Die approbierte Originalversion dieser Diplom-/Masterarbeit ist an der Hauptbibliothek der Technischen Universität Wien aufgestellt (http://www.ub.tuwien.ac.at).

The approved original version of this diploma or master thesis is available at the main library of the Vienna University of Technology (http://www.ub.tuwien.ac.at/englweb/).

D I P L O M A R B E I T

Adaptive Mesh Generation

ausgeführt am
Institut für Mikroelektronik
der Technischen Universität Wien

unter Anleitung von
Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Erasmus Langer
und
Dipl.-Ing. Dr.techn. René Heinzl

durch

JOSEF WEINBUB

Alter Ziegelweg 8/2/10
A-3430 Tulln, Österreich

Matr. Nr. 0225909
geboren am 19. Dezember 1982, Wien

Wien, im September 2009 _____

Abstract

This thesis explores approaches towards generic algorithms used in the field of adaptive mesh generation. Typically, the applied mesh adaption approach depends solely on the field of application at hand resulting in significant differences in underlying implementations. However, applying modular and generic software design enables algorithm reusability thereby reducing overall implementation efforts considerably. The thesis introduces an approach to generalize typical algorithms used for mesh adaption and generation. The generalization aims to extend the applicability of such algorithms to new areas. By using the results of the generalization procedure for mesh adaptation, the functionality of the generalization approach is ensured. Furthermore, a mesh simplification algorithm based on the evaluation of color data is introduced which provides facilities to convert images into meshes. Additionally, the initial mesh is simplified to remove mesh elements of low color variation which facilitates a reasonable reduction of the mesh size while simultaneously preserving details.

Kurzfassung

Diese Arbeit untersucht Ansätze hinsichtlich generischer Algorithmen, angewandt im Bereich der adaptiven Erzeugung von Diskretisierungsgittern. Typischerweise hängen die angewandten
Gitter-Adaptierungsansätze einzig und allein vom vorliegenden Anwendungsbereich ab, was signifikante Unterschiede in den zugrundeliegenden Implementierungen mit sich bringt. Die Anwendung von modularem und generischem Software-Design erlaubt die Wiederverwertbarkeit von Algorithmen, sodass der generelle Implementierungsaufwand beträchtlich sinkt. Die Arbeit präsentiert einen Ansatz zur Generalisierung von typischen Algorithmen, eingesetzt für Gitter-Adaptierung und Erzeugung. Ziel dieser Generalisierung ist es, die Anwendbarkeit von solchen Algorithmen in neue Bereiche zu erweitern. Die Funktionalität des Generalisierungsansatzes kann durch die Anwendung der Ergebnisse der Generalisierungsprozedur für Gitter-Adaptierung gezeigt werden. Weiters wird ein Gitter-Vereinfachungsalgorithmus vorgestellt, basierend auf der Evaluierung von Farbdaten, der die Konvertierung von Bildern in Gittern erlaubt. Das initiale Gitter wird vereinfacht - bei gleichzeitiger Erhaltung wichtiger Details -, indem die Elemente von niedriger Farbänderung entfernt werden. Dadurch wird eine angemessene Reduzierung der Gitter Größe ermöglicht.

Acknowledgments

Many thanks go to Prof. Langer for supporting my work. My sincere thanks is given to René Heinzl who always believed in me and has supported me since the beginning of my project based work at the Institute for Microelectronics in 2005. René introduced me to many different fields of scientific computing. For me the most formative fields were mesh generation, mesh adaption, generic programming and computational topology. Without his unwearying aim to broaden my horizon I would not have evolved as I have. Many thanks go to Philipp Schwaha who supported me in many different ways. His profound knowledge in physics, the Linux operating system and in the English language were crucial in many cases. Furthermore, I want to thank Franz Stimpfl who supported me with mesh generation tasks and implementation advice. Many thanks go to Carlos Rafael Giani who supplied me with extended visualization capabilities. I also want to thank Georg Mach for discussions concerning several mathematical interpretations and formulations. Many thanks go to Clemens Klöckler who was a helpful companion throughout all my studies. I want to thank Prof. Selberherr for all the possibilities to participate in projects at the institute as an undergraduate student. Since these project participations were crucial for my development. Last but not least my sincere thanks are given to my parents and my sister. Their steady support especially at the beginning of my studies was of particular importance.

Contents

1 Introduction **1**

1.1 General 1

1.2 Overview 2

2 Related Work **3**

2.1 Mesh Generation and Adaption 3

2.1.1 Triangle 3

2.1.2 Computational Geometry Algorithms Library 4

2.1.3 TetGen 4

2.2 Further Related Work 5

2.2.1 Generic Scientific Simulation Environment 5

2.2.2 Boost 6

2.2.3 Wings 3D 6

3 Theory **8**

3.1 Simplicial Complex 9

3.1.1 Simplex 9

3.1.2 Simplicial complex 10

3.2 Triangulation 11

3.2.1 General 11

3.2.2 Mesh 12

3.3 Cell Complex 16

3.3.1 Boundary Operation 17

3.4 Algorithm Generalization 18

3.4.1 Algorithms 18

3.4.2 Generalization 19

3.4.3 Boundary Operation 21

3.5 Mesh Quality 23

4 Toolkit **25**

4.1 Hull extractor . . . 26
4.2 Mesh Classification . . . 27
4.3 Generic Mesh Generator Interface . . . 30
4.4 Generic Functor Queue . . . 32

5 Applications and Results **34**
5.1 Mesh Adaption based on Image Data . . . 35
5.1.1 Implementation . . . 35
5.1.2 Input Data . . . 39
5.1.3 Mesh adaptions . . . 41
5.1.4 Conclusion . . . 48
5.2 Mesh Adaption based on Quality Evaluation . . . 49
5.2.1 Implementation . . . 49
5.2.2 Input Data . . . 51
5.2.3 Mesh adaptions . . . 55
5.2.4 Conclusion . . . 79
5.3 Mesh Adaption based on Inclusion Tests . . . 80
5.3.1 Implementation . . . 80
5.3.2 Input Data . . . 82
5.3.3 Mesh adaptions . . . 83
5.3.4 Conclusion . . . 89

6 Summary and Outlook **90**

Bibliography **90**

List of Tables

2.1 Comparison of the advantages and disadvantages of the *Triangle* implementation. 3

2.2 Comparison of the advantages and disadvantages of the CGAL implementation. 4

2.3 Comparison of the advantages and disadvantages of the TetGen mesh generation engine. . 4

2.4 Comparison of the advantages and disadvantages of the *GSSE*. 5

2.5 Comparison of the advantages and disadvantages of the *MPL*, *Fusion*, *Phoenix*, *GIL* and *TypeTraits* implementations. 6

2.6 Comparison of the advantages and disadvantages of Wings 3D. 7

3.1 The type of information required to setup a triangulation and a mesh. Note that a mesh additionally requires topological information to define the boundary of the discretized region. 13

3.2 The topological information used to define the boundary of a mesh domain. Note that the numbers relate to the corresponding numbers in **Figure 3.5** . It may be interpreted as a recipe how the geometric information has to be connected to obtain the boundary. 14

3.3 Overview of the relations between an arbitrary p-cell (**left**), topological objects (**middle**) and the geometrical counterparts (**right**) for a dimension p of up to two. Note that there is only a unique relation between the geometrical entities and the topological counterparts for $p = 0, 1$. A unique relation for $p > 1$ can only be achieved by additional information called *Cell Topology* [6] 16

3.4 Overview of the relations between p-cell notations (**left**), topological objects (**middle**) and the geometrical counterparts (**right**) for a dimension p of three. Note that there is only a unique relation between the geometrical entities and the topological counterparts for $p = 0, 1$. A unique relation for $p > 1$ can only be achieved by additional information called *Cell Topology* [6] 16

3.5 The dependencies of typical mesh adaption/generation algorithms. Note that each name of the algorithm is somehow related to the object to which the algorithm is supposed to be applied. 19

3.6 Overview of the relation between algorithms and their generalized counterparts. Note that the generalized algorithms lack information about the dimension and the associated geometrical entity. Analogously to the p-cell notation a unique relation can only be established by providing the dimension and the cell topology. 19

3.7 Examples of typical applications of the generalization procedure. 20

3.8 Derivation of geometrical algorithms based on generalized algorithms, dimensions and cell topologies. Note that dimension 1 relates to the dimension of first p-cell, and dimension 2 relates to the dimension of the second p-cell. The generalized algorithm in line 1 reads: 0-cell-in-2-cell. The unique relation to the point-in-triangle algorithm can be established using the simplex cell topology. 21

3.9 Overview of the results of the boundary operator applied on a 2-cell (**top**) and on a 3-cell (**bottom**). 21

3.10 Overview of application cases based on the boundary operation and the *cell metric* algorithm. Note that the cell topology is omitted. Therefore, the cell is considered to have a simplex topology. Of course this concept can be applied to other cell topologies, i.e. cubes, as well. 22

3.11 Overview of application cases based on the boundary operation and the *p-cell in q-cell* algorithm. Note that the cell topology is omitted. Therefore the cell is considered to have a simplex topology. This algorithm processes two cells, therefore the boundary operation has to be applied on both cells. Investigation of the depicted example in line 1: The cell 1 dimension result is $(p - n) = (3 - 2) = 1$-cell. A 1-cell in combination with a simplex cell topology yields an edge. Analog for the second cell $(q - n) = (3 - 1) = 2$-cell. A 2-cell with a simplex cell topology yields a triangle. The final decoding yields: *edge in triangle*. However, as each result of the boundary operations is processed, the decoding actually yields in own words: Test each 1-cell of the source 3-cell if it is inside of one of the 2-cells of the target 3-cell. 23

4.1 Overview of currently and prospectively available mesh engines. Note that **x** denotes availability, **o** denotes under development. 31

5.1 Overview of the simplification results. **Column 1** identifies the different meshes. **Column 2** provides the file size of the input *png* file. **Column 3** lists the number of pixels, which is computed by multiplying the width by the length in pixel. **Column 4-7** provide the 0-cell size of the generated meshes using the different thresholds. Note the quite unexpected behavior of the 0-cell size for the AUSTRIA, DEVICE and TU VIENNA LOGO meshes. Although the threshold increases, the number of 0-cells partially increases as well, which is due to the intervention of the mesh engine to preserve mesh generation constraints. . . . 48

5.2 Informal evaluation of the most effective threshold values based on visual qualification. Four of six simplifications seem to perform best with a threshold value of 10%. The TUX mesh simplification looses graphical accents for thresholds greater than 4%. The KNEE mesh sacrifices substantial anatomical details for a threshold greater than 8%. 48

List of Figures

3.1 Possible nonempty types of simplices in $\mathbb{R}^3$. From left to right: 0-simplex (point), 1-simplex (line segment), 2-simplex (triangle), 3-simplex (tetrahedron) 10

3.2 Possible violations of one of the properties required for a simplex complex. **left:** Two 0-simplices, and one 1-simplex are missing. **middle:** The 2-simplices meet along a segment that is not an 1-simplex of either 2-simplices. **right:** A 1-simplex crosses the 2-simplex at an interior point. 11

3.3 Illustration of the topological triangulation of a closed disk. Note that from a topological point of view, the triangle and the disk are the same (*homeomorph*). Therefore, one *topological triangulation* is capable of discretizing several *geometric objects*, as long as they are homeomorphic to each other. This illustrates the strength of topological formulations and their power of generalization. 12

3.4 Comparison of exemplary discretization results of a star shaped object based on a *triangulation* (**left**) and a *mesh* (**right**). The triangulation does not preserve the *concave* regions, while a mesh does. 13

3.5 Input data for generating a mesh of a simple star object. 13

3.6 Overview of the topological decomposition of a 3-cell (simplex topology) due to the application of the boundary operator ∂_p. Note that each time the boundary operator is applied, $(p-1)$-cells are retrieved. 17

3.7 A geometrical algorithm can be separated into a generalized algorithm, cell dimension and cell topology. 20

3.8 A geometrical algorithm which processes two cells can be separated into a generalized algorithm, two cell dimensions (one for each cell) and the cell topology. 20

3.9 A geometrical algorithm which processes one cell can be separated into a generalized algorithm, cell dimension and cell topology. The initial approach introduced in **Figure 3.7** is extended by the derivation of the cell dimension. The cell dimension is computed by subtracting the cell boundary dimension from the cell base dimension as depicted in **Table 3.9**. 21

3.10 A geometrical algorithm which processes two cells can be separated into a generalized algorithm, two cell dimensions (one for each cell) and the cell topology. The initial approach introduced in **Figure 3.8** is extended by the derivations of the cell dimensions. The cell dimensions are computed by subtracting the corresponding cell boundary dimensions from the related cell base dimensions as depicted in **Table 3.9**. 22

3.11 Influence of large angles of a 2-simplex on ∇g. Note the quantities associated with each 0-cell. At the middle of the bottom $1 - cell$ the interpolated value is 50. Computing $\frac{\partial g}{\partial y}$ reveals that the partial derivative tends to infinity if the **red** angle reaches $180°$. Note that a 3-simplex shares this behavior, but instead of investigating the angles between the 1-simplices the dihedral angles between the 2-simplices have to be investigated. 24

4.1 Visualization of the functionality of the *hull extractor* tool. **left:** The input volume mesh is shown. Note that the interior elements are larger in size than the boundary elements. **right:** The derived hull mesh is shown. Note that a halfspace has been set to be invisible. Therefore a view of the interior of the meshes is provided. 26

4.2 Classification of different types of 2-simplices. **left:** Two of the three angles of a *blade* are small. The edges are more ore less of equal length. A blade is considered *degenerate*. **middle:** A *dagger* has one considerably shorter edge compared to the other two edges. A dagger is considered *degenerate*. **right:** A *round* has edges and angles of nearly the same size. A round is considered *optimal*. 27

4.3 Element quality visualization of an exemplary 2-simplex and 3-simplex mesh. **left:** A 2-simplex mesh of the TU VIENNA LOGO is investigated. Elements with quality of up to 0.6 are highlighted. **right:** A 3-simplex mesh of a CUBE is analyzed. Elements with quality of up to 0.2 are highlighted. 27

4.4 Visualization of *degenerated* elements of an exemplary 2-simplex and 3-simplex mesh. **left:** A 2-simplex mesh of the TU VIENNA LOGO is investigated. *Dagger* triangles of 60% and above are highlighted. **right:** A 3-simplex mesh of a CUBE is analyzed. Degenerated *sliver* tetrahedrons of 90% and above are highlighted. 28

4.5 Visualization of *optimal* elements of an exemplary 2-simplex and 3-simplex mesh. **left:** A 2-simplex mesh of the TU VIENNA LOGO is investigated. *Round* triangles of 60% and above are highlighted. **right:** A 3-simplex mesh of a CUBE is analyzed. *Round* tetrahedrons of 90% and above are highlighted. 28

4.6 Overview of the distribution of classified elements in percent of an exemplary 2-simplex and 3-simplex mesh. **left:** The classification of the TU VIENNA LOGO. Note that the optimal *round* elements hold a fraction of more than 50%. **right:** The classification of the CUBE mesh. There is a vast majority of optimal *round* tetrahedrons. Degenerated *sliver* and *wedge* elements are both below 10%. 29

4.7 Distribution of angles and dihedral angles respectively of an exemplary 2-simplex and 3-simplex mesh. **left:** The angle distribution of the TU VIENNA LOGO. Note, that there are no angles smaller than $20°$, and only very few greater than $120°$. **right:** The distribution of dihedral angles of the CUBE mesh is presented. Note that the majority of dihedral angles is greater than $20°$ and smaller than $140°$. However, a few angles appear outside of this region, which indicates very bad elements. 29

5.1 The workflow of the image data based mesh simplification. **top:** The input image is the logo of the *Vienna University of Technology*. **bottom left:** The mesh after direct conversion from the image to a mesh. The number of points is 32526. Note that each pixel is converted into two triangles. Further note, that the background layer has been ignored for the conversion process. Therefore the boundaries are preserved precisely. **bottom right:** The simplified mesh has a point size of 12395. Note, that the mesh free regions of the raw mesh have been triangulated to setup one single connected mesh. Further note the decrease of the point density in the homogeneous regions, for example the letters. 36

5.2 Basic principle of the implemented mesh simplification based on image data. 36

5.3 Visualization of the incremental postprocessing algorithm. **top:** A detailed view of an image with a thick changeover. **middle left:** The simplification result based on the initial evaluation. Note that the grey regions of the input image result in a thick, homogeneous mesh region. **middle right:** One iteration of the thinning algorithm is applied. Note that the homogeneous mesh region shrinks quite symmetrically. **bottom left:** Two iterations of the postprocessing algorithm are applied. **bottom right:** After three iterations the initial thick changeover is nearly reduced to a line. Note that thin mesh regions of the initial evaluation are more or less preserved. 38

5.4 Image of a map of *Austria*. 39

5.5 Image of a doping concentration within an electronic device (*MOSFET*). Note that the image has been recorded from an existing mesh, therefore the 2-simplices of the mesh are visible. 39

5.6 The logo of the *Vienna University of Technology*. 39

5.7 An image of the official mascot of the the *Linux* operating system named *Tux*. 40

5.8 A *Magnetic Resonance Image* (*MRI*) of a human head. 40

5.9 A *Magnetic Resonance Image* (*MRI*) of a human knee. 40

5.10 Visualization of the simplification process applied to the AUSTRIA image. Due to the solid color within the federal states, the influence of different threshold values is limited and no significant changes can be identified. Therefore, only one simplified mesh is provided. . . 42

5.11 Visualization of the simplification process applied to the DEVICE image using different threshold values. Note that the *black* mesh lines of the input image have been ignored by the simplification algorithm, as it is possible to specify a certain color layer (including a tolerance level) to be ignored by the simplification process. In other words, it enables to specifically remove color from the mesh. **middle left:** Note the fine region which contains a coarse region at the *pn*-junction. This is due to the sensible threshold value of 4%. The green region between the blue and yellow regions is thick enough to be coarsed for this threshold value. **bottom right:** The *pn*-junction region is effectively preserved. 43

5.12 Visualization of the simplification process applied to the TU VIENNA LOGO image using different threshold values. Note that the input image does not offer many color gradients. Therefore the use of different threshold values is negligible. However, the transition regions from the background to the foreground (blue) should be sharper. Note the thick mesh regions surrounding the *vienna* writing and the horizontal line. 44

5.13 Visualization of the simplification process applied on the TUX image using different threshold values. Note the steady disappearance of the dark grey shadows on the left and right side of the chest and on the head (from **middle left** to **bottom right**). 45

5.14 Visualization of the simplification process applied to the HEAD image using different threshold values. **middle left:** Clearly the threshold level is too sensitive, as almost no contours can be identified. This is due to the fact that large regions of the image provide very smooth color gradients which constantly exceed the given threshold. For example, the areas around the mouth and the nose only change in shades of grey. **bottom right:** If the threshold level is sufficiently large to overcome the color gradients in these regions, coarsement takes effect. This behavior depicts the nonlinear dependency of the threshold on the color variation. 46

5.15 Visualization of the simplification process applied to the KNEE image using different threshold values. **middle left:** For the left part of the image the threshold value is too small to enable contour identification. For this region the same nonlinear behavior with respect to the threshold value can be observed as with the HEAD image. As the threshold increases contours become more discernible. **bottom right:** Contours can be identified reasonably. 47

5.16 Basic principle of the implemented mesh adaption application. 49

5.17 **left:** A mesh discretizing a map of *austria*. **right:** A mesh carrying a doping profile of a *mosfet*. 51

5.18 **left:** A mesh of the *TU Vienna Logo*. **right:** A mesh of the official mascot of the the *Linux* operating system named *Tux*. 51

5.19 **left:** A mesh of an *magnetic resonance image* (*MRI*) of a human head. **right:** A mesh of an *magnetic resonance image* (*MRI*) of a human knee. 52

5.20 **left:** A mesh of the *stay puft marshmallow man*. **right:** A mesh of a sculpture named *StGallen*. 52

5.21 **left:** A mesh of an ordinary *cube*. **right:** A mesh of a chinese *dragon*. 53

5.22 **left:** A mesh of a *cow*. **right:** A mesh of a *house*. 53

5.23 Visualization of the element quality distribution of the AUSTRIA mesh. The detailed view reveals, that the adaption removes bad input elements, but introduces new degenerated elements. The lowest occurring element quality is increased from a value of 0.4206 to 0.4463. 55

5.24 Angle distributions in the AUSTRIA mesh before and after adaption. The number of elements with an angle of 120° has decreased, the number containing 20° angles has even halved, while the count of elements with an angle of 30° and 60° has increased after the adaption. The improvement due to adaption is made explicit by the shift to this intermediate range of angles. 56

5.25 Overview of classified element types in the AUSTRIA mesh before and after adaption. The number of *dagger* elements has been reduced while the number of *blades* and *rounds* has been increased by using the adaption. 56

5.26 Visualization of the mesh adaption result of the DEVICE mesh. The zoomed view shows that there are less elements with a quality ≤ 0.6 after adaption than in the input mesh. The lowest occurring element quality has increased only slightly, from 0.4279 to 0.4281. . . . 57

5.27 Angle distributions in the DEVICE mesh before and after adaption. The number of elements with an angle of 20°, 25° and 130° has decreased, while the count of elements with an angle of 30°, 60° and 90° has increased after the adaption. The improvement due to adaption is made explicit by the shift to this intermediate range of angles. 58

5.28 Overview of classified element types in the DEVICE mesh before and after adaption. The number of *dagger* elements has been reduced while the number of *blades* and *rounds* has been increased by using the adaption. 58

5.29 Mesh adaption of the TU VIENNA LOGO. A reduction of bad elements can clearly be recognized. The lowest occurring element quality is slightly decreased, from 0.4153 to 0.4043. 59

5.30 Angle distributions in the TU VIENNA LOGO mesh before and after adaption. The number of elements with an angle of $20°$, $40°$, $110°$ and $130°$ has decreased, while the count of elements with an angle of $35°$ and $70°$ has increased after the adaption. The improvement due to adaption is made explicit by the shift to this intermediate range of angles. 60

5.31 Overview of classified element types in the TU VIENNA LOGO mesh before and after adaption. The number of *dagger* elements has been reduced while the number of *blades* and *rounds* has been increased by using the adaption. 60

5.32 Mesh adaption of the TUX mesh. Clearly a reduction of bad elements can be recognized. The lowest occurring element quality is slightly increased, from 0.4118 to 0.4124. 61

5.33 Angle distributions in the TUX mesh before and after adaption. The number of elements with an angle of $20°$, $40°$, $110°$ and $130°$ has decreased, while the count of elements with an angle of $35°$, $60°$ and $83°$ has increased after the adaption. The improvement due to adaption is made explicit by the shift to this intermediate range of angles. 62

5.34 Overview of classified element types in the TUX mesh before and after adaption. The number of *dagger* elements has been reduced while the number of *blades* and *rounds* has been increased by using the adaption. 62

5.35 Mesh adaption of the HEAD mesh. A decrease of elements offering a quality of ≤ 0.6 can be recognized. The lowest occurring element quality is increased notably, from 0.4059 to 0.4385. 63

5.36 Angle distributions in the HEAD mesh before and after adaption. The number of elements with an angle of $20°$ and $110°$ has decreased, while the count of elements with an angle of $30°$ and $70°$ has increased after the adaption. Generally, there is no significant change in the angle distribution. 64

5.37 Overview of classified element types in the HEAD mesh before and after adaption. The number of *dagger* elements has been reduced while the number of *blades* and *rounds* has been increased by using the adaption. 64

5.38 Mesh adaption of the KNEE mesh. A reduction of degenerated elements can be identified. The lowest occurring element quality is slightly increased, from 0.4204 to 0.4231. 65

5.39 Angle distributions in the KNEE mesh before and after adaption. The number of elements with an angle of $20°$, $80°$ and $90°$ has decreased, while the count of elements with an angle of $40°$ and $60°$ has increased after the adaption. Generally, there is no significant change in the angle distribution. 66

5.40 Overview of classified element types in the KNEE mesh before and after adaption. The number of *dagger* elements has been reduced while the number of *blades* and *rounds* has been increased by using the adaption. 66

5.41 Visualization of the element quality distribution of the STAYPUFT mesh. Elements with quality ≤ 0.2 are highlighted. The total number of elements having a quality of ≤ 0.2 is decreased considerably, while the lowest occurring element quality is decreased, from 0.0920 to 0.0090. 67

5.42 Angle distributions in the STAYPUFT mesh before and after adaption. The number of elements with an angle of $10°$ and $20°$ has decreased, while the count of elements with an angle of of $30°$, $70°$ and $90°$ has increased after the adaption. The improvement due to adaption is made explicit by the shift to this intermediate range of angles. 68

5.43 Overview of classified element types in the STAYPUFT mesh before and after adaption. The number of *needle* and *wedge* elements has been reduced while the number of *slivers* and *rounds* has been increased by using the adaption. 68

5.44 Visualization of the element quality distribution of the STGALLEN mesh. Elements with quality ≤ 0.2 are highlighted. Clearly the total number of elements offering a quality of ≤ 0.2 is decreased considerably. The lowest occurring element quality is slightly decreased, from 0.0816 to 0.0623. 69

5.45 Angle distributions in the STGALLEN mesh before and after adaption. The number of elements with an angle of 20° and 160° has decreased, while the count of elements with an angle of 30° and 70° has increased after the adaption. The improvement due to adaption is made explicit by the shift to this intermediate range of angles. 70

5.46 Overview of classified element types in the STGALLEN mesh before and after adaption. The number of *needle* and *wedge* elements has been reduced while the number of *slivers* and *rounds* has been increased by using the adaption. 70

5.47 Visualization of the element quality distribution of the RAND2 mesh. Elements with quality ≤ 0.2 are highlighted. Clearly the total number of elements offering a quality of ≤ 0.2 is decreased considerably. The lowest occurring element quality is slightly increased, from 0.0885 to 0.1071. 71

5.48 Angle distributions in the RAND2 mesh before and after adaption. The number of elements with an angle of 10° has decreased, while the count of elements with an angle of 170° has increased after the adaption. The improvement due to adaption is made explicit by the shift to this intermediate range of angles. 72

5.49 Overview of classified element types in the RAND2 mesh before and after adaption. The number of *round* and *sliver* elements has been reduced while the number of *wedges* has been increased by using the adaption. 72

5.50 Visualization of the element quality distribution of the DRAGON mesh. Elements with quality ≤ 0.2 are highlighted. Clearly the total number of elements offering a quality of ≤ 0.2 is decreased considerably. The lowest occurring element quality is decreased, from 0.0825 to 0.0069. 73

5.51 Angle distributions in the DRAGON mesh before and after adaption. The number of elements with an angle of 10° and 170° has decreased, while the count of elements with an angle of 90° has increased after the adaption. The improvement due to adaption is made explicit by the shift to this intermediate range of angles. Generally there is no significant change in the angle distribution. 74

5.52 Overview of classified element types in the DRAGON mesh before and after adaption. The number of *wedges* has been increased by using the adaption. 74

5.53 Visualization of the element quality distribution of the COW mesh. Elements with quality ≤ 0.2 are highlighted. Clearly the total number of elements offering a quality of ≤ 0.2 is decreased considerably. The lowest occurring element quality is decreased, from 0.0592 to 0.0103. 75

5.54 Angle distributions in the COW mesh before and after adaption. The number of elements with an angle of 10° and 160° has decreased, while the count of elements with an angle of 50° has increased after the adaption. The improvement due to adaption is made explicit by the shift to this intermediate range of angles. 76

5.55 Overview of classified element types in the COW mesh before and after adaption. The number of *rounds* has been increased by using the adaption. 76

5.56 Visualization of the element quality distribution of the HOUSE mesh. Elements with quality ≤ 0.2 are highlighted. Clearly the total number of elements offering a quality of ≤ 0.2 is decreased considerably. The lowest occurring element quality is decreased, from 0.1300 to 0.0791. 77

5.57 Angle distributions in the HOUSE mesh before and after adaption. The count of elements containing angles between $40°$ and $120°$ has increased after the adaption. The improvement due to adaption is made explicit by the shift to this intermediate range of angles. . . 78

5.58 Overview of classified element types in the HOUSE mesh before and after adaption. The number of *round* elements has been reduced while the number of *slivers* has been increased by using the adaption. 78

5.59 Input meshes used for inclusion tests. 82

5.60 Visualization of a basic collision simulation based on the DISK and TRIANGLE mesh. The penetration progresses from **top** to **bottom**. The **left** column depicts the identification of collided cells. The colliding elements are highlighted in **red**. The **right** column outlines the mesh adaption result. All colliding cells are removed. 83

5.61 Visualization of a basic collision simulation based on the DISK and RECTANGLE mesh. The penetration progresses from **top** to **bottom**. The **left** column depicts the identification of collided cells. The colliding elements are highlighted in **red**. The **right** column outlines the mesh adaption result. All colliding cells are removed. 84

5.62 Visualization of a basic collision simulation based on the DISK and RING mesh. The penetration progresses from **top** to **bottom**. The **left** column depicts the identification of collided cells. The colliding elements are highlighted in **red**. The **right** column outlines the mesh adaption result. All colliding cells are removed. 85

5.63 Visualization of a basic collision simulation based on the SPHERE and CONE mesh. The penetration progresses from **top** to **bottom**. The **left** column depicts the identification of collided cells. The colliding elements are highlighted in **red**. The **right** column outlines the mesh adaption result. All colliding cells are removed. 86

5.64 Visualization of a basic collision simulation based on the SPHERE and CYLINDER mesh. The penetration progresses from **top** to **bottom**. The **left** column depicts the identification of collided cells. The colliding elements are highlighted in **red**. The **right** column outlines the mesh adaption result. All colliding cells are removed. 87

5.65 Visualization of a basic collision simulation based on the SPHERE and TORUS mesh. The penetration progresses from **top** to **bottom**. The **left** column depicts the identification of collided cells. The colliding elements are highlighted in **red**. The **right** column outlines the mesh adaption result. All colliding cells are removed. 88

Chapter 1

Introduction

1.1 General

A mesh can be interpreted as a transition from our continuous physical perception to a discrete representation. Meshes are used in several different fields such as scientific computing and computer graphics. For example, a mesh may represent an electronic device which enables to utilize numerical methods for the solution of the partial differential equations describing its behavior.

Mesh generation refers to the construction of meshes from some form of input data. Basically a mesh is composed of elements, which represent geometrical entities, eg., triangles, tetrahedra, quadrilaterals, hexahedrons etc.

Each particular field of application prefers a mesh based on certain elements, as each of the introduced elements offers advantages and disadvantages [1][2][3]. Since each field of application has its own, specific requirements, the generation process is not generalized but aimed at the specific area of application. For example, the field of computer graphics does not require the mesh elements to be perfectly aligned face to face. In contrast to this, the field of scientific computing such misalignment is not permissible and the mesh must not contain any holes additionally to the elements meeting at common faces and at common points. Such requirements are due to the prerequisites of the subsequently used numerical methods, as they otherwise fail to obtain reasonable solutions.

Mesh adaption denotes the adjustment of an existing mesh to meet certain requirements. An initially generated mesh, for example, may not contain enough points to model a complex shape with reasonable accuracy. Therefore additional mesh elements are introduced at distinctive places of the shape which locally increase the element density. Consequently the shape can be modelled more accurately. Another example would be to decrease the number of mesh elements in regions of constant simulation values which have been obtained by numerical methods. The conclusion may be that the element density in such regions can be reduced as there is no change in value. Typically the reduction of mesh elements eases subsequent computational efforts. This is due to the fact that algorithms usually processes each of the mesh's element. Therefore, a reduction of the number of elements decreases the computational effort.

The title of this thesis, *adaptive mesh generation*, denotes the merging of three differently handled fields: mesh, adaption and generation. Therefore this expression relates to an unified approach for generating discretizations which specifically fit an aimed application area. Typically the adaption and generation of meshes is computational intensive, especially in three dimensions. Therefore, those applications are typically investigated and provided separately. However, as the algorithms improve in efficiency and the available computational performance increases the original mesh adaption tools can evolve to standalone adaptive mesh generation applications [4].

The goal of this thesis is to develop generic algorithms for adaptive mesh generation tasks with a focus on the application of modern programming concepts to achieve highly reusable and maintainable implementations. Typical algorithms used in the field of adaptive mesh generation should be generalized to arbitrary dimensions and elements, thereby increasing the scope of these algorithms significantly. The implemented applications act as prototypes to investigate the applicability of generalized algorithms in the field of adaptive mesh generation.

The challenge is to implement a highly generic code basis, where the expression *highly generic* should emphasize the focus on reusability. For example, the implementation has to handle different data structures, different mesh element types and dimensions in a concise manner. Such highly generic implementations have been researched in the past years [5][6].

Generic code requires modern programming techniques. Furthermore high-performance is of utmost interest as well. Therefore the C++ programming language is used in combination with the Boost libraries [7]. To store and process meshes of arbitrary type the *Generic Scientific Simulation Environment* (GSSE [8]) is used.

1.2 Overview

Chapter 2 puts the thesis into context by introducing related applications and libraries presenting advantages and disadvantages of each of them as well as improvements implemented during this thesis.

Chapter 3 provides the theoretical foundations for the thesis, defining the basics of discrete representations as well as how to evaluate the quality of meshes. Furthermore, an approach to algorithm generalization, based on element type and element dimension abstractions, is presented.

Chapter 4 discusses applications which are used in conjunction with the actual adaptive mesh generation applications. Furthermore, a programming technique is introduced which enables to elegantly concatenate algorithms, which consequently yields the ability to implement even complex algorithm chains using a modular implementation design.

Chapter 5 presents three applications for adaptive mesh generation. The first application generates a mesh from images and simplifies them according to the color information. The second application analyzes the quality of the elements of a given mesh and enforces the reduction of degenerated elements. The third application introduces a basic collision application based on element inclusion tests. Two meshes are processed and a new mesh is generated which is free of any inclusion.

Chapter 6 summarizes the presented information. Furthermore an outlook is provided which suggests further advancements in the field of generic algorithm implementation for adaptive mesh generation.

Chapter 2

Related Work

This chapter introduces applications and libraries which have been crucial for the writing of this thesis.

The first part deals with applications for mesh generation and adaption. The advantages and disadvantages are presented as well as the improvements implemented during the work for this thesis.

The second part introduces applications and environments which are used in the actual implementation and for further related tasks.

2.1 Mesh Generation and Adaption

In the following mesh generation engines are introduced and discussed. The study of these engines has led to the development of an unified mesh generation framework.

2.1.1 Triangle

Triangle [9] is a two dimensional mesh generation application which generates high-quality triangular meshes. The implementation utilizes the *C* programming language. The following table provides an overview of the advantages and disadvantages of the Triangle implementation.

advantages	**disadvantages**
fast	no orthogonal software interface
robust	difficult material handling
mathematically guaranteed mesh algorithm	no algorithm reusability
	cumbersome mesh refinement
	no mesh coarsement
	no parallel mode

Table 2.1: Comparison of the advantages and disadvantages of the *Triangle* implementation.

