UNIVERSITÀ DEGLI STUDI DI MILANO
FACULTY OF SCIENCE AND TECHNOLOGY

COMPUTER SCIENCE DEPARTEMENT
GIOVANNI DEGLI ANTONI

Master's degree course in
Computer Science

EFFICIENT REPRESENTATIONS OF HIGH-RESOLUTION POLYGONAL SURFACES: ADDING ANISOTROPY CONTROL TO THE MICRO-MESHES SCHEMA

Supervisor: Prof. Marco Tarini

Master's thesis of:
Manuel Pagliuca
Matr. Nr. 975169

ACADEMIC YEAR 2022-2023

Ringraziamenti

Prima di immergerci nell'approfondito lavoro svolto in questi mesi, desidero esprimere la mia profonda gratitudine a diverse persone.

In primis vorrei ringraziare il mio relatore, il Prof. Marco Tarini, quest'ultimo è stato un punto di riferimento durante lo sviluppo di questo progetto. La sua profonda conoscenza ed esperienza sono stati di grande aiuto nei momenti di incertezza e smarrimento.

Desidero esprimere profonda gratitudine ai miei cari amici e compagni di corso, i quali sono stati sempre al mio fianco, offrendo sostegno e distrazioni quando ne avevo bisogno.

Un ringraziamento speciale va alla mia famiglia e alla mia fidanzata. Siete stati al mio fianco in ogni istante di questa sfida, e vi sono grato per il vostro costante sostegno, anche nei momenti più difficili. In particolare ringrazio i miei genitori, senza i vostri sacrifici tutto questo non sarebbe stato possibile. Avete sempre creduto in me anche quando le situazioni non erano delle più rosee, non vi sono parole per esprimere quanto vi sia grato.

Un sentito ringraziamento è dovuto a tutte le persone che generosamente condividono il loro sapere pubblicamente attraverso la rete. Anche se la maggior parte di loro rimane nell'ombra, il vostro contributo ha giocato un ruolo cruciale nel plasmare il mio percorso accademico e mi ha ispirato a restituire il gesto.

Concludendo, questo lungo percorso di studi è stato reso speciale dalle tante relazioni personali di cui ho il privilegio di vivere. A tutti voi voglio dirvi grazie.

Grazie
Manuel Pagliuca

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The evolution of 3D computer graphics has led to a continuous escalation in the intricacy of models, thereby necessitating technologies that can ensure judicious resource utilization. This imperative arises from the need to circumvent resource overconsumption. This thesis is driven by the goal of undertaking an empirical exploration into the potential efficacy of new data structures capable of adeptly representing 3D polygonal surfaces with extraordinary geometric resolutions. The focus is on enabling **multi-resolution** rendering on GPUs, a context demanding both efficiency and performance.

1.1 Motivations

In the realm of **geometry processing**, the pursuit of an optimal data structure capable of enhancing extreme geometric precision without commensurately inflating memory utilization, specifically video RAM, and computational overhead, has been a prominent objective. This thesis is centered on the conceptualization and construction of a variant data structure termed "*anisotropic μ-mesh*" to the already existing one "*μ-mesh*" by implementing a different subdivision scheme. These data structures receive two meshes of the same model at low and high polygonal resolution, respectively. Starting from the low-resolution one, they are able to approximate the second one through an appropriate **vertex displacement** performed on an intermediate subdivided mesh of the low-resolution model. The overarching aim is to realize the aforementioned goal of markedly heightening the geometric accuracy of vertex displacements and enabling *multi-resolution* rendering while reducing the memory allocation footprint.

1.2 Context of the study

Geometry processing, is a research field that utilizes applied mathematics concepts to design algorithms for acquiring, reconstructing, and analyzing 3D models. It plays a critical role in various domains like computer graphics, virtual reality, medical imaging, and robotics. With the increasing demand for 3D content, efficient manipulation of geometric data becomes essential. Geometry processing researchers employ advanced mathematical techniques, including differential geometry, optimization, and numerical methods, to address challenges associated with complex 3D structures represented as meshes. These challenges involve tasks like surface reconstruction, mesh denoising, simplification, parameterization, shape correspondence, and deformation. By developing robust algorithms and innovative methodologies, geometry processing drives advancements in 3D modeling and analysis, benefiting diverse fields and industries.

1.3 Objectives and contributions

The central aim of this thesis is to establish the viability of an *alternative* **subdivision scheme** as a functional counterpart to the established μ-mesh scheme from the paper "Micro-Mesh Construction" [Maggiordomo et al., 2023]. The μ-mesh scheme utilizes two distinct input meshes of the same model: a **base mesh** characterized by relatively *low* polygonal resolution and a **target mesh** featuring significantly *higher* polygonal complexity. Through a thorough investigation, this thesis seeks to demonstrate the applicability and effectiveness of the proposed alternative scheme. The μ-mesh scheme, with its unique methodology, subdivides the base mesh using a construction algorithm, augmenting the geometrical structures within the original base mesh to introduce new vertices ("*μ-vertices*") and faces ("*μ-faces*"), generating the **subdivided mesh**. These augmented structures play a key role in computing displacement information towards the target mesh.

Consequently, the μ-mesh scheme achieves the creation of a *high-fidelity representation* of the target, originating from a low-resolution version of the same model. This capability holds significant implications for graphical applications by optimizing memory usage and enabling real-time management of various levels of detail, without the need for generating different resolution meshes of the same model. This, in turn, effectively mitigates the undesirable "*popping*" effect commonly observed during transitions between detail levels.

In contrast, this thesis explores an alternative partitioning scheme, known as the anisotropic scheme ("*anisotropic μ-mesh*"). The design of the new subdivision scheme aims to provide **control of the anisotropy** of base meshes, that is, the handling of *shortened* and *elongated* faces (Fig. 7b) during the subdivision. The

new scheme differs from the classical scheme in that the latter does not control anisotropy and favors base meshes with *equilateral* faces.

This means that in order to achieve a uniform mesh subdivision, with the classical method one must use a mesh with faces as equilateral as possible. Imposes a *preprocessing* step to create base meshes with equilateral faces, a nontrivial and time-consuming goal to achieve. Controlling anisotropy allows one to start from *less-than-perfect* meshes and correct them during subdivision in order to create more equilateral triangles in the **subdivided mesh** (yellow colored portion) and as a consequence a better **displacement** (green colored portion).

The primary focus of this thesis is to conduct an exhaustive comparative analysis (Chapter 6) to rigorously evaluate the error between the two subdivision schemes. This analysis aims to establish the feasibility of adopting the anisotropic μ-mesh as a valid alternative to the μ-mesh scheme, considering its potential advantages.

1.4 Overview

The thesis follows an organized format, consisting of sequential chapters that systematically delve into various aspects of the research and implementation of the data structure. The following bulleted list describes the chapters that will be addressed throughout the thesis:

- **Chapter 1, Introduction**, review of the contemporary state-of-the-art technologies similar to the one under study, and introduces the field of research, establishing the basis for understanding the context and central significance of the problem addressed in the thesis.

- **Chapter 2, State of the art**, detailed exposition of the data structure currently available in the state of the art, elucidating the intricate methodology behind the construction of the μ-meshes and outlining the salient processes inherent in its construction and function.

- **Chapter 3, Micro-Mesh: current schema**, before the implementation of the new schema, it is necessary to re-implement the current schema in the project to make the comparison later. To make the structure of the μ-meshes clearer to the reader, the terminology, partitioning scheme, and objectives of the data structure are discussed extensively in this chapter.

- **Chapter 4, Anisotropic Micro-Mesh: a new proposal**, details the terminology, construction design, and implementation steps, intrinsic features, and challenges of the anisotropic subdivision scheme construction process.

- **Chapter 5, Methodology**, description of project structural composition, graphical user interface, algorithmic optimizations, and all those components included in the development environment that enabled the realization of this project. It will act much like a user's manual guiding the use of the various programs involved in the project.

- **Chapter 6, Empirical analysis**, empirical comparison between the two subdivision schemes. This comparison is executed through the utilization of three different comparison metrics, which effectively showcase the relative performance of the new scheme to the other.

- **Chapter 7, Conclusions**, zenith of the thesis, this chapter formulates definitive conclusions concerning the efficacy and efficiency of the devised data structure in fulfilling the objectives elucidated at the commencement of the research. The thesis thus furnishes a coherent and carefully conducted analysis of the subject matter, thereby delivering profound insights and contributing significantly to the domain of study.

Chapter 2

State of the art

Micro-Meshes are a new **structured graphics primitive** supporting a large increase in *geometric fidelity*, without commensurate memory and run-time processing costs, consisting of a *base* mesh enriched by a *displacement map*.

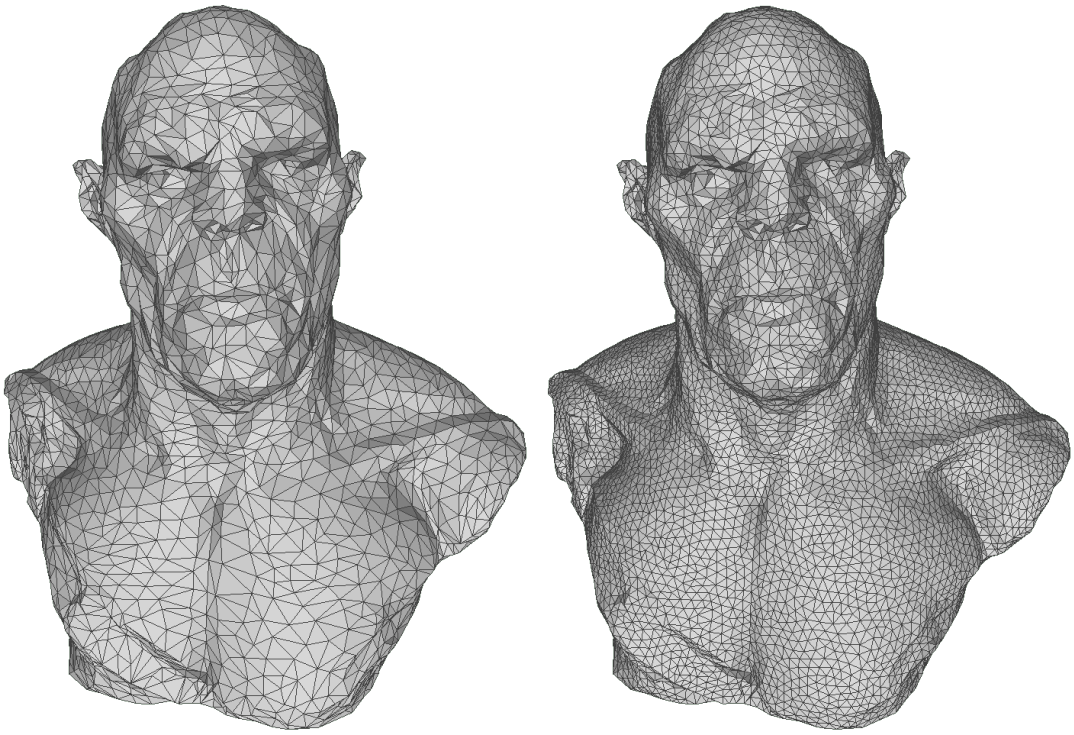
A new generation of GPUs supports this structure with native hardware μ-mesh *ray-tracing*, that leverages a self-bounding, compressed displacement mapping scheme to achieve these efficiencies.

The approach could be considered as a particular type of **remeshing** algorithm, which specifically produces a high-quality subdivided base mesh, along with a displacement map, which can be consumed directly by GPUs. This offers BVH (Bounding Volume Hierarchy) savings and good tracing performance, such a combination lowers the barrier of rendering *highly detailed* geometries in interactive and *real-time graphics* applications.

This process of subdividing the mesh into a more complex geometric scheme and extracting displacements concerning the target mesh will serve as the foundation for the study of the anisotropic variant scheme.

2.1 Remeshing and mesh simplification

Remeshing [Khan et al., 2022] and **mesh simplification** [Luebke, 2001] are two kinds of techniques used in the field of computer graphics to manipulate 3D polygonal networks, commonly called meshes.



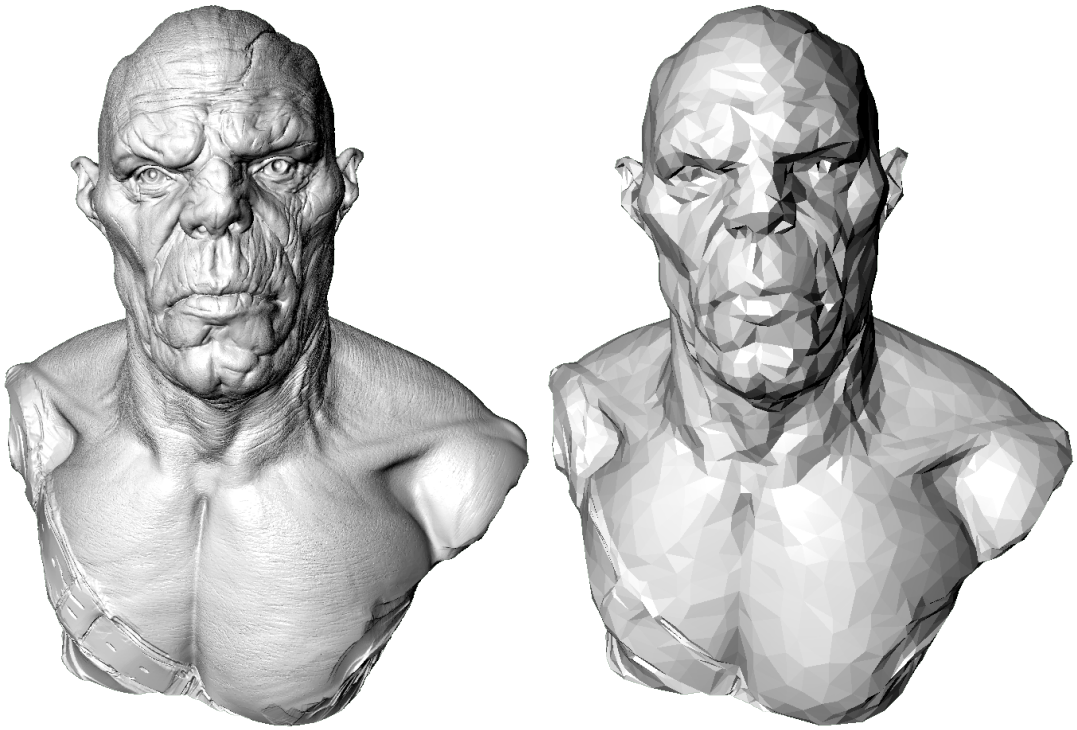
(a) Original (b) Remeshed

Figure 1: Comparison of original mesh topology and the remeshed version.

The first technique pertains to the process of reconstructing or reconfiguring a mesh to enhance its quality or modify its topology. This approach is frequently employed to *eliminate irregular* or *undesired* attributes from a mesh, *enhance* the distribution of polygons, or *adjust* polygon density across different regions of the mesh.

In the model [FRKN, 2023] depicted in Figure 1b a specific type of remeshing called "Isotropic Explicit Remeshing" [Hoppe et al., 1993] occurred. This special type of remeshing repeatedly applies edge flip, collapse, relax, and refine operations to regularize the size and aspect ratio of the triangular meshing (trying to achieve more isotropic triangles).

Remeshing can also serve to transform a mesh with intricate topology into a simpler counterpart, thereby facilitating subsequent processing or animation. This procedure addresses the challenge of attaining an improved discrete representation of the underlying surface within a 3D mesh.



(a) Original (1,500,000 faces) (b) Decimated (7,500 faces)

Figure 2: Comparison of original mesh and the decimated one through.

Mesh simplification, as the term implies, denotes the procedure of diminishing the intricacy of a mesh while retaining its principal attributes. This methodology is frequently employed to curtail the count of polygons within a mesh, all the while upholding its visual essence.

Mesh simplification finds considerable utility in real-time applications, particularly in video games, where complex 3D models require simplified versions for less-performing hardware.

2.2 Displacement mapping

Displacement mapping [Cook, 1984] is a computer graphics technique, which belongs to the family of **texture mapping** techniques. It employs a **texture** or **height map** to induce changes in the geometric positions of mesh points, aligning them with the corresponding points on the textured surface.

While displacement is commonly applied along the local normal to the surface, it's important to note that this is not a strict rule. In fact, the technique exhibits various implementations depending on the specific context.

The term "*mapping*" in this technique alludes to the use of a texture map for modulating the strength of displacement, with the displacement direction typically following the local surface normal.

In contemporary computer graphics, many rendering engines support **programmable shaders**, enabling the creation of high-quality *procedural* textures and patterns at arbitrarily high frequencies. Consequently, the use of the term "mapping" becomes debatable, as it no longer involves the use of a traditional texture map.

The term **displacement** is used to refer to a *super* concept that also includes displacement based on a texture map. The generation of these displacement values, coupled with their storage in a texture image, can be likened to a form of texture baking [Cignoni et al., 1999].

The displacement values themselves can manifest as either vector or scalar data. While scalar data offer greater computational efficiency, they necessitate the development of intricate problem-solving strategies. Two types of displacement mapping can be discerned:

- **Geometric displacement** acts directly on polygon mesh points, this type of displacement requires a *high level of tessellation* of the mesh to produce good results, thus has the disadvantage of producing very *heavy* and difficult-to-manage models.

- **Micro-displacement** it automatically generates a substantial number of small triangular faces (even many millions) and is capable of producing *highly detailed models*. The distinctive and significant advantage of this system lies in the fact that model *tessellation* occurs exclusively during rendering, without altering the fundamental geometry, allowing it to remain straightforward.

2.3 Level of detail

In computer graphics, **level of detail** (LOD) refers to the practice of representing objects or data differently depending on their distance from the observer or the context in which they are used. The LOD can be decreased as the model moves away from the viewer or according to other metrics such as object relevance, viewpoint-relative speed, or position.

LOD is an *optimization* technique for increasing the efficiency of rendering by decreasing the workload on the graphic pipeline, and resource (such as computing power or bandwidth), and enhancing user experience by ensuring that the objects or data being displayed are adequately detailed only when necessary. Real-world applications usually employ specialized methods tailored to the information being rendered.

Depending on the context, two main methods are used:

- **Discrete Levels of Detail** (DLOD), involves creating multiple, discrete versions of the original geometry with decreased levels of geometric detail at run-time, the fully detailed models are substituted with the models of reduced detail as necessary. Due to the discrete nature of the levels, the "*popping*" effect may occur during the model swapping.

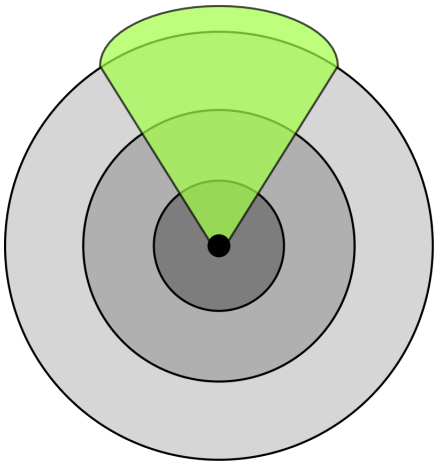


Figure 3: Darker areas are rendered with higher polygon detail.

- **Continuos Levels of Detail** (CLOD), uses a structure that contains a continuously variable spectrum of geometric detail. The structure can then be propped to smoothly choose the appropriate level of detail required for the situation. A significant advantage of this technique is the ability to locally vary the detail (on the model).

- **View-Dependent Level of Detail** (VLOD), uses the parameters of the current view to more accurately represent the quality of the models from the viewer's point of view. A single object may span several levels of detail, it is a selective refinement of CLOD. It shows nearby portions of the object at a higher resolution than distant portions. VLOD has a better granularity than CLOD because it allocates polygons where they are most needed.

- **Hierarchical Levels of Detail** (HLOD), the different DLODs of a model will be grouped into a hierarchy and can create better scalability for large structure models. Then, during the intersection test between the ray originating from the viewer and the object, the hierarchical structure of the intersected model will be accessed, within which the distance of the ray is used as a key to retrieve the DLOD involved.

Chapter 3

Micro-mesh: current schema

As introduced in Chapter 2, μ-meshes are a compact format that is directly consumed on GPU's hardware (which will natively support the format) for representing extremely detailed geometries. This format consists of a special type of *subdivided* mesh and a set of scalar displacement values modeling the high-frequency details.

3.1 Terminology

In this subsection, we provide key terminology that is essential for gaining a clear understanding of the concepts related to **μ-meshes**. These terms serve as the fundamental building blocks of our discussion.

- **Base mesh** is the low-resolution *input* mesh on which the subdivision scheme is applied, it is a much more coarse version of the *target mesh*. Each face is divided into a grid of smaller triangles called μ-faces, the distribution of triangles will change according to the subdivision scheme.
- **Target mesh** is the high-resolution *input* mesh that we aim to approximate starting from the subdivided base mesh through the displacements ("*μ-displacements*") of the new vertices ("*μ-vertices*").
- **Micro-displacements**, at each μ-vertex obtained by the subdivision, the base mesh positions and direction vectors are barycentrically interpolated. The *direction vector* is scaled by a per-μ-vertex **scalar displacement** value and added to the interpolated position generating the displaced μ-vertex.
- **Micro-vertices**, consists of the new vertices generated by the algorithm which subdivided the base mesh according to the μ-mesh scheme.

- **Micro-faces**, they are the new μ-triangles generated by the connection of the previously generate μ-vertices by the execution of the subdivision algorithm.

3.2 Subdivision scheme

The method is based on a simplification scheme tailored to the generation of high-quality *base meshes*, optimized for tessellation and displacement sampling, in conjunction with algorithms for determining *displacement vectors* to control the direction and range of displacements.

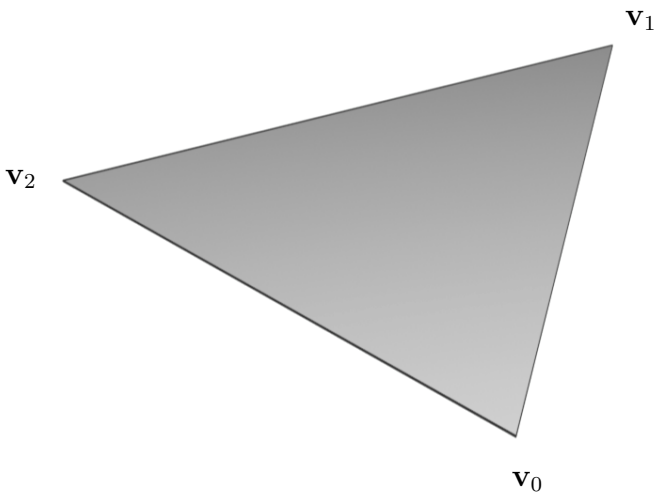


Figure 4: Isotropic face of a mesh.

The triangular faces of the mesh are taken as they are and are subdivided using a *generative* criterion, which in the implementations of this study (both for μ-mesh and anisotropic μ-mesh) is the length of the subdivision, so-called "**target edge length**". The aforementioned value is employed to compute the **subdivision level** (represented as a power of 2 of the **subdivision index**) for each edge of the face.

To elaborate, the length of each edge is divided by the target edge length, and the outcome is rounded to the nearest power of 2. The resultant value is the subdivision index for the respective edges of the face. The target edge length is the criterion that determines the complexity in terms of μ-faces of the subdivided mesh (the lower it is, the more μ-faces the subdivision will produce).

$$\max\left(\left[\log_2\left(\frac{\|\mathbf{v}_j - \mathbf{v}_i\|}{\textit{target edge length}}\right)\right], 0\right), \quad \mathbf{v} \in \text{Faces}$$

$$i \in \{0,1,2,3\}, \quad j = \begin{cases} i+1, & \text{if } i \neq 2 \\ 0, & \text{if } i = 2 \end{cases} \tag{1}$$

Once the **subdivision indices** i, j and k are obtained, the μ-mesh subdivision scheme is applied (Fig. 5). This scheme subdivides each side of the triangle into a power of two raised to the corresponding subdivision index (i.e., $2^i, 2^j$ and 2^k) segments. Subsequently, these segments are connected to form the new μ-vertices and μ-faces.

