FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Processing Medical Surface Meshes for 3D Printing

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Anna Gostler
Matrikelnummer 1129406

an der Fakultät für Informatik
der Technischen Universität Wien

Betreuung: Ao. Univ-Prof. Dipl.-Ing. Dr. techn. Eduard Gröller
Mitwirkung: Ing. Dr. techn. Peter Mindek

Wien, 14. März 2016

Anna Gostler

Eduard Gröller

FAKULTÄT
FÜR !NFORMATIK

Faculty of Informatics

Processing Medical Surface Meshes for 3D Printing

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Media Informatics and Visual Computing

by

Anna Gostler
Registration Number 1129406

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao. Univ-Prof. Dipl.-Ing. Dr. techn. Eduard Gröller
Assistance: Ing. Dr. techn. Peter Mindek

Vienna, 14th March, 2016

Anna Gostler	Eduard Gröller

Erklärung zur Verfassung der Arbeit

Anna Gostler
Klostermanngasse 4, 1230 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 14. März 2016

Anna Gostler

Kurzfassung

Mit einem 3D-Drucker können physische, anatomische Modelle basierend auf medizinischen 3D-Volumsdaten (CT, MRI) angefertigt werden. Diese können für chirurgische Planung, Diagnose, die Herstellung von Implantaten und Bildung genutzt werden. Polygonnetze, die aus gescannten Daten extrahiert wurden, enthalten typischerweise Rauschen, sowie andere Arten von Artefakten. Diese Bachelorarbeit vergleicht verbreitete Denoising Algorithmen und beurteilt, wie gut diese für medizinische Polygonnetze geeignet sind – insbesondere in Hinblick auf einen anschließenden 3D-Druck. Darüber hinaus wird ein Ansatz vorgestellt um dünne, längliche Artefakte, die häufig in medizinischen Daten vorkommen, jedoch von Denoising Algorithmen nicht entfernt werden, zu erkennen und zu entfernen. Dadurch können sowohl Deformationen im geglätteten Polygonnetz als auch Stützstrukturen für den 3D-Druck reduziert werden.

Abstract

3D printed anatomical models obtained from medical volume data (CT, MRI) can be used for surgery-planning, diagnosis, fabrication of implants, and education. Surface meshes that are extracted from real-world data typically suffer from noise, as well as other types of artifacts. This thesis compares common denoising algorithms and assesses their applicability to medical surface meshes, particularly with regard to subsequent 3D printing. Additionally, this thesis proposes an approach to detect and remove frayed parts, an artifact commonly found in medical data that cannot be reduced through smoothing. By removing them, deformations in the smoothed mesh can be reduced and less support structures are needed for 3D printing.

Contents

Kurzfassung **vii**

Abstract **ix**

List of Figures **xii**

List of Algorithms **xiii**

1 Introduction **1**

2 Denoising Algorithms **3**
- 2.1 What are denoising algorithms? 3
- 2.2 Formal conventions 3
- 2.3 Data structure 3
- 2.4 Laplacian smoothing / Umbrella operator 4
- 2.5 Issues 5
- 2.6 HC-algorithm / Improved Laplacian Smoothing 6
- 2.7 Signal Processing Approach 7
- 2.8 Mean Curvature Flow 8
- 2.9 Scale-Dependent Umbrella 10
- 2.10 Comparison 11
- 2.11 Models 12
- 2.12 Metrics 12

3 Removing Frayed Parts **19**
- 3.1 Motivation 19
- 3.2 Related Work 20
- 3.3 Algorithm to Remove Frayed Parts 22
- 3.4 Results 27
- 3.5 Conclusion 29

Bibliography **31**

List of Figures

1.1 Pipeline from image acquisition to 3D print. The focus on this thesis lies on Denoising and Removing Frayed Parts from the extracted surface mesh. . . . 2

2.1 Data Structure 4
2.2 The Laplacian operator is sometimes called umbrella operator, because the central vertex surrounded by its neighbors resembles an umbrella. 5
2.3 Central vertex' position is first replaced by average of its neighbors (Laplacian operation), then pushed back. 6
2.4 Central vertex (marked red) surrounded by its neighboring vertices 9
2.5 Comparison of a) Laplacian Smoothing and b) Scale-Dependent Umbrella. When applying the Scale-Dependent Umbrella the neighbors' influence on the new position of $current_i$ does not depend on the edge lengths. 11
2.6 total change in relation to iterations 13
2.7 The distribution of approximately the same amount of total change indicated in false colors. (original mesh (a), Laplacian (b), Improved Laplacian (c), Signal Processing (d), Mean Curvature Flow (e), Scale-Dependent Umbrella (f)) 14
2.8 change of surface area in relation to total change 15
2.9 change of total mean curvature in relation to total change 16
2.10 change of volume in relation to total change 16
2.11 Comparison of shape preservation (sphere). (original mesh (a), Laplacian (b), Improved Laplacian (c), Signal Processing (d), Mean Curvature Flow (e), Scale-Dependent Umbrella (f)) 17
2.12 Comparison of shape preservation (vertebra). (reference model (ref), original model (a), Laplacian (b), Improved Laplacian (c), Signal Processing (d), Mean Curvature Flow (e), Scale-Dependent Umbrella (f)) 18

3.1 Calculating the exact distance between a vertex and a face 23
3.2 If the two vertices current_i and current_j do not lie on parallel faces (see right image), the angle between the vector connecting the two vertices (connection_{ij}) and the normal of current_i is less than 180° 23
3.3 Removing a spike: In each iteration only vertices which are labelled "artifact-vertices" (marked red) and which have "non-artifact" neighbors (marked green) are moved 24
3.4 original, noisy mesh (a), after smoothing with Mean Curvature Flow algorithm (b), and removal of frayed parts (c) 29
3.5 A 3D print was created after removal of frayed parts from and subsequent denoising (with Signal Processing algorithm) of the original, noisy mesh (a, d). Before (c) and after (b) removing support structures. 30

List of Algorithms

2.1 Laplacian smoothing . . . 5
2.2 HC-algorithm . . . 7
2.3 Signal Processing Approach . . . 8
2.4 Curvature Flow . . . 10
2.5 Scale-Dependent Umbrella . . . 11

3.1 Identify and Remove Frayed Parts . . . 24
3.2 detectSpikes(vertices, minimumDistance, toleranceOfDetection, sortedBy) . . . 25
3.3 vertexToTriangle(vertex, v1, v2, v3) . . . 26
3.4 growArtifactRegion(vertices) . . . 26
3.5 removeSpikes(vertices) . . . 27

CHAPTER 1

Introduction

3D printing medical image data offers great potential to medicine. 3D printed models of anatomical structures can be useful as tools for planning and rehearsing surgery, they can serve educational purposes, and can even be used directly on the human body in the form of implants. [1, 2]

Clinicians commonly rely on 3D images of anatomical data rendered on 2D displays for diagnosis and surgical planning. Compared to 2D images, 3D renderings give them a better understanding of their patient's body. A tangible 3D model can offer additional advantages. Surgeons operate using their hands, so a physical model enables them to practice their cuts just like they will perform them on the patient during surgery. [1] 3D printed models can also be used in anatomy classes to give students a chance to examine a wide range of accurate reproductions of diverse human anatomy. [3] Laypeople can get an intuitive understanding of pathologies from a physical model. [4]

The first step to printing a physical 3D model from anatomical data is usually image acquisition. A CT or MRI scanner produces a stack of gray-scale images of the human body, which can be combined into a volumetric representation of the body. The images are segmented to identify and delineate different parts of the body. The result of the segmentation is a binary mask, in which each voxel of the volume data is labeled as either part of a target structure or part of the background. Then, a surface mesh is extracted from the segmented volume data. When noise and artifacts have been removed, a slicer transforms the surface model into layers (and support structures if needed) and writes machine-readable code for the 3D printer. Then, the 3D printer, following these instructions, builds the physical model layer by layer. [5]

However, before the model is ready to print, noise and other artifacts have to be removed to reconstruct the originally smooth surface and reduce the need for support structures for 3D printing. The contribution of this thesis is to compare widely used mesh denoising algorithms and their performance on medical surface meshes, visually and quantitatively. We compare the following methods: Laplacian smoothing, improved

Laplacian smoothing [6], Taubin's signal processing approach [7], mean curvature flow [8] and the scale-dependent umbrella [8, 10].

Additionally, a novel algorithm is presented that detects and removes frayed parts, a type of artifact common in medical surface meshes, that cannot be reduced through smoothing. The algorithm consists of three steps: (1) identifying frayed parts by finding very thin parts of the mesh, (2) growing the identified regions to fully cover them, and (3) removing them from the mesh. Finally, we 3D printed a medical surface mesh from which noise and frayed parts were removed.

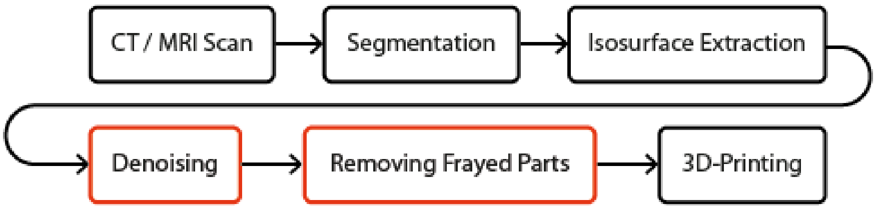


Figure 1.1: Pipeline from image acquisition to 3D print. The focus on this thesis lies on Denoising and Removing Frayed Parts from the extracted surface mesh.

CHAPTER 2

Denoising Algorithms

2.1 What are denoising algorithms?

Data obtained by scanning real-life objects typically contain noise, which is caused by imprecise measurements of the technical scanning device. Noise is distributed fairly regularly across the surface mesh. The goal of denoising algorithms is to remove noise, while preserving the inherent shape and characteristic features of the scanned object.