The encapsulation of the Triangle mesh engine within the generic mesh generation framework enables to efficiently counter most of the listed disadvantages. A parallel mode can be introduced in conjunction with the ability to handle regions of different material. For example different threads can be made responsible for meshing different material regions. Algorithm reusability can be achieved by implementing generic algorithm interfaces which handle data type conversions by a wrapping approach[1]. However, the parallel mesh mode and the generic algorithm interface for internal Triangle algorithms have not been

[1] No data copy approach should be used, as this results in unnecessary overhead.

implemented yet. The mesh refinement and coarsement disadvantages have been addressed by using post-processing tools. The orthogonal software concept has been applied, as the postprocessing tools have been implemented separately. Therefore, the modular tools can be used in different applications cases. The Triangle mesh engine has been used for several mesh generation tasks.

2.1.2 Computational Geometry Algorithms Library

The *Computational Geometry Algorithms Library* (*CGAL*) [10] is a *C++* based collection of algorithms from the field of computational geometry including mesh generation engines. The following table provides an overview of the advantages and disadvantages of the CGAL implementation.

advantages	**disadvantages**
fast	no orthogonal software interface
exchangeability of numerical kernels	no material handling
mathematically guaranteed mesh algorithm	only minor use of generic implementation concepts
extended quality mesh generation	no parallel mode
basic mesh refinement	no mesh coarsement
	no reusability

Table 2.2: Comparison of the advantages and disadvantages of the CGAL implementation.

Note that CGAL enables to exchange the numerical kernels. Hence, applications can switch between *exact* and *inexact* implementations conveniently. Further, CGAL offers more quality mesh generation options compared to Triangle.

Similar to Triangle, CGAL's two dimensional mesh engine has been encapsulated using a generic mesh generation interface. Therefore, similar enhancements can be certified for the CGAL mesh engine.

Generally, the algorithms provided by CGAL lack reusability. This is due to the extensive use of algorithms encapsulated in member functions and free functions which are named specifically according to their purpose. One of the goals of this thesis is to implement algorithms which adhere to orthogonal software design principles. Therefore the work introduced in this thesis can be seen as an advancement of the CGAL implementation regarding implementation efficiency. The two dimensional mesh generation engine was used to generate several meshes.

2.1.3 TetGen

TetGen [11] is a three dimensional mesh generation application which generates high-quality tetrahedral meshes. The implementation uses the *C++* programming language. The following table provides an overview of the advantages and disadvantages of the TetGen implementation.

advantages	**disadvantages**
fast	no orthogonal software interface
robust	difficult material handling
mathematical guaranteed mesh algorithm	no algorithm reusability
intersection test	no parallel mode
	insufficient information retrieval
	cumbersome mesh refinement
	no mesh coarsement

Table 2.3: Comparison of the advantages and disadvantages of the TetGen mesh generation engine.

Informally TetGen can be seen as a three dimensional extension of Triangle and the implementation shares similar advantages and disadvantages. However, an intersection test is provided which is capable of reporting intersecting input elements. Unfortunately this information can not be accessed by the API.

The TetGen mesh engine has been encapsulated within a generic mesh generation interface. All tetrahedral meshes have been generated using this mesh generation engine.

2.2 Further Related Work

The following tools are crucial for the actual development and implementation of the applications and related investigations.

2.2.1 Generic Scientific Simulation Environment

The *Generic Scientific Simulation Environment* (*GSSE*) [8] is a programming environment for scientific computing with a focus on generic software design. The library uses the *C++* programming language and utilizes the *Boost* [7] libraries. The environment is the result of research in the field of software development for scientific computing [6]. Several generic and combinatorial datastructures are provided which are capable of handling arbitrary discretization schemes. Topological operations are provided which make use of the combinatorial datastructure. Such operations enable to compute the topological relations of arbitrary elements. Furthermore, they enable to determine the interface of two adjacent mesh elements for example. Another example is the extraction of the hull of a volume mesh. Traversal and data storage is separated which models the *orthogonal software philosophy*[2]. The following table provides an overview of the advantages and disadvantages of the GSSE.

advantages	**disadvantages**
highly generic concepts	steep learning curve
header only library	difficult debugging

Table 2.4: Comparison of the advantages and disadvantages of the *GSSE*.

A highly generic environment requires a profound and indepth study of the implementation to fully master all the provided capabilities. Due to the extensive use of the modern programming concepts provided by the Boost libraries, debugging can be challenging, especially for beginners. However, the benefit of the generic implementation justifies the implementation efforts, as one efficient, generic algorithm can easily replace several conventionally implemented algorithms.

The applications implemented for this thesis are based on the *GSSE*. A development version of the *GSSE* based visualization application *GSSE::CAD* has been used to create the vast majority of figures. The generalized approach of the *GSSE* and the related theoretical background have been crucial for the development of applications introduced in this thesis.

[2] Orthogonality in software development refers to the fact that interacting with a module does not influence another module. In other words, modules can be implemented and combined arbitrarily.

2.2.2 Boost

Boost [7] is a collection of *C++* libraries. Several libraries have been used for this thesis:

- *Generic Image Library* (*GIL*)

  GIL is a library focused on manipulating image data. A convenient interface enables to access image data from several image datatypes. The project is sponsored by *Adobe* [12].

- *Meta Programming Library* (*MPL*)

  MPL is a metaprogramming environment which enables to implement compile time programs.

- *Fusion*

  Fusion enables to implement algorithms and containers based on compile time and run time implementations.

- *Phoenix*

  Phoenix is part of the *Spirit* parser library and enables functional programming in *C++*.

- *TypeTraits*

  TypeTraits provide a large set of specific traits classes. Type checks and modifications can conveniently be implemented and can be used in combination with metaprogramming techniques.

advantages	**disadvantages**
highly generic concepts	steep learning curve
header only library	difficult debugging

Table 2.5: Comparison of the advantages and disadvantages of the *MPL*, *Fusion*, *Phoenix*, *GIL* and *TypeTraits* implementations.

Note that the GSSE and the introduced Boost libraries offer the same advantages and disadvantages. This is due to the strong interaction of GSSE with the Boost libraries. Therefore, the conclusion is similar to the GSSE. Several implementation details have been extracted and have been used as guidelines for parts of the thesis applications.

The *MPL*, *Fusion*, *Phoenix* and *TypeTraits* libraries have been used extensively throughout all implementations connected to the thesis. The *GIL* has been used to process image files.

2.2.3 Wings 3D

Wings 3D [13] is a three dimensional, open source modelling software supporting several different file-formats. Furthermore, a surface tessellation algorithm is provided which enables to generate triangular surface meshes. The following table provides an overview of the advantages and disadvantages of the Wings 3D implementation.

advantages	disadvantages
easy to use	no programming interface
user interface	no volume mesh generation
mathematical guaranteed mesh algorithm	no quality mesh generation
materials	no mesh adaption
reasonable support for fileformats	

Table 2.6: Comparison of the advantages and disadvantages of Wings 3D.

Note that due to the lack of a programming interface this mesh generation application has not been added to the generic mesh generation framework. However, a conversion application has been implemented to convert Wings 3D meshes into GSSE readable meshes. These meshes can be improved further or volume meshed using the generic mesh generation toolkit. Several three dimensional shapes have been generated by using *Wings 3D*.

Chapter 3

Theory

This chapter introduces the theoretical foundations required for the applications developed in **Chapter 5**.

Section 3.1 introduces the idea of simplex decompositions, which are generalizations of decompositions into triangles.

Section 3.2 extends the abstract *simplicial complex* approach to common technical formulations. Definitions emerged from different fields are discussed.

Section 3.3 generalizes the *simplicial complex* approach to arbitrary elements. This generalization introduces a layer of abstraction which enables to implement algorithms capable of processing arbitrarily shaped elements of arbitrary dimension.

Section 3.4 applies the combinatorial and generalized approach of a *Cell Complex* to typical algorithms for mesh generation and adaption. Algorithms are generalized by extracting the dimension and the related geometrical element, therefore increasing the applicability significantly.

Section 3.5 provides a brief overview on quality evaluation. The evaluation method for triangles and tetrahedrons is derived and discussed. This approach is extended from a single element to a set of elements. Therefore, a qualification of decompositions based on triangles and tetrahedrons is enabled.

3.1 Simplicial Complex

A *Simplicial Complex* is one of the fundamental modelling techniques of space[1]. Simplicial complexes are so called *unstructured meshes* [1][2][3]. The points in such meshes may have an arbitrarily varying number of local neighbors. In contrast to this, the adjacency information of points in the interior of a *structured mesh* is constant. For example, an interior point of a two dimensional structured mesh always has exactly four neighbors. By exploiting this constant structure the computational effort of traversal is reduced considerably. While this simplification is not available for unstructured meshes, they are superior to structured meshes when it comes to the modelling of complex irregular shapes and for mesh adaption applications. This is due to the fact that the overall number of elements in the mesh can be kept to a minimum compared to structured meshes, which, due to the restrictive nature of their construction, force the creation of many new elements even if only a single additional point is desired. In contrast to this, inserting an additional point into an unstructured mesh only requires the generation of new points in the direct vicinity of the inserted point.

Since a simplicial complex is a set of so called *simplices* having certain properties, therefore it is prudent to first examine the notions of a simplex.

3.1.1 Simplex

A *simplex* is understood to be the *simplest polytope*[2] in a given dimension. However, a more formal definition should not be omitted and is therefore also provided.

Definition 1 (Affine Independence) *A finite collection of points is affinely independent if no affine space of dimension i contains more than $i+1$ of the given points, $\forall i \in \{0, 1, 2, ..., k\}$. In $\mathbb{R}^d$, the largest number of affinely independent points is $d+1$.*

This rather abstract definition can be reformulated using *linear independence*.

Definition 2 (Linear Independence) *A finite collection of vectors $x_1, x_2, ..., x_k \in \mathbb{R}^d$ is linearly independent if the unique solution to $\sum_{i=1}^{k} \lambda_i x_i = 0$ is $\lambda_i = 0, \forall i = 1, 2, ..., k$. Otherwise, the vectors are linearly dependent.*

Definition 3 (Affine Independence - 2) *A finite collection of vectors $x_1, x_2, ..., x_k \in \mathbb{R}^d$ is affinely independent if the vector differences $(x_2 - x_1), ..., (x_k - x_1)$ are linearly independent. There are $(k-1)$ vector differences. In $\mathbb{R}^d$, the largest number of linearly independent vectors is d. Therefore the largest number k of vectors has to be $d+1$.*

The previous definition reveals a connection between the determination of the affine independence of a set of points[3] and linear independence tests, for example, collinear and coplanar tests.

The determination if three points in $\mathbb{R}^2$ are affinely dependent, for example, may be decided using a *collinearity* test. If the points are affinely dependent, they are collinear. Analogously, in three dimensions four points in $\mathbb{R}^3$ are affinely dependent if they are *coplanar*.

Using the detailed investigation of affine independence a *k-simplex* can be defined.

[1] Other discretization schemes exist as well which decompose a domain using other element types, such as: quadrilaterals, hexahedra, etc.

[2] A *polytope* is a generalization of a *polygon* to arbitrary dimensions. Therefore, a two dimensional *polytope* is a *polygon* and a three dimensional *polytope* is a *polyhedron*.

[3] A point can be seen as a vector pointing from an origin to the location of the point in space.

Definition 4 (k-simplex [14]) *A k-simplex is the convex hull*[4] *of a collection of $k+1$ affinely independent points, $\sigma = conv(S)$. The dimension of σ is $dim(\sigma) = k$.*

The following types of simplices exist in $\mathbb{R}^d$

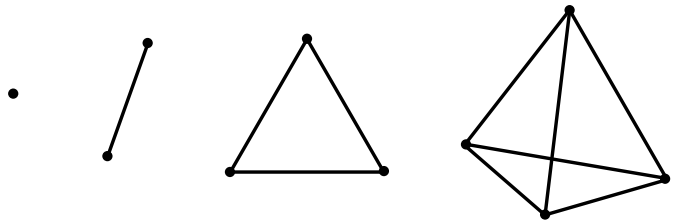


Figure 3.1: Possible nonempty types of simplices in $\mathbb{R}^3$. From left to right: 0-simplex (point), 1-simplex (line segment), 2-simplex (triangle), 3-simplex (tetrahedron)

A (-1)-simplex is also often encountered, which denotes the *empty set* $\emptyset$.

Note that the convex hull of any subset $T \subseteq S$ is again a simplex. It is a subset of $conv(S)$ and called a *face* of σ, denoted as $\tau \leq \sigma$. The empty set $\emptyset$ is a face of any simplex.

3.1.2 Simplicial complex

Using the previously introduced notion of a simplex it is possible to also formalize the description of sets of simplices. The so called simplex complex is among the most important such sets.

Definition 5 (Simplex complex [14]) *A simplicial complex contains the collection of faces of a finite number of simplices, any two of which are either disjoint or meet in a common face. For a collection K the following properties have to be satisfied:*

- *(1)* $\sigma \in K \wedge \tau \Rightarrow \tau \in K$
- *(2)* $\sigma, \upsilon \in K \Rightarrow \sigma \cap \upsilon \leq \sigma, \upsilon$

Condition **(1)** states that if σ is an element of a collection of simplices, than a face τ of σ has to be an element of the same collection as well.

Condition **(2)** defines how the simplices have to be connected to form a simplicial complex. The *intersection* of two simplices of the complex has to yield the simplices themselves or faces of the simplices.

Figure 3.2 depicts a few examples of violations for the previously stated properties.

[4]The *convex hull* is the smallest enclosing convex polytope [15].

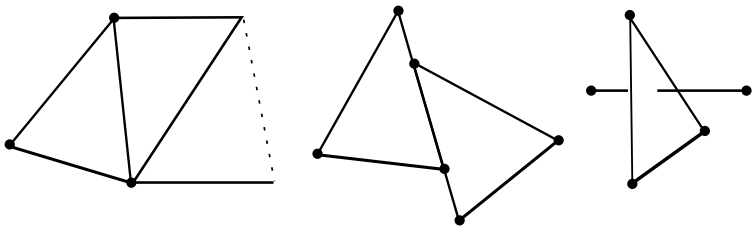


Figure 3.2: Possible violations of one of the properties required for a simplex complex. **left:** Two 0-simplices, and one 1-simplex are missing. **middle:** The 2-simplices meet along a segment that is not an 1-simplex of either 2-simplices. **right:** A 1-simplex crosses the 2-simplex at an interior point.

3.2 Triangulation

3.2.1 General

In general a *triangulation* denotes a discretization of an object, which typically models a real world structure. This section discusses several different definitions of the technical term triangulation. A *mesh* is introduced which is a specialized kind of triangulation. Furthermore, the difference of a triangulation and a mesh is discussed.

In the field of *geometry* there is no general definition what a triangulation is, but typically a *simplex complex* is implied.

However, in the field of *topology*, triangulation has indeed a precise definition.

Definition 6 (Triangulation - Topology [14]) *A triangulation of a topological space $\mathbb{X}$ is a simplicial complex K whose underlying space is homeomorphic to $\mathbb{X}$, $|K| \approx \mathbb{X}$. The space $\mathbb{X}$ is triangulable if it has a triangulation.*

Definition 6 asserts, that the triangulation of an arbitrary region offers the same orientation and connection properties as the region itself. **Figure 3.3** depicts this fact, where one topological triangulation is capable of discretizing several different geometrical objects, as long as the same orientation and connection information can be used.

Note that **Definition 6** utilizes a *simplicial complex* to define the *triangulation*. This indicates, that the term triangulation can also be used for three dimensional simplex discretizations. Even though in this case the elements used are tetrahedrons, each tetrahedron consists of triangles[5]. Therefore a three dimensional simplex discretization may also be referred to as a triangulation. Typically, however, the term *tetrahedrization* is used in such a case, which emphasizes that the used elements are tetrahedrons.

The *graph theory* provides a definition as well.

Definition 7 (Triangulation - Graph Theory [16]) *A triangulation of a closed surface[6] is a simple graph embedded on the surface so that each face is a triangle and any two faces share at most one edge.*

Note that **Definition 7** has to deal explicitly with adjacency. That is, that two faces share at most one edge. This is obsolete for **Definition 6**, as the definition is based on a simplex complex which already implies this property.

[5] The *simplex* property is applied.
[6] A *closed surface* is a surface without holes.

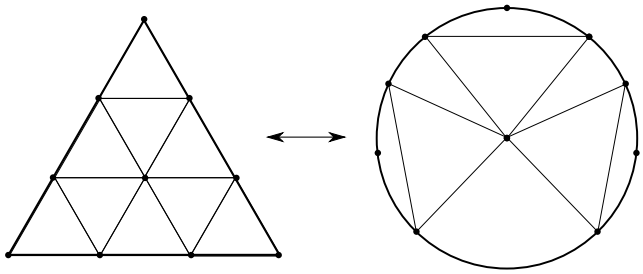


Figure 3.3: Illustration of the topological triangulation of a closed disk. Note that from a topological point of view, the triangle and the disk are the same (*homeomorph*). Therefore, one *topological triangulation* is capable of discretizing several *geometric objects*, as long as they are homeomorphic to each other. This illustrates the strength of topological formulations and their power of generalization.

3.2.2 Mesh

One of the meanings of a *mesh*[7] is a specialized type of *triangulation*. Typically, a triangulation is related to the result of a *convex hull* algorithm, which implies that *concave* objects[8] are discretized towards *convex* objects[9]. Consequently, the boundary information of the domain is lost, which is not always intended. Concave shapes require a boundary preserving approach.

Definition 8 (Conformal mesh [21]) *Let Ω be a closed, bounded domain in $\mathbb{R}^d, d \in \{2, 3\}$. The question is how to construct a conforming[10] triangulation of this domain. Such a triangulation will be referred to as a mesh of Ω and will be denoted by $\mathcal{T}_h$, with K a mesh element, ie., a simplex.*

$\mathcal{T}_h$ is a mesh of Ω if

- *(1)* $\Omega = \overline{\overset{\circ}{\bigcup_{K \in \mathcal{T}_h}} K}$.
- *(2) The interior of every element K in $\mathcal{T}_h$ is non empty.*
- *(3) The intersection of the interior of two elements in $\mathcal{T}_h$ is either:*
  the empty set $\emptyset$, a vertex, an edge or a face (if $d = 3$).

Condition **(1)** states that Ω is the *interior* of the *closure*[11] of the *union* of all elements $K \in \mathcal{T}_h$[12]. This condition ensures the preservation of the boundary, therefore allows to discretize concave regions.

Condition **(2)** ensures that there are no overlapping elements.

Condition **(3)** clarifies how elements K are to be concatenated. For example, if two elements are not neighbors, the intersection yields the empty set $\emptyset$. If they share a common vertex, the intersection results in the common vertex, and so forth.

For the sake of simplicity and brevity, a *conformal mesh* is denoted simply as a *mesh* in the following.

[7] In computer science a mesh may relate to a cipher or a network as well.

[8] In the field of *computer science* concave objects are common [17] [18] [19].

[9] Boundary preserving discretization algorithms exist which generate an initial convex hull. Based on the convex result the concave object is extracted by recovering the boundary [20].

[10] Conforming in the sense that the boundary of the domain is preserved.

[11] Generally a closure of an element is a *closed set*.

[12] As Ω is the *interior* of the *closure* it is an *open set* which corresponds to the domain.

Note, that K does not have to be a simplex, therefore **Definition 8** applies, for example, to *cuboid* based meshes as well. As long as the conditions apply, it is considered a valid mesh.

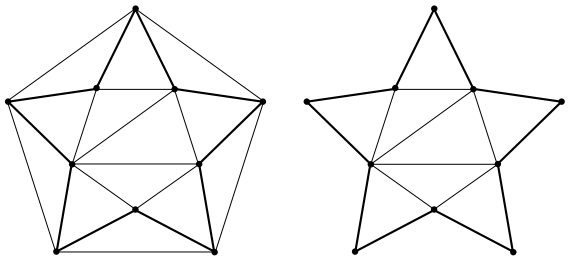


Figure 3.4: Comparison of exemplary discretization results of a star shaped object based on a *triangulation* (**left**) and a *mesh* (**right**). The triangulation does not preserve the *concave* regions, while a mesh does.

Several mesh generation algorithms have been developed. Among the most commonly used are:

- Delaunay [1] [3]
- Advancing Front [3]
- Octree [3]

Due to the boundary preserving nature of a mesh, additional topological information is required. The following table provides a simple overview of this fact.

	Geometry	**Topology**
Triangulation	x	
Mesh	x	x

Table 3.1: The type of information required to setup a triangulation and a mesh. Note that a mesh additionally requires topological information to define the boundary of the discretized region.

The topological information is used to connect and orient[13] the provided geometrical information to define the boundary of the discretized region.

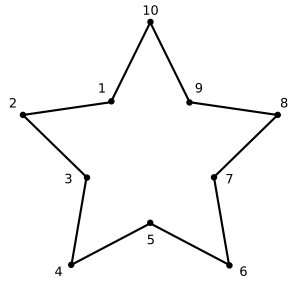


Figure 3.5: Input data for generating a mesh of a simple star object.

For example, according to **Figure 3.5** the following topology data set has to be used to define the mesh boundary:

[13]The orientation is derived from the sequence of the topology of the boundary. For example, the boundary edge $1, 2$ can be seen as a *directed edge* pointing from vertex 1 to vertex 2. This information is inherited and preserved in the mesh elements.

Topological Information
1, 2
2, 3
3, 4
4, 5
5, 6
6, 7
7, 8
8, 9
9, 10
10, 1

Table 3.2: The topological information used to define the boundary of a mesh domain. Note that the numbers relate to the corresponding numbers in **Figure 3.5** . It may be interpreted as a recipe how the geometric information has to be connected to obtain the boundary.

The definition of the boundary by topological information is called *planar straight line graph* (*PSLG*) and *piecewise linear complex* (*PLC*) for two and higher dimensions, respectively.

Definition 9 (Planar Straight Line Graph [1]) *A planar straight line graph is a collection of vertices and segments. Segments are edges whose endpoints are vertices in the PSLG, and whose presence in any mesh generated from the PSLG is enforced.*

In three dimensions, a generalization of the *PSLG* is used:

Definition 10 (Piecewise Linear Complex [1]) *A piecewise linear complex X is required to have the following properties:*

- *For any facet in X, every edge and vertex of the facet must appear as a segment or vertex of X.*
- *X contains both endpoints of each segment of X.*
- *X is closed under intersection.*
- *If a segment of X intersects a facet of X at more than a finite number of points, then the segment must be entirely contained in the facet.*

In general, *quality*[14] mesh generation is based on the *Delaunay* property.

Definition 11 (Delaunay Triangulation [1]) *A Delaunay triangulation of a vertex set is a triangulation of the vertex set with the property that no vertex in the vertex set falls in the interior of the circumcircle*[15] *of any triangle in the triangulation.*

Typically mesh generation engines provide two different types of *Delaunay* triangulations:

[14] Quality evaluation of mesh elements is discussed in **Section 3.5**.
[15] A *circumcircle* is a circle that passes through all three vertices of a triangle.

Definition 12 (Constrained Delaunay Triangulation [1]) *A constrained Delaunay triangulation of a PSLG is similar to a Delaunay triangulation, but each PSLG segment is present as a single edge in the triangulation. A constrained Delaunay triangulation is not truly a Delaunay triangulation. Some of its triangles might not be Delaunay*[16]*, but they are all constrained Delaunay*[17].

Definition 13 (Conforming Delaunay Triangulation [1]) *A conforming Delaunay triangulation of a PSLG is a true Delaunay triangulation in which each PSLG segment may have been subdivided into several edges by the insertion of additional vertices, called Steiner points. Steiner points are necessary to allow the segments to exist in the mesh while maintaining the Delaunay property.*

In the three dimensional case the *Delaunay* property (**Definition 11**) is a straightforward generalization of the two dimensional case. However, instead of a circumcircle a circumsphere[18] is used.

However, it is important to note, that three dimensional mesh generation is more difficult. The major difference between two an three dimensional mesh generation is, that every polygon can be triangulated without the use of *Steiner*[19] vertices. On the contrary, there are polyhedra that cannot be tetrahedrized without the use of Steiner vertices [22].

Any *convex polyhedron* can be tetrahedrized. However, for a convex polyhedron is not always possible to setup a tetrahedrization which conforms to the interior boundaries. Those interior boundaries may be facets of untetrahedrizable objects. As a consequence *constrained tetrahedrizations* do not always exist.

Consequently, three dimensional mesh generation is much more challenging than its two dimensional counterpart. This fact especially applies for *quality mesh generation*. The generation of high quality three dimensional mesh elements is one of the major challenges in mesh generation.

[16] A triangle is said to be *Delaunay* if and only if its circumcircle is empty.

[17] An edge or triangle is said to be *constrained Delaunay* if it satisfies two conditions. First, its vertices are *visible* to each other. Visibility is obstructed if a segment of X lies between two vertices. Second, there exists a circle that passes through the vertices of the edge or triangle in question, and the circle contains no vertices of X that are visible from the interior of the edge or triangle.

[18] A *circumsphere* is a sphere that passes through all four vertices of a tetrahedron.

[19] Steiner vertices (points) are vertices which are added by the mesh generation process to enable the generation of the mesh or to increase the quality of the mesh.

3.3 Cell Complex

A *cell complex* can be interpreted as a generalization of a simplex complex. Such complexes have been applied in several application cases, for example in image analysis [23]. The elements of a cell complex are called *cells* and may be interpreted as placeholders for the actual elements, i.e., simplices. Therefore, a combinatorial structure can be defined which is capable of representing arbitrary elements[20]. This fact generalizes the decomposition methods introduced so far. Consequently, the generalization of a cell complex can be used to implement generic mesh algorithms which process arbitrary elements of arbitrary dimension of a mesh.

A p-cell is introduced analogously to the k-simplex notation. However, due to topological abstraction of a cell, there is no unique relation between a p-cell and a geometrical entity for $p > 1$. The following tables depict this fact by analyzing the elements of a 2-cell complex and a 3-cell complex.

p-**cell**	**topological object**	**geometrical object**
0-cell	vertex	point
1-cell	edge	line
2-cell	cell	triangle, quadrilateral, ...

Table 3.3: Overview of the relations between an arbitrary p-cell (**left**), topological objects (**middle**) and the geometrical counterparts (**right**) for a dimension p of up to two. Note that there is only a unique relation between the geometrical entities and the topological counterparts for $p = 0, 1$. A unique relation for $p > 1$ can only be achieved by additional information called *Cell Topology* [6]

.

p-**cell**	**topological object**	**geometrical object**
0-cell	vertex	point
1-cell	edge	line
2-cell	facet	triangle, quadrilateral, ...
3-cell	cell	tetrahedron, cuboid, ...

Table 3.4: Overview of the relations between p-cell notations (**left**), topological objects (**middle**) and the geometrical counterparts (**right**) for a dimension p of three. Note that there is only a unique relation between the geometrical entities and the topological counterparts for $p = 0, 1$. A unique relation for $p > 1$ can only be achieved by additional information called *Cell Topology* [6]

.

[20] A combinatorial structure enables to operate on the elements of a cell complex. Therefore, it is possible to implement algorithms which process for example all 2-cells which are connected to a certain 0-cell.

3.3.1 Boundary Operation

Due to the combinatorial structure of a p-cell, topological operations may be introduced naturally. The *boundary operation* is a typical example and is commonly denoted by ∂_p. The application of the boundary operation on p-cells yields cells of dimension $(p-1)$.

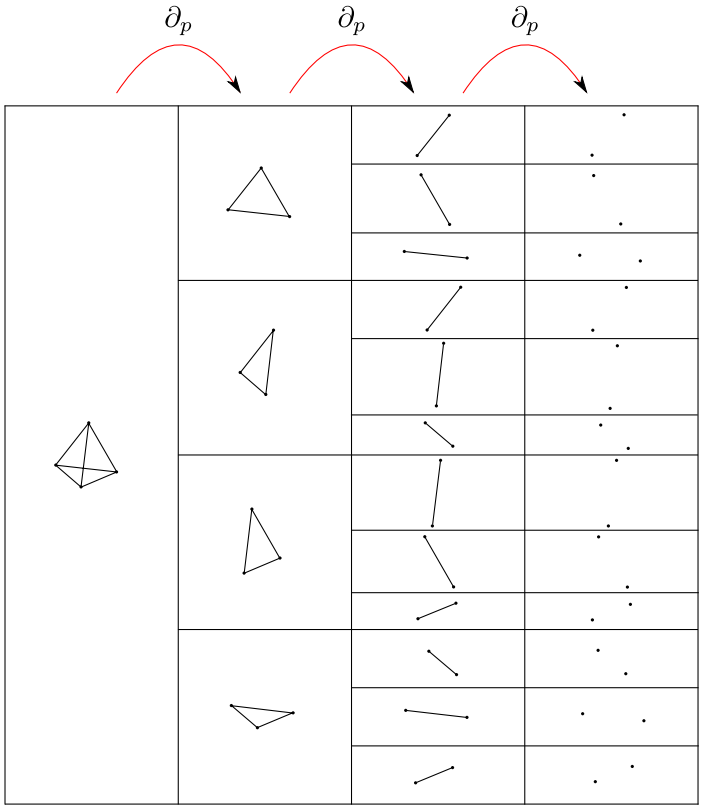


Figure 3.6: Overview of the topological decomposition of a 3-cell (simplex topology) due to the application of the boundary operator ∂_p. Note that each time the boundary operator is applied, $(p-1)$-cells are retrieved.

As a result the incident topological objects are retrieved by applying the boundary operator ∂_p. This allows to implement generic algorithms to process various parts of a p-cell.

3.4 Algorithm Generalization

This section discusses an approach to generalize typical algorithms used for mesh generation and adaption. The use of the generalized algorithms enables to implement applications which are able to process arbitrary geometrical entities of arbitrary dimensions.

3.4.1 Algorithms

The following list provides an overview of typical algorithms used for *adaptive mesh generation*.

- *Length*

  Let a, b be two points in $\mathbb{R}^n$. The algorithm returns the square-root of the sum of the squared components of the vector $\overrightarrow{ab}$; $\sqrt{\sum_{i=1}^{n}(b_i - a_i)^2}$

- *Orientation 2D*

  Let a, b, c be three points in $\mathbb{R}^2$. The algorithm returns a positive value if c is on the left of the directed line ab[21]. A negative value is returned if the point is on the right and zero if the point is collinear with the line. Note that the signed value is also an approximation of two times the area of a triangle defined by the points a, b, c. Due to the fact that the value is signed, it is also referred to as a *signed area* [1].

- *Orientation 3D*

  Let a, b, c, d be four points in $\mathbb{R}^3$. The algorithm returns a positive value if d is below the oriented plane[22] passing through a, b, c. Note that the signed value is also an approximation of six times the volume of a tetrahedron defined by the points a, b, c, d. Due to the fact that the value is signed, it is also referred to as *signed volume* [1].

- *Point in triangle*

  Let a, b, c, d be four points in $\mathbb{R}^2$. A positive value is returned if d is located inside the triangle a, b, c.

- *Point in tetrahedron*

  Let a, b, c, d, e be five points in $\mathbb{R}^3$. A positive value is returned if e is located inside the tetrahedron a, b, c, d.

- *Area to length ratio*

  A numerical quality measurement value for triangles, denoted A/l_{rms}^2. It returns the signed area of the triangle divided by the square of the root-mean-squared edge length [24].

- *Volume to length ratio*

  A numerical quality measurement value for tetrahedrons, denoted V/l_{rms}^3. It returns the signed volume of the tetrahedron divided by the cube of the root-mean-squared edge length [24].

[21] A directed line ab represents a connection from a to b

[22] The orientation of the plane is defined by the points a, b, c, which has to counterclockwise when viewed from above the plane.

3.4.2 Generalization

An in depth investigation of the algorithms presented in section **3.4.1** regarding the underlying geometrical objects and dependencies of one to another reveals that many algorithms can be extracted and grouped together.

algorithm	**preconditioned algorithm**
Area to length ratio	*Area, Length*
Volume to length ratio	*Volume, Length*
Point in triangle	*Orientation2D*
Point in tetrahedron	*Orientation3D*

Table 3.5: The dependencies of typical mesh adaption/generation algorithms. Note that each name of the algorithm is somehow related to the object to which the algorithm is supposed to be applied.

A general formulation of algorithms regarding dimension and cell topology can be achieved by extracting the underlying group of the algorithm.

algorithm	**generalized algorithm**
Orientation2D	*Cell orientation*
Orientation3D	*Cell orientation*
Length	*Metric quantity*
Area	*Metric quantity*
Volume	*Metric quantity*
Area length ratio	*Cell quality*
Volume length ratio	*Cell quality*
Point in triangle	p*-cell in* q*-cell*
Point in tetrahedron	p*-cell in* q*-cell*

Table 3.6: Overview of the relation between algorithms and their generalized counterparts. Note that the generalized algorithms lack information about the dimension and the associated geometrical entity. Analogously to the p-cell notation a unique relation can only be established by providing the dimension and the cell topology.

It is important to note that the generalized algorithms are based on cells. Hence, by using the p-cell notation the algorithms can be associated with a topological object of a certain dimension. A unique mapping from the generalization to the specific algorithm can be established by additionally providing the cell topology.

This fact leads to an important conclusion:

Conclusion *A geometrical algorithm can be abstracted into a generalized algorithm, if the cell topology and the cell dimension are extracted.*

Figure 3.7 illustrates this conclusion.

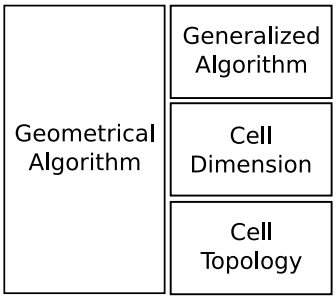


Figure 3.7: A geometrical algorithm can be separated into a generalized algorithm, cell dimension and cell topology.

The following table provides typical examples for such algorithm generalizations.

generalized algorithm	**cell dimension**	**cell topology**	**geometrical algorithm**
Metric quantity	1	simplex / cube	length of a line
Metric quantity	2	simplex	area of a triangle
Metric quantity	2	cube	area of a rectangle
Metric quantity	3	simplex	volume of a tetrahedron
Metric quantity	3	cube	volume of a cuboid
Cell orientation	2	simplex	2D orientation of a triangle
Cell orientation	3	simplex	3D orientation of a tetrahedron
Cell quality	2	simplex	area-length-ratio of a triangle
Cell quality	3	simplex	volume-length-ratio of a tetrahedron

Table 3.7: Examples of typical applications of the generalization procedure.

For algorithms which process two cells, the generalization concept still applies, however, the cell dimensions of both cells have to be separated to retrieve the generalized algorithm.

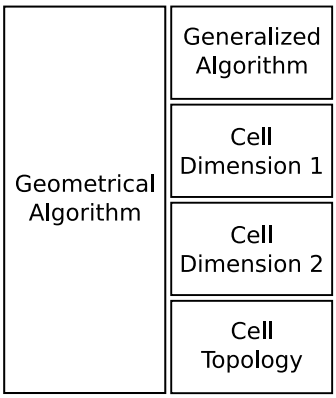


Figure 3.8: A geometrical algorithm which processes two cells can be separated into a generalized algorithm, two cell dimensions (one for each cell) and the cell topology.

