

UNIVERSITÀ DEGLI STUDI DI MILANO

Dipartimento di Informatica "Giovanni Degli Antoni"

Doctoral Program in Computer Science - Cycle XXXV

INF/01 INFORMATICA

Optimization of Photogrammetric 3D Assets

Doctoral Dissertation of
Andrea Maggiordomo

Supervisor
Prof. Marco Tarini

Co-Supervisor
Dr. Paolo Cignoni

Doctoral Program Coordinator
Prof. Roberto Sassi

Academic Year 2021/2022

Abstract

The photogrammetric 3D reconstruction pipeline has emerged in recent years as the prominent technology for a variety of use cases, from heritage documentation to the authoring of high-quality photo-realistic assets for interactive graphics applications. However, the reconstruction process is prone to introduce modeling artifacts, and models generated with such tools often require careful post-processing before being ready for use in downstream applications. In this thesis, we face the problem of optimizing 3D models from this particular data source under several quality metrics.

We begin by providing an analysis of their distinguishing features, focusing on the defects stemming from the automatic reconstruction process, and then we propose robust methods to optimize high-resolution 3D models under several objectives, addressing some of the identified recurring defects.

First, we introduce a method to reduce the fragmentation of photo-reconstructed texture atlases, producing a more compact UV-map encoding with significantly fewer seams while minimizing the resampling artifacts introduced when synthesizing the new texture images.

Next, we design a method to address texturing defects. Our technique combines recent advancements in image inpainting achieved by modern CNN-based models with the definition of a local texture-space inpainting domain that is semantically coherent, overcoming limitations of alternative texture inpainting strategies that operate in screen-space or global texture-space.

Finally, we propose a method to efficiently represent densely tessellated 3D data by adopting a recently introduced efficient representation of displacement-mapped surfaces that benefits from GPU hardware support and does not require explicit parametrization of the 3D surface. Our approach generates a compact representation of high-resolution textured models, augmented with a texture map that is optimized and tailored to the displaced surface.

Acknowledgements

First, I want to thank Marco Tarini and Paolo Cignoni for supervising my research. Their advice and guidance have been invaluable.

When I joined Marco at the University of Milan, I didn't know anyone there, but Alessandro Rizzi and Alice Plutino immediately made me feel welcome at the MIPS lab.

Besides Paolo, I also want to thank Roberto Scopigno and everyone I met at the Visual Computing Lab of CNR-ISTI in Pisa before starting my Ph.D. I learned so much working there. I am particularly grateful to Francesco Banterle and Federico Ponchio, who helped me with my work, and to Daniela Giorgi, who reviewed the first draft of my Ph.D. research proposal.

Thank you to Henry Moreton and Yury Uralsky at NVIDIA for allowing me to work on cutting-edge graphics technology and for all the knowledge and perspective they shared during our meetings.

Publishing papers is almost always a collective effort, and it would not have been possible for me without my co-authors: Marco Tarini, Paolo Cignoni, Federico Ponchio, Marco Farronato, and Gianluca Tartaglia.

The path that led me to submit this dissertation started with meeting Matteo Dellepiane. Matteo co-supervised my master's thesis and encouraged me to pursue research in Computer Graphics. We all miss him greatly.

Contents

1 Introduction **1**
1.1 Contributions . . . 2
1.2 Thesis overview . . . 3

2 Background **5**
2.1 Photogrammetry-based 3D reconstruction . . . 5
2.1.1 Acquisition . . . 5
2.1.2 Feature extraction and matching . . . 6
2.1.3 Sparse 3D reconstruction . . . 7
2.1.4 Multi-view Stereo reconstruction . . . 9
2.1.5 Mesh extraction . . . 9
2.1.6 Texturing . . . 10
2.2 Mesh parametrization . . . 11
2.2.1 Parametrization of triangle meshes . . . 12
2.2.2 Methods based on energy minimization . . . 12
2.2.3 Fixed boundary methods . . . 13
2.2.4 Conformal parametrizations . . . 14
2.2.5 Area preservation and isometric maps . . . 16
2.2.6 Optimization of non-convex distortion energies . . . 16
2.2.7 The local-global approach . . . 19
2.2.8 Surface cutting . . . 20
2.3 Image inpainting . . . 20
2.3.1 Artificial Neural Networks . . . 21
2.3.2 Deep and Convolutional Neural Networks . . . 22
2.3.3 CNN models for Image Inpainting . . . 22
2.3.4 Context Encoders . . . 23
2.3.5 U-Net with Partial Convolutions . . . 25
2.4 High-resolution mesh encoding with displacement maps . . . 27
2.4.1 Mesh simplification and optimization . . . 28
2.4.2 Sampling refinement attributes on a barycentric grid . . . 31

3 Photogrammetry mesh data analysis **33**
3.1 Introduction . . . 33
3.1.1 Background in 3D Photo-Reconstruction . . . 35
3.2 Related Work . . . 36
3.2.1 Quality Measurement Tools . . . 37
3.3 Dataset . . . 37
3.3.1 Dataset Curation . . . 38

3.4 Quality Measures for Textured 3D Meshes . . . 39
3.4.1 General Measures . . . 40
3.4.2 UV Map Measures . . . 40
3.4.3 Defects . . . 44
3.5 Measurements of RWTT Models: an Analysis . . . 44
3.5.1 Measures on Geometry . . . 44
3.5.2 Texture Maps and Parametrizations . . . 45
3.5.3 Defects . . . 48
3.6 Recomputing UV Maps . . . 49
3.7 The TexMetro Tool . . . 50
3.8 Conclusions . . . 51

4 Texture map optimization **53**
4.1 Introduction . . . 53
4.1.1 Context and motivations . . . 53
4.2 Overview . . . 56
4.3 Related Work . . . 58
4.4 Phases of the algorithms . . . 60
4.4.1 Enumeration of candidate merge operations . . . 61
4.4.2 Performing a merge operation . . . 61
4.4.3 Ranking of the candidate operations . . . 64
4.4.4 Updating candidate operations . . . 65
4.4.5 Atlas Repacking . . . 65
4.4.6 Extension to Multiple Texture Sheets . . . 67
4.5 Results . . . 68
4.5.1 Ablation study . . . 70
4.5.2 Comparison against computing an entirely new UV-map . . . 72
4.6 Conclusions . . . 73

5 Texture inpainting **79**
5.1 Introduction . . . 79
5.1.1 Our solution . . . 81
5.1.2 Contributions . . . 81
5.2 Related work . . . 82
5.2.1 Parametrizing photogrammetric models . . . 82
5.2.2 Image Inpainting . . . 83
5.2.3 Geometric deep learning and textured meshes . . . 83
5.2.4 Local parametrization . . . 84
5.3 Overview . . . 84
5.3.1 Texture inpainting operations . . . 84
5.3.2 Sources of texture defects . . . 85
5.4 Frameworks of use . . . 85
5.4.1 Fully automatic pipeline . . . 85
5.4.2 Interactive mode . . . 86
5.5 Auxiliary parametrization . . . 86
5.5.1 Objectives for the auxiliary parametrization . . . 86
5.5.2 Construction of auxiliary parametrization . . . 87
5.6 Texture inpainting network . . . 90
5.6.1 Texture re-resampling . . . 90

5.6.2 Training . . . 91
5.7 Automatic UV-map defect identification . . . 91
5.7.1 Per triangle labeling . . . 91
5.7.2 Clustering defective faces . . . 93
5.8 Results and experiments . . . 93
5.8.1 Examples of results . . . 93
5.8.2 Comparative evaluation . . . 94
5.8.3 Comparison with inpainting in global texture space . . . 97
5.8.4 Quantitative evaluation and ablation study . . . 98
5.8.5 Effectiveness of the learning process . . . 99
5.8.6 Comparison with real photographic data . . . 99
5.9 Conclusions . . . 100

6 Efficient representation of high-resolution geometry 103
6.1 Introduction . . . 103
6.1.1 Problem statement . . . 104
6.2 Related work . . . 105
6.3 Overview . . . 108
6.3.1 Method . . . 109
6.4 Determination of base displacement directions . . . 110
6.4.1 Finding the optimal visibility direction . . . 112
6.5 Base mesh generation by iterative coarsening . . . 114
6.5.1 Bounded anisotropy . . . 115
6.5.2 Collapse updates . . . 117
6.6 Proxy-mesh generation . . . 118
6.6.1 Topological and shape filtering . . . 119
6.6.2 Fast mesh decimation with random collapses . . . 119
6.7 Determination of base-mesh refinement level . . . 120
6.7.1 Uniform micro-face area . . . 120
6.7.2 Adaptive refinement . . . 122
6.7.3 Adjusting subdivision levels . . . 122
6.8 Determination of micro displacement values by raycasting . . . 123
6.8.1 Borders and tangential displacements . . . 123
6.9 Determination of base displacement lengths . . . 124
6.10 Barycentric ARAP . . . 125
6.10.1 Background: ARAP parametrization on standard 3D meshes . . . 126
6.10.2 Extension to micro-triangles . . . 127
6.11 Results . . . 129
6.11.1 Compression and representation accuracy . . . 130
6.11.2 Level of details . . . 130
6.11.3 Conversion of high-resolution models . . . 131
6.11.4 Textured models . . . 131
6.11.5 Comparison with standard mesh simplification . . . 135
6.11.6 Comparison with Bijective shells [JSZP20] . . . 136
6.12 Conclusions . . . 136

7 The inverse barycentric mapping problem 139
7.1 Introduction . . . 139
7.2 Problem definition . . . 140

7.3 Geometric derivation of the solution . . . 141
7.3.1 Deriving the polynomial coefficients . . . 142
7.3.2 Finding all valid solutions (if any). . . . 142
7.4 Example applications . . . 143
7.4.1 Texture baking via rendering . . . 143
7.4.2 Perfect reprojectability test . . . 145
7.5 Conclusions . . . 146
7.5.1 Problem variants . . . 147

8 Conclusions **149**
8.1 List of publications . . . 150
8.2 Future work . . . 151

Chapter 1

Introduction

The demand for 3D assets with high-resolution geometry and textures is ever-increasing, as rendering systems take advantage of the huge storage and computing capabilities of modern graphics hardware. There is a clear trend for modern interactive graphics applications towards "pixel-scale" geometric detail [KSW21], which can only be enabled by the extensive availability of high-resolution 3D assets. Manually authoring this kind of data is, however, a lengthy and expensive process that requires expert knowledge and masterful craft from artists. Moreover, the technology of state-of-the-art and industry-standard modeling tools oftentimes simply cannot cope with high-resolution geometry data due to computational bottlenecks that simply forbid interactive editing of such assets.

At the same time, in recent years the photogrammetry-based 3D reconstruction pipeline has emerged as the prominent choice for aiding 3D artists in authoring photo-realistic 3D content at a fraction of the cost that would otherwise be required by manual sculpting and texture painting. These tools are particularly popular because they do not require any particular expertise or dedicated hardware to produce realistic-looking high-resolution textured reconstructions of incredible visual quality. The convenient trade-off between required knowledge and modeling quality of photogrammetry-based 3D reconstruction tools resulted in the widespread adoption of this ground-breaking technology in a number of important fields. Photogrammetric reconstructions have found prominent applications in cultural heritage documentation and dissemination, with 3D acquisition campaigns carried out to allow online fruition and virtual examination of artifacts and sites of historical significance [Smi19, RSB*20]. Other application fields include structural and environmental monitoring, data acquisition for manufacturing, and virtual product presentation for electronic commerce.

While the advantages of this technology are obvious, automation and lack of manual intervention in the 3D asset generation process come at a cost. As we show in Chapter 3, the automatic reconstruction process is prone to introduce modeling artifacts, and 3D data generated by photogrammetry-based reconstruction tools often require careful post-processing to clean the data before being ready for use in downstream applications. Reconstructed meshes often exhibit geometric inconsistencies and texture defects due to incomplete coverage of the object or scene, inaccuracies and inconsistencies in the global registration of large photograph collections, difficulties in selecting the optimal subset of views that maximizes reconstruction accuracy, and so on. This poses significant challenges not only for applications but also for geometry processing algorithms that manipulate this kind of data, as their description is defined at the end of the photo-reconstruction process by the combination of the 3D geometric

description of the object *and* its texture map, i.e. a parametrization (usually discontinuous) combined with one or multiple texture images. Broadly speaking, computer graphics research deals with geometric modeling and texturing as two separate problems, and only a few works lie at the intersection of these two complementary aspects of production-ready 3D assets.

1.1 Contributions

In this thesis, we set out on a quest to reduce the gap in representation quality between artist-authored 3D assets and photogrammetry meshes, optimizing the latter for fruition and processing in downstream applications.

We begin our investigation with a comprehensive analysis of textured 3D models generated with a variety of photo-reconstruction tools (Chapter 3), collecting a significant number of representatives and quantitatively evaluating their quality according to several metrics. These include a new *invariant* metric we carefully design to assess the amount of fragmentation of the UV map, which is one of the main weaknesses potentially hindering the usability of these models. Our analysis highlights distinguishing features and recurrent defects of the mesh and texture data, backing the widely shared notion that models of this new class are still not fit for direct use in downstream applications and require manual cleaning and challenging processing steps. We have shared the corpus of textured 3D models used in our analysis as a public repository at `https://texturedmesh.isti.cnr.it`, along with the metadata we collected and an open source tool, TexMetro, to compute our set of quality measures on generic textured datasets.

Texture map fragmentation in photogrammetry mesh data is largely inherent to the texture reconstruction process itself, which generates texture samples by projecting color information from aligned views onto the reconstruction surface. To alleviate this defect, we design a method to improve the existing parametrization of a textured 3D model by *improving* the existing UV-map instead of replacing it entirely (Chapter 4). Our solution balances the reduction of the input map fragmentation with signal degradation due to resampling while preserving the original texture samples distribution. We evaluate the efficiency, robustness, and practical impact of our method on the dataset collected for our preliminary analysis, witnessing significant and consistent improvement of our targeted quality metrics.

Redundant discontinuities in the texture map are only one of many texturing defects that can be found in photo-reconstructed assets. Often, the quality of the reconstructed texture map is sub-optimal not because its geometric representation is flawed, but because the content itself is defective. Varying lighting conditions, occlusions from spurious objects, alignment errors during the scene reconstruction, and insufficient resolution of the available color data can result in noticeable artifacts that cannot always be characterized geometrically (Chapter 5). To tackle this, we design a method to perform texture inpainting directly on 3D models. Instead of solving the problem in screen space or in global texture space, we exploit recent advancements in image inpainting methods based on Deep Learning and Convolutional Neural Networks to perform inpainting in local texture space, using a support parametrization localized to the defect. Having a large corpus of textured models at our disposal enables us to train the neural network on local texture patches, noticeably improving the quality of the inpainting result. We implement our framework around two use cases: manual inpainting with an interactive tool, and automatic fixing of texture defects that can be detected using geometric and image-based criteria such as parametrization distortion or color discontinuity at seams.

Challenges when working with photo-reconstructed meshes are not only due to the presence of artifacts and general lack of *cleanliness*, but the high resolution of their geometric description, which also poses a cost on the distribution and rendering systems that enable their fruition. While the ability to cope with higher polygon counts has increased steadily in the past years, both in terms of storage and streaming capabilities as well as the raw computing power of graphics hardware, efficient and compact data representations remain a necessary requirement to enable the real-time fruition of highly detailed 3D objects. Currently, there are two main paradigms supported by real-time rendering systems to compress and efficiently render high-resolution meshes: multi-resolution encoding [GG*03, YSGM04, SM05, CGG*04, KSW21] and displacement maps [Coo84]. To compactly represent finely tessellated meshes, we develop a method to generate high-quality displacement maps from coarse geometric representations. We adopt a recently introduced compact mesh representation, the μ-Mesh, that does not require explicit parametrization of the coarse base geometry: displacement data is stored per face on a barycentric grid, and the high-resolution surface is obtained by regular 1-to-4 subdivision, with simple rules to control the tessellation level across the surface without introducing cracks. A key strength of this representation is that it benefits from hardware support in recent GPU architectures [NVI22b]. Our method is robust and scalable to extremely complex geometries, building on mesh decimation methods based on iterative edge collapses, and we demonstrate its practical use in converting real-world photogrammetry data for faster rendering at a reduced GPU memory footprint.

Accurately representing the original texture data is of fundamental importance to preserve the visual quality of the resulting displacement-mapped μ-Mesh, but changing the surface representation forces us to resample the original texture data. Texture mapping μ-Meshes can be done by defining texture coordinates for the base mesh. Since the displacement map is sampled on the implicit parametrization induced by the barycentric grid, we can use its information to define a distortion energy that is *displacement aware*, minimizing distortion with respect to the displaced surface instead of the coarse base mesh geometry (as standard practice suggests). This would not be possible with traditional displacement maps, which are sampled on a texture image and require the prior existence of an explicit parametrization function.

We conclude by presenting a solution to the *inverse barycentric displacement problem*, where given a displaced location we want to invert the map to obtain the original *undisplaced* location. The solution to this problem is non-trivial: we derive a closed-form solution for it and discuss some possible use cases.

1.2 Thesis overview

The core of this thesis consists of six chapters. Chapter 2 introduces background information that supports the rest of this thesis. It reviews the stages of the photogrammetric 3D reconstruction process, the theory of mesh parametrization, and the current state-of-the-art, convolutional neural network models for image inpainting. It also briefly discusses the foundations of mesh simplification and topology-optimization algorithms based on local operators, and sampling and indexing textures stored on a barycentric grid. The findings of our preliminary analysis of photo-reconstructed 3D data are reported in Chapter 3. Chapter 4 presents our method to reduce fragmentation and texture seams in existing texture atlases, while Chapter 5 describes our inpainting framework for correcting texturing defects. Our approach for representing high-resolution textured 3D data with hardware-supported displacement maps is detailed in Chapter 6. The solution to the inverse barycentric displacement problem is briefly presented

in Chapter 7, before concluding this thesis with a recapitulation of core contributions and discussing possible future work directions in Chapter 8.

The content of Chapters 3, 4 and 5 is also the subject of accepted publications. For a complete list of publications see Section 8.1.

Chapter 2

Background

In this chapter, we review some fundamental background knowledge and literature that serve as prerequisites and motivation for the works we present in this thesis. In Section 2.1 we review the steps of the photogrammetric 3D reconstruction pipeline, highlighting the peculiarities of this 3D data generation process and how they may affect the quality of the final reconstructed models, which will be the basis for our analysis in Chapter 3. In Section 2.2 we review mesh parametrization methods which are instrumental to enable surface texturing; this information is relevant for Chapters 4, 5 and 6 which all introduce parametrizations tailored to different task-specific needs. In Section 2.3 we review methods for image inpainting; we review both classic methods based on image optimization and similarity searches, as well as modern generative frameworks based on Convolutional Neural Network architectures and deep learning. Finally, in Section 2.4 we review the literature on mesh simplification, compression, and multi-resolution encoding, which is preparatory for the method we discuss in Chapter 6 to generate efficient representations of very high-resolution and finely-detailed mesh data.

2.1 Photogrammetry-based 3D reconstruction

The photogrammetric 3D reconstruction pipeline has been now implemented in several commercial and open-source software distributions [Cap22, 3Df22, Aut22, Pix19, Agi22, SF16, GGC*21]. There is a vast literature covering the methods upon which each of the pipeline stages is based; here we do not provide a complete review of them, since the methods developed in this thesis operate on the final data without interacting with the reconstruction process. Instead, we focus on describing the pipeline stages at a high level, to build an understanding of the reconstruction process as a whole and an intuition of what seeds the emergence of the distinctive features that are reported in our data analysis in Chapter 3.