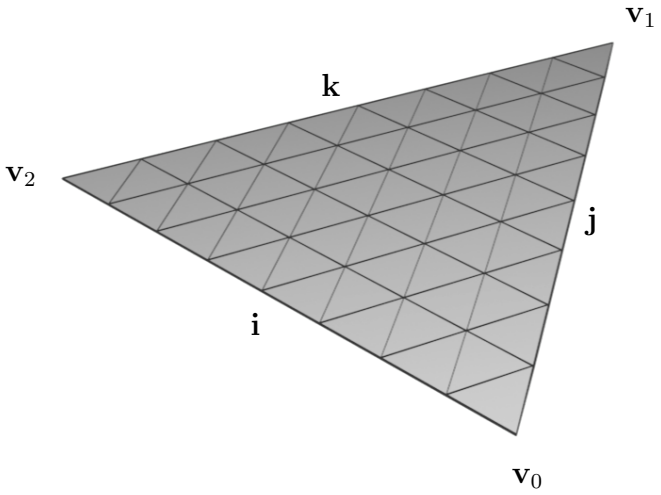


Figure 5: $i = j = k = 3$ resulting in $2^3 = 8$ segments per side.

The subdivision scheme strictly adheres to a precise internal mesh subdivision policy. According to this rule, two sides maintain an identical level of subdivision, while one side exhibits a subdivision level that is at most power of 2 lower than the other two sides. Taking this into account, the quantity of internal μ-faces is determined by the Expression 2. Here, j corresponds to the subdivision index of the side that possesses a subdivision level equal to or lower than the subdivisions of the other two sides[1] expressed by i.

$$2^i \times 2^i \times 2^j, \text{where } j \leq i \text{ and } |j - i| = 1 \tag{2}$$

[1]This means $k = i$, for a more elegant formula in Equation 2 only the i index will be considered.

This means that two sides must have the same level of subdivision. For instance, combinations that could satisfy the condition include $8 \times 8 \times 4$, $16 \times 16 \times 16$, $8 \times 8 \times 2$, and so forth. On the other hand, combinations that do not adhere to the requirement might include $16 \times 8 \times 4$, $8 \times 8 \times 16$, and others.

The scheme is established through a process that involves *iteratively* processing the mesh's faces. Before executing the algorithm, temporary indices are assigned to each edge of the mesh's faces, with the target edge length serving as the guiding parameter.
The process unfolds as follows:

1. **Initial subdivision levels** are assigned to edges based on their length and the computed target edge length.

2. The **μ-mesh scheme** is repeatedly **enforced** on the base mesh. This is accomplished by employing multiple times an enforcement procedure on the mesh, iterating until *stability* is achieved. Stability is attained when the μ-mesh subdivision scheme is consistently applied to all faces. Given that setting a subdivision level for one edge could disrupt the subdivision of neighboring faces, given this reason, enforcement needs to be carried out iteratively.

This approach ensures that the desired subdivision pattern is systematically applied to the mesh, optimizing the outcome by considering the edge length and target specifications. Additionally, the iterative enforcement of the μ-mesh scheme guarantees that adjacent faces remain compatible and maintain the desired subdivision levels.

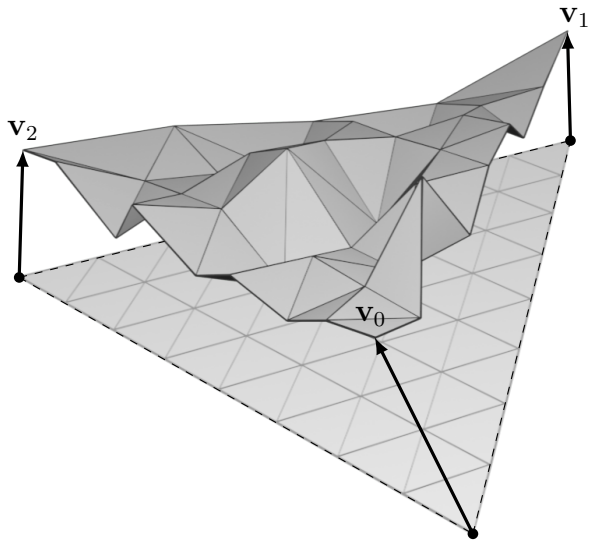


Figure 6: Application of target mesh displacements on the μ-vertices.

After obtaining the new μ-vertices and μ-faces, it is time to compute the **displacements vector** concerning the target mesh. This involves performing *line-casting* using the positions of the μ-vertices as origins and casting rays along the direction (forward and backward) of the vertex normal. The intersection of the ray with the faces [Möller and Trumbore, 1997] of the target mesh is computed, and the **smallest absolute displacement** from the μ-vertex is saved. Once the displacements vector is generated, it is used to perform the displacement of the μ-vertices (as shown in Fig. 6) of the subdivided base mesh. The result is an extremely detailed approximation of the target mesh.

3.3 Goals

The primary aim is to establish a robust construction methodology capable of generating *precise* and *efficient* μ-mesh representations from a high-resolution triangle surface input.

In pursuit of this objective, it becomes imperative to navigate a series of trade-offs inherent to the nature of μ-mesh representation. Specifically, these trade-offs emerge due to the unique characteristics of μ-meshes (in bold).

Coarseness of base mesh, an essential consideration in the realm of μ-mesh construction pertains to the *granularity* of the base mesh. It is empirically established that the memory efficiency of μ-mesh representations increases as the coarseness of the base mesh intensifies.

This optimization arises from the allocation of more intricate geometric details into scalar displacements, with the generation of μ-displacements executed as needed. Notably, the current design can accommodate substantial amplification factors, often exceeding a ratio of 1000 to 1. Consequently, the strategic objective is to employ a base mesh that exhibits granularity up to three orders of magnitude coarser than that of the input surface, commonly referred to as the *target mesh*.

Reprojectability, this concept in the context of μ-mesh is fundamentally concerned with the ability to faithfully reproduce the target surface. This entails the necessity for the μ-mesh to intersect the target mesh precisely along the path defined by the μ-mesh's interpolated displacement vectors.

It's important to note that meeting this requirement presents a significant challenge, both in terms of achieving it and assessing its fulfillment. Ideally, the rays representing displacement directions should exhibit local orthogonality as they interact with and deform the input surface.

Isotropy, within the domain of geometry processing, isotropy refers to a fundamental characteristic arising from the regular grid structure. This regularity significantly impacts the shape and aspect ratio of the resultant μ-faces before displacement occurs.

Consequently, the regular grid tends to impart an essential quality—equilateralness—especially to those base triangles that are already approximately equilateral in shape. This *equilateral quality* enhances the efficiency of surface sampling. To maintain sampling efficiency, preference is given to base triangles with lower aspect ratios. In contrast, longer, more elongated base triangles, referred to as *anisotropic* triangles, yield comparatively less accuracy in the sampling process.

Minimal prismoid volume, the goal is to minimize the volume of the prismoid, which encompasses the represented surface. Given that the base triangle area expands as the mesh becomes *coarser*, the length of displacement vectors plays a crucial role in achieving a reduction in prismoid volume.

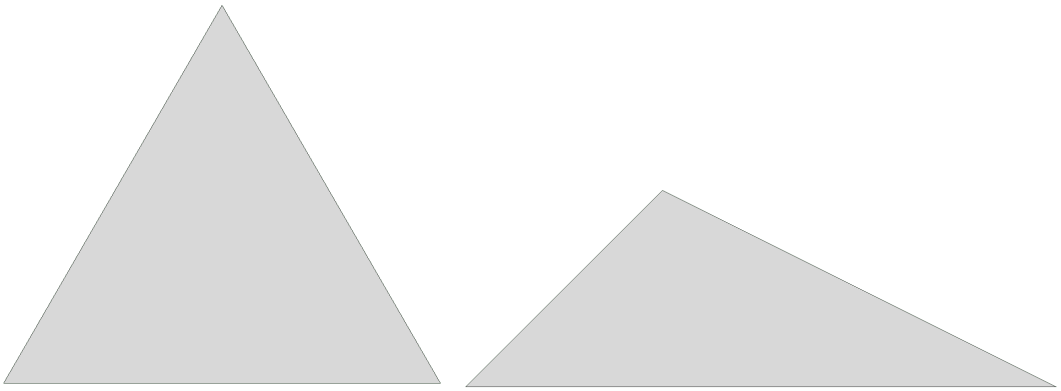
Shorter displacement vectors are desirable for two significant reasons: Firstly, shorter vectors result in greater *accuracy*. Secondly, shorter displacement vectors ensure that the prismoid acts as a tighter *bounding hull*, thereby enhancing culling and rendering performance. It's important to note that, as the prismoid serves as an envelope, a base mesh that is further from the surface incurs a penalty in terms of the prismoid volume.

Regarding the discourse opened in the last point, it should be said that Micro-Mesh are specifically designed to work well with *ray tracing*. This means that μ-triangles can be generated during the rendering process when needed. The displaced surface is created within the boundaries defined by the base mesh and target mesh face, forming a shape called a *prismoid*. This bounding nature of the prismoid is utilized in constructing the ray-tracer's BVH and when tracing rays for the actual rendering.

Chapter 4

Anisotropic Micro-Mesh: a new proposal

Micro-Mesh can perform the subdivision of the base mesh into *uniform* and *regular* triangles, a technique referred to as **isotropic meshing**. In this process, each side of the mesh element is of uniform length, resulting in *geometrically balanced* shapes (see Fig. 7a). This type of meshing is typically utilized to depict straightforward geometric objects or scenarios where the physical characteristics of the object exhibit little variation across different directions.



(a) Isotropic (b) Anisotropic

Figure 7: Comparison of isotropic and anisotropic triangle.

On the other hand, the anisotropic μ-mesh scheme utilizes an **anisotropic meshing**, in which the faces can vary in shape, size, and orientation in different directions. This kind of meshing is used when the physical or geometric properties of the object vary significantly in different directions. An anisotropic mesh is a mesh with *long*, *elongated* triangles (Fig. 7b) in the right places. They often provide

better interpolation of multivariate functions with fewer triangles and they are used in finite element methods to resolve boundary layers and shocks.
The main distinction between isotropic and anisotropic meshing lies in the regularity and adaptability of the grid elements to the properties of the object or material being represented. Anisotropic meshing provides greater flexibility and precision in representing complex objects or materials with directional properties.

In the field of remeshing, the ability to construct isotropic meshes facilitates the creation of high-quality meshes [Jakob et al., 2015]. Anisotropic μ-meshes can produce isotropic meshes even from anisotropic triangles, unlike μ-meshes. Micro-Meshes follow a pattern that tends to generate μ-faces which maintains the same structure as the "*macro-faces*" (faces of the base mesh), without controlling the anisotropy.

The *ability* to **control anisotropy** represents a key feature that enables the meshes to achieve a more equilateral subdivision. In the paper describing the construction of μ-meshes, considerable challenges are faced in attempting to generate base meshes with equilateral faces. This is a complex task that requires a considerable investment of time and resources. The anisotropic μ-meshes approach aims to free the user from this burden, as it is the scheme itself that ensures a more equilateral subdivision during the subdivision process.

Developing an anisotropic version of NVIDIA's μ-mesh format introduces potential advantages that are worth considering, although rigorous testing is required to verify its benefits in comparison to the standard μ-mesh format.

The anisotropic μ-mesh variation offers a heightened level of *flexibility* and *adaptability* compared to its regular counterpart. While the conventional format relies on uniform triangles, the anisotropic approach allows for the mesh to be tailored to intricate shapes and orientations. This adaptability could result in improved *accuracy* when capturing complex details and ensuring a more comprehensive representation of intricate features.

An appealing aspect of the anisotropic approach lies in its suitability for objects with preferred directional characteristics, such as surfaces that exhibit light reflection or refraction. By aligning the mesh with these directions, the potential for enhanced visual quality becomes evident, particularly in rendering surfaces with varied reflective or textured attributes.

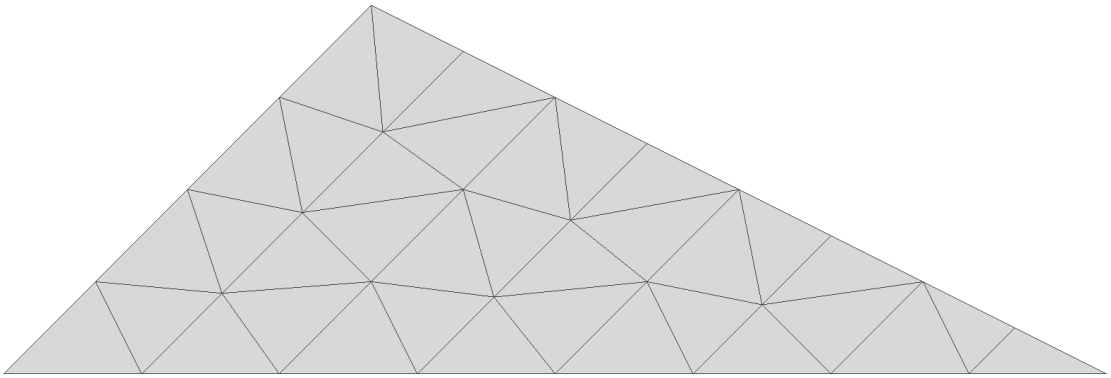
4.1 Subdivision scheme

The anisotropic edge subdivision scheme consists of two edges that share the same maximum subdivision level and a third edge that is of a lower subdivision level (regardless of the extent, as long as it is lower). In contrast to the μ-mesh subdivision scheme (Eq. 2), the anisotropic μ-mesh scheme (Eq. 3) ensures a scenario

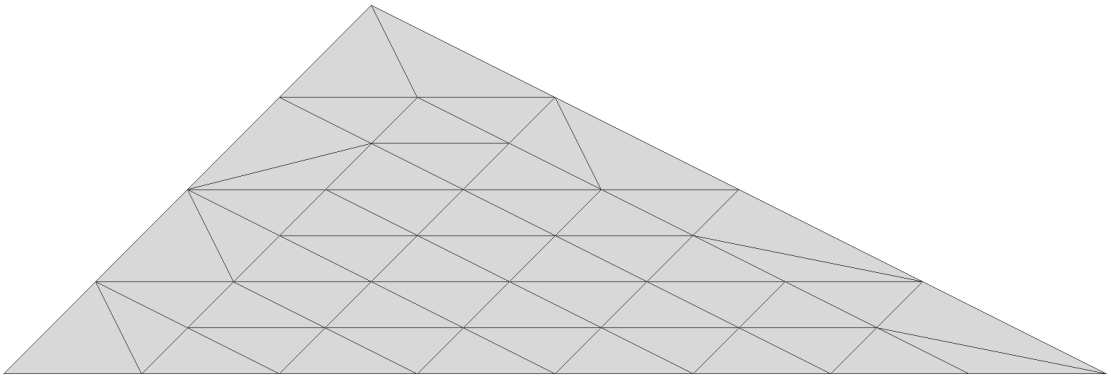
where the distribution of subdivisions is non-*uniform* and *heterogeneous*.

$$2^i \times 2^i \times 2^j, \text{where } j < i \tag{3}$$

Considering an anisotropic triangle, when employing the anisotropic μ-mesh subdivision scheme, two of its edges will reach the maximum subdivision level, while one edge will attain a lower subdivision level, as depicted in Figure 8a. Conversely, with the μ-mesh scheme, only one edge will reach the maximum subdivision level, while the other two edges will undergo a lesser degree of subdivision (Fig. 8b).



(a) $2^3 \times 2^3 \times 2^2$ — Anisotropic Micro-Mesh.



(b) $2^3 \times 2^2 \times 2^2$ — Micro-Mesh.

Figure 8: An examination of the two subdivision schemes applied to an identical elongated triangular shape.

From the scheme comparison, it is evident that the anisotropic μ-mesh scheme exhibits a notably more *homogeneous* behavior when applied to elongated triangles of this nature. It achieves a *uniform* distribution of the μ-faces. This observation is particularly intriguing because it generates significantly fewer μ-faces and μ-vertices compared to other schemes, while simultaneously demonstrating efficient surface sampling.

To conduct a meaningful assessment (Chapter 6), it is essential to compare the schemes using an equal number of μ-faces generated by each scheme and evaluate their response to μ-vertices displacement towards a target mesh.

4.2 Implementation

Taking into account the triangular configuration illustrated in Figure 8a, our objective is to contemplate the methodology for extracting all requisite triangles. In furtherance of this objective, we have formulated a logical diagram of the configuration (depicted in Fig. 9) that accentuates the logical spatial orientation of these triangles.

The diagram depicts the *enumeration* of border segments we aim to obtain through subdivision along all three sides of the triangle, namely the shorter side, the irregular longer side, and the regular longer side, respectively.

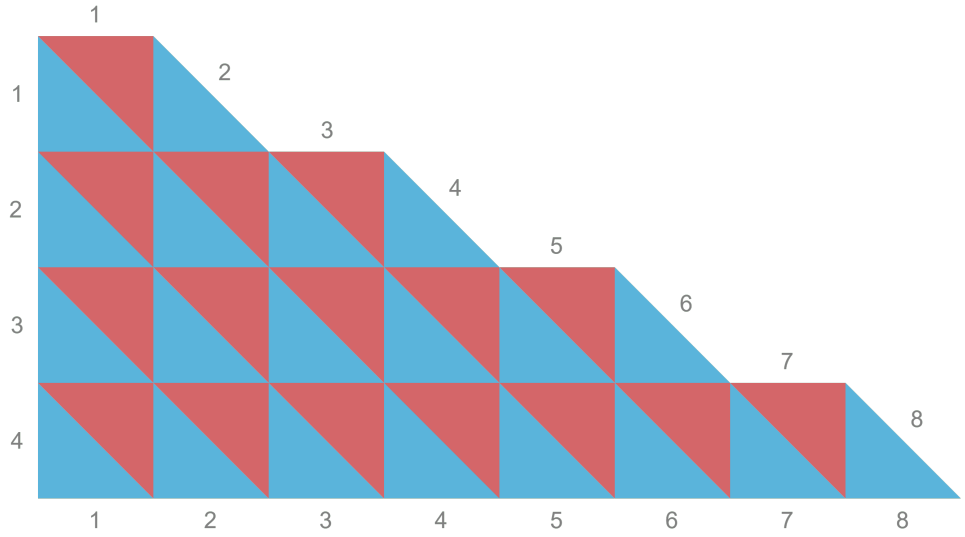


Figure 9: Anisotropic configuration for $2^3 \times 2^3 \times 2^2$.

This diagram will be reused and expanded upon in the future for the generation of the μ-faces. Currently, it has been presented to establish that this blueprint structure will serve as the foundation for constructing the algorithm.

The structure of anisotropic μ-mesh is obtained by executing a dedicated subdivision algorithm (Algo. 1) on the base mesh. This algorithm yields the base mesh with increased vertices (μ-vertices) and faces (μ-faces), the subdivided mesh. The algorithm iterates over the faces of the mesh and first extracts the μ-vertices, subsequently constructing the μ-faces.

Iterating over each face (outermost loop) of the base mesh involves performing a series of operations at each step:

1. Declaration of auxiliary variables.
2. Construction of μ-vertices (Algo. 2).
3. Construction of μ-faces (Algo. 3).

Once the subdivided mesh is generated *update* (e.g., bounding box) and *clean* (e.g., removing duplicated vertices) procedures will be executed. In the initial step, various auxiliary variables are defined for each step to facilitate calculations.

These variables are essential for determining the subdivision level of the longer side, the subdivision level of the shorter side, the level of anisotropy, and the number of vertices in the subdivided mesh at a specific iteration of the outermost loop.

- n, represents the *subdivision level* of the **longer side**, specifically, the edge with the highest subdivision index. It is denoted as "n," and it takes on values that are powers of 2. The index of the power corresponds to the subdivision index of the edge.
- m, denotes the *subdivision level* for the **shorter side** of the triangle. It is associated with the edge having the lowest subdivision index. Like "n," it consists of powers of 2.
- aniso, quantifies the **level of anisotropy** within the problem. It is also a power of 2, where the index reflects the difference between "n" and "m." In other words, it is a measure of the disparity in subdivision levels between the longer and shorter sides.
- k, represents the *number of vertices* that are part of the subdivided mesh during a specific *iteration* of the outermost loop. This variable helps keep track of the **subdivided mesh vertices size** as it evolves through each iteration.

These auxiliary variables collectively aid in characterizing and managing the subdivision process and anisotropy within the mesh.

Algorithm 1 Algorithm overall structure.

```
1: procedure ANISOTROPICMICROMESHSUBDIVIDE
2:     subdivided ← Mesh()
3:
4:     for ∀f ∈ baseMesh.faces do
5:         subLvlEdge0 ← subdivision level for the long edge
6:         subLvlEdge1 ← subdivision level for the long edge
7:         subLvlEdge2 ← subdivision level for the shortest edge
8:
9:         n ← 2^subLvlEdge0
10:        m ← 2^subLvlEdge2
11:        aniso ← 2^(subLvlEdge0−subLvlEdge2)
12:        k ← subdivided.vertices.size()
13:
14:        Generation of μ-vertices (Algorithm 2)
15:        Generation of μ-faces (Algorithm 3)
16:        ...
17:    end for
18:
19:    subdivided.updateBoundingBox()
20:    subdivided.updateEdges()
21:    ...
22:
23: return subdivided
24: end procedure
```

Second step, the μ-vertices has to be generated and added to the vertices data structure of the subdivided mesh, in Figure 10, a diagram can be seen illustrating how the data structure organizes them. They descend in sequence from left to right and from top to bottom.

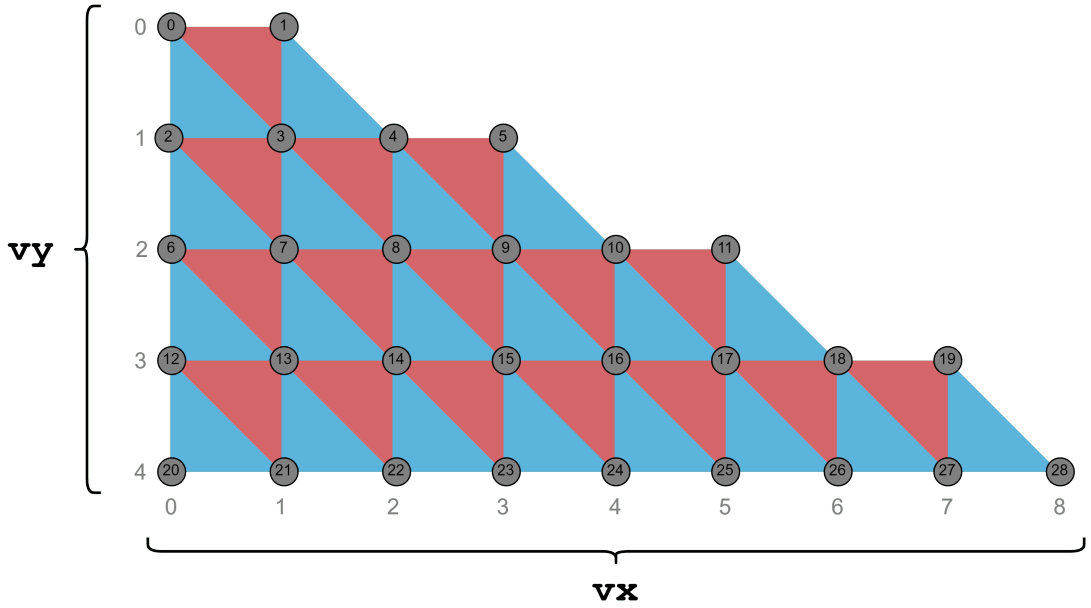


Figure 10: Micro-vertices structural diagram.

To implement the Algorithm 2 we have to consider the diagram of Figure 10. Two indices are needed, vx for the horizontal coordinates and vy for the vertical coordinates.

For the generation two nested loops are needed: the first one iterates over the vertical coordinates from 0 to m (included), while the other loop iterates over the horizontal coordinates, from 0 to the last available horizontal coordinate[1] ζ. The value of ζ is determined in the *outermost* loop, and can change depending on the value of m:

$$\zeta = \begin{cases} vy \cdot \text{aniso} + \text{aniso} - 1, & \text{if } m > 0 \\ vy \cdot \text{aniso}, & \text{if } m = 0 \end{cases} \tag{4}$$

Within the nested loop, the integer coordinates vx and vy become accessible, necessitating the derivation of the barycentric coordinates $a, b, andc$ of the μ-vertex. The formula for barycentric coordinates posits that their sum yields the value 1.

$$\begin{aligned} c &= \frac{vx}{n} \\ a &= \begin{cases} 0, & \text{if } vx = n \\ (1-c) \cdot \dfrac{m - vy}{m - \dfrac{vx}{\text{aniso}}}, & \text{if } vx \neq n \end{cases} \\ b &= 1 - a - c \end{aligned} \tag{5}$$

[1] referred to as *lastVx* within the codebase.