An example for an algorithm generalization based on two cells is provided in the following.

generalized algorithm	dimension 1	dimension 2	cell topology	geometrical algorithm
p-*cell in* q-*cell*	0	2	simplex	point-in-triangle
p-*cell in* q-*cell*	0	3	simplex	point-in-tetrahedron

Table 3.8: Derivation of geometrical algorithms based on generalized algorithms, dimensions and cell topologies. Note that dimension 1 relates to the dimension of first p-cell, and dimension 2 relates to the dimension of the second p-cell. The generalized algorithm in line 1 reads: 0-cell-in-2-cell. The unique relation to the point-in-triangle algorithm can be established using the simplex cell topology.

3.4.3 Boundary Operation

By coupling the generalized algorithms with topological operations such as the boundary operation (**Section 3.3.1**) the algorithms can be applied to the results of the boundary operations.

The boundary operation retrieves all $(p - n)$-cells of a p-cell, where p is the dimension of the base cell and n is the dimension of the boundary operation. The following table depicts this operation for typical applications:

dimension p	dimension n	cell result
2	0	2-cell
2	1	1-cell
2	2	0-cell
3	0	3-cell
3	1	2-cell
3	2	1-cell
3	3	0-cell

Table 3.9: Overview of the results of the boundary operator applied on a 2-cell (**top**) and on a 3-cell (**bottom**).

The generalization procedure (**section 3.4.2**) can be extended to deal with boundary operations.

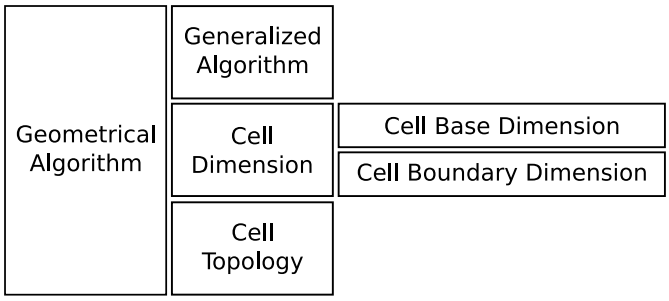


Figure 3.9: A geometrical algorithm which processes one cell can be separated into a generalized algorithm, cell dimension and cell topology. The initial approach introduced in **Figure 3.7** is extended by the derivation of the cell dimension. The cell dimension is computed by subtracting the cell boundary dimension from the cell base dimension as depicted in **Table 3.9**.

Extended examples based on the *metric quantity* algorithm discussed in **Table 3.7** are provided in the following.

generalized algorithm	**dimension p**	**dimension n**	**geometrical algorithm**
Metric quantity	2	0	area of a triangle
Metric quantity	2	1	lengths of the edges of a triangle
Metric quantity	2	2	not valid: a point does not have a size
Metric quantity	3	0	volume of a tetrahedron
Metric quantity	3	1	areas of the faces of a tetrahedron
Metric quantity	3	2	lengths of the edges of a tetrahedron
Metric quantity	3	3	not valid: a point does not have a size

Table 3.10: Overview of application cases based on the boundary operation and the *cell metric* algorithm. Note that the cell topology is omitted. Therefore, the cell is considered to have a simplex topology. Of course this concept can be applied to other cell topologies, i.e. cubes, as well.

The derivation of the cell dimension, based on the boundary operation, can be extended to more complex algorithms as well.

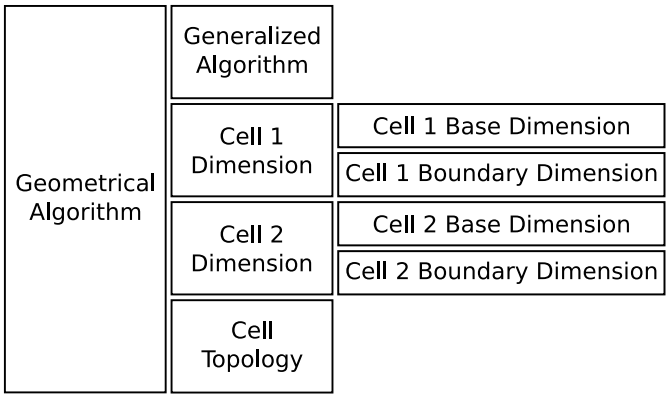


Figure 3.10: A geometrical algorithm which processes two cells can be separated into a generalized algorithm, two cell dimensions (one for each cell) and the cell topology. The initial approach introduced in **Figure 3.8** is extended by the derivations of the cell dimensions. The cell dimensions are computed by subtracting the corresponding cell boundary dimensions from the related cell base dimensions as depicted in **Table 3.9**.

Extended examples based on the *p-cell in q-cell* algorithm discussed in **Table 3.8** are provided in the following.

generalized algorithm	cell 1 dimension		cell 2 dimension		geometrical algorithm
	p	n	q	n	
p-*cell in* q-*cell*	3	2	3	1	edge in triangle
p-*cell in* q-*cell*	2	0	3	0	triangle in tetrahedron

Table 3.11: Overview of application cases based on the boundary operation and the p-*cell in* q-*cell* algorithm. Note that the cell topology is omitted. Therefore the cell is considered to have a simplex topology. This algorithm processes two cells, therefore the boundary operation has to be applied on both cells. Investigation of the depicted example in line 1: The cell 1 dimension result is $(p - n) = (3 - 2) = 1$-cell. A 1-cell in combination with a simplex cell topology yields an edge. Analog for the second cell $(q - n) = (3 - 1) = 2$-cell. A 2-cell with a simplex cell topology yields a triangle. The final decoding yields: *edge in triangle*. However, as each result of the boundary operations is processed, the decoding actually yields in own words: Test each 1-cell of the source 3-cell if it is inside of one of the 2-cells of the target 3-cell.

3.5 Mesh Quality

Mesh generation and adaption tools need to evaluate the quality of an element of a mesh and the quality of a mesh in total. Having a measure for quality at hand, mesh adaption algorithms can be applied which aim to increase the mesh quality. Therefore, this section clarifies what a *bad* element actually is. The presented evaluation of the quality of an element summarizes related parts of detailed research in the field of *Delaunay mesh generation* [1][24]. The approach is extended to sets of elements to enable quality evaluation of whole meshes.

Interpolation on a triangular or a hexahedral mesh relates to the approximation of some *true* function by an *interpolating* function. As this is a rather important task in scientific computing, the possible interpolation errors deserve a more detailed investigation. The following provides a derivation of the two different types of interpolation errors.

Let T be a triangular or tetrahedral mesh, and let $f(p)$ be a continuous scalar function defined on the mesh. Let $g(p)$ be a piecewise linear approximation to $f(p)$, where $g(v) = f(v)$ at each vertex v of T, and $g(p)$ is linear over any single element of T.

The norm

$$||f - g||_\infty$$

corresponds to the maximum pointwise interpolation error over the element t,

$$max_{p \in t}|f(p) - g(p)|.$$

The norm

$$||\nabla f - \nabla g||_\infty$$

corresponds to the maximum of the pointwise error in the interpolated gradient of the element t,

$$max_{p \in t}|\nabla f(p) - \nabla g(p)|.$$

An investigation of the error bounds of the norms reveals that the interpolation error of $||f - g||_\infty$ and $||\nabla f - \nabla g||_\infty$ can be minimized by reducing the size of the element. However, the error of $||\nabla f - \nabla g||_\infty$ can be further decreased by improving the shape of the element.

Due to the additional consideration of the element's shape the $||\nabla f - \nabla g||_\infty$ norm can be considered superior to the $||f - g||_\infty$ norm with respect to the evaluation of element quality.

The element's shape is determined by the angles of the triangle or the dihedral angles of the tetrahedron. In general an element should be as *round*[23] as possible, as large and small angles can degrade the quality of numerical solutions.

Large angles may cause large discretization errors, which as a consequence impairs the accuracy of solutions obtained by numerical methods, such as the finite element method.

Furthermore large angles cause large errors in the derivatives of solutions, the previously mentioned *interpolation errors*. This is due to the fact that ∇g becomes arbitrarily large (**Figure 3.11**).

Small angles on the other hand can cause a system of equations[24] to be *ill-conditioned*. Therefore, the accuracy of the solution of the equation system is again decreased if a solution is obtained at all.

Due to the balance of angles in triangles and dihedral angles in tetrahedra a bounding on the smallest angle also results in bounds for the largest angle. For example, in two dimensions, if no angle is smaller than θ, then no angle is larger than $180° - 2\theta$. Typically mesh generation algorithms aim to bound the smallest angle and therefore simultaneously bound the largest angle as well.

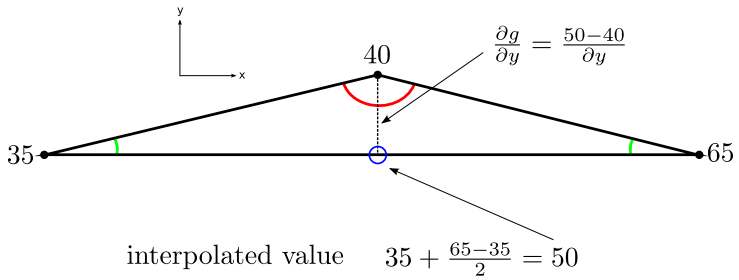


Figure 3.11: Influence of large angles of a 2-simplex on ∇g. Note the quantities associated with each 0-cell. At the middle of the bottom $1 - cell$ the interpolated value is 50. Computing $\frac{\partial g}{\partial y}$ reveals that the partial derivative tends to infinity if the **red** angle reaches $180°$. Note that a 3-simplex shares this behavior, but instead of investigating the angles between the 1-simplices the dihedral angles between the 2-simplices have to be investigated.

Typically, mesh generation or adaption tools base the evaluation of quality on one single numerical value. Several different approaches to compute such values have already been investigated [24] [4]. In the following the *volume/area to length ratio* is used to compute the quality of a tetrahedron or triangle respectively. The reason for using these ratios is due to the fact that this quality measure is smooth[25] and reasonably penalizes degenerated elements (skinny needles).

To evaluate the quality of whole meshes, the assessment of quality has to be extended from single elements to a set of elements. It is important to note that the *worst* elements have far more influence on numerical simulations than elements of average quality [4]. Consequently it is reasonable to investigate the worst occurring element quality of a mesh. Based on this investigation, adaption techniques can be investigated regarding their effectiveness. Furthermore, iterative adaption techniques can be applied which are guided by the worst occurring element quality.

[23] A perfectly round element would be an equilateral triangle or tetrahedron respectively.

[24] A system of equations is typically obtained by numerical methods.

[25] A *smooth* quality measure is *scale-invariant*, meaning that scaling a triangle/tetrahedron does not influence the evaluated quality value.

Chapter 4

Toolkit

This chapter introduces implementations used in conjunction with the applications introduced in **Section 5**. A mesh hull extraction tool is introduced, as well as a mesh classification tool and a generic mesh generation interface. Furthermore an approach to concatenate algorithms to complex tasks is presented.

Section 4.1 describes a hull extraction tool which enables the derivation of hull meshes from volume meshes.

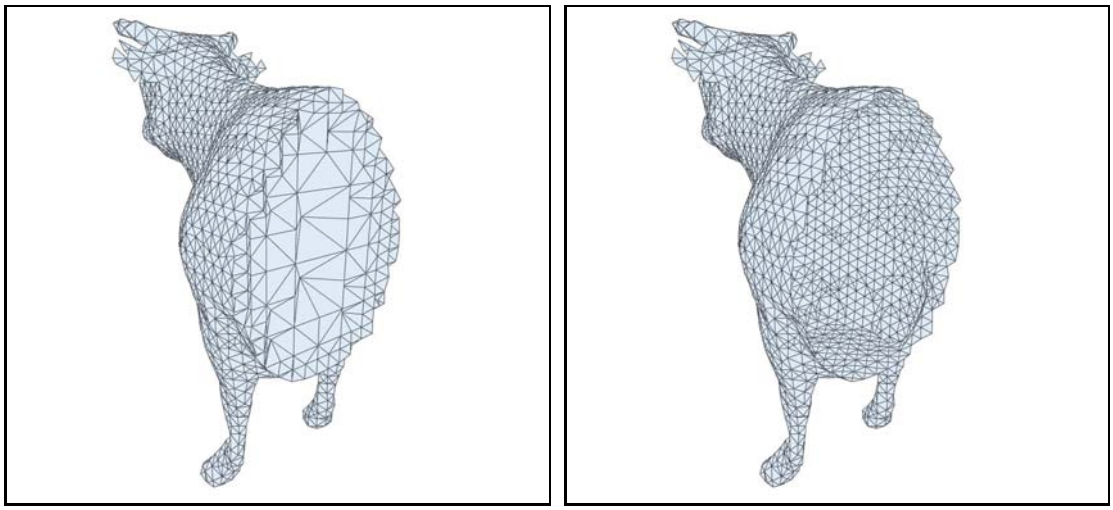
Section 4.2 introduces a mesh qualification application. Several different forms of degeneration of elements as well as optimal shapes of elements are identified, thereby providing a quality evaluation of meshes.

Section 4.3 discusses the implementation of a generic mesh generation interface. Different mesh engines can be accessed using a generic interface. This approach enables the implementation of applications which are decoupled from the actual mesh generation engine. Therefore, different mesh engines can be accessed in a consistent manner.

Section 4.4 describes an approach to combine algorithms encapsulated in *C++ function objects* together building a sequence of function objects. The sequence is executed, so that the result of the first algorithm is forwarded to the second algorithm acting as input and so forth. This approach allows to setup complex tasks based on concatenating separately implemented algorithm modules.

4.1 Hull extractor

The *hull extractor* tool enables the derivation of hull meshes from volume meshes. The generated hull meshes can be used as an input for three dimensional mesh generation engines. Therefore volume meshes can be rebuild by using different mesh generation engines.



Volume mesh Hull mesh

Figure 4.1: Visualization of the functionality of the *hull extractor* tool. **left:** The input volume mesh is shown. Note that the interior elements are larger in size than the boundary elements. **right:** The derived hull mesh is shown. Note that a halfspace has been set to be invisible. Therefore a view of the interior of the meshes is provided.

The key part of the implementation is the determination of the boundary of the volume mesh. This determination makes use of the the combinatorial structure of a *cell complex* (**Section 3.3**). The goal is to determine all 2-cells of the input 3-cell complex which are on the boundary. A 2-cell is on the boundary, if it is part of exactly one 3-cell[1].

The described operation relates to the application of the so called *boundary operator*. The boundary of a volume Ω is a *closed surface* $\partial\Omega$ which relates to the hull mesh[2].

[1] This procedure can be generalized to arbitrary cell complex dimensions. For a p-cell complex, a $(p-1)$-cell is on the boundary, if it is part of exactly one p-cell.

[2] Note that the boundary of such a boundary is zero, $\partial\partial\Omega = 0$, as a *closed* surface does not have a boundary.

4.2 Mesh Classification

To be able to evaluate the quality of a mesh, a mesh classification tool for 3-simplicial complexes has been extended to additionally classify 2-simplicial complexes [18].

The following figure depicts the three different classification types of a 2-simplex.

Figure 4.2: Classification of different types of 2-simplices. **left:** Two of the three angles of a *blade* are small. The edges are more ore less of equal length. A blade is considered *degenerate*. **middle:** A *dagger* has one considerably shorter edge compared to the other two edges. A dagger is considered *degenerate*. **right:** A *round* has edges and angles of nearly the same size. A round is considered *optimal*.

The following figures depict the evaluation capabilities of this application.

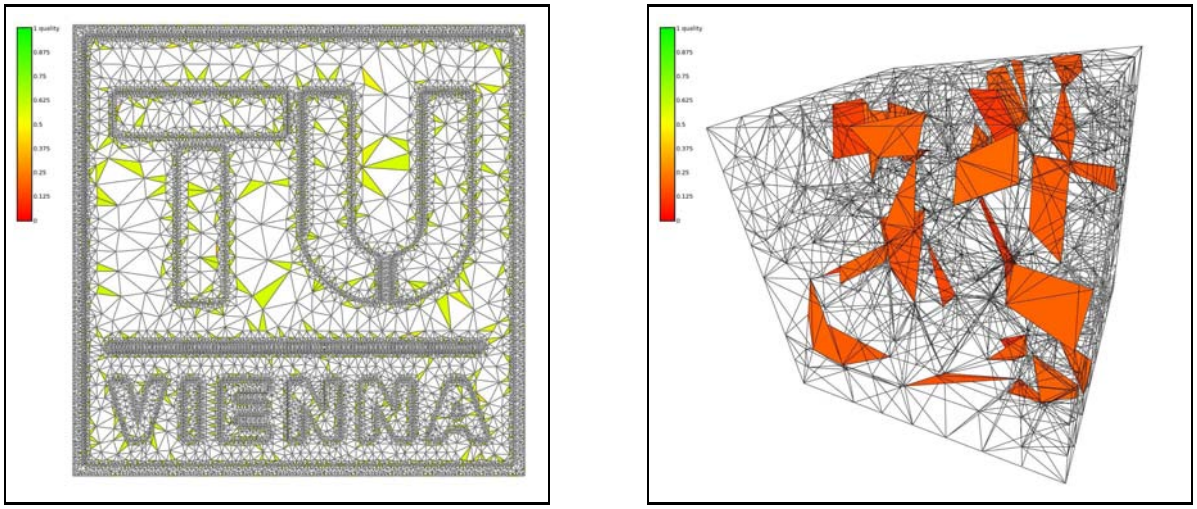


Figure 4.3: Element quality visualization of an exemplary 2-simplex and 3-simplex mesh. **left:** A 2-simplex mesh of the TU VIENNA LOGO is investigated. Elements with quality of up to 0.6 are highlighted. **right:** A 3-simplex mesh of a CUBE is analyzed. Elements with quality of up to 0.2 are highlighted.

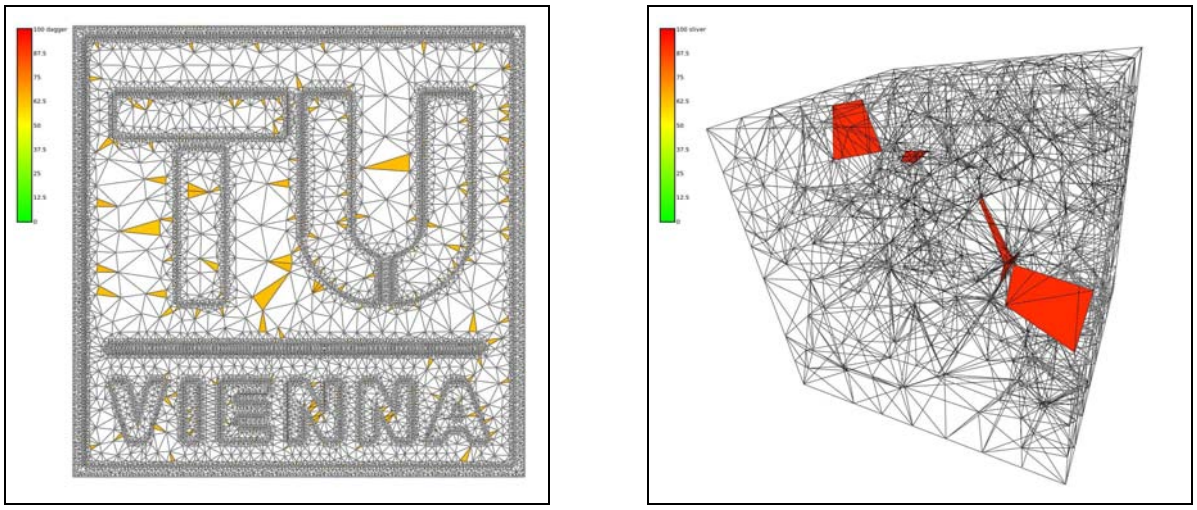


Figure 4.4: Visualization of *degenerated* elements of an exemplary 2-simplex and 3-simplex mesh. **left:** A 2-simplex mesh of the TU VIENNA LOGO is investigated. *Dagger* triangles of 60% and above are highlighted. **right:** A 3-simplex mesh of a CUBE is analyzed. Degenerated *sliver* tetrahedrons of 90% and above are highlighted.

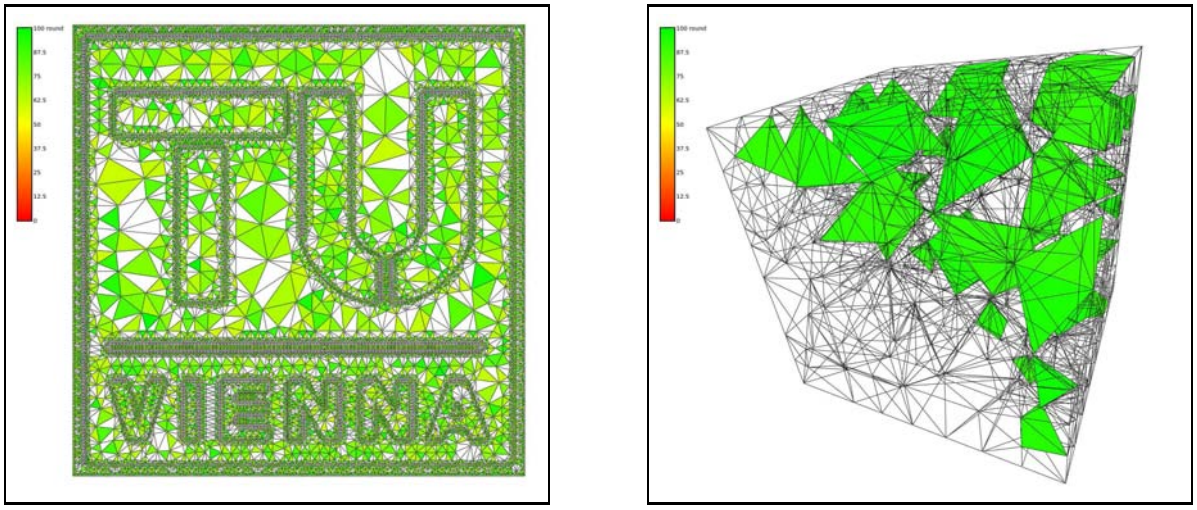


Figure 4.5: Visualization of *optimal* elements of an exemplary 2-simplex and 3-simplex mesh. **left:** A 2-simplex mesh of the TU VIENNA LOGO is investigated. *Round* triangles of 60% and above are highlighted. **right:** A 3-simplex mesh of a CUBE is analyzed. *Round* tetrahedrons of 90% and above are highlighted.

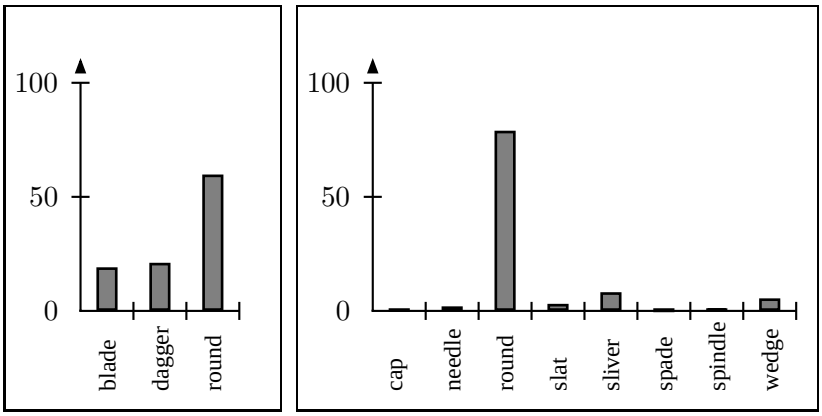


Figure 4.6: Overview of the distribution of classified elements in percent of an exemplary 2-simplex and 3-simplex mesh. **left:** The classification of the TU VIENNA LOGO. Note that the optimal *round* elements hold a fraction of more than 50%. **right:** The classification of the CUBE mesh. There is a vast majority of optimal *round* tetrahedrons. Degenerated *sliver* and *wedge* elements are both below 10%.

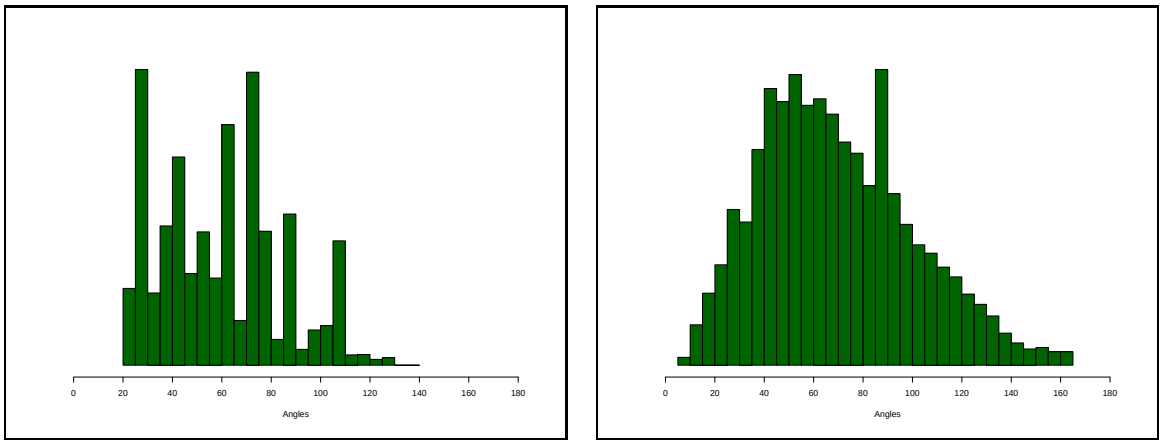


Figure 4.7: Distribution of angles and dihedral angles respectively of an exemplary 2-simplex and 3-simplex mesh. **left:** The angle distribution of the TU VIENNA LOGO. Note, that there are no angles smaller than $20°$, and only very few greater than $120°$. **right:** The distribution of dihedral angles of the CUBE mesh is presented. Note that the majority of dihedral angles is greater than $20°$ and smaller than $140°$. However, a few angles appear outside of this region, which indicates very bad elements.

4.3 Generic Mesh Generator Interface

Several different mesh generation tools are available for public use [11][25][10]. Each tool offers a different programming interface. To be able to use different mesh engines in a consistent manner a generic interface has been implemented.

As stated in **Section 3.2.2** the boundary of the mesh has to be defined. Therefore, a mesh generation application has to process two different types of input data[3]:

- **Geometric Information**

  The geometric information describes the position of the parts of the mesh, for example the points, in space[4].

- **Topological Information**

  The topological information defines the connection and orientation of the geometric information to setup the boundary of the mesh domain.

Typically the dimensions of the geometric and the topological space of the resulting mesh are of the same size. For example, volume mesh engines may produce 3-simplex mesh elements which are embedded in $\mathbb{R}^3$.

However, mesh generation engines are available which generate so called *hull meshes* [10]. In this case the dimension of the topological space of the output mesh is smaller than the dimension of the geometric space. For example, a hull mesh engine may generate 2-simplex mesh elements which are embedded in $\mathbb{R}^3$.

Therefore the generic interface supports arbitrary dimensions of the topology and the geometric space.

The following code snippet depicts a meta-function[5] which declares a default mesh engine for a certain topology dimension.

```
1 typedef typename result_of::mesher< TopologyDimension >::type Mesher;
```

Specialized mesh engines can be used by adding additional tags.

```
1 typedef typename result_of::mesher< TopologyDimension, tag_cgal >::type Mesher;
```

[3] Note that for triangulations only geometrical information is required, as the convex hull is defined on a point set.
[4] Typically the space is of an n-dimensional euclidian type, $\mathbb{R}^n$.
[5] A C++ meta-function is evaluated by the compiler at compile time.

A convenient input method has been implemented which is able to automatically differentiate between geometry, topology and property input data:

```
1  //
2  // a mesher object is instantiated
3  //
4  Mesher mesher;
5  //
6  // adding geometry information
7  //
8  metric_object< double, 2 >    point( 1.0, 2.0 );
9  add( point )( mesher );
10 //
11 // adding topology information
12 //
13 metric_object< long, 2 >      cell( 0, 1 );
14 add( cell )( mesher );
15 //
16 // adding a mesh property
17 //
18 add( conforming_delaunay )(mesher);
```

Note the unchanged syntax of the *add*-function (**Lines 9, 14, 18**) even though different types of information is passed to the mesher object.

The mesh process can be initiated by

```
1  start( mesher );
```

After the mesh process is finished, the result can be accessed by

```
1  geometry( mesher );
2  topology( mesher );
```

Note that the access to the results returns container references which are directly accessible, for example:

```
1  for_each( geometry( mesher ), std::cout << _1 << std::endl );
```

Note that above code line utilizes a specialized *for_each* function which is based on *Boost Range* [7]. Further note the use of a *Lambda placeholder*, _1, which is part of the *Boost Phoenix* library.

The following table provides an overview of the currently and prospectively available mesh engines.

Mesh Engine	**2D**	**3D**
Triangle [25]	**x**	
TetGen [11]		**x**
CGAL [10]	**x**	**o**

Table 4.1: Overview of currently and prospectively available mesh engines. Note that **x** denotes availability, **o** denotes under development.

4.4 Generic Functor Queue

This section introduces a generic approach to combine different algorithms to an algorithm sequence. The execution of the sequence applies a *fold* operation. Therefore the result of the first algorithm is forwarded as an input to the second algorithm and so forth.

To enable functional programming the algorithms are accessible by *functors*. In computer programming a functor is understood to be a *function object*. A function object is an object which behaves like a function but may optionally have a state. The following code snippet depicts an example implementation of a unary function object[6].

```
1 struct Increment {
2    template < typename T >
3    void operator()( T & t ) {
4       t++;
5    }
6 };
```

A typical use case would be:

```
1 for_each( container, Increment() );
```

Note that the *Constructor* of the *increment* object is called once prior to further processing. The *for_each* function traverses the container, and passes each element of the container to the *operator()* of the increment object. The functor operates on references of the elements, therefore incrementing each element of the container.

More complex tasks, for example *mesh adaption*, require a set of different algorithms applied on a dataset. Therefore, a functor sequence can be used to concatenate these algorithms together. A possible processing queue[7] may look like this:

$$Element \rightarrow^{(double)} Increment \rightarrow^{(double)} LessThan \rightarrow^{(bool)} Result$$

Note the input data types at each stage of this example queue. This example outlines an important property of such queues: The data type may change.

To show an example of such a queue the following functors may be implemented:

```
1 template < typename T >
2 struct Increment {
3    typedef T result_type;
4    result_type operator()( T const& t ) {
5       return (t+1);
6    }
7 };
```

Note the result type of the functor has to be present at the time of object instantiation (**Lines 1, 3, 4**). This fact can be considered as a drawback, as an instance of *Increment* can only operate on elements of type *T*. In contrast to the previous implementation of *Increment*. Further note that the result is returned, and the actual element remains unchanged (**Lines 4, 5**).

[6] A unary operation is an operation with only one operand. A binary operation is an operation with two operands.
[7] Queue in the sense, that the result of the first functor is forwarded to the second functor and so on.

```
template < typename T >
struct LessThan {
   typedef bool result_type;
   LessThan( T threshold ) : _threshold(threshold) {}
   result_type operator()( T const& t ) {
      if( t < _threshold ) return true;
      else                 return false;
   }
   double _threshold;
};
```

Note that the *LessThan* functor offers a state which provides the threshold level.

A *Boost Fusion* sequence is used to store instances of the function objects [7]:

```
typedef typename sequence< Increment, LessThan >::type      Queue;
Queue queue( increment, less_than );
```

Note that a C++ metafunction is used to evaluate the actual *Boost Fusion* sequence. This approach enables extensibility and maintainability, as the interface (the wrapper) remains unchanged whereas the internal container implementation may change, i.e., another sequence may be used.

A *fold* algorithm has been implemented to process functor sequences[8].

```
for_each( container, std::cout << fold( queue, _1 ) << std::endl );
```

Each element of the container is extracted and forwarded to the processing queue.

Such kind of sequences are able to store differing types of data[9]. Therefore, different functors can be stored, which consequently allows to implement arbitrary functor queues.

This fact concludes that above code line is able to process arbitrary functor queues.

The presented example may also be conveniently implemented suing *Boost Phoenix* [7].

```
for_each( container,
      if_( (_1 + 1) < ref(threshold) )
         [ std::cout << val(true) << std::endl ]
      .else_[ std::cout << val(false) << std::endl ]
);
```

Note that the keywords *ref* and *val* are specialized functions which relate to *reference* and *value* respectively. These keywords enable to access data which is present outside the *Phoenix* expression. As the complexity of the underlying algorithms and of the functors increases, the complexity of the *Phoenix* expressions increases dramatically. Such an increase of complexity decreases readability and consequently maintainability.

[8] The implementation is based on the *Boost Fusion* implementation, however, a specialized version has been implemented to support different return types.

[9] Containers which are able to store differing datatypes are called *heterogeneous containers*.

Chapter 5

Applications and Results

This chapter introduces adaptive mesh generation applications with a focus on modern programming techniques and generalized algorithms. Key parts of the application implementations are discussed in detail to outline the internal mechanisms. The goal is to describe the functionality of the implemented algorithms and to investigate the advantages and disadvantages of the implementations. The applications should act as prototypes to investigate the applicability of generalized algorithms in the field of adaptive mesh generation. The main focus is on abstract formulations of mesh adaption algorithms. New workflows are made available by supporting arbitrary combinations of separately implemented algorithm modules.

Section 5.1 introduces a mesh adaption application which implements a simplification algorithm which is controlled by the evaluation of image data. The underlying algorithm is described and the functionality is shown.

Section 5.3 outlines a mesh adaption algorithm which refines a mesh with respect to degenerated elements and discusses an implemented application.

Section 5.2 outlines the implementation of a mesh adaption application which performs inclusion tests of arbitrary cells. More specifically, two input meshes are processed and merged removing all intersecting cells. The key parts of the implementation are discussed in detail.

5.1 Mesh Adaption based on Image Data

This section introduces an implementation which is able to convert arbitrary images into meshes. A direct conversion approach yields meshes of tremendous sizes, as each pixel is converted into two triangles. The goal is to simplify the mesh by evaluating the variation of the image data. Regions of solid colors should be removed, whereas regions of high color variation should be conserved.

The first part outlines implementations details, presenting and discussing several code snippets. The second part introduces the input images. The third part discusses the adaption results. The final part sums up the achieved results and, as a conclusion, discusses possible improvements.

5.1.1 Implementation

The application is implemented on top of an already implemented converter application which is capable of converting *png* files to a processable mesh file[1].

The implementation assembles the *generic mesh generator interface* (**Section 4.3**) and the *generic functor queue* (**Section 4.4**).

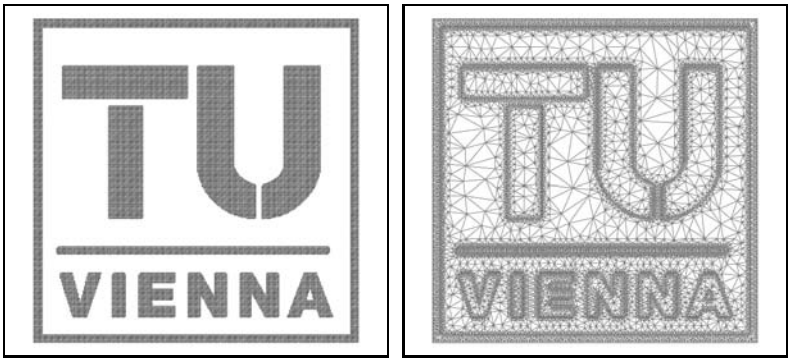
The tool evaluates *RGB-Alpha*[2] data. The challenge is to reduce the size of the mesh[3] as much as possible, without sacrificing details. The following figure depicts a basic example of the conversion process.