2.1.1 Acquisition

The preliminary step of any photogrammetric reconstruction is the acquisition of pictorial data of the object to reproduce. One of the attractive features of photogrammetry-based reconstruction is that they do not require a controlled environment or dedicated hardware compared to, e.g., 3D scanning. Nevertheless, the proper acquisition of photographic data is essential to ensure a high-quality and detailed reconstruction. In practice, it is now very common to generate reconstruction from hundreds and thousands of photographs; complete

coverage of the object surface is necessary, as well as ensuring that the surface is viewed from nearly-orthogonal angles from multiple images. Uniform lighting also plays a crucial role, as it helps with the subsequent feature extraction and image matching phase, as well as the texturing stage where the photographic data is projected onto the reconstructed surface and stored as a high-resolution color signal in the forms of texture images.

Although not strictly necessary, markers may also be employed to speed up the initial rough alignment process.

2.1.2 Feature extraction and matching

Once the image set has been acquired, the actual reconstruction process can start. In the feature extraction stage, image descriptors are collected across the image collection and used to match image pairs that capture similar parts of the real-world scene. This is the first step in reconstructing the spatial relationships between the cameras within the 3D scene.

Local image features should be encoded as descriptors that are invariant to rotations, translations, and uniform scaling, as well as changes in illumination; this ensures that the matching and reconstruction process can aggregate data from photographs taken at different resolutions, at different distances from the object, by different camera sensors and under varying lighting conditions. Popular choices for feature descriptors are SIFT descriptors [Low04, TM*08]; more recently, learned descriptors [BHW11, DMR18] have also been proposed, where the training process is designed to minimize the matching error produced by a feature classification algorithm (e.g. Nearest Neighbor) or maximize the robustness of the extracted feature points.

Finding matching image pairs across a (possibly quite large) collection of samples can be a prohibitively expensive task if carried out in a naive way. To efficiently find matching image pairs, the image database is typically organized in data structures that support fast image retrieval queries based on the similarity of the spatial features. State-of-the-art reconstruction pipelines rely on the "Bag-of-Words" (BoW) [SZ03, CDF*04] model, representing the image contents as collections of local feature descriptors and inverted indices to quickly map between visual words and images; the Bag-of-Words terminology is borrowed from the Text Retrieval and Natural Language Processing fields, where it denotes models that treat sentences as unorganized collections of words, disregarding any semantic structure. The image descriptors are organized in a Visual Vocabulary [SZ03], possibly organized hierarchically in a Vocabulary Tree [NS06] to speed up queries on larger image sets.

Since the BoW model does not incorporate spatial information in the image feature descriptors, candidate image pairs must be spatially verified to ensure the matching is valid [PCI*07]. This is a critical step, as the spatial arrangement of the image features is critical to provide accurate matches. The spatial verification of an image pair seeks a linear transformation between the coordinates of the two feature sets, using e.g. RANSAC [FB81] to estimate correspondences across images. Image pairs that cannot be spatially verified, i.e. ones where the found linear transformation produces an inaccurate match (up to a specified error threshold) are rejected as invalid. This has been shown to significantly improve robustness both in general-purpose image retrieval queries, and in finding accurate matches to seed the sparse 3D reconstruction in Structure-from-Motion (SfM) frameworks [SF16].

To compensate for the lack of spatial information in the image descriptors of the BoW models, a spatial verification step is used to refine matches and reject image pairs whose common features do not align Images must be verified with a spatial verification test that seeks an affine

transformation from the features of one image to the features of another [PCI*07, SPS*17], again relying on statistical outlier detection methods like RANSAC to robustly determine valid correspondences.

More recently, state-of-the-art performance in feature matching has been achieved by deep-learning methods [SDMR20, SSW*21], without requiring the explicit geometric verification stage. Such methods operate on image pairs, therefore finding candidate image matches remains delegated to the image-retrieval algorithms outlined above.

Two-view reconstruction

In SfM, the geometric verification is not delegated to finding a simple affine matching between feature correspondences in image space. Rather, geometric verification amounts to solving a two-view 3D reconstruction problem using epipolar geometry to compute the Fundamental matrix that encodes the motion and projection information to map the points between the views [HZ03]. Solving the two-view reconstruction problem can be done with the Eight-Point algorithm [LH87, Har97] or the Five-Point algorithm [Nis04]; typically, denser correspondences are employed and the minimum number of points are obtained by filtering them again with RANSAC or similar algorithms [SF16]. This information serves not only as additional prior knowledge to improve the robustness of the geometric verification step but also to seed the construction of the camera network graph of the full-scene reconstruction.

2.1.3 Sparse 3D reconstruction

The next stage of the photogrammetric pipeline is to reconstruct the full 3D scene from the verified image pairs. This stage computes a globally coherent set of camera poses and view matrices together with a sparse set of triangulated 3D points corresponding to the features detected and used to match the image pairs. There are two common approaches to generate the sparse 3D scene reconstruction: incremental SfM seeds the reconstruction from two initial cameras and iteratively incorporates new information by selecting the next best view and refining the parameters of the reconstructed scene; global SfM [MMM13] instead aggregates the local reconstruction information to enforce global consistency in the system. Here, we review the incremental reconstruction process, as this is the method of choice for large-scale reconstructions [SF16, FLG14, GGC*21, ASS*09, Wu13, HSDF15].

Incremental SfM reconstruction

The output of the image matching and verification stage is a collection of pair-wise metric reconstructions from calibrated cameras; the calibration requires finding the intrinsic camera parameters, which can be recovered either from a database of known sensors or from the image metadata itself. If the camera parameters are unknown, they can be estimated using the Kruppa equations [FLM92].

Incremental SfM reconstructs the full 3D scene iteratively, starting from the reconstruction of a matched image pair. At each iteration, a novel view is selected from the set of unregistered images, and its pose is estimated from feature points that have already been triangulated by the reconstruction process. After registering the new image, image features that are shared with the images already registered in the scene can be used to triangulate new points in space: this allows to iterate the process, as new images are used to triangulate new points that in turn are used to register new images in the global reconstruction.

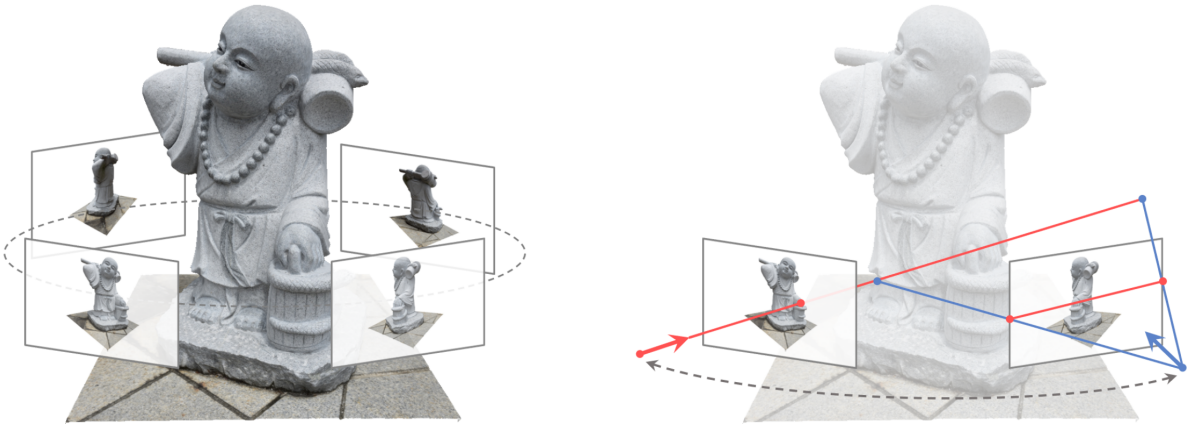


Figure 2.1: Epipolar geometry describes the geometric relationship between two views in the reconstructed scene. With projection matrices and the rigid motion between the two views known, finding dense correspondences reduces to a 1-dimensional search problem, as an incoming light ray in one view (red line) projects to a 2D line in the view plane of the other.

The choice of the initial image pair used to seed the incremental reconstruction process is critical. However, another source of errors is the fact that the estimation of novel camera poses is prone to drift due to approximation errors introduced by the camera parametrization and the inexact feature matching across images. This can result in the inability to generate a closed camera path (motion cycle) among the reconstructed cameras, or to register new views in the scene because of misalignments between new candidate views and the previously recovered poses.

To improve the robustness of the incremental SfM reconstruction, *Bundle Adjustment* [TMHF00] is used to jointly optimize a global parametrization of the scene. The goal of bundle adjustment is to minimize the global reprojection error of the triangulated points across the entire scene reconstruction, this is achieved by optimizing a parametrization of the scene reconstruction, i.e. to minimize the following energy function

$$\min \sum_i \sum_j w_{ij} \left(\mathcal{C}_i(\mathbf{x}_j) - \bar{\mathbf{x}}_{ij} \right) \tag{2.1}$$

where $\mathbf{x}_j$ denotes a triangulated 3D point, $\mathcal{C}_i$ the parametrization of camera i, $\bar{\mathbf{x}}_{ij}$ the image coordinates of the feature corresponding to x_j in the view produced by camera i, and w_{ij} a weighting coefficient for the re-projection error (modeling, e.g., confidence in the feature-to-point correspondence, with 0 denoting a non-visible point).

The optimization problem 2.1 is, in general, a non-linear least squares instance (the camera projection model can incorporate non-linear deformations such as lens distortion) that can be solved with Levenberg–Marquardt [Mor78]. While the number of parameters can be quite high, the structure of the system is sparse, as only correspondences across matched images are generally used to refine and stabilize the global registration. In practice, bundle adjustment is not solved at every iteration, and fast solvers exist that approximate linear time costs by requiring a constant number of iterations, using pre-conjugate gradient method solvers tailored to the problem structure [Wu13, WACS11, ASSS10].

2.1.4 Multi-view Stereo reconstruction

The sparse 3D reconstruction produced by the SfM stage is used to generate a dense sampling of the 3D scene before computing and extracting a mesh representation of the object.

Multi-view Stereo (MVS) algorithms [FH15, SCD*06] are designed to extract dense, possibly per-pixel 3D information from each of the calibrated camera views. This requires triangulating pixels (or image patches) by finding dense correspondences from neighboring views. Typically, dense image correspondences are found by computing a disparity map to estimate pixel shifts across images, which can be computationally expensive unless priors are used to filter image regions when searching for likely matches (e.g. assuming that the image sequence is consistent with the view motion). If the relative camera poses between two images are available, as is the case for the output of the SfM stage, then epipolar geometry describes the relationship between light rays in two views. With this relationship known, searching for correspondences of an image pixel in image pairs reduces to a 1-dimensional line search, as the path of the incoming light ray at a pixel for a given view corresponds to an epipolar line in the other (Figure 2.1). Dense matches are determined across pixel or patch pairs according to a measure of photometric consistency, such as Normalized Cross-Correlation (NCC) [FH15].

The output of the Multi-view stereo stage is an unstructured dense volumetric description of the reconstructed scene. Typically, MVS algorithms will produce a depth map per image, that must be filtered and fused in a global point cloud before the mesh extraction stage [GSC*07, ZDJF14, SZFP16, CVHC08, GLS15]. Recently, deep learning methods that compute single-image depth maps from aggregating multiple views have also been proposed [YLL*18, YLL*19]. These methods aggregate matching costs from multiple views over a voxel grid and use 3D convolutions to synthesize a depth map for a single reference image, rivaling, and surpassing, state-of-the-art traditional methods in terms of output quality and performance.

Another option is to directly reconstruct a globally coherent dense point cloud that is grown incrementally from the sparse reconstruction, using previously triangulated points as a guide for computing new stereo matches from nearby image regions [FP10, FCSS10, LQ05].

2.1.5 Mesh extraction

The dense sampling produced by the MVS stage is then used to extract a triangular mesh representing the surface of the reconstructed scene. There are two main approaches for extracting surfaces from unstructured point clouds: implicit surface methods and methods based on a Delaunay partitioning of the reconstruction volume.

Implicit surface methods model the reconstruction surface as the level-set of a Signed Distance Function sampled over a volumetric discretization of the scene [CL96], which can then be converted to a discrete triangle mesh using e.g. the marching cubes algorithm [LC87] or graph cuts [BK03]. The indicator function describing the surface determines the quality of the reconstruction: Poisson surface reconstruction method [KBH06, KH13] aligns the gradient of the indicator function to the approximate discretization given by the dense oriented point cloud; other methods model the problem as the minimization of a global reconstruction energy function that penalizes geometric inconsistencies [HK06, LB07].

Delaunay methods [ABK98, ACDL00, ACK01, DG03, LPK09] extract a triangle mesh from the Delaunay tetrahedralization of the reconstruction volume induced by the dense point cloud, iteratively removing redundant tetrahedra guided by the inside-outside labeling of the discrete volume elements [Boi84].

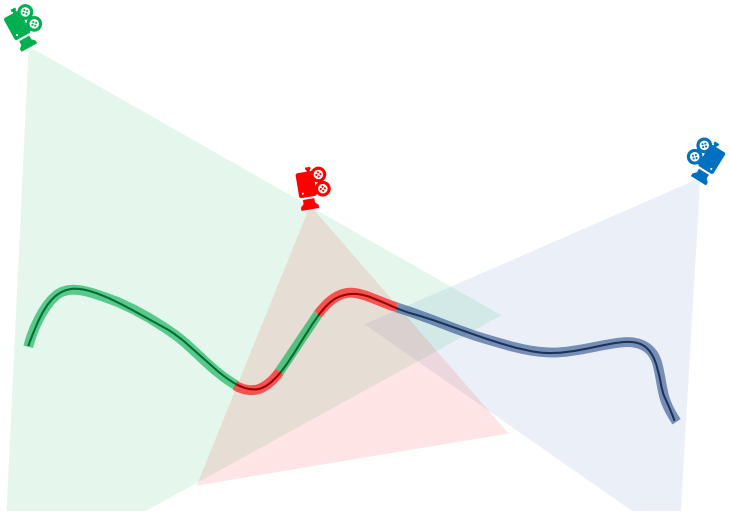


Figure 2.2: Texturing from aligned views is done by projection, and requires selecting one or multiple views according to metrics such as distance to the surface and viewing angle.

2.1.6 Texturing

The final stage of the photogrammetry reconstruction pipeline produces a texture image for the reconstructed mesh from the available photographic data. The process generally involves selecting one or multiple views to project the color data from, as depicted in Figure 2.2, and blending them to correct for the color discrepancies induced by varying lighting conditions that would otherwise negatively affect the perceived quality of the synthesized color texture image.

View selection methods can be classified by the number of views they use to generate the texture samples. Multiple-view methods blend color information from more than one view to achieve a consistent and smooth texture across the entire surface and reduce the visibility of color discontinuities at texture seams [CCCS08, Bau02]; such methods can suffer from excessive blurriness of the final texture and ghosting artifacts. [GAKC14] proposes a global optimization framework that mixes color data from all the available views to synthesize super-resolved texture maps, greatly increasing the sharpness and visual quality of the final texture. Single-view methods, instead, select a single view per face, solving a discrete labeling problem that optimizes an energy function designed to promote consistency and smoothness of the mapped color signal [GWO*10, LI07, VSJ07, SSS*08, WMG14]. Hybrid approaches that blend multiple views near chart borders have also been proposed [RCMS99, APK08]. A likely source of color artifacts is the inexact alignment of the available views, which some authors correct by relying on globally optimized per-view local warping [ZK14], or by synthesizing entirely novel views by combining photometrically-consistent image patches [BKR17].

A peculiarity of this texturing process is that the underlying parametrization is largely determined by the surface projection onto the available views, as this is the mechanism that guides the synthesis of texture samples. This is in stark contrast with the state-of-the-art surface parametrization methods presented in the next Section, that instead optimize for coarse layout and minimum mapping distortion.

2.2 Mesh parametrization

A continuous injective function ϕ that maps values from a domain $\Omega \subseteq \mathbb{R}^2$ to values in $\mathbb{R}^3$ is called a *parametrization* and its image S describes a 2D surface embedded in $\mathbb{R}^3$

$$S = Im\,\phi = \{\phi(u,v) : (u,v) \in \Omega\}. \tag{2.2}$$

The set Ω is referred to as *parameter space*, and surfaces that can be described in this manner are called *parametric surfaces*.

Since a parametrization is a function that maps a planar surface to a curved one, it is interesting to study how much distortion it introduces around a point – i.e., how lengths and angles change in the neighborhood of a parameter point $\mathbf{u} = (u,v)$ when it is mapped onto the surface point $\mathbf{p} = \phi(u,v) \in \mathbb{R}^3$.

Consider the explicit expression of ϕ as a vector-valued function:

$$\phi(u,v) = \begin{pmatrix} x_\phi(u,v) \\ y_\phi(u,v) \\ z_\phi(u,v) \end{pmatrix}. \tag{2.3}$$

Given a displacement $(\Delta u, \Delta v)$ the function ϕ can be linearly approximated with the first-order Taylor's expansion

$$\phi(u+\Delta u, v+\Delta v) \approx \phi(u,v) + \mathbf{J}_\phi(u,v) \begin{pmatrix} \Delta u \\ \Delta v \end{pmatrix} \tag{2.4}$$

where $\mathbf{J}_\phi(u,v)$ is the 3×2 Jacobian matrix of ϕ whose i-th row is the transposed gradient of the i-th component

$$\mathbf{J}_\phi(u,v) = \begin{pmatrix} \frac{\partial x_\phi}{\partial u} & \frac{\partial x_\phi}{\partial v} \\ \frac{\partial y_\phi}{\partial u} & \frac{\partial y_\phi}{\partial v} \\ \frac{\partial z_\phi}{\partial u} & \frac{\partial z_\phi}{\partial v} \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} \tag{2.5}$$

The linear approximation given by Taylor's formula represents the tangent plane at the point $\mathbf{u} = (u,v)$, spanned by the columns of $\mathbf{J}_\phi(\mathbf{u})$. The columns of the Jacobian matrix contain the component-wise partial derivatives of ϕ with respect to the u and v variables (denoted as ϕ_u and ϕ_v). These vectors provide a frame of reference for the tangent space at the point $\phi(\mathbf{u})$: in particular, we can use these vectors to move in the neighborhood of $\phi(\mathbf{u})$ when we take an infinitesimal step $d\mathbf{u} = (du, dv)^\top$ in parameter space

$$\phi(\mathbf{u} + d\mathbf{u}) \approx \phi(\mathbf{u}) + \mathbf{J}_\phi(\mathbf{u})\, d\mathbf{u} = \phi(\mathbf{u}) + du\phi_u + dv\phi_v. \tag{2.6}$$

When we use this linear approximation to map a circle in the neighborhood of $\mathbf{u}$ to the tangent space, it gets distorted into an ellipse according to the magnitude of the tangent vectors ϕ_u and ϕ_v and the angle between them. This distortion depends on the parametrization function ϕ and its Jacobian matrix $\mathbf{J}_\phi$ at the point; therefore, different parametrizations can exhibit different local distortions, and the goal of mesh parametrization methods is to compute parametrizations with very low distortion, as these entail desirable properties – e.g. to ensure texture samples are distributed uniformly over the mesh surface.

The distortion of a parametrization at a point can be characterized by the singular values of its Jacobian matrix. The singular values decomposition (SVD) of $\mathbf{J}_\phi$ is

$$\mathbf{J}_\phi = \mathbf{U} \begin{pmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \\ 0 & 0 \end{pmatrix} \mathbf{V}^\top \tag{2.7}$$

where σ_1 and σ_2 are the singular values of the Jacobian, $\mathbf{U}$ and $\mathbf{V}$ are a 3×3 and 2×2 unitary matrices and we omitted $\mathbf{u}$ to simplify the notation.