2.2 Formal conventions

A mesh consists of vertices. These vertices form triangles, called faces: each face consists of three vertices. A vertex is a point that has a position in $\mathbb{R}^3$. Vertices are connected to each other through edges: one edge connects two vertices. The 1-ring-neighborhood of a vertex consists of all vertices that are connected to that vertex by one edge. When using the term neighborhood we refer to the 1-ring-neighborhood of a vertex. All denoising algorithms presented below use the neighborhood of a vertex to calculate its new position. The vertex that is being modified, will be denoted as the central vertex. The position of a vertex in the original, noisy mesh is denoted as $original_i$. The position of a vertex that has not yet been modified by the current iteration of smoothing is called $current_i$. The modified position of that vertex after an iteration of smoothing is denoted as $smooth_i$.

2.3 Data structure

Only vertex positions and local neighborhood-information are needed for smoothing. Neighborhood relationships can be derived from information about which vertices are part of a face: all vertices that share a face with a given vertex are part of that vertex' neighborhood-set.

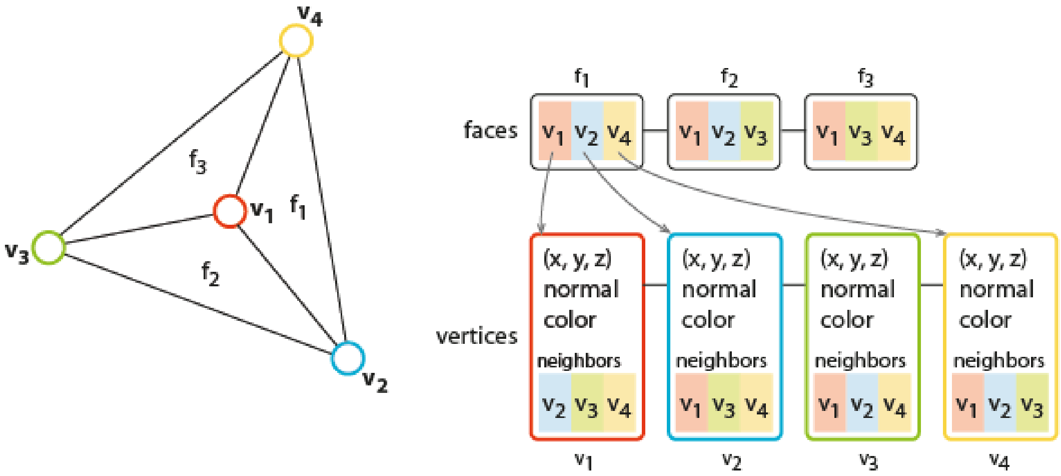


Figure 2.1: Data Structure

2.4 Laplacian smoothing / Umbrella operator

Laplacian smoothing is the most basic approach to smoothing. Its origin lies in image processing, where it is used to blur an image by replacing each pixel's color value with the average of its neighboring pixel's color values. This approach can be extended to mesh smoothing by averaging vertex positions instead of pixel colors.[6]

$$smooth_i := \frac{1}{neighbors(i).size} \sum_{j \in neighbors(i)} current_j \tag{2.1}$$

Laplacian smoothing strongly shrinks the mesh, and can be slowed down by using a weighing factor λ on the displacement vector (i.e. the vector that moves the central vertex towards its new position).

$$smooth_i := current_i + \frac{\lambda}{neighbors(i).size} \sum_{j \in neighbors(i)} (current_j - current_i) \tag{2.2}$$

Algorithm 2.1: Laplacian smoothing

```
smooth ⟵ original;
while not smooth enough do
    current ⟵ smooth;
    for $i \in vertices$ do
        n ⟵ neighbor(i).size;
        if $n > 0$ then
            $\text{smooth}_i \longleftarrow \frac{1}{n} \sum_{j \in neighbor(i)} \text{current}_j$;
        end
    end
end
```

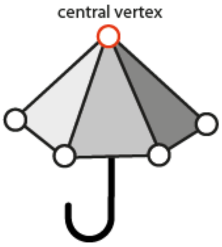


Figure 2.2: The Laplacian operator is sometimes called umbrella operator, because the central vertex surrounded by its neighbors resembles an umbrella.

2.5 Issues

Denoising algorithms deal with three main issues [9] when applied to 3-dimensional meshes:
1) Irregularity: In a 2-dimensional image pixels are positioned at regular distances. In a mesh the number of neighbors, the distances between neighboring vertices, and the angles between edges can vary. If the influence of the neighbors is not weighed according to these factors, deformations occur.
2) Shrinkage: Smoothing algorithms shrink the mesh if no counter-measures are taken. The shrinking becomes more pronounced with the number of smoothing iterations.
3) Vertex-Drifting: If vertices are moved in other directions than along their normal, the result is tangential drifting. Features do not stay in place, vertices move even if they are on flat surfaces. Vertex-drifting is especially strong for meshes with different sampling rates.

2.6 HC-algorithm / Improved Laplacian Smoothing

The HC-algorithm [6] improves the Laplacian algorithm by reducing shrinking and deformation. The basic idea is to push back the smoothed vertices toward their previous and / or original positions. Obviously, the vertices cannot be pushed back to their exact same, old positions – that would just result in an unchanged mesh. Instead, each modified vertex is pushed back, not only by its own difference vector but by an average of its own and its neighbors difference vectors.

$$diff_i := smooth_i - (\alpha * original_i + (1-\alpha) * current_i) \quad \alpha \in [0,1] \tag{2.3}$$

$$pushBack_i := \beta * diff_i + \frac{1-\beta}{neighbors(i).size} \sum_{j \in neighbors(i)} diff_j \quad \beta \in [0,1] \tag{2.4}$$

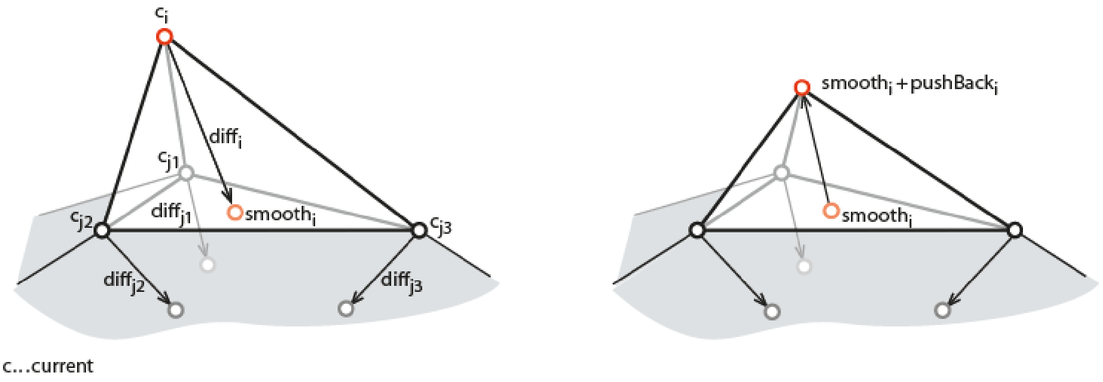


Figure 2.3: Central vertex' position is first replaced by average of its neighbors (Laplacian operation), then pushed back.

The influence of the central vertex can be adjusted by a parameter, and must be included. Including the original vertices can reduce shrinkage and to avoid the possibility of the mesh collapsing to one point, but it also preserves the noise of the original mesh.

Each iteration of the HC-algorithm consists of two steps. In the first step all vertices are smoothed using the Laplacian operation, and the differences between the old and new position of each vertex are stored. In the second step each vertex is pushed back by an average of its own and its neighbors' distances, calculated in the first step. Notably, the second step also uses the idea of smoothing by averaging the local neighborhood – but this time using differences.

The algorithm can be performed sequentially (immediately updating vertex positions, so that some new vertex positions will be calculated using neighbors, that have been modified already) or simultaneously (calculating all new vertex positions using only previous vertex positions). While the sequential version saves storage space, the simultaneous version yields better results. Since storage space was not an issue, the simultaneous version was implemented. The HC-algorithm yields the same degree of smoothing as the Laplacian algorithm, while preserving shape and size far better. The HC-algorithm is very fast, using only a local neighborhood for calculations. Also, calculations require only simple vector arithmetic, which makes it easy to implement.

Algorithm 2.2: HC-algorithm

```
smooth ⟵ original;
while not smooth enough do
    current ⟵ smooth;
    for $i \in$ vertices do
        n ⟵ neighbors(i).size;
        if $n > 0$ then
            $\text{smooth}_i \longleftarrow \frac{1}{n} \sum_{j \in neighbors(i)} \text{current}_j$;
        end
        $\text{diff}_i \longleftarrow \text{smooth}_i$ - ($\alpha$ * $\text{original}_i$ + (1-$\alpha$) * $\text{current}_i$);
    end
    for $i \in$ vertices do
        n ⟵ neighbors(i).size;
        if $n > 0$ then
            $\text{smooth}_i \longleftarrow \text{smooth}_i$ - ($\beta$ * $\text{diff}_i$ + $\frac{1-\beta}{n} \sum_{j \in neighbors(i)} \text{diff}_j$);
        end
    end
end
```

2.7 Signal Processing Approach

The basic idea of Taubin's algorithm [7] is similar to the HC-algorithm: Vertices are first smoothed and then pushed back to correct the shrinking, that was caused by smoothing. Unlike the HC-algorithm, shrinking can be eliminated completely, because this approach allows to fine-tune the shrinking and un-shrinking steps so that they balance each other out.

Taubin interpreted the positions of the vertices on a surface mesh as signals. He approached the problem of denoising by extending signal processing to signals defined on polyhedral surfaces. From this point of view the underlying data (the inherent shape of the mesh) is represented by the low frequencies, while the high frequency-components should be discarded as noise.

The usual approach to decompose a signal into its frequency components is the Fourier-analysis. But Taubin found that even performing the Fast-Fourier-transform on meshes with a very large number of vertices – which are typical for medical data – was not computationally feasible. He made the observation, however, that a mesh can be projected onto the space of low frequencies only approximately, using low-pass filters, which can be computed in linear time. He introduced a non-shrinking smoothing algorithm that consists of two scaled Laplacian smoothing steps: A shrinking step that is followed by a non-shrinking step. The shrinking step uses a positive scale factor λ and the un-shrinking step a negative scale factor μ. The parameters λ and μ should be chosen in a way that the shrinking and the non-shrinking steps balance each other out.