The computation of coordinates follows the sequence of equations as *presented*. The variable c represents a fraction of n, while a involves a more intricate computation that accounts for the contribution of c as a fraction.

The denominator $m - \frac{vx}{\text{aniso}}$ indicates the count of vertical segments connecting the μ-vertices (Fig. 10), and the numerator $m - vy$ represents the vertical segments below the current μ-vertex. The computation of b is straightforward. Once the barycentric coordinates are computed, a new μ-vertex is generated. This vertex is produced by a *linear combination* of the computed barycentric weights applied to the current vertex position.

Algorithm 2 Micro-vertices generation.

```
1:  procedure anisotropicMicromeshSubdivide
2:      ...
3:      for vy ← 0 to m do
4:          lastVx ← vy · aniso + aniso − 1
5:
6:          if vy = m then
7:              lastVx ← lastVx − (aniso − 1)
8:          end if
9:
10:         for vx ← 0 to lastVx do
11:             c ← vx/n
12:             a ← 0
13:
14:             if vx ≠ n then
15:                 a ← (1 − c) · (m − vy)/(m − vx/aniso)
16:             end if
17:
18:             b ← (1 − a − c)
19:             bary ← [0, 0, 0]
20:
21:             bary[w0] ← b
22:             bary[w1] ← c
23:             bary[w2] ← a
24:
25:             subdivided.vertices.push(getSurfaceVertex(f, bary))
26:         end for
27:     end for
28:     ...
29: end procedure
```

In the second phase of the algorithm, the objective is to generate the μ-faces and add them to the faces data structure of the anisotropic μ-mesh. The triangle faces should be generated such that they follow the configuration diagram of Figure 9. An interesting observation is that by rotating the *red triangles* by an angle of 180°, a rectangular grid is formed (Fig. 11).

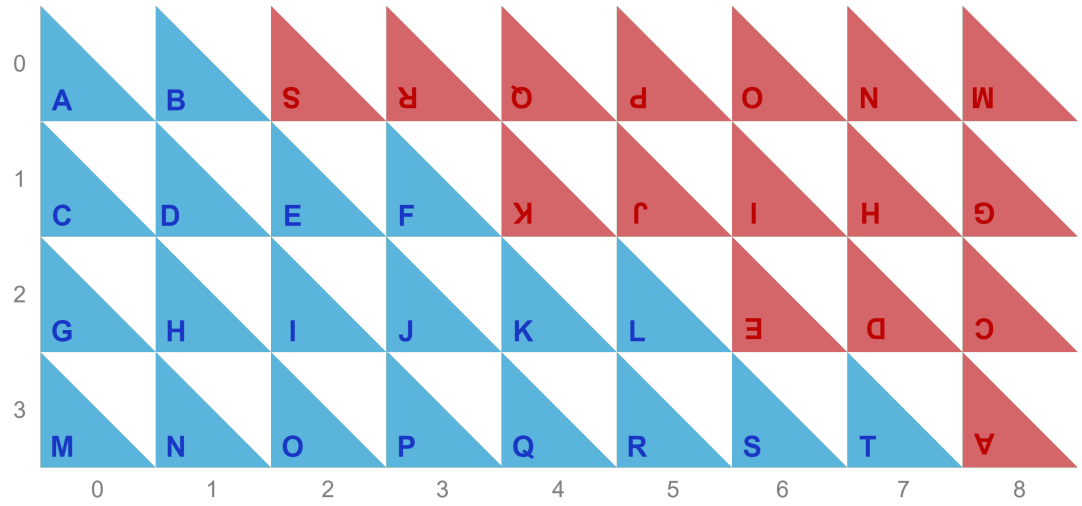


Figure 11: Rectangular grid of μ-faces, with labels indicating pairs of attached blue-red triangles.

This **rectangular grid** of triangles proves to be highly advantageous in the codebase implementation. The process of generating the grid is trivially obtained by the utilization of two nested loops. Subsequently, the algorithm adjusts the rotated red triangles at their coveted position (Fig. 12), by rotating their vertices back of 180°.

Two *indices* will be used in the practical implementation, denoted as fx and fy, which are necessary to reference an individual μ-face. The outer loop is responsible for indexing the *vertical* coordinates, and the inner loop is for the *horizontals*. Specifically, the outer loop iterates through the range from 0 up to m (excluded). Conversely, the inner loop operates within the bounds of 0 to $n + \text{aniso} - 1$ (excluded).

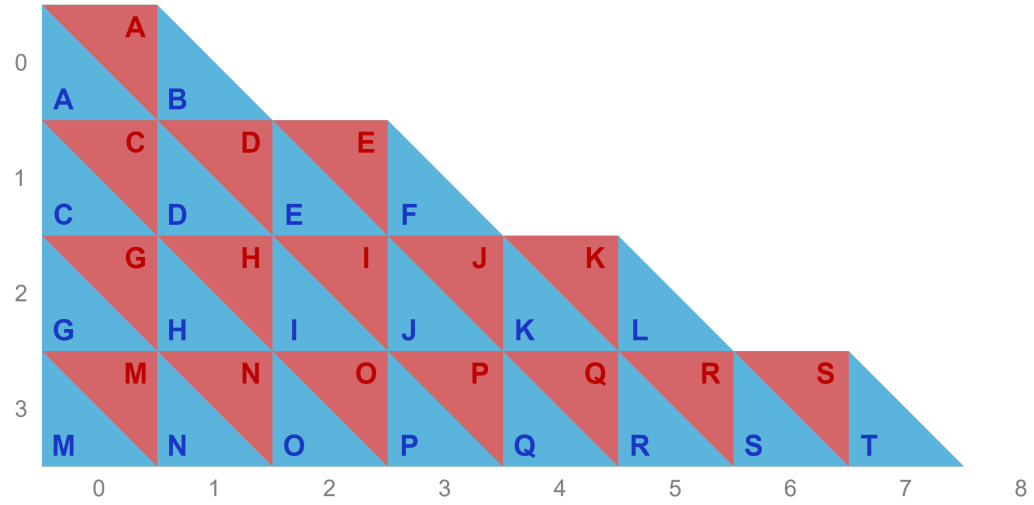


Figure 12: Red triangles rotated to their coveted position.

In the codebase, is introduced a *local function*[2] φ, which is designed to operate within a confined scope. This function takes as input a bi-dimensional vector of integers $\mathbf{v}$, representing the local coordinates of a μ-face. Its primary objective is to determine the corresponding index within the face data structure, a task accomplished through the following formula:

$$\varphi(\mathbf{v}) = \frac{\mathbf{v}_y \cdot \text{aniso} \cdot (\mathbf{v}_y + 1)}{2 + \mathbf{v}_x} \tag{6}$$

Within specific loops, three vertices $\mathbf{v}_0 = (fx, fy)$, $\mathbf{v}_1 = (fx, fy + 1)$, and $\mathbf{v}_2 = (fx+1, fy+1)$ are defined, representing the bi-dimensional local coordinates of the μ-face. Due to their localized nature, these vertices require transformation into a format compatible with the faces data structure of the subdivided mesh (μ-mesh).

This transformation involves the variable k (representing the current number of vertices) in conjunction with the local function φ. For a given iterative step i, the 3D face vector $\mu\text{-face}_i$ is defined as follows:

$$\mu\text{-face}_i = \begin{bmatrix} k + \varphi(\mathbf{v}_0) \\ k + \varphi(\mathbf{v}_1) \\ k + \varphi(\mathbf{v}_2) \end{bmatrix} \tag{7}$$

The code has to handle also the *red triangles* (Fig. 11), which are triangles that are rotated in the current data structure, so they have to be rotated back otherwise they will be misplaced in the final subdivided mesh. To ascertain whether a red triangle is currently under processing, a condition is employed. This condition

[2]referred to as *toVertexIndex* within the codebase.

involves an assessment of whether the horizontal coordinate fx exceeds the value of $(fy + 1) \cdot \text{aniso} - 1$. If this condition is satisfied, signifying the presence of a red triangle to be processed, the subsequent action is performed, which entails rotating it by 180°.

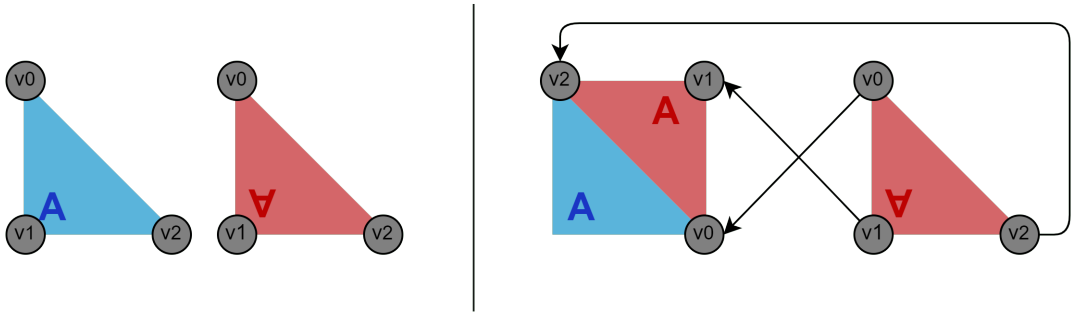


Figure 13: Red triangles rotation.

To accomplish this rotation, an *auxiliary* bi-dimensional vector of integers, denoted as $\alpha = (n + \text{aniso} - 1, m)$, is introduced (variable referred as *rotate* within the codebase). The subtraction of the μ-face coordinates from the α vector serves as a **pivot** for the rotation operation applied to the red triangles. Consequently, for a given iterative step denoted as i, the definition of the 3D face vector $\mu\text{-face}_i$ will be adjusted as follows:

$$\mu\text{-face}_i = \begin{bmatrix} k + \varphi\,(\alpha - \mathbf{v}_0) \\ k + \varphi\,(\alpha - \mathbf{v}_1) \\ k + \varphi\,(\alpha - \mathbf{v}_2) \end{bmatrix} \tag{8}$$

Distinct face vectors, denoted as $\mu\text{-face}_i$, will be generated for both red and blue triangles. However, ultimately, these face vectors will be incorporated into the face data structure of the anisotropic μ-mesh.

Algorithm 3 Micro-faces generation.

```
1: procedure anisotropicMicromeshSubdivide
2:     ...
3:     function toVertexIndex(v)
4:         return v.y · aniso · (v.y + 1)/2 + v.x
5:     end function
6:
7:     for fy ← 0 to m − 1 do
8:         for fx ← 0 to n + aniso − 2 do
9:             v0 ← [fx, fy]
10:            v1 ← [fx, fy + 1]
11:            v2 ← [fx + 1, fy + 1]
12:
13:            if fx > (fy + 1) · aniso − 1 then
14:                rotate ← [n + aniso − 1, m]
15:
16:                v0 ← rotate − v0
17:                v1 ← rotate − v1
18:                v2 ← rotate − v2
19:            end if
20:
21:            v0k ← k + toVertexIndex(v0)
22:            v1k ← k + toVertexIndex(v1)
23:            v2k ← k + toVertexIndex(v2)
24:
25:            subdivided.addFace(v0k, v1k, v2k)
26:        end for
27:    end for
28:    ...
29: end procedure
```

Following the generation process, several subsequent steps are performed. These include the computation of the *bounding box* for the newly created mesh, updating the relationships between edges, and eliminating *duplicated* vertices.

It's worth noting that the presence of duplicated vertices is a remnant from the μ-vertices generation procedure.

4.3 Implicit LOD

One of the reasons why these formats for geometric processing are very attractive is the *directly* proportional calibration of the level of detail (LOD) with respect to the level of subdivision adopted.

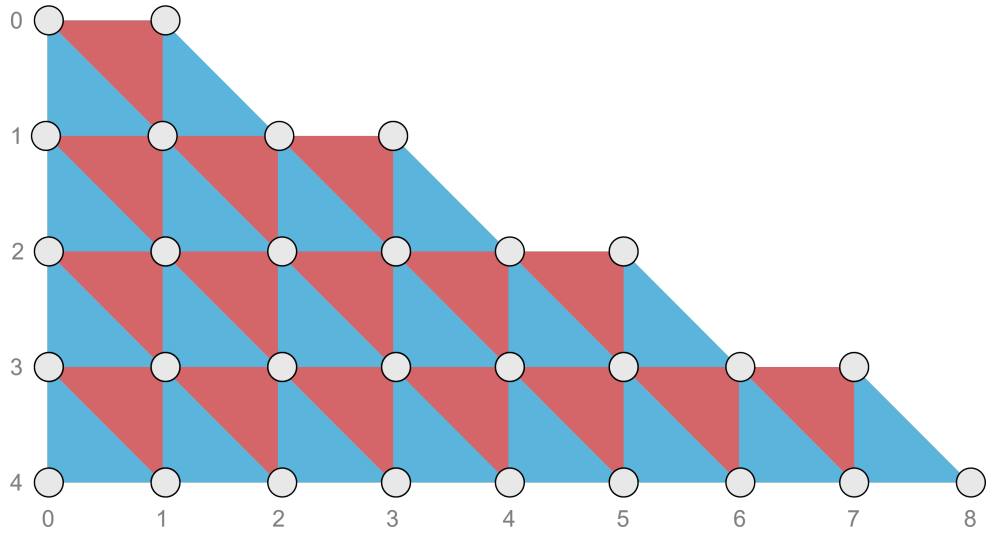


Figure 14: LOD for short distances from the observer.

It, therefore, turns out to be possible to adopt specific configurations based on distances and quantities of μ-faces to perform scaling of the model´s polygonal resolution, all in real time without taking up memory space.

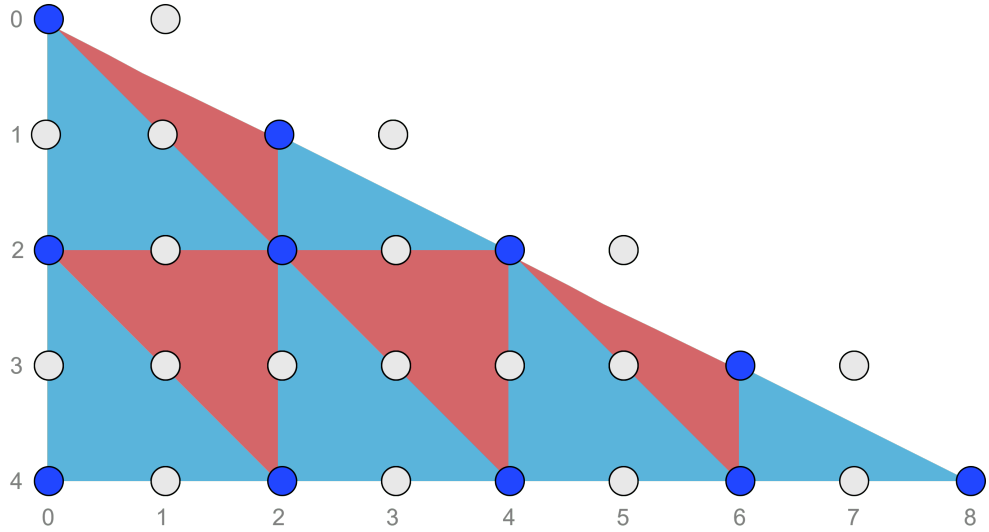


Figure 15: LOD for medium distances from the observer.

The advantage of having this implicit run-time LOD comes from not having to store different resolutions of the same model; two mesh inputs will be sufficient to interpolate all desired resolutions.

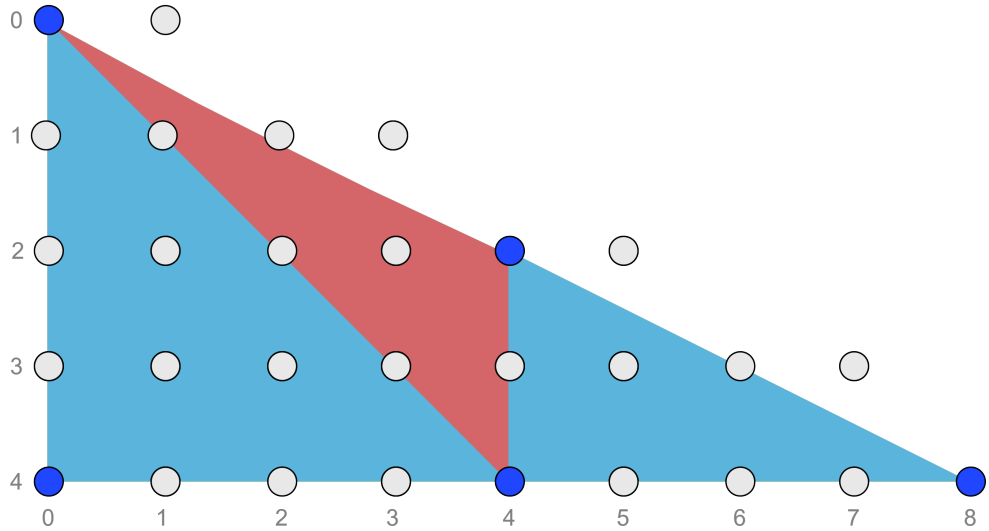


Figure 16: LOD for long distances from the observer.

A second advantage is the ability to choose a wide range of resolutions from those interpolated, and thus to provide different resolutions of the same model for a multitude of distances. In this way, the *popping* effect will be perfectly mitigated, since the precision with which the desired resolution of the triangles can be set avoids large jumps in resolution from one distance to another.

4.4 Highly obtuse isosceles triangles: a challenge

There is one specific case that undermines the quality of the final result: the anisotropic scheme cannot gracefully handle highly *obtuse* isosceles triangles. The triangles that are considered to perform the subdivision are not well matched to each other, as can be seen in Figure 17 (this is a problem that also plagues classical triangles μ-mesh).

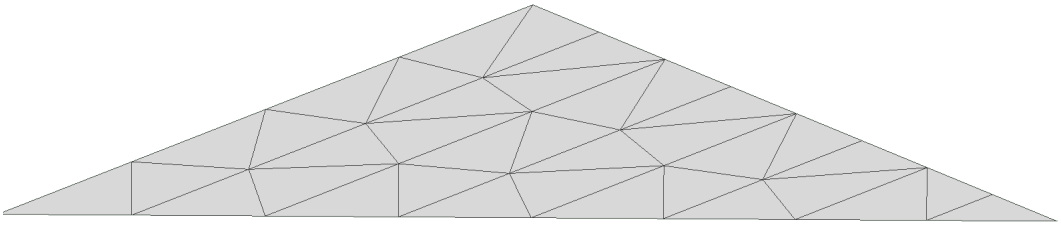
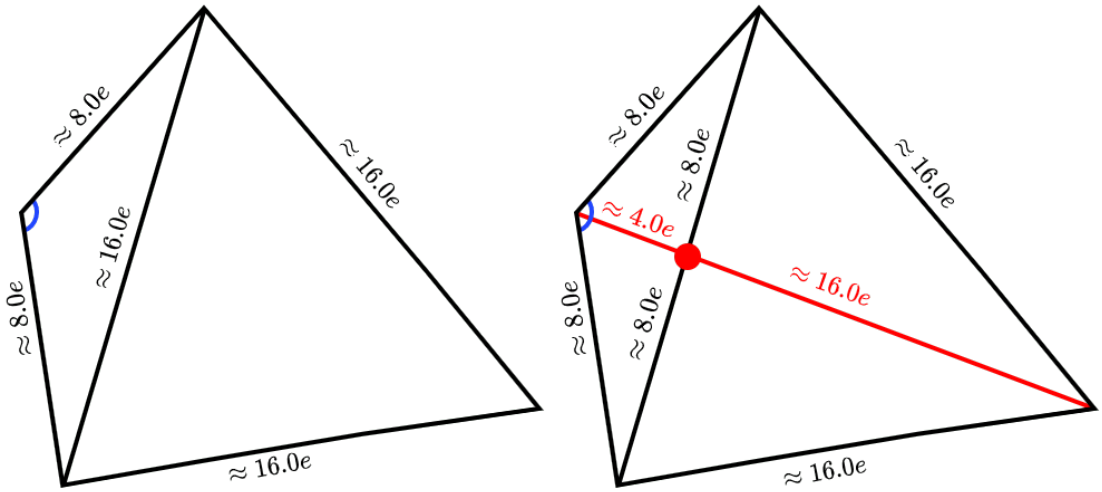


Figure 17: Anisotropic division of a highly obtuse isosceles triangle.

A possible solution to this problem may also be trivial, considering the diagram exhibited in Figure 18a, we note that the longest side will surely be adjacent to another triangle, which in this particular case is an equilateral isosceles triangle. Then simply divide the obtuse angle into two, that is, divide the two adjacent triangles into four, thus obtaining four right triangles, with adjacent right angles.



(a) Highly obtuse isosceles triangle. (b) Isosceles split.

Figure 18: Fix procedure for highly obtuse isosceles triangles.

After the isosceles split the subdivision scheme will take on a topology that will qualitatively favor the aspect ratio of the triangles, and consequently be more pleasing to the eye.

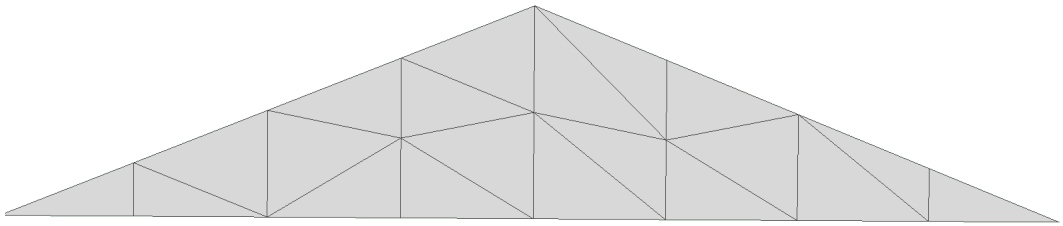


Figure 19: Anisotropic subdivision after the split.

It should be emphasized that the proposed solution was not implemented during project development, so the empirical analyses performed (Chapter 6) do not benefit from this optimization.[3].

[3]It is not expected to have enough impact to justify implementation efforts.

Chapter 5

Methodology

In this chapter, the methodologies employed in this research will be examined. This examination will include an elucidation of the technical and software frameworks that underpin the development and analysis. The subsequent sections will provide insights into the development environment, tools, and software platforms used to conduct the study effectively.

5.1 Project structure

The project has been structured in a way that input files are well *separated* from those produced by the application. The structure is depicted in Fig. 20. Each folder serves a specific purpose, and the project files have been organized in a manner to achieve the cleanest possible directory structure.

- **Root** folder contains the source code files (`.h` and `.cpp`), the Python script for computing the metrics (`face-stats.py`), and the configuration files for the project and repository (`mainwindow.ui`, `master_thesis.pro`, Makefile, `.gitignore` and `README.md`).
- **Release** and **Debug** are the compiler output folders containing the executable and file objects. The release build is optimized for the application's usage, while debug builds are intended for debugging purposes.
- **Dependencies** contains the external libraries utilized by the application. For this project, only GLM [G-Truc Creation, 2005] (OpenGL Mathematics), a mathematics library exclusively for C++, is used for graphics software based on the GLSL [OpenGL ARB, 2020] specifications.
- **Models** is the directory housing the base and target mesh models (inputs).

- **Evaluation** is the directory that contains the subdivided and displaced subfolders of the samples built using the command-line interface of the application (Sec. 5.2.2).

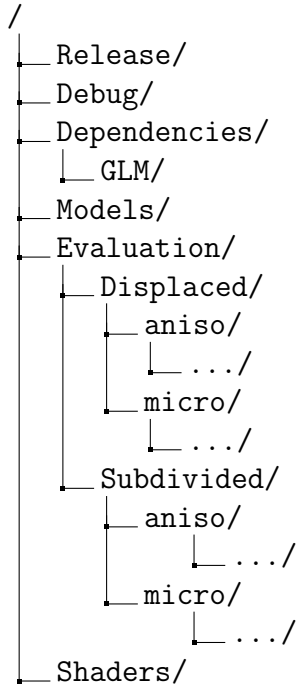


Figure 20: Project directory structure.