The SVD represents the local shape deformation as a combination of unitary (orthogonal) transformations and non-uniform scaling. Since $\mathbf{U}$ and $\mathbf{V}$ are rigid transformations, the local distortion of a parametrization is completely characterized only by the singular values of its Jacobian matrix at the point.

A parametrization is locally *conformal* if it preserves angles, in which case $\sigma_1 = \sigma_2$. Conformal maps scale quantities uniformly across both dimensions and are characterized by having orthogonal tangent vectors.

A parametrization is locally *equiareal* if it preserves areas, i.e. if $\sigma_1\sigma_2 = 1$. Equiareal maps are generally not sought since they allow arbitrary non-uniform scaling, thereby failing to preserve shapes.

A parametrization that is both conformal and equiareal is said to be *isometric* or an *isometry*. Isometric maps preserve lengths and introduce no deformation when the parameter domain is mapped to $\mathbb{R}^3$, but only exist for the restricted set of developable surfaces.

2.2.1 Parametrization of triangle meshes

Let $M = (\mathcal{V}, \mathcal{E}, \mathcal{F})$ be a triangle mesh defined by the set of its vertices $\mathcal{V}$, its edges $\mathcal{E}$, and its faces $\mathcal{F}$. Furthermore, let the set of vertices $\mathcal{V}$ be partitioned into the set of boundary vertices $\mathcal{V}_B$ and interior vertices $\mathcal{V}_I$, where a boundary vertex is a vertex incident on an edge that belongs to only one face. Given a vertex v_i, let also N_i denote the set of its neighbors and $\delta_i = |N_i|$ its degree.

Parametrizing a triangle mesh requires assigning to each vertex $v_i \in \mathcal{V}$ a position $\mathbf{u}_i$ in parameter space. This means that the sought function is the *inverse* function ϕ^{-1} instead of ϕ, as in general the 3D vertex coordinates are known while the parameter-space coordinates are not.

Many applications require an injective parametrization so that the mapping between the parameter space and the 3D surface is unambiguous. In the literature, it is common to distinguish between two different characterizations of injectivity, emerging from different properties of the flattened surface. A parametrization is *locally injective* if there are no triangle folds in parameter space, i.e. if each parameter triangle t is oriented coherently relative to its neighbors. A parametrization is *globally injective* if its boundary does not self-intersect.

Note that a parametrization only exists if the mesh has disk-like topology; if this is not the case, the surface must be cut first: we briefly review surface cutting methods in Section 2.2.8.

2.2.2 Methods based on energy minimization

The vast majority of mesh parametrization methods proposed in the literature compute maps that minimize an energy function obtained by integrating the local mapping distortion over the mesh surface.

Distortion energy functions are defined in terms of the singular values of the parametrization in such a way that their minima satisfy some desired property (for example a combination of conformality and area preservation). The energy E of the map ϕ is then the integral of the

local energy over the parameter domain:

$$E(\phi) = \int_{\Omega} E(\sigma_1(\mathbf{u}), \sigma_2(\mathbf{u}))d\mathbf{u}. \tag{2.8}$$

In the discrete case, the parametrization $\mathcal{P}$ of a mesh is piece-wise linear and the distortion is constant per face, reducing the integral to the area-weighted sum of the energies $E(f)$ of the linear functions that map the 3D configuration of a face $f \in \mathcal{F}$ to the corresponding parameter triangle t_f

$$E(\mathcal{P}) = \sum_{f \in \mathcal{F}} E(\mathcal{P}(f)) = \sum_{f \in \mathcal{F}} \mathcal{A}(f) E(\sigma_1(f), \sigma_2(f)), \tag{2.9}$$

where we use $\mathcal{A}$ to denote triangle areas.

Since distortion energies are invariant to rigid transformations, it is common practice to formulate the local (per face) distortion with respect to a 2D isometric projection T_f of the mesh face, rather than its 3D embedding. This yields a square 2×2 Jacobian matrix while leaving the singular values and the integrated energy unchanged.

Methods based on energy minimization all follow these common steps:

1. Define an energy function as outlined above.
2. Find a formulation of the energy that depends on the coordinates of the mesh vertices.
3. Minimize the energy by finding a suitable assignment of parameter coordinates to each vertex v.

2.2.3 Fixed boundary methods

Fixed boundary methods are some of the earliest methods proposed for computing mesh parametrizations. First, these methods fix the parameter space coordinates of boundary vertices to a convex shape, e.g. a square or a circle

$$\mathbf{u_k} = \bar{\mathbf{u}}_k, \qquad v_k \in \mathcal{V}_B. \tag{2.10}$$

Then, they compute the parametrization by minimizing the following quadratic energy [Flo97]

$$F(\{\mathbf{u}_1 \dots \mathbf{u}_n\}) = \sum_{(v_i, v_j) \in \mathcal{E}} w_{ij} \|\mathbf{u}_i - \mathbf{u}_j\|^2, \tag{2.11}$$

where $\{\mathbf{u}_1 \dots \mathbf{u}_n\}$ are the parameter space coordinates of the internal vertices and w_{ij}'s are suitable weight coefficients.

This approach has an intuitive physical analogy: it is equivalent to modeling the mesh edges as springs, with the weight coefficients w_{ij} corresponding to spring tensions (stiffness); fixing the boundary and minimizing this energy computes the vertex configuration corresponding to the steady state of the system.

The global minimum of F can be computed by setting the partial derivatives $\partial F / \partial \mathbf{u}_i$ to zero for all interior vertices $v_i \in \mathcal{V}_I$, i.e.

$$\frac{\partial F}{\partial \mathbf{u}_i} = 2 \sum_{v_j \in N_i} w_{ij}(\mathbf{u}_i - \mathbf{u_j}) = 0 \qquad v_i \in \mathcal{V}_I. \tag{2.12}$$

These equations can be arranged in a sparse linear system where each row i only has $N_i + 1$ non-zero coefficients for internal vertices and a single coefficient (the diagonal entry) for

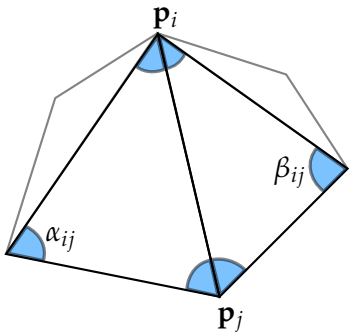


Figure 2.3: Notation used for determining the cotangent weights of a barycentric mapping.

boundary vertices. The system is solved independently for the u_i and v_i coordinates of the parameter space points $\mathbf{u}_i = (u_i, v_i)$.

If the boundary vertices are fixed, then a solution to the above system always exists. Moreover, if the weights w_{ij} are all positive and the boundary is mapped to a convex shape, no triangle flips can occur and the parametrization is guaranteed to be injective [Flo97].

The simplest choice is to use uniform edge weights ($w_{ij} = 1$) which corresponds to setting the position of each parameter space vertex to the barycenter of its neighbors. This approach was originally proposed by Tutte [Tut63, Tut60] to draw planar graphs without edge intersections, and is referred to as Tutte embedding.

Choosing edge weights that take into account the actual geometry and shape attributes of the 3D mesh leads to less distorted parametrizations. *Discrete Harmonic coordinates* (or *cotangent weights*)

$$w_{ij} = \cot \alpha_{ij} + \cot \beta_{ij}, \tag{2.13}$$

were first adopted in [PP93] to compute discrete minimal surface, and later to compute conformal parametrizations [EDD*95, DMA02] (refer to Figure 2.3 for the notation). These weights produce a harmonic map, i.e. a map that minimizes the Dirichlet energy of the mesh (the next section will expand on this). Note that the cotangent weights can be negative, so they no longer guarantee that the resulting parameterization is injective.

The biggest drawback of this family of methods is that fixing the boundary in advance to a convex shape typically introduces significant distortions. Even though the boundary can be mapped to any arbitrary shape, there is no way to know what would be a good outline for the parameterization *a priori*, although using a "virtual boundary" [LKL02], i.e. padding the mesh geometry with a strip of extra faces around the border can help mitigate the distortion. The methods described in the next sections do not require the boundary to be fixed in advance and compute it naturally as part of the energy minimization process.

2.2.4 Conformal parametrizations

A popular choice for the distortion energy function is the Dirichlet energy, which for a continuous parametrization is defined as

$$E_D(\phi) = \frac{1}{2} \int_\Omega \|\phi_u\|^2 + \|\phi_v\|^2 du dv. \tag{2.14}$$

The surface area of S provides a lower bound on the Dirichlet energy of a parametrization

$$\begin{aligned}\mathcal{A}(S) &= \int_\Omega \|\phi_u \times \phi_v\| du dv \\ &\leq \int_\Omega \|\phi_u\| \|\phi_v\| du dv \\ &\leq \frac{1}{2}\int_\Omega \|\phi_u\|^2 + \|\phi_v\|^2 du dv. \end{aligned} \tag{2.15}$$

It is easy to see that the first inequality becomes an equality if and only if the tangent vectors ϕ_u and ϕ_v are orthogonal (norm of the cross product), and the second inequality becomes an equality if and only if $\|\phi_u\|^2 = \|\phi_v\|^2$ by observing that $(\|\phi_u\| - \|\phi_v\|)^2 \geq 0$; therefore, the minimum of the energy is obtained when the map is conformal. Note that, using the SVD of $\mathbf{J}_\phi$ and the cyclic property of the trace operator, the integral argument can be rewritten in terms of the singular values of $\mathbf{J}_\phi$ as

$$\|\phi_u\|^2 + \|\phi_v\|^2 = \operatorname{tr}\left(\mathbf{J}_\phi^\top \mathbf{J}_\phi\right) = \sigma_1^2 + \sigma_2^2. \tag{2.16}$$

A formulation of the Dirichlet energy for triangular meshes is due to [PP93]. Denoting with α_i the angle at the vertex i of f in model space, the discrete Dirichlet energy can be expressed as

$$E_D(f) = \frac{1}{2\mathcal{A}(T_f)} \sum_{i=0}^{2} \cot \alpha_{i+2} \|\mathbf{u}_{i+1} - \mathbf{u}_i\|^2 \tag{2.17}$$

where the indices are given modulo 2. This formulation yields a quadratic function that can be rearranged to match the structure of Equation (2.11) with $w_{ij} = \frac{1}{2}(\cot \alpha_{ij} + \beta_{ij})$, leading to the Discrete Harmonic Coordinates of Equation (2.13). Instead of fixing the boundary vertices to a convex shape, [DMA02] introduces conditions for computing energy minimizers with *natural boundaries*. Boundary vertices are constrained according to the following conditions

$$\nabla E_D(f) - \nabla \mathcal{A}(t_f) = 0, \tag{2.18}$$

that are explained by observing that, by Equation (2.15), these are the conditions under which the gradient of the *conformal energy* [PP93] $E_C(f) = E_D(f) - \mathcal{A}(t_f)$ vanishes. With these additional constraints, the resulting linear system can be solved without requiring explicit fixing of the boundary; however, two vertices must still be fixed to ensure the matrix is invertible and avoid degenerate (zero) solutions.

Least Squares Conformal Maps [LPRM02] are computed by minimizing the distance of the tangent space vectors from being orthogonal (in the least squares sense)

$$E_C(\phi) = \frac{1}{2}\int_\Omega \|\phi_u^\perp - \phi_v\|^2 du dv, \tag{2.19}$$

where we use $^\perp$ to denote a 90 degrees counter-clockwise rotation. In the discrete case, the conformal energy is integrated over the mesh surface leading to a least squares system.

Despite different formulations, conformal parametrizations with natural boundaries and LSCM are equivalent and produce the same results [HLS07, MTAD08].

2.2.5 Area preservation and isometric maps

Conformal maps have the attractive characteristic of being computed as the solution of a simple sparse linear system, but their energy only depends on the angles of the surface triangles, and minimizing it makes no attempt at area preservation. As already mentioned this becomes an issue especially when the parameterization is used in the context of texture mappings because it means that the texture data is not sampled uniformly, leading to areas where the detail is very low. In practice, arbitrarily large area distortions often affect optimal conformal parametrizations, unless careful cuts are added to the surface to mitigate the metric distortion.

For the aforementioned reasons, the adoption of methods based on area-preserving and isometric energies has received considerable attention in the literature [HG00, DMK03, FLG15, SS15]. An isometric energy function that has become very popular in recent years and has been adopted in several works is the symmetric Dirichlet energy [SAPH04, SS15]

$$E_{SD}(f) = \left(1 + \sigma_1^{-2}\sigma_2^{-2}\right)\left(\sigma_1^2 + \sigma_2^2\right). \tag{2.20}$$

2.2.6 Optimization of non-convex distortion energies

The minimization of isometric energies is a challenging optimization problem: area-preserving energy formulations are non-convex, and iterative numerical optimization methods must be used to minimize them. Unfortunately, this implies that computing area-preserving parametrizations is much less efficient than quadratic methods; for this reason, early methods minimize nonlinear energies using a hierarchical mesh data structure [Hop96] and multi-grid solvers [HGC99, SSGH01, DMK03, SLMB05]).

Hierarchical solvers compute parametrizations with a multi-resolution framework where the mesh is processed at different levels of detail to speed up the overall process. In conjunction with this, these methods often adopt an iterative descent optimization scheme where a single vertex at a time is optimized. This approach yields fast iterations but is numerically very slow to converge. Some authors advocate the use of inexact block descent optimization and report faster convergence times as opposed to performing exact line search [FLG15].

Numerical optimization with iterative descent methods

More recently, several works have emerged that minimize non-convex distortion energies by applying iterative descent methods to the full set of parameter space coordinates [SKPSH13, APL14, SS15, KGL16, RPPSH17, CBSS17, SPSH*17, ZBK18].

Given a (discrete) parametrization $\mathcal{P}$, a descent direction $\mathcal{D} = \{\mathbf{d}_i\}_i$ is a collection of 2D vertex displacements (Figure 2.4) such that

$$E(\mathcal{P} + \lambda\mathcal{D}) < E(\mathcal{P}) \tag{2.21}$$

for a sufficiently small descent step $\lambda > 0$. Iterative descent methods update the current solution $\mathcal{P}^k$ by moving along the descent direction $\mathcal{D}^k$ computed at $\mathcal{P}^k$

$$\mathcal{P}^{k+1} = \mathcal{P}^k + \lambda_k\mathcal{D}^k, \tag{2.22}$$

where the iteration step size λ_k is found by minimizing the left-hand side of Equation 2.21 as a function of λ, subject to suitable bounds that ensure local monotonicity of the line search function. Iterations proceed until reaching a stationary point of the energy function.

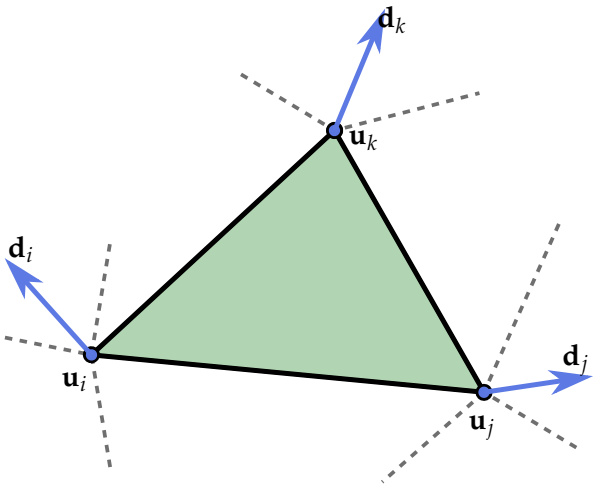


Figure 2.4: A descent direction for the parametrization problem is represented as a set of 2D displacement vectors per vertex.

The local steepest descent direction is given by the negative gradient

$$\mathcal{D} = -\nabla E(\mathcal{P}), \tag{2.23}$$

This direction is generally easy to compute, but only relies on first-order information from the energy function and is generally extremely slow to converge to a local minimum.

Faster numerical convergence can be achieved by incorporating curvature information in the computation of the descent direction $\mathcal{D}$. Consider the second-order Taylor's expansion of E, which yields a local quadratic model $E_2(\mathcal{D})$ of the energy function

$$E(\mathcal{P} + \lambda\mathcal{D}) \approx E(\mathcal{P}) + (\nabla E(\mathcal{P}))^\top \mathcal{D} + \frac{1}{2}\mathcal{D}^\top \left[\nabla^2 E(\mathcal{P})\right] \mathcal{D} = E_2(\mathcal{D}). \tag{2.24}$$

Newton's method computes the descent direction by minimizing $E_2(\mathcal{D})$

$$\nabla E_2(\mathcal{D}) = 0 \iff \mathcal{D} = -\left[\nabla^2 E(\mathcal{P})\right]^{-1} \nabla E(\mathcal{P}), \tag{2.25}$$

which is guaranteed to produce a descent direction if the Hessian matrix $\nabla^2 E(\mathcal{P})$ is positive definite [NW06]. Convergence is much faster, but the drawback is that Newton's method requires computing the Hessian of E, which is computationally expensive and has a lesser sparse structure than the matrices appearing in the quadratic conformal energy minimization problems presented in previous sections. To avoid explicit computation of the Hessian, previous literature [SS15] adopted quasi-Newton methods such as BFGS [Bro70, Fle70, Gol70, Sha70] and L-BFGS [Noc80] to approximate the Hessian matrix at the current point.

Different quadratic models yield different local approximations of the energy function (Figure 2.5). Instead of using Newton's method directly, current state-of-the-art methods design geometric quadratic proxies that accurately model the local energy curvature with a sparser matrix structure [KGL16, RPPSH17, CBSS17, SPSH*17, ZBK18], achieving fast numerical convergence at a moderate computational cost.

Note that second-order optimization methods require solving a quadratic minimization problem *per iteration*, compared to a single sparse linear system solution that is required to minimize quadratic conformal energies.

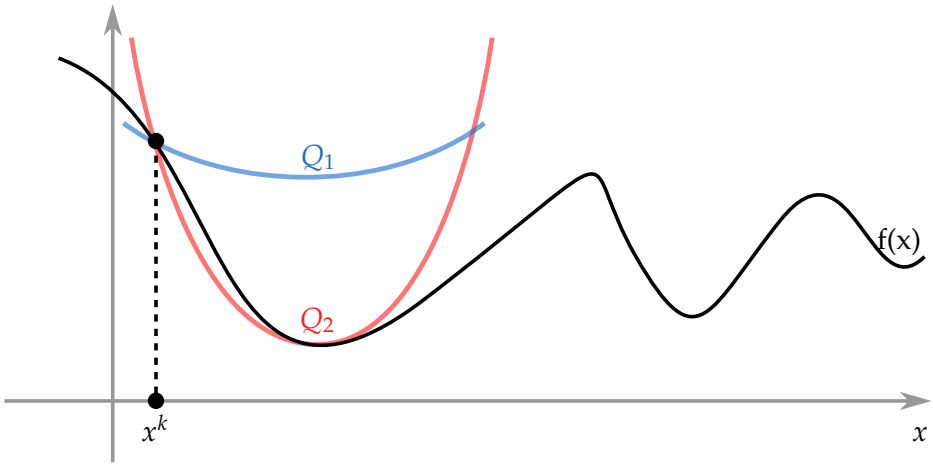


Figure 2.5: Different quadratic models Q of a 1D non-convex function f. In higher dimensions, more accurate local approximations of the energy function yield better descent directions and faster numerical convergence to local minima.