Algorithm 2.3: Signal Processing Approach

```
$\lambda \longleftarrow 0.63$;                                        ▷ exemplary values for $\lambda$ and $\mu$
$\mu \longleftarrow -0.67$;
current ⟵ original;
while not smooth enough do
                                                                     ▷ shrinking step
    for $i \in vertices$ do
        n ⟵ neighbors(i).size;
        if $n > 0$ then
            $\text{diff}_i \longleftarrow \frac{1}{n} \sum_{j \in neighbors(i)} (current_j - current_i)$;
            $\text{smooth}_i \longleftarrow \text{current}_i + \lambda * \text{diff}_i$;
        end
    end
                                                                     ▷ un-shrinking step
    for $i \in vertices$ do
        n ⟵ neighbors(i).size;
        if $n > 0$ then
            $\text{diff}_i \longleftarrow \frac{1}{n} \sum_{j \in neighbors(i)} (smooth_j - smooth_i)$;
            $\text{current}_i \longleftarrow \text{smooth}_i + \mu * \text{diff}_i$;
        end
    end
end
```

2.8 Mean Curvature Flow

Desbrun et al. [8] use curvature flow to approximate the Laplacian in a discrete setting. High curvature is associated with high frequency and thus noise. Additionally, when using curvature flow, vertices are only moved along the normal direction – there is no movement in the tangential direction, as with the other presented smoothing methods. This prevents vertex drifting, so characteristic features stay in place. The distance a vertex is moved depends on its local mean curvature: the higher the curvature the further the movement. Curvature is independent from sampling rate, so areas with the same curvature but different sampling rates will have the same curvature after smoothing. Curvature flow will also keep flat areas intact, because the curvature of a flat surface is zero.

The discrete curvature normal of a vertex is given as

$$-\overline{\kappa}\ n = \frac{1}{4A} \sum_{j \in neighbors(i)} (cot\alpha_j + cot\beta_j)(current_j - current_i) \tag{2.5}$$

where A is the smallest region around this vertex: the triangles of the neighborhood.

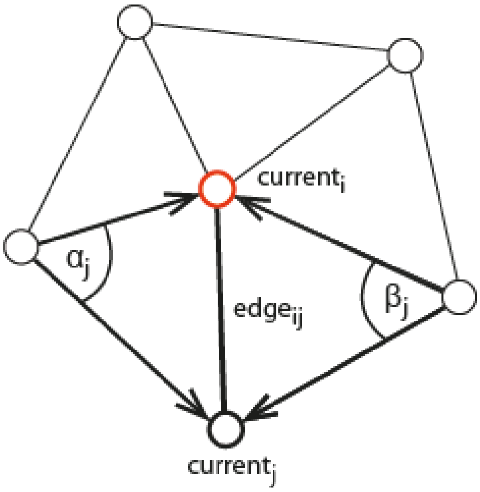


Figure 2.4: Central vertex (marked red) surrounded by its neighboring vertices

Algorithm 2.4: Curvature Flow

```
smooth ⟵ original;
while not smooth enough do
    current ⟵ smooth;
    for $i \in vertices$ do
        sumDiff ⟵ 0;
        n ⟵ neighbors(i).size;
        if $n > 0$ then
            for $j \in neighbors(i)$ do
                        ▷ find vertices opposite the edge $\overrightarrow{current_i\ current_j}$
                $opposite \longleftarrow intersection(neighbors(j), neighbors(i))$;
                        ▷ find the cotangent of the angles $\alpha_j$ and $\beta_j$ (see Figure 2.4)
                sumCotangent ⟵ 0;
                for $o \in oppositeVertices$ do
                    cosinus ⟵ $\overrightarrow{o\ current_i} \cdot \overrightarrow{o\ current_j}$;
                    sinus ⟵ $\overrightarrow{o\ current_i} \times \overrightarrow{o\ current_j}$;
                    sumCotangent ⟵ sumCotangent + $\frac{cosinus}{sinus}$;
                end
                sumDiff ⟵ sumDiff + (current$_j$ - current$_i$) * sumCotangent;
            end
            smooth$_i$ = sumDiff $\frac{1}{4\ area_i}$;
        end
    end
end
```

2.9 Scale-Dependent Umbrella

The Scale-Dependent Umbrella [8, 10] is based on the Laplacian operator, but the influence of each neighboring vertex is based on to the length of the edge that connects it to the central vertex.

One important difference between a 2D-image and a 3D-mesh is that in an image all the pixels are positioned at regular distances. The distance between a pixel and each of its neighboring pixels is the same, and distances are also regular across the entire image. In a typical 3D-mesh this is not the case; distances between vertices are usually irregular. So, if all neighboring vertices are weighed equally, vertices farther from the central vertex will have a stronger influence on its new position. This can be especially problematic if sampling rates differ across the mesh, leading to strong deformations: regions with larger distances between vertices will be smoothed more quickly than regions, where vertices are connected by shorter edges. The Scale-Dependent Umbrella reduces tangential vertex drifting: a more distant neighboring vertex will not pull the central vertex stronger towards itself than a closer neighboring vertex. However, in contrast to the Mean Curvature Flow algorithm, tangential movement of vertices is still possible, so tangential vertex drifting cannot be fully eliminated.

$$L(current_i) = \frac{2}{E} \sum_{j \in neighbors(i)} \frac{current_j - current_i}{|edge_{ij}|} \tag{2.6}$$

$$E = \sum_{j \in neighbors(i)} |edge_{ij}|$$

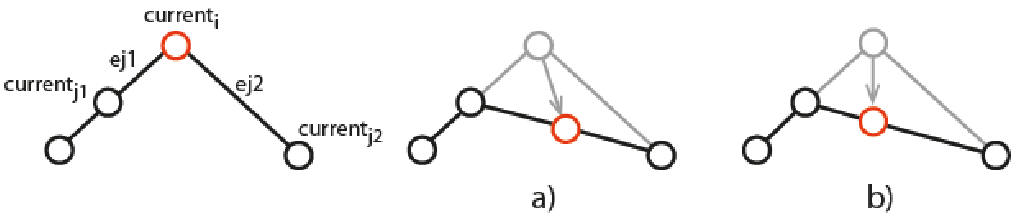


Figure 2.5: Comparison of a) Laplacian Smoothing and b) Scale-Dependent Umbrella. When applying the Scale-Dependent Umbrella the neighbors' influence on the new position of *current_i* does not depend on the edge lengths.

Algorithm 2.5: Scale-Dependent Umbrella

```
smooth ⟵ original;
while not smooth enough do
    current ⟵ smooth;
    for $i \in vertices$ do
        n ⟵ neighbors(i).size;
        if $n > 0$ then
            for $j \in neighbors(i)$ do
                sumDiff ⟵ sumDiff + (current$_j$ - current$_i$) * $\frac{1}{|edge_{ij}|}$;
                sumWeight ⟵ sumWeight + $|edge_{ij}|$;
            end
            smooth$_i$ ⟵ current$_i$ + $\frac{sumDiff}{sumWeight}$;
        end
    end
end
```

2.10 Comparison

We have described five smoothing algorithms, which are widely used for denoising surface meshes. Now we will evaluate which of these algorithms is best suited for medical data.

Medical surface meshes differ from technical models, in that organic shapes do not contain sharp, straight edges. When a model is acquired by a scanning device in an everyday clinical setting, there is no perfect reference model available– only the original model, which contains both noise and artifacts from different sources. Common artifacts are missing or additional object parts, holes, frayed parts, stairs and plateaus. [11]

2.11 Models

The performance of the denoising algorithms will be tested on three different models:

A computer-generated **sphere** partially corrupted by artificial, uniformly distributed noise. The sphere consists of four regions: one of coarse resolution with noise, one of coarse resolution without noise, one of fine resolution with noise and one of fine resolution without noise. The original, smooth model will be used as reference.

A **vertebra** (C7) extracted from a CT scan obtained from the Laboratory of Human Anatomy and Embryology, University of Brussels (ULB), Belgium. To enhance the step artifacts typical of a CT scan only a third of the images in the original dataset were used to extract the surface model. A surface model extracted from the complete dataset will be used as a reference.

A **spine and pelvis** that contain light noise and frayed parts mostly on the vertebrae. The only reference model available is the original, noisy surface model.

2.12 Metrics

The impact of the denoising algorithms on the models will be evaluated quantitatively and through visual inspection. Surface area, curvature and visual inspection are suitable for assessing visual improvement; the distance between smoothed mesh and original mesh, as well as volume are appropriate measures for the deviation from the original model. [11]

The different algorithms smooth the mesh at different speeds. To enable a meaningful comparison between them, we do not compare the smoothed meshes after the same number of iterations, but at the same total change in vertex positions. We used appropriate parameters to slow down some of the algorithms to achieve this.

The changes in surface area, curvature and volume are shown in line graphs. To give an idea when sufficient smoothing was reached and how this amount of smoothing affects the mesh, the point, where the model looked “best”, subjectively, when smoothed with each algorithm is marked with a star.

False color images will show how the effects of the different smoothing algorithms are distributed across the surface meshes.

To assess shape deformations caused by the application of the denoising algorithms on the model of the vertebra and sphere, the silhouette of the reference model was traced over images of the smoothed surface meshes, shown at their “best” look, as determined by visual inspection.

2.12.1 Total Change

The different denoising algorithms smooth the surface at different speeds. To enable a better comparison of the smoothed meshes, they are being compared at the same level of change, instead of at the same number of iterations.

Change will be defined as the Euclidian distance between the position of a vertex in the original mesh and its new position in the smoothed mesh, and total change as the sum of these distances.

$$change_i = \sqrt{(original_i - smooth_i)^2} \tag{2.7}$$

$$total\ change = \sum_{i \in vertices} change_i$$

The following graph (Figure 2.6) shows how the number of iterations of each denoising algorithm relates to total change. For all algorithms more iterations lead to an increase in total change.

