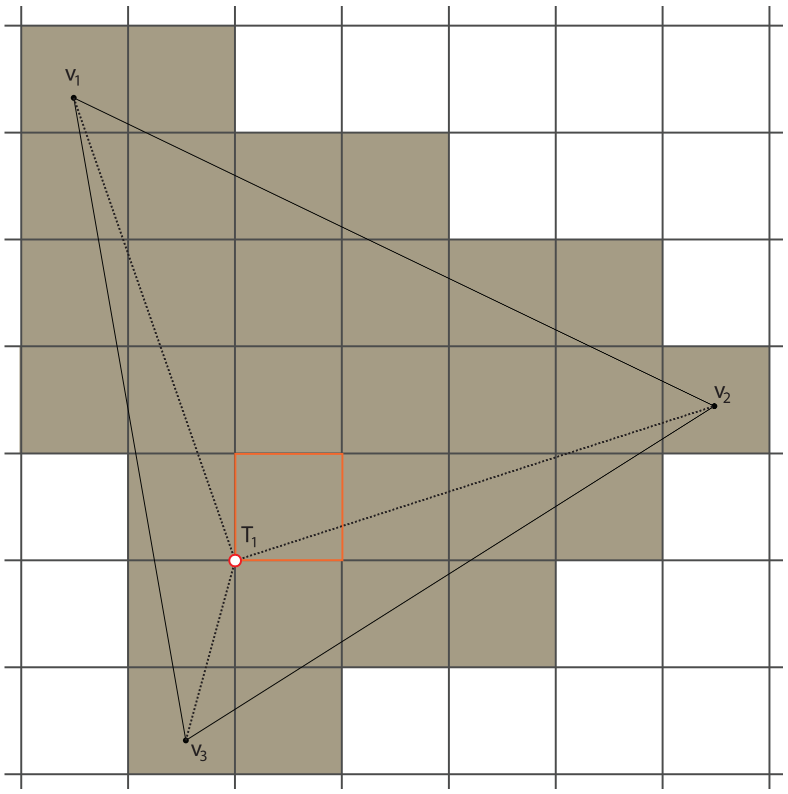


Figure 3.17.: Grid based face split before rasterizing new triangles.

As said, the face split does not change the voxel representation. Therefore the following rule applies,

$$\zeta(\triangle_{\mathbf{v_1 v_2 v_3}}) = \zeta(\triangle_{\mathbf{v_1 v_3 T_1}}) \cup \zeta(\triangle_{\mathbf{v_1 v_2 T_1}}) \cup \zeta(\triangle_{\mathbf{v_3 v_2 v_1}}) \tag{3.13}$$

where $\cup$ defines a set union. The first step is to rasterize the edges. Because equation 3.13 must hold, the set of candidates the edges can go through is defined by $\zeta(\triangle_{\mathbf{v_1 v_2 v_3}})$. For each cell edge we begin by finding the start and end cell. For each vertex on the edge, a list is stored with possible candidates. If the vertex is a *type 1* vertex all the four adjacent cells are inserted in the list. If the vertex is of *type 2* the two adjacent cells are inserted. If the vertex lies inside a cell, only one cell is inserted. The start and end cell is then determined by the pair of cells in the candidates lists that are closest using grid coordinates. Similar to the edge rasterization described in section 3.4.2, we walk from the start to end cell. The only difference is that the next cell must be one of the cells in $\zeta(\triangle_{\mathbf{v_1 v_2 v_3}})$. Figure 3.18 illustrates the set of new edge cells in blue shade.
Inner cells are found using the same method described in section 3.4.2. However, instead of considering the bounding cells, only the set of cells describes in equation