Line search with local injectivity preservation

Area-preserving and isometric energies asymptotically approach infinity as the parameter triangle shrinks. Therefore, the descent step computed during line search should be small enough to prevent triangle inversions.

Given a descent direction $\mathcal{D}$ and a mesh face f, let $\mathbf{d_0}$, $\mathbf{d_1}$ and $\mathbf{d_2}$ be the two-dimensional vectors along which displacing the parameter space coordinates $\mathbf{u_0}, \mathbf{u_1}, \mathbf{u_2}$ of f causes a decrease in the energy value. The largest descent step that is guaranteed to preserve the parameter triangle orientation and avoid flips is bounded by the smallest positive root of the following equation, quadratic in the step size λ [SS15]

$$\det \begin{pmatrix} (\mathbf{u_1} + \lambda \mathbf{d_1}) - (\mathbf{u_0} + \lambda \mathbf{d_0}) \\ (\mathbf{u_2} + \lambda \mathbf{d_2}) - (\mathbf{u_0} + \lambda \mathbf{d_0}) \end{pmatrix} = 0, \tag{2.26}$$

where the left-hand side is twice the signed area of the displaced triangle, as a function of λ.

If the initial parameter space configuration is overlap-free (e.g. using Tutte's embedding [Tut63] as initialization, or any fixed boundary barycentric mapping with positive weights as discussed in Section 2.2.3) then the iterative descent optimization with flip-preventing line search is guaranteed to preserve the local injectivity of the parametrization and not to produce inverted triangles. For highly tessellated and irregular meshes Tutte's initialization may be not injective due to numerical approximation errors, requiring either splitting the mesh into separate components or iterative refinements to mitigate the issue [SJZP19].

Global injectivity

While local injectivity can be achieved intrinsically using isometric, area-preserving flip-preventing energies starting from an injective initialization, avoiding global overlaps is more challenging and requires reformulating the energy function. Two effective approaches are introducing a barrier term that goes to infinity as the boundary approaches a self-intersection [SS15], or adopting a scaffolding structure that acts as padding for the parametrization boundary [ZMT05, JSP17].

2.2.7 The local-global approach

Another popular method to compute isometric parametrizations is to minimize the ARAP energy [LZX*08] exploiting the local-global geometric optimization framework pioneered in [SA07]. The local-global approach leverages the invariance of isometric and conformal maps under specific classes of linear transformations. Specifically, isometric mappings are invariant to orthogonal transformations while conformal mappings are invariant to similarity transformations.

Let $\mathcal{R} = \{\mathbf{R}_f\}_{f\in\mathcal{F}}$ be a set of linear transformations under which the targeted energy is invariant, then the idea is to minimize the following quantity

$$E_{LG}(\mathcal{F},\mathcal{R}) = \sum_{f\in\mathcal{F}} \mathcal{A}(T_f) E_{LG}(\mathcal{P}_f, \mathbf{R}_f) = \sum_{f\in\mathcal{F}} \mathcal{A}(T_f) \|\mathbf{J}_f - \mathbf{R}_f\|_F^2 \tag{2.27}$$

where $\|\cdot\|_F$ is the Frobenius norm and $\mathbf{J}_f$ is the Jacobian of the parametrization $\mathcal{P}_f$ of the mesh face f.

Since the Jacobian matrix $\mathbf{J}_f$ encodes the linear transformation of the 2D isometric projection T_f to the parameter triangle t_f, the expression above is a measure of the distance of that transformation to the matrix $\mathbf{R}_f$ assigned to the face. The matrix $\mathbf{R}_f$ was chosen from the class of matrices that map T_f without distortion, so the idea is to minimize the energy in order to "push" the parameterization of each face closer to the relative transformation. However, the set of transformations $\mathcal{R}$ can also be freely chosen to match the Jacobians, therefore the minimization involves two distinct sets of variables: the parameter space coordinates which determine the values of each Jacobian matrix $\mathbf{J}_f$ and the corresponding linear transformations $\mathbf{R}_f$.

Starting from an initial configuration of the vertices in parameter space (e.g. computed with any of the quadratic methods discussed earlier), the local-global approach minimizes the energy E_{LG} using an iterative scheme that involves a local step and a global step. At each iteration, the local step keeps the Jacobian matrices fixed and computes for each f a matrix $\mathbf{R}_f$ that best matches $\mathbf{J}_f$; then, a global step minimizes the energy by computing a new parameter space configuration while the local transformations are kept fixed.

The $\mathbf{R}_f$ matrix that is closest to the corresponding Jacobian is computed using the SVD decomposition of $\mathbf{J}_f$. Given a matrix $\mathbf{J}_f = \mathbf{U}\Sigma\mathbf{V}^\top$, the orthogonal matrix $\mathbf{P}$ that is closest to $\mathbf{J}_f$ with respect to the Frobenius norm is $\mathbf{P} = \mathbf{U}\mathbf{V}^\top$ [Hig89]. To preserve the orientation of parameter faces, however, $\mathbf{P}$ cannot be used directly as $\mathbf{R}_f$ if it is a reflection. This is easily detected by checking if the determinant of $\mathbf{U}\mathbf{V}^\top$ is negative, in which case the sign of the last column of $\mathbf{U}$ is changed to make sure that $\mathbf{R}_f$ has determinant equal to +1 [SHR16]. If similarity transformations are allowed, then the best fitting matrix is $\mathbf{R}_f = \mathbf{U}(\frac{\sigma_1+\sigma_2}{2})\mathbf{I}\mathbf{V}^\top$.

After the linear transformations $\mathcal{R}$ are computed, the global step minimizes the energy of Equation (2.27) by solving for the unknown parameter space coordinates. For fixed $\mathbf{R}_f$ matrices, the energy contribution of a face f can be formulated as

$$E_{LG}(\mathcal{P}_f, \mathbf{R}_f) = \frac{1}{2}\sum_{i=0}^{2} \cot\alpha_{i+2} \|(\mathbf{u}_{i+1} - \mathbf{u}_i) - \mathbf{R}_f(\mathbf{p}_{i+1} - \mathbf{p}_i)\|^2, \tag{2.28}$$

which is again quadratic in $\mathbf{u}$ and can be minimized by solving a sparse linear system with two constrained vertex positions to ensure the matrix is full rank.

If similarity transformations are allowed, the energy is equivalent to the conformal energies presented earlier: it is quadratic in both $\mathcal{P}$ and $\mathcal{R}$ and can be optimized directly with a unique sparse linear system solve; the authors called those parametrizations As-Similar-As-Possible (ASAP).

When only rigid transformations are allowed, the formulation for both sets of variables is no longer quadratic and the local-global method must be applied iteratively, alternating between the local and the global step. This energy and the resulting parametrization are called As-Rigid-As-Possible (ARAP).

2.2.8 Surface cutting

All the optimization strategies we discussed so far assume that the mesh for which we are computing the parametrization satisfies the topology requirements that are needed to admit a parametrization, namely that it is open and has genus zero, i.e. it is topologically equivalent to a disk.

This is not the case in general, and models with complex topology may require cutting them before computing the parametrization. Finding the optimal (shortest) cut that turns a mesh into a topological disk is an NP-Hard problem [EHP04], so cutting for genus reduction typically resorts to greedy schemes that quickly find a valid cut layout and then iteratively refine it to shorten the cut length [EHP04, DS95]. An alternative strategy is to grow surface charts with a greedy scheme while checking that the chart genus remains zero [LPRM02, SWG*03].

Another reason to introduce cuts in the surface is to reduce the metric distortion induced by the parametrization. A simple approach is to cut from distortion maxima to the mesh border [GGH02]; however, this approach requires repeatedly computing a parametrization for each added cut and does not scale to shapes that have lots of local energy maxima. Other methods identify high-curvature vertices and compute a minimum Steiner tree joining all the vertices with the border. Such vertices can be characterized by local curvature and distortion information [SH02] or by looking at the local metric scaling of an optimal global conformal map [KSS06, BCGB08, SSP08, SC18].

A series of works perform cutting and parametrization *jointly* solving a multi-objective optimization that aims to penalize both distortion and cut lengths. Some methods compute a parametrization triangle by triangle with a greedy strategy, welding adjacent vertices only if the resulting distortion is below a certain threshold [SCOGL02, PCCS11]. Others formulate a complex energy function with continuous and discrete variables (the cut locations), employing tailored minimization strategies to jointly optimize the cut layout and parametrization distortion [PTH*17, LKK*18a].

2.3 Image inpainting

Image inpainting is an important computer vision problem with applications ranging from image restoration to object removal. Given an input image x and a binary mask M denoting with 1 the valid image pixels and with 0 the invalid or *missing* image pixels, the inpainting task consists in assigning colors to missing pixels to produce a plausible and complete output image.

Traditionally, image inpainting methods relied on image statistics to complete the image with an iterative process that progressively shrunk by the filling process until it disappears. Early

methods focused on completing image structure alone [BSCB00], but suffer from excessive smoothness when inpainting large image regions. Other approaches combine the iterative structure completion with a patch selection heuristic, filling the missing image region with valid pixels according to a measure of patch similarity [CPT04, DCOY03]. Differently from such local methods, global methods [WSI07, BSFG09] use a multi-scale approach and fill the missing region by aggregating valid image patches using a similarity-weighted global average.

Recently, deep learning approaches based on Convolutional Neural Networks (CNNs) have brought significant advancements to this specific problem, taking advantage of the extraordinary capabilities of CNNs synthesizing realistic images. In the rest of this section, we briefly review artificial neural networks and convolutional neural network models, with a focus on models proposed for learning the image inpainting task. This serves as relevant background information for the texture inpainting framework presented in Chapter 5.

2.3.1 Artificial Neural Networks

Artificial Neural Networks are parametric models in which the computation is described by a directed acyclic graph (this is not true for the case of recurrent neural networks, which are not discussed in this report). Each node of the graph is an artificial neuron, with a set of input and output connections from and towards other neurons in the network. The function computed by each neuron is an affine transformation of its inputs followed by a non-linear *activation* function, whose value is then forwarded to the output neurons. The weights and bias terms of the affine combinations computed at each neuron are the model parameter, randomly initialized and adjusted during the training process.

Let f be a parametric model described by an artificial neural network, $\{x_i, y_i\}_i$ a sequence of (input, expected output) pairs, θ the model parameters and $\mathcal{L}_i$ an error function measuring the distance between two output values (the loss function). During the *forward pass*, the model computes a sequence of predictions $\{f(x_i, \theta)\}_i$. From these predictions, we can compute the gradient of the loss function w.r.t. the model parameters $\nabla_\theta \mathcal{L}$:

$$\nabla_\theta \mathcal{L} = \frac{1}{n} \sum_i \nabla_\theta \mathcal{L}_i \left(f(x_i, \theta), y_i \right) \tag{2.29}$$

(here a global loss is taken as the average of each sample loss); this step is referred to as the *backward pass* because the gradient of the model parameters is computed from a backward walk of the computation graph using the chain rule.

Then, we can use the computed gradient to update the weights of the model parameters to decrease the value of the loss function:

$$\theta' = \theta - \lambda \nabla_\theta \mathcal{L} \tag{2.30}$$

where λ is a small step size, as the negated gradient is only guaranteed to be a *local* descent direction. In the context of machine learning, the step size is usually referred to as the "learning rate". Since supervised learning generally requires large amounts of data to be effective, the model weights are generally updated from small subsets (*batches*) of the entire data sequence. This strategy is referred to as stochastic gradient descent.

Over-fitting occurs when the learned parameters perform well on the data used for training but not on arbitrary data. This can be verified during training by monitoring the model performance on a separate (typically smaller) set of validation data: if the loss function

computed on the validation data is increasing, the model is losing its generalization ability and overfitting the training data. This issue can be caused either by an inadequate model architecture, whose parameters cannot be effectively optimized by the training process to achieve the desired output quality, or by biases in the training data that fail to capture the correct distribution of inputs the model is applied to in practice.

2.3.2 Deep and Convolutional Neural Networks

Deep artificial neural networks are organized in a hierarchy of *layers*, i.e. nodes that are at the same level in the computation graphs. The advantage of a deep network is that its architecture can learn an effective *representation* for the task at hand, i.e. the model learns the desired function by also learning a transformation of the input data to a suitable feature space.

With high-dimensional data, such as images, *convolutional neural networks* (CNNs) have proved to be highly effective. In a CNN, the feature extraction is performed by convolution layers, where the learnable parameters are the convolution filter weights; convolution layers are typically followed by non-linear activation layers, similar to traditional artificial neural networks. A convolution layer typically performs a number of convolution operations that is equal to the output channels of the layer, each computed over the entire set of input feature channels.

Convolutional networks can be used to project high-dimensional inputs to a low-dimensional feature space, or vice-versa. Auto-encoders are encoder-decoder CNN architectures where the encoder module learns to project the input to a low-dimensional *latent* space, and the decoder module learns to decode the latent feature vector to reproduce the input. Auto-encoders can be used to learn compact representations and have several uses, e.g. data compression, image segmentation, image translation, and so on.

Generative models are trained to generate synthetic data by decoding latent space vectors. This is achieved by learning a decoding function that maps random latent vectors to output data that follows a desired distribution. Typically, generative models are trained using adversarial training with a generator network and a discriminator network jointly trained to compete with each other. The discriminator network is a binary classifier that learns to discern synthetic images from genuine ones, trying to maximize accuracy; the generator network is trained to trick the discriminator into classifying synthesized data as genuine. Generative models have shown remarkable results, especially in image synthesis, but, in general, it is difficult to condition the output to produce specific results. Training generative models can be challenging, as several "failure modes" can occur; as an example, if the discriminator becomes too effective at detecting synthesized images it will not provide sufficient information for the generator to improve, halting its progress.

2.3.3 CNN models for Image Inpainting

The Context Encoder [PKD*16] is a generative model designed to synthesize a missing patch from the surrounding context; although designed to work with input images of limited resolution, it is one of the first deep networks able to synthesize large missing regions. [ISSI17] extend the Context Encoder approach to arbitrarily sized inputs and employ both a local (patch-focused) and global discriminator to improve the inpainting quality. [LRS*18] propose to use partial convolution [LSW*18] layers to account for missing regions, and train their model with irregularly shaped masks. Recently, complex architectures with multiple modules have been proposed. [YLY*18] employ a coarse network to perform an initial rough reconstruction of the missing region followed by a refinement network whose output is then subject to both

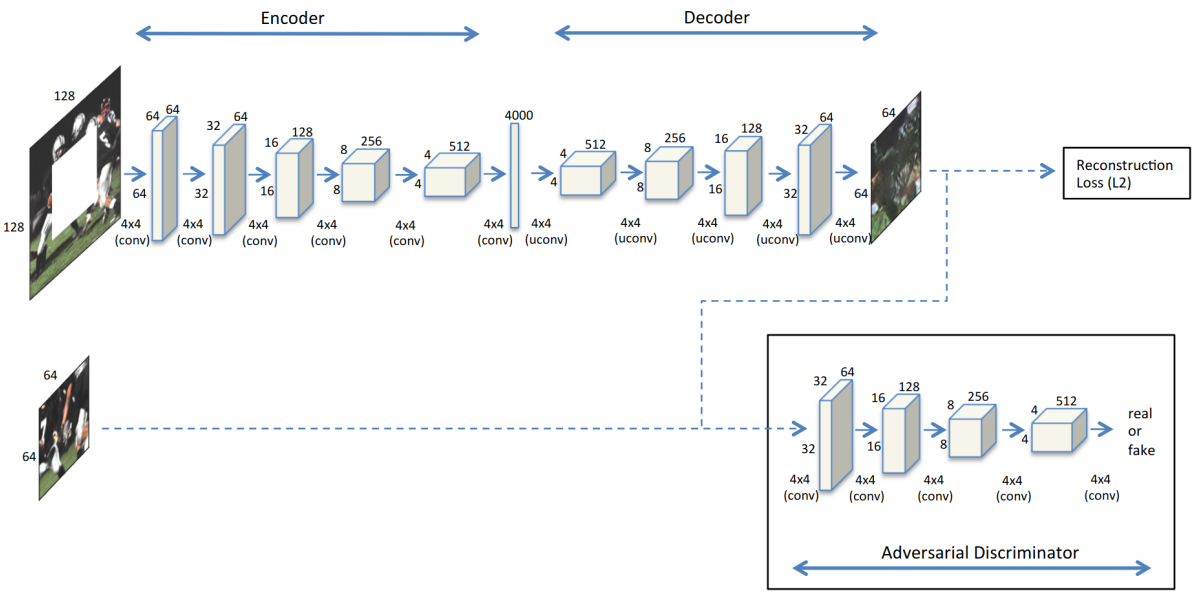


Figure 2.6: The Context Encoder training architecture. The generator network projects the masked input image to a 4000-dimensional latent vector, which is used to seed the generation of the missing image patch. The adversarial discriminator is jointly trained with the generator to classify original and synthesized image patches. Image from [PKD*16].

a local and global discriminator. They later evolve their architecture [YLY*19] adding gated convolutions and improving the refinement and discriminator networks. Similarly, [ZLY*20] also propose a two-stage approach with coarse-scale inpainting for low-frequency features and a refinement network to add high-frequency detail; however, they exploit an iterative inpainting scheme where the refinement network also outputs a confidence map, which is used to fill only high-confidence pixels and iterate again over low-confidence regions. They also train their model on low-resolution images, exploiting an upsampling network to restore the full resolution of high-resolution inputs.

2.3.4 Context Encoders

The Context Encoder [PKD*16] is an encoder-decoder architecture that takes as input a *context*, i.e. an image missing a center square region, and synthesizes an image patch to replace the missing region. This model is an extension of DCGAN [RMC16], from which it borrows the adversarial training scheme and the decoder and discriminator architecture. The model architecture (Figure 2.6) consists of a generator network F and a discriminator network D.

Input RGB images have a fixed resolution of 128×128 pixels and are normalized in the [-1,1] range, with missing pixels set to zero (medium intensity). The encoder projects the input image to a latent feature vector of dimension 4000, which is then decoded into a 64×64 image patch. The network is fully convolutional, with batch normalization [IS15] used both in the encoder and decoder module after each convolution or transposed convolution layer, followed by leaky ReLU activations in the encoder and ReLU activations in the decoder. The final output is ultimately mapped to the [-1, 1] range with a hyperbolic tangent activation layer.

The discriminator network is analogous to the encoder module, except the last convolution layer projects the image to a vector of dimension 1 and forwards it to a sigmoid activation layer to output a likelihood of the patch being real and *not* produced by the generator network.