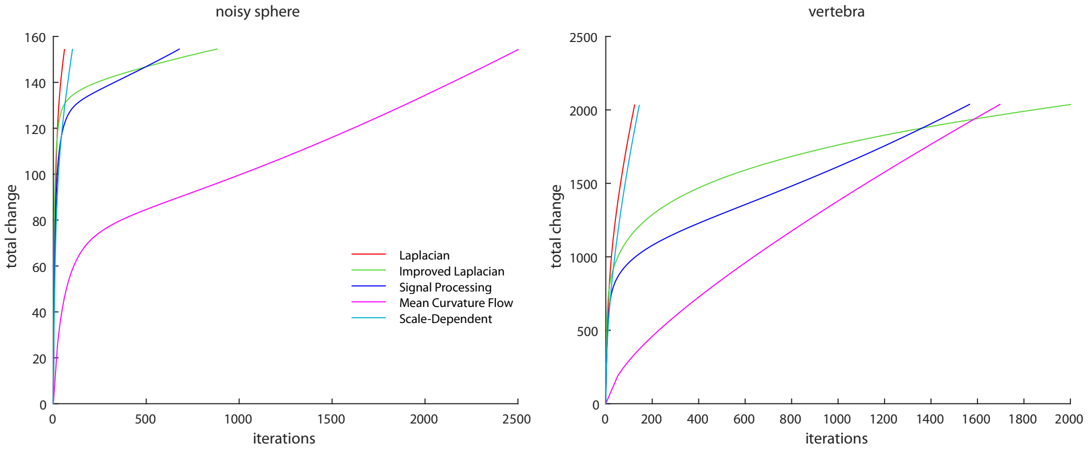
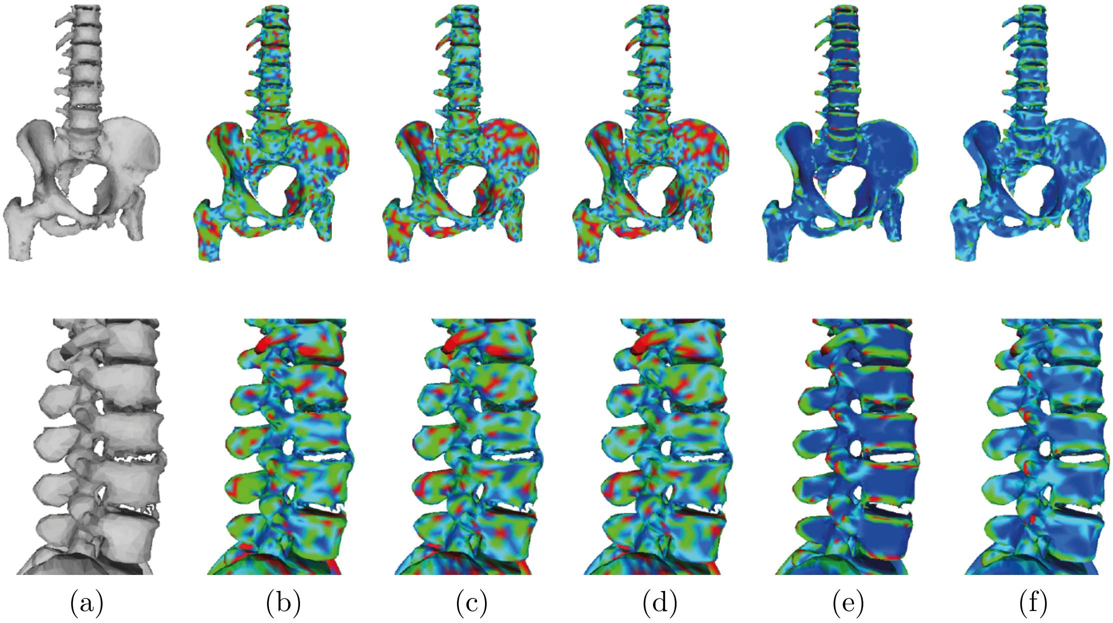


Figure 2.6: total change in relation to iterations

Figure 2.7 shows a false color visualization of the model of the pelvis and spine after the same amount of total change was applied by each denoising algorithm. With the Mean Curvature Flow algorithm change is much more focused on certain regions of the mesh: more change occurs in noisy regions and on edges, while flatter regions were moved less. The same pattern can be seen with the Scale-Dependent umbrella algorithm but less pronounced. The Laplacian, Improved Laplacian and Signal Processing algorithm distribute change more evenly across the entire mesh.



(a) (b) (c) (d) (e) (f)

Figure 2.7: The distribution of approximately the same amount of total change indicated in false colors. (original mesh (a), Laplacian (b), Improved Laplacian (c), Signal Processing (d), Mean Curvature Flow (e), Scale-Dependent Umbrella (f))

2.12.2 Surface Area

Surface area is an indicator of smoothness: the less wrinkled a surface is, the smaller its area. But the surface area of an object is dependent on its volume: if an object is scaled to a greater size its surface area will increase. In the process of denoising, the volume can change. To make the comparison of surface area scale-independent, we will measure the deviation of the surface area of a smoothed mesh from the minimal surface area, given the same volume – which is the surface area of a sphere with the same volume. The graph (Figure 2.8) shows how much the model's actual surface area deviates from that of a perfectly round sphere at a given amount of total change. Using the Mean Curvature Flow algorithm surface area decreases the fastest and the meshes reach their "best" look (marked with a star) with the least total change. With the Signal Processing algorithm the surface area does no longer decrease beyond a certain point – which is close to the point where the volume starts to increase (best seen in the right side of the graphs(vertebra)).

$$surfaceArea_{sphere} = 4\pi * \left(\frac{volume_{current}}{\frac{4}{3}\pi}\right)^{\frac{2}{3}} \tag{2.8}$$

$$deviation_{current} = ((surfaceArea_{current}/surfaceArea_{sphere}) - 1) * 100$$

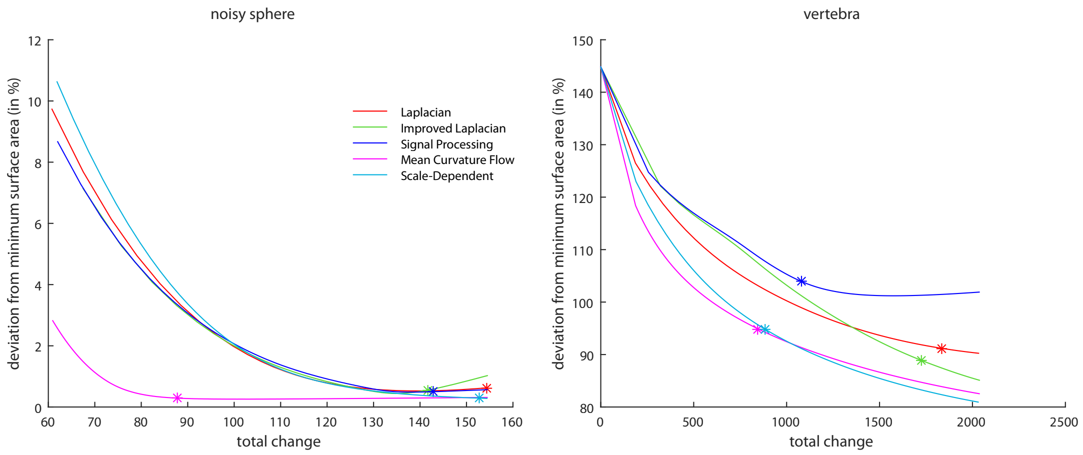


Figure 2.8: change of surface area in relation to total change

2.12.3 Mean Curvature

Mean curvature locally describes the curvature of a surface and can be approximated as the length of the mean curvature normal $\overline{\kappa}\, n = \frac{1}{4A}\sum_{j\in neighbors(i)} cot\alpha_j + cot\beta_j(current_j - current_i)$. The Mean Curvature Flow algorithm minimizes mean curvature and we can see in the graph that this algorithm decreases mean curvature the fastest (Figure 2.9). Second fastest is the Scale-Dependent Umbrella algorithm. It is noticeable that the other algorithms – Laplacian, Improved Laplacian and Signal Processing algorithm – affect mean curvature very unsteadily, but do decrease it eventually.

2.12.4 Volume

Volume is relevant for medical diagnosis as it may indicate whether a tissue is healthy. Denoising does not shrink the object uniformly: a coarser resolution leads to faster shrinking when Laplacian and Improved Laplacian are used (see Figure 2.11). The Mean Curvature Flow and the Scale-Dependent Umbrella algorithm shrink high-curvature parts of the surface shrink faster than other parts. The Mean Curvature Flow keeps overall volume constant by rescaling the mesh after smoothing. The Laplacian, Improved Laplacian and Scale-Dependent Umbrella algorithm shrink the mesh with increased total

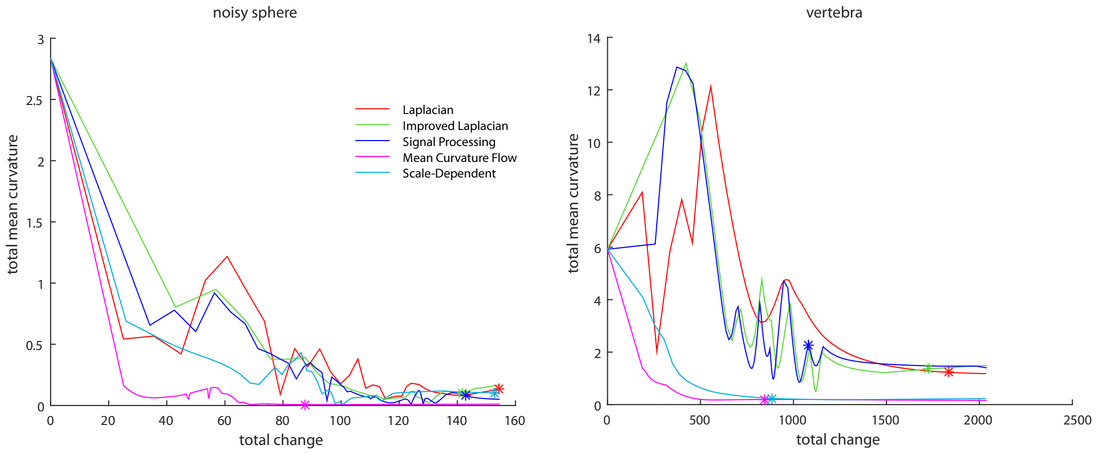


Figure 2.9: change of total mean curvature in relation to total change

change, while the Signal Processing algorithm eventually increases the volume. While the Laplacian and less strongly the Scale-Dependent Umbrella start to decrease the volume already with low total change, the Improved Laplacian and the Signal Processing algorithm keep the volume constant up to a certain point. (Figure 2.10)