[1] The implementation of the image converter tool is based on the *generic image library* (*GIL*) which is part of the *Boost* libraries [7].

[2] Each vertex is associated with a quantity vector which contains the red, green, blue and alpha values.

[3] The size of a mesh can be related to the number of vertices, or any other size of an occurring element in the mesh.

Input image



Raw mesh Simplified mesh

Figure 5.1: The workflow of the image data based mesh simplification. **top:** The input image is the logo of the *Vienna University of Technology*. **bottom left:** The mesh after direct conversion from the image to a mesh. The number of points is 32526. Note that each pixel is converted into two triangles. Further note, that the background layer has been ignored for the conversion process. Therefore the boundaries are preserved precisely. **bottom right:** The simplified mesh has a point size of 12395. Note, that the mesh free regions of the raw mesh have been triangulated to setup one single connected mesh. Further note the decrease of the point density in the homogeneous regions, for example the letters.

The following figure depicts the principle of the simplification scheme.

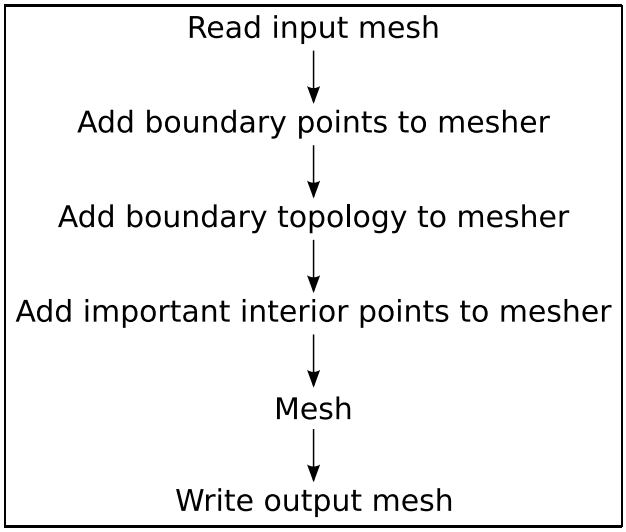


Figure 5.2: Basic principle of the implemented mesh simplification based on image data.

Note that the determination of the *important interior points* is the core of the implementation. This part is

based on an initial evaluation and an iterative postprocessing part which incrementally improves the initial evaluation.

The initial evaluation is based on two parts:

- The direct vicinity[4] of each interior 0-cell is evaluated if the *RGBA* values fall within some predefined interval. This first step allows to specifically enable or disable certain *RGBA* values in the sense that they are considered or ignored for the meshing process. Consequently, certain layers can be ignored. A typical example would be to skip the color of the background.
- The direct vicinity of each interior 0-cell is evaluated regarding the variation of the *RGBA* data. If the values of one of the channels change considerably (are larger than a threshold), the 0-cell is considered to be important.

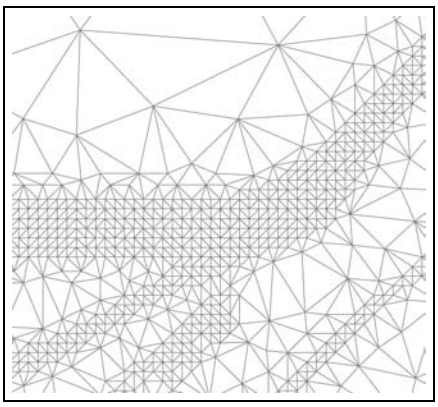
This initial evaluation already produces a considerable simplification of the mesh. However, color transitions result in unnecessary dense mesh regions. Therefore a postprocessing and iterative improvement algorithm has been developed.

The postprocessing part investigates 0-simplices which are tagged for preservation by the initial evaluation. The algorithm revokes the initial tags in a way that the thick mesh regions are thinned out symmetrically. This means, that the thick mesh regions are thinned from both sides. Optimally, this should result in a single line which should be situated in the center of the original thick mesh region.

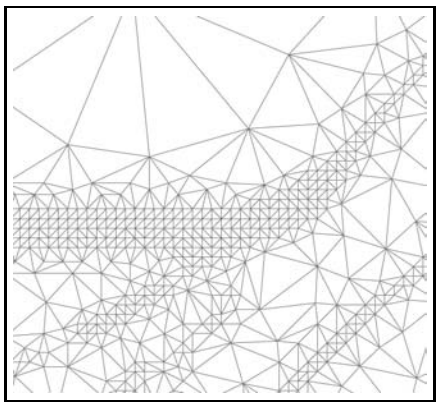
Figure 5.3 illustrates the behavior of the incremental postprocessing algorithm.

[4] The *direct vicinity* relates to the adjacent 0-cells.

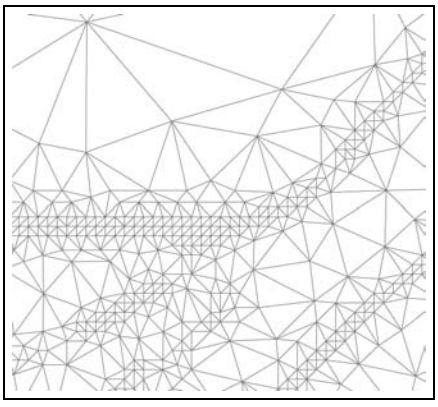
Detailed view on a typical changeover



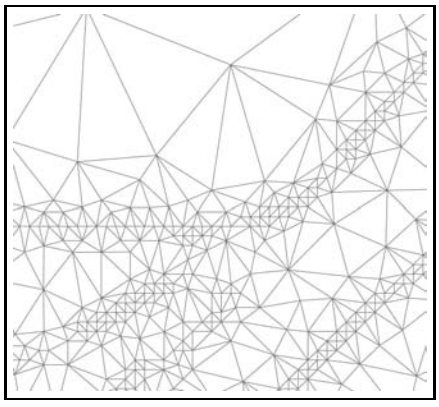
Initial evaluation



Iteration 1



Iteration 2

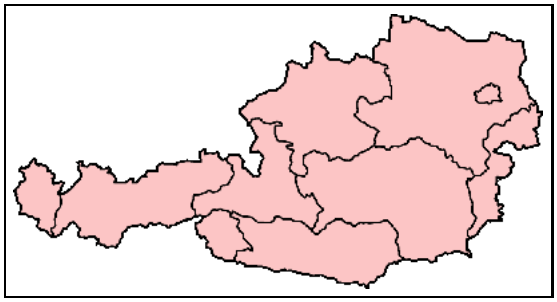


Iteration 3

Figure 5.3: Visualization of the incremental postprocessing algorithm. **top:** A detailed view of an image with a thick changeover. **middle left:** The simplification result based on the initial evaluation. Note that the grey regions of the input image result in a thick, homogeneous mesh region. **middle right:** One iteration of the thinning algorithm is applied. Note that the homogeneous mesh region shrinks quite symmetrically. **bottom left:** Two iterations of the postprocessing algorithm are applied. **bottom right:** After three iterations the initial thick changeover is nearly reduced to a line. Note that thin mesh regions of the initial evaluation are more or less preserved.

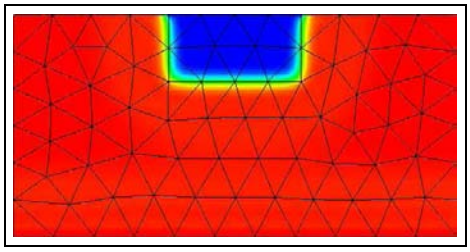
5.1.2 Input Data

In the following the input images for the mesh adaption application are introduced, providing sources and possible licenses.



AUSTRIA

Figure 5.4: Image of a map of *Austria*.

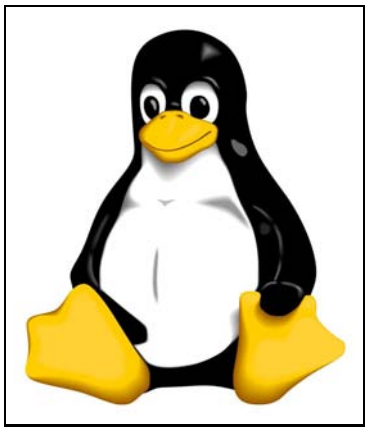


DEVICE

Figure 5.5: Image of a doping concentration within an electronic device (*MOSFET*). Note that the image has been recorded from an existing mesh, therefore the 2-simplices of the mesh are visible.

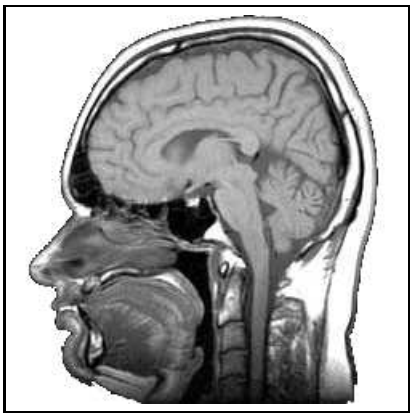
TU VIENNA LOGO

Figure 5.6: The logo of the *Vienna University of Technology*.



TUX

Figure 5.7: An image of the official mascot of the the *Linux* operating system named *Tux*.



HEAD

Figure 5.8: A *Magnetic Resonance Image* (*MRI*) of a human head.

KNEE

Figure 5.9: A *Magnetic Resonance Image* (*MRI*) of a human knee.

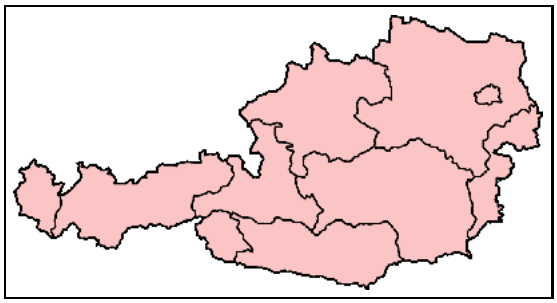
The sources of the individual images:

- DEVICE comes courtesy of René Heinzl [26].
- TU VIENNA LOGO is available online [27].
- TUX has been created by *Larry Ewing*, *Simon Budig* and *Anja Gerwinsk*. The image is available online [28].
- AUSTRIA, HEAD and KNEE are licensed [29] and are available online [28]. Note that the AUSTRIA image has been slightly altered, as the labels of the federal states have been removed.

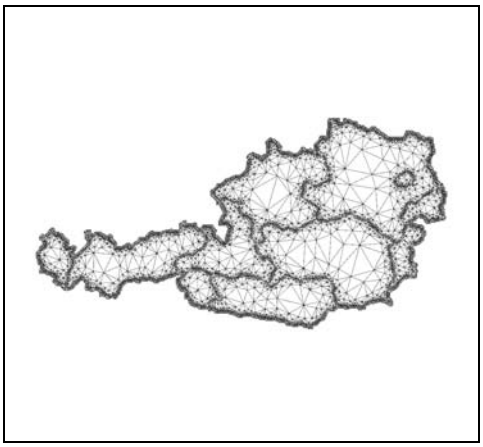
5.1.3 Mesh adaptions

This section presents and discusses several mesh simplification results for the given images. The influence of different threshold values on the preservation of image detail is outlined, where the threshold values relate to the permitted variation of the color value components (red, green, blue or alpha in percent) in the range of 8-bit color data. This means that if one of the four channels exceeds the allowed variation it is considered *important* and is therefore preserved. To provide a reasonable value range the 8-bit range is converted to the corresponding decimal range, which yields 256. Therefore, a threshold value of 4%, for example, yields 10.24[5]. Finally a 0-cell is tested by investigating the variation towards an adjacent 0-cell. If the variation exceeds the threshold, the central 0-cell is preserved.

[5] Although the decimal color range is integer based, the decision process relies on floating point values.

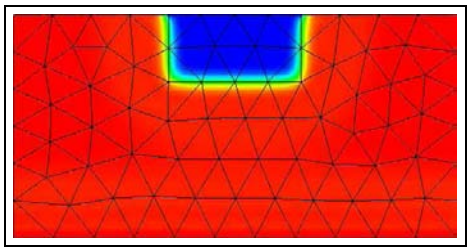


AUSTRIA IMAGE

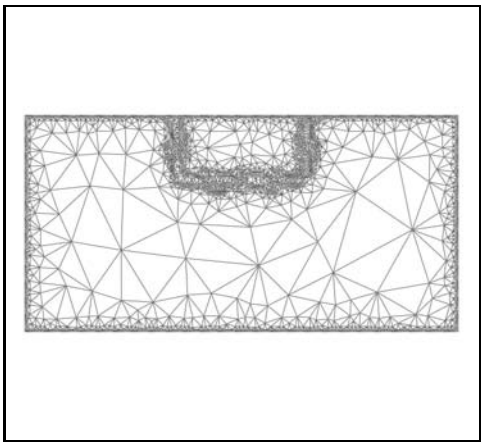


SIMPLIFIED MESH

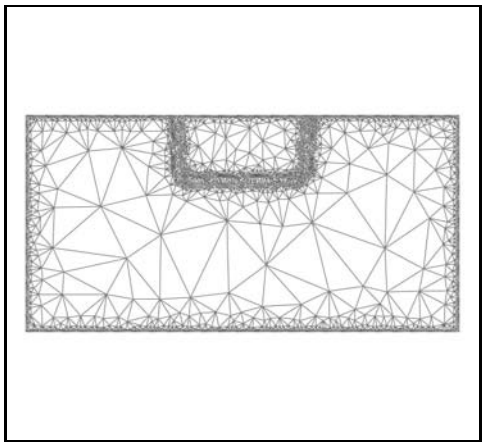
Figure 5.10: Visualization of the simplification process applied to the AUSTRIA image. Due to the solid color within the federal states, the influence of different threshold values is limited and no significant changes can be identified. Therefore, only one simplified mesh is provided.



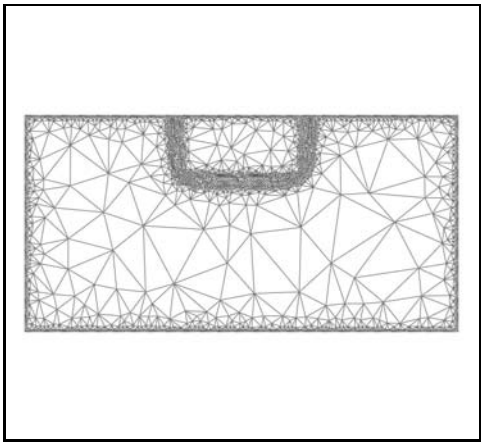
DEVICE IMAGE



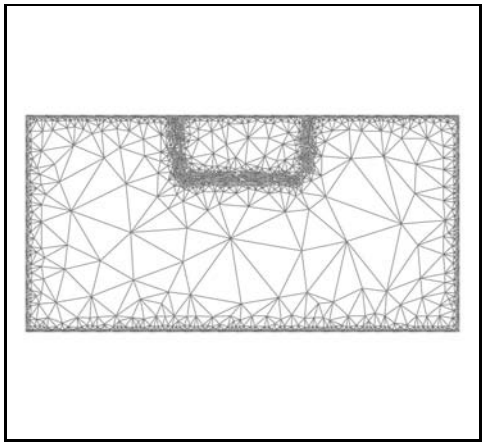
SIMPLIFIED MESH - THRESHOLD 4



SIMPLIFIED MESH - THRESHOLD 6



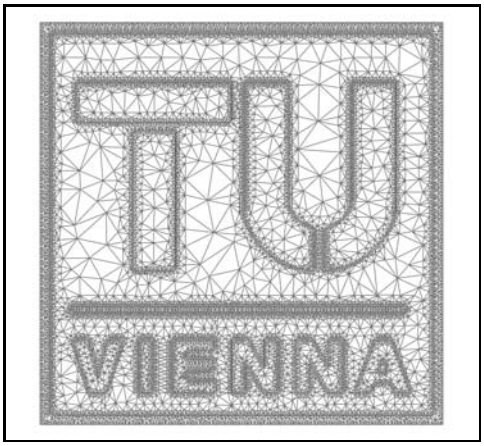
SIMPLIFIED MESH - THRESHOLD 8



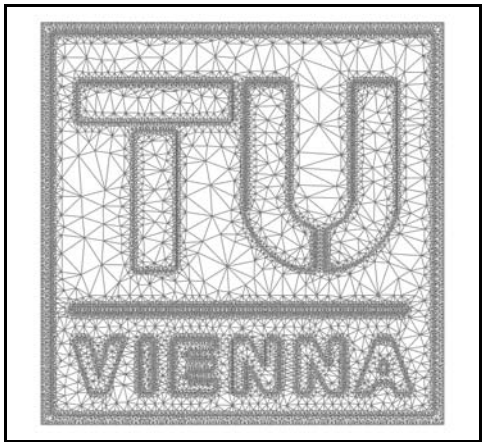
SIMPLIFIED MESH - THRESHOLD 10

Figure 5.11: Visualization of the simplification process applied to the DEVICE image using different threshold values. Note that the *black* mesh lines of the input image have been ignored by the simplification algorithm, as it is possible to specify a certain color layer (including a tolerance level) to be ignored by the simplification process. In other words, it enables to specifically remove color from the mesh. **middle left:** Note the fine region which contains a coarse region at the *pn*-junction. This is due to the sensible threshold value of 4%. The green region between the blue and yellow regions is thick enough to be coarsed for this threshold value. **bottom right:** The *pn*-junction region is effectively preserved.

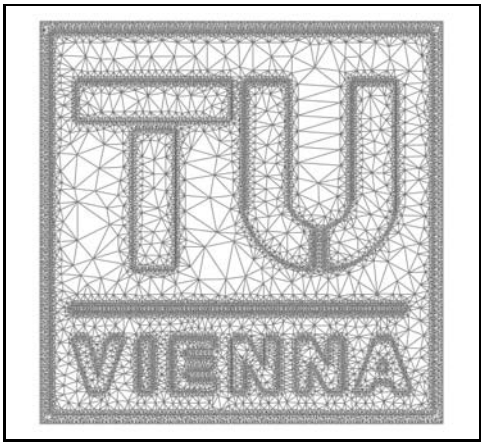
TU VIENNA LOGO IMAGE



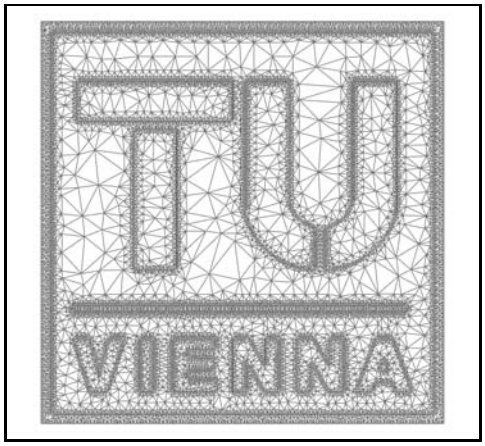
SIMPLIFIED MESH - THRESHOLD 4



SIMPLIFIED MESH - THRESHOLD 6

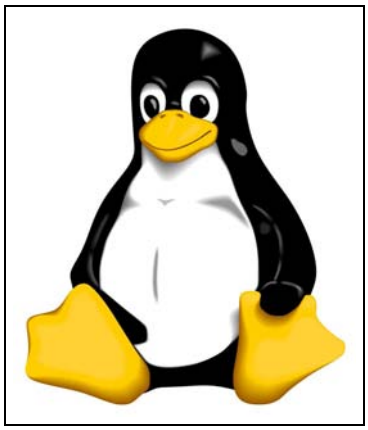


SIMPLIFIED MESH - THRESHOLD 8

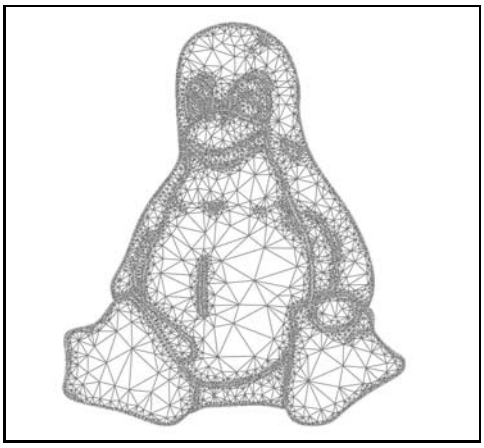


SIMPLIFIED MESH - THRESHOLD 10

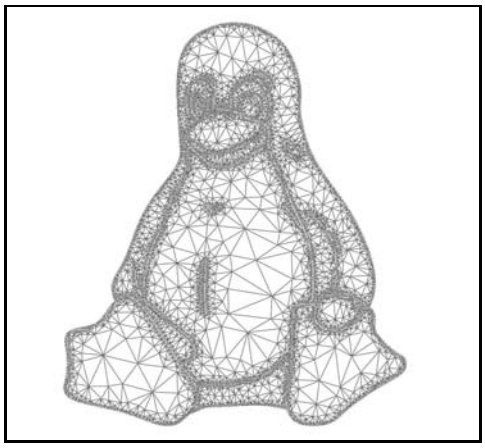
Figure 5.12: Visualization of the simplification process applied to the TU VIENNA LOGO image using different threshold values. Note that the input image does not offer many color gradients. Therefore the use of different threshold values is negligible. However, the transition regions from the background to the foreground (blue) should be sharper. Note the thick mesh regions surrounding the *vienna* writing and the horizontal line.



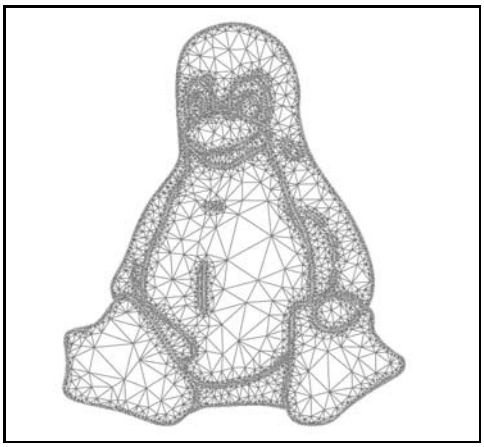
TUX IMAGE



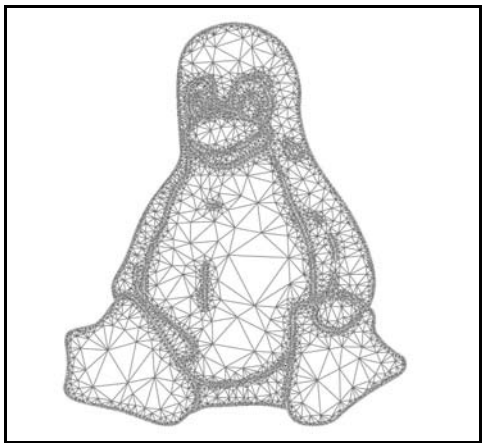
SIMPLIFIED MESH - THRESHOLD 4



SIMPLIFIED MESH - THRESHOLD 6

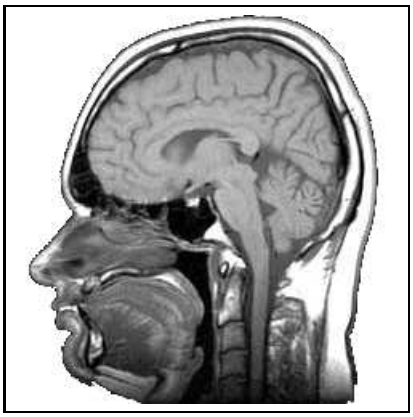


SIMPLIFIED MESH - THRESHOLD 8

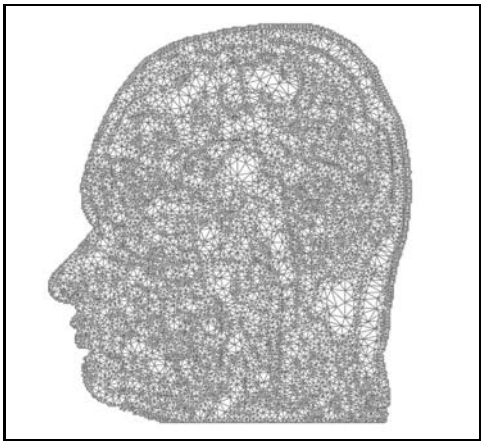


SIMPLIFIED MESH - THRESHOLD 10

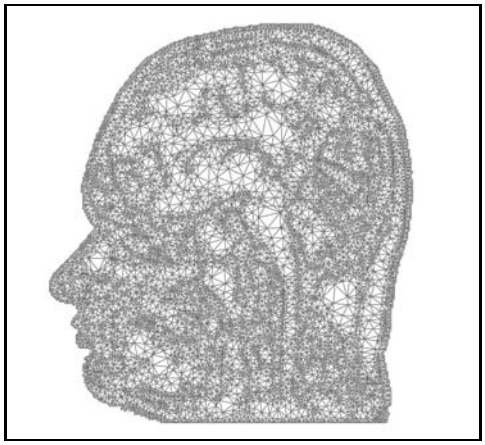
Figure 5.13: Visualization of the simplification process applied on the TUX image using different threshold values. Note the steady disappearance of the dark grey shadows on the left and right side of the chest and on the head (from **middle left** to **bottom right**).



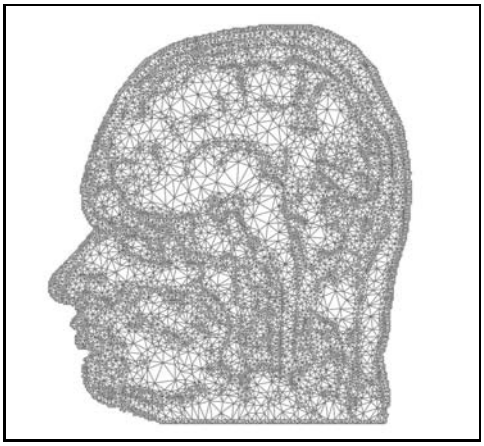
HEAD IMAGE



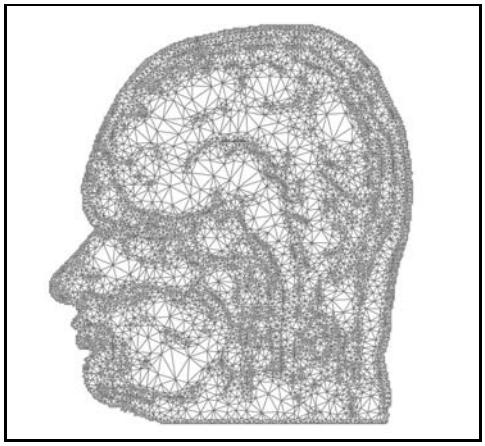
SIMPLIFIED MESH - THRESHOLD 4



SIMPLIFIED MESH - THRESHOLD 6



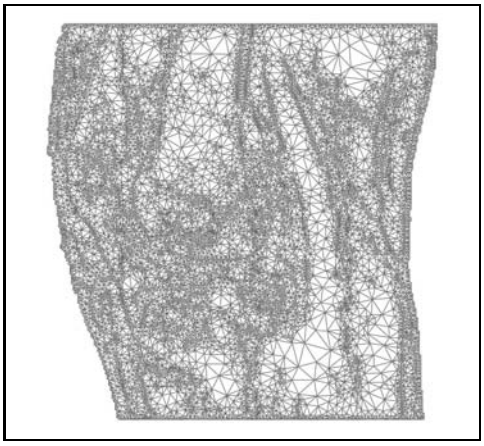
SIMPLIFIED MESH - THRESHOLD 8



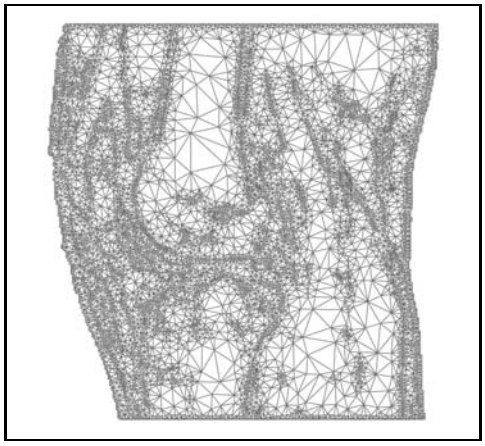
SIMPLIFIED MESH - THRESHOLD 10

Figure 5.14: Visualization of the simplification process applied to the HEAD image using different threshold values. **middle left:** Clearly the threshold level is too sensitive, as almost no contours can be identified. This is due to the fact that large regions of the image provide very smooth color gradients which constantly exceed the given threshold. For example, the areas around the mouth and the nose only change in shades of grey. **bottom right:** If the threshold level is sufficiently large to overcome the color gradients in these regions, coarsement takes effect. This behavior depicts the nonlinear dependency of the threshold on the color variation.

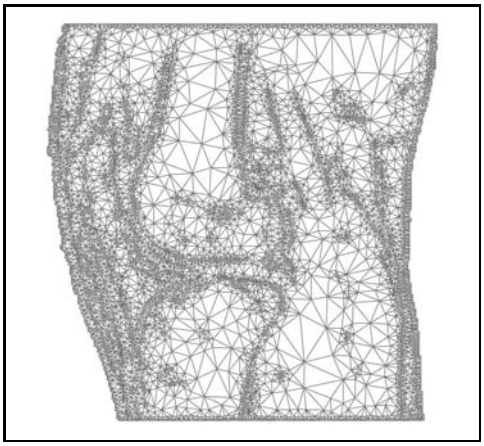
KNEE IMAGE



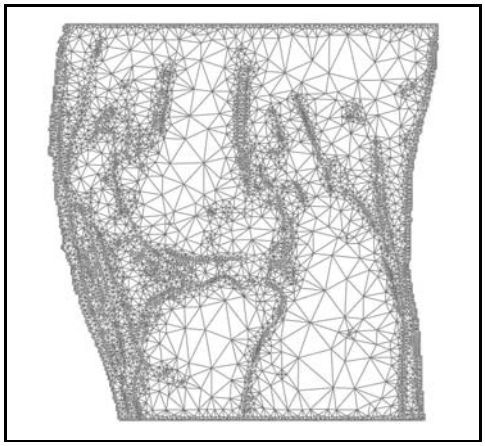
SIMPLIFIED MESH - THRESHOLD 4



SIMPLIFIED MESH - THRESHOLD 6



SIMPLIFIED MESH - THRESHOLD 8



SIMPLIFIED MESH - THRESHOLD 10

Figure 5.15: Visualization of the simplification process applied to the KNEE image using different threshold values. **middle left:** For the left part of the image the threshold value is too small to enable contour identification. For this region the same nonlinear behavior with respect to the threshold value can be observed as with the HEAD image. As the threshold increases contours become more discernible. **bottom right:** Contours can be identified reasonably.

5.1.4 Conclusion

The following table provides a more detailed overview of the simplification results.

	Image file size	**Image pixel size**	**Th** 4%	**Th** 6%	**Th** 8%	**Th** 10%
AUSTRIA	10.1 kByte	81385	6602	6574	6566	6583
DEVICE	36.1 kByte	154012	5017	4863	5164	4904
TU VIENNA LOGO	3.3 kByte	80089	12402	12349	12370	12377
TUX	114.2 kByte	752800	18638	17783	17162	16437
HEAD	58.9 kByte	58322	12173	11222	10304	9520
KNEE	56.6 kByte	61500	12494	9906	8329	7161

Table 5.1: Overview of the simplification results. **Column 1** identifies the different meshes. **Column 2** provides the file size of the input *png* file. **Column 3** lists the number of pixels, which is computed by multiplying the width by the length in pixel. **Column 4-7** provide the 0-cell size of the generated meshes using the different thresholds. Note the quite unexpected behavior of the 0-cell size for the AUSTRIA, DEVICE and TU VIENNA LOGO meshes. Although the threshold increases, the number of 0-cells partially increases as well, which is due to the intervention of the mesh engine to preserve mesh generation constraints.

A subjective evaluation of the results is provided in the following table.

	Th 4%	**Th** 6%	**Th** 8%	**Th** 10%
AUSTRIA				**x**
DEVICE				**x**
TU VIENNA LOGO				**x**
TUX	**x**			
HEAD				**x**
KNEE			**x**	

Table 5.2: Informal evaluation of the most effective threshold values based on visual qualification. Four of six simplifications seem to perform best with a threshold value of 10%. The TUX mesh simplification looses graphical accents for thresholds greater than 4%. The KNEE mesh sacrifices substantial anatomical details for a threshold greater than 8%.

The major contribution of this mesh adaption application is to generate a simplified 2-simplex mesh of an arbitrary image. Additionally, the details of the image are transferred to the mesh by preserving the corresponding mesh regions, which are detected using color gradients. This approach facilitates a reasonable reduction of mesh size while simultaneously preserving details. This fact is explicitly visible by comparing the sizes of the image and the simplified mesh of the TUX mesh. An image of around 750000 pixels can be represented by a mesh of a 0-cell size of around 18000. Thusly, this application eases the computational effort for external applications which process the generated mesh[6]. Generally, the approach offers reasonable results and provides a good basis for further improvements. For example, the algorithm can be revisited regarding variation sensitivity. A larger influence region around the investigated 0-cells can be used to investigate the color variation. Another improvement would be to implement a parallel mode by using *OpenMP* [30]. Furthermore, a three dimensional implementation can be implemented by applying the introduced two dimensional scheme on a set of images. A separate algorithm has to be implemented which combines these generated two dimensional meshes to build a three dimensional representation.

[6]Typically, mesh algorithms process a mesh by traversing the mesh elements. When the number of mesh elements is reduced, the overall computational effort decreases as well.

5.2 Mesh Adaption based on Quality Evaluation

This section introduces an implementation which refines an input mesh based on element quality evaluation. Degenerated elements are refined by inserting additional points at the barycenter. This basic adaption strategy shall act as an example of the applicability of the generalized algorithm approach introduced in **Section 3.4.2**. Furthermore the application assembles the *generic mesher suite* (**Section 4.3**) and the *generic functor queue* (**Section 4.4**). In principle the application is capable of processing meshes of arbitrary dimension and arbitrary mesh elements. However, only 2 and 3-simplex complex meshes are considered.

The first part outlines the implementations details using several code snippets for presentation and discussion. The second part introduces the input meshes which are to be adapted. The third part discusses the adaption results for 2-simplex and 3-simplex meshes, respectively. The final part sums up the achieved results up and outlines possible improvements.

5.2.1 Implementation

The following figure depicts the basic principle:

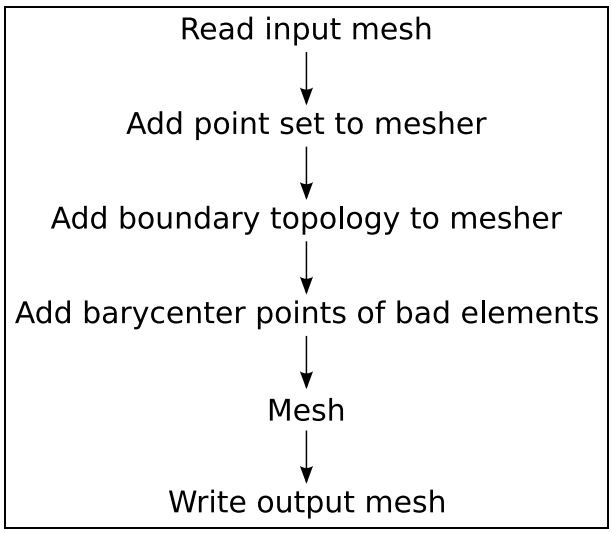


Figure 5.16: Basic principle of the implemented mesh adaption application.