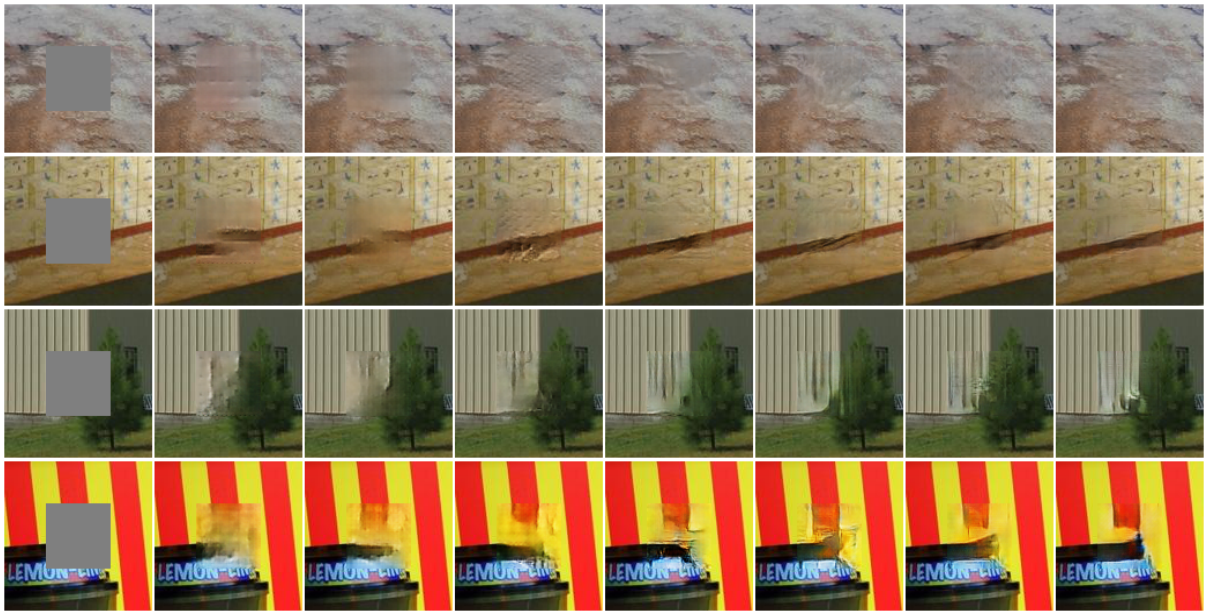


Figure 2.7: Context Encoder inpainting progress during training. Samples taken from the validation set.

Loss Function

The loss function used to train the context encoder combines a patch reconstruction loss $\mathcal{L}_{rec}$ and an adversarial loss term $\mathcal{L}_{adv}$. Let $\mathbf{M}$ be the binary mask inpainting mask with value 1 to denote valid context pixels and 0 the square hole in the middle. Let $\mathbf{X}$ be the image to reconstruct, $\mathbf{1}$ a matrix with all elements set to 1, and $\odot$ denote component-wise multiplication.

The reconstruction loss is defined as the squared error of the reconstructed patch pixels relative to the ground truth image:

$$\mathcal{L}_{rec}(\mathbf{X}) = \|(\mathbf{1} - \mathbf{M}) \odot (\mathbf{X} - F(\mathbf{M} \odot \mathbf{X}))\|_2^2. \tag{2.31}$$

According to the authors, training the context encoder with the reconstruction error alone does not allow the network to produce image patches with sharp detail, favoring rough outlines and blurry content. For this reason, they also employ an adversarial loss term

$$\mathcal{L}_{adv} = \max_D \mathbf{E}_{\mathbf{X} \in \mathcal{X}}[\log(D(\mathbf{X})) + \log(\mathbf{1} - D(F(\mathbf{M} \odot \mathbf{X})))]. \tag{2.32}$$

As is typical in this scenario, the generator network G and the discriminator network D are trained together using alternating optimization: at each iteration, the discriminator is trained with a batch of real patches and a batch of synthesized patches. After updating the discriminator, new scores for the synthesized patches are computed and used to minimize the combined loss

$$\mathcal{L} = 0.999\mathcal{L}_{rec} + 0.001\mathcal{L}_{adv} \tag{2.33}$$

with respect to the generator parameters.

To ensure a smooth transition from the border of the predicted patch to the surrounding context, the authors suggest using a larger weight for the reconstruction loss of pixels near the border of the reconstructed patch. Figure 2.7 shows some reconstructed images during the training process.

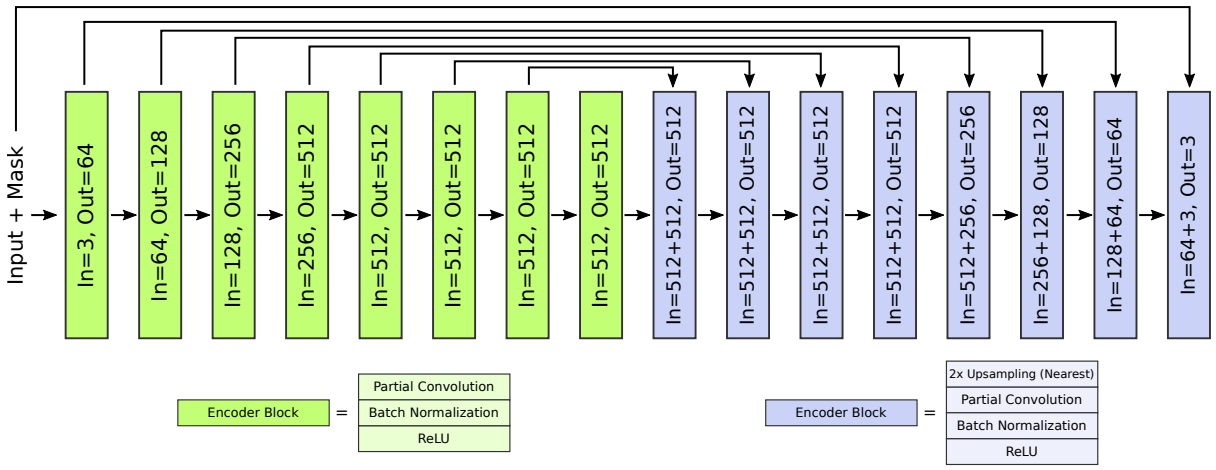


Figure 2.8: The U-Net architecture with the number of input and output features at each block.

2.3.5 U-Net with Partial Convolutions

A potential issue when using CNNs for image inpainting is that convolution layers in general are unable to discern between valid and masked pixels. Filling the image holes is a process that is learned during training and requires explicitly zeroing the missing pixels, but no additional semantic information is used by the network as a whole and the convolution layers in particular. If the mask location is deterministic as is the case for the Context Encoder model, this is not a significant issue as there is a clear statistical prior for what pixels are to be considered valid; however, this poses a challenge for free-form inpainting tasks. This problem is tackled in [LRS*18] by replacing standard convolution layers with *partial convolutions* [LSW*18].

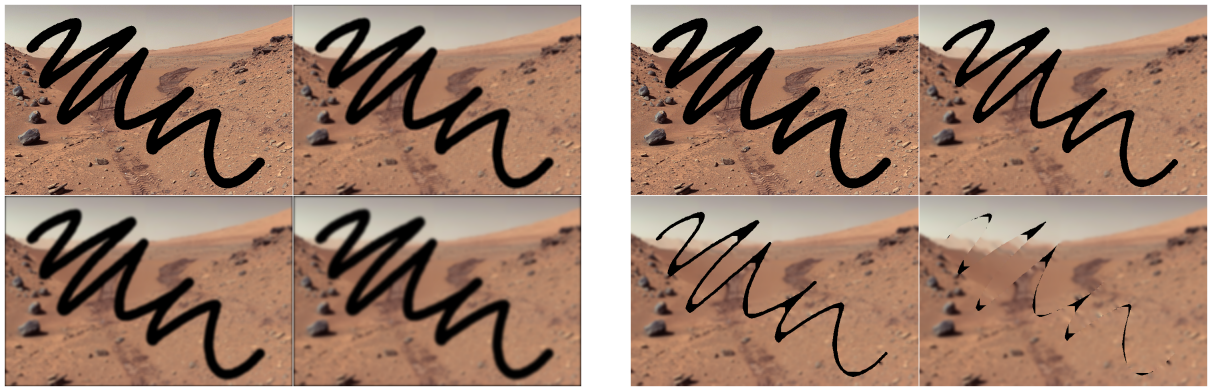
Their model is based on U-Net [RPB15], an encoder-decoder fully convolutional neural network with *skip connections*. A skip connection is used to bypass the hierarchical architecture of a convolutional network and connect the output of a layer not only to the input of the following layer, but also to the input of one or more other layers at deeper levels in the hierarchy. In this particular case, the network design is symmetrical: the model reconstructs a complete image, and skip links are used to connect the output of each encoder block to the input of the corresponding decoder block.

Each encoder block is composed of a (partial) convolution layer, a batch normalization layer, and a ReLU layer. Decoder blocks perform a 2-factor nearest neighbor upsampling, concatenate the upsampled feature map with the output feature map of the corresponding encoder block, and perform a (partial) convolution followed by batch normalization and leaky ReLU activation. Input images are assumed to be normalized in the [0, 1] range, therefore the model uses a sigmoid function as the final activation layer. The network architecture is outlined in Figure 2.8.

Partial Convolutions

The partial convolution layer performs two operations: it computes a re-weighted convolution, where each output value is scaled according to the number of valid pixels inside the corresponding convolution window, and produces an updated mask for the generated output feature map.

Let $\mathbf{w}_i$ be the convolution weights of a kernel window of size N, $\mathbf{x}_i$ the input feature values (i.e. the input convolution window) for the output location $\mathbf{x}$, and m_i the corresponding binary



(a) 5×5 Mean Filter – Standard Convolution

(b) 5×5 Mean Filter – Partial Convolution

Figure 2.9: Comparison between successive applications of a mean filter to an image with missing pixels, using standard and partial convolutions. The partial convolution propagates valid information inside the missing region, progressively shrinking the masked area; it also prevents bleeding of invalid signal into the image due to padding, whereas with standard convolutions the padding value (zero in this case) causes the border area to become darker as the number of applied convolutions increases.

mask values. The output $\mathbf{x}'$ of the partial convolution at x is defined as

$$\mathbf{x}' = \begin{cases} \sum_i \mathbf{w}_i m_i \mathbf{x}_i \frac{N}{\sum_i m_i} + b & \text{if } \sum_i m_i > 0, \\ 0 & \text{otherwise.} \end{cases} \tag{2.34}$$

The updated mask value at the corresponding location m' is defined as

$$m' = \begin{cases} 1 & \text{if } \sum_i m_i > 0, \\ 0 & \text{otherwise.} \end{cases} \tag{2.35}$$

As partial convolutions are repeatedly applied, the input binary mask progressively shrinks until no invalid pixels are left. As an example, Figure 2.9 shows the output of progressive applications of a 5×5 mean filter using the partial convolution scheme: border pixels are progressively smoothed inside the masked region, and the missing region area decreases in size until it completely disappears.

Loss function

The loss function the authors employ to train this model combines five different terms to capture per-pixel, style, and content reconstruction accuracy, as well as smoothness to avoid sharp transitions from the reconstructed missing region to the surrounding areas. Let $\mathbf{X}$ denote the ground truth image, $\mathbf{X}_{in} = \mathbf{M} \odot \mathbf{X}$ the input region with cleared missing pixels, $\mathbf{X}_{out}$ the network output and $\mathbf{X}_{comp} = \mathbf{M} \odot \mathbf{X}_{in} + (\mathbf{1} - \mathbf{M}) \odot x_{out}$ the reconstructed image with valid pixels recovered from the input image.

The first two terms $\mathcal{L}_{hole} = \frac{1}{N_\mathbf{X}} \|(\mathbf{1} - \mathbf{M}) \odot (\mathbf{X}_{out} - \mathbf{X})\|_1$ and $\mathcal{L}_{valid} = \frac{1}{N_\mathbf{X}} \|\mathbf{M} \odot (\mathbf{X}_{out} - \mathbf{X})\|_1$ are the mean absolute reconstruction errors restricted to the masked and unmasked pixels respectively (with $N_\mathbf{X}$ denoting the number of pixels in the ground truth image).

The third term is the perceptual loss $\mathcal{L}_{perceptual}$ introduced in [GEB16]:

$$\mathcal{L}_p = \sum_l \frac{\|\Phi_l(\mathbf{X}_{out}) - \Phi_l(\mathbf{X})\|_1}{N_{\Phi_l}} + \sum_l \frac{\|\Phi_l(\mathbf{X}_{comp}) - \Phi_l(\mathbf{X})\|_1}{N_{\Phi_l}}, \tag{2.36}$$

where $\Phi_l(\cdot)$ is the rectified feature map produced by the l-th layer of a VGG-16 [SZ15] network pre-trained on the Image-Net [RDS*15] dataset. The feature maps are extracted from the first three pooling layers. This loss is used to reproduce the content of the image independently from the particular style or texture. Intuitively, feature maps extracted from a neural network trained for image recognition become progressively more representative of the image *content* and independent from the image texture or style as the depth increases. Penalizing mismatches in the feature maps of the reconstructed and ground truth image guides the model toward restoring the correct image content in the missing regions.

The fourth and fifth terms model the style loss, again introduced in [GEB16]:

$$\mathcal{L}_{style_{out}} = \sum_l \frac{1}{C_l C_l} \left\| K_l \left([\Phi_l(\mathbf{X}_{out})]^\top \Phi_l(\mathbf{X}_{out}) - [\Phi_l(\mathbf{X})]^\top \Phi_l(\mathbf{X}) \right) \right\|_1, \tag{2.37}$$

$$\mathcal{L}_{style_{comp}} = \sum_l \frac{1}{C_l C_l} \left\| K_l \left([\Phi_l(\mathbf{X}_{comp})]^\top \Phi_l(\mathbf{X}_{comp}) - [\Phi_l(\mathbf{X})]^\top \Phi_l(\mathbf{X}) \right) \right\|_1, \tag{2.38}$$

where $K_l = C_l H_l W_l$ is a normalization factor applied to the gram matrix of the rectified feature maps $\Phi_l^T \Phi_l$, which represents their auto-correlation. This loss term is motivated by the idea that what we perceive as texture and style in an image can be captured with spatial statistics of the image features [PS00] and, in particular, can be modeled accurately by just considering the correlation of deep feature maps.

The final loss term is the total variation loss $\mathcal{L}_{tv}$ restricted to the 1-pixel dilation of the missing region M_+, and is used to ensure a smooth transition from the reconstructed missing region to the surrounding valid pixels:

$$\mathcal{L}_{tv} = \frac{\|\nabla((\mathbf{1} - \mathbf{M}_+) \odot \mathbf{X}_{comp})\|_1}{N_{\mathbf{X}_{comp}}} \tag{2.39}$$

The composite loss the authors propose is then

$$\mathcal{L} = \mathcal{L}_{valid} + 6\mathcal{L}_{hole} + 0.05\mathcal{L}_{perceptual} + 120(\mathcal{L}_{style_{out}} + \mathcal{L}_{style_{comp}}) + 0.1\mathcal{L}_{tv}. \tag{2.40}$$

Figure 2.10 shows inpainting results obtained with a U-Net augmented with partial convolutions and trained from scratch. In Chapter 5 we will use a similar inpainting architecture as a modular component for our proposed Texture inpainting framework.

2.4 High-resolution mesh encoding with displacement maps

In Chapter 6 we propose a method to represent high-resolution mesh data based on the refinement and displacement mapping of a coarse base mesh, exploiting recently introduced hardware support for this particular encoding [NVI22b]. This representation – the μ-Mesh [NVI22a] – stores displacement values on a regular barycentric grid, bypassing the need for an explicit parametrization of the base geometry to represent the tessellated high-resolution surface (Figure 2.11). The fundamental challenge in taking advantage of this



Input Inpainted Ground truth Input Inpainted Ground truth

Figure 2.10: U-Net with partial convolutions validation samples.

efficient representation is how to compute a coarse base mesh for the displacement map [CMO97, CH02, PPT*11, Cam17, JSZP20]. To produce a high-quality coarse base mesh optimized for accurate displacement mapping, we will adapt mesh simplification and mesh optimization methods based on local topological operations: in this section, we briefly introduce some background knowledge, as well as the indexing scheme used to sample barycentric location on base triangles.

2.4.1 Mesh simplification and optimization

A large body of literature developed mesh simplification and topology optimization methods based on local simplicial complex transformations [HDD*93]. Several operators have been defined, but the vocabulary of primitive operators that are sufficient to transform any triangulation in another (for a fixed genus number) can be restricted to edge flips, collapses, and splits (Figure 2.12).

Remeshing and mesh simplification algorithms based on local edge-collapse operations have been extensively studied [HDD*93, GH97, GZ05, GH98, Hop99, COM98, LT00, BK04, SAG03, HYB*17], due to their efficiency and reliability in producing good quality coarse low-resolution representations from high-resolution input meshes. For these reasons, we adapt these approaches to construct high-quality base meshes in Chapter 6.

When applying local transformations, care must be taken in ensuring the decimation does not change the local mesh topology, as this can result in artifacts such as the introduction of non-manifold edges and vertices: this can be guaranteed by checking after every collapse that the local mesh connectivity does not violate topological constraints known as *link conditions*

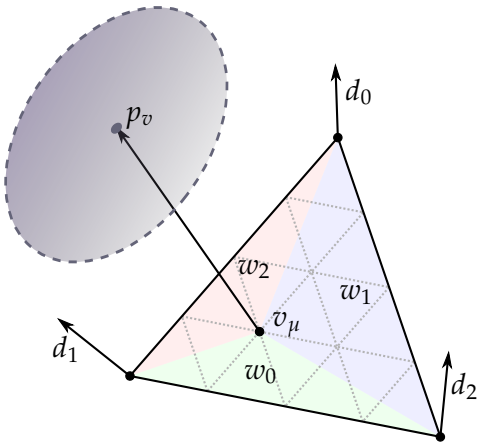
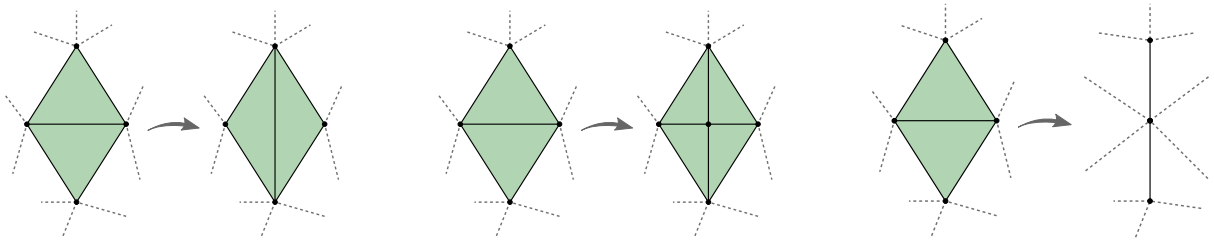


Figure 2.11: Displacement mapping with implicit barycentric parametrization. The geometry is refined with regular 1-to-4 subdivisions and displacement samples are stored on a barycentric grid. The displacement direction is obtained by linearly interpolating the vectors d_0, d_1, and d_2, defined at the base mesh vertices.



(a) Edge flip (b) Edge split (c) Edge collapse

Figure 2.12: Primitive local simplicial complex transformations.

[DEGN99].

An extremely important choice, particularly for mesh decimation algorithms, is how to determine the location of a vertex after a collapse operation. Despite being introduced a long time ago, the vertex placement using Quadric Error Metrics (QEM) [GH97] remains the state-of-the-art approach for designing fast and robust mesh simplification algorithms, thanks to its ability to compactly represent a local per-vertex shape descriptor of the high-resolution surface.

A quadratic function $Q\colon \mathbb{R}^3 \to \mathbb{R}$

$$Q(\mathbf{x}) = \mathbf{x}^T\mathbf{A}\mathbf{x} + \mathbf{x}^\top\mathbf{b} + c \tag{2.41}$$

that approximates the squared distance of a point $\mathbf{x}$ from the local surface is defined per vertex and aggregated at vertices during the mesh simplification. The optimal vertex location after an edge collapse is then chosen as the quadric minimizer by setting the gradient of Q to zero

$$\nabla Q(\mathbf{x}) = 0 \iff \mathbf{x} = -2\mathbf{A}^{-1}\mathbf{b}. \tag{2.42}$$

The matrix $\mathbf{A}$ is symmetric, so a quadratic function can be represented compactly with 10 scalars per vertex (6 scalars to store the lower or upper triangular part of $\mathbf{A}$, 3 scalars for $\mathbf{b}$ and 1 scalar for c).