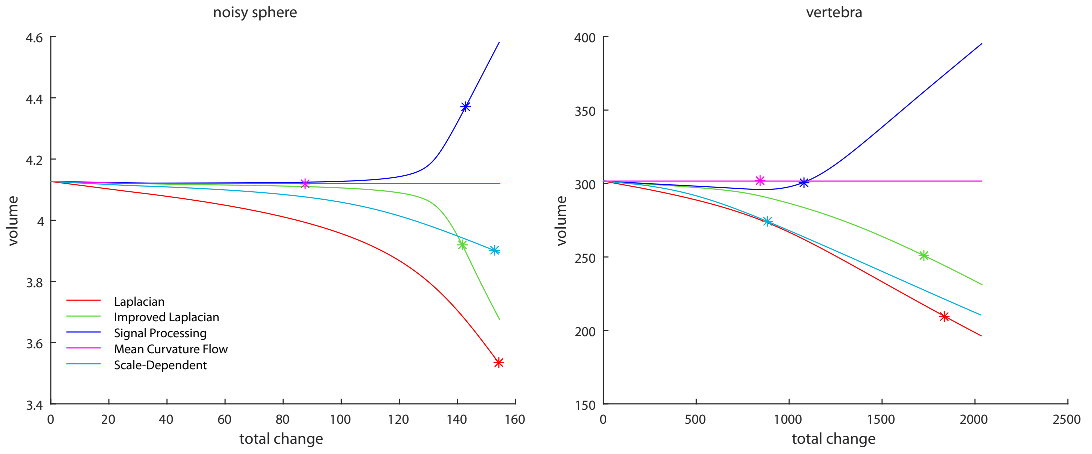


Figure 2.10: change of volume in relation to total change

2.12.5 Shape preservation

Shape preservation is evaluated through visual inspection (see Figures 2.11, 2.12). To facilitate comparison of the results, the silhouette of the original shape was traced over the smoothed meshes (shown at the "best" level of smoothing). A strong shrinking at the curved tip of the vertebra's process is noticeable. We decided to determine the "best", smooth look of the vertebra with regard to removal of step-artifacts: we chose the point at which no more steps were visible. A property of the Mean Curvature Flow algorithm is, that preserves flat areas well (if a vertex lies in the same plane as its neighbors, it has mean curvature 0 and is not moved). As a consequence, steps only disappear after significant smoothing with the Mean Curvature Flow algorithm.

When using Laplacian smoothing the sphere shows a strong shrinking of the right part (coarser resolution). This effect is less pronounced, but still noticeable, when using Improved Laplacian smoothing. The Signal-Processing algorithm leads to a slight growing of the coarser part of the sphere. The Mean Curvature Flow and Scale-Dependent Umbrella algorithm restore the round shape best. The Mean Curvature Flow algorithm restores the original volume through rescaling, and leads to the best result on the noisy sphere overall (see Figure 2.11).

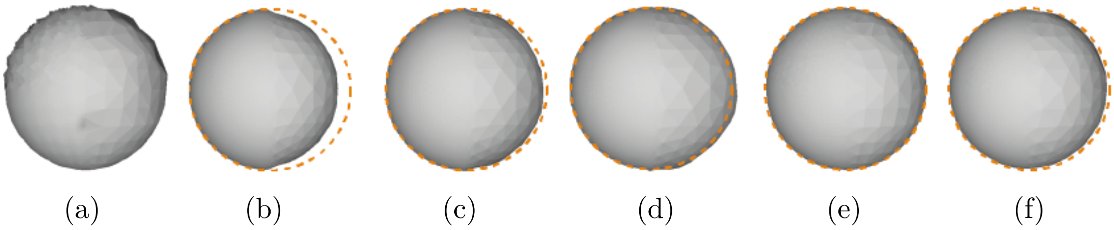

(a) (b) (c) (d) (e) (f)

Figure 2.11: Comparison of shape preservation (sphere). (original mesh (a), Laplacian (b), Improved Laplacian (c), Signal Processing (d), Mean Curvature Flow (e), Scale-Dependent Umbrella (f))

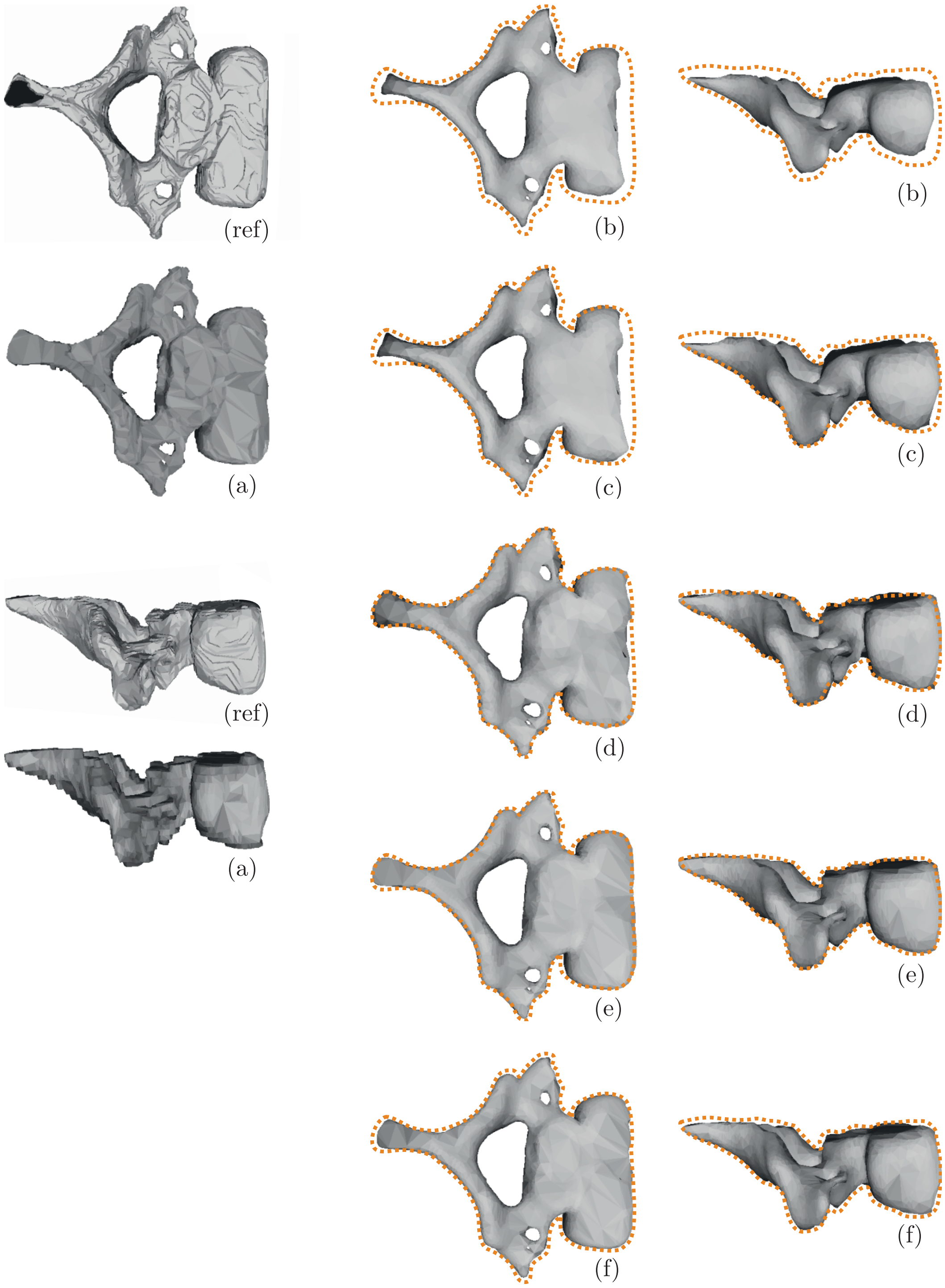


Figure 2.12: Comparison of shape preservation (vertebra). (reference model (ref), original model (a), Laplacian (b), Improved Laplacian (c), Signal Processing (d), Mean Curvature Flow (e), Scale-Dependent Umbrella (f))

CHAPTER 3

Removing Frayed Parts

3.1 Motivation

In 3D printing thin features can be difficult to print. Fine details might not be printed at all, if they are thinner than the layers the 3D printer can print. Thin, long features sticking out of the surface can require additional support structures to keep them from collapsing. After the print is finished these support structures need to be removed manually with specific tools, causing extra work and leaving traces on the finished model. Usually, removed support structures will then be discarded as waste. [5]

Thin parts do not only pose problems in the printing process but are also a common type of artifact in medical surface meshes, which cannot be eliminated through denoising [11], because the do not follow the typical pattern of noise. Noise is distributed (roughly) uniformly across the surface: some incorrectly placed vertices occur above the “correct” surface and some below. The denoising algorithms presented above smooth the surface by basically finding an average surface. The vertices in frayed parts, on the other hand, are positioned above the surface. As a result, denoising algorithms will make this type of artifact only thinner but will not remove it.

We want to reduce these problems by detecting and removing frayed parts before smoothing. Users can choose a minimum diameter according to what they identify as artifacts, as opposed to desired features. Frayed parts are marked red in the rendered 3D-model, so that users can clearly distinguish them. Detected frayed parts can then be removed automatically.

3.2 Related Work

Technical scanning devices add physical noise to the image data. As a result, the measurements of the points deviate from the positions, that they have on the real surface. Thus, smoothing the mesh leads to a more realistic representation of the surface. However, excessive smoothing decreases accuracy, as it causes deformations, change in volume and a loss of detail. Accuracy is an important issue in medicine, as an anomaly, e.g. in volume, might indicate a pathology. Surgical planning and drilling guides require especially high accuracy. [16]

Smoothing is a trade-off between achieving a smooth surface and preserving important features. [21] However, feature-sensitive algorithms, which are often applied to engineering data [22] preserve edges and are thus unsuitable for organic shapes. The only edges typically present in medical data are staircases, which should be removed instead of preserved. Because of that, advanced smoothing techniques specifically for medical data were developed.