The key part of the implementation is the use of a *generic functor queue* which evaluates the element quality and tags the elements which should be refined.

In the following the *generic functor queue* is shown:

$$Input \rightarrow^{(element_type, geometry_container)} Quality \rightarrow^{(double)} LessThan \rightarrow^{(bool)} TagIf \rightarrow^{(bool)} Output$$

Note that *Quality* is a binary functor, as the topological element and the geometry container is passed to the functor as an input. Therefore a specialized *fold* function has been implemented which takes two data elements and forward them to the first functor in queue.

The following code snippet depicts the declaration, definition and execution of the evaluation sequence:

```
1 typedef typename sequence< Quality, LessThan, TagIf >::type      EvaluationSequence;
2 EvaluationSequence evaluation_sequence(quality, less_than, tag_if);
3 for_each( cell_container,
4    fold2( evaluation_sequence, ref( geometry ), _1 )
5 );
```

Note that the last functor in the queue (*tag_if*) stores the boolean input data on an externally provided container. Therefore, the evaluation results are preserved.

This is an important fact as this approach decouples the evaluation process from the adaption process. As a consequence, the implementation can be mixed and matched with other implementations, which do not necessarily have to relate to the current task (mesh adaption). Consequently, reapplicability and reusability are increased, as already implemented code parts can be reused by other implementations.

Finally, the adaption implementation part has been implemented:

```
1 typedef typename sequence< Barycenter >::type      BarycenterSequence;
2 BarycenterSequence   barycenter_sequence(barycenter);
3 for_each( cell_container, tag_container,
4    if_( _2 == 1 ) [
5       let( _a = fold2( barycenter_sequence, ref( geometry ), _1 ) ) [
6          add( at( _a, 0 ), ref( mesher ) )
7       ]
8    ]
9 );
```

The *tag_container* contains the result of the evaluation sequence. Note that an adapted *for_each* function has been implemented, which offers a parallel traverse[7] of two containers (**Line 3**). This parallel traverse enables the access of the element of the cell container and the corresponding result of the evaluation queue. The *Lambda placeholders _1, _2* provide access to the cell and the related evaluation result respectively (**Lines 4, 5**). The *local name*[8] *_a* temporarily stores the result of the barycenter (**Lines 5, 6**).

Even though *barycenter_sequence* consists solely of the barycenter functor, the encapsulation within a sequence allows direct application of functors within *Boost Phoenix* environments.

The algorithm processes as follows: For each cell and for each corresponding tag, test if the tag is one. If so, compute the barycenter for this cell, and add the result to the mesher's input geometry.

The implemented refinement approach is based on adding the barycenters of degenerated elements. Note that the algorithm does not model the *barycentric subdivision* [14] algorithm. The implemented tool adds the barycenter to the input geometry container of the mesh engine. Therefore, the internal mesh quality algorithms deal with possibly inadequatenesses due to the additional barycenter point. Note that *barycentric subdivision* is a technique to refine an element manually without the use of a mesh engine.

Barycentric subdivision is simple to implement, but lacks numerical quality. This is due to the fact that the barycenter[9] is always inside the element. Due to the construction principle, degenerated elements result in the introduction of elements of worsening quality.

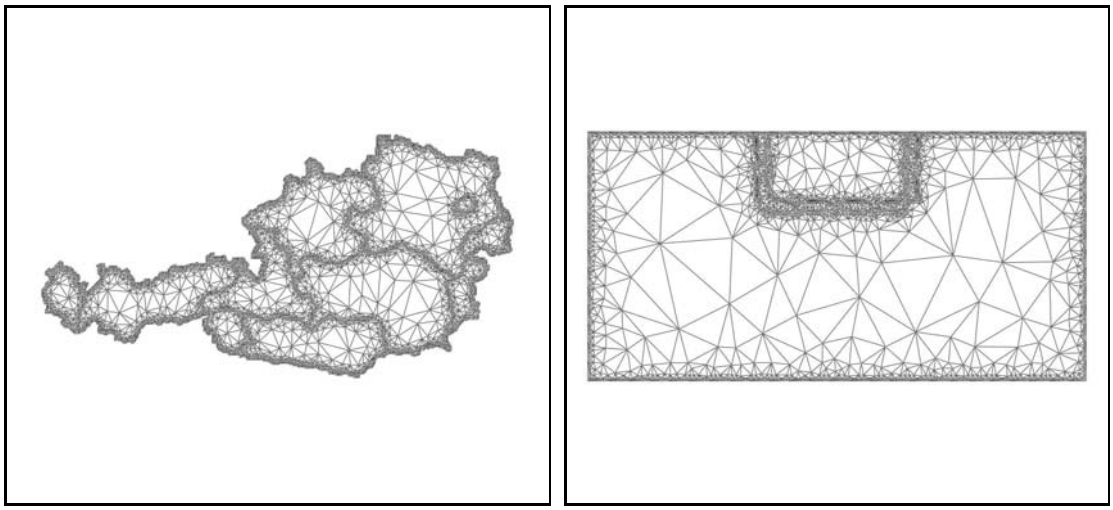
[7] The extension of *for_each* to enable parallel traverse has been inspired by the *Boost Fusion zip_view*. This function enables *parallel traversal* of two *Boost Fusion Sequences*.

[8] *Local names* as well as *Lambda placeholders* are part of the *Boost Phoenix Library* [7].

[9] The barycenter is also referred to as *centroid* or *center of mass*.

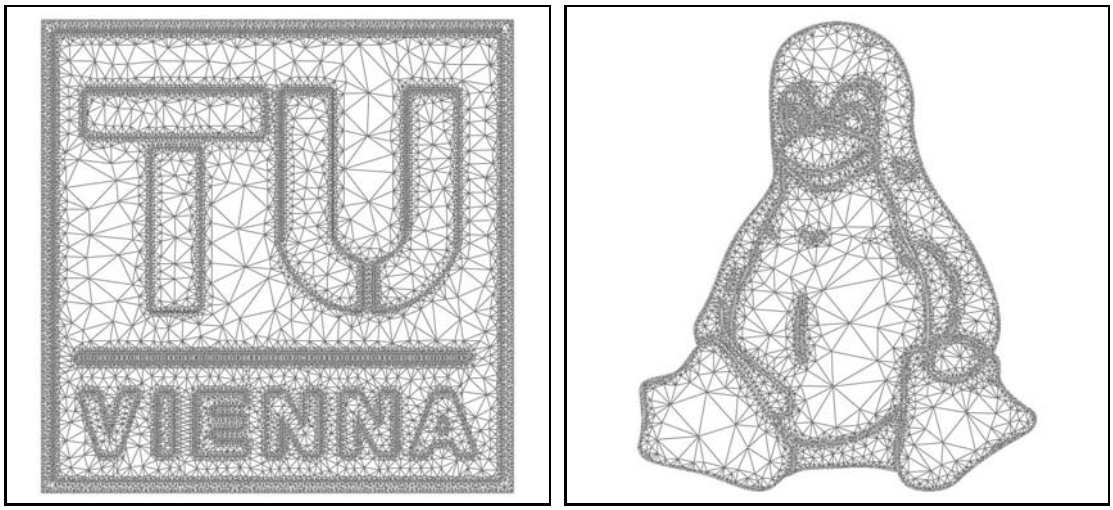
5.2.2 Input Data

The following introduces the input meshes for the mesh adaption application.



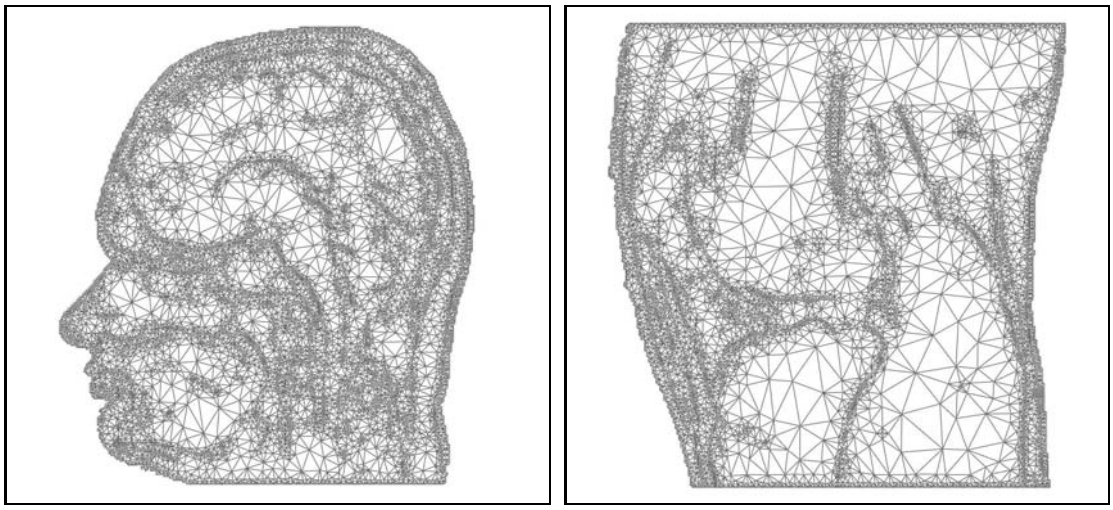
AUSTRIA DEVICE

Figure 5.17: left: A mesh discretizing a map of *austria*. **right:** A mesh carrying a doping profile of a *mosfet*.



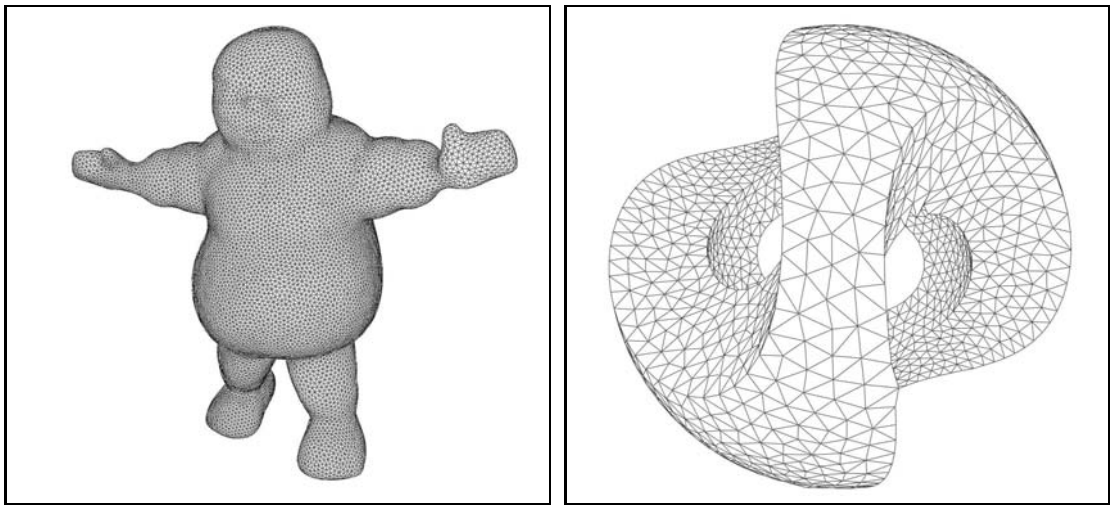
TU VIENNA LOGO TUX

Figure 5.18: left: A mesh of the *TU Vienna Logo*. **right:** A mesh of the official mascot of the the *Linux* operating system named *Tux*.



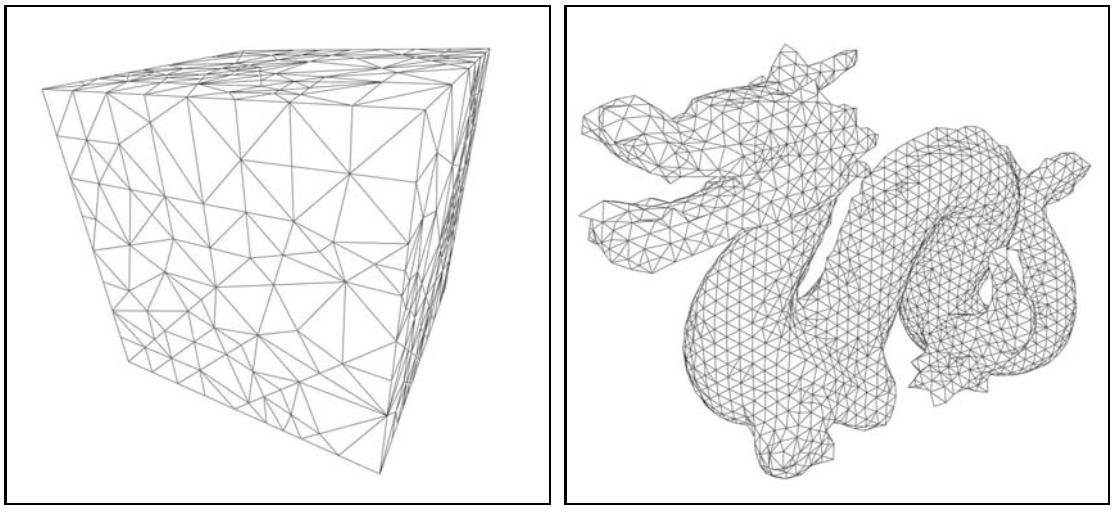
HEAD KNEE

Figure 5.19: left: A mesh of an *magnetic resonance image* (*MRI*) of a human head. **right:** A mesh of an *magnetic resonance image* (*MRI*) of a human knee.



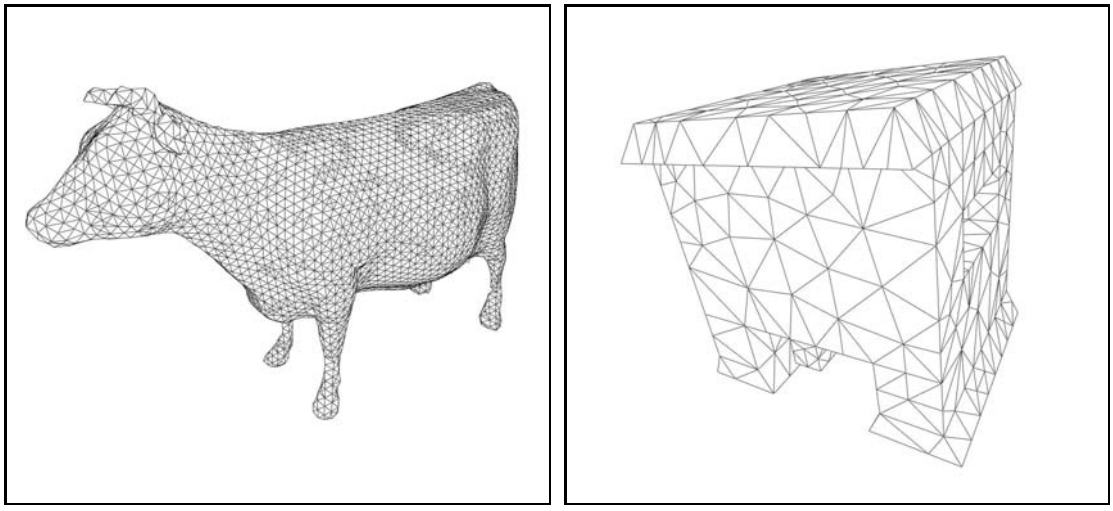
STAYPUFT STGALLEN

Figure 5.20: left: A mesh of the *stay puft marshmallow man*. **right:** A mesh of a sculpture named *StGallen*.



RAND2 DRAGON

Figure 5.21: left: A mesh of an ordinary *cube*. **right:** A mesh of a chinese *dragon*.



COW HOUSE

Figure 5.22: left: A mesh of a *cow*. **right:** A mesh of a *house*.

- AUSTRIA, DEVICE, TU VIENNA LOGO, TUX, HEAD and KNEE are meshes created by the tool introduced in **Section 5.1**. All meshes are meshed using the *Triangle2D* mesh generation engine to produce a *constrained Delaunay triangulation*[10].

- STAYPUFT, STGALLEN, RAND2, DRAGON, COW and HOUSE are meshes used for testing a novel mesh improvement technique based on vertex insertion [4]. The *hulls* of these volume meshes have been extracted using the tool described in **Section 4.1** before being volume meshed by the *Tetgen* mesh generation engine using *conforming Delaunay triangulation*[11]. For this reason the input volume mesh has been created by the same mesh generation engine as the adapted mesh, which consequently results that the adaption process of the implemented approach is not influenced by different mesh engines.

[10] Typically, this introduces elements with bad quality. This approach enables a greater scope for element adaption based on element quality. A *conforming Delaunay triangulation* would generate high quality meshes, which would narrow the test spectrum as the input meshes would already be of high quality.

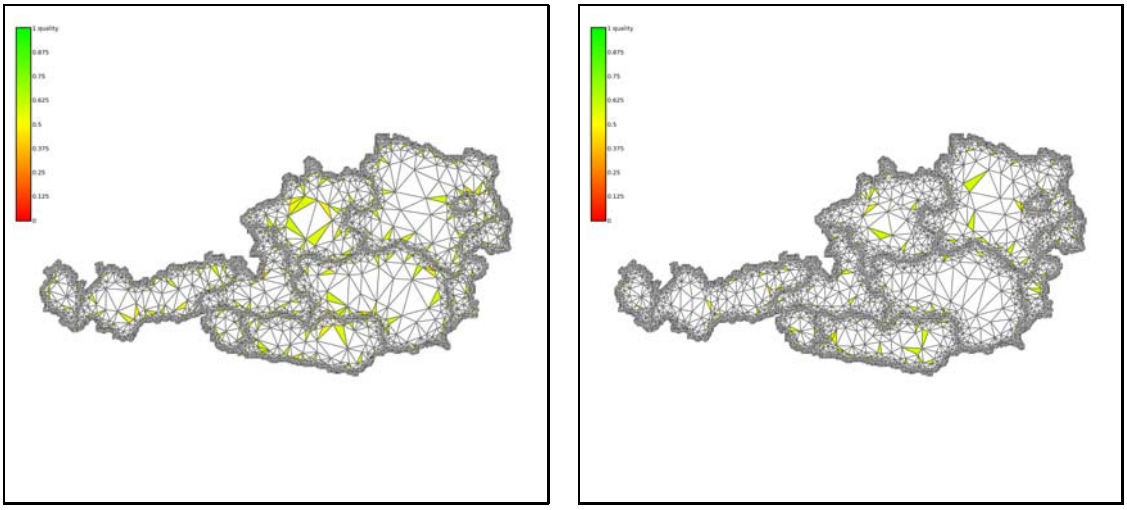
[11] Three dimensional mesh generation is much more challenging compared to the two dimensional counterpart (**Section 3.2.2**). Therefore, the quality of *conforming Delaunay tetrahedrizations* is typically low compared to the two dimensional cases. As a consequence, the three dimensional meshes have been created using the *conforming Delaunay* property.

5.2.3 Mesh adaptions

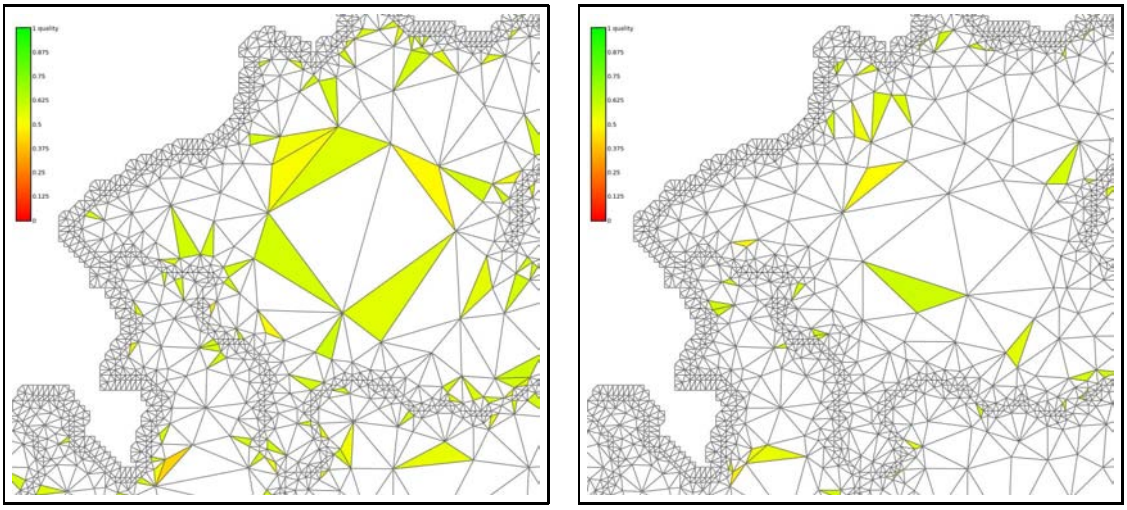
2-simplicial complexes

In the following, all figures on the *left* relate to the input meshes, whereas all figures on the *right* relate to the adapted meshes to provide a direct comparison.

The meshes with evaluated quality are shown, in total and zoomed view. Note, that only cells with a cell quality of ≤ 0.6 are highlighted[12]. Cells resulting in a quality in this range are refined.



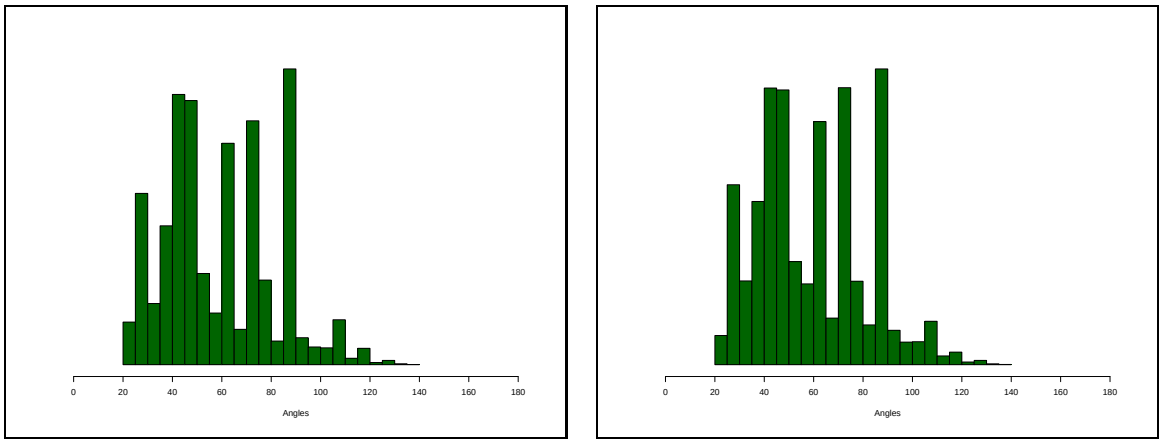
Original | Refined



Original - Detailed View | Refined - Detailed View

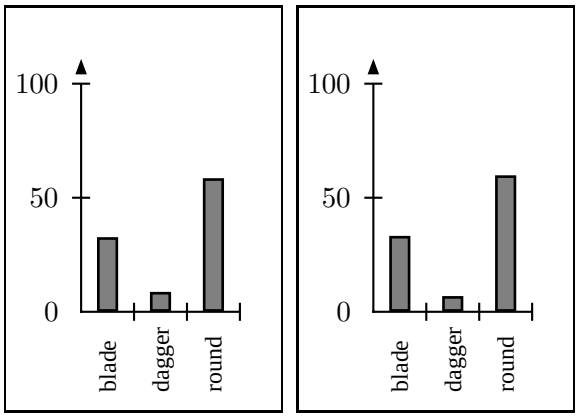
Figure 5.23: Visualization of the element quality distribution of the AUSTRIA mesh. The detailed view reveals, that the adaption removes bad input elements, but introduces new degenerated elements. The lowest occurring element quality is increased from a value of 0.4206 to 0.4463.

[12] The depicted mesh adaptions should outline the adaption capabilities of this basic adaption approach. An exemplary quality threshold has been chosen, as different quality thresholds yield similar results.



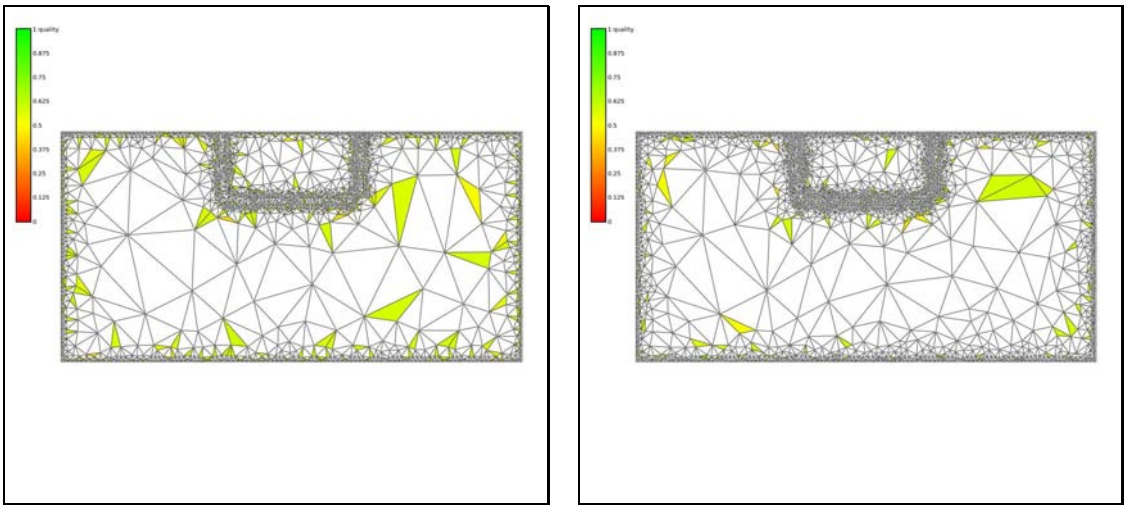
Original Refined

Figure 5.24: Angle distributions in the AUSTRIA mesh before and after adaption. The number of elements with an angle of $120°$ has decreased, the number containing $20°$ angles has even halved, while the count of elements with an angle of $30°$ and $60°$ has increased after the adaption. The improvement due to adaption is made explicit by the shift to this intermediate range of angles.

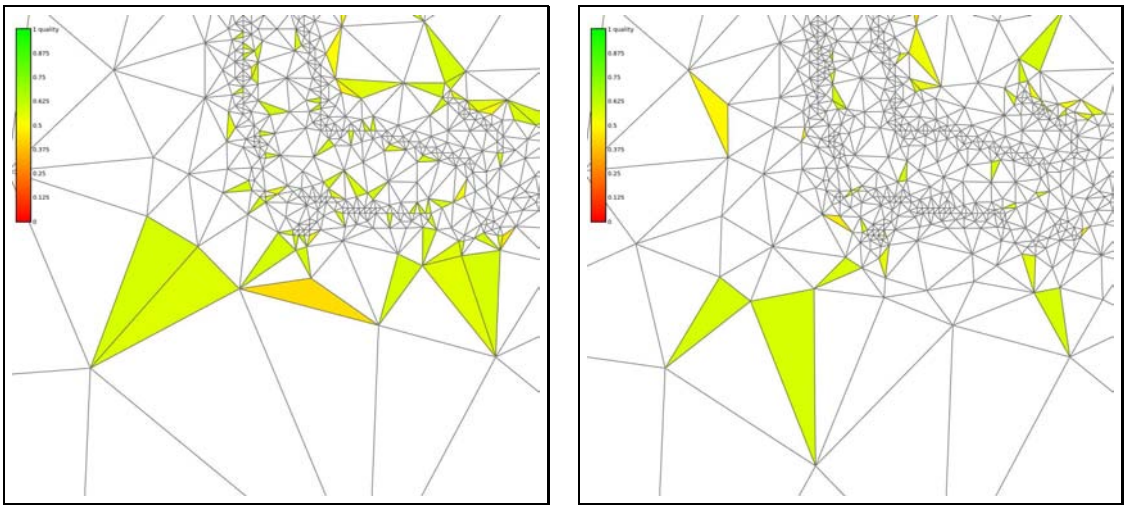


Original Refined

Figure 5.25: Overview of classified element types in the AUSTRIA mesh before and after adaption. The number of *dagger* elements has been reduced while the number of *blades* and *rounds* has been increased by using the adaption.

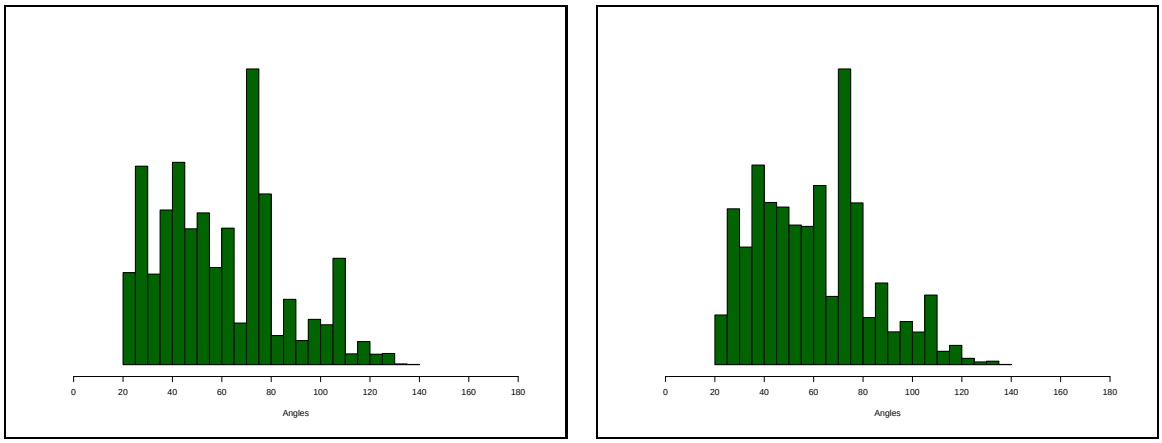


Original | Refined



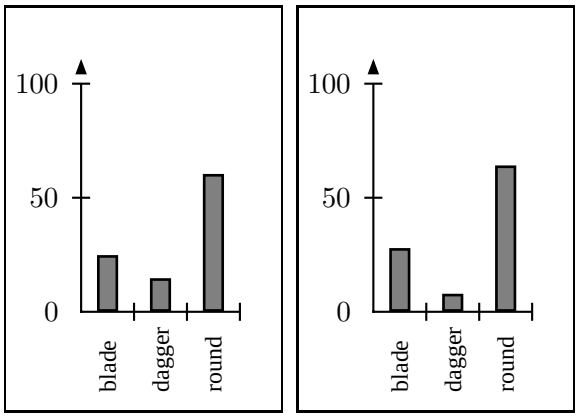
Original - Detailed View | Refined - Detailed View

Figure 5.26: Visualization of the mesh adaption result of the DEVICE mesh. The zoomed view shows that there are less elements with a quality ≤ 0.6 after adaption than in the input mesh. The lowest occurring element quality has increased only slightly, from 0.4279 to 0.4281.



Original Refined

Figure 5.27: Angle distributions in the DEVICE mesh before and after adaption. The number of elements with an angle of $20°$, $25°$ and $130°$ has decreased, while the count of elements with an angle of $30°$, $60°$ and $90°$ has increased after the adaption. The improvement due to adaption is made explicit by the shift to this intermediate range of angles.



Original Refined

Figure 5.28: Overview of classified element types in the DEVICE mesh before and after adaption. The number of *dagger* elements has been reduced while the number of *blades* and *rounds* has been increased by using the adaption.

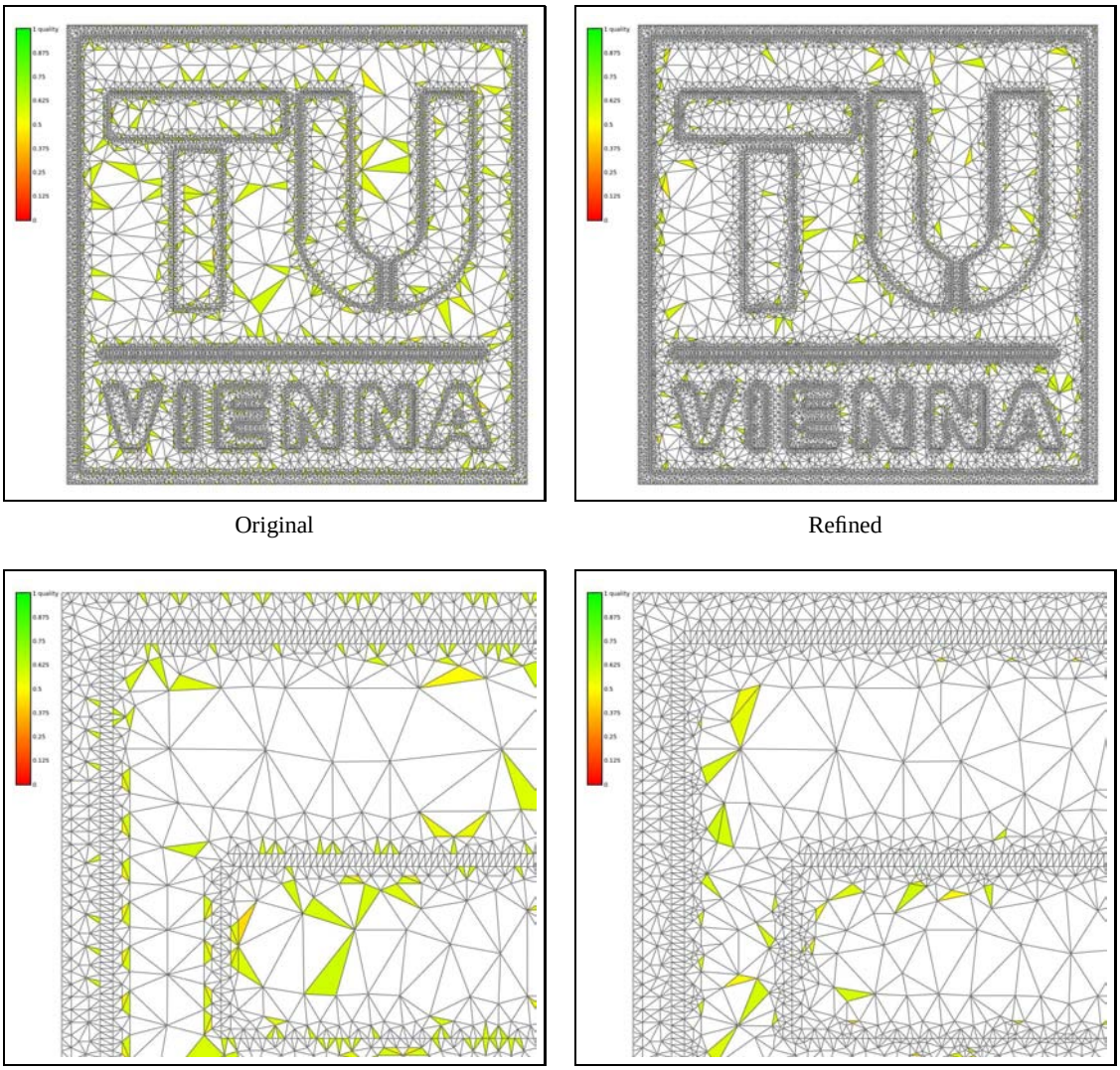
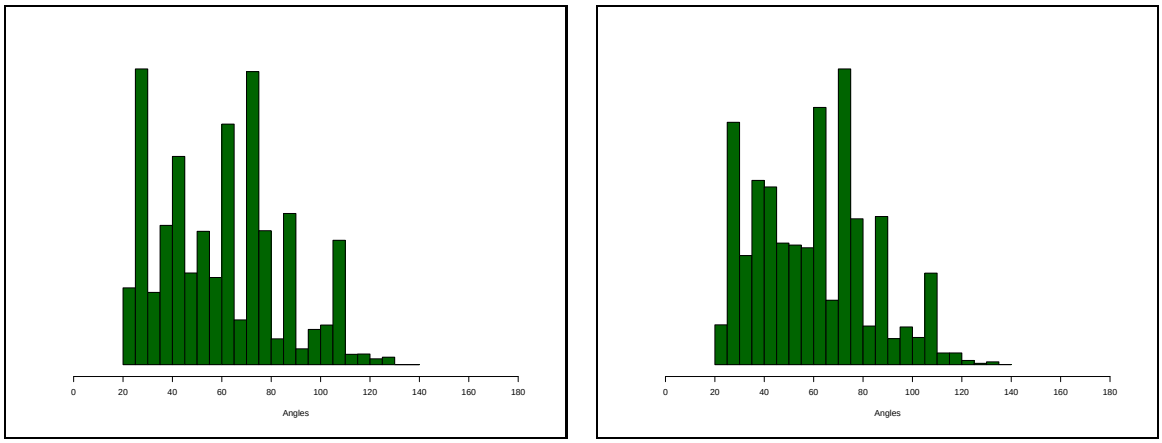
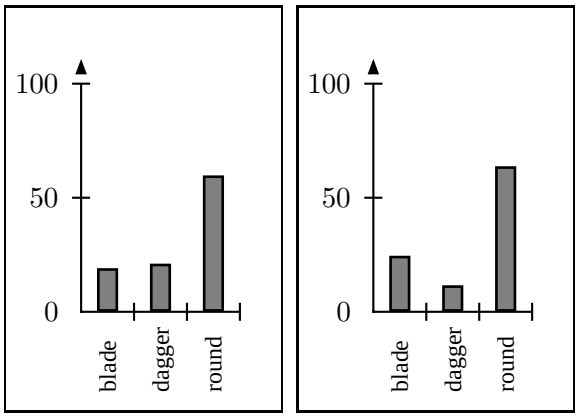


Figure 5.29: Mesh adaption of the TU VIENNA LOGO. A reduction of bad elements can clearly be recognized. The lowest occurring element quality is slightly decreased, from 0.4153 to 0.4043.