To model the local surface distance metric, vertex quadrics are usually initialized as the area-weighted sums of the squared distances from the adjacent face planes. Given a plane

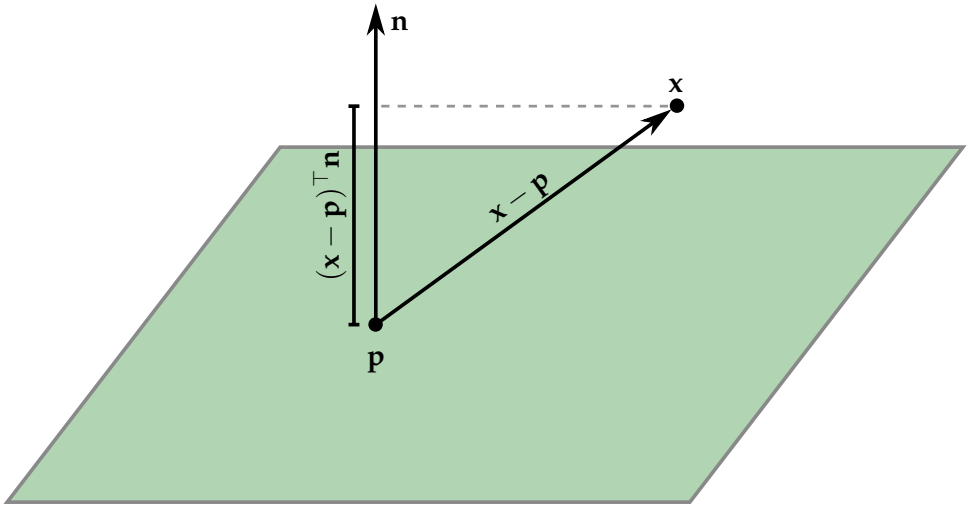


Figure 2.13: Geometric view of the point-plane quadric. The quadric measures the squared length of the projection of $\mathbf{x} - \mathbf{p}$ onto $\mathbf{n}$.

encoded as a point $\mathbf{p}$ and a normal vector $\mathbf{n}$, a plane quadric can be computed as the squared point-plane distance (see Figure 2.13 for a geometric derivation)

$$Q(\mathbf{x}) = \left((\mathbf{x}-\mathbf{p})^\top\mathbf{n}\right)^2 = \left((\mathbf{x}-\mathbf{p})^\top\mathbf{n}\right)\left((\mathbf{x}-\mathbf{p})^\top\mathbf{n}\right) \tag{2.43}$$

which expands to

$$Q(\mathbf{x}) = \mathbf{x}^T\left[\mathbf{n}\mathbf{n}^\top\right]\mathbf{x} - 2\mathbf{p}^\top\left[\mathbf{n}\mathbf{n}^\top\right]\mathbf{x} + \mathbf{p}^T\left[\mathbf{n}\mathbf{n}^\top\right]\mathbf{p}. \tag{2.44}$$

Using plane quadrics alone does not guarantee a unique minimum for the distance function if a vertex region is planar, i.e. if the surrounding faces all lie on the same plane. To mitigate this issue and ensure the quadric's matrix is invertible, it is common to also include a small quadratic term that measures the distance to the original vertex location. Given a point $p \in \mathbb{R}^3$, the point-point distance quadric can be trivially derived as the squared norm of the distance vector

$$Q(\mathbf{x}) = \|\mathbf{p}-\mathbf{x}\|^2 = (\mathbf{p}-\mathbf{x})^\top(\mathbf{p}-\mathbf{x}) \tag{2.45}$$

If the mesh surface has a border, face quadrics are unable to accurately model the surface distance metric, as they fail to penalize vertex displacements that move along tangent directions away from the border. This is usually countered by introducing special border quadrics at boundary edges that are orthogonal to the surface [GH97, GH98], accumulated with a large weight in the adjacent vertices to ensure they do not drift away from the original border during the simplification. Without explicitly penalizing border deviations, the decimation process can introduce serious artifacts at the borders [GZ05]. Instead of using orthogonal border quadrics, which are still unable to penalize displacements along the surface normals, we show here how to define a line quadric that can be used to accurately model the distance from the boundary polyline.

We assume a line is parametrized from an origin point $\mathbf{p}$ and a normalized direction vector $\mathbf{d}$ as $\mathbf{r}(t) = \mathbf{p} + t\mathbf{d}$ with $t \in \mathbb{R}$. To measure the point-line distance from an arbitrary point $\mathbf{x}$, we find the projection $\mathbf{x}_r$ of $\mathbf{x}$ onto the line

$$\mathbf{x}_r = \mathbf{p} + (\mathbf{x}-\mathbf{p})^\top\mathbf{d}\cdot\mathbf{d} \tag{2.46}$$

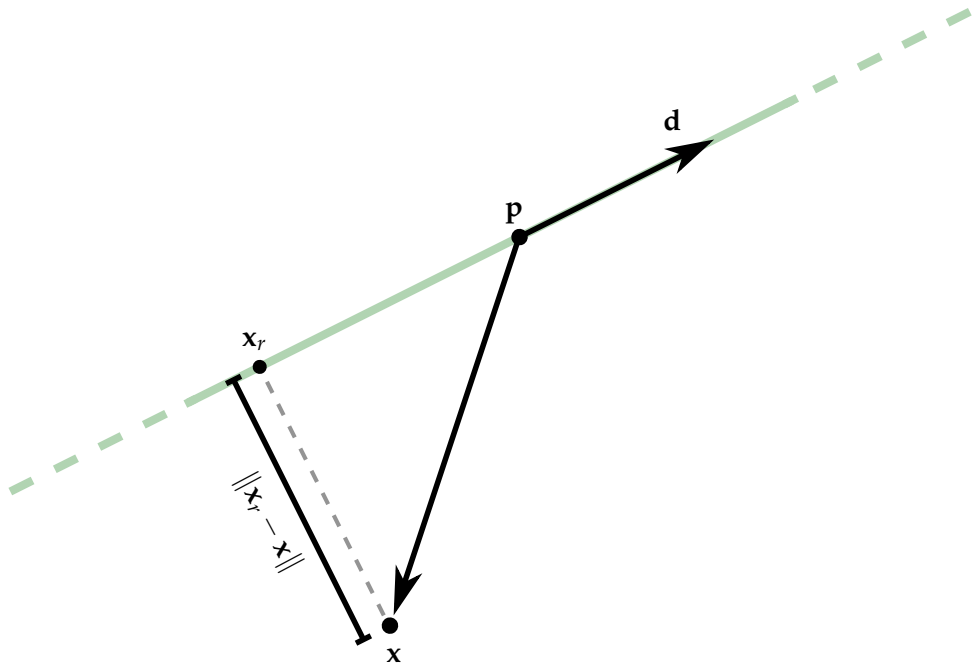


Figure 2.14: Geometric view of the point-line quadric. The quadric measures the squared length of the vector $\mathbf{x}_r - \mathbf{x}$.

and measure the squared distance of $\mathbf{x}$ from its projection

$$Q(\mathbf{x}) = \|\mathbf{x}_r - \mathbf{x}\|^2. \tag{2.47}$$

We can rewrite Equation 2.47 as a dot-product, which reduces to the following quadratic function

$$Q(\mathbf{x}) = \mathbf{x}^T \left(\mathbf{I} - 2\mathbf{d}\mathbf{d}^\top + \mathbf{d}\mathbf{d}^\top\mathbf{d}\mathbf{d}^\top\right)\mathbf{x} - \tag{2.48}$$

$$2\mathbf{p}^\top \left(\mathbf{I} - 2\mathbf{d}\mathbf{d}^\top + \mathbf{d}\mathbf{d}^\top\mathbf{d}\mathbf{d}^\top\right)\mathbf{x} + \tag{2.49}$$

$$\mathbf{p}^\top \left(\mathbf{I} - 2\mathbf{d}\mathbf{d}^\top + \mathbf{d}\mathbf{d}^\top\mathbf{d}\mathbf{d}^\top\right)\mathbf{p} \tag{2.50}$$

where $\mathbf{I}$ is the identity matrix. The derivation of the above quadric is mechanical and we do not report it here.

When an edge is collapsed as part of the iterative mesh simplification process, the quadrics of the two edge endpoints are aggregated at the resulting vertices, thereby incorporating surface information from a larger region.

2.4.2 Sampling refinement attributes on a barycentric grid

We discuss here a storage and indexing scheme for storing attributes on barycentric grid locations produced by regular 1-to-4 triangle subdivisions, taking advantage of the implicit parametrization induced by the barycentric coordinates. Assume a triangle is mapped to a lower-triangular matrix storing the attributes: each base face is laid out on top of a barycentric grid, which is represented as a lower-triangular matrix. Then, we can lay out grid samples (e.g. scalar displacements) by row and assume a conventional winding order where the first base vertex is at grid coordinates $(0, 0)$, the second vertex is at coordinates $(n-1, 0)$ and the third vertex is at coordinates $(n-1, n-1)$, where $n = 2^{l_b} + 1$ is the dimension of the barycentric

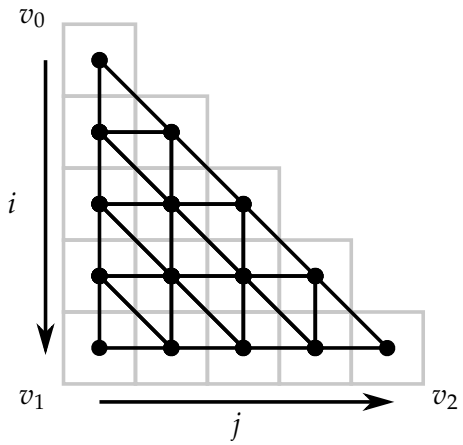


Figure 2.15: A simple indexing scheme for storing samples on a barycentric grid. Data samples are stored in row-major order, with v_0, v_1, and v_2 base vertices as indexed by the corresponding base triangle.

grid (matrix) sides and l_b is the maximum allowed subdivision level (the grid resolution). The scheme is outlined in Figure 2.15.

Storing data in row-major order, the linear index of the sample at grid coordinates (i, j) can be easily retrieved as

$$\frac{i(i+1)}{2} + j, \tag{2.51}$$

by recalling that the number of elements in a lower-triangular square matrix of size k is $k(k-1)/2$.

The grid coordinates of the linear index k, instead, can be retrieved by observing that the number of vertices in a regularly subdivided barycentric grid is a triangular number, and finding the row index using the formula to compute a triangular root

$$i_k = \left\lfloor \frac{-1 + \sqrt{1 + 8k}}{2} \right\rfloor, \tag{2.52}$$

$$j_k = k - \frac{i_k(i_k+1)}{2}. \tag{2.53}$$

Chapter 3

Photogrammetry mesh data analysis

We are witnessing a proliferation of textured 3D models captured from the real world with automatic photo-reconstruction tools by people and professionals without a proper technical background in computer graphics. Digital 3D models of this class come with a unique set of characteristics and defects – especially concerning their parametrization – setting them starkly apart from 3D models originating from other, more traditional, sources. We study this class of 3D models by collecting a significant number of representatives and quantitatively evaluating their quality according to several metrics. These include a new invariant *metric we carefully design to assess the amount of fragmentation of the UV map, which is one of the main weaknesses potentially hindering the usability of these models. We have released the models we collected as a new public repository,* Real-World Textured Things *(RWTT), intended as a benchmark to systematic field-test and compare future algorithms. RWTT consists of 568 carefully selected textured 3D models representative of the most popular photo-reconstruction tools currently available, along with the quantitative measures we extracted.*

3.1 Introduction

In the last decade, many research and implementation efforts have culminated in the development of algorithms, technologies, and suites that facilitate creating and publishing 3D content , attracting users not necessarily equipped with a technical background in computer graphics or geometry processing. Among the most influential and exciting advancements, photo-reconstruction techniques played a primary role, impacting a variety of application fields such as artwork and architectural documentation, environmental monitoring, data acquisition for manufacturing, 3D assets creation. Commercial and free software solutions are now available to both professionals and hobbyists, and can be used to generate accurate 3D digital copies of real-world objects in an almost fully automatic way.

This democratization of 3D content creation determined the diffusion of textured 3D digital models sharing certain specific characteristics, differing both qualitatively and quantitatively from those of 3D representations typically targeted – and assumed – by the large majority of academic works on computer graphics and geometry processing. Consequently, there is now an increased demand for robust techniques designed to process and manipulate this class of 3D models, and further research efforts are required to fill this gap.

Characteristics of Textured Models Generated with Photo-Reconstruction Tools Digital 3D models acquired with photo-reconstruction techniques share a combination of traits that cannot

be found in any other class of 3D models (such as meshes digitally sculpted or manually designed by professional 3D modelers, range-scanned models, or procedurally generated models).

The **meshing** is typically high-resolution but low quality, triangle-based, and completely irregular. The frequent inaccuracies in the 3D reconstruction result in geometrical noise and meshing defects. While many solutions have been suggested to face these problems [ACK13, Ju04, CPLW19], the proposed approaches focus on the geometrical aspect and do not take into account the presence of an existing parametrization. Even more importantly, geometry is but one aspect of the description: photo-reconstructed meshes are enriched with one or multiple high-resolution textures derived from the same set of photographs used to reconstruct the shape; these are a central part of the description of the object, as well as its UV map, i.e. the parametrization defined to map the texture data onto the object surface.

The **UV maps** of the 3D models present the most distinctive traits, directly reflecting the way they have been constructed (see Section 3.1.1 below). Specifically, they are atlas-based parametrizations where the atlases are typically extremely fragmented and with irregularly shaped charts, to a much larger extent than other categories of models (see for example Fig. 3.1).

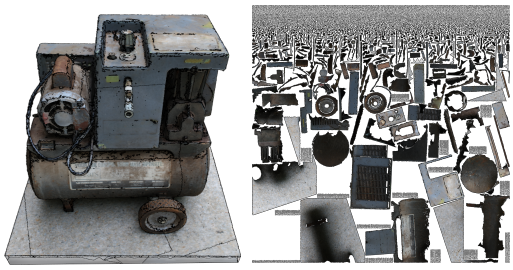


Figure 3.1: One model of the RWTT dataset, with its texture and UV map.

This drastically exacerbates the numerous defects and costs related to texture seams and UV map discontinuities, impacting downstream applications. The reader is referred to [YLT19] for a more comprehensive review of these issues. In summary, they include poor GPU memory usage, due to a combination of packing inefficiency and the need to replicate texels on a small band around each seam to avoid noticeable texture "bleeding" artifacts; unavoidable minor rendering artifacts caused by the inconsistent bilinear interpolation on the seams, even in the presence of said texel duplication; inability to use MIP-mapping without incurring in additional bleeding artifacts (unless the width of the texel duplication bands is enlarged exponentially to the maximal MIP-map level); vertex duplications in the data structure encoding the mesh, which are necessary to be able to encode the seams in the first place, implying an additional cost both in memory and per-vertex workloads; consequent constraints on processing or preprocessing performed over the mesh data structure, for example hindering to some extent the compression, streaming, coarsening, construction of multi-resolution structures (such as [CGG*05]).

Contributions With this in mind, in this chapter we offer the following contributions. We present an **analysis** of the aforementioned class of 3D models and their characterizing features, by collecting a large sample of representative models and performing systematic measurements, according to several criteria that we identify as relevant for geometry processing algorithms and computer graphics applications in general. We focus on the UV maps, its presence being one defining characteristics of this class of models.

The analysis includes a **new quality measure**, the atlas *crumbliness*, designed to capture the severity of texture seams in a given UV map. This new measure is succinct, easy to compute, and has desirable properties such as being invariant to UV scaling and mesh subdivision.

Since our analysis shows that the quality of the parametrizations in our sample data is low, we **evaluate** a selection of commercial, open source and academic software on the task of recomputing UV maps for these 3D models, and show how existing algorithms and implementations typically either fail, are unable to improve the quality of the maps, or are not practical.

We make our **dataset**, Real-World Textured Things (RWTT), publicly available on the web. The dataset has been curated and organized to facilitate batch-processing, and is suited to test, benchmark, and challenge existing and future geometry processing algorithms. Overall, RWTT features 568 models generated by users of the aforementioned photo-reconstruction technologies and released with permissive licenses, striving to represent the wide variety of existing user categories, photo-reconstruction tools, application fields and styles. Our aim is to provide a standard, realistic, and challenging testbed for algorithms, able to indicate their usability in the real world. All models in RWTT come pre-assessed according to the measures we collected, and are browsable and searchable by them.

Finally, we provide the **measuring tool** we used to evaluate the models as a publicly available, stand-alone software application: TexMetro (Section 3.7). TexMetro produces a set of meshing and UV quality measures and statistics for user-provided models (identical to the ones we used to evaluate the RWTT dataset), and is intended as a tool to help researchers and practitioners assessing and comparing the effectiveness of future algorithms and techniques designed to target this class of models. In this, we are inspired by the impact that tools of this kind had in the geometry processing community (Section 3.2.1).

3.1.1 Background in 3D Photo-Reconstruction

Photo-reconstruction techniques [HZ04, SCD*06, RSN*14] generate a textured 3D mesh starting from a set of high-resolution 2D pictures shot at the subject. In Chapter 2 we have presented the steps of the photogrammatric reconstruction pipeline, which we briefly review here. There are many possible variations, but the general principle can be summarized as follows.

First, Structure-From-Motion techniques are used to estimate camera parameters: feature points are identified on each input image, then the matching feature points are identified across the images using appropriate similarity measures. In a global adjustment phase, a set of camera positions is estimated, together with 3D positions of feature points, to maximize consistency, e.g. by minimizing the back-projection error, or by maximizing the color consistency across images. Once the camera parameters have been computed, Multi-View-Stereo algorithms are employed to generate a dense point cloud.

Finally, the point cloud is processed to remove noise and outliers, and fused into one polygonal mesh representation. The mesh representation is used to estimate the occlusion and visibility of each pixel in the picture. A subset of the visible pixels are then copied into one or more high-resolution textures, which provide the majority of the visual detail of the final result. Textures are sometimes post-processed to alleviate shadow artifacts, color jumps when switching from one source image to another, or can optionally store additional material and surface information (e.g. surface normals).

Each step of this chain constitutes a challenging problem, which has been studied in separation or within complete systems. The research on each sub-problem has recently matured enough so that a large number of off-the-shelf tools integrating implementations of all the used techniques are now available either as commercial or open source software [Cap22, Aut22,

Agi22, 3Df22, Pix19, MMPM16, RCD*13, Ben19, itS19].

For our purposes, the most relevant part is the construction of the UV map. Parametrization construction techniques traditionally employed in other contexts are often not adequate to the task, in part because these methods often lack the full automatism required in this scenario. Moreover, many methods (automatic or user-assisted) are hindered by the characteristics of the meshing (i.e. its large resolution and various defects). At the same time, the presence of a set of images detailing every part of the object surface unlocks UV map construction strategies that are only available in this scenario. Namely, each mesh triangle can be assigned to one of the photos using heuristics based on estimated visibility or other factors, implicitly defining the components of the UV atlas which then just need to be packed into one or more textures.