Advanced smoothing algorithms adapt smoothing locally, instead of smoothing the entire mesh equally. [18] Context-aware smoothing limits smoothing to certain areas, by taking into account local properties. Moench et al. [17] present an algorithm to remove staircase artifacts from the surface mesh. First, the vertices of the mesh are divided into artifact (part of a staircase) and non-artifact vertices. Then, vertices in artifact-regions are smoothed more strongly, while vertices in non-artifact regions are (optionally) smoothed with less weight.

Spatial relationships also play an important part in medicine. However, smoothing changes distances between vertices. Distance-aware smoothing [21] preserves small distances between neighboring structures during smoothing – achieving a smooth, and therefore more natural look, while keeping critical distances accurate.

Smoothing is also not suitable to remove artifacts, that change the topology of the mesh, because smoothing will not change the topology. Guskov et al. [13] developed an algorithm that identifies and removes handles from the surface mesh, letting the user control the size of handles that should be removed. They found that the loss of accuracy through smoothing is reduced when the artifacts are removed first.

Medical surface meshes often contain artifacts, that are the result of an incorrect segmentation (over- or under-segmentation): frayed parts, holes, additional or detached parts. These artifacts can generally not be reduced through smoothing. [23]

Physicians generally prefer automatic solutions due to time constraints in everyday clinical settings. But a reliable, automatic segmentation is often prevented by the large anatomical variety in medical image data. Segmentation of medical images identifies and delineates different regions of the body. Basic segmentation methods use thresholds, seed points and region growing, or border detection. These methods face difficulties, when the images contain noise, if regions are not homogenous, or values of neighboring structures are not sufficiently distinct. More advanced techniques work based on models, but they can fail when applied to diverse types of anatomy or pathologies. [19]

Thus, manual correction of an automatically generated segmentation is often necessary.

However, a fully manual segmentation can take hours. [4] Heckel et al. [12] developed a tool that lets users edit the automatically generated segmentation of a tumor by sketching on the slices. The segmentation is then recomputed based on the user input.

Moench et al. [20] recommend manual removal of segmentation artifacts from the surface mesh using 3D computer graphics software. This is time-consuming, however, and requires experience with such software tools. Additionally, medical experts are needed to validate changes made to the mesh, distinguishing between artifacts and anatomical anomalies.

3D printing allows the creation of highly accurate replicas of individual patient anatomy [14] – even on a relatively low budget. Bortolotto et al. [15] found, that a state-of-the-art 3D printer for consumer use can produce 3D objects from CT scans, which are sufficiently accurate, even for demanding medical applications. Accuracy is influenced much stronger by the quality of segmentation than by technical factors. [16]

In anatomy classes 3D printed models can be used in place of real human bones or anatomical models. This allows students to directly examine a diverse range of highly accurate copies of human tissue, that might otherwise be too fragile, rare or expensive to distribute among students. [3]

3D printed models of patient's pathologies have been successfully used for surgical planning. A tactile model can help surgeons to better understand their patient's condition and to practice before the actual operation. This can reduce operation time and complication rates for the patients. [14]

The haptic component and actual three-dimensional shape of a 3D printed model makes it better suited for the brain's perceptive mechanisms – like perception of depth and relative size – than a virtual 3D rendering on a flat 2D monitor. This allows laypeople an intuitive understanding of a pathology, which can be useful in courtrooms and education. [4]

3D printing adds specific requirements to mesh processing. While it is not always necessary to remove certain artifacts for visualization, they can cause problems in the process of 3D printing. [20] For a successful 3D print the mesh must be manifold: this means – among others – that the surface mesh must be watertight (i.e. have no holes) and each triangle edge must be shared by exactly two triangles. [5] A 3D printer builds an object by placing layer upon layer. Each layer supports the layer placed on top of it. Overhanging parts – by approximately 45° – need to be supported by support structures, otherwise these parts might collapse. Some 3D printing technologies do not require support material, but FDM (Fused Deposition Modeling), which is the most commonly used 3D printing technology [15] relies on support material. Support material must be removed manually after printing, which is tedious – especially in hard-to-reach areas – and produces waste. Thus, support material should be avoided as much as possible – e.g. by eliminating noise and other artifacts, such as frayed parts, which protrude from the surface. [5]

3.3 Algorithm to Remove Frayed Parts

We want to find and remove frayed parts – a type of artifact which cannot be reduced through smoothing. [11] We will do this by labeling each vertex as either an artifact-vertex, if it it is part of a frayed part of the mesh, or a non-artifact-vertex, if it does not lie in a frayed part. Then, only artifact-vertices will be removed, other regions of the mesh are left unchanged. Non-artifact-vertices can be smoothed before or after removing the frayed parts.

To find frayed parts in the mesh, we need to define what a frayed part is. A frayed part is a region of the mesh, where parts of a neighboring structure were included in the mesh because of over-segmentation. [11] It consists of thin spikes, sticking out of the surface mesh. To find these thin spikes, we will identify “very thin” parts of the mesh. The user can give a threshold value, to control what Euclidian distance between vertices should constitute a “very thin” part.

Finding thin parts

A very thin part of a mesh, is a part where vertices come close together on opposite sides of the mesh. However, they might lie on two sides of the mesh facing towards each other. Additionally, vertices are usually close to each other if they are neighbors. To distinguish between these three cases, we will use normals. Normals are defined for each triangle of the mesh and determine the direction that it faces. To approximate the normal of a vertex, we can use the average of the normals of all triangles, that this vertex is a part of.

First, we need to find vertices that are close in distance. To limit the number of comparisons necessary, we sort the list of vertices along x-, y- and z-coordinate and then compare each vertex only to vertices with a x-, y- and z-coordinate within the distance given by the user. Since vertices are positioned at discrete distances, the distance between two vertices is often larger than the distance between a vertex and a face (see Figure: 3.1). We can get a more reliable distance by calculating the distance between the current vertex ($current_i$) and the faces connected to the vertices($current_{j1}, current_{j2}$) that the current vertex is close to. The minimum distance between a triangle and a point, is the distance between the point and the intersection point on the triangle, which is found by moving the point along the normal and intersecting it with the triangle. The exact distance is calculated, when the distance between the two vertices is slightly larger than the given distance.

When we found two vertices ($current_i, current_j$) that are close to each other, we determine the angle between the normal of $current_i$ and the “connection vector” to $current_j$. The “connection vector” is the normalized vector pointing from $current_i$ to the vertex $current_j$. An angle of 90° indicates two neighbors, with normals perpendicular to the plane they lie in; an angle of 180° indicates two vertices on opposing, parallel sides of the mesh. If the two vertices lie on two sides of the mesh, that face each other, the angle would be 0°.

However, two sides of a spike are rarely perfectly parallel (see Figure: 3.2) . So, we have to allow angles less than 180°. What angle should still be counted as indicating

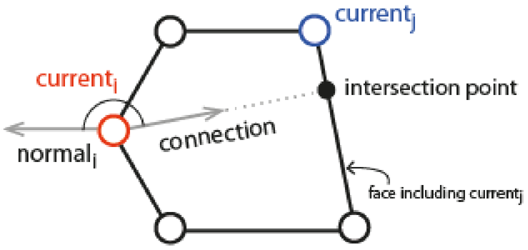


Figure 3.1: Calculating the exact distance between a vertex and a face

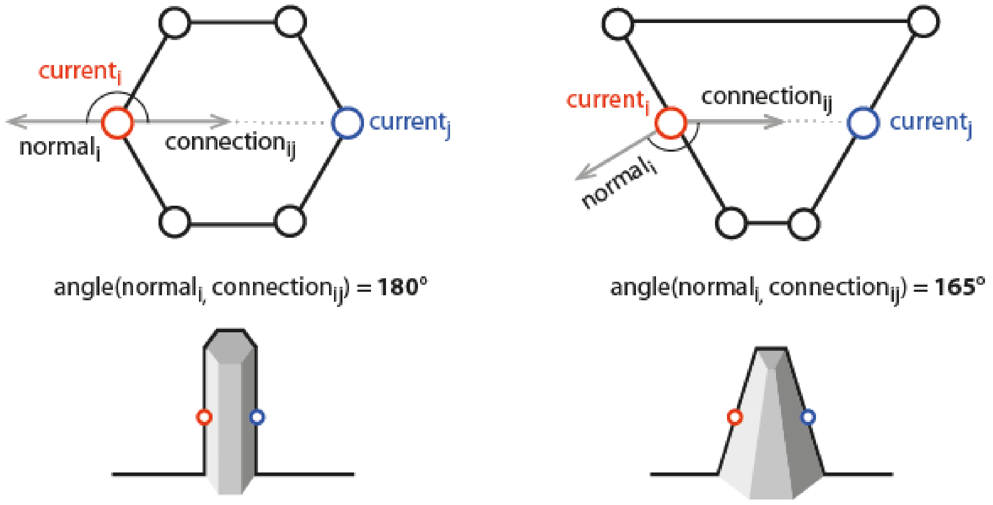


Figure 3.2: If the two vertices current$_i$ and current$_j$ do not lie on parallel faces (see right image), the angle between the vector connecting the two vertices (connection$_{ij}$) and the normal of current$_i$ is less than 180°

vertices on opposite sides of the mesh can be controlled by the user with the parameter "tolerance of detection". "Tolerance of detection" can range from 0 (angle must be 180°) to 1 (angle from 90° to 180° is allowed). This parameter has to be used carefully to maintain the distinction between vertices on different sides of the mesh and neighbors on the same side of the mesh.

Expanding artifact regions

This approach will not reach the tip of a spike, if there is no close vertex opposing the tip. Because of this, it is often necessary to grow the artifact region to cover the entire spike. This is done by additionally labeling vertices as artifact-vertices, that have more than a given amount of neighbors that are artifact vertices. Taking 30% gave good results.