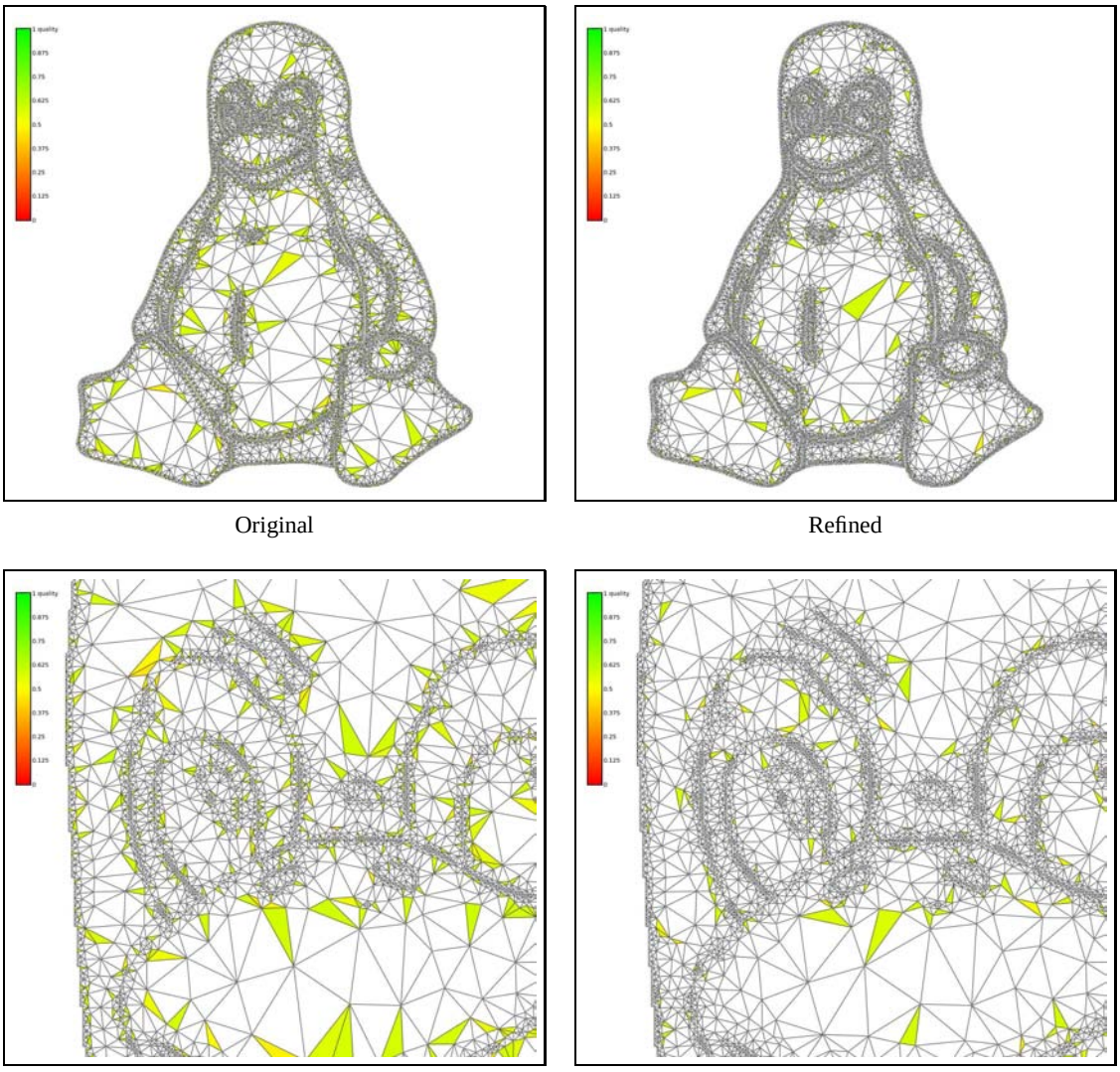
Original Refined

Figure 5.30: Angle distributions in the TU VIENNA LOGO mesh before and after adaption. The number of elements with an angle of $20°$, $40°$, $110°$ and $130°$ has decreased, while the count of elements with an angle of $35°$ and $70°$ has increased after the adaption. The improvement due to adaption is made explicit by the shift to this intermediate range of angles.



Original Refined

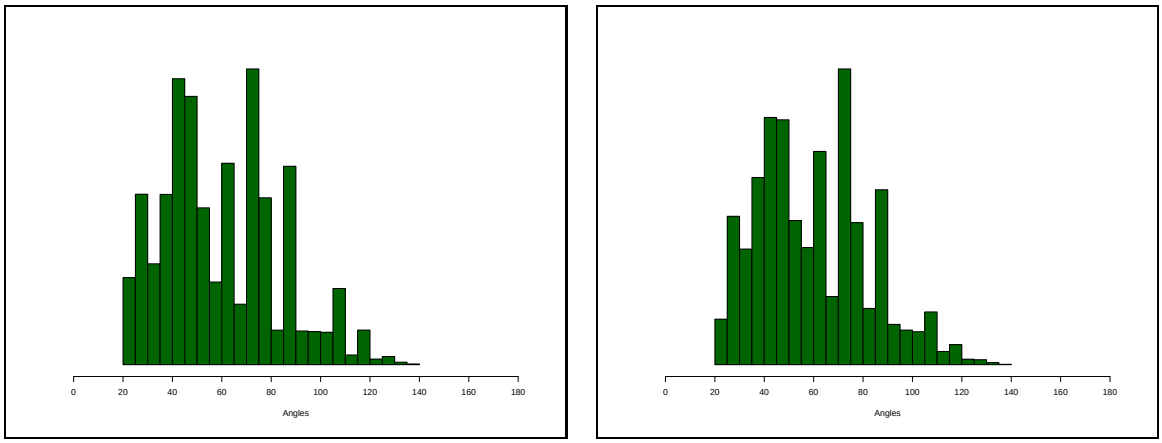
Figure 5.31: Overview of classified element types in the TU VIENNA LOGO mesh before and after adaption. The number of *dagger* elements has been reduced while the number of *blades* and *rounds* has been increased by using the adaption.



Original | Refined

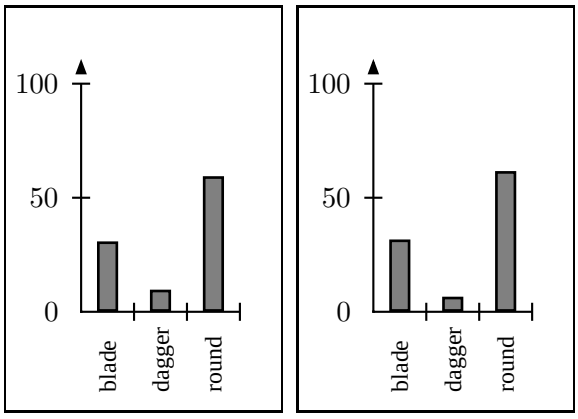
Original - Detailed View | Refined - Detailed View

Figure 5.32: Mesh adaption of the TUX mesh. Clearly a reduction of bad elements can be recognized. The lowest occurring element quality is slightly increased, from 0.4118 to 0.4124.



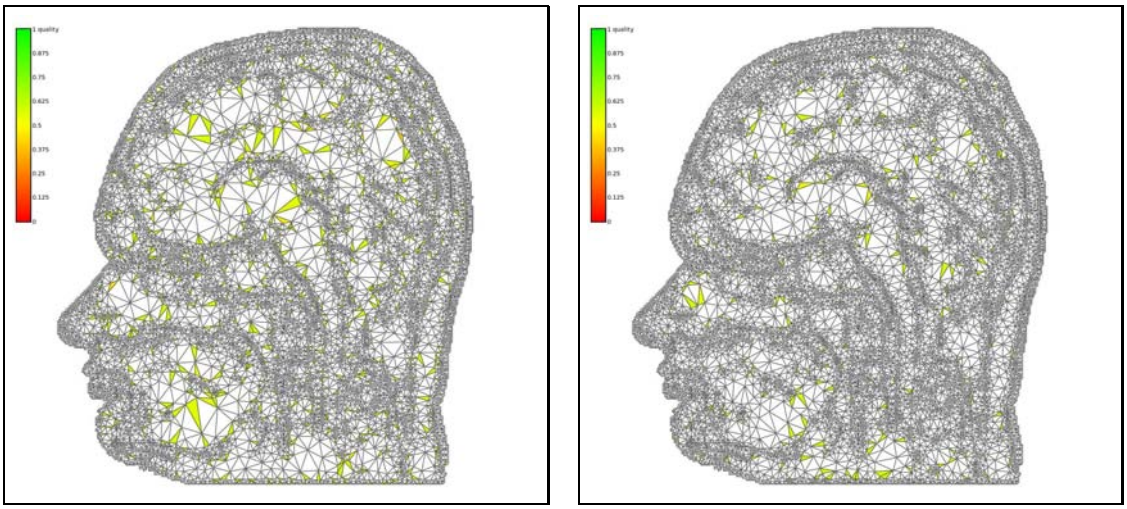
Original Refined

Figure 5.33: Angle distributions in the TUX mesh before and after adaption. The number of elements with an angle of $20°$, $40°$, $110°$ and $130°$ has decreased, while the count of elements with an angle of $35°$, $60°$ and $83°$ has increased after the adaption. The improvement due to adaption is made explicit by the shift to this intermediate range of angles.

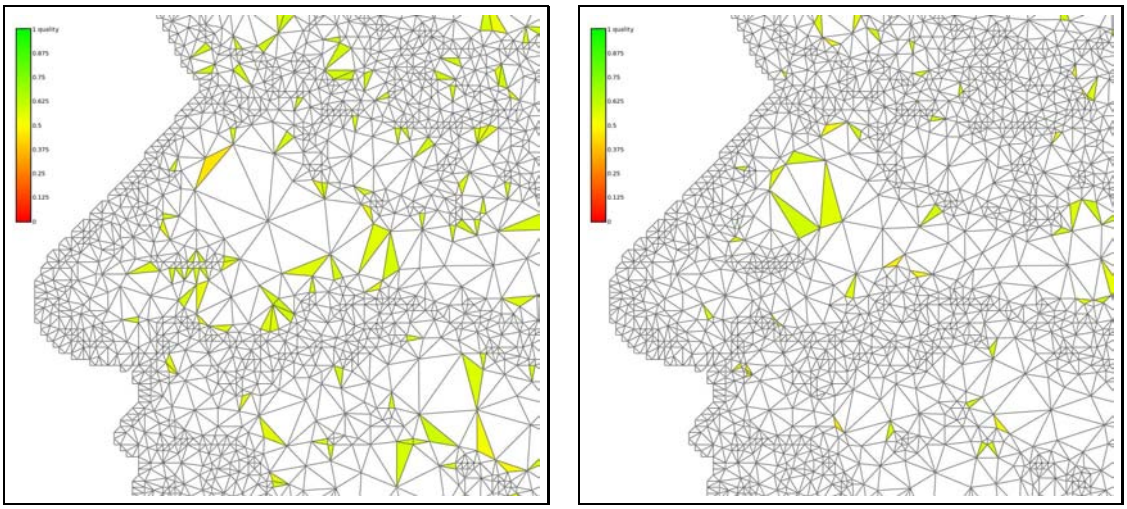


Original Refined

Figure 5.34: Overview of classified element types in the TUX mesh before and after adaption. The number of *dagger* elements has been reduced while the number of *blades* and *rounds* has been increased by using the adaption.

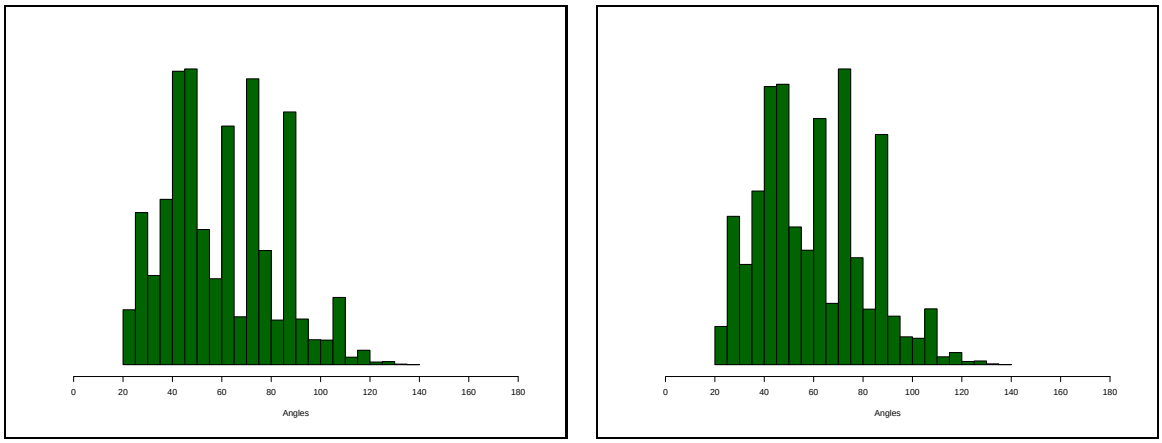


Original | Refined



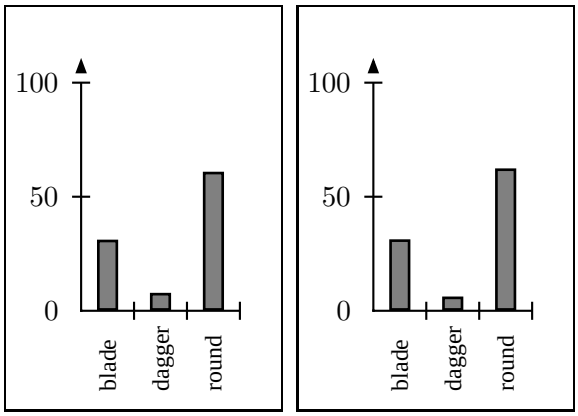
Original - Detailed View | Refined - Detailed View

Figure 5.35: Mesh adaption of the HEAD mesh. A decrease of elements offering a quality of ≤ 0.6 can be recognized. The lowest occurring element quality is increased notably, from 0.4059 to 0.4385.



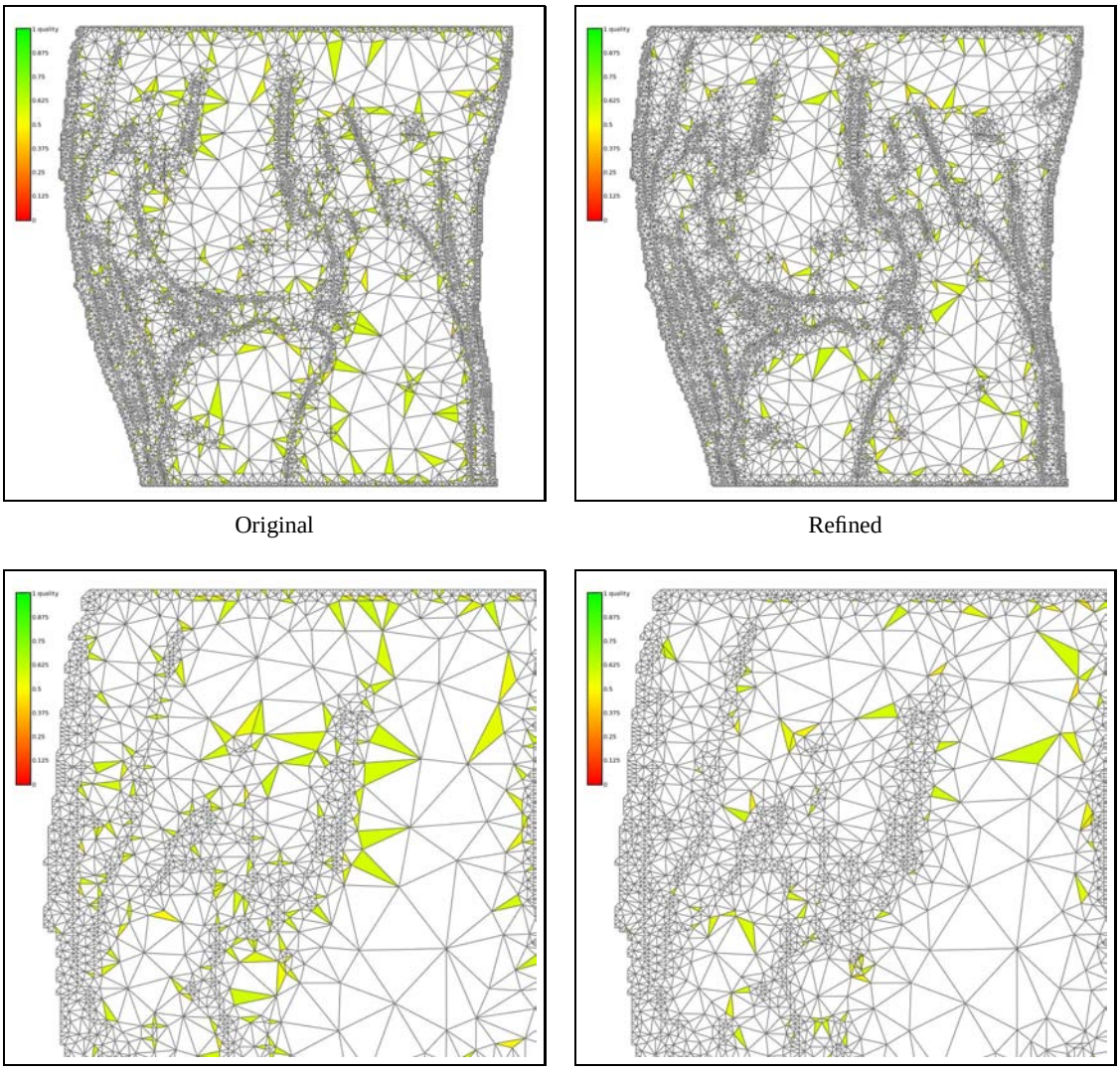
Original Refined

Figure 5.36: Angle distributions in the HEAD mesh before and after adaption. The number of elements with an angle of $20°$ and $110°$ has decreased, while the count of elements with an angle of $30°$ and $70°$ has increased after the adaption. Generally, there is no significant change in the angle distribution.



Original Refined

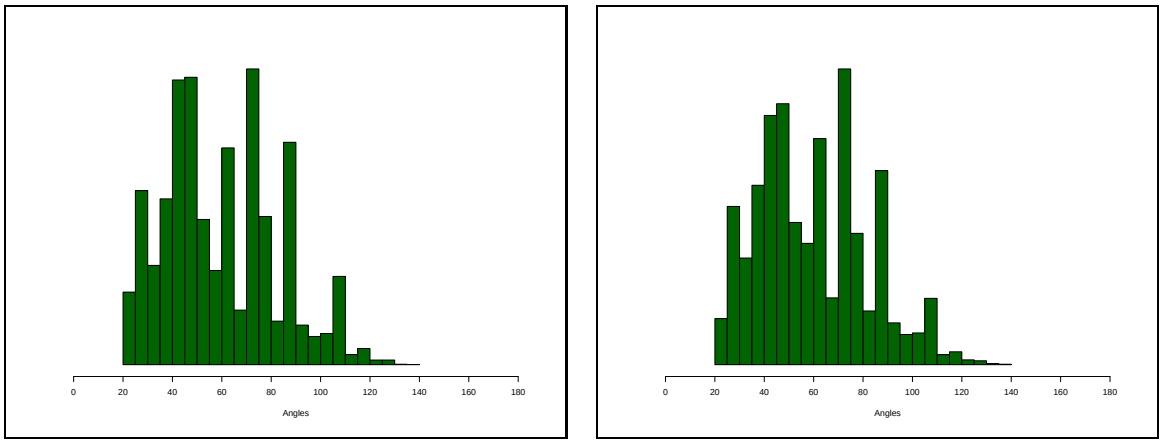
Figure 5.37: Overview of classified element types in the HEAD mesh before and after adaption. The number of *dagger* elements has been reduced while the number of *blades* and *rounds* has been increased by using the adaption.



Original | Refined

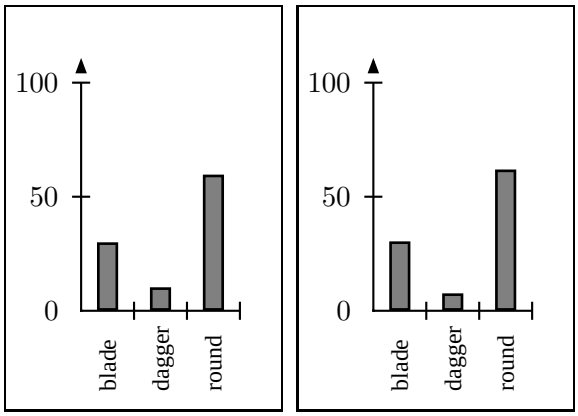
Original - Detailed View | Refined - Detailed View

Figure 5.38: Mesh adaption of the KNEE mesh. A reduction of degenerated elements can be identified. The lowest occurring element quality is slightly increased, from 0.4204 to 0.4231.



Original Refined

Figure 5.39: Angle distributions in the KNEE mesh before and after adaption. The number of elements with an angle of $20°$, $80°$ and $90°$ has decreased, while the count of elements with an angle of $40°$ and $60°$ has increased after the adaption. Generally, there is no significant change in the angle distribution.

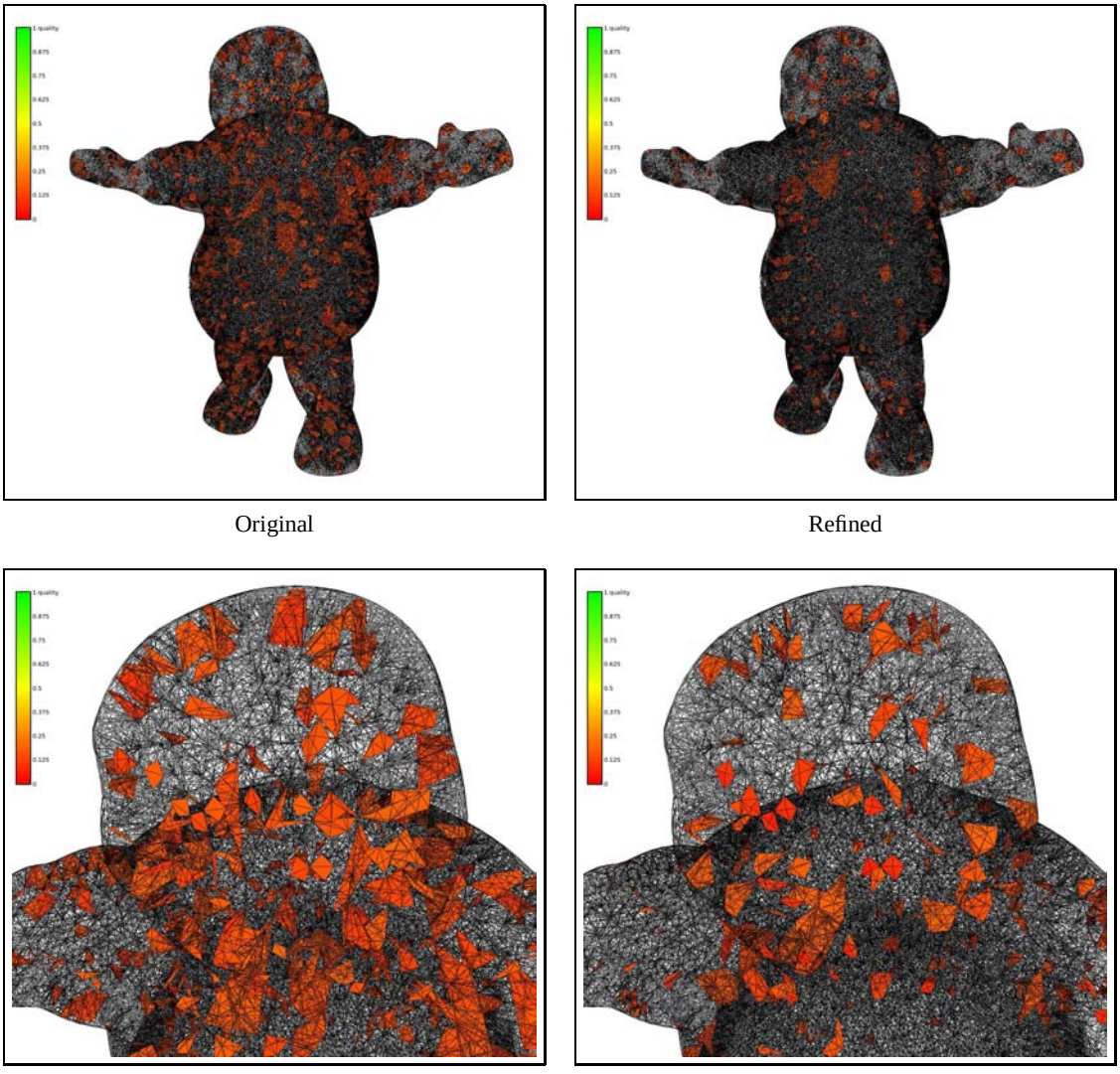


Original Refined

Figure 5.40: Overview of classified element types in the KNEE mesh before and after adaption. The number of *dagger* elements has been reduced while the number of *blades* and *rounds* has been increased by using the adaption.

3-simplicial complexes

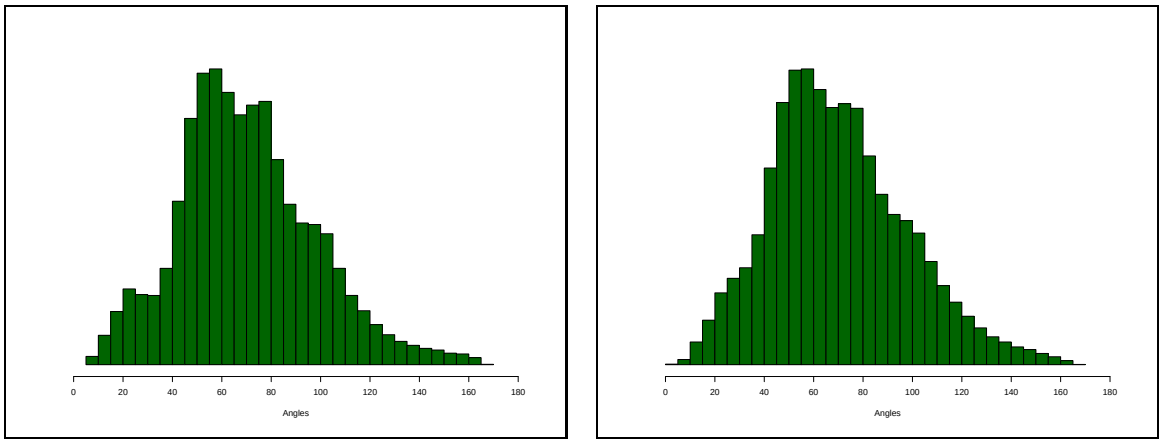
Results of the implemented mesh adaption approach applied to 3-simplex complex meshes are discussed in the following. The meshes with evaluated quality are shown in total and zoomed view. Note that only cells with a cell quality of ≤ 0.2 are highlighted. Cells resulting in a quality in this range are refined.



Original | Refined

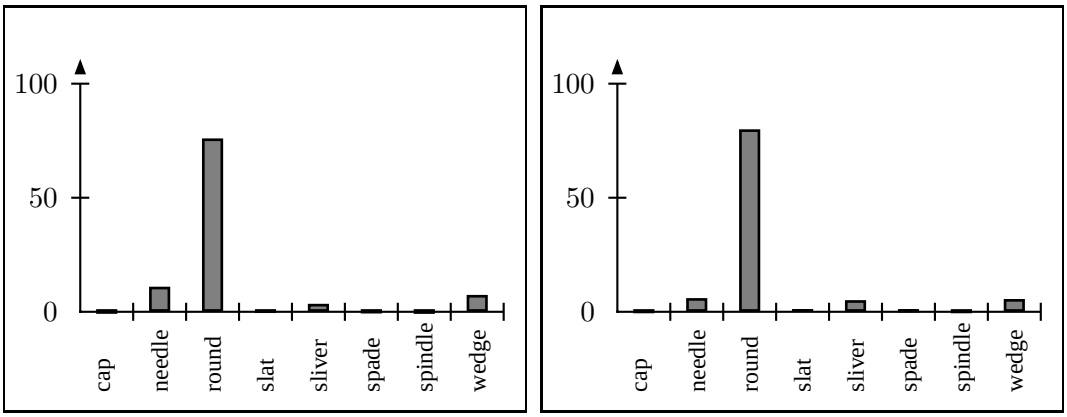
Original - Detailed View | Refined - Detailed View

Figure 5.41: Visualization of the element quality distribution of the STAYPUFT mesh. Elements with quality ≤ 0.2 are highlighted. The total number of elements having a quality of ≤ 0.2 is decreased considerably, while the lowest occurring element quality is decreased, from 0.0920 to 0.0090.



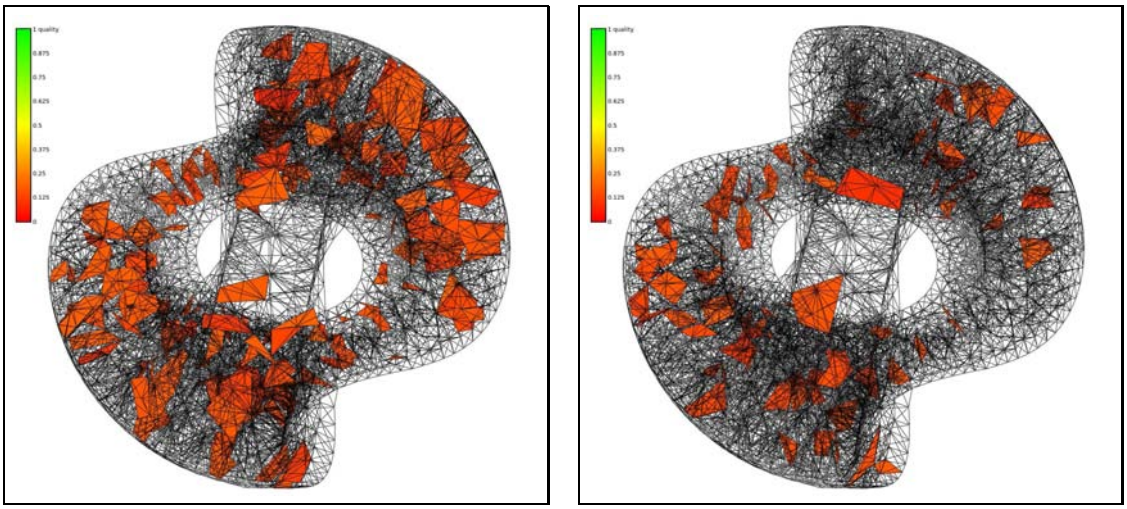
Original Refined

Figure 5.42: Angle distributions in the STAYPUFT mesh before and after adaption. The number of elements with an angle of $10°$ and $20°$ has decreased, while the count of elements with an angle of of $30°$, $70°$ and $90°$ has increased after the adaption. The improvement due to adaption is made explicit by the shift to this intermediate range of angles.

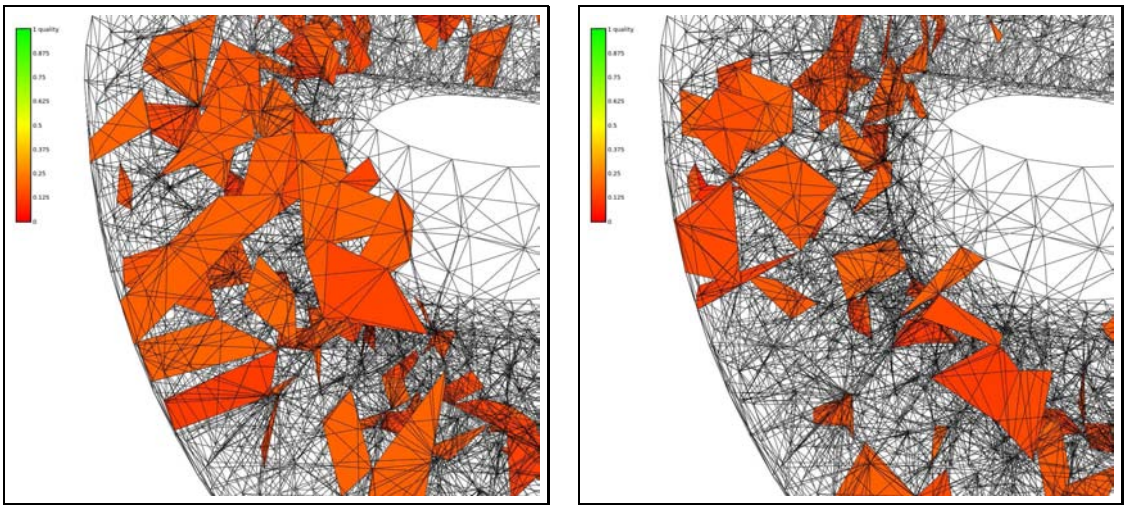


Original Refined

Figure 5.43: Overview of classified element types in the STAYPUFT mesh before and after adaption. The number of *needle* and *wedge* elements has been reduced while the number of *slivers* and *rounds* has been increased by using the adaption.