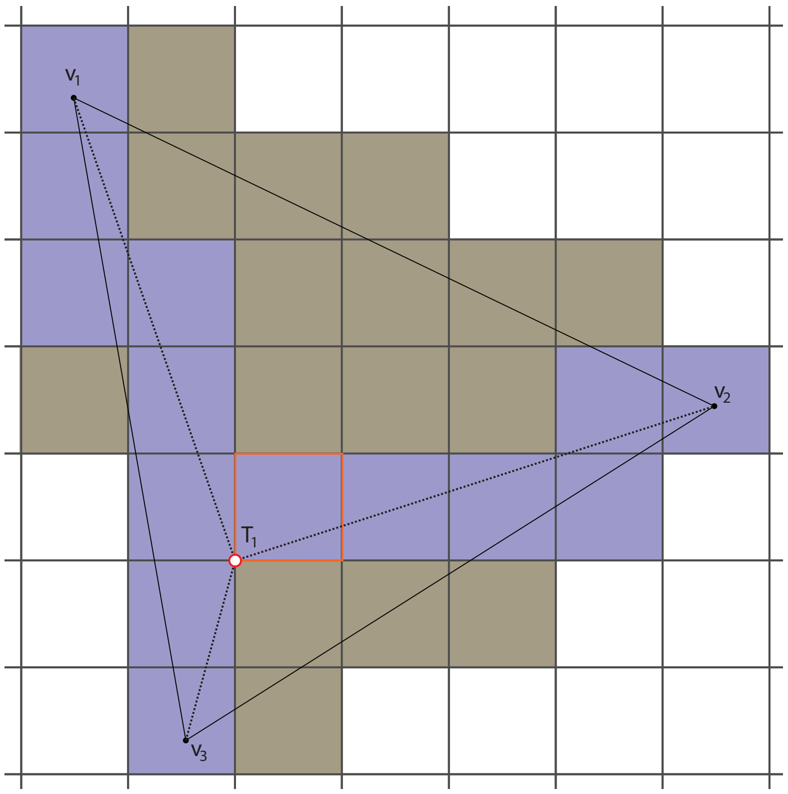


Figure 3.18.: Illustrating the cells that are chosen for the new edges.

3.14 are checked. For example,

$$\zeta_{inner}(\triangle_{\mathbf{v}_1\mathbf{v}_2\mathbf{T}_1}) = \zeta(\triangle_{\mathbf{v}_1\mathbf{v}_2\mathbf{v}_3})\backslash(\xi_{\mathbf{v}_1\mathbf{v}_2} \cup \xi_{\mathbf{v}_1\mathbf{T}_1} \cup \xi_{\mathbf{v}_2\mathbf{T}_1}) \tag{3.14}$$

where $\backslash$ is the set-theoretic difference.

Grid Based Edge Split

An edge split involves creating four new triangles and four new edges. As for the face split, the edges are treated first. The dotted edge that is split is contained by $\xi_{\mathbf{v}_1\mathbf{v}_2}$, refer to figure 3.19 where these cells are shaded blue.
The *type 2* vertex that splits the edge always lie on the cell face. Therefore, splitting the cell representation of the edge is basically a matter of finding each subset of cells that are on opposite sides of the cell face. A starting cell is chosen from one of the two original vertices $\mathbf{v}_1$ and $\mathbf{v}_2$ e.g, $\mathbf{v}_1$. Assuming that the set of cells in $\xi_{\mathbf{v}_1\mathbf{v}_2}$ are sorted in the direction $\mathbf{v}_1 \rightarrow \mathbf{v}_2$, the starting cell must be the first cell in the set. Therefore, there is no need to consider candidates which was done for the face split. After the starting cell has been determined we simply walk

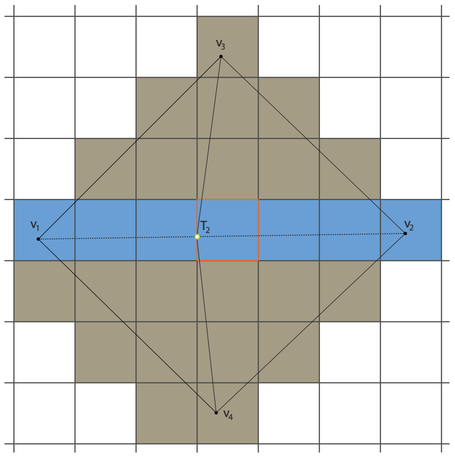


Figure 3.19.: Original set of edge cells before grid based edge split.

through cell faces until reaching the cell face where the split was made. All cells that were walked through are inserted in $\xi_{\mathbf{v}_1\mathbf{T}_2}$. The cells for the second edge is then given by remaining cell in the list which is shown in figure 3.20.

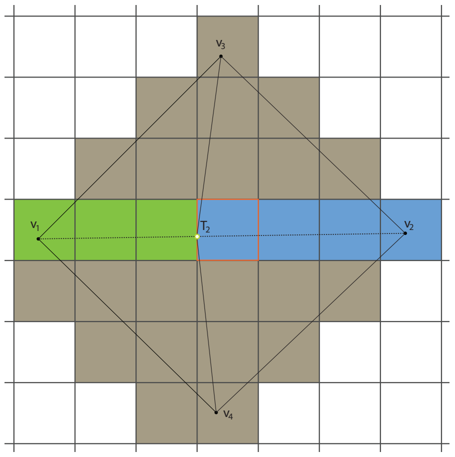


Figure 3.20.: Edge cells distributed to the two new edges created from splitting the edge.