5.2 Tools and technologies used

5.2.1 Qt framework

The development of robust and visually appealing **Graphical User Interfaces** (GUIs) is a multifaceted endeavor that requires expertise across various domains, encompassing *graphics programming*, *user interface design*, and *software architecture*. Within this context, the Qt framework [Qt Software, 1995] from Qt Software (also known as Trolltech) has emerged as a potent and versatile tool that significantly streamlines the process of crafting graphical applications.

Qt stands as a *cross-platform* C++ framework offering a comprehensive array of libraries and tools tailored for constructing applications endowed with sophisticated graphical interfaces. With its extensive feature set and *user-centric* design,

Qt has garnered substantial acclaim among software developers across diverse domains, spanning *computer graphics*, *multimedia*, *gaming*, and *scientific visualization*.

One of the foremost advantages of Qt is its seamless integration with OpenGL [Khronos Group, 2017] (version 4.6.0 is used in the project), a widely adopted **graphic specification** renowned for its capacity to render both 2D and 3D graphics.

By harmonizing the capabilities of Qt and OpenGL, developers can create applications that seamlessly intertwine advanced graphics prowess with user interfaces renowned for their intuitiveness. This integration proves particularly valuable for applications that necessitate real-time rendering, intricate visualizations, and interactive graphical elements.

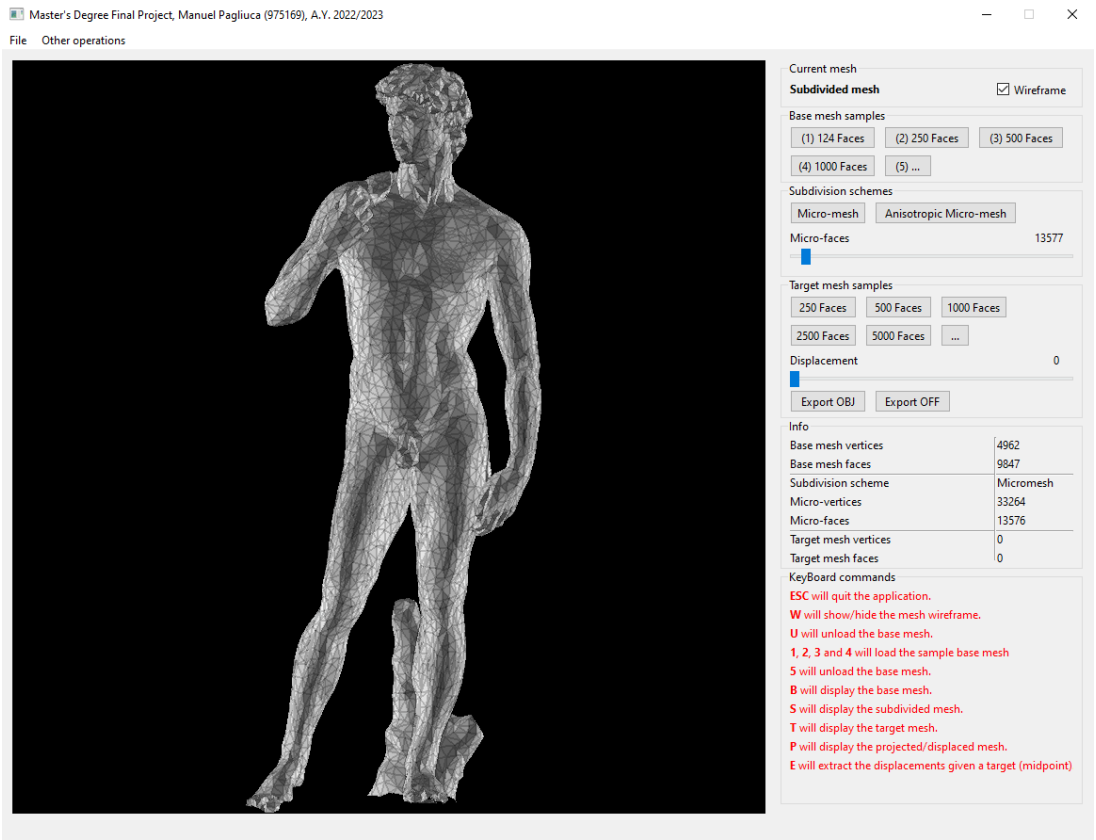


Figure 21: Graphical User Interface.

The merits of leveraging Qt to construct graphical applications that harness OpenGL are manifold. Chief among these is Qt's high-level API, which abstracts

myriad low-level intricacies, allowing developers to concentrate on refining application functionality and design. This abstraction mitigates the complexity of graphics programming, making it more accessible to a broader spectrum of developers.

Additionally, Qt's *cross-platform* nature ensures that applications fashioned with the framework can operate across diverse operating systems with minimal adaptations. This *cross-compatibility* substantially diminishes the development effort requisite to target multiple platforms, empowering developers to engage a more extensive user base with their applications.

The developed **graphical user interface** (GUI) enables users to visualize the mesh through an *OpenGL widget*. Interaction with the widget is facilitated using mouse input, allowing users to rotate the view using an ArcBall mechanism [Shoemake, 1992].

The application provides the capability to display four types of meshes: the base mesh, the subdivided mesh, the target mesh, and the projected mesh (which is the subdivided mesh displaced toward the target mesh). The user can effortlessly switch between these using keyboard shortcuts: `B` for the base mesh, `S` for the subdivided mesh, `T` for the target mesh, and `P` for the projected mesh.

The application's main controls are located on a bar to the right, divided into six group boxes:

- **Current mesh** (Fig. 22).
- **Base mesh selection** (Fig. 23).
- **Subdivision schemes** (Fig. 24).
- **Target mesh selection** (Fig. 25).
- **Mesh informations** (Fig. 26).
- **Keyboard command legend** (Fig. 27).

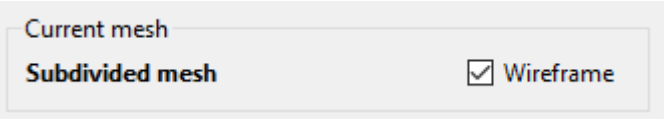


Figure 22: Current mesh group box.

The first group box is responsible for displaying the currently rendered mesh on the OpenGL widget and includes a checkbox to indicate the presence of the wireframe.

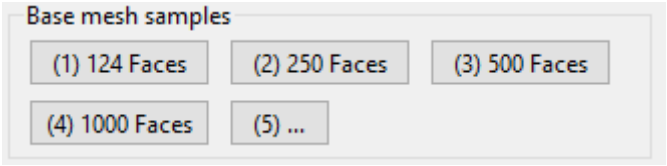


Figure 23: Base mesh samples.

Subsequently, the second group box handles the loading of the base mesh. For testing the application, four shortcut buttons have been provided for loading various complexities of the same sample model. The `(5)` button enables the user to select an arbitrary base mesh.

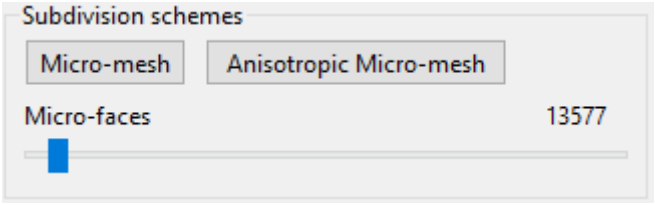


Figure 24: Subdivision schemes group box.

The third group box contains two buttons that allow switching between the two subdivision schemes. Once one of the schemes is selected, a slider titled "Micro-faces", will become enabled, the value of this slider will determine the number of μ-faces that the user would like to use in the subdivided base mesh.

This is made possible by the implementation of the dichotomous search of the target edge length that best approximates the number of targets μ-faces (Sec. 8).

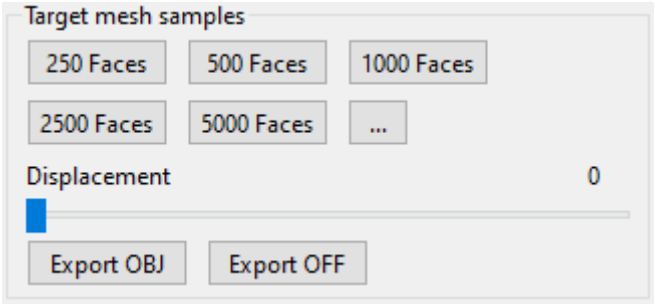


Figure 25: Target mesh selection and morphing group box.

The fourth group box, similar to the second (Fig. 23), this group box allows the loading of the *target mesh*, and the morphing of the *subdivided mesh* towards it. It is possible to use the shortcut buttons for loading different resolutions of the same

sample target mesh or to arbitrarily pick a file. After the selection, minimum displacements (*line-casting*, Fig. 6) calculations are performed. The morphing slider will be enabled once the target mesh displacements will be computed.

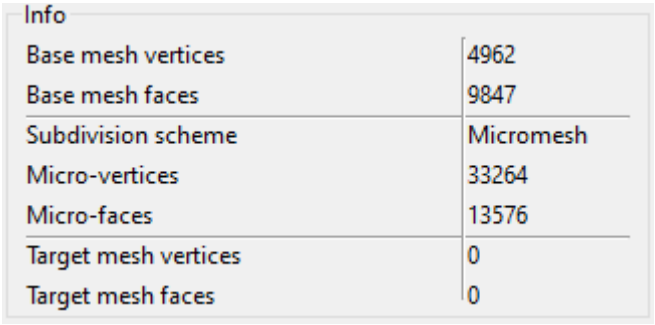


Figure 26: Meshes information's group box.

In the fifth group box the information regarding the amount of vertices and faces of all the meshes, in the case of the subdived mesh the subdivision scheme will be displayed.

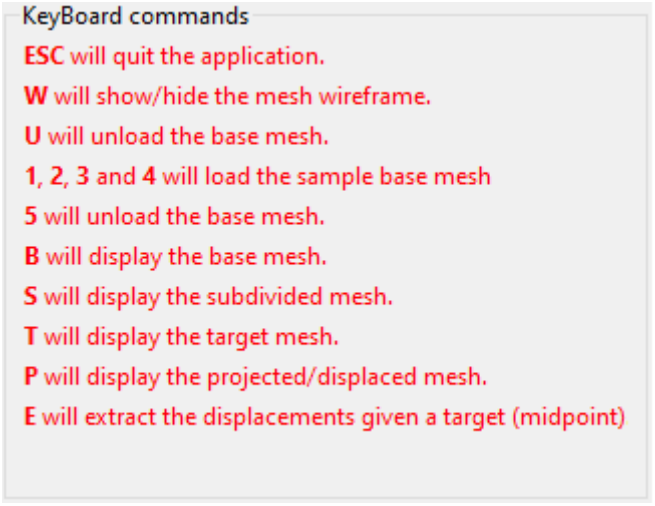
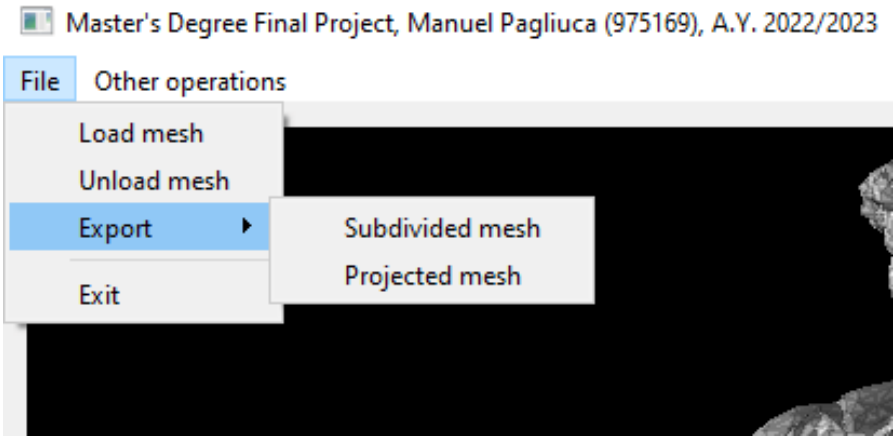


Figure 27: Legend group box.

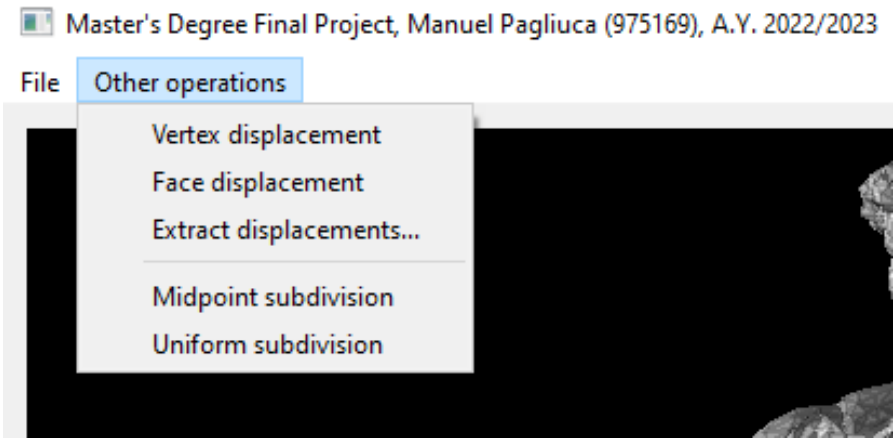
In the last group box, there is a legend for all the keyboard commands that the user can use.

To conclude our overview of the graphical interface, let's examine the application's menu bar. It consists of only two menus: the first one, labeled "File", contains options to load or remove the mesh within the widget. Additionally, it

provides the option to export the subdivided or projected mesh when the displacement slider is set to 100, if applicable.



(a) File menu.



(b) Other operations menu.

Figure 28: Menu bar.

The second menu is labeled "Other operations", as the name suggests. It encompasses operations that are not directly involved in the application's primary usage but may be employed for larger computations. For instance, it includes functions like displacing vertices by a scalar.

5.2.2 Command-line execution

The application is equipped with a **command-line interface** (CLI) that provides a range of advantages for its practical utilization. This approach gains significance

due to multiple contributing factors.

Notably, the GUI, discussed in Section 5.2.1, demands substantial *computational resources*, encompassing both RAM (random-access memory) and CPU (central processing unit) capabilities, for the execution of contextual operations and calculations. Moreover, there exists a clear imperative to execute standardized generative procedures.

The principal motivation for adopting the command-line execution lies in its role as a *supplementary* tool for the comprehensive assessment and meticulous examination of the alternative scheme expounded upon in Chapter 6. In this context, the CLI encompasses distinct commands.

`gen-sample`[1] (Sec. 5.2.6) to conduct empirical analyses, as detailed in Chapter 6. is responsible for generating a **displaced** base mesh towards a specified target mesh, based on a desired amount of μ-faces (using the option `--microfaces`) and the subdivision scheme (using the option `--scheme`). This command executes a dichotomous search (Sec. 5.3.3) to determine the required target edge length (Sec. 3.2) for a given quantity of μ-faces. The resulting displaced mesh will have an approximated amount of μ-faces respect the given value, and it will be stored in the following directory: `./Output/Evaluation/{scheme}/{base mesh name}` [2].

`gen-subdivided-sample` is responsible for exporting the subdivided version of the given base mesh according to the subdivision scheme (`--scheme`) and number of μ-faces desired (`--microfaces`). The resulting subdivided mesh will be stored in the following directory: `./Output/Subdivided/{scheme}/{base mesh name}`

The CLI uses four distinct options for the aforementioned commands:

- `--base-mesh` allows for the specification of the base mesh, with the default being `pallas_1000.obj`
- `--target-mesh` allows the specification of the target mesh, with the default being `pallas.obj`
- `--factor` sets the amount of target μ-faces using a multiplication factor F of the target faces (e.g., F=2.1, the amount of μ-faces generated by the subdivision will be $\sim 2.1 \times$ target μ-faces).

[1] `gen-sample` is the only command used in the Python evaluation script (Sec. 5.2.6).

[2] `{scheme}` is placeholder which can assume the value of "`micro`" or "`aniso`" depending on the subdivision scheme adopted.

- `--scheme` sets the subdivision scheme.

This CLI-driven approach serves to streamline the evaluation process while upholding consistency and reproducibility, thus significantly contributing to the research objectives.

5.2.3 Data structures

In the realm of *geometry processing*, the selection of a programming language plays a pivotal role in determining the efficiency, flexibility, and robustness of implemented solutions.

The C++17 offered numerous benefits, including improved *syntax*, enhanced support for *modern programming paradigms*, and additional *standardized libraries*.

For the geometric processing of 3D meshes, a suitable data structure must be implemented; the solution consists of smaller classes for faces, vertices, and edges. And a compound class of the latter to represent the actual mesh.

Mesh class

```
class Mesh : protected QOpenGLExtraFunctions
{
public:
    std::vector<Vertex> vertices;
    std::vector<Face> faces;
    std::vector<Edge> edges;

    BoundingBox bbox;

    float R;
    int maxAxis;
};
```

Listing 5.1: Mesh class definition.

The Mesh class consists of three dynamic vectors for vertices, faces, and edges respectively, the latter vector relating to the previous ones. The class has its own BoundingBox object and two members R and $maxAxis$ are used for line-casting optimization (Sec. 5.3.1).

Face class

```
struct Face
{
    uint index[3];
    uint edges[3];
    vec3 norm;
    float posMiddle;
};
```

Listing 5.2: Face class definition.

The Face class contains a three-dimensional array containing the indices (of the "vertices" vector) of the vertices that make up the face. A second three-dimensional array containing the indices (of the edge vector) of the edges that make up the face. A *mathematical* vector representing the normal of the face and a $posMiddle$ member is used for line-casting optimization.

Edge class

```
struct Edge
{
    uint faces[2];
    uint side[2];
    uint subdivisions = 0;
};
```

Listing 5.3: Edge class definition.

The Edge class consists of a two-dimensional array containing the indices (of the vector "faces") of the faces shared by the edge. A two-dimensional array indicates to me whether the left or right side of the edge is considered (duplicate edges are created for faces). And an integer value to indicate the subdivision index of the considered edge.

Vertex class

```
struct Vertex
{
    vec3 pos;
    vec3 norm;
};
```

Listing 5.4: Vertex class definition.

The Vertex class is composed of two mathematical vectors, one for the position and the other for the vertex normal (average of the normals of the faces adjacent to the vertex).

5.2.4 OpenGL Mathematics

OpenGL Mathematics [G-Truc Creation, 2005] (GLM) is a C++ mathematics library for graphics software based on the OpenGL Shading Language [OpenGL ARB, 2020] (GLSL) specification.

GLM provides classes and functions designed and implemented with the same naming conventions and functionalities as GLSL (*3D transformations*, *vector* and *matrix operations*, and other mathematical operations essential for computer graphics) so that when a programmer knows GLSL, he knows GLM as well which makes it really easy to use.

5.2.5 MeshLab

The choice to utilize MeshLab [Cignoni et al., 2008] is based on several convincing reasons. Its adaptability to a wide range of 3D data formats and mesh complexities aligns seamlessly with the project's requirements.

Furthermore, MeshLab provides a comprehensive suite of tools and algorithms, that address various tasks such as *mesh cleaning*, *simplification*, *smoothing*, *texturing*, and *visualization*. These capabilities prove indispensable for the intricate manipulation and analysis of 3D data necessitated by the project's objectives.

In addition, MeshLab boasts a *user-friendly* interface and *extensive* documentation, significantly reducing the learning curve for team members and collaborators. This fosters effective collaboration and facilitates knowledge transfer throughout the project.

The choice to utilize MeshLab as the primary software for 3D mesh processing is underpinned by its *versatility*, *comprehensive toolset*, *user-friendly interface*, and *seamless integration* into the project's workflow. MeshLab's capabilities align with the intricate demands of 3D data manipulation, analysis, and visualization, making it an instrumental asset in achieving the project's objectives. This strategic decision reflects a commitment to *efficiency*, *accuracy*, and the successful *execution* of 3D data processing tasks.

5.2.6 Evaluation script

Python [Van Rossum and Drake, 2009], a *versatile* and *dynamically typed* programming language, was also utilized in certain aspects of this project. Python's ease of use, extensive libraries, and rapid development capabilities made it an ideal

choice for various scripting tasks, data preprocessing, and automation processes within the application.

The scripting language was employed to create a script that interfaces with MeshLab to compute per-face statistics[3]. This script can be run from the command line and can receive three distinct options, each of which, if omitted, holds a default value. In the context of conducted empirical analysis (Chapter 6), the following command has been used:

```
python ./face-stats.py --base-mesh=koma-inu_850.obj
   --target-mesh=koma-inu.obj
```

Possible options accepted by the script:

- `--base-mesh`, this option allows setting the base mesh. If left blank, the script defaults to the mesh `pallas_124.obj`. The option is used for retrieving the actual mesh from the `./Models`.
- `--target-mesh`, this option enables the setting of the target mesh. If omitted, the script utilizes `pallas_5000.obj`. The target mesh option is used for retrieving the actual mesh from the `./Models` directory, it will be used repeatedly for computing the distance with respect to the displaced samples.
- `--clean`, if this option is present it will start the cleaning procedure erasing all the elements in `./Evaluation/micro` and `./Evaluation/aniso`

The Python evaluation script will run the command line `gen-sample` (Sec. 5.2.2) *multiple* times giving input to the base mesh and the target mesh through two distinct loops. Both the loops will run the commands with `--factor` option that variates from 1,0 to 4,0.

The first loop will use the classical subdivision scheme (`--scheme=micro` or default), and the second is the anisotropic (`--scheme=aniso`). Subsequently, the script proceeds to access the directories of the new exported projected sample meshes (as discussed in Chapter 6) for both subdivision schemes:

- `./Output/Evaluation/micro/{base mesh_{faces}}`
- `./Output/Evaluation/aniso/{base mesh_{faces}}`

After opening the list of pertinent files, the script iterates through each sample, calculating the scalar statistics per face quality, according to the aspect ratio per face (using the dedicated method from the PyMeshLab library [Muntoni and Cignoni, 2021]). The resulting distances are then stored within the

[3]This metric will be extensively discussed in Chapter 6

same folder containing the samples.

In summary:

1. Generate 30 subdivided and displaced samples ($F \in [1.0, 4.0]$) for the μ-mesh scheme.
2. Generate 30 subdivided and displaced samples ($F \in [1.0, 4.0]$) for the anisotropic μ-mesh scheme.
3. Export scalar statistics by inradius/circumradius aspect ratio per face for the μ-mesh scheme.
4. Export scalar statistics by inradius/circumradius aspect ratio per face for the anisotropic μ-mesh scheme.

Upon completion of execution, two text files are generated and saved in the same directory of the samples, containing the per-face statistics of the displaced samples for both subdivision schemes.

5.2.7 Git

In the world of **version control systems** (VCS), the selection of an appropriate tool holds paramount significance for the success of any software development endeavor. In this context, the choice of Git [Linus Torvalds, 2005] as the primary version control system for this project merits elucidation.

The decision to employ Git is underpinned by several compelling factors. Firstly, Git is renowned for its *distributed nature*, affording collaborators the flexibility to work offline while preserving the full version history.

Secondly, Git boasts a robust and *efficient branching* mechanism. The ability to create isolated branches for feature development, bug fixing, and experimentation not only facilitates collaborative development but also enhances project organization and code stability. The branching model, epitomized by Git, aligns with the principles of agile software development [Martin, 2003], where iterative, incremental progress is the cornerstone.

Furthermore, Git's widespread adoption across the software development industry underscores its *reliability* and *robustness*. Its extensive user base and comprehensive documentation facilitate problem resolution and knowledge sharing, mitigating development bottlenecks.

The created repository is available on GitHub [Manuel Pagliuca, 2023] and encompasses all the files outlined in the structure detailed in Section 5.1.

5.3 Optimizations

In this chapter, optimizations that have been integrated towards the final stages of the project are outlined. These optimizations encompass both time complexity and conceptual optimizations for the μ-mesh schema (Sec. 5.3.2).

5.3.1 Line-casting

The line-casting operation is a fundamental process for finding the minimum displacements for the μ-vertices of the mesh subdivided in the direction of the target mesh.
The initial implementation of the line-casting algorithm for μ-vertices towards the faces of the *target* mesh was not an optimized approach but rather a brute-force one. The latter traced all possible line *combinations* originating from the μ-vertices and intersecting all faces of the target mesh.

It is evident that this approach consumes a lot of resources; specifically, the algorithm had a time complexity of $O(n^2)$. This algorithm did not consume too much time with low-resolution meshes (thousands of triangles) and served its purpose in the initial research phase of the anisotropic scheme.