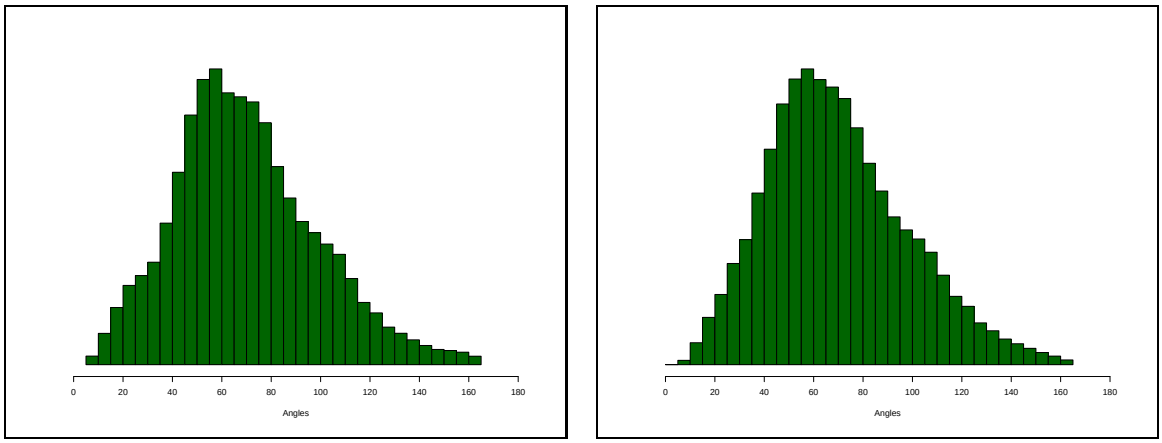


Original | Refined



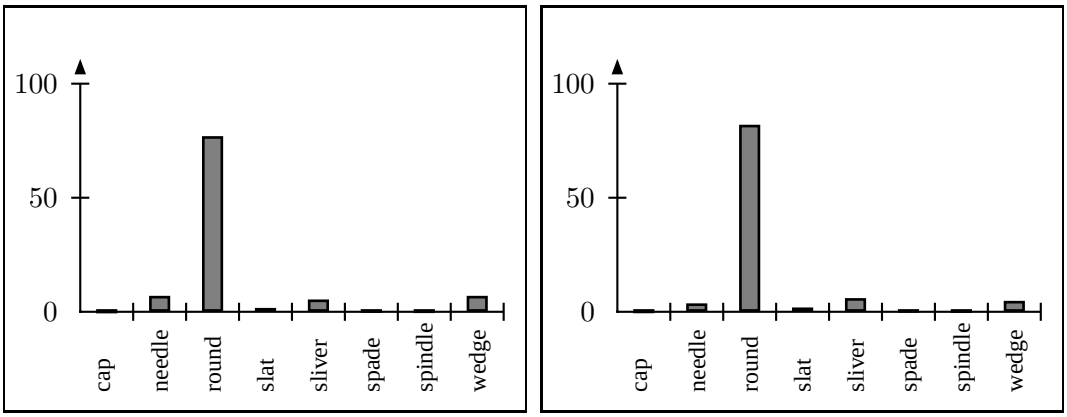
Original - Detailed View | Refined - Detailed View

Figure 5.44: Visualization of the element quality distribution of the STGALLEN mesh. Elements with quality ≤ 0.2 are highlighted. Clearly the total number of elements offering a quality of ≤ 0.2 is decreased considerably. The lowest occurring element quality is slightly decreased, from 0.0816 to 0.0623.



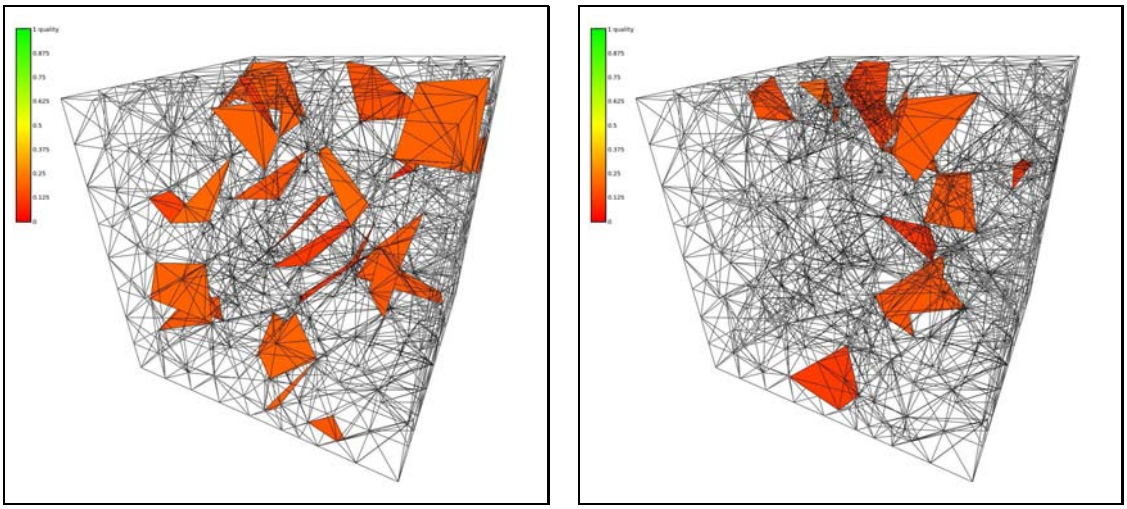
Original Refined

Figure 5.45: Angle distributions in the STGALLEN mesh before and after adaption. The number of elements with an angle of $20°$ and $160°$ has decreased, while the count of elements with an angle of $30°$ and $70°$ has increased after the adaption. The improvement due to adaption is made explicit by the shift to this intermediate range of angles.

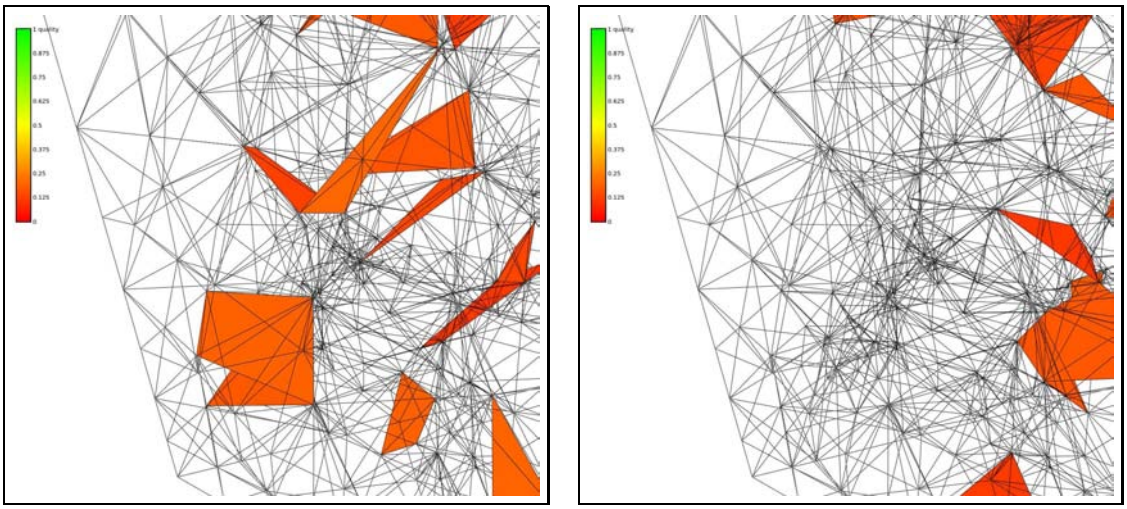


Original Refined

Figure 5.46: Overview of classified element types in the STGALLEN mesh before and after adaption. The number of *needle* and *wedge* elements has been reduced while the number of *slivers* and *rounds* has been increased by using the adaption.

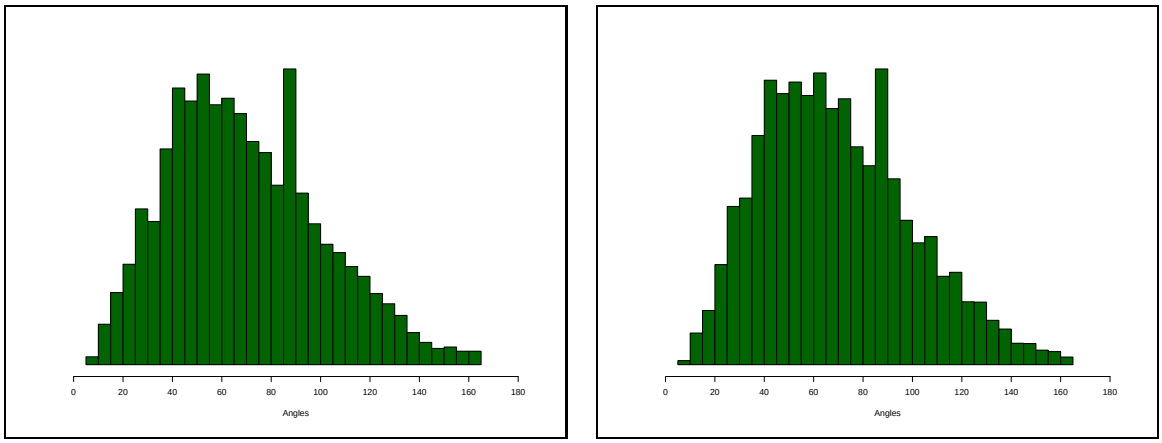


Original | Refined



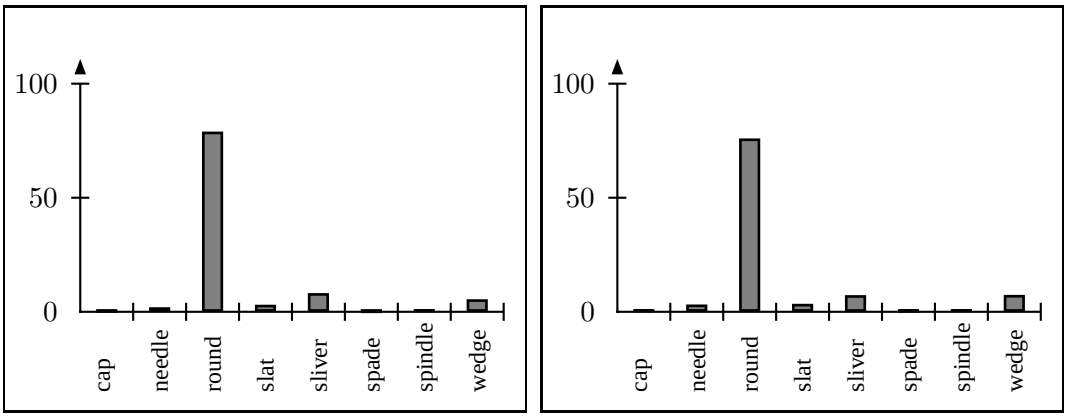
Original - Detailed View | Refined - Detailed View

Figure 5.47: Visualization of the element quality distribution of the RAND2 mesh. Elements with quality ≤ 0.2 are highlighted. Clearly the total number of elements offering a quality of ≤ 0.2 is decreased considerably. The lowest occurring element quality is slightly increased, from 0.0885 to 0.1071.



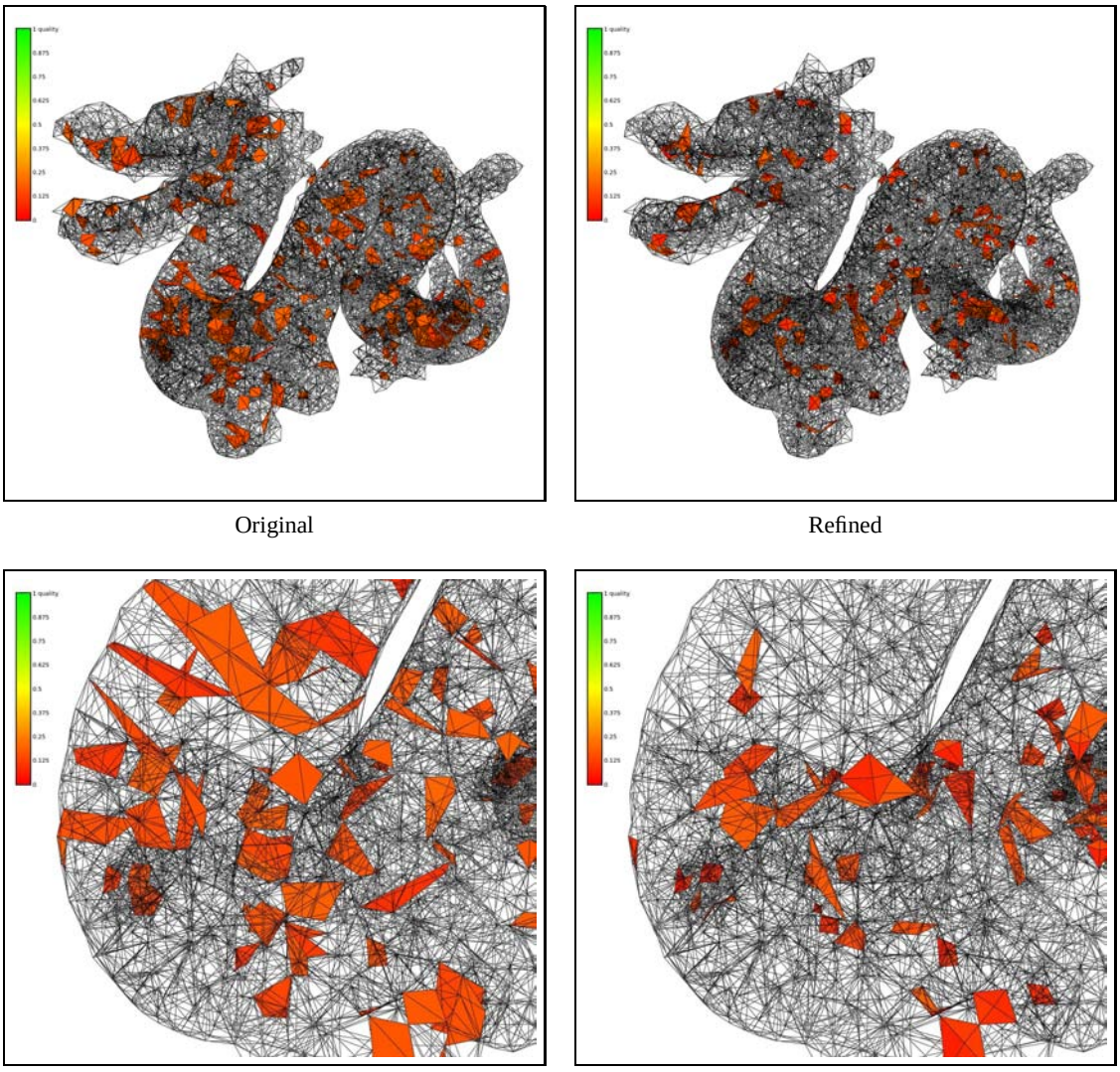
Original Refined

Figure 5.48: Angle distributions in the RAND2 mesh before and after adaption. The number of elements with an angle of 10° has decreased, while the count of elements with an angle of 170° has increased after the adaption. The improvement due to adaption is made explicit by the shift to this intermediate range of angles.



Original Refined

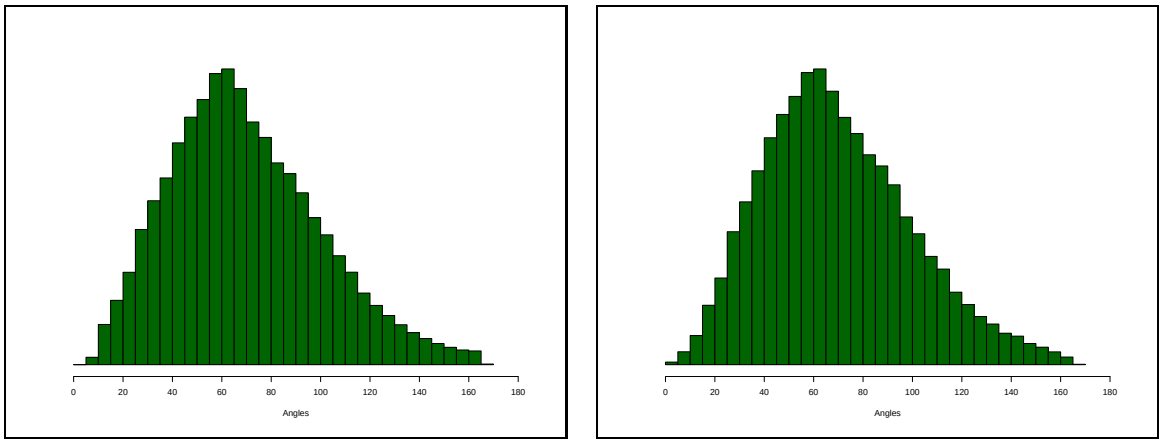
Figure 5.49: Overview of classified element types in the RAND2 mesh before and after adaption. The number of *round* and *sliver* elements has been reduced while the number of *wedges* has been increased by using the adaption.



Original | Refined

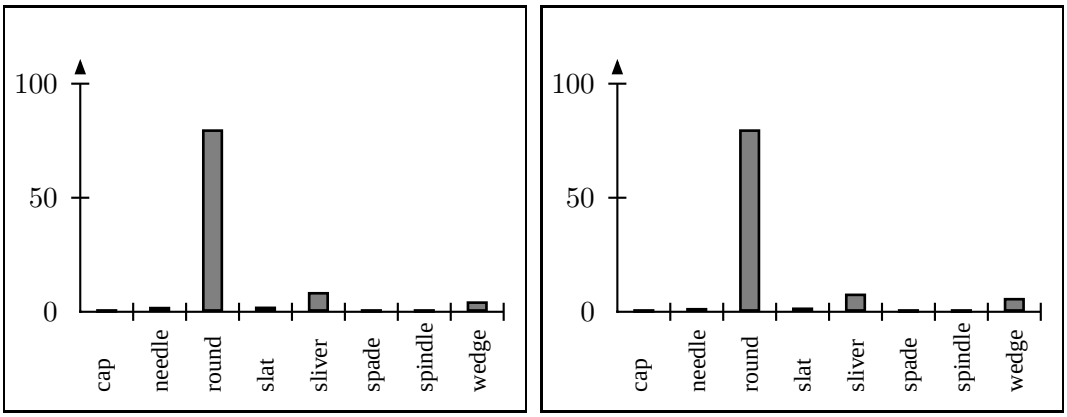
Original - Detailed View | Refined - Detailed View

Figure 5.50: Visualization of the element quality distribution of the DRAGON mesh. Elements with quality ≤ 0.2 are highlighted. Clearly the total number of elements offering a quality of ≤ 0.2 is decreased considerably. The lowest occurring element quality is decreased, from 0.0825 to 0.0069.



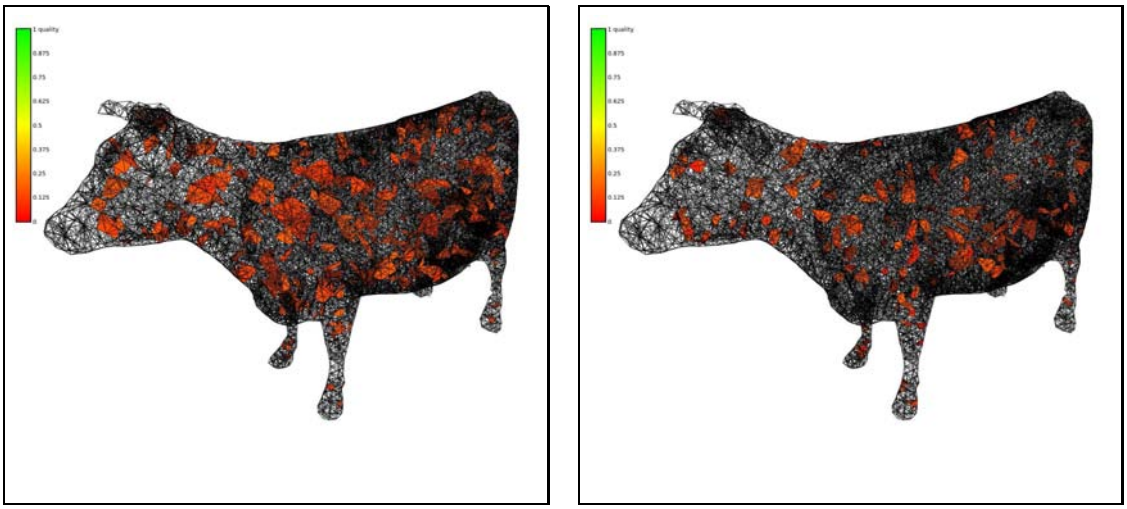
Original Refined

Figure 5.51: Angle distributions in the DRAGON mesh before and after adaption. The number of elements with an angle of $10°$ and $170°$ has decreased, while the count of elements with an angle of $90°$ has increased after the adaption. The improvement due to adaption is made explicit by the shift to this intermediate range of angles. Generally there is no significant change in the angle distribution.

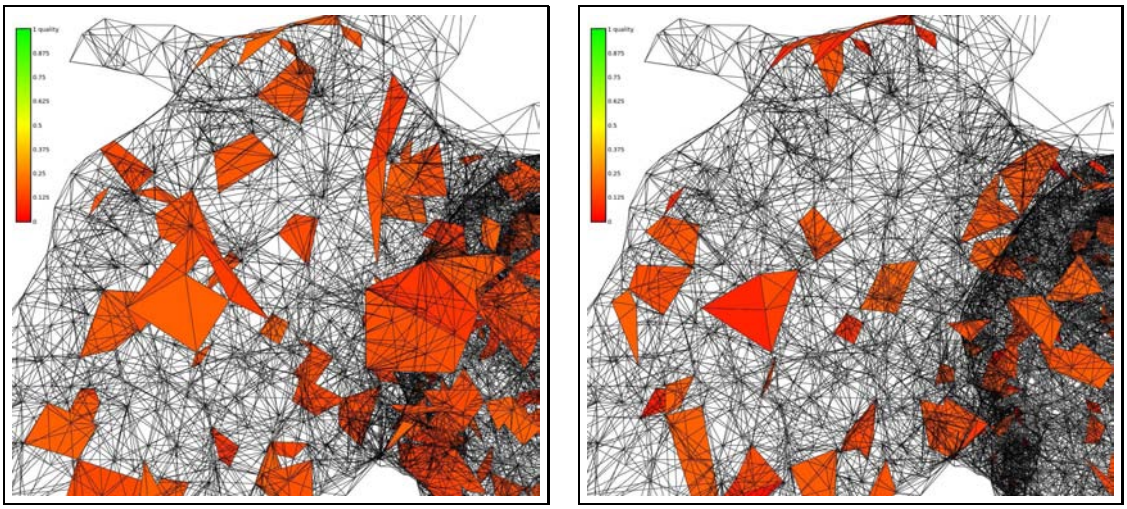


Original Refined

Figure 5.52: Overview of classified element types in the DRAGON mesh before and after adaption. The number of *wedges* has been increased by using the adaption.

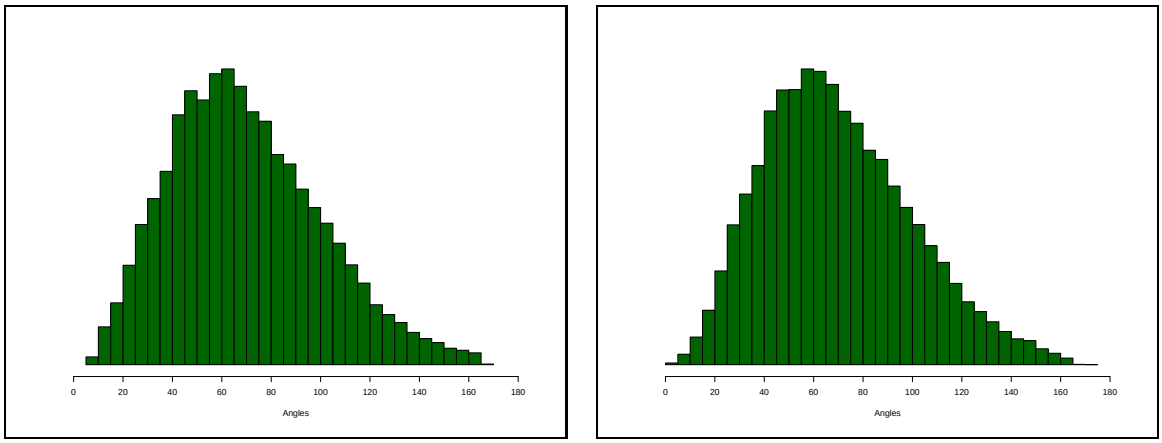


Original | Refined



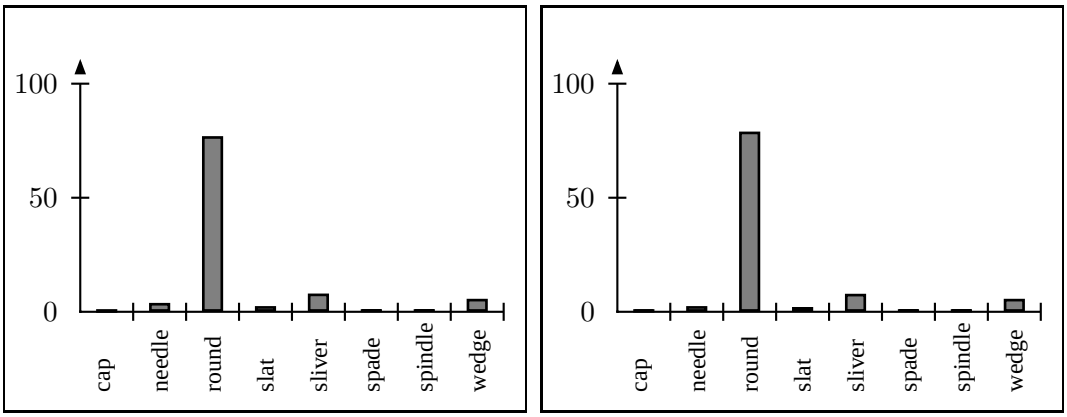
Original - Detailed View | Refined - Detailed View

Figure 5.53: Visualization of the element quality distribution of the COW mesh. Elements with quality ≤ 0.2 are highlighted. Clearly the total number of elements offering a quality of ≤ 0.2 is decreased considerably. The lowest occurring element quality is decreased, from 0.0592 to 0.0103.



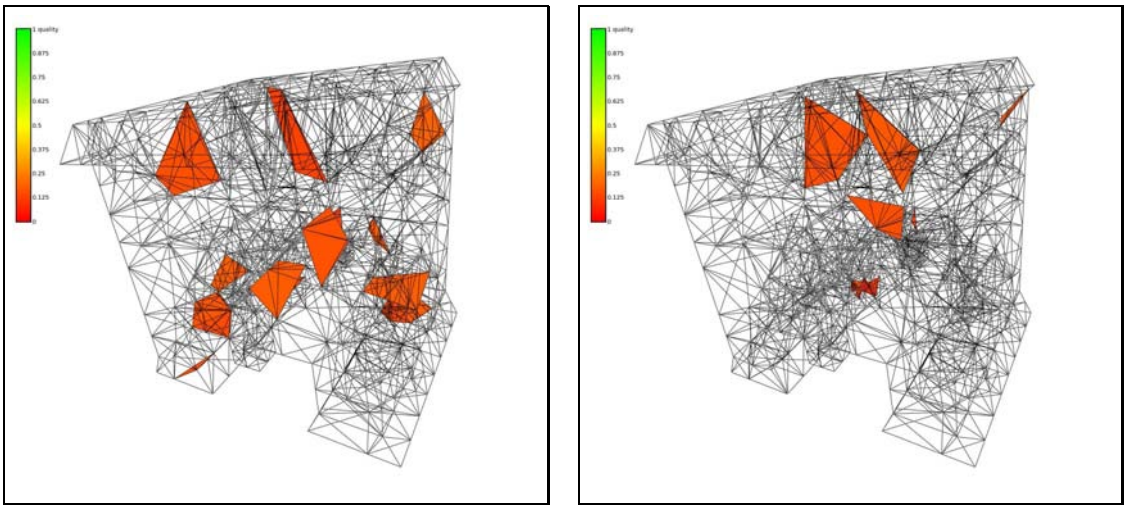
Original Refined

Figure 5.54: Angle distributions in the COW mesh before and after adaption. The number of elements with an angle of $10°$ and $160°$ has decreased, while the count of elements with an angle of $50°$ has increased after the adaption. The improvement due to adaption is made explicit by the shift to this intermediate range of angles.

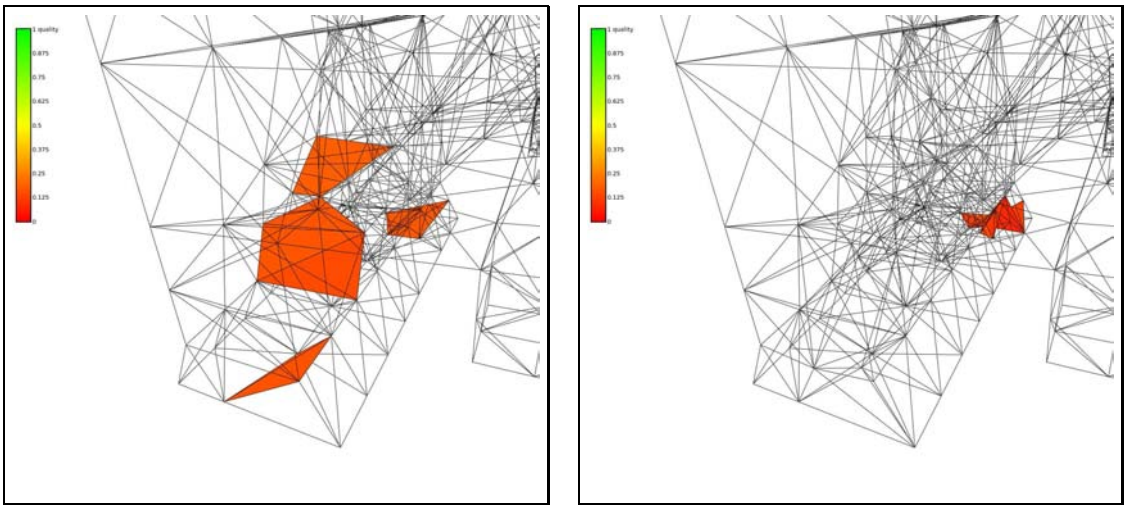


Original Refined

Figure 5.55: Overview of classified element types in the COW mesh before and after adaption. The number of *rounds* has been increased by using the adaption.

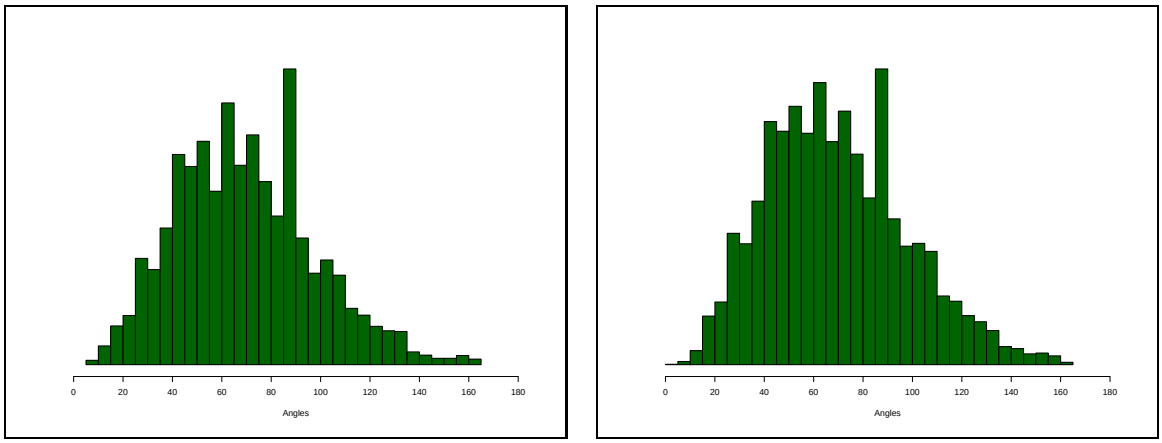


Original | Refined



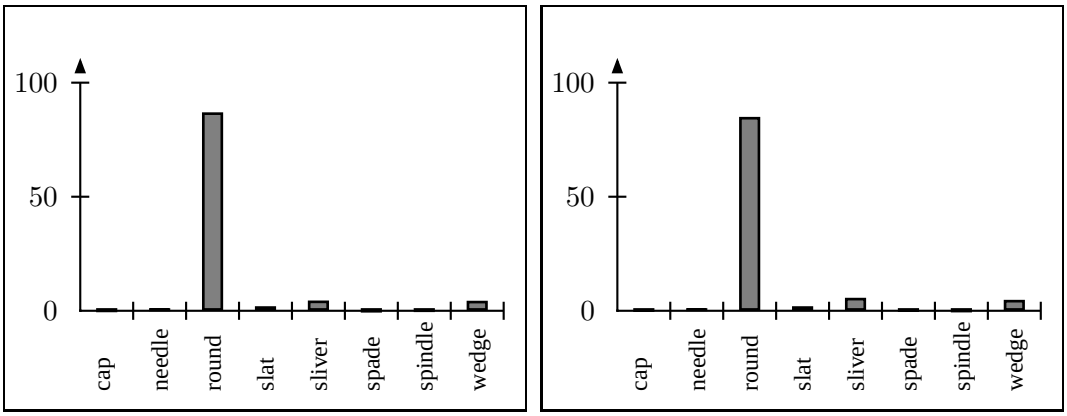
Original - Detailed View | Refined - Detailed View

Figure 5.56: Visualization of the element quality distribution of the HOUSE mesh. Elements with quality ≤ 0.2 are highlighted. Clearly the total number of elements offering a quality of ≤ 0.2 is decreased considerably. The lowest occurring element quality is decreased, from 0.1300 to 0.0791.



Original Refined

Figure 5.57: Angle distributions in the HOUSE mesh before and after adaption. The count of elements containing angles between $40°$ and $120°$ has increased after the adaption. The improvement due to adaption is made explicit by the shift to this intermediate range of angles.