The edge split can also be considered as dividing each of the two original triangles into to new triangles. Therefore, we exploit the fact that $\zeta(\triangle_{\mathbf{v}_1\mathbf{v}_2\mathbf{v}_4}) = \zeta(\triangle_{\mathbf{v}_1\mathbf{T}_2\mathbf{v}_4}) \cup \zeta(\triangle_{\mathbf{T}_2\mathbf{v}_2\mathbf{v}_4})$ and treat each original triangle separately, see figure 3.21. The same method as for the face split is then used for both the edge and the two new faces.

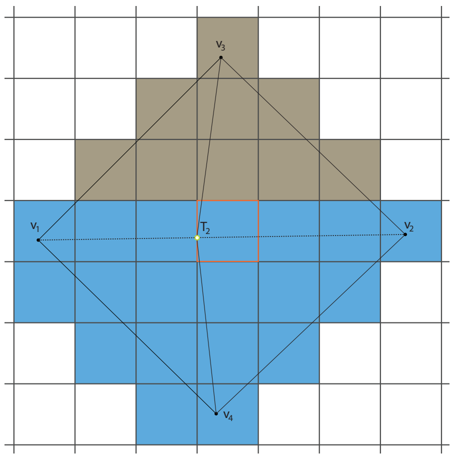


Figure 3.21.: Original set of cells that can be distributed to the two new triangles created from the edge split.

Summary

The problem of using intersection tests during the cutting is that each choice of triangle to split creates rules for which mesh edges that has to be split to be able to close the edge loop around the complex region. This can not be guaranteed in cases where e.g, an edge is close to parallel to a cell face, a vertex is close to a cell face or edge or a cell edge intersects a triangle edge. A grid based approach does not suffers from these problems because we only make the decisions once. In the case illustrated in 3.15, if the rasterization data states that the cell edge intersects the blue shaded triangle, it is guaranteed that the dotted edge will intersect the cell face to the right of the *type 1* vertex. With the cell based approach the example in 3.15 will instead look like illustrated in figure 3.22

When the complex region has been cut out of the mesh we end up with a set of edge loops and a set of holes in the source mesh. The edge loops connect each *type 1* vertex through a number of *type 2* on the boundary of the complex region. The next section describes how the holes are triangulated to fit the target mesh.

3.4.7. Re-meshing

In *Deforming meshes that split and merge* Wojtan et.al., uses marching cubes templates to re-mesh holes [18]. However, there is one major problem with marching cubes, in that it might produce holes in the output. Such a case is illustrated in figure 3.23. The problem is that the shared cell face has been treated differently when choosing the template for the two cells.

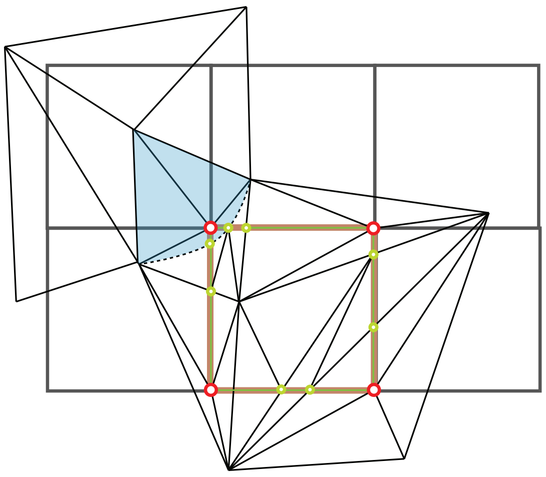


Figure 3.22.: Difficult case solved with grid based cutting.

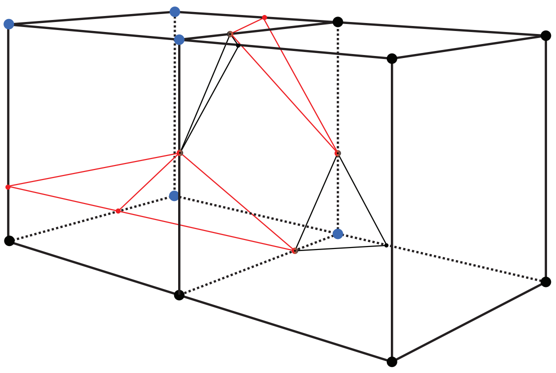


Figure 3.23.: Ambiguous cell face causing hole in marching cubes triangulation. Red lines are triangles in the left cell. Black lines are triangles in the right cell.