However, for obtaining a better empirical analysis (Sec. 6), it was desired to use many models with higher resolutions (millions of triangles), and such complexity would have made this process extremely slow (we are talking about weeks if not months of rendering). This approach is detailed in the pseudo-code[4] of Algorithm 4.

In summary, member function $getDisplacements(...)$ is called on the subdivided mesh, with the target mesh passed as a parameter. The function iterates over the vertices of the subdivided mesh, passing position and normal information to another auxiliary function, $minimumDisplacement(...)$.

This second function is responsible for returning the **minimum displacement** from the origin of the μ-vertex with respect to the faces of the target mesh. Using the received positions and normals, two rays in opposite directions are created, which will be required for the line-casting. The line-cast is resolved using the method $intersectTriangle(...)$ of the Ray class.

This method returns a boolean that identifies whether an intersection along the ray has occurred. If this is the case, it checks if the returned displacement is smaller than the current minimum $minDisp$. If it is, then the current minimum is replaced with the newly found one. The result is returned and added to the displacement vector at the end of the loop.

[4]In the pseudo-code the elements in bold represent mathematical vectors.

Algorithm 4 Brute-force μ-vertices line-casting.

```
1: procedure MINIMUMDISPLACEMENT(origin, direction, Mesh target)
2:     minDisp ← ∞
3:     for Face f ∈ target.faces do
4:         v_0 ← (target.vertices[f.index[0]].pos)
5:         v_1 ← (target.vertices[f.index[1]].pos)
6:         v_2 ← (target.vertices[f.index[2]].pos)
7:
8:         fwdRay ← Ray(origin, direction)
9:         bwdRay ← Ray(origin, −direction)
10:
11:        fwdDisp, bwdDisp ← 0
12:        fwdIntersect ← fwdRay.intersectTriangle(v_0, v_1, v_2, fwdDisp)
13:        bwdIntersect ← bwdRay.intersectTriangle(v_0, v_1, v_2, bwdDisp)
14:
15:        if forwardIntersect then
16:            if |minDisp| < |forwardDisp| then
17:                minDisp = forwardDisp
18:            end if
19:        end if
20:
21:        if backwardRay then
22:            if |minDisp| < |backwardDisp| then
23:                minDisp = −backwardDisp
24:            end if
25:        end if
26:    end for
27:
28: return minDisp
29:
30: end procedure
31:
32: procedure GETDISPLACEMENTS(Mesh target)
33:    displacements ← [ ]
34:
35:    for Vertex v ∈ vertices do
36:        displacements.push(minimumDisplacement(v.pos, v.norm, target))
37:    end for
38: end procedure
```

To be precise, the complexity of this algorithm is $O(|\text{vertices}| \cdot |\text{target.faces}|)$, which, given the size of the meshes, falls into the class of quadratic functions. Our solution reduces the complexity from $O(n^2)$ to $O(n \cdot \log n)$ (*pseudo-linear*) by employing specific techniques that allow us to reduce the number of intersections with the faces of the target mesh to a significant neighborhood around the considered μ-vertex.

Introduction of new fields in the data structures representing the mesh:

- $maxAxis$ and R in the Mesh class.
- $posMiddle$ in the Face class.

In the Mesh class, the integer $maxAxis$ represents the axis on which the mesh extends the most, the integer maps respectively the X, Y, and Z axes to 0,1, and 2. Instead, the decimal R represents the maximum half-extension on the $maxAxis$ of the largest bounding box of all faces.

In the Face class, the decimal $posMiddle$ denotes the midpoint in the range between the face vertices with minimum and maximum coordinates on the $maxAxis$ (axis on which the mesh extends the most).

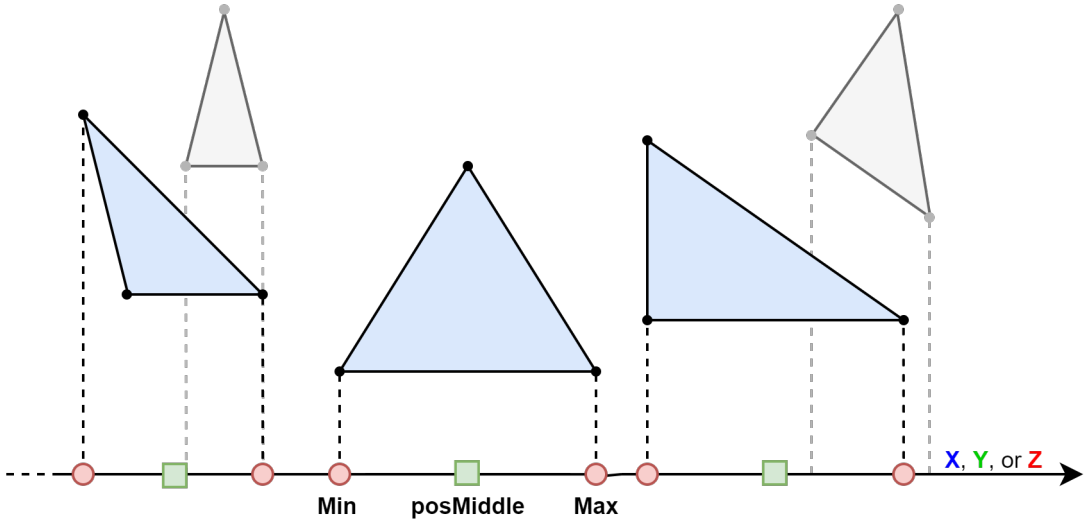


Figure 29: Finding R and $posMiddle$ by considering faces coordinates projected on the axis of maximum extension ($maxAxis$).

Algorithm 5 Update $posMiddle$ and R.

```
1: procedure updatePosMiddleAndR
2:     R = −∞
3:
4:     for Face f ∈ faces do
5:         pos0 ← vertices[f.index[0]].pos[maxAxis]
6:         pos1 ← vertices[f.index[1]].pos[maxAxis]
7:         pos2 ← vertices[f.index[2]].pos[maxAxis]
8:
9:         posMax ← maxFloat3(pos0, pos1, pos2)
10:        posMin ← minFloat3(pos0, pos1, pos2)
11:        halfExt ← (posMax − posMin)/2
12:
13:        f.posMiddle ← (posMax + posMin)/2
14:
15:        if R < halfExt then R = halfExt
16:        end if
17:    end for
18: end procedure
```

The general scheme is not particularly changed from the brute-force implementation, there are two extra steps before extracting the displacements:

1. Computing and updating the R and $posMiddle$ fields (see Algorithm 5).
2. Sorting the mesh faces by $posMiddle$ (non-decreasing).
3. Iterating on the μ-vertices of the subdivided mesh and calling the $minimumDistance(...)$ function, passing position, normal, and target mesh as parameters (identical step as the brute-force approach).

The real radical change occurs in the new implementation of $minimumDisplacement(...)$, the explanation in words is fundamental to such optimization the code unfortunately can be cryptic to decipher without clues. Outline of the $minimumDisplacement()$ method:

1. Definition of useful variables and constants:
    - DIST_MAX, constant representing the maximum distance to consider casting. This time it will not be ∞ but $\frac{1}{100}$ of the target mesh bounding box, this is to avoid the infinite loop.

- *minDisp*, a variable that will contain the minimum displacement at the end of the procedure.
- *line*, an instance of a Line object, this object represents a line and contains in itself the member method $intersectTriangle(...)$ to perform line casting.
- *posOrigin*, a variable that contains the coordinate of the origin with respect to the axis where the mesh extends the most ($maxAxis$), is initialized to $origin[maxAxis]$
- Two indices i and j, initialized out-of-bound to -1 and $|target.faces|$ that will later be updated to the indices of the faces before and after the $posMiddle$ equal to *posOrigin*, through a binary search.

2. Binary search to set the values of i and j indices to those of the target mesh faces that have a $posMiddle$ less (for i) and greater (for j) than $originPos$, respectively.

3. Binary search for minimum displacement.

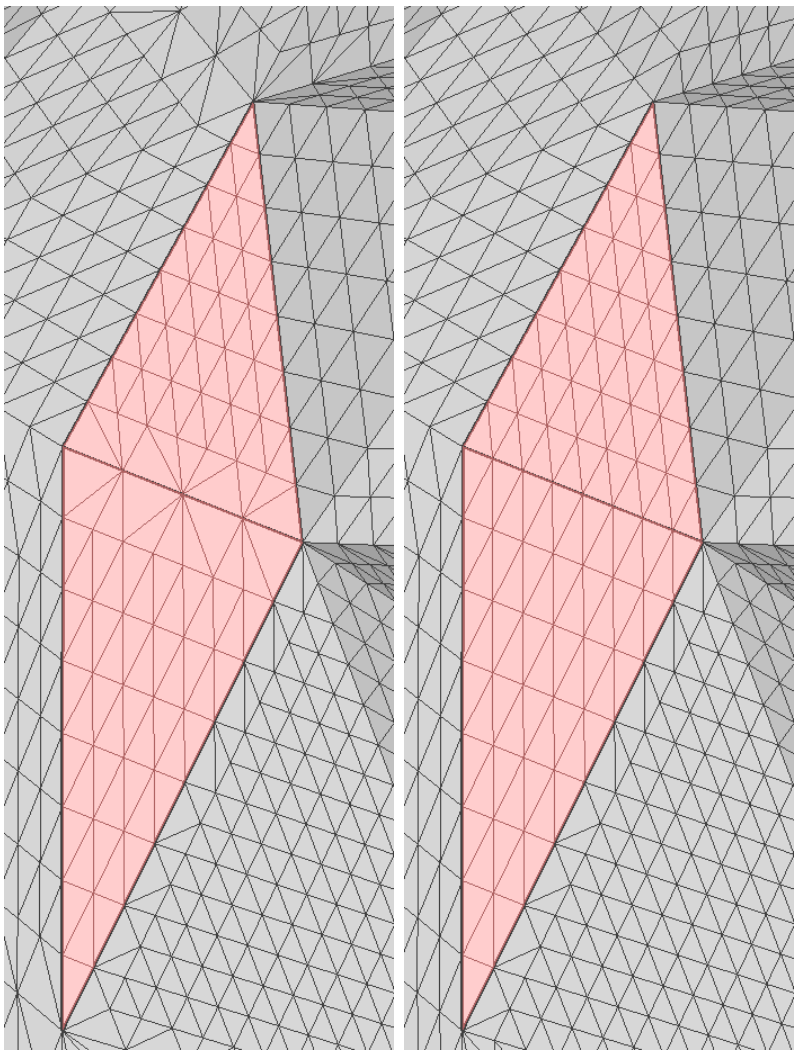
   - The condition of this loop is that as long as indexes i and j are in-bound, then it can continue the search, otherwise it stops.
   - Important note to remember, the $updatePosMiddleAndR()$ function after it is run sorts the mesh faces using $posMiddle$ as a criterion. This is a very important step for the optimization to work.
   - The search operation traverses both the left and right of the *target.posMiddle*, conducting **early rejection tests** on the *posMiddle* indexed by i and j. For both indices, we validate whether the *posMiddle*, in consideration of the *posOrigin*, falls within the interval defined by R. This validation helps us identify the triangle as a face of *reasonable consideration* for subsequent intersection testing. The significant breakthrough lies in the fact that the vector is sorted based on *posMiddle*. Consequently, if the early rejection test were to fail for one of the indices, it would imply that elements preceding i or succeeding j would inherently possess too low or too high *posMiddle* values. As a result, it becomes possible to terminate the search in either direction of the array, whether leftward or rightward.

Algorithm 6 Logarithmic implementation of *minimumDisplacement*(...)

```
1: procedure MINIMUMDISPLACEMENT(origin, direction, Mesh target)
2:     DIST_MAX ← bbox.diagonal · 0.01
3:     minDisp ← DIST_MAX
4:     line ← Line(origin, direction)
5:     posOrigin ← origin[maxAxis]
6:
7:     i ← −1
8:     j ← |target.faces|
9:
10:    Binary Search of target.posMiddle
11:    Set indices i and j as predecessor and successor of target.posMiddle
12:
13:    while i ≥ 0 ∨ j < |target.faces| do
14:        if i ≥ 0 then
15:            if early rejection test for the left side of the array then
16:                i ← −1
17:            else
18:                target.intersectTriangle(i − −, line, minDisp)
19:            end if
20:        end if
21:
22:        if j < |target.faces| then
23:            if early rejection test for the right side of the array then
24:                j ← |target.faces|
25:            else
26:                target.intersectTriangle(j + +, line, minDisp)
27:            end if
28:        end if
29:    end while
30:
31:    if minDisp == DIST_MAX then
32:        minDisp = 0
33:    end if
34:
35: return minDisp
36:
37: end procedure
```

5.3.2 Micro-Mesh common edge fix

After the μ-mesh scheme is enforced n times, at the end of this operation the pattern will turn out to be respected for all faces of the mesh base before it is subdivided again. However, it turns out that this strategy can *occasionally* produce an edge that is unnecessarily coarser.



(a) Subscaled common edge (b) Fixed common edge

Figure 30: Common edge with coarser level of subdivision (before and after fix).

Let us consider the *adjacent* faces highlighted in Figure 30a, we have two adjacent faces with configuration $2^3 \times 2^3 \times 2^2$, where 2^2 is the level of subdivision of the edge in common. It turns out to be immediate to understand that the coarser level of

subdivision is not appealing on the common edge. Because the internal number of triangles has no reason to be made in-homogeneous at one point when for most of the surface area covered by the triangles there is a 2^3 subdivision.

The cause of this phenomenon is a side effect of the application of the μ-mesh subdivision scheme (Eq. 3), that even if applied several times on the same mesh until convergence is reached, occasionally will leave adjacent edges of similar triangles less subdivided.

The solution is quite simple, we need to add a correction routine to be executed after the scheme reinforcement to get the correct edge topology (Fig. 30b). The correction algorithm (Algo. 7) iterates on the edges of the subdivided mesh, ignoring the open edges, in the case where the **maximum subdivision indices** of both the adjacent faces coincide, and the shared edge has a subdivision index less than 1 with respect to the maximum. In that case, the subdivision index of the shared edge will have to be equated with that of the maximum edges.

Algorithm 7 Adjacent edge fix routing for μ-mesh scheme.

```
1:  procedure fixEdgesSubdivisionIndicesMicromesh
2:      for Edge e ∈ Edges do
3:
4:          if e.faces[0] == −1 ∨ e.faces[1] == −1 then
5:              continue
6:          end if
7:
8:          eMax0 ← getFaceSubdivisionIndex(e.faces[0])
9:          eMax1 ← getFaceSubdivisionIndex(e.faces[1])
10:
11:         if eMax0 == eMax1 ∧ e.subdivisions + 1 == eMax0 then
12:             e.subdivisions = eMax0
13:         end if
14:     end for
15: end procedure
```

5.3.3 Arbitrary number of Micro-faces

Initially, the subdivision was done using a special slider in the GUI (Sec. 5.2.1), which allowed the user to set the value of the target edge length (Eq. 1) and immediately afterward perform subdivision according to the selected pattern.

This may make sense for those who developed the project, but not much for an outside user to whom the work is being exposed for the first time. Instead, it is much more useful to provide a slider that would allow the user to set the number

of desired μ-faces. The solution is to perform a binary search of the target edge length required to find a quantitative approximation of the μ-faces entered by the user.

The implementation of this binary search requires an auxiliary function $predictMicroFaces(...)$ which receives as a parameter the subdivision scheme and the target edge length, and is able to predict with specific formulas, the needed target edge length for the given μ-faces. More formally, we can define the two prediction equations by considering the following variables:

- k the maximum subdivision index of a face.
- h the minimum subdivision index of a face.
- $aniso$ is the level of anisotropy of a face, expressed as the difference between k and h (this variable is available only for the anisotropic μ-meshscheme).

It should be remembered that before the prediction is calculated, the reinforcement (and edge fix for μ-mesh scheme) of the chosen subdivision scheme is carried out, based on the provided target edge length. Only after the subdivision indices are assigned will it be possible to compute the prediction for the μ-mesh scheme (Eq. 9), and for the anisotropic μ-mesh scheme (Eq. 10).

$$\sum_{f \in \text{Faces}} 2^k \cdot 2^k \tag{9}$$

$$\sum_{f \in Faces} 2^h \cdot (2^k + 2^{aniso} - 1) \tag{10}$$

Having provided this information, it will now be easier to understand to read the pseudocode of the dichotomous search.

Algorithm 8 Binary search of target edge length given μ-faces target.

```
1: procedure binarySearchTargetEdgeLength(target, Scheme scheme, a, b)
2:     aFcs ← −1
3:     bFcs ← −2
4:     patience = 10
5:
6:     while true do
7:         c ← (a + b)/2.0
8:         cFcs ← predictMicroFaces(scheme, c)
9:
10:        if (cFcs == aFcs ∨ cFcs == bFcs) ∧ patience − − == 0 then
11:            if |target − aFcs| < |target − bFcs| then return a
12:            else return b
13:            end if
14:        end if
15:
16:        if cFcs < target then
17:            bFcs = cFcs
18:            b = c
19:        else
20:            aFcs = cFcs
21:            a = c
22:        end if
23:
24:        if cFcs == target then return c
25:        end if
26:    end while
27:
28: end procedure
```

The search function operates as follows: it receives three parameters, the desired number of μ-faces, and the a and b extreme intervals (as lengths) over which the search will be performed. The extremes are initialized by the caller to 0 and 10 times the diagonal of the base mesh bounding box (a very large number but yet proportional).

Before the binary search cycle, the variables that will contain the number of μ-faces for the edge length a and b are created; these are $aFcs$ and $bFcs$.

Within the loop, we look for the midpoint of the currently covered interval, c, and save in the variable $cFcs$ the prediction of mfs in the edge length c (given the model).

Inside the loop, three very important checks occur in sequence:

- Check whether the number of μ-faces expected at the midpoint c is equal to the number of μ-faces expected at either end of the current interval. If this is true, it means that the edge length c yields the same amount of faces as a or b. The extreme that predicts the number of μ-faces closest to the target is returned.
- If the predicted number of μ-faces at the midpoint is less than the desired target, which means you need to move the search to the right of the midpoint; otherwise you need to move the search to the left.
- If the number of μ-facespredicted at the midpoint exactly matches the desired target, then c is the desired edge length (very rare case).

The variable *patience* is used to make sure that the condition is actually verified (testing it 10 times); since we are dealing with decimal numbers and step functions, it is best to be cautious.

Chapter 6

Empirical analysis

The objective of this thesis is to assess the validity of the anisotropic scheme with respect to the isotropic one. To accomplish this, three different comparison approaches have been considered for the samples.

- Average of per-face *inradius/circumradius* aspect ratio (with visual comparison).
- Comparison of the coefficients of variation inherent to the areas of the faces (expressed as a percentage).

$$CV = \frac{\sigma}{|\mu|} \cdot 100\% \tag{11}$$

Where μ is the *arithmetic mean*, σ is *standard deviation*, N is the size of the samples, and x_i states the individual element of the sample.

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2} \tag{12}$$

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{13}$$

- Visual comparison of the two displaced subdivision schemes.

Examining error behavior between the two subdivision schemes involves two sample sets (subdivided meshes) with the same number of elements, which is 30. The first set comprises models subdivided using the μ-mesh scheme, and the second set adopts the anisotropic μ-mesh scheme. Each mesh element in both groups

represents the same model, but the way they are subdivided is different for all of them (i.e., they will have the same number of faces and vertices, but a different number of μ-faces and μ-vertices). As a result, each element within one group possesses a counterpart in the other, sharing an almost equal number of μ-faces, diverging slightly in arrangement due to different partitioning strategies.

To ascertain an equivalent count of μ-faces, the process involves determining a target edge length (Sec. 3.2) that equalizes the input amount of μ-faces. This is obtained by performing a special binary search discussed in Section 8. The sample generation process is done with an evaluation script (Sec. 5.2.6) which calls multiple times a command exposed by the application when is run in the terminal.

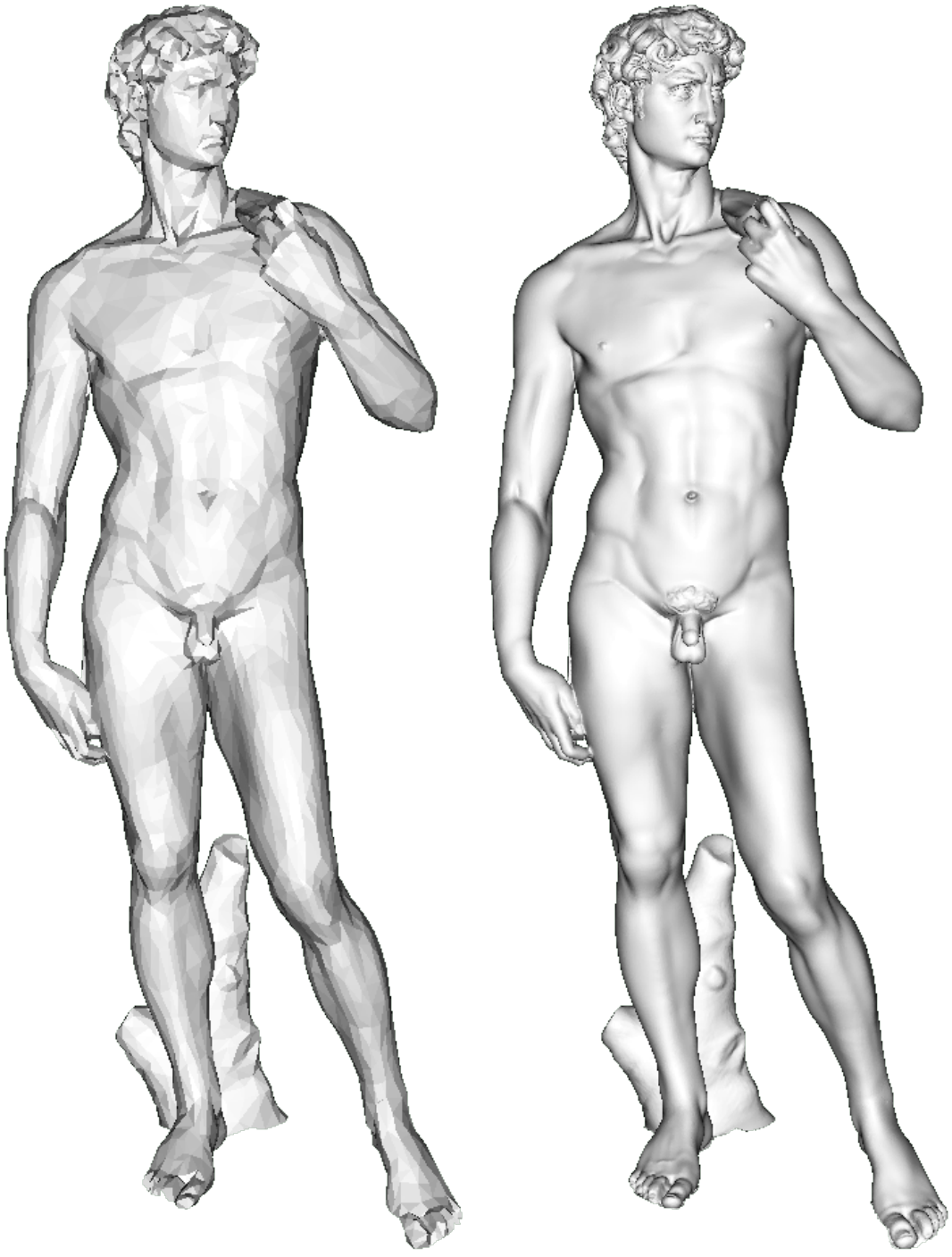
6.1 Target and base meshes

Several models were chosen to conduct this experiment, most of which have in common that they consist of triangular meshes and are dense with faces. However, another feature of these models is that they are 3D scans of significant cultural objects. I used two open-source distribution sources for these, the project "Three D Scans" [Oliver Laric, 2012], only for the "Homo Heidelbergensis" model (Sec. 6.2.5), and "Scan The World" [Beck and d'Antona, 2014] for the rest of the models.

The line-casting algorithm was optimized (Sec. 5.3.1) by taking it from time complexity $O(n^2)$ to $O(n \cdot \log n)$, using a particular R-factor that consists of the maximum *deviation* (divided by 2) between the x-components of each triangle in the mesh. This R-factor is used to exclude all those triangles that it does not make sense to consider in line-casting; the problem is that if a highly heterogeneous mesh has very large triangles, these will cancel out the R-factor and slow down the casting execution (too far triangles are tested).