Note also that that mapping distortions, traditionally of central importance in the effort of constructing a good quality UV map for a given mesh, are not particularly relevant in this context. Differently from most other scenarios, where a UV map is constructed before the texture is filled with the signal samples, here the texture content *precedes* the UV map construction. Because the source of the texture colors are limited to the images, there is no gain in diminishing the mapping distortions which can already present from these images to the 3D surface. Therefore, instead of reducing the distortion by optimizing a suitable map, the optimization strategy can only consist in choosing the least distorted input image to use as a source for a given triangle (for example, the image offering the most orthogonal view of that portion of the surface, barring other considerations such as occlusion, presence of highlights, optimization of island shapes, and others).

These factors determine that the UV maps commonly found in this important class of models feature unique and specific characteristics, motivating the analysis presented in this chapter.

3.2 Related Work

The practice of creating good, comprehensive, realistic, challenging benchmarks is well recognized, and there is no need to state their importance in promoting good practices and improving replicability, comparability, and validation of research. In this section we briefly summarize other dataset contributions made to the geometry processing community and public repositories of textured models. There are many geometric and computer vision datasets available that target a large variety of tasks.

Datasets for machine learning generally feature large collections of objects, and include metadata and annotations to facilitate the training and validation of learning-based algorithms. The ModelNet benchmark [WSK*15] is a collection of more than a hundred thousand CAD models, and has been successfully used to train and benchmark object recognition algorithms. ShapeNet [CFG*15] is a large collection of annotated 3D objects, providing geometric and volumetric data, screenshots, textual descriptions and categorizations. On a similar note, the ABC dataset [KMJ*19] is a massive dataset of one million CAD models proposed to test the effectiveness of learning-based approaches to the solution of common geometric problems such as computing surface normals.

Scanning campaigns have produced popular model repositories, such as the Digital Michelangelo Project [LPC*00] and the Stanford Scanning Repository [LGCP05], whose models are known to the entire computer graphics community. None of these models, however, have textures, and they have been generated using 3D scanners; in contrast, the models featured in our dataset are created with photo-reconstruction algorithms.

The Princeton Shape Benchmark [SMKF04] is a large collection of low-resolution models with very simple geometries. [TM03] focus instead on high-resolution models. Both datasets are now rather outdated.

The wide availability of consumer-level hardware capable of capturing depth images to produce object scans has resulted in a proliferation of datasets based on RGB-D data. The review of those datasets is beyond our scope, and we refer the reader to the excellent survey by [Fir16].

More related datasets are Thingi10K [ZJ16], which features real-world objects and targets 3D printing shapes, and [CZMK16], a large dataset of more than ten thousand object scans and the associated RGB-D data that, like ours, features models obtained from 3D reconstruction, but again without textures.

SceneNN [HPN*16] and ScanNet [DCS*17] feature annotated scenes reconstructed from RGB-D video sequences. Among the existing datasets, these are, to our knowledge, the most similar to ours; however, they do not contain models reconstructed with photogrammetric techniques, feature mostly indoor scenes, and the 3D reconstructions are not particularly accurate.

As outlined above, geometric and computer vision datasets rarely include textures. Textured models can be found in asset stores such as the Unity Asset store [Uni19], the Unreal Marketplace [Epi19], and online repositories such as TurboSquid [Tur19]. Models from these sources however are typically non-free, sculpted, and manually textured by 3D artists.

In contrast to all of the above, Real-World Textured Things is a collection of large, textured 3D models generated with a variety of photo-reconstruction software tools, and published with permissive licenses to allow use in research and other practical contexts.

3.2.1 Quality Measurement Tools

The fruitful idea of providing public tools intended to be used to evaluate specific quality measures of user-provided models has several predecessors. Among the most notable is *Metro* [CRS98], which computes and reports the Hausdorff distance between pairs of surfaces provided as triangular meshes. Over the last two decades, *Metro* has helped to assess and compare vast amounts of algorithms, e.g. for surface simplification, remeshing, surface reconstruction, and other geometry processing tasks, as reported by thousands of research papers [Goo19]. As an additional benefit, the adoption of *Metro* contributed to uniform the way quality measures are defined and implemented. Overall, *Metro* has proven to be a tremendous aid to the research community striving to invent new algorithms and solutions toward a given set of geometry processing problems.

3.3 Dataset

As mentioned in the introduction, one of the motivations behind the creation of the Real-World Textured Things dataset is to provide researchers and developers a challenging testbed of textured 3D models indicative of the actual models that can be encountered in practice. Rather than generating the models in-house, we have collected publicly available models created and released by professional practitioners and hobbyists. This strategy has been successfully employed by other authors (e.g. ZJ16, KMJ*19) and has the important advantage of allowing us to include models generated with various software packages, which is crucial to obtain a representative sample of the 3D data that can be found in the real world. In particular, it is reasonable to expect the emergence of some common traits characterizing the geometry and

textures of the models in the dataset according to the implementation details and algorithmic choices of different reconstruction pipelines.

At the time of writing, the dataset features 568 models created by 129 unique authors using at least 18 different acquisition pipelines. Fig. 3.2 reports a taxonomy of the models featured in the RWTT dataset, categorized according to the reconstruction software used to generate them; this information was obtained from the textual descriptions and tags available when the models were retrieved.

We have collected models that were published across 6 years from 2013 to 2019 from three sources: [Ske19], the Smithsonian 3D Program [Smi19] and the Zamani Project [RSB*20]. SketchFab is a popular online repository for publishing and sharing 3D content used by thousands of users and constantly growing; it features a large collection of 3D models reconstructed from photographs and conveniently exposes a web API to automatically retrieve metadata such as the model author, description and tags, among others. The Smithsonian 3D Program is a digitization project from the Smithsonian Institution which published on the web and made available a selection of 3D reconstructions of artifacts and relics. The Zamani Project is an ongoing effort to document important heritage and archaeological sites in Africa by providing, among other material, high-resolution textured 3D reconstructions.

We have been careful to almost exclusively include models that have been released with permissive licenses to allow anyone using the dataset to freely share their results: all of the models from SketchFab and the Zamani Project are released under various declinations of the Creative Commons 4.0 license; the models from the Smithsonian 3D Project follow the "Smithsonian Terms of Use", allowing fair use as long as the original source is cited. We plan to keep the dataset updated by adding new models in the future.

The Real-World Textured Things dataset is available and can and browsed by model metadata at `https://texturedmesh.isti.cnr.it`.

3.3.1 Dataset Curation

The dataset is organized as a collection of folders using a progressive naming scheme to facilitate the automatic processing of the 3D models. Each model in the dataset is assigned a progressive identifier ID and is stored in a folder named RWTID.

For each model, we provide the following data:

- The 3D model in OBJ format with a corresponding MTL file defining the materials
- The texture images referenced by the materials
- A JSON file with the provenance metadata of the model
- A JSON file with geometric and parametrization measures

Although the original models and textures came in a variety of formats, we have converted all of them to the same format for uniformity and convenience; in several cases, manual fixing of the object files was required to restore correct material and texture image references. All the models are provided as OBJ files, and the textures as PNG images. Whenever necessary, the conversion only involved the file formats while the 3D and image data was left untouched; the models archived in their original formats are available separately. The textures we provide generally encode surface colors; however, several models also come with high-resolution

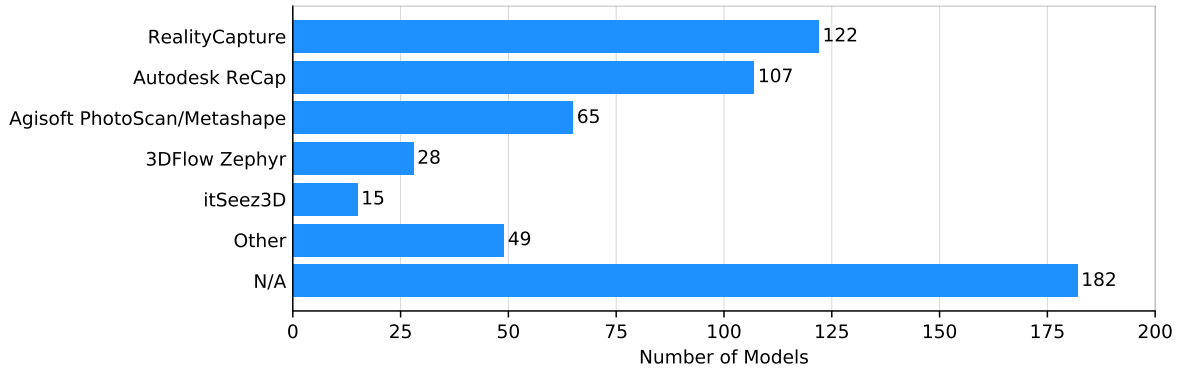


Figure 3.2: Overview of the different software tools used to create the models featured in Real-World Textured Things. There are at least 18 different reconstruction pipelines represented in the dataset; here for compactness we only report the most frequent occurrences.

textures storing other signals such as surface normals, roughness, specularity, and ambient occlusion.

Each 3D model comes with a corresponding *metadata.json* file reporting information such as the model name, the original author, the date on which the model was published and the license used, a semantic category we assigned to the models for classification purposes, and a unique resource identifier. The unique identifier points to the original data source: SketchFab models use the URI assigned by SketchFab itself, models published by the Smithsonian 3D Program use the web link to the object information page, while models published by the Zamani Project use their Digital Object Identifiers (DOIs). The metadata file optionally also reports, if available, the software used to generate the model, a textual description of the subject and a collection of tags that the original author may have provided.

Finally, we also include a *texmetro.json* file generated with the TexMetro tool. This JSON file contains information about the geometry, parameterization, and textures of the models; a detailed description of the TexMetro tool and its output is given in Section 3.7.

3.4 Quality Measures for Textured 3D Meshes

The quality of a textured mesh can be assessed by several quantifiable criteria, reflecting the resources that will be necessary to deploy the model (e.g. store it, stream it, or render it) and the quality of the rendered images, and thus the degree of usability of a given model in a specific context.

Models in RWTT are scored according to a suitable set of such measures, which is described in this Section. Different measures can impact downstream applications in different ways. Collectively, the set of measures reveals the peculiarity of the class of meshes we are focusing on, setting it apart from meshes commonly encountered in contexts different from automated photo-reconstruction.

Our set of measures includes general criteria such as data size and surface genus, UV map measures concerning the parametrization, and defects (number of occurrences of typical inconsistencies).

3.4.1 General Measures

Geometric and topological measures include the model resolution in terms of faces, vertices, and edges, the surface area, the number and total length of the mesh boundaries and the surface genus. Topological information is relevant since many geometry processing algorithms require strict assumptions on the topology of the input data. Texture resolution and number of textures are also meaningful measures, as these are all quantities that would make this data impractical to be handled by algorithms that do not scale well with the input size.

3.4.2 UV Map Measures

The quality of the surface parametrizations and texture maps are the main focus of our analysis and in the following we provide a detailed description of the measures we have adopted.

UV Occupancy

We define UV occupancy as the ratio of the number of texels strictly inside any UV triangle over the total number of texels. A better UV occupancy results in a higher texel density over the surface for a fixed texture resolution, or a lower GPU memory footprint for the same rendering quality.

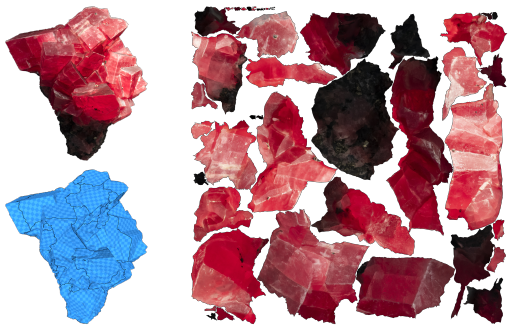
Occupancy is affected by the quality of the atlas packing: tightly packed atlases waste fewer texels between the charts. Atlases that are highly fragmented also have a worse occupancy, as they require comparatively more UV area to ensure that the charts are suitably distanced from each other to avoid mip-mapping artifacts.

Atlas Crumbliness and Solidity

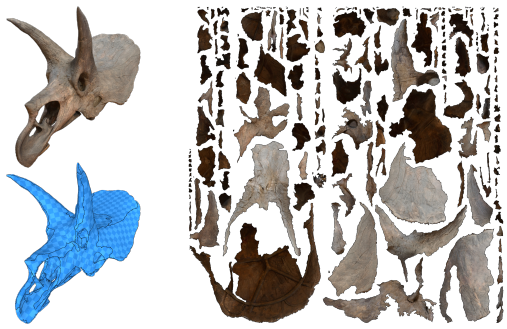
As summarized in the introduction, the presence of texture seams is somewhat detrimental to the usability of the dataset, for several different reasons. While texture seams are, to some extent, necessary in any textured mesh, we focus on the presence of redundant texture seams that are due to the fragmentation of the UV atlas independent from the complexity of the reconstructed 3D shape (see Fig. 3.3). Therefore, we need to adopt a quality measure to assess the level of fragmentation of an existing texture atlas.

Because there is no standard, we devise a new measure for this. We consider several alternatives. We observe that the number of charts is not, alone, a good descriptor, for two reasons: first, charts can have varying sizes and boundaries of varying complexity; second, a chart may include "non-disconnecting" cuts, i.e. cuts that do not split it into disconnected components. The number of seam edges, or the ratio of seam edges to non-seam edges, are also not adequate: these measures would be affected by a regular subdivision of the mesh, which does not affect the level of fragmentation. Similarly, the total length of the seam edges would not be invariant to UV scaling.

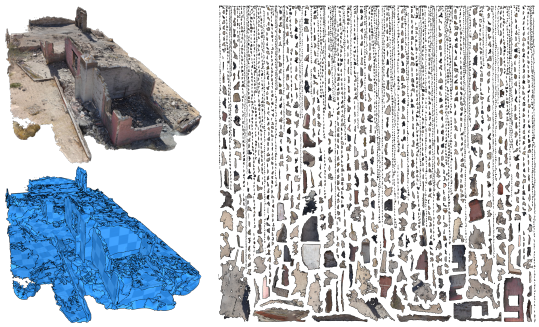
Instead, we adopt a new measure, which we name *crumbliness*. Crumbliness is the ratio of the total length of the perimeter of the atlas charts, summed over all charts, over to the perimeter of an ideal circle having the same area as the summed area of all charts. Because the circle is the shape maximizing the area for a given perimeter, this measure has 1 as the lower bound. In formulas, if a_i is the area of chart i and p_i is its perimeter, the crumbliness C of the atlas is



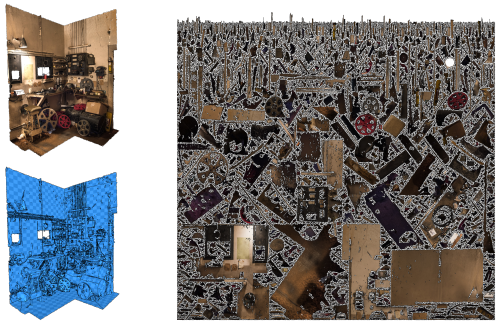
(a) Crumbliness = 7.4, Solidity = 0.135



(b) Crumbliness = 21.1, Solidity = 0.047



(c) Crumbliness = 94.7, Solidity = 0.010



(d) Crumbliness = 136.8, Solidity = 0.007

Figure 3.3: Four models with their texture seams highlighted. A few of the texture maps featured in our dataset have a relatively low chart count (3.3a). Many others exhibit a surprisingly large number of texture seams, unjustified by their geometrical or topological complexity (3.3b, 3.3c, 3.3d). *Crumbliness* and *solidity* measures capture the fragmentation of the texture atlas.

given by

$$C = \left(\sum_i p_i\right)\left(4\pi \sum_i a_i\right)^{-\frac{1}{2}}$$

Crumbliness is independent of the mesh resolution, invariant under element subdivision, and scale-independent. As such, it can be used to compare atlases of different models and for textures of varying sizes.

We also define *atlas solidity* as the inverse of crumbliness. Compatibly with many other quality measures, atlas solidity has an upper bound of 1, and higher values indicate a better quality.

We observe that the square of the crumbliness gives, up to a constant factor, the quantity sometimes referred to, in geometry, as the "isoperimetric ratio" of a given 2D shape, which is the ratio between its squared perimeter and its area. We favor the use of "crumbliness" and "solidity", as their name is more descriptive and more intuitive to correlate with quality, their numerical quantities are easier interpreted (being normalized), and they empirically appear to distribute better among real-world datasets, compared to their squared counterparts (see Fig. 3.7).

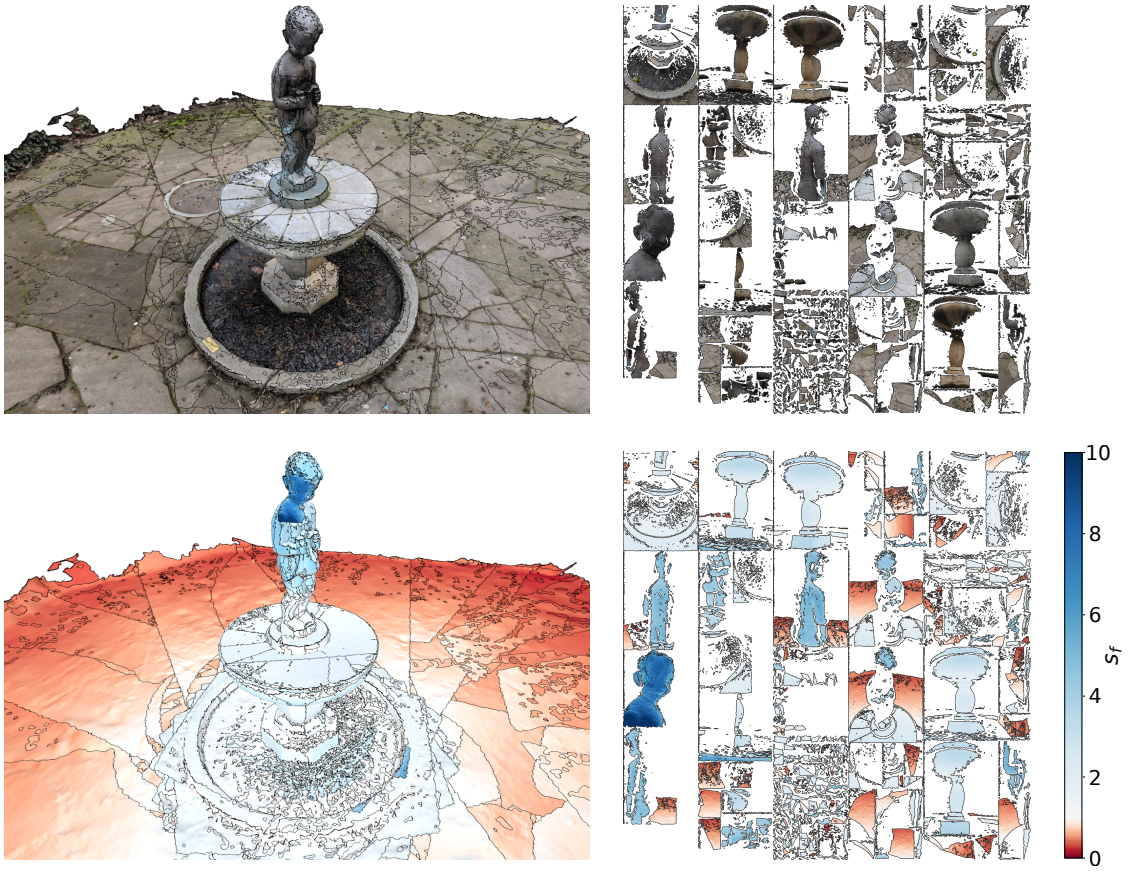


Figure 3.4: Visualization of the scalar field s_f on a model affected by uneven sampling frequency. Regions densely and sparsely sampled are colorized in blue and red respectively, reflecting the amount of color information encoded in the texture image. In this case, the model was clearly reconstructed using close-ups of the statue with the pavement left in the background.