Instead of relying on marching cubes templates we exploit that we have stored the edge loops on the boundary of the complex region. Using that information we find the inner edge loops of each cell. After the cutting, a possible outcome could be as illustrated in figure 3.24, where the red dots are *type 1:s* and green dots are *type 2:s*.
What is left in the figure is to decide weather to create on or two edge loops. In fact it does not really matter in the sense of leaving holes so we just pick one, two in this case. The important step is to store which choice was made such the adjacent cell is triangulated correctly. When the next cell is reached all the red lines, in figure 3.24, are already known from the cutting and the red dotted lines are known from the previous cell. Once all inner edge loops have been found the

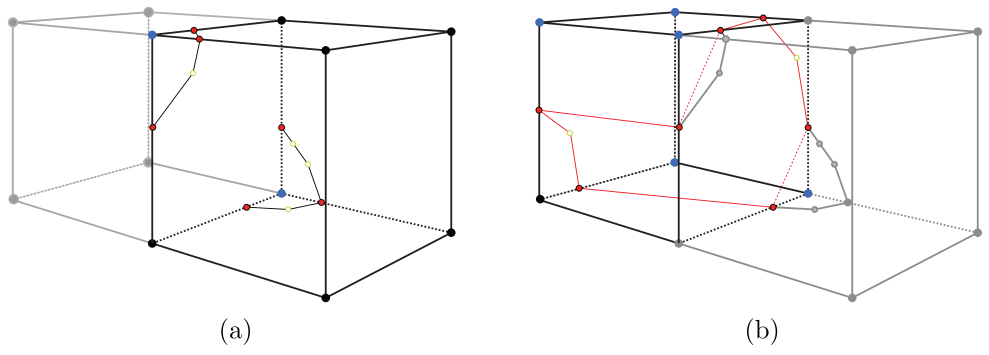


Figure 3.24.: Determining inner edge loops and handling ambiguous cell faces.

triangles are created. For each edge loop in each cell the vertices are sorted in some direction of the loop, see figure 3.25.

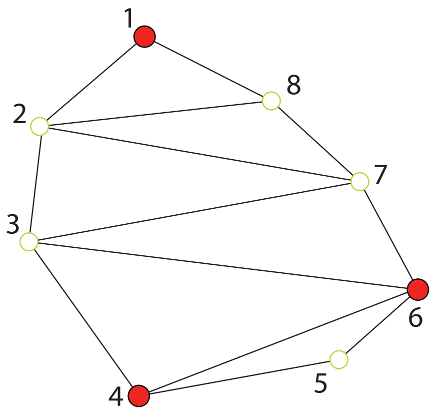


Figure 3.25.: Triangulation of inner edge loop.

A simple zigzag scheme is then used to triangulate the loop which is described below,

1. Initialize two counters *start*=0 and *end*=N-1, where N is the number of vertices in the sorted list, *l*, of vertices.
2. Create triangle from vertices $l[start{+}{+}]$, $l[start]$ and $l[end]$
3. If $start >= end$ - 1 $\rightarrow$ break.
4. Create triangle from vertices $l[end{-}{-}]$, $l[start]$ and $l[end]$
5. If $start < end$ - 1 $\rightarrow$ break. Otherwise repeat from (2).

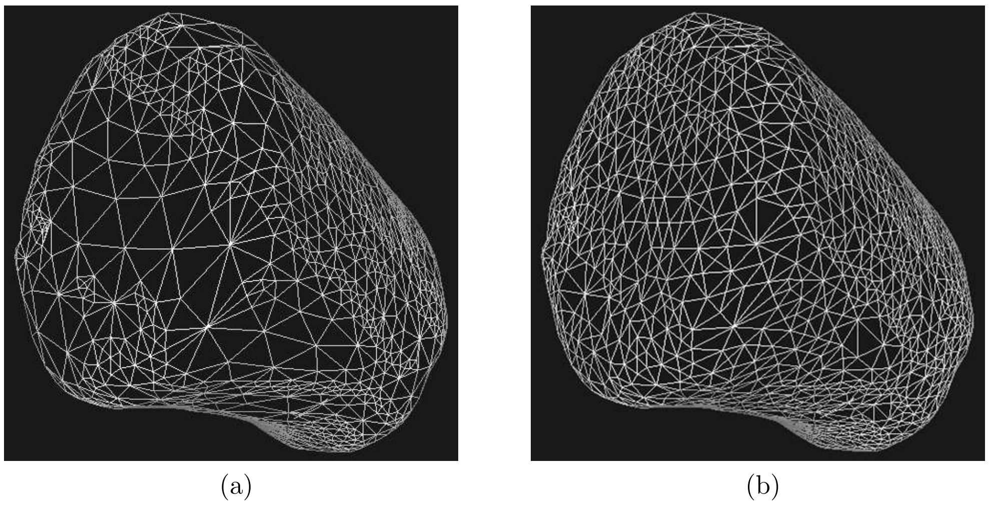
where $[\cdot] + +, [\cdot] - -$ are post- increment and decrement. Following the scheme produces the triangulation illustrated in 3.25. Note that this simple triangulation scheme can produce overlapping triangles if a point lies inside of the convex hull points. To get the correct windings for the triangles we always start to stitch new triangles at the boundary of the complex region. We can then use the winding information from the original triangles and propagate that into to the new triangles.