For this reason, the mesh was cleaned up, in particular, the pedestals of some of the models considered were removed, as they had very large triangles. In addition to these procedures, all the original formats were unified with the `OBJ` format [Wavefront Technologies, 1990], both of which were made with MeshLab. It's noteworthy that the application is capable of effectively managing both the OBJ format and OFF [Geometry Center, 1998].

The target mesh models have been subjected to a decimation process of the original faces. This reduction was carried out by employing the *Quadric Edge Collapse Decimation* [Garland and Heckbert, 1997] technique within MeshLab [Cignoni et al., 2008]. A practical example of a mesh base generated by target decimation can be seen in Figure 31a. For a complete overview of all these models and their data, see Table 1.



(a) Base mesh — 9,847 faces (b) Target mesh — 984,888 faces

Figure 31: Michelangelo's David input meshes.

	Base		Target	
Model	Faces	Vertices	Faces	Vertices
Dragon	23,488	11,744	2,349,078	1,174,539
Borghese Ares	17,366	8,743	1,736,666	869,124
Dancing Faun	15,110	7,577	1,511,081	755,811
Michelangelo's David	9,847	4,962	984,888	493,189
Homo Heidelbergensis	4,156	2,078	413,258	206,690
Koma Inu	850	427	85,000	45,502

Table 1: Comparison table of target and base meshes.

6.2 Scheme comparison

For the purpose of sample comparison, a dedicated evaluation script was employed. This script invokes the command-line application iteratively, supplying it with the necessary arguments (further elaborated in Section 5.2.6).

```
python ./face-stats.py --base-mesh=decimated_model.obj
   --target-mesh=original_model.obj
```

The script is responsible for generating 30 samples per scheme, which consist of subdivided base meshes shifted to the target mesh (displaced mesh). The **subdivision level** increases as samples are generated, and consequently the number of μ-faces. The criterion for determining the number of subdivisions is based on a multiplicative factor F multiplied by the number of μ-faces in the target mesh.

This factor ranges from 1.0 to 4.0, with an incremental step of 0.1. In total you will have 60 samples, the script will output in a text file the statistics per-face quality based on the inradius/circumradius aspect ratio of the circles inscribed and circumscribed to the triangle.

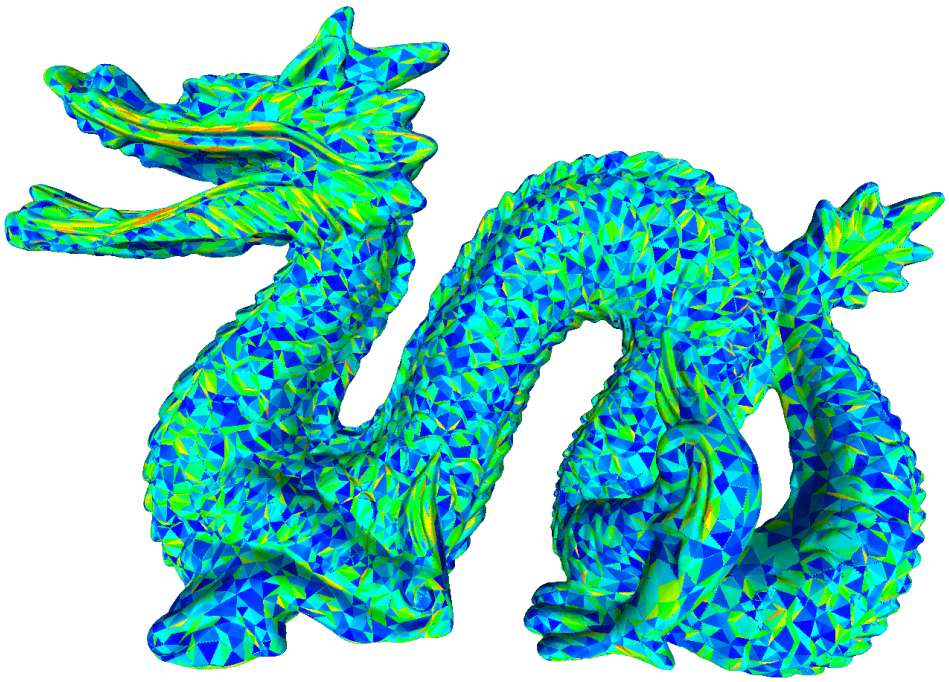
The average of the mean values of these statistics is used for the final comparison (Sec. 6.3), ultimately showing the percentage of improvement/worsening of the anisotropic pattern compared to the isotropic pattern.

The scheme comparison tables are organized so that the face quality statistics are visible under the "Face quality" column (mean, standard deviation, and variance), while a separate column shows the second metric, the coefficient of variation. Following the table are two face-quality comparison images

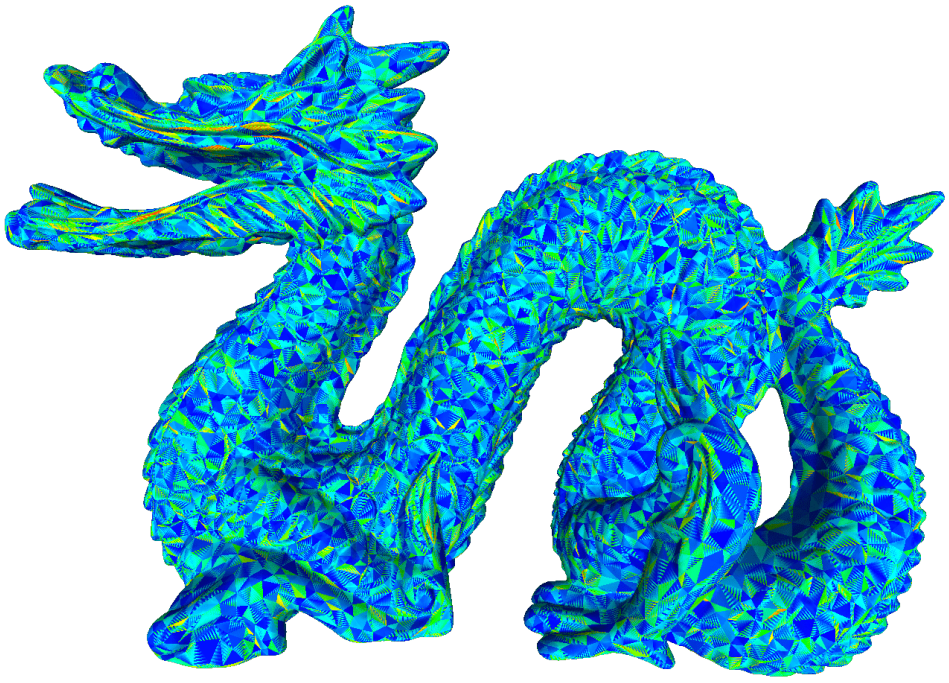
6.2.1 Dragon

	Isotropic				Anisotropic			
	Face quality			CV (%)	Face quality			CV (%)
F	Mean	σ	σ^2		Mean	σ	σ^2	
1.0	0.40	0.17	0.03	62.07	0.51	0.16	0.03	36.82
1.1	0.40	0.17	0.03	62.76	0.51	0.16	0.03	36.28
1.2	0.40	0.17	0.03	63.29	0.51	0.16	0.03	35.58
1.3	0.40	0.17	0.03	63.27	0.51	0.16	0.03	35.36
1.4	0.40	0.17	0.03	63.14	0.51	0.16	0.03	35.15
1.5	0.40	0.17	0.03	62.52	0.51	0.16	0.03	35.35
1.6	0.40	0.17	0.03	62.09	0.51	0.16	0.03	35.49
1.7	0.40	0.17	0.03	61.19	0.51	0.16	0.03	36.03
1.8	0.41	0.17	0.03	59.88	0.51	0.16	0.03	35.95
1.9	0.41	0.17	0.03	59.47	0.51	0.16	0.03	36.16
2.0	0.41	0.17	0.03	58.63	0.51	0.16	0.03	36.33
2.1	0.41	0.17	0.03	58.08	0.51	0.16	0.03	36.42
2.2	0.41	0.17	0.03	57.28	0.51	0.16	0.03	36.31
2.3	0.41	0.17	0.03	56.80	0.51	0.16	0.03	36.47
2.4	0.41	0.17	0.03	56.74	0.51	0.16	0.03	36.48
2.5	0.41	0.17	0.03	56.64	0.50	0.16	0.03	36.69
2.6	0.41	0.17	0.03	56.79	0.50	0.16	0.03	36.73
2.7	0.41	0.17	0.03	57.07	0.50	0.16	0.03	36.91
2.8	0.41	0.16	0.03	57.26	0.50	0.16	0.03	36.92
2.9	0.41	0.16	0.03	57.44	0.50	0.16	0.03	36.97
3.0	0.41	0.16	0.03	57.56	0.50	0.16	0.03	36.98
3.1	0.41	0.17	0.03	58.17	0.50	0.16	0.03	36.95
3.2	0.41	0.17	0.03	58.50	0.50	0.16	0.03	36.78
3.3	0.41	0.16	0.03	58.69	0.50	0.16	0.03	36.68
3.4	0.41	0.16	0.03	59.07	0.50	0.16	0.03	36.65
3.5	0.41	0.16	0.03	59.35	0.50	0.16	0.03	36.22
3.6	0.41	0.16	0.03	59.68	0.50	0.16	0.03	36.08
3.7	0.41	0.16	0.03	59.89	0.51	0.16	0.03	35.99
3.8	0.41	0.16	0.03	60.07	0.51	0.16	0.03	35.72
3.9	0.41	0.16	0.03	60.26	0.51	0.16	0.03	35.71
4.0	0.41	0.16	0.03	60.48	0.51	0.16	0.03	35.57

Table 2: Dragon — Statistical comparison of subdivision patterns.

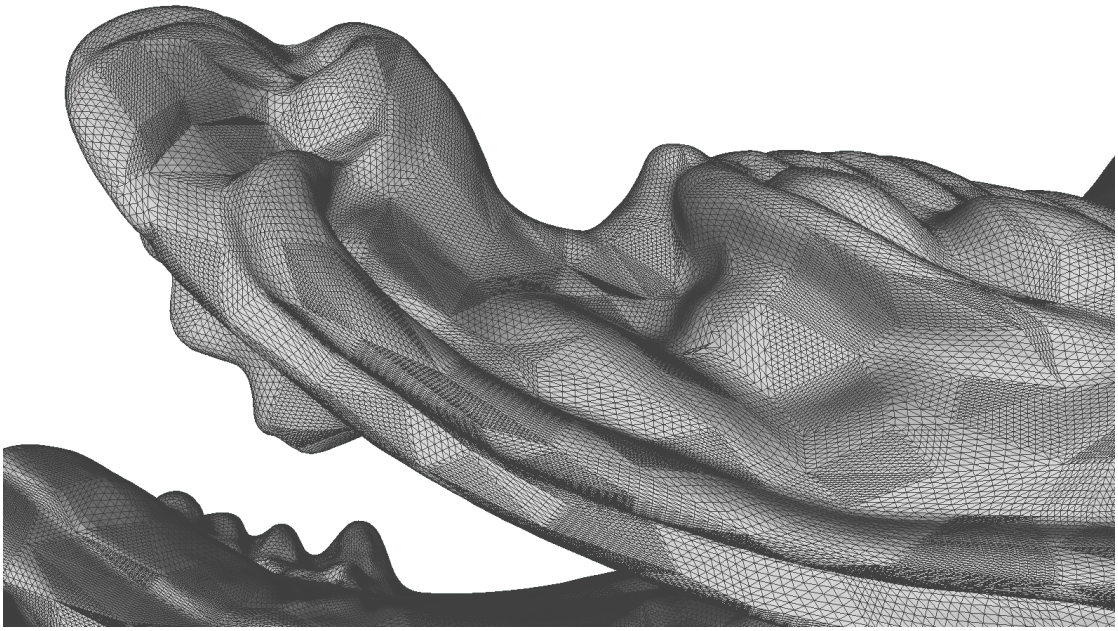


(a) Isotropic

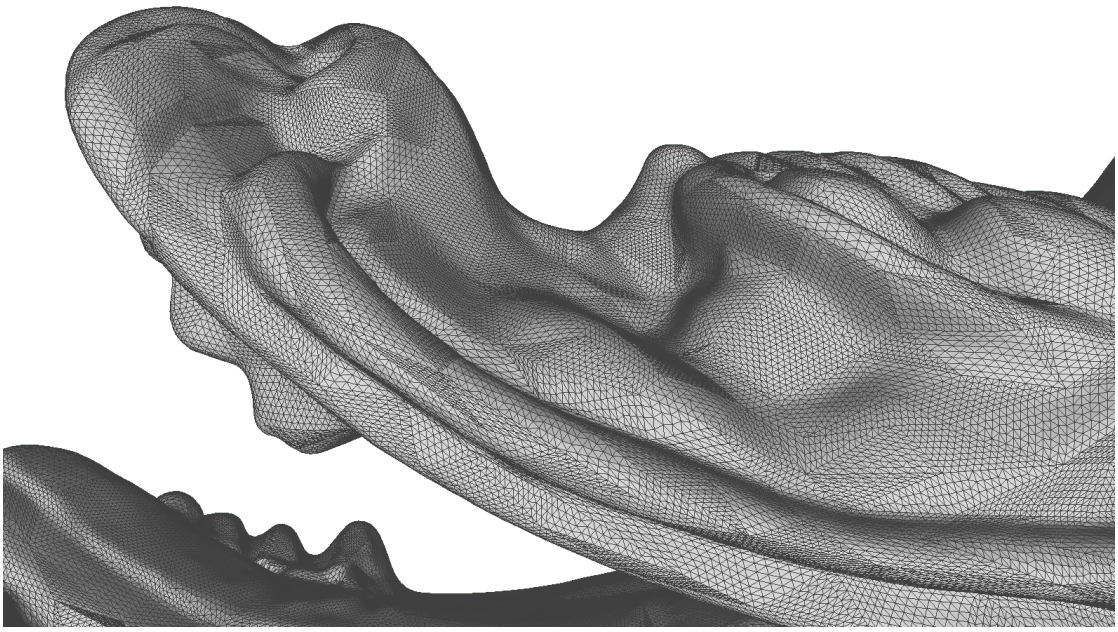


(b) Anisotropic

Figure 32: Dragon — Graphical comparison of face quality.



(a) Micro-Mesh



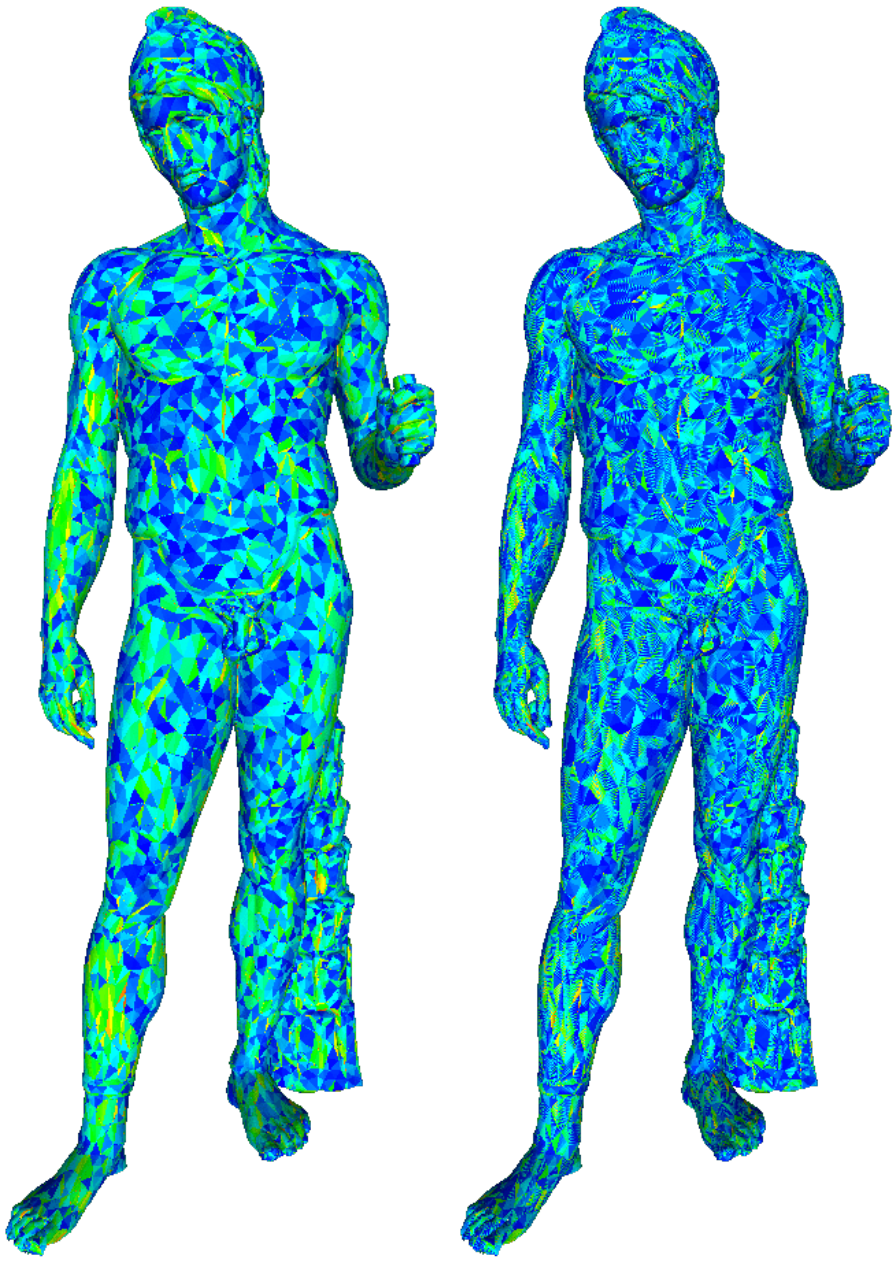
(b) Anisotropic Micro-Mesh

Figure 33: Dragon — Graphical comparison of displaced subdivision schemes.

6.2.2 Borghese Ares

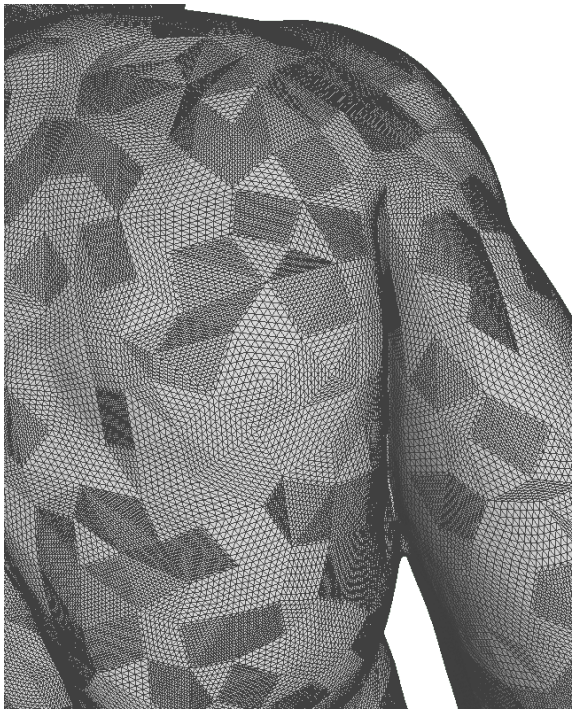
	Isotropic				**Anisotropic**			
	Face quality			CV (%)	Face quality			CV (%)
F	Mean	σ	σ^2		Mean	σ	σ^2	
1.0	0.43	0.17	0.03	61.55	0.53	0.16	0.02	34.52
1.1	0.43	0.17	0.03	62.55	0.53	0.16	0.02	33.83
1.2	0.43	0.17	0.03	62.13	0.53	0.16	0.02	33.61
1.3	0.44	0.17	0.03	61.50	0.53	0.16	0.02	33.31
1.4	0.44	0.17	0.03	60.68	0.53	0.16	0.02	33.53
1.5	0.44	0.17	0.03	59.29	0.53	0.16	0.02	33.46
1.6	0.44	0.17	0.03	57.70	0.53	0.16	0.02	33.63
1.7	0.45	0.17	0.03	55.78	0.53	0.16	0.02	33.78
1.8	0.45	0.17	0.03	54.37	0.53	0.16	0.02	34.07
1.9	0.45	0.16	0.03	53.24	0.53	0.16	0.02	34.07
2.0	0.45	0.16	0.03	52.18	0.53	0.16	0.02	34.26
2.1	0.45	0.17	0.03	51.48	0.53	0.16	0.02	34.43
2.2	0.45	0.17	0.03	51.21	0.53	0.15	0.02	34.39
2.3	0.45	0.17	0.03	50.86	0.53	0.15	0.02	34.53
2.4	0.45	0.16	0.03	51.48	0.53	0.15	0.02	34.78
2.5	0.45	0.16	0.03	51.96	0.53	0.15	0.02	34.71
2.6	0.45	0.16	0.03	52.70	0.53	0.16	0.02	34.91
2.7	0.44	0.16	0.03	53.43	0.53	0.15	0.02	34.71
2.8	0.44	0.16	0.03	54.19	0.53	0.15	0.02	34.66
2.9	0.44	0.16	0.03	54.93	0.53	0.15	0.02	34.59
3.0	0.44	0.16	0.03	55.38	0.53	0.15	0.02	34.67
3.1	0.44	0.16	0.03	56.12	0.53	0.15	0.02	34.54
3.2	0.44	0.16	0.03	56.64	0.53	0.15	0.02	34.47
3.3	0.44	0.16	0.03	57.16	0.53	0.16	0.02	34.35
3.4	0.44	0.16	0.03	57.88	0.53	0.16	0.02	34.32
3.5	0.44	0.16	0.03	58.39	0.53	0.16	0.02	34.07
3.6	0.44	0.16	0.03	58.95	0.53	0.16	0.02	33.84
3.7	0.44	0.16	0.03	59.39	0.53	0.16	0.02	33.60
3.8	0.44	0.16	0.03	59.65	0.53	0.16	0.02	33.45
3.9	0.44	0.16	0.03	60.06	0.53	0.16	0.02	33.37
4.0	0.44	0.16	0.03	60.47	0.53	0.16	0.02	33.26

Table 3: Borghese Ares — Statistical comparison of subdivision patterns.

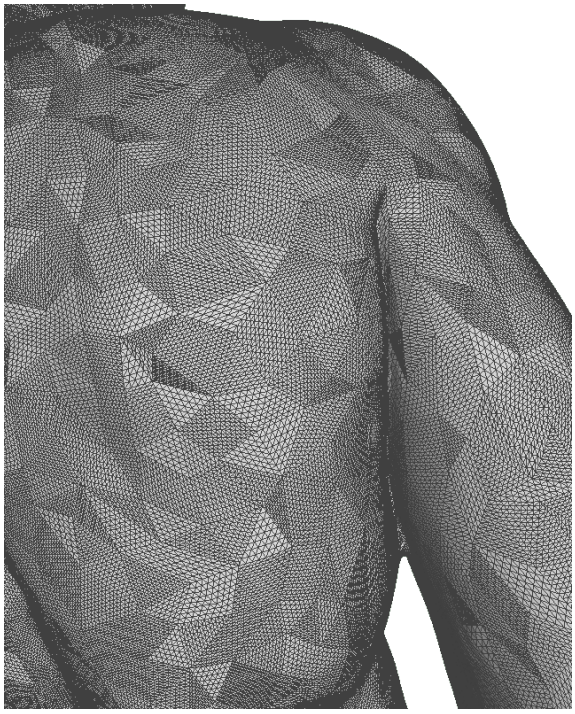


(a) Isotropic (b) Anisotropic

Figure 34: Borghese Ares — Graphical comparison of face quality.