Removing frayed parts

Artifact-vertices are removed from the mesh by moving them to the average position of their non-artifact neighbors. This can take a few iterations, if there are artifact-vertices, which do not have any non-artifact neighbors (usually artifacts further up a spike). In each iteration all artifact-vertices with non-artifact-neighbors, are moved to their new position and are labeled as non-artifact-vertices at the end of the iteration. This is repeated until all vertices have been labeled non-artifact-vertices. This way, the frayed parts are "pulled down" onto the correct surface. Removing the frayed parts will leave no holes in the mesh, and vertices are only moved within their neighborhood, so that no changes in topology can occur.

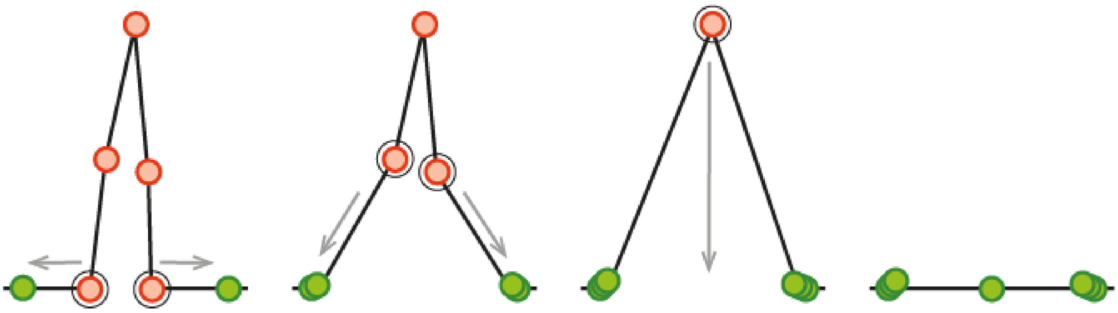


Figure 3.3: Removing a spike: In each iteration only vertices which are labelled "artifact-vertices" (marked red) and which have "non-artifact" neighbors (marked green) are moved

Algorithm 3.1: Identify and Remove Frayed Parts

```
sortX(vertices);
detectSpikes(vertices, minimumDistance, toleranceOfDetection, x);
sortY(vertices);
detectSpikes(vertices, minimumDistance, toleranceOfDetection, y);
sortZ(vertices);
detectSpikes(vertices, minimumDistance, toleranceOfDetection, z);
growArtifactRegion(vertices);
removeSpikes(vertices);
```

Algorithm 3.2: detectSpikes(vertices, minimumDistance, toleranceOfDetection, sortedBy)

```
for $i \in vertices$ do
    j ⟵ i;
    while $|current_j.sortedBy - current_i.sortedBy| < minimumDistance$ do
        j ⟵ j + 1;
        distance ⟵ distance(current$_i$, current$_j$);
        if $distance \leq minimumDistance$ then
            connection ⟵ current$_j$ - current$_i$;
            connection ⟵ connection/connection.length;
                                ▷ toleranceOfDetection between 0 (180°) and 1 (90°)
            if $(cos(connection, current_i.normal) < toleranceOfDetection - 1)$ then
                colorRed(current$_i$);
                colorRed(current$_j$);
            end
        else
            if $distance \leq minimumDistance + \varepsilon$ then
                for $face \in$ faces including $current_j$ do
                    $connection \longleftarrow vertexToTriangle(current_i, face.vertex1, face.vertex2, face.vertex3)$;

                    if $connection \neq NULL$ and $connection.length \leq minimumDistance$ then
                        connection ⟵ connection / connection.length;
                        if $(cos(connection, current_i.normal) < toleranceOfDetection - 1)$ then
                            colorRed(current$_i$);
                            colorRed(current$_j$);
                        end
                    end
                end
            end
        end
    end
end
```

Algorithm 3.3: vertexToTriangle(vertex, v1, v2, v3)

```
iPoint.normal ⟵ (v1.normal + v2.normal + v3.normal)/3;
iPoint.normal ⟵ iPoint.normal / iPoint.normal.length;
t ⟵ iPoint.normal · v1 - iPoint.normal · vertex;
iPoint ⟵ vertex + t · iPoint.normal;
        ▷ check if iPoint is inside triangle(v1, v2, v3) using barycentric coordinates
$\alpha$ ⟵ triangleArea(v1, v2, iPoint) / triangleArea(v1, v2, v3);
$\beta$ ⟵ triangleArea(v2, v3, iPoint) / triangleArea(v1, v2, v3);
$\gamma$ ⟵ triangleArea(v3, v1, iPoint) / triangleArea(v1, v2, v3);
if $(\alpha + \beta + \gamma) == 1$ then
    return (iPoint - vertex);
else
    return NULL;
end
```

Algorithm 3.4: growArtifactRegion(vertices)

```
while $artifact\ region\ not\ fully\ covered$ do
    for $i \in vertices$ do
        incorrectNeighbors ⟵ 0;
        n ⟵ neighbors(i).size;
        if $n > 0$ then
            for $j \in neighbors(i)$ do
                if $current_j.correct == false$ then
                    incorrectNeighbors ⟵ incorrectNeighbors + 1;
                end
            end
            if $incorrectNeighbors\ /\ n > 0.3$ then
                colorRed(current$_i$);
            end
        end
    end
end
```

Algorithm 3.5: removeSpikes(vertices)

```
removing ⟵ true;
while removing do
    removing ⟵ false;
    for $i \in vertices$ do
        if $current_i.correct == false$ or $current_i$ has artifact-neighbors then
            removing ⟵ true;
            correctNeighbors ⟵ 0;
            for $n \in neighbors(i)$ do
                if $current_n.correct == true$ then
                    sum ⟵ sum + $current_n$;
                    correctNeighbors ⟵ correctNeighbors + 1;
                end
            end
            if $correctNeighbors > 1$ then
                $smooth_i$ ⟵ sum / correctNeighbors;
                $smooth_i$.correct ⟵ true;
            else
                $smooth_i$ ⟵ $current_i$;
            end
        end
    end
    current ⟵ smooth;
end
```

3.4 Results

We used different criteria to compare the effects of common mesh denoising algorithms on surface meshes: total change, surface area, mean curvature, volume and shape preservation.

We determined the “best” look for each algorithm to see how much change to the mesh was necessary for each algorithm to reach its “best” look. However, when comparing the individual “best” looks (Figure 2.12) we can see that Mean Curvature Flow, Scale-Dependent Umbrella and Signal Processing algorithm produce much better results than Laplacian and Improved Laplacian smoothing. Overall, the Mean Curvature Flow algorithm produced the best results. When applied to the model of the sphere it clearly reached its “best” look with causing the least total change to the mesh, decreased surface area the fastest and did not change the volume of the mesh. On the model of the vertebra the Mean Curvature Flow algorithm reached its “best” look only slightly before the Scale-Dependent Umbrella and Signal Processing algorithm. The staircases in the model of the vertebra increased the amount of smoothing necessary for achieving sufficient visual improvement for all algorithms – but especially the Mean Curavature Flow algorithm required more smoothing to remove staircases, because it tends to preserve flat parts of the mesh. Thus, the results could likely be improved by removing staircases from the mesh [17] before smoothing. The worst results were produced by simple Laplacian smoothing: it caused the most loss in volume and shape deformation and is therefore not suitable to preserve accuracy. The smoothing algorithms improved the meshes by reducing noise. However, they are not suitable for reducing frayed artifacts. We implemented the algorithm to remove frayed parts described above and applied it to the model of a spine and pelvis, which contains frayed artifacts mostly on the vertebrae (remaining parts of the intervertebral discs). The mesh was first denoised using the Mean Curvature Flow algorithm, which could not remove the frayed parts (see Figure 3.4). Then we applied our algorithm with 0.7 for "tolerance of detection" ($\sim 134°$) and could detect and remove the majority of frayed parts. Some parts of the mesh were incorrectly identified as frayed parts, though.

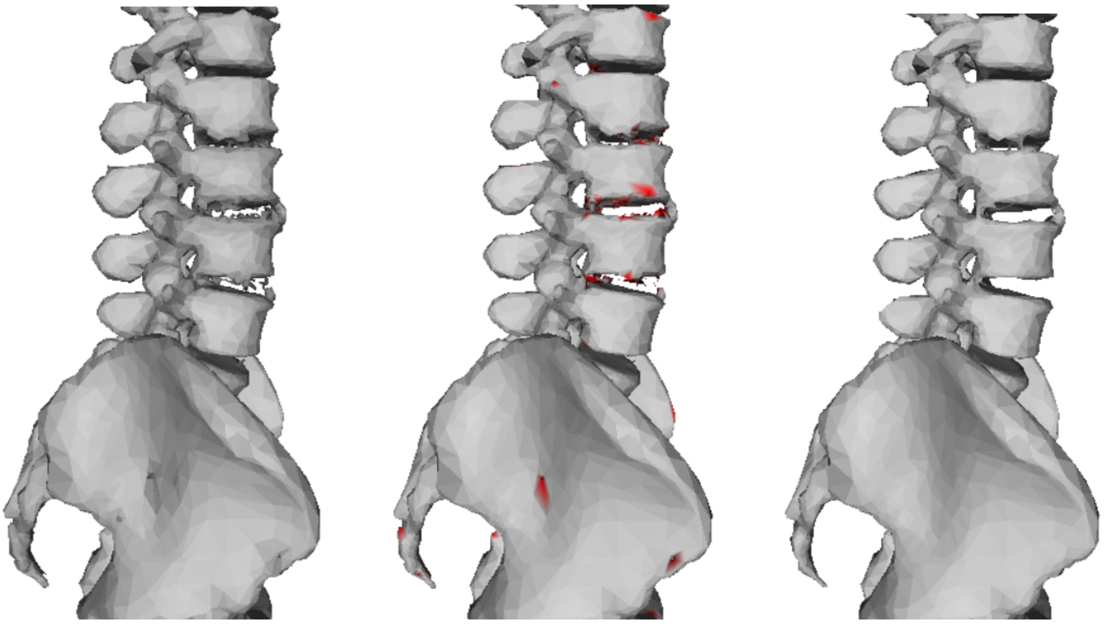

(a) (b) (c)