4. Results

In this chapter we show and discuss the results of using the method proposes in the previous chapter.

4.1. Mesh Improvement

Figure 4.1 illustrates a single frame of a sequence where mesh improvement has been applied. Image *a* shows the original mesh. Remember that the goal was to regularize the mesh. Image *b* shows the mesh after we have applied one *edge flip*, *edge collapse* and one *edge split* pass. We can see that most of the large triangles have been subdivided. However, there is still a band of quite small triangles in the upper middle part of the mesh.

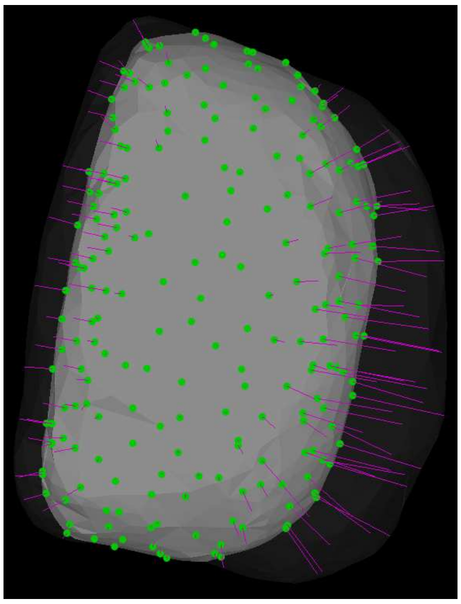


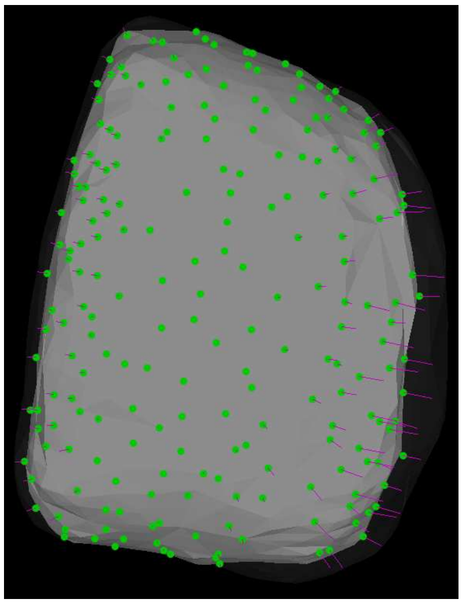
(a) (b)

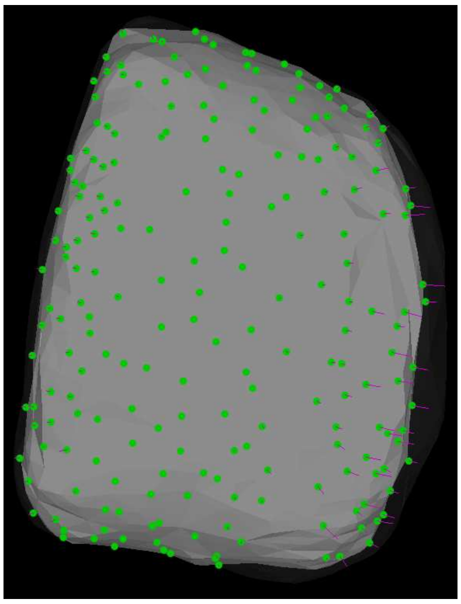
Figure 4.1.: Result of mesh improvement scheme. a: Original mesh b: Improved mesh after performing *edge flip*, *edge collapse* and *edge split*

4.2. Surface Registration

The set of images in figure 4.2 shows the result of the surface registration. Images a, b and c shows the coarse surface registration after 0, 15 iterations and convergence after 30 iterations in the case. The green dots are deformation graph nodes and the magenta lines depicts the correspondence vectors. The final image, d, shows the result after the detail registration step has been applied where the red dots signals that the distance between the deformation graph node and the target surface is zero. In this simple case the registration is as good as it gets. However there are still areas where we haven't been able to match the input. The cause is often that large triangles in the input has been matched to an area in th target that is more detailed. Therefore the density of correspondence vectors are limited and there is no possible way of finding a good match. Another cause could be the solver got stuck in a local minima. With a more complex target as was used in figure 4.3 the registration clearly struggles.