Uniformity of Texture Sampling

A well-understood characteristic of any parametrization is the distortion it introduces. In the ideal case, the mapping between the surface and the texture preserves areas, angles, or lengths (implying both areas and angles); the failure in preserving these features is measured by various *distortion measures*. Conformal distortion reflects angles variation, area distortion reflects area variation, and isometric distortion reflects length variation. A multitude of different measures have been presented in the literature (for each case), usually to minimize them during automatic UV map construction (see FH05 for a survey and, for more recent examples, LZX*08, SS15, JSP17, PTH*17 among many others).

In our context, which is texture mapping, distortion is relevant because it directly determines how texture samples will be distributed on the surface. Area distortion translates to nonuniform sampling density, and conformal distortion translates to anisotropic sampling.

Sampling density can be hugely varying in models of RWTT, due to the different relative resolution of the pictures used to reconstruct the models, and the different distance between the acquiring camera and the captured object. Instead of adopting any specific measure of area distortion, we choose to measure directly its effect by analyzing the density of texture samples over the mesh.

To this end, we define a per-element scalar quantity s_f that represents the texture sampling density as the ratio between the local (per-element) and global (per-mesh) change of scale from 3D to UV:

$$s_f = \frac{\mathcal{A}_{\text{UV}}(f) / \mathcal{A}_{\text{3D}}(f)}{\sum_f \mathcal{A}_{\text{UV}}(f) / \sum_f \mathcal{A}_{\text{3D}}(f)}.$$

Where $\mathcal{A}_{\text{UV}}$ and $\mathcal{A}_{3D}$ indicate element areas computed in UV and 3D space respectively. The local value of s_f is close to 1 for uniform sampling distributions, approaches 0 as the mapping resolution of f gets smaller relative to the global resolution of the map, and becomes greater than 1 otherwise. Therefore the scalar field s_f indicates, over a model, where the color signal is undersampled and oversampled w.r.t. to the average sampling density of one given model. Fig. 3.4 shows an example of how uneven sampling can impact the quality of a textured model when it is rendered.

As aggregate measures, we take the variance and percentile ranges (which may be less sensitive to outliers) of s_f over the mesh (note that the average is 1 by construction). Small values reflect more uniform sampling, and the minimal value 0 represents a perfectly uniform sampling. Note that this is not necessarily desirable for models in our class; in fact, even a vastly nonuniform sampling density can correctly reflect the different semantic importance of certain parts of the model relative to others (such as an object in focus vs. background) or different quantities of available color data.

UV Conformal Distortion

UV maps that exhibit low conformal distortion are desirable because they imply a more isotropic texel sampling distribution over the surface, which positively affects the quality of the signal reconstruction and avoids texture stretching artifacts. Because this factor is not captured by the sampling density, we measure conformal distortion separately.

To this end, we adopt a measure based on the quasi-conformal distortion [KSS06], which is the ratio of the smallest to largest singular value of the Jacobian of the mapping. We define the UV atlas conformality as the inverse of the quasi-conformal distortion; we opt for this formulation so that the maximal of value of 1 corresponds to the best case (a perfectly conformal map), consistently with most of the other quality measures in our set.

Discrepancy in the Signal Reconstruction at Seam Edges

A specific defect linked to the presence of texture seams is the possibility of observing discrepancies in the signal reconstruction at each side of the seam, which can be due to either mismatching texel values, or to inconsistent bilinear interpolation between texture values. This does not depend on the UV map alone, but on the texture data as well; therefore, we include a separate measure to assess the extent of this problem for a given model.

To this end, we adopt a measure of seam discrepancy recently defined (for the purpose of minimizing it) in [LFJG17], which, for completeness, we describe in the following.

Given an UV edge e, let $e(t)$ denote the linear combination of its endpoints, with the parameter t restricted to the $[0, 1]$ interval. A (manifold) edge e of the 3D mesh is a seam edge if corresponds to two non coinciding UV edges e_1, e_2. Let f be the function returning the texture lookup value for a given UV coordinate, using bilinear interpolation of the four closest texels.

The reconstruction discrepancy at a seam edge e is then defined as:

$$D(e) = \int_0^1 |f(e_1(t)) - f(e_2(t))|\, dt.$$

As a global discrepancy measure, we report the average discrepancy of all the seam edges in a 3D model

$$D(S) = \frac{\sum_{e \in S} \|e\| \cdot D(e)}{\sum_{e \in S} \|e\|},$$

where S denotes the set of seam edges, and each seam edge contribution is weighted by its length $\|e\|$ to capture the visibility and severity of the resulting artifacts independently from the mesh subdivision. In our implementation, edge discrepancies are estimated using discrete samples.

3.4.3 Defects

Another defining characteristic of datasets of RWTT is the common occurrence of defects and inconsistencies of specific nature. These issues are not usually considered when geometry processing algorithms are developed on paper, which often assume to be receiving sanitized input. Different issues can be encountered in the geometric part, or the parametrization part.

Defects of geometry include extremely irregular triangulations, zero-area triangles, topologically degenerate triangles (referring the same vertex more than once), and unreferenced or duplicated primitives. Topological defects include errors such as local loss of two-manifoldness, holes, and incomplete meshes. Another potential source of issues is the presence of "topological noise", i.e., small handles unrelated to the ideal topology of the reconstructed 3D shape (typically generated during the surface reconstruction phase from noisy dense point clouds), which is revealed by incongruous genus number with respect to the overall model shape.

Defects of the UV map include lack of injectivity caused by UV overlaps, which result in the same texture data being erroneously mapped to multiple parts of the mesh. Overlaps in the UV map can be either *local*, when triangles flip in UV space and change orientation relative to their neighbors, or *global*, when the boundary of one or several charts intersect in UV space. While they cause the same problem, it is useful to distinguish between local and global overlaps, because the strategies available to counter them are typically different. Additionally, several datasets feature only partial maps, that is, UV maps that do not map a model surface in its entirety, likely due to unavailable or insufficient photographic data.

3.5 Measurements of RWTT Models: an Analysis

We extracted our set of measures from each model of RWTT. In this section, we offer an analysis of the results, which serves as a picture of the merits and limitations of the currently available off-the-shelf photo-reconstruction software.

3.5.1 Measures on Geometry

Real-World Textured Things features models of many different sizes, ranging from low-poly models made of only a few thousand primitives to high-resolution ones with millions of vertices and faces. The histogram of the geometric complexity of the dataset, expressed as the number of vertices, is reported in Fig. 3.5. On average, our models tend to be fairly large compared to those found in similar datasets: as an example, 37% of the RWTT models have

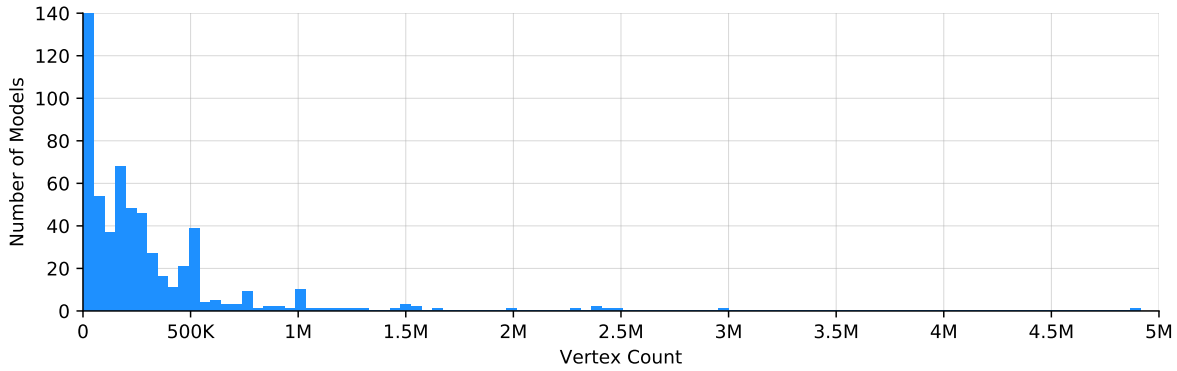


Figure 3.5: Geometric complexity of the 3D models found in our dataset. Models generated from 3D reconstruction technology are typically stored at high resolution to maximize accuracy and fidelity to their real-world counterpart.

more than 250K vertices compared to less than 2% of the models found in Thingi10K [ZJ16]. This is expected, as objects are typically reconstructed at high resolution and then optimized for efficient use in downstream applications if needed.

3.5.2 Texture Maps and Parametrizations

Texture Size Modern photo-reconstruction pipelines are able to generate accurate digital reproductions from hundreds or thousands of high-resolution images. Consequently, the majority of textures found in RWTT are high-resolution, ranging from 4K upwards. Moreover, several models feature multiple or non-square textures.

Texture Space Occupancy In Fig. 3.6 we plot a histogram of texture occupancy values found in RWTT, clearly showing how atlas packing is often sub-par, leading to unnecessarily large memory requirements for texture signals sampled at relatively lower densities. This is particularly significant as GPU memory is often a critical bottleneck in graphics applications.

Atlas Solidity We report the atlas solidity distribution over our dataset in Fig. 3.7. The plot highlights a bias towards low solidity values, indicating the frequent occurrence of texture maps affected by the presence of large amounts of seams.

The surprising fact is, however, that the degree of fragmentation witnessed in the majority of cases is almost never justified by the topological and geometrical complexity of the models we have collected (see Fig. 3.3). We conjecture that this happens because of two distinct causes. First, this happens because the textures are synthesized using large numbers of photographs, with reconstruction tools optimizing for color fidelity and attempting to store as much information as possible from the available images, disregarding the creation of many atlas charts. Second, this may also happen because of intrinsic limitations in existing automatic UV mapping technology, which we investigate in Section 3.6.

Conformality and Sampling Frequency Fig. 3.8 reports the distribution of average quasi-conformal distortion values for our dataset. Unsurprisingly, most of the texture maps exhibit good angle preservation, which is to be expected given the high degree of fragmentation affecting many of the atlases.

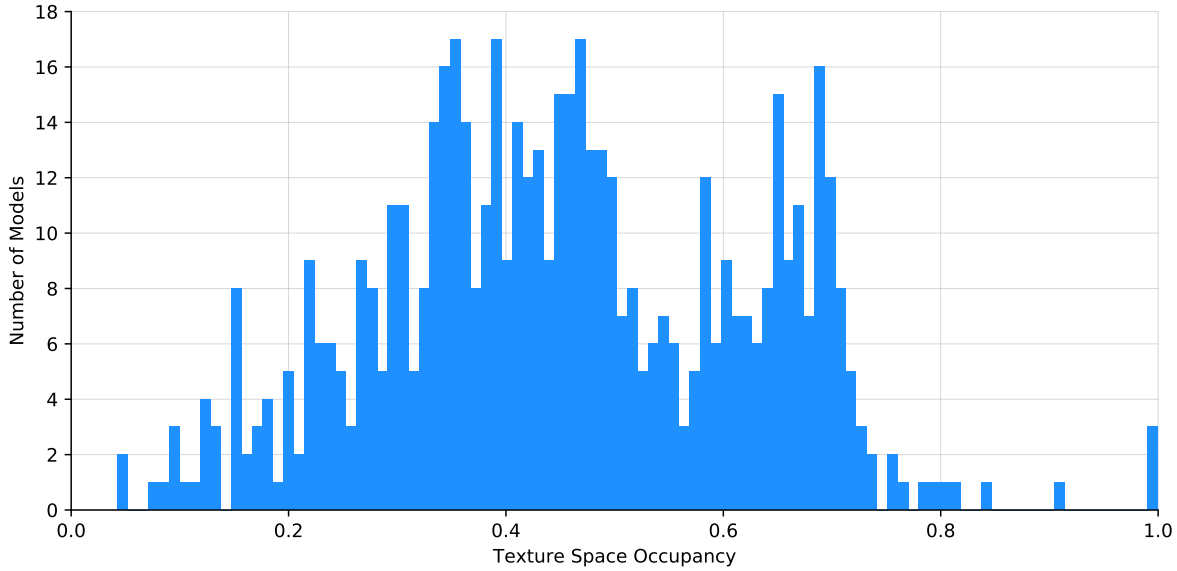


Figure 3.6: Distribution of the RWTT textures occupancy. Many textures are not efficiently packed, leading to a waste of GPU memory when the models are rendered in real-time applications.

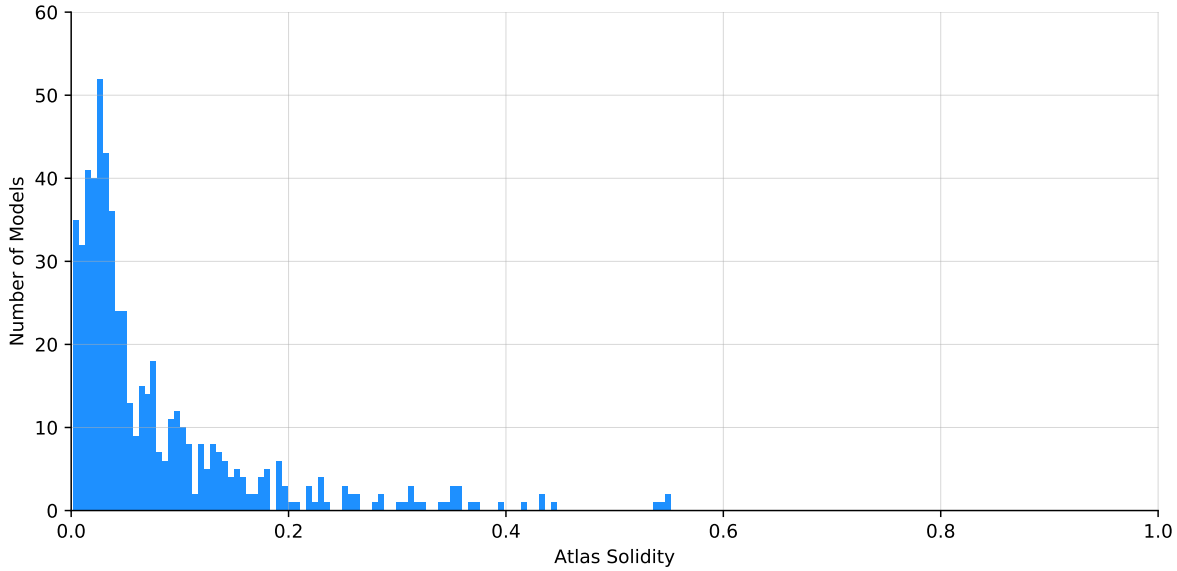


Figure 3.7: Distribution of atlas Solidity for models featured in the RWTT dataset. Many atlases exhibit high-to-extreme fragmentation: the model with the highest chart count has 555252 charts.

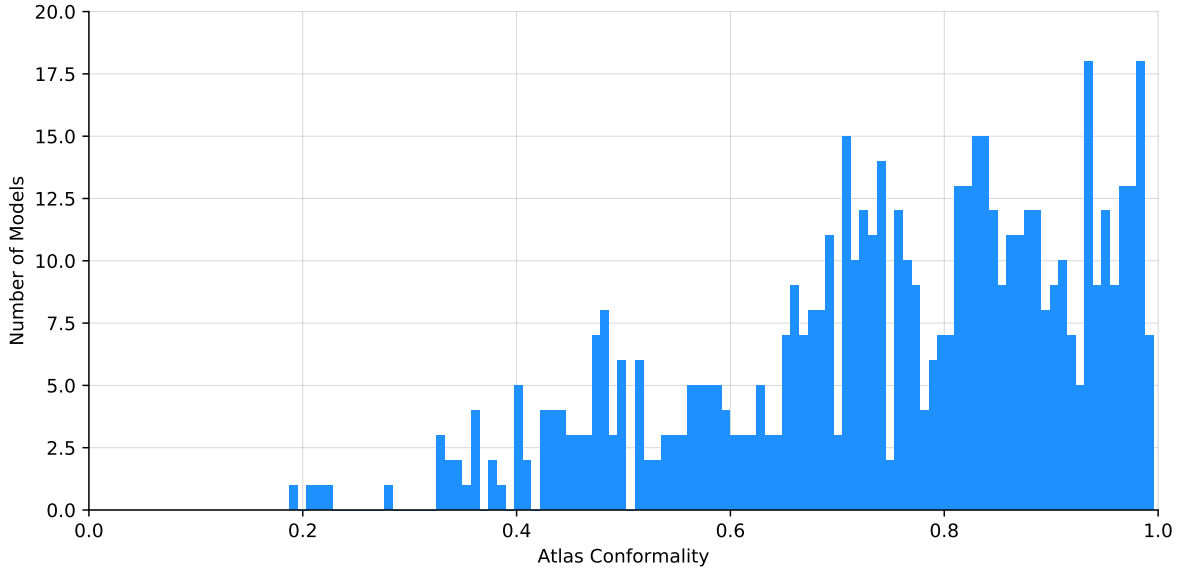


Figure 3.8: Distribution of atlas Conformality, where a lower value implies larger angle deviations. Despite many fine-grain chart decompositions, many parametrizations are still affected by non-negligible conformal distortion.

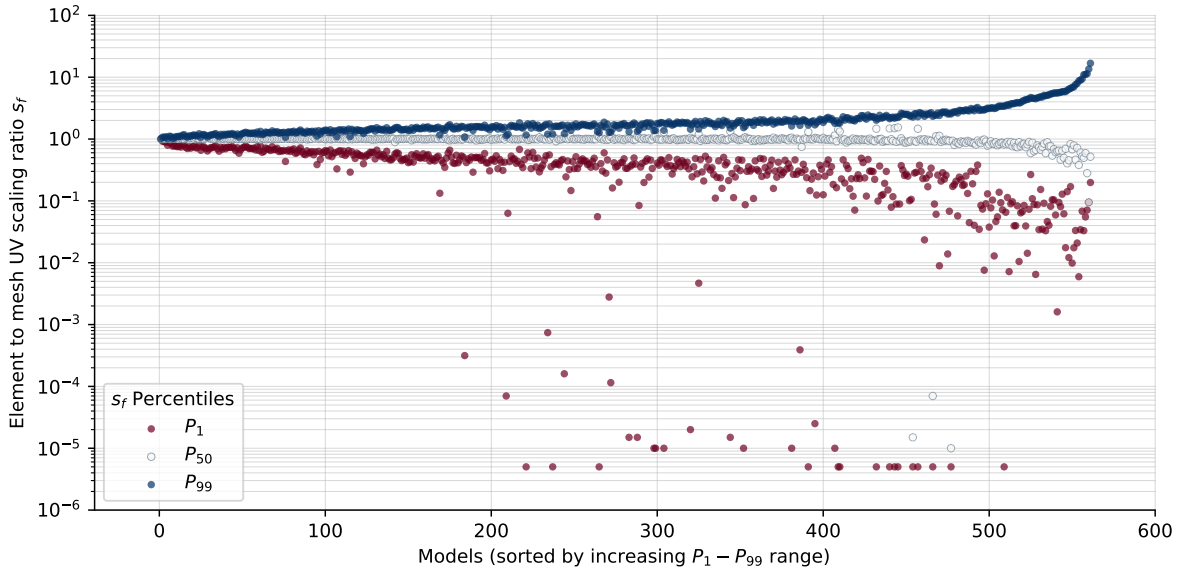


Figure 3.9: Visualization of the UV scaling ranges present in the dataset. We plot the 1st, 50th and 99th percentiles for the models sorted by UV scaling range width, highlighting how a significant portion of the dataset exhibits large variances in the mapping resolution.