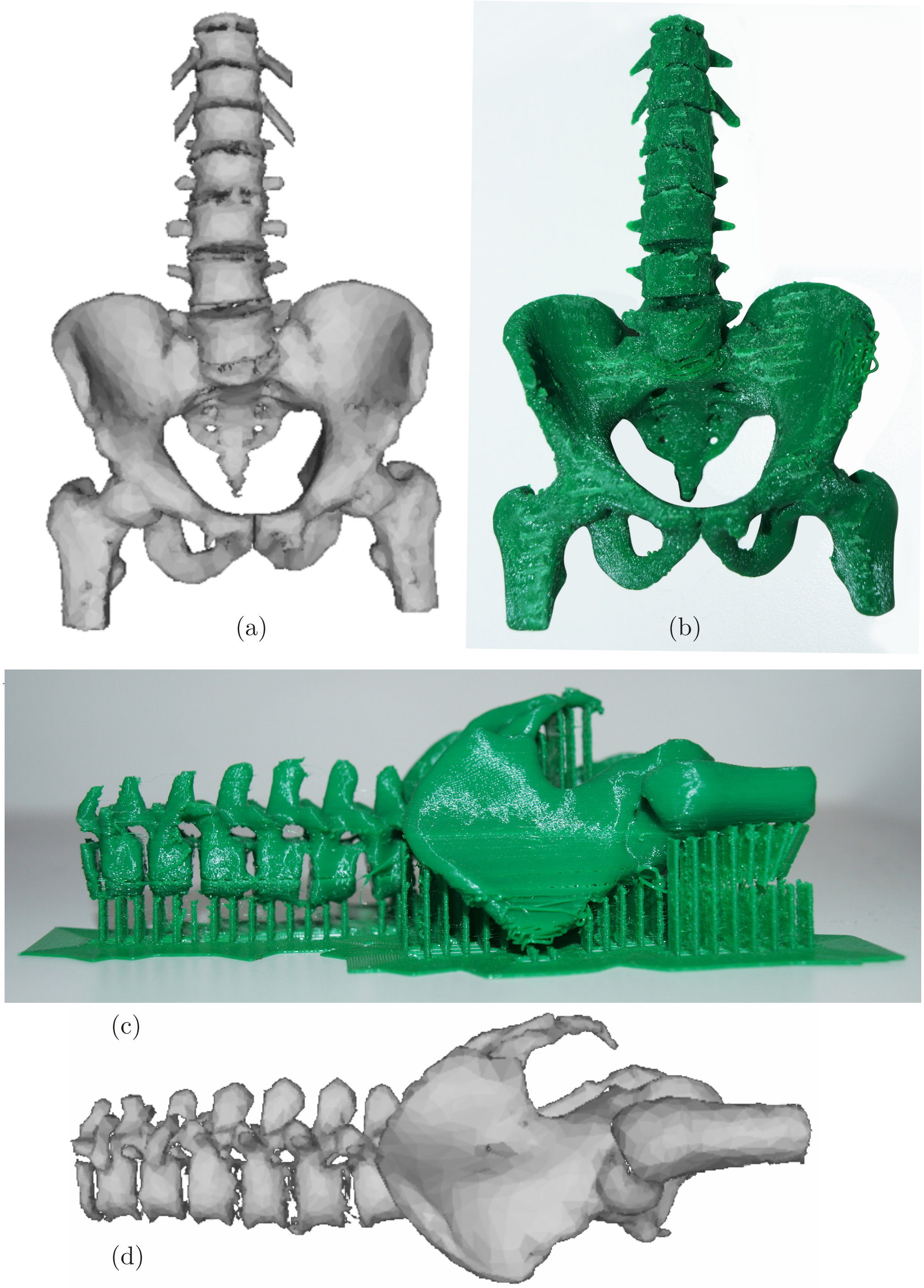
Figure 3.4: original, noisy mesh (a), after smoothing with Mean Curvature Flow algorithm (b), and removal of frayed parts (c)

3.5 Conclusion

In this thesis, we quantitatively and visually compared several mesh smoothing methods, which are commonly used to remove noise from medical surface meshes. It turned out that artifacts other than noise – like staircases and frayed parts – inhibited the effects of smoothing, but that a high amount of smoothing decreased the accuracy of the surface mesh.

We presented a semi-automatic method to detect and remove frayed parts from the surface mesh, which are artifacts that are not reducible through smoothing. Our approach is faster than manual correction with 3D computer graphics software. When the model is 3D printed, our method helps to reduce the need for support structures.

Future work should improve the algorithm to target frayed parts more reliably. It could be useful to add an option that lets the user select which of the parts, that were identified as frayed parts, should be removed and which should be kept. In order to increase accuracy, a tool could be developed, that lets the user compare the recommended corrections with the original segmented slides.



(a) (b) (c) (d)

Figure 3.5: A 3D print was created after removal of frayed parts from and subsequent denoising (with Signal Processing algorithm) of the original, noisy mesh (a, d). Before (c) and after (b) removing support structures.

Bibliography

[1] John Brennan *Production of Anatomical Models from CT Scan Data* Master's Dissertation. De Montfort University, Leicester, United Kingdom, 2010

[2] F. Rengier, A. Mehndiratta, H. von Tengg-Kobligk, C. M. Zechmann, R. Unterhinninghofen, H.-U. Kauczor, F. L. Giesel *3D printing based on imaging data: Review of medical applications* 2010: International Journal of Computer Assisted Radiology and Surgery, Volume 5, Issue 4, pp. 335-341

[3] Yousef AbouHashem, Manisha Dayal, Stephane Savanah, Goran ętrkalj *The application of 3D printing in anatomy education* 2015: Medical Education Online 2015, Vol. 20

[4] Lars Chr. Ebert, Michael J. Thali, Steffen Ross *Getting in touch – 3D printing in Forensic Imaging* 2011: Forensic Science International, Volume 211, Issues 1-3, pp. e1-e6

[5] Joan Horvath *Mastering 3D Printing* 2014: Apress: Berkeley, CA

[6] J. Vollmer, R. Mencl, H. Müller *Improved Laplacian Smoothing of Noisy Surface Meshes* 1999: EUROGRAPHICS '99 Vol. 18

[7] Taubin, Gabriel *A Signal Processing Approach To Fair Surface Design* 1995: SIGGRAPH '95 Proceedings of the 22nd annual conference on Computer graphics and interactive techniques, pp. 351-358

[8] Mathieu Desbrun, Mark Meyer, Peter Schröder, Alan H. Barr *Implicit fairing of irregular meshes using diffusion and curvature flow* 1999: SIGGRAPH '99 Proceedings of the 26th annual conference on Computer graphics and interactive techniques, pp. 317-324

[9] Shachar Fleishman, Iddo Drori, Daniel Cohen-Or *Bilateral Mesh Denoising* 2003: ACM Transactions On Graphics, Vol.22(3), pp. 950-953

[10] Koji Fujiwara *Eigenvalues of Laplacians on a closed riemannian manifold and its nets*1995: In Proceedings of AMS 123, pages 2585-2594

[11] Ragnar Bade; Jens Haase; Bernhard Preim *Comparison of Fundamental Mesh Smoothing Algorithms for Medical Surface Models* 2006: SimVis, pp. 289-304

[12] Frank Heckel, Jan H. Moltz, Christian Tietjen, Horst K. Hahn *Sketch-based Image-independent Editing of 3D Tumor Segmentations using Variational Interpolation*, 2012: Proceedings of Eurographics Workshop on Visual Computing for Biology and Medicine, pp. 73-80

[13] Igor Guskov, Zoë J. Wood *Topological Noise Removal* 2001 Graphics Interface Proceedings: Ottawa, Canada, pp: 19-26

[14] Tamir Friedman, Mark Michalski, T. Rob Goodman, J. Elliott Brown *3D printing from diagnostic images: a radiologist's primer with an emphasis on musculoskeletal imaging-putting the 3D printing of pathology into the hands of every physician* 2016: Skeletal Radiology, Volume 45(3), pp. 307-321

[15] Chandra Bortolotto, Esmeralda Eshja, Caterina Peroni, Matteo Orlandi, Nicola Bizzotto, Paolo Poggi *3D Printing of CT Dataset: Validation of an Open Source and Consumer-Available Workflow* 2016: Journal of Digital Imaging, Volume 29(1) pp.14-21

[16] Zacharias Fourie, Janalt Damstra, Rutger H. Schepers, Peter O. Gerrits, Yijin Ren *Segmentation process significantly influences the accuracy of 3D surface models derived from cone beam computed tomography* 2012: European Journal of Radiology, Volume 81, Issue 4, e524 - e530

[17] Tobias Moench, Rocco Gasteiger, Gabor Janiga, Holger Theisel, Bernhard Preim *Context-Aware Mesh Smoothing for Biomedical Applications* 2011: Computers & Graphics Volume 35, Issue 4, pp. 755-767

[18] Bernhard Preim, Charl Botha *Surface Rendering* In 2014: Visual Computing for Medicine (Second Edition) – Theory, Algorithms, and Applications, Morgan Kaufmann, pp. 229-267

[19] Bernhard Preim, Charl Botha *Image Analysis for Medical Visualization* In 2014: Visual Computing for Medicine (Second Edition) – Theory, Algorithms, and Applications, Morgan Kaufmann, pp. pp. 111-175

[20] Tobias Moench, Mathias Neugebauer, Bernhard Preim *Optimization of Vascular Surface Models for Computational Fluid Dynamics and Rapid Prototyping.* 2011: Second International Workshop on Digital Engineering pp. 16-23

[21] Tobias Moench, Simon Adler, Peter Hahn, Ivo Roessling, Bernhard Preim *Distance-aware smoothing of surface meshes for surgical planning* 2010: Proceedings of the First International Workshop on Digital Engineering, pp. 45-51

[22] Leif Kobbelt, Mario Botsch, *Feature Sensitive Mesh Processing* 2003: Computer Graphics: Proceedings of the 19th Spring Conference (SCCG '03), pp. 17-22

[23] Ragnar Bade, Jens Haase, Bernhard Preim *Comparison of Fundamental Mesh Smoothing Algorithms for Medical Surface Models* 2006: Simulation und Visualisierung, pp. 289-304