(a)


(b)


(c)

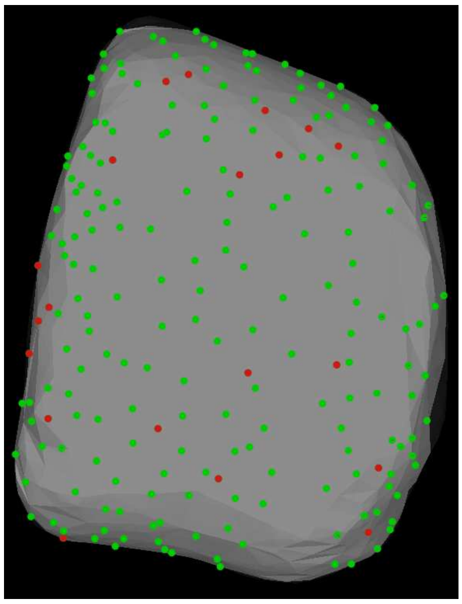

(d)

Figure 4.2.: Surface registration results. Green points are deformation graph nodes. Magenta lines are correspondence vectors. a) 0 iterations. b) 15 iterations. c) Non-rigid registration convergence. d) After detail alignment.

4.3. Grid Based Topology Changes

Figure 4.3 illustrates grid based topology changes. Image (a) the source mesh where the unmatched region have been cut out. The kept regions are marked blue and the boundary triangles are white. The transparent surface is the target surface. In this image we can clearly see the grid structure of the cutting border.

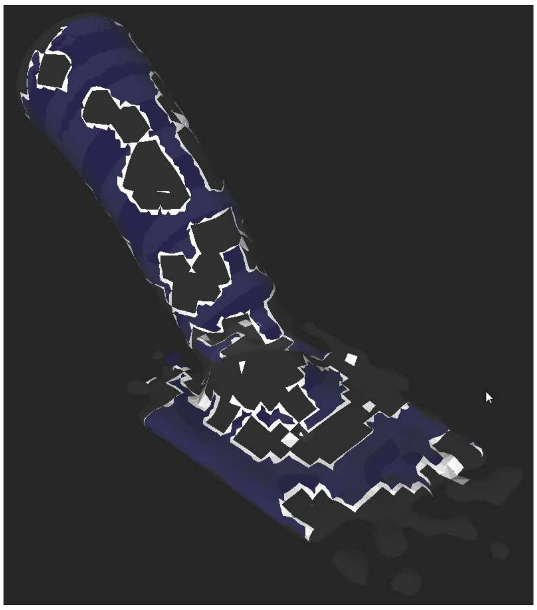
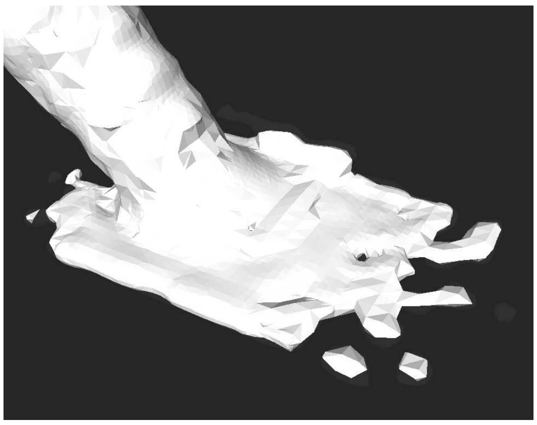
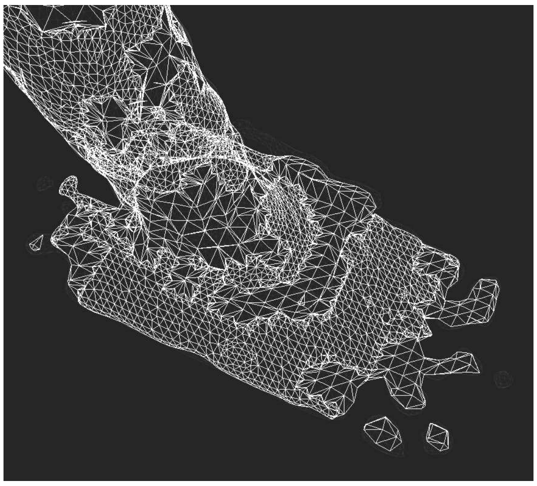


Figure 4.3.: Result after complex region has been cut out.