(a) Micro-Mesh



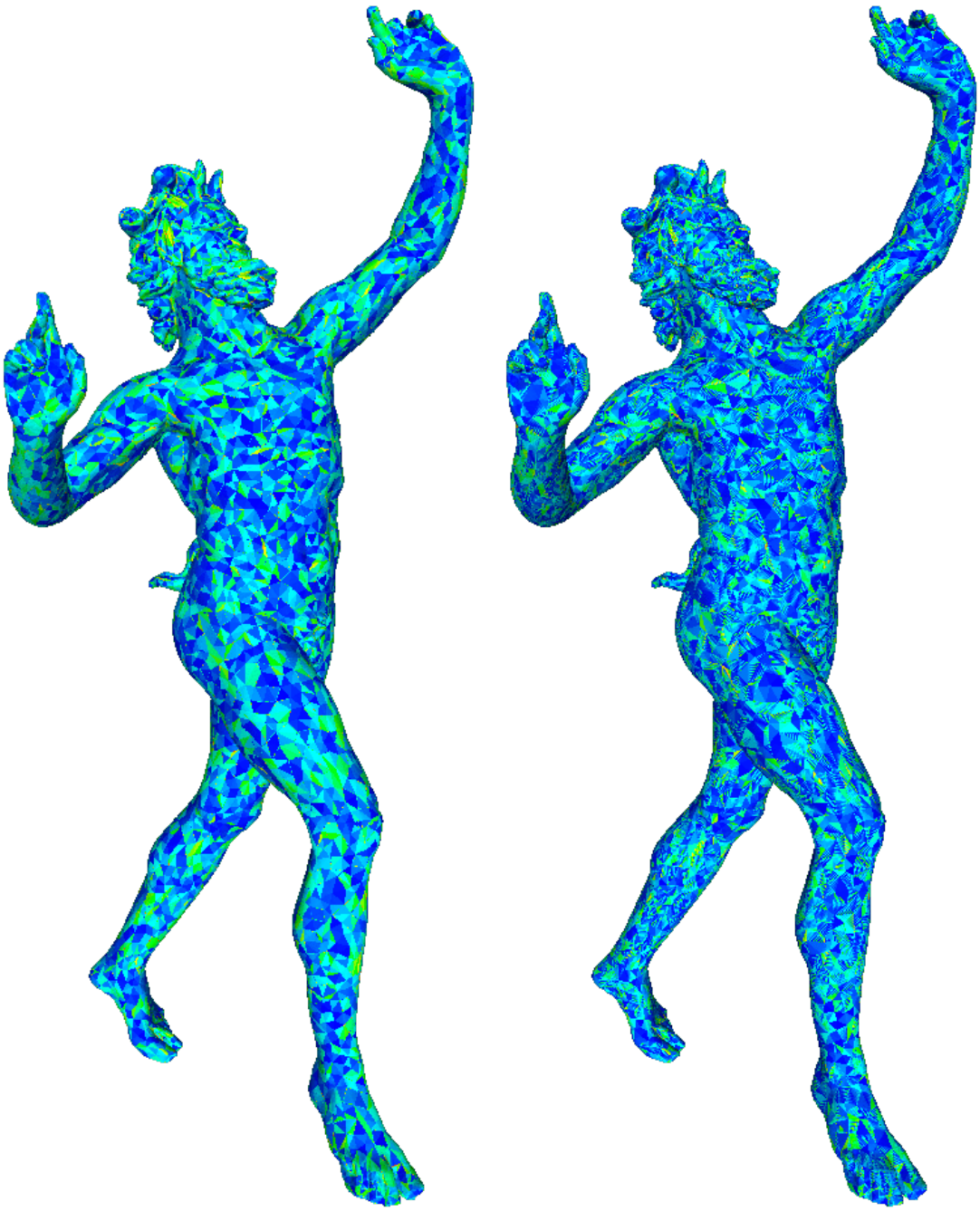
(b) Anisotropic Micro-Mesh

Figure 35: Ares Borghese — Graphical comparison of displaced subdivision schemes.

6.2.3 Dancing Faun

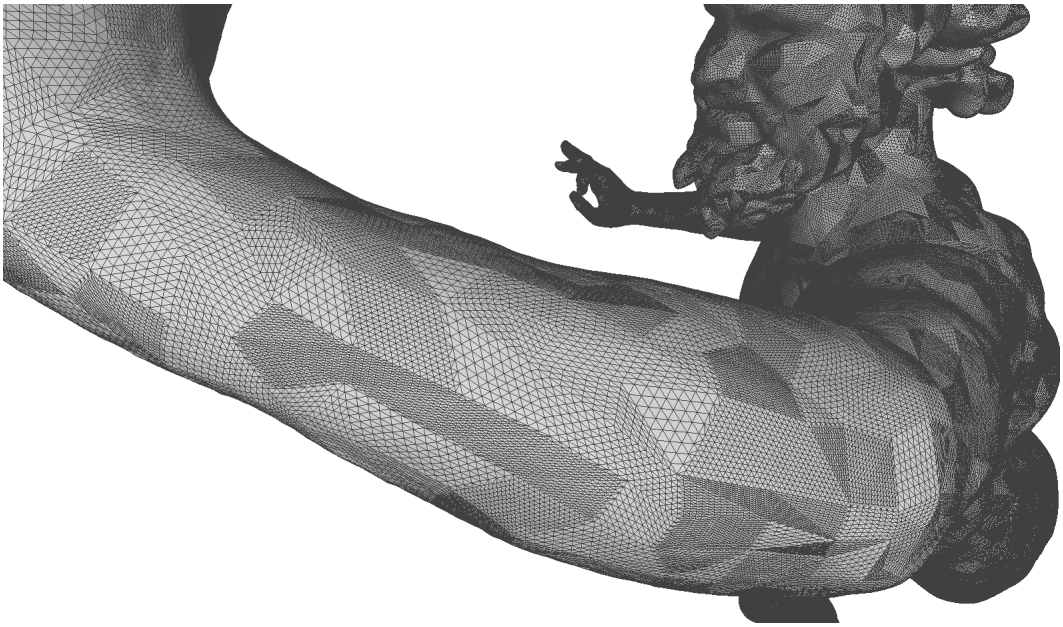
	Isotropic				Anisotropic			
	Face quality			CV (%)	Face quality			CV (%)
F	Mean	σ	σ^2		Mean	σ	σ^2	
1.0	0.46	0.17	0.03	60.02	0.54	0.15	0.02	33.55
1.1	0.46	0.17	0.03	60.52	0.46	0.16	0.03	32.93
1.2	0.46	0.16	0.03	60.60	0.54	0.15	0.02	32.47
1.3	0.46	0.16	0.03	59.97	0.46	0.16	0.03	32.22
1.4	0.46	0.16	0.03	58.92	0.54	0.15	0.02	32.09
1.5	0.47	0.16	0.03	57.65	0.47	0.16	0.03	31.94
1.6	0.47	0.16	0.03	56.14	0.55	0.15	0.02	32.20
1.7	0.47	0.16	0.03	54.39	0.48	0.16	0.03	32.33
1.8	0.47	0.16	0.03	52.76	0.55	0.15	0.02	32.43
1.9	0.47	0.16	0.03	50.82	0.48	0.16	0.03	32.60
2.0	0.48	0.16	0.03	49.54	0.55	0.15	0.02	32.75
2.1	0.48	0.16	0.03	48.95	0.48	0.16	0.03	33.00
2.2	0.48	0.16	0.03	48.64	0.55	0.15	0.02	33.15
2.3	0.48	0.16	0.03	48.53	0.47	0.17	0.03	33.47
2.4	0.48	0.16	0.03	49.04	0.54	0.15	0.02	33.76
2.5	0.47	0.16	0.03	49.78	0.47	0.17	0.03	33.77
2.6	0.47	0.16	0.03	50.85	0.54	0.15	0.02	33.88
2.7	0.47	0.16	0.03	51.68	0.47	0.17	0.03	33.95
2.8	0.47	0.16	0.03	52.35	0.54	0.15	0.02	33.87
2.9	0.47	0.16	0.03	53.16	0.54	0.15	0.02	33.86
3.0	0.47	0.16	0.03	54.06	0.47	0.17	0.03	33.80
3.1	0.47	0.16	0.03	54.61	0.54	0.15	0.02	33.73
3.2	0.47	0.16	0.02	55.61	0.46	0.17	0.03	33.57
3.3	0.47	0.16	0.02	56.35	0.54	0.15	0.02	33.37
3.4	0.47	0.16	0.02	56.85	0.46	0.17	0.03	33.25
3.5	0.47	0.16	0.02	57.16	0.54	0.15	0.02	33.00
3.6	0.47	0.16	0.02	57.67	0.46	0.17	0.03	32.94
3.7	0.46	0.16	0.02	58.22	0.54	0.15	0.02	32.96
3.8	0.46	0.16	0.02	58.54	0.46	0.17	0.03	32.71
3.9	0.46	0.16	0.02	58.90	0.54	0.15	0.02	32.45
4.0	0.46	0.16	0.02	59.08	0.45	0.17	0.03	32.18

Table 4: Dancing Faun — Statistical comparison of subdivision patterns.

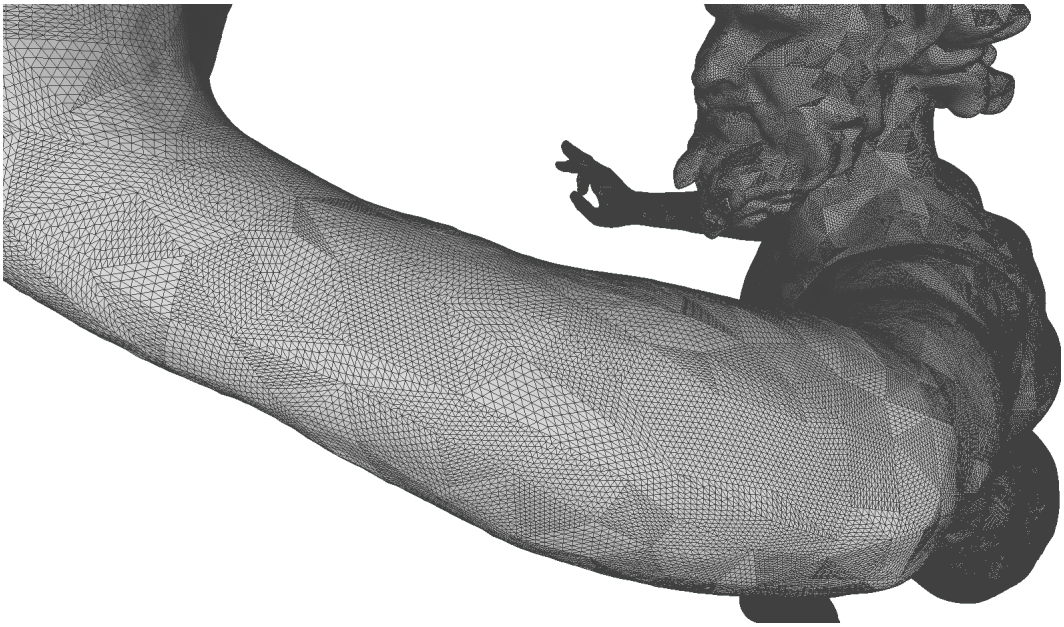


(a) Isotropic (b) Anisotropic

Figure 36: Dancing Faun — Graphical comparison of face quality.



(a) Micro-Mesh



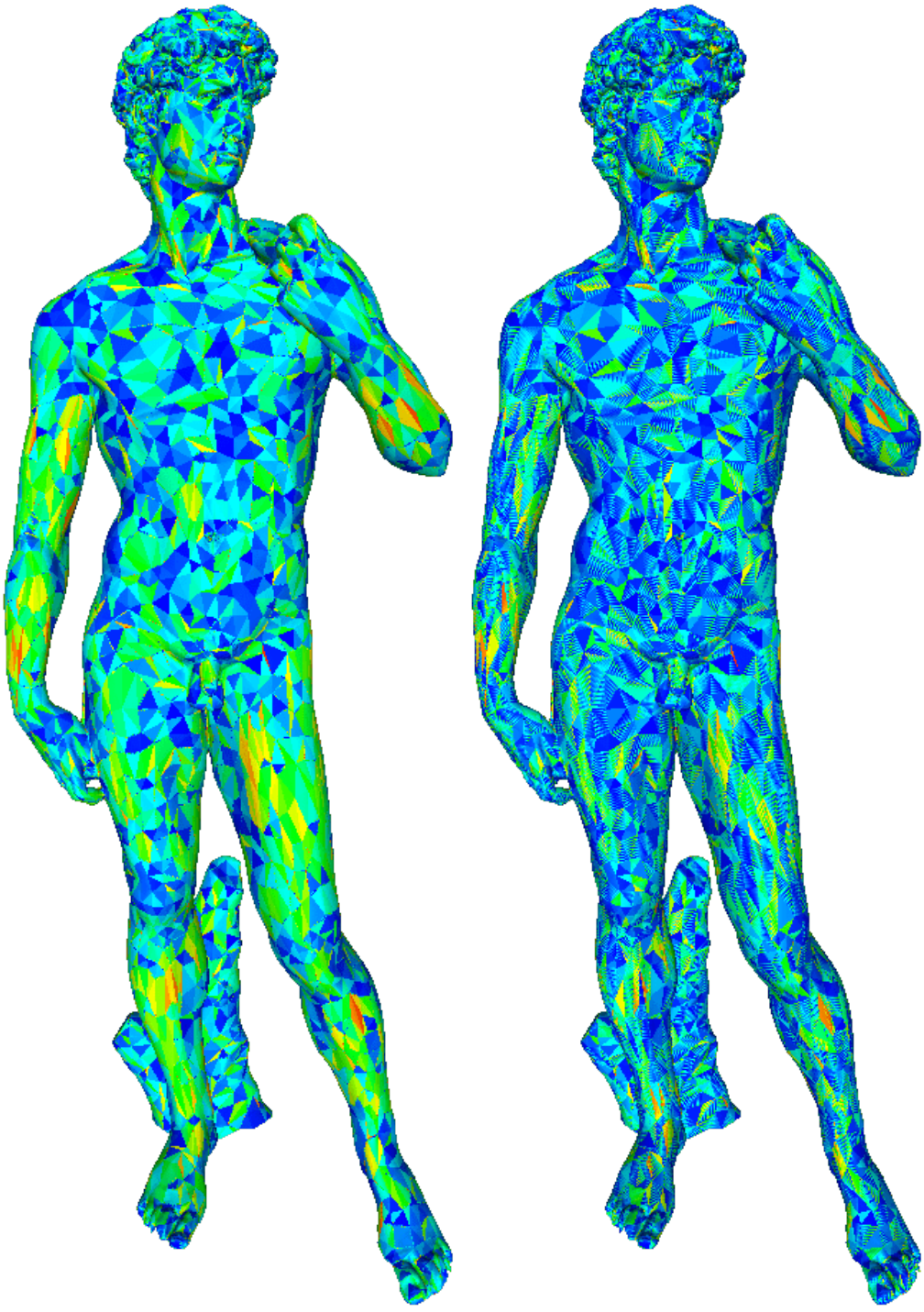
(b) Anisotropic Micro-Mesh

Figure 37: Dancing Faun — Graphical comparison of displaced subdivision schemes.

6.2.4 Michelangelo's David

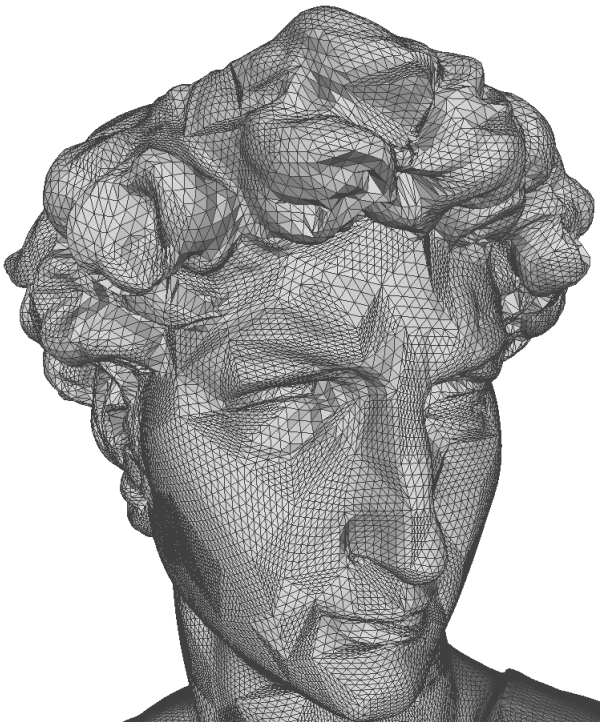
	Isotropic				**Anisotropic**			
	Face quality			CV (%)	Face quality			CV (%)
F	Mean	σ	σ^2		Mean	σ	σ^2	
1.0	0.38	0.17	0.03	66.66	0.5	0.17	0.03	37.11
1.1	0.38	0.17	0.03	67.13	0.5	0.17	0.03	36.74
1.2	0.38	0.17	0.03	66.94	0.5	0.17	0.03	36.82
1.3	0.38	0.17	0.03	66.65	0.5	0.17	0.03	37.53
1.4	0.38	0.17	0.03	65.67	0.5	0.17	0.03	37.85
1.5	0.38	0.17	0.03	64.76	0.5	0.17	0.03	37.74
1.6	0.38	0.17	0.03	63.33	0.5	0.17	0.03	37.80
1.7	0.39	0.17	0.03	62.76	0.5	0.17	0.03	37.62
1.8	0.39	0.17	0.03	62.57	0.5	0.17	0.03	37.41
1.9	0.39	0.17	0.03	62.62	0.5	0.17	0.03	37.43
2.0	0.39	0.17	0.03	62.63	0.5	0.17	0.03	37.53
2.1	0.39	0.17	0.03	62.38	0.5	0.17	0.03	37.15
2.2	0.39	0.17	0.03	62.36	0.5	0.17	0.03	37.13
2.3	0.39	0.17	0.03	62.54	0.5	0.17	0.03	36.72
2.4	0.39	0.17	0.03	62.38	0.5	0.17	0.03	36.63
2.5	0.38	0.17	0.03	63.28	0.5	0.17	0.03	36.45
2.6	0.38	0.17	0.03	64.14	0.5	0.17	0.03	36.33
2.7	0.38	0.17	0.03	64.42	0.5	0.17	0.03	36.20
2.8	0.38	0.17	0.03	65.00	0.5	0.17	0.03	36.19
2.9	0.38	0.17	0.03	65.41	0.5	0.17	0.03	36.29
3.0	0.38	0.17	0.03	66.11	0.5	0.17	0.03	36.43
3.1	0.38	0.17	0.03	66.25	0.5	0.17	0.03	36.51
3.2	0.38	0.17	0.03	66.60	0.5	0.17	0.03	36.19
3.3	0.38	0.17	0.03	66.80	0.5	0.17	0.03	35.83
3.4	0.38	0.17	0.03	66.70	0.5	0.17	0.03	35.70
3.5	0.38	0.17	0.03	67.03	0.5	0.17	0.03	35.46
3.6	0.38	0.17	0.03	67.25	0.5	0.17	0.03	35.53
3.7	0.38	0.17	0.03	67.64	0.5	0.17	0.03	35.70
3.8	0.38	0.17	0.03	67.81	0.5	0.17	0.03	35.88
3.9	0.38	0.17	0.03	67.71	0.5	0.17	0.03	36.15
4.0	0.38	0.17	0.03	67.35	0.5	0.17	0.03	35.92

Table 5: Michelangelo's David — Statistical comparison of subdivision patterns.

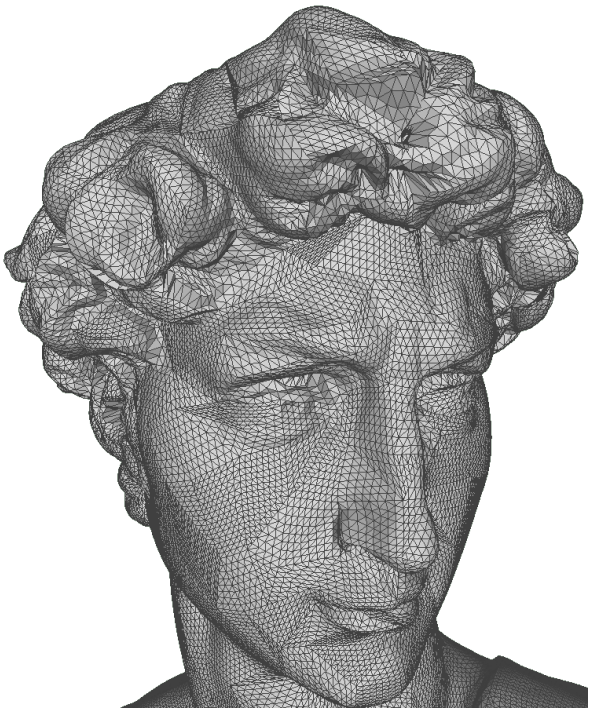


(a) Isotropic (b) Anisotropic

Figure 38: Michelangelo's David — Graphical comparison of face quality.



(a) Micro-Mesh



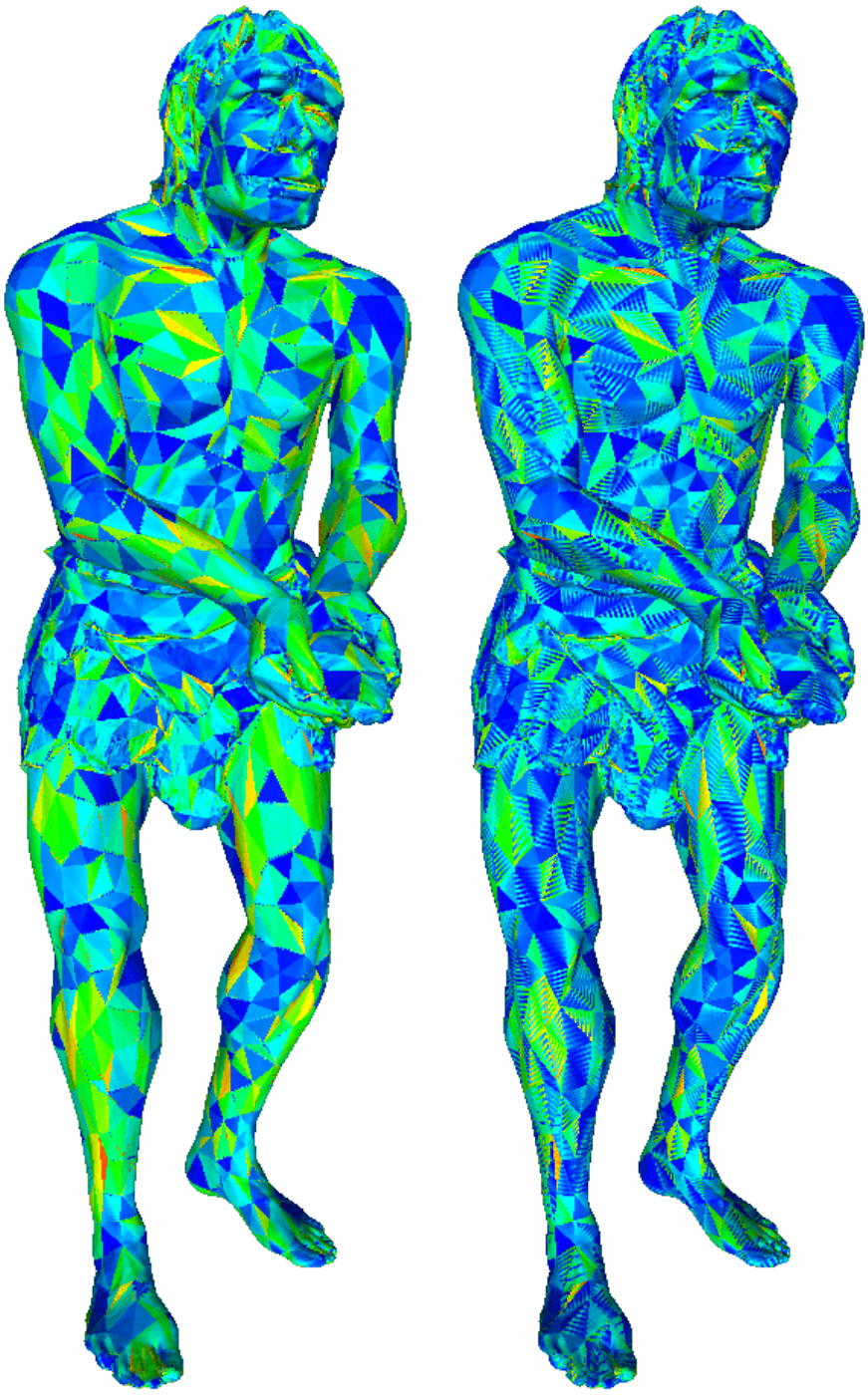
(b) Anisotropic Micro-Mesh

Figure 39: Michelangelo's David — Graphical comparison of displaced subdivision schemes.