Original Refined

Figure 5.58: Overview of classified element types in the HOUSE mesh before and after adaption. The number of *round* elements has been reduced while the number of *slivers* has been increased by using the adaption.

5.2.4 Conclusion

The major drawback of the implemented mesh adaption approach is the generation of new degenerated elements. This behavior can be clearly recognized in the two and three dimensional application cases. Apparently most of the results reveal that the lowest occurring element quality is lower than in the input mesh. The creation of such highly degenerated elements is mostly associated with the three dimensional adaptions. This suggests an iterative control loop which explicitly tests for such cases and removes them. The removal of such degenerated elements is called *sliver exudation* and has been well investigated [31][32]. However, the focus of the presented implementation is on the generalization of algorithms for mesh adaption applications. Furthermore, the used approach is based on a modular design which enables to conveniently exchange mesh generation engines. This approach therefore facilitates the implementation of new workflows by making it possible to arbitrarily combine different algorithm modules.

The angle distributions were improved by the adaption of each of the examples. The resulting decrease of small and large angles certifies the overall improvement.

In general the number of *round* elements was notably improved. However, at least one kind of degenerated element type was also increased.

The limited quality improvement capabilities of the implemented mesh adaption application can clearly be recognized. However, the applicability of the generic adaption implementation as mesh adaption tool has been proven to be not only feasible but also practically operational.

It is important to note that the adaption implementation is capable of processing two and three dimensional meshes without any change of code. Based on the obtained results more complex adaption algorithms can be implemented which are able to counter or prevent the mentioned drawbacks.

5.3 Mesh Adaption based on Inclusion Tests

This section discusses a mesh intersection application which makes use of the generalized p-*cell in* q-*cell* algorithm introduced in **Section 3.4.2** and the *generic functor queue* discussed in **Section 4.4**.

The application processes two meshes and tests for each 0-cell of the first mesh if it is inside of any of the p-cells of the second mesh. Note that p denotes arbitrary cell dimension, however the implementation is tested for dimensions $p = 2, 3$. Generally, meshes of different topological dimensions can be processed by one consistent implementation. This illustrates the power and versatility of the generalized algorithm approach introduced in **Section 3.4.2**.

Generally, there is no typical intersection test performed. Such tests are based on different approaches [33]. Furthermore available intersection tests[13] offer far better performance, as certain precondition tests are applied which reduce the computational effort.

The first part discusses the implementations details presenting and discussing several code snippets. The second part introduces the used input meshes. The third part discusses the adaption results for 2-simplex and 3-simplex meshes, respectively. The last part sums up the achieved results and concludes by providing possible improvements.

5.3.1 Implementation

The following code snippet depicts the declaration and instantiation of the *cell in cell* object.

```
1 typedef boundary< DIMT, DIMT >                                    TopologyOperation1;
2 typedef boundary< DIMT, 0 >                                       TopologyOperation2;
3 typedef property_data< TopologyOperation1, CellContainer,
4                        TopologyOperation2, CellContainer>         PropertyData;
5 typedef typename cell_in_cell< PropertyData >::type               CellInCell;
6 CellInCell cell_in_cell;
```

Note that the topological dimension (*DIMT*) is automatically derived from the input mesh. In dependence on the dimension the corresponding *boundary* operations are computed at compile time (**Lines 1,2**). The first and second operation relate to the topological decay of the source and target element, respectively. The first operation yields 0-cells[14]. The second operation yields p-cells, where p is the dimension of the cell complex. Note that this complex declaration enables to setup arbitrary *cell in cell* inclusion tests.

The topological operations and the cell container types are gathered to a unified property type (**Lines 3,4**). Note that this property type implements the concept requirements for the generalized algorithms. The actual *cell in cell* object is declared and instantiated (**Lines 5,6**). Note that this object models the *property* concept.

The following code snippet outlines the setup of the *generic functor queue*.

```
1 typedef typename sequence< CellInCell, Size, GreaterThan >::type   FunctorSequ;
2 FunctorSequ functor_sequ(cell_in_cell, size_functor, greater_than);
```

Basically the functor queue reads as follows: Execute the *cell in cell* algorithm, compute the *size* of the result and check if the size is *greater than* a threshold. Note that the cell in cell functor returns all elements of the source cell which are inside the target cell. Consequently, inspecting the number of these resulting elements reveals if an intersection has taken place. For obvious reasons the threshold value of the *greater than* functor is zero, meaning if there are any intersections at all, return true.

[13] Intersection tests are also denoted as *collision* tests.

[14] The second parameter is subtracted from the first parameter to retrieve the resulting dimension. Consequently this operation yields 0 for any case.

The following code snippet depicts the traversal of the elements, the execution of the functor queue and the tagging of the elements.

```
1  long ci = 0;
2  long intersecting = false;
3  for_each( cell_container_1, (
4     let( _a = _1 ) [
5        for_each(
6           ref(cell_container_2),
7           lambda [
8              if_(!ref(intersecting)) [
9                 if_(fold4(functor_sequ, _a, cref(geometry1), _1, cref(geometry2))) [
10                   ref(intersecting)=true
11                ]
12             ]
13          ]
14       ),
15       if_(ref(intersecting)) [
16          at(ref(tag_container), ref(ci)) = 1
17       ]
18       .else_ [
19          at(ref(tag_container), ref(ci)) = 0
20       ]
21    ],
22    ref(ci)++,
23    ref(intersecting)=false
24 ));
```

In the following the previous code fragment is described in detail.

The cells of the source cell container are traversed (**Line 3**). Each cell is temporarily stored (**Line 4**), as the placeholder loses its validity in the upcoming lines. The cells of the target cell container are traversed (**Lines 5,6**). A *Lambda* environment is opened which introduces new placeholders (**Line 7**). A check is performed if the current source cell is already flagged as an intersecting cell (**Line 8**). If this is not the case, apply the functor queue, check the result of the queue and raise the flag accordingly (**Line 9, 10**). If all target cells have been investigated, check if the flag has been raised for the current source cell and register the result within an external tag container by using the source cell access index (**Lines 15-20**).

In own words: Test for each source cell if it is inside of any of the target cells. If the source cell is inside of at least one target cell, flag this cell to be an intersecting cell.

Further note that for each source and target cell, the corresponding geometry containers are forwarded to the *cell in cell* function object (**Line 9**). This approach has been implemented on purpose, as the topology elements may refer to different geometry environments. This approach models the *generic concept*, as arbitrary topology and geometry combinations are enabled.

On a sidenote

Note that the cell intersection tag part (**Lines 15-20**) is not performed by the *functor queue*, as the cell index is lost. This is due to the inner traversal of the target cell container (**Lines 5-14**).

5.3.2 Input Data

In the following the input meshes for the mesh adaption application are introduced.

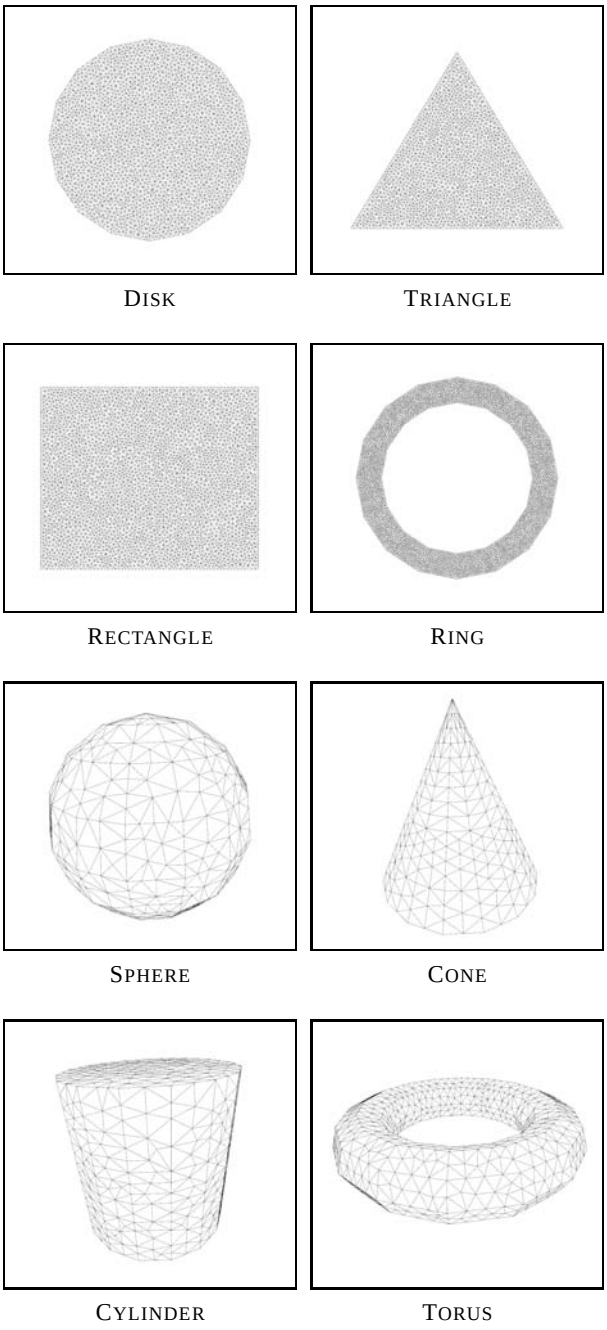


Figure 5.59: Input meshes used for inclusion tests.

- DISK, TRIANGLE, RECTANGLE and RING are meshes generated by utilizing the *generic mesh generation suite* (**Section 4.3**).
- SPHERE, CONE, CYLINDER and TORUS are meshes generated by *Wings 3D* [13] in conjunction with the *generic mesh generation suite*.

5.3.3 Mesh adaptions

This section discusses results of the mesh adaption based on inclusion tests. Several use cases based on simple geometrical entities in two and three dimensions are presented.

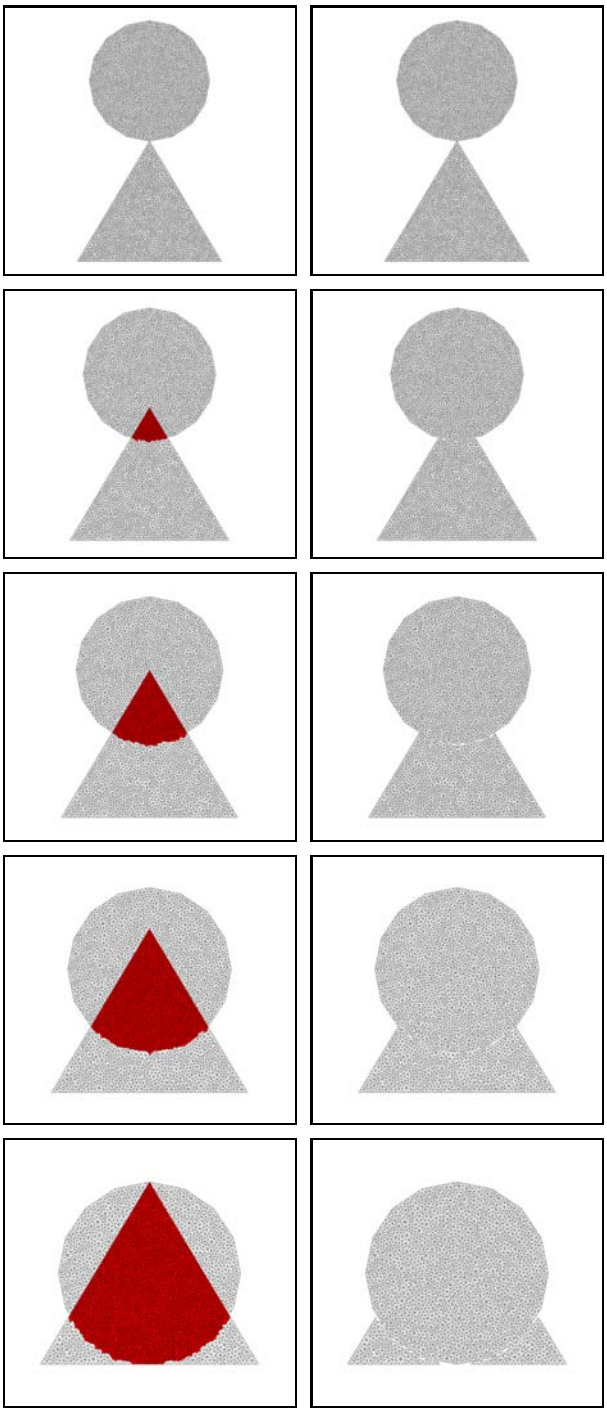


Figure 5.60: Visualization of a basic collision simulation based on the DISK and TRIANGLE mesh. The penetration progresses from **top** to **bottom**. The **left** column depicts the identification of collided cells. The colliding elements are highlighted in **red**. The **right** column outlines the mesh adaption result. All colliding cells are removed.

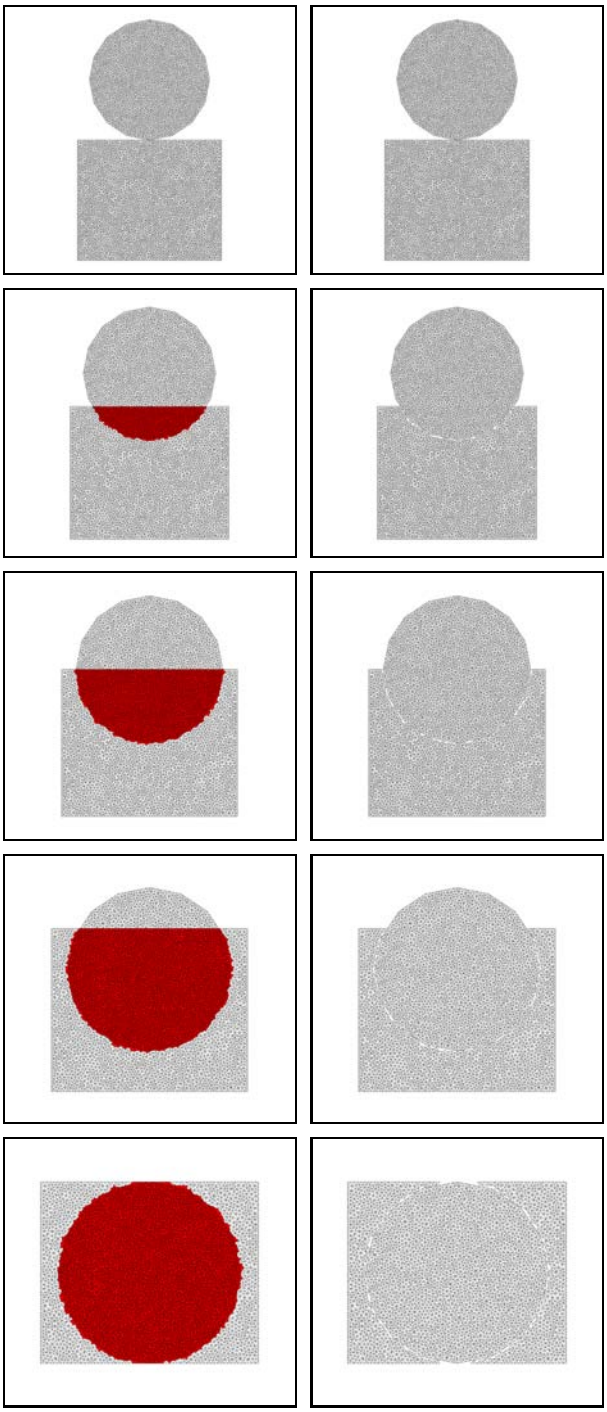


Figure 5.61: Visualization of a basic collision simulation based on the DISK and RECTANGLE mesh. The penetration progresses from **top** to **bottom**. The **left** column depicts the identification of collided cells. The colliding elements are highlighted in **red**. The **right** column outlines the mesh adaption result. All colliding cells are removed.

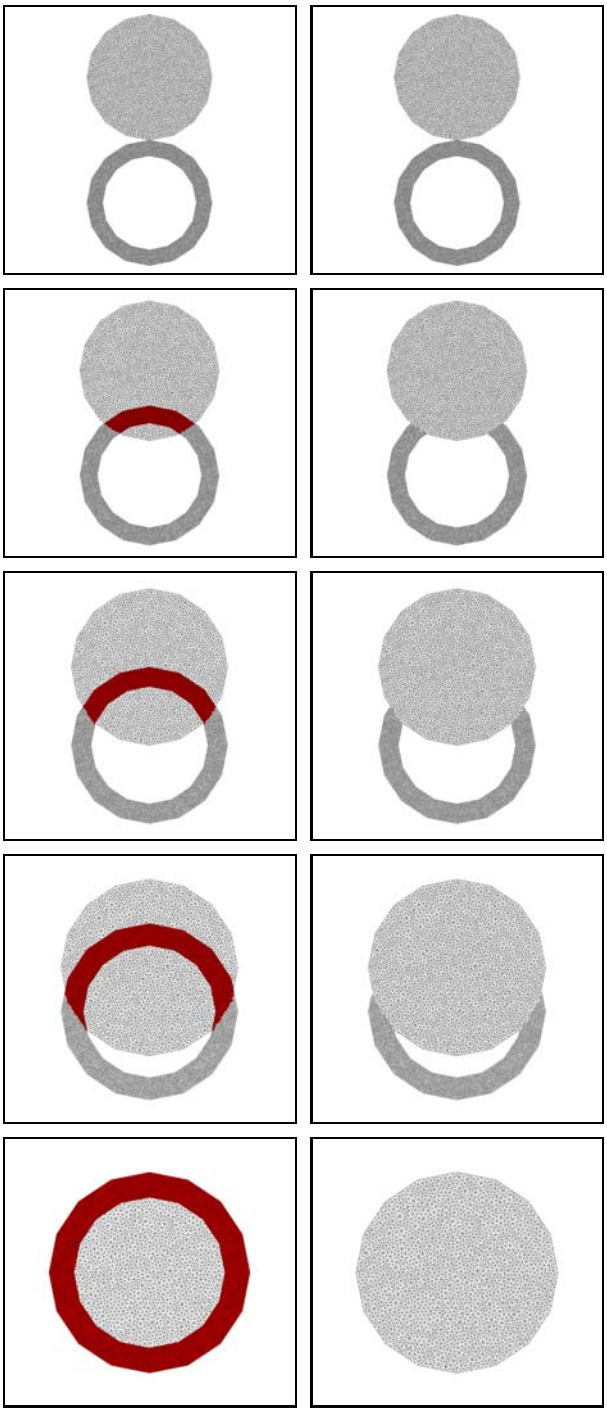


Figure 5.62: Visualization of a basic collision simulation based on the DISK and RING mesh. The penetration progresses from **top** to **bottom**. The **left** column depicts the identification of collided cells. The colliding elements are highlighted in **red**. The **right** column outlines the mesh adaption result. All colliding cells are removed.

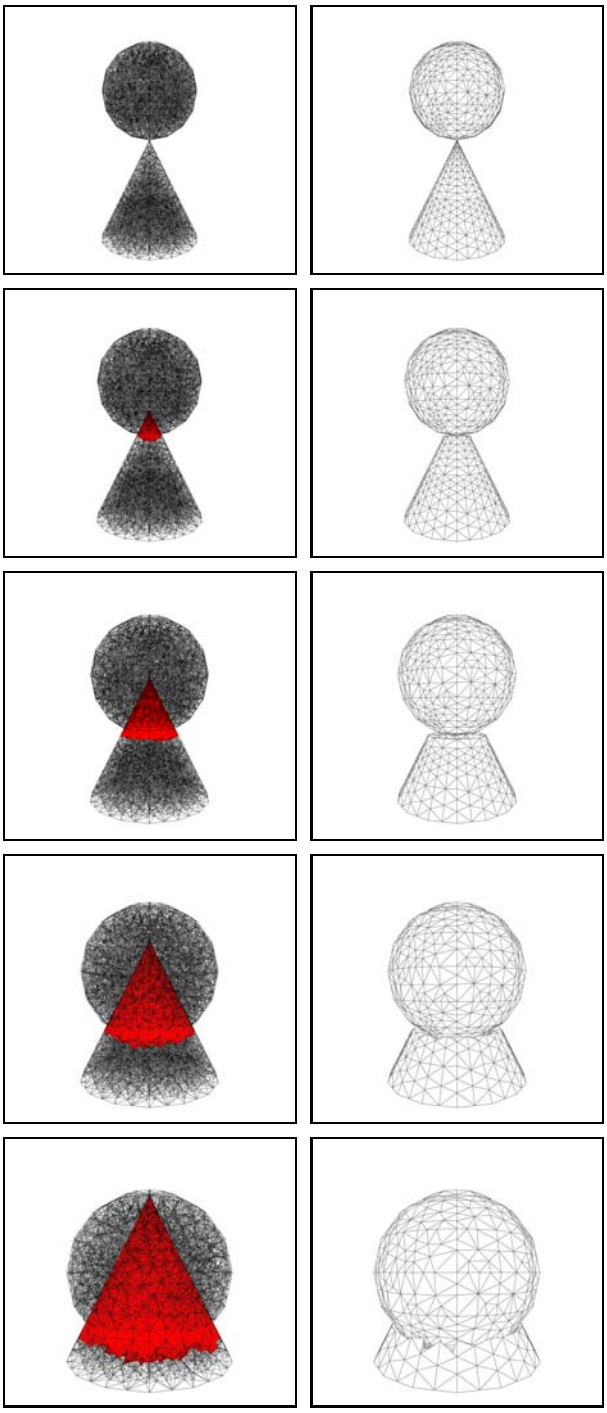


Figure 5.63: Visualization of a basic collision simulation based on the SPHERE and CONE mesh. The penetration progresses from **top** to **bottom**. The **left** column depicts the identification of collided cells. The colliding elements are highlighted in **red**. The **right** column outlines the mesh adaption result. All colliding cells are removed.

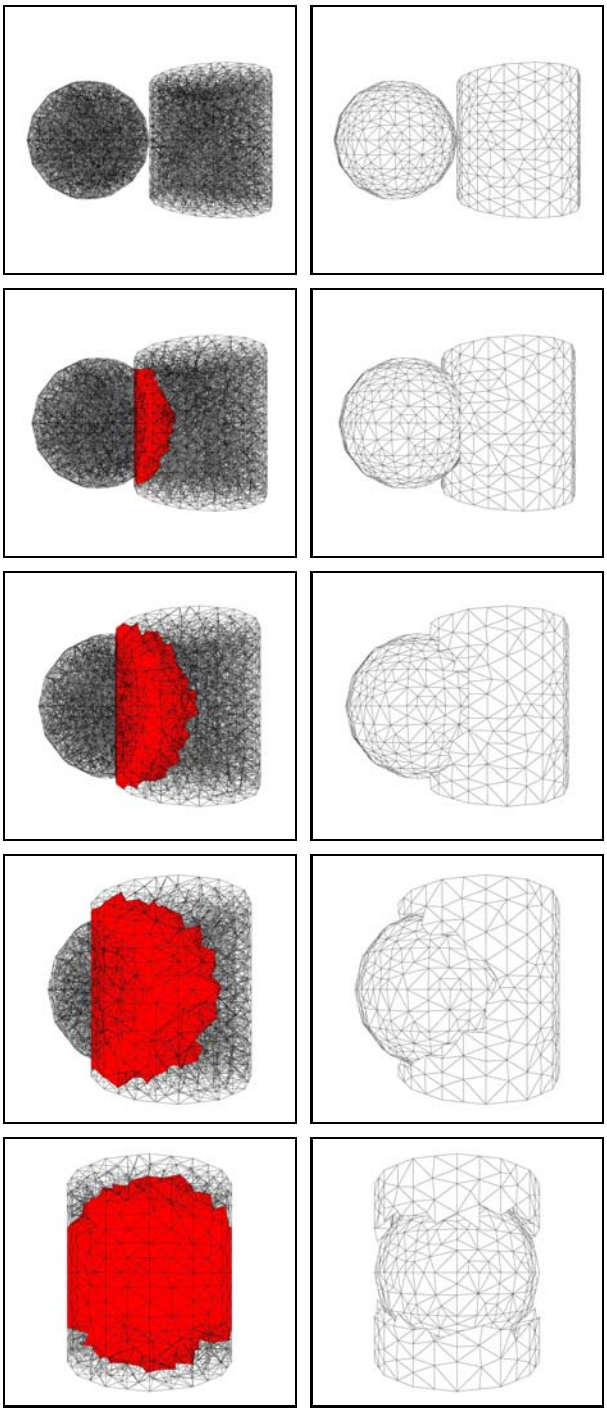


Figure 5.64: Visualization of a basic collision simulation based on the SPHERE and CYLINDER mesh. The penetration progresses from **top** to **bottom**. The **left** column depicts the identification of collided cells. The colliding elements are highlighted in **red**. The **right** column outlines the mesh adaption result. All colliding cells are removed.

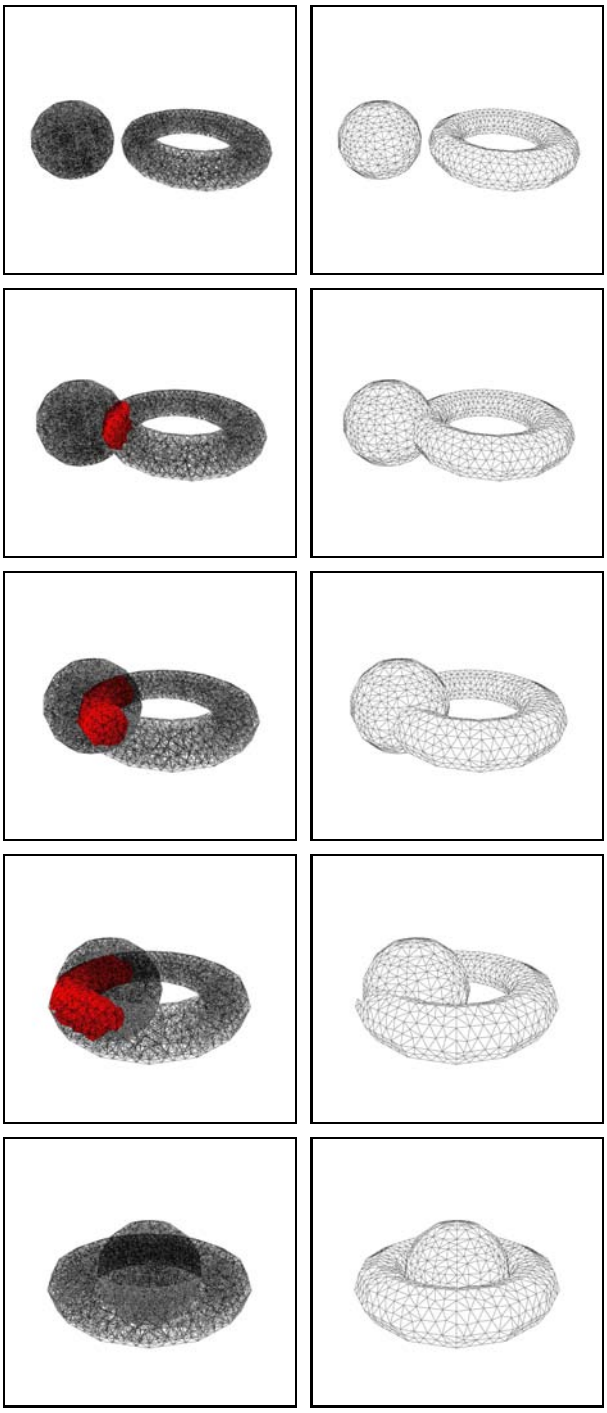


Figure 5.65: Visualization of a basic collision simulation based on the SPHERE and TORUS mesh. The penetration progresses from **top** to **bottom**. The **left** column depicts the identification of collided cells. The colliding elements are highlighted in **red**. The **right** column outlines the mesh adaption result. All colliding cells are removed.

5.3.4 Conclusion

The introduced results have been generated by a single generic implementation. The application derives the dimension from the input meshes and applies the corresponding specializations. The specializations are precomputed by the compiler using the meta programming facilities of the *Boost* libraries. Therefore no runtime decisions are made regarding mesh type specializations.

The presented inclusion algorithm does not offer high performance as a basic approach is used. Each cell of one object is tested if it is inside of a cell of the second object. No preprocessing algorithm is applied to ease the computational effort[15]. However, the presented algorithm depicts the functionality of a generic inclusion algorithm capable of processing two and three dimensional cell complexes.

The introduced basic implementation can be improved further by extending the support to other cell complex types and dimensions. Furthermore, preprocessing algorithms should be added to ease the burden of computation.

[15] Such as geometrical predicates which provide a preliminary decision whether or not two cells are in close proximity.

Chapter 6

Summary and Outlook

An approach to generalize typical algorithms in the field of adaptive mesh generation has been presented. This generalization enables the application of such algorithms to new areas. Furthermore, code reusability is greatly improved due to the use of modern programming techniques based on C++ and the Boost libraries. The applications which are based on the generalized algorithm approach demonstrate that the procedure yields operational results. The advantages and disadvantages have been discussed as well as further improvements.

An algorithm for the simplification of meshes based on image data has been introduced and discussed in detail, providing not only implementation details but also showing achieved results.

The discussed generalization approach as well as the image based simplification algorithm are a solid foundation for future work. The existing algorithm implementations should be revisited and reevaluated with regard to run time efficiency. The image simplification algorithm should be investigated concerning an extension regarding the treatment of three dimensional entities based on several layers of image.

Furthermore, new algorithms should be determined for generalization and the resulting collection of algorithms should be integrated into a concise and generic environment.

Additional mesh generation engines should be added to the generic mesher suite. In the following the concrete steps are summarized.

- revisit existing algorithm implementation
- image layer based three dimensional mesh generation
- add additional generalized algorithms, i.e., cell angles
- add additional mesh generation engines, i.e., Netgen [34]
- provide generic implementations by a generic environment

Generally, a highly reusable framework of generalized algorithms as well as an extended unified mesh generation toolkit should be the focus of future work. The goal is to implement a generic environment which provides mesh generation and adaption algorithm modules, thusly combining two originally different fields: mesh adaption and mesh generation.

Bibliography

[1] J. R. Shewchuk, “Delaunay Refinement Mesh Generation,” Ph.D. dissertation, School of Computer Science, Carnegie Mellon University, May 1997.

[2] M. Bern and D. Eppstein, “Mesh Generation And Optimal Triangulation,” 1992.

[3] M. Bern and P. Plassmann, “Mesh generation,” in *Handbook of Computational Geometry. Elsevier Science*, 2000, pp. 291–332.

[4] B. M. Klingner and J. R. Shewchuk, “Agressive tetrahedral mesh improvement,” in *Proceedings of the 16th International Meshing Roundtable*, Oct. 2007, pp. 3–23. [Online]. Available: http://www.eecs.berkeley.edu/b-cam/Papers/Klingner-2007-ATM/index.html

[5] L. Simonson and G. Suto, “Geometry Template Library for STL-like 2D Operations,” BoostCon’09, May 2009.

[6] R. Heinzl, “Concepts for Scientific Computing,” Ph.D. dissertation, Vienna University of Technology, August 2007.

[7] “Boost C++ Libraries, BOOST.” [Online]. Available: http://www.boost.org

[8] “Generic Scientific Simulation Environment, GSSE.” [Online]. Available: http://www.gsse.at

[9] “Triangle.” [Online]. Available: http://www.cs.cmu.edu/~quake/triangle.html

[10] “Computational Geometry Algorithms Library, CGAL.” [Online]. Available: http://www.cgal.org

[11] “TetGen.” [Online]. Available: http://tetgen.berlios.de

[12] “Adobe.” [Online]. Available: http://www.adobe.com/

[13] “Wings 3D.” [Online]. Available: http://www.wings3d.com/

[14] H. Edelsbrunner, M. J. Ablowitz, S. H. Davis, E. J. Hinch, A. Iserles, J. Ockendon, and P. J. Olver, *Geometry and Topology for Mesh Generation (Cambridge Monographs on Applied and Computational Mathematics)*. New York, NY, USA: Cambridge University Press, 2006.

[15] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Second Edition*, 2001.

[16] A. Nakamoto and K. Ota, “Note on irreducible triangulations of surfaces,” *J. Graph Theory*, vol. 20, no. 2, pp. 227–233, 1995.

[17] P. Liepa, “Filling holes in meshes,” in *SGP ’03: Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2003, pp. 200–205.

[18] R. Heinzl and T. Grasser, “Generalized Comprehensive Approach for Robust Three-Dimensional Mesh Generation for TCAD,” in *International Conference on Simulation of Semiconductor Processes and Devices 2005*, 2005, pp. 211–214.

[19] F. Stimpfl, R. Heinzl, P. Schwaha, M. Spevak, and T. Grasser, “Multi-mode Mesh Generation for Scientific Computing,” in *The 2007 European Simulation and Modelling Conference Proceedings*, 2007, pp. 425–429.

[20] Q. Du and D. Wang, “Boundary recovery for three dimensional conforming Delaunay triangulation,” *Computer Methods in Applied Mechanics and Engineering*, vol. 193, pp. 2547–2563, 2004.

[21] W. Wessner, “Mesh Refinement Techniques for TCAD Tools,” Ph.D. dissertation, Vienna University of Technology, November 2006.

[22] E. Schönhardt, “Über die Zerlegung von Dreieckspolyedern in Tetraeder,” 1928.

[23] V. A. Kovalevsky, “Finite topology as applied to image analysis,” *Comput. Vision Graph. Image Process.*, vol. 46, no. 2, pp. 141–161, 1989.

[24] J. R. Shewchuk, “What is a Good Linear Element? Interpolation, Conditioning, and Quality Measures,” in *11th International Meshing Roundtable*, 2002, pp. 115–126.

[25] ——, “Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator,” in *Applied Computational Geometry: Towards Geometric Engineering*, ser. Lecture Notes in Computer Science, M. C. Lin and D. Manocha, Eds. Springer-Verlag, May 1996, vol. 1148, pp. 203–222, from the First ACM Workshop on Applied Computational Geometry.

[26] “Shenteq.” [Online]. Available: http://www.shenteq.com/

[27] “Vienna University of Technology, TU Vienna.” [Online]. Available: http://www.tuwien.ac.at

[28] “Wikimedia Commons.” [Online]. Available: http://commons.wikimedia.org

[29] “GNU Free Documentation License, GFDL.” [Online]. Available: http://www.gnu.org/copyleft/fdl.html

[30] “Openmp.” [Online]. Available: http://openmp.org

[31] H. Edelsbrunner and D. Guoy, “An Experimental Study of Sliver Exudation,” in *10th International Meshing Roundtable, Sandia National Laboratories*, October 2001, pp. 307–316.

[32] S.-W. Cheng, T. K. Dey, H. Edelsbrunner, M. A. Facello, and S.-H. Teng, “Sliver exudation,” in *SCG ’99: Proceedings of the fifteenth annual symposium on Computational geometry*. New York, NY, USA: ACM, 1999, pp. 1–13.

[33] O. Tropp, A. Tal, and I. Shimshoni, “A fast triangle to triangle intersection test for collision detection,” in *Computer Animation and Virtual Worlds*, 2006, vol. 17, no. 5, pp. 527–535.

[34] “Netgen.” [Online]. Available: http://www.hpfem.jku.at/netgen/