Figure 4.4 shows the result of triangulating the complex region that was cut out in 4.3. We show both a wireframe and a shaded rendering. Note that we have had to add many triangles on the boundary of complex region. These are quite difficult to get rid of with the mesh improvement scheme. We can also see that the resolution of the grid has a big impact on the final result in terms of the error between the triangulated surface and the target surface. We have even missed one of the blobs in the right part if the image.



(a)



(b)

Figure 4.4.: Triangulation of complex region. a) Shaded rendering. b) Wireframe rendering

5. Conclusion

Assessing the implementation and results throughout the thesis we can not see this method as a valid approach. There are several problems. One major reason is the use of a grid. The resolution of the grid has a great impact on the accuracy. Even if the grid resolution is increased there is always some loss of information compared to the explicit surface representation. The surface registration is also not accurate enough. Take fluid simulation as an example. High resolution fluid simulations are turbulent meaning that there are can be a high number of small deformations. The change between frames can be severe. The registration struggles a lot with this kind of input. Because adjacent correspondences are so different it is difficult to find good match. Also, since we compared rasterized version of the input and target even small fluctuation in the surface can cause a mismatch. The result of this is that in some parts of the mesh, every other cell does not between the input and the target, causing a great deal of re-meshing. Every time the mesh is cut we create poor quality triangles that are difficult to get rid of, at least with the mesh improvement scheme describes in this thesis. The final issue is cutting the mesh. Even though the grid based approach explained in this thesis is more robust compared to only using intersection tests it can not handle overlapping triangles.

The general idea of the method that has been described is very appealing. However, in practice as a robust production grade system there is much work to be done.

5.1. Future Work

Improving the surface registration would improve the result greatly. There are several improvements that could be made. We used approximated geodesic distances which causes the radius of influence for each graph node to be jagged instead of smooth. The deformation graph generation is supposed to be uniform. However, we sample the mesh by walking through mesh edges as described in section 3.3. This can in some cases cause the sampling to be sparse in some areas. This also touches on another improvement that could be made. If the input mesh is far away from being regularly triangulated some areas will have larger triangles even after the mesh improvement scheme is applied. If the corresponding target surface for that region is highly detailed it is difficult to get a good registration. One approach would be to completely re-triangulate the first frame such that at least the first frame is close to being regular.

As mentioned above, if the corresponding area of the mesh is more detailed then the source it is difficult to find a good registration since there will be to few points to deform. One idea is to detect such regions and add nodes to the coarse deformation graph. A similar idea would be to subdivide triangles in such region to gain more control in the detailed alignment. This would cause re-triangulation. However, it would be interesting to research the trade-off between doing local subdivision and gaining accurate registration compared to cutting and stitching new geometry. We believe that subdividing triangles could be preferred since it would be easier to keep high quality triangles. Another approach would be analyse to whole sequence to find how much resolution we need to represent the most detailed frame. Using this information we could subdivide the first frame with enough triangles that we would need.

A completely different approach that we would like to research is moving away from cutting and stitching and instead re-triangulate the target mesh using the triangulation of the previous frame. Imagine that the source an the target frame is a sphere, yet the triangulation is different. In this simple case we could choose a random position on the target surface and place a starting triangle from the list of triangles in the source mesh. Then insert the adjacent triangles using the connectivity information of the source mesh. The new vertices is placed using information from the target surface. If the source and target meshes have the same genus and are closed we should be able to produce a closed triangulation. However, if the source mesh is a sphere and the target is a donut the problem becomes tricky. The idea would be to use surface registration to get an approximate correspondences and potentially use a grid to detect topological changes. The areas with "good enough" correspondences would become candidates for starting points. If reaching a state were the triangulation can not be closed we would have to triangulate the holes. This would happen in cases such as merging and splitting. This idea is completely untested but would be interesting to research. The meshing would always use explicit information thus gaining accuracy compared to grid based meshing. There would not be a need for complicated cutting algorithms that creates poor triangles.

A. Rasterization Data Structure

Detailed description of the data structure used to store information that is needed for classifying complex cells, cutting and stitching new triangles.

```
struct CellCorner
{
  bool inside
}

struct CellEdge
{
  List meshFaces
  int  type1Vertex
}

struct CellFace
{
  List meshEdges
  List type2Vertices
}

struct Cell
{
  Vec3 ijk
  CellCorner* corners[8]
  CellEdge* edges[12]
  CellFace* faces[6]
}
```

Bibliography

[1] M. Bojsen-Hansen, H. Li, and C. Wojtan, "Tracking surfaces with evolving topology," *ACM Trans. Graph.*, vol. 31, pp. 53:1–53:10, July 2012.