6.2.5 Homo Heidelbergensis

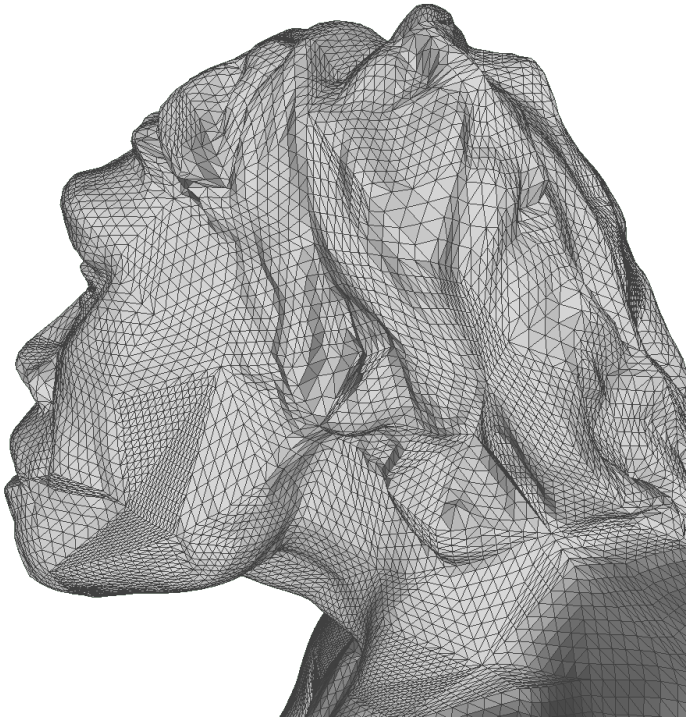
	Isotropic				**Anisotropic**			
	Face quality			CV (%)	Face quality			CV (%)
F	Mean	σ	σ^2		Mean	σ	σ^2	
1.0	0.39	0.17	0.03	62.97	0.49	0.17	0.03	38.03
1.1	0.39	0.17	0.03	64.30	0.42	0.17	0.03	38.36
1.2	0.39	0.17	0.03	65.13	0.49	0.17	0.03	38.15
1.3	0.39	0.17	0.03	65.14	0.43	0.17	0.03	37.82
1.4	0.39	0.17	0.03	65.30	0.49	0.17	0.03	37.91
1.5	0.39	0.17	0.03	65.69	0.41	0.17	0.03	37.59
1.6	0.39	0.17	0.03	65.60	0.49	0.17	0.03	37.18
1.7	0.39	0.17	0.03	66.18	0.40	0.17	0.03	37.79
1.8	0.39	0.17	0.03	65.38	0.49	0.17	0.03	37.86
1.9	0.39	0.17	0.03	65.28	0.40	0.17	0.03	37.65
2.0	0.39	0.17	0.03	65.36	0.49	0.17	0.03	37.39
2.1	0.39	0.17	0.03	64.72	0.39	0.17	0.03	36.87
2.2	0.39	0.17	0.03	64.00	0.49	0.17	0.03	37.10
2.3	0.39	0.17	0.03	63.28	0.40	0.17	0.03	36.69
2.4	0.39	0.17	0.03	63.38	0.49	0.17	0.03	36.57
2.5	0.39	0.17	0.03	62.72	0.49	0.17	0.03	36.72
2.6	0.39	0.17	0.03	62.74	0.39	0.17	0.03	36.52
2.7	0.40	0.17	0.03	62.24	0.49	0.17	0.03	35.95
2.8	0.40	0.17	0.03	62.19	0.39	0.18	0.03	36.11
2.9	0.39	0.17	0.03	62.31	0.49	0.17	0.03	36.03
3.0	0.40	0.17	0.03	62.11	0.39	0.18	0.03	35.92
3.1	0.39	0.17	0.03	62.60	0.49	0.17	0.03	36.08
3.2	0.39	0.17	0.03	62.56	0.39	0.17	0.03	35.68
3.3	0.39	0.17	0.03	63.22	0.49	0.17	0.03	36.19
3.4	0.39	0.17	0.03	63.21	0.39	0.18	0.03	36.23
3.5	0.39	0.17	0.03	63.30	0.49	0.17	0.03	36.40
3.6	0.39	0.17	0.03	63.46	0.39	0.17	0.03	36.84
3.7	0.39	0.17	0.03	63.64	0.49	0.17	0.03	36.49
3.8	0.39	0.17	0.03	63.92	0.39	0.17	0.03	36.90
3.9	0.39	0.17	0.03	63.71	0.49	0.17	0.03	36.97
4.0	0.39	0.17	0.03	63.75	0.40	0.17	0.03	37.26

Table 6: Homo Heidelbergensis — Statistical comparison of subdivision patterns.

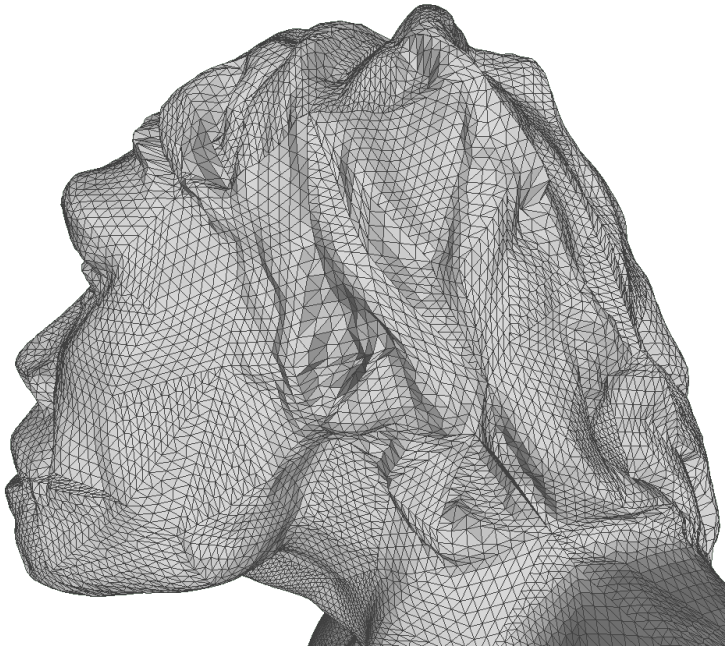


(a) Isotropic (b) Anisotropic

Figure 40: Homo Heidelbergensis — Graphical comparison of face quality.



(a) Micro-Mesh



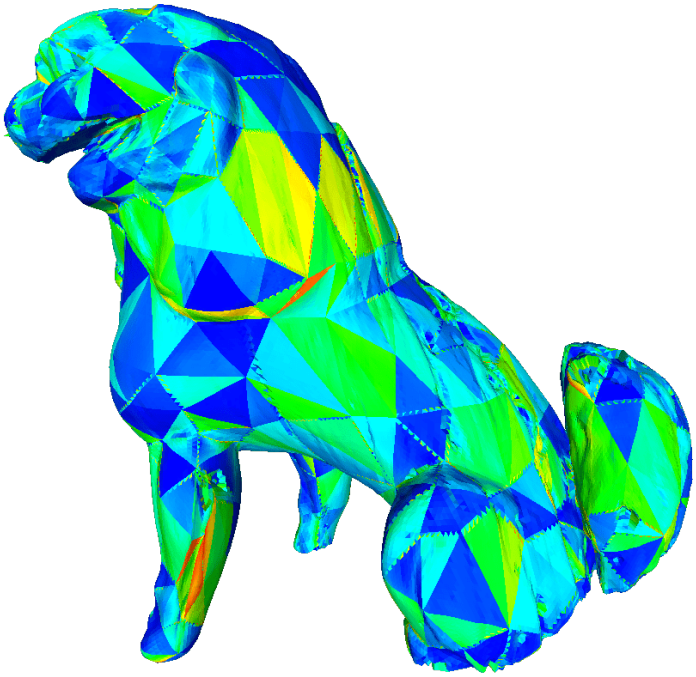
(b) Anisotropic Micro-Mesh

Figure 41: Homo Heidelbergensis — Graphical comparison of displaced subdivision schemes.

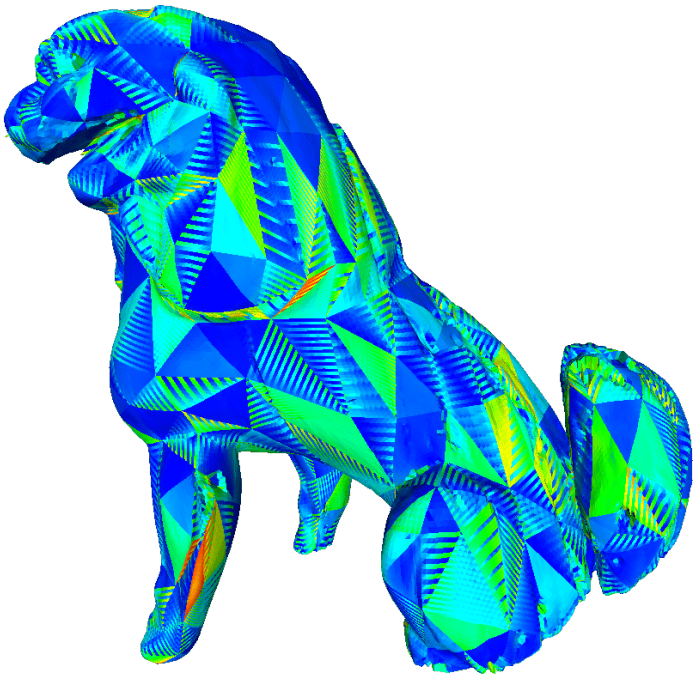
6.2.6 Koma Inu

	Isotropic				Anisotropic			
	Face quality			CV (%)	Face quality			CV (%)
F	Mean	σ	σ^2		Mean	σ	σ^2	
1.0	0.42	0.17	0.03	61.57	0.51	0.16	0.03	36.51
1.1	0.42	0.17	0.03	61.84	0.51	0.16	0.03	36.62
1.2	0.43	0.17	0.03	61.20	0.51	0.17	0.03	37.06
1.3	0.43	0.17	0.03	61.96	0.51	0.17	0.03	35.92
1.4	0.43	0.17	0.03	60.98	0.51	0.16	0.03	36.58
1.5	0.43	0.17	0.03	59.79	0.52	0.16	0.03	35.44
1.6	0.44	0.17	0.03	57.49	0.52	0.16	0.03	35.78
1.7	0.43	0.17	0.03	58.41	0.52	0.16	0.03	35.92
1.8	0.43	0.17	0.03	57.57	0.52	0.16	0.03	34.66
1.9	0.43	0.17	0.03	58.14	0.52	0.16	0.03	34.53
2.0	0.43	0.17	0.03	57.24	0.52	0.16	0.03	34.36
2.1	0.43	0.17	0.03	57.37	0.52	0.16	0.03	33.03
2.2	0.43	0.17	0.03	55.68	0.52	0.16	0.03	33.42
2.3	0.43	0.17	0.03	55.34	0.52	0.16	0.03	33.26
2.4	0.44	0.17	0.03	53.99	0.52	0.16	0.03	33.49
2.5	0.43	0.17	0.03	54.73	0.51	0.16	0.03	33.68
2.6	0.43	0.17	0.03	54.11	0.51	0.16	0.03	34.25
2.7	0.43	0.17	0.03	54.55	0.51	0.16	0.03	33.47
2.8	0.43	0.17	0.03	54.22	0.52	0.16	0.03	34.10
2.9	0.43	0.17	0.03	53.87	0.52	0.16	0.03	34.11
3.0	0.43	0.17	0.03	54.61	0.52	0.16	0.03	34.58
3.1	0.43	0.17	0.03	55.08	0.52	0.16	0.03	33.95
3.2	0.43	0.17	0.03	56.32	0.51	0.16	0.03	33.83
3.3	0.43	0.17	0.03	57.26	0.51	0.16	0.03	33.64
3.4	0.43	0.17	0.03	57.55	0.51	0.16	0.03	34.18
3.5	0.43	0.17	0.03	58.18	0.51	0.16	0.03	34.46
3.6	0.43	0.17	0.03	58.54	0.51	0.16	0.03	34.42
3.7	0.43	0.17	0.03	59.02	0.51	0.16	0.03	36.29
3.8	0.43	0.16	0.03	59.35	0.51	0.17	0.03	36.28
3.9	0.43	0.16	0.03	59.90	0.51	0.16	0.03	35.70
4.0	0.42	0.16	0.03	60.53	0.51	0.16	0.03	35.41

Table 7: Koma Inu — Statistical comparison of subdivision patterns.

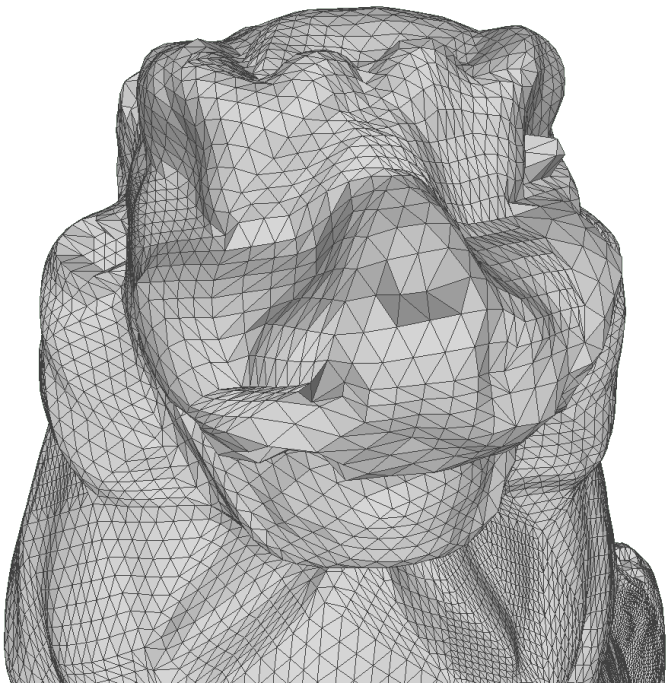


(a) Isotropic

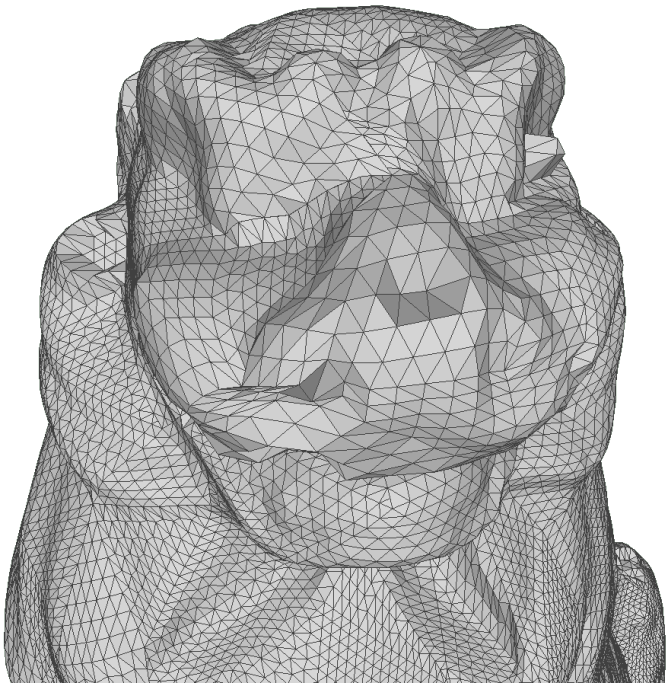


(b) Anisotropic

Figure 42: Koma Inu — Graphical comparison of face quality.



(a) Micro-Mesh



(b) Anisotropic Micro-Mesh

Figure 43: Koma Inu — Graphical comparison of displaced subdivision schemes.

6.3 Final results

To check for possible improvements of the new anisotropic partitioning scheme, the arithmetic mean value of the average face quality, and the coefficient of variation are considered for each model. By comparing these results, we can calculate the percentage performance of the new anisotropic scheme. It is important to note that we can speak of improvement when we observe increases in the average values of the quality of the faces and/or decreases in the average coefficient of variation. In the Table 8 such increases and decreases are indicated by the symbols "+" and "-".

	Isotropic		**Anisotropic**		**Results**	
	Average		Average		Mean (%)	CV (%)
Model	Mean	CV (%)	Mean	CV (%)		
Dragon	0.41	59.49	0.51	36.25	+24.39	-39.06
Borghese Ares	0.44	56.56	0.53	34.12	+20.45	-39.67
Dancing Faun	0.47	54.88	0.51	33.04	+8.51	-39.79
Michelangelo's David	0.38	65.19	0.50	36.64	+31.57	-43.79
Homo Heidelbergensis	0.39	63.85	0.45	36.94	+25.64	-42.14
Koma Inu	0.43	57.63	0.51	34.80	+18.60	-39.61

Table 8: Table of obtained improvements over average face quality and average coefficients of variation.

Analyzing the table, we can assert that all models had an increase in the average quality of faces between 8.51% and 31.57%, while the average coefficient of variation decreased by a value between 39.06% and 43.79%.

The results obtained can be considered satisfactory for the dataset under consideration. On average, utilizing the anisotropic scheme led to an average improvement in face quality of 21.53% and an average reduction in the coefficient of variation of 40.68%. The anisotropic scheme demonstrates positive performance across all models in the dataset and in both numerical statistics considered. It even surpasses the visual comparison of schemes, resulting in a more homogeneous subdivision with the anisotropic scheme.

Chapter 7

Conclusions

7.1 Conclusions

In conclusion, the study conducted has affirmed that the anisotropic schema represents a *viable* and promising alternative to the conventional μ-mesh schema. These affirmations are grounded in a comprehensive empirical analysis, as expounded in Chapter 6. The analysis has showcased not only the encouraging numerical results (Sec. 6.3) in comparisons but also visually satisfying improvements with respect to the μ-mesh subdivision scheme. In a retrospective evaluation, it is evident that the incorporation of **anisotropy control** is a *favorable* attribute within the μ-mesh schema.

It not only enhances the generation of more isotropic μ-faces, which are generally considered more aesthetically pleasing (symmetrical) but also plays a pivotal role in effectively mitigating the coefficient of variation when contrasted with the classical approach.

These achievements deserve commendation, which constituted a primary research constraint. It is essential to underscore that the current findings provide substantial evidence that the anisotropic schema holds considerable promise not only within graphics applications but also as a catalyst for pioneering new developments in alternative subdivision schemes.

As a result, this research not only reinforces the feasibility of anisotropic control within the context of μ-mesh subdivision but also underscores its potential to extend into a broader spectrum of applications, presenting an exciting avenue for further exploration and innovation within the field of geometry processing and computer graphics.

7.2 Future developments

As mentioned in the conclusions, this research on anisotropic μ-mesh schemes paves the way for new implementation schemes. The scheme can still be refined; a first optimization that could be done to increase the quality of the faces is to *subdivide* highly obtuse isosceles triangles (Sec. 4.4) in rectangular triangles.

As for the actual research, the development of a **hybrid** subdivision scheme can be considered. A scheme where it is possible to tailor the control of triangle anisotropy in areas where the triangles are thin and elongated (e.g., arms, legs, ...) and utilize the μ-mesh scheme on more isotropic macro-faces (e.g., face). Alternatively, further exploration could involve parallel implementation with a comprehensive analysis of subdivision scheme efficiency.

Bibliography

[Beck and d'Antona, 2014] Beck, J. and d'Antona, E. (2014). Scan The World. https://www.myminifactory.com/scantheworld.

[Cignoni et al., 2008] Cignoni, P., Callieri, M., Corsini, M., Dellepiane, M., Ganovelli, F., and Ranzuglia, G. (2008). MeshLab: an Open-Source Mesh Processing Tool. In Scarano, V., Chiara, R. D., and Erra, U., editors, *Eurographics Italian Chapter Conference*. The Eurographics Association. https://vcg.isti.cnr.it/Publications/2008/CCCDGR08/MeshLabEGIT.final.pdf.

[Cignoni et al., 1999] Cignoni, P., Montani, C., Rocchini, C., Scopigno, R., and Tarini, M. (1999). Preserving attribute values on simplified meshes by resampling detail textures. *The Visual Computer*, 15(10):519–539. https://doi.org/10.1007/s003710050197.

[Cook, 1984] Cook, R. L. (1984). Shade trees. *ACM SIGGRAPH Computer Graphics*, 18(3):223–231. https://doi.org/10.1145/964965.808602.

[FRKN, 2023] FRKN (2023). Orc Bust — cgtrader, 3d model marketplace for vr/ar and cg projects. https://www.cgtrader.com/free-3d-print-models/art/sculptures/orc-bust-d7a81f2f-6e47-45a7-88ce-108ea008be67.

[G-Truc Creation, 2005] G-Truc Creation (2005). OpenGL Mathematics — glm.g-truc.net. https://glm.g-truc.net/0.9.9/.

[Garland and Heckbert, 1997] Garland, M. and Heckbert, P. S. (1997). Surface simplification using quadric error metrics. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '97, page 209–216, USA. ACM Press/Addison-Wesley Publishing Co. https://doi.org/10.1145/258734.258849.

[Geometry Center, 1998] Geometry Center (1980-1998). Geometry center (university of minnesota), geomview — off file object. http://www.geomview.org/docs/html/OFF.html.

[Hoppe et al., 1993] Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., and Stuetzle, W. (1993). Mesh optimization. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '93, page 19–26, New York, NY, USA. Association for Computing Machinery.

[Jakob et al., 2015] Jakob, W., Tarini, M., Panozzo, D., and Sorkine-Hornung, O. (2015). Instant field-aligned meshes. *ACM Transactions on Graphics (Proceedings of SIGGRAPH ASIA)*, 34(6). https://dl.acm.org/doi/10.1145/2816795.2818078.

[Khan et al., 2022] Khan, D., Plopski, A., Fujimoto, Y., Kanbara, M., Jabeen, G., Zhang, Y. J., Zhang, X., and Kato, H. (2022). Surface remeshing: A systematic literature review of methods and research directions. *IEEE Transactions on Visualization and Computer Graphics*, 28(3):1680–1713. https://doi.org/10.1109/tvcg.2020.3016645.

[Khronos Group, 2017] Khronos Group (2017). OpenGL 4.6 Specification. https://www.khronos.org/registry/OpenGL/specs/gl/glspec46.core.pdf.

[Linus Torvalds, 2005] Linus Torvalds (2005). Git: Distributed version control system. https://git-scm.com/.

[Luebke, 2001] Luebke, D. (2001). A developer's survey of polygonal simplification algorithms. *IEEE Computer Graphics and Applications*, 21(1):24–35. https://doi.org/10.1109/38.920624.

[Maggiordomo et al., 2023] Maggiordomo, A., Moreton, H., and Tarini, M. (2023). Micro-mesh construction. *ACM Trans. Graph.*, 42(4). https://doi.org/10.1145/3592440.

[Manuel Pagliuca, 2023] Manuel Pagliuca (2023). Anisotropic micromesh. https://github.com/manuelpagliuca/anisotropic_micromesh.

[Martin, 2003] Martin, R. C. (2003). *Agile software development: principles, patterns, and practices*. Prentice Hall PTR.

[Möller and Trumbore, 1997] Möller, T. and Trumbore, B. (1997). Fast, minimum storage ray-triangle intersection. *Journal of Graphics Tools*, 2(1):21–28. https://doi.org/10.1080/10867651.1997.10487468.

[Muntoni and Cignoni, 2021] Muntoni, A. and Cignoni, P. (2021). PyMeshLab.

[Oliver Laric, 2012] Oliver Laric (2012). Three D Scans. https://threedscans.com/.

[OpenGL ARB, 2020] OpenGL ARB (2020). Opengl shading language specification. https://www.khronos.org/registry/OpenGL/specs/gl/GLSLangSpec.4.60.pdf.

[Qt Software, 1995] Qt Software (1995). Qt Framework. Available at https://www.qt.io.

[Shoemake, 1992] Shoemake, K. (1992). ARCBALL: A user interface for specifying three-dimensional orientation using a mouse. In *Proceedings of Graphics Interface '92*, GI '92, pages 151–156, Toronto, Ontario, Canada. Canadian Human-Computer Communications Society. http://graphicsinterface.org/wp-content/uploads/gi1992-18.pdf.

[Van Rossum and Drake, 2009] Van Rossum, G. and Drake, F. L. (2009). *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA. https://docs.python.org/3/reference/index.html.

[Wavefront Technologies, 1990] Wavefront Technologies (1990). Library of congress: Wavefront obj file format. https://www.loc.gov/preservation/digital/formats/fdd/fdd000507.shtml.