[2] K. Mamou, T. Zaharia, F. Preteux, N. Stefanoski, and J. Ostermann, "Frame-based compression of animated meshes in mpeg-4," in *Multimedia and Expo, 2008 IEEE International Conference on*, pp. 1121 –1124, 23 2008-april 26 2008.

[3] Mohr and Gleicher, "Deformation sensitive decimation," in *University of Wisconsin Graphics Group Technical Report*, May 2003.

[4] C. Huang, Chen and Ouhyoung, "Animation model simplifications," in *National Taiwan University*, May 2005.

[5] E. Landreneau and S. Schaefer, "Simplification of articulated meshes.," *Comput. Graph. Forum*, vol. 28, no. 2, pp. 347–353, 2009.

[6] S. Zhang, J. Zhao, and B. Wang, "A local feature based simplification method for animated mesh sequence," in *Computer Engineering and Technology (ICCET), 2010 2nd International Conference on*, vol. 1, pp. V1–681 –V1–685, april 2010.

[7] M. Corsini, M.-C. Larabi, G. Lavou, O. Petk, L. Va, and K. Wang, "Perceptual metrics for static and dynamic triangle meshes," in *Eurographics, State of The Art Report*, May 2012.

[8] L. W. Tu, *An Introduction to Manifolds (Universitext).* Springer, second edition ed.

[9] M. Chang, Li and Pauly, "Geometric registration for deformable shapes," *Eurographics 2010 Tutorial.*

[10] H. Li, B. Adams, L. J. Guibas, and M. Pauly, "Robust single-view geometry and motion reconstruction," *ACM Trans. Graph.*, vol. 28, pp. 175:1–175:10, Dec. 2009.

[11] H. Li, L. Luo, D. Vlasic, P. Peers, J. Popović, M. Pauly, and S. Rusinkiewicz, "Temporally coherent completion of dynamic shapes," *ACM Trans. Graph.*, vol. 31, pp. 2:1–2:11, Feb. 2012.

[12] J. U. Brackbill and H. M. Ruppel, "FLIP - A method for adaptively zoned, particle-in-cell calculations of fluid flows in two dimensions," *Journal of Computational Physics*, vol. 65, pp. 314–343, Aug. 1986.

[13] Y. Zhu and R. Bridson, "Animating sand as a fluid," *ACM Trans. Graph.*, vol. 24, pp. 965–972, July 2005.

[14] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," *COMPUTER GRAPHICS*, vol. 21, no. 4, pp. 163–169, 1987.

[15] C. Wojtan, M. Müller-Fischer, and T. Brochu, "Liquid simulation with mesh-based surface tracking," in *ACM SIGGRAPH 2011 Courses*, SIGGRAPH '11, (New York, NY, USA), pp. 8:1–8:84, ACM, 2011.

[16] J. R. Shewchuk, "What is a good linear element? - interpolation, conditioning, and quality measures," in *In 11th International Meshing Roundtable*, pp. 115–126, 2002.

[17] X. Jiao, A. Colombi, X. Ni, and J. Hart, "Anisotropic mesh adaptation for evolving triangulated surfaces," *Eng. with Comput.*, vol. 26, pp. 363–376, Aug. 2010.

[18] C. Wojtan, N. Thürey, M. Gross, and G. Turk, "Deforming meshes that split and merge," *ACM Trans. Graph.*, vol. 28, pp. 76:1–76:10, July 2009.

[19] T. Brochu and R. Bridson, "Robust topological operations for dynamic explicit surfaces," *SIAM J. Sci. Comput.*, vol. 31, pp. 2472–2493, June 2009.

[20] M. Isenburg and P. Alliez, "Compressing polygon mesh geometry with parallelogram prediction.," in *IEEE Visualization*, pp. 141–146, 2002.

[21] R. W. Sumner, J. Schmid, and M. Pauly, "Embedded deformation for shape manipulation," *ACM Trans. Graph.*, vol. 26, July 2007.

[22] M. Teschner, B. Heidelberger, M. Mueller, D. Pomeranets, and M. Gross, "Optimized spatial hashing for collision detection of deformable objects," pp. 47–54, 2003.

[23] G. Varadhan, S. Krishnan, T. Sriram, and D. Manocha, "Topology preserving surface extraction using adaptive subdivision," in *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, SGP '04, (New York, NY, USA), pp. 235–244, ACM, 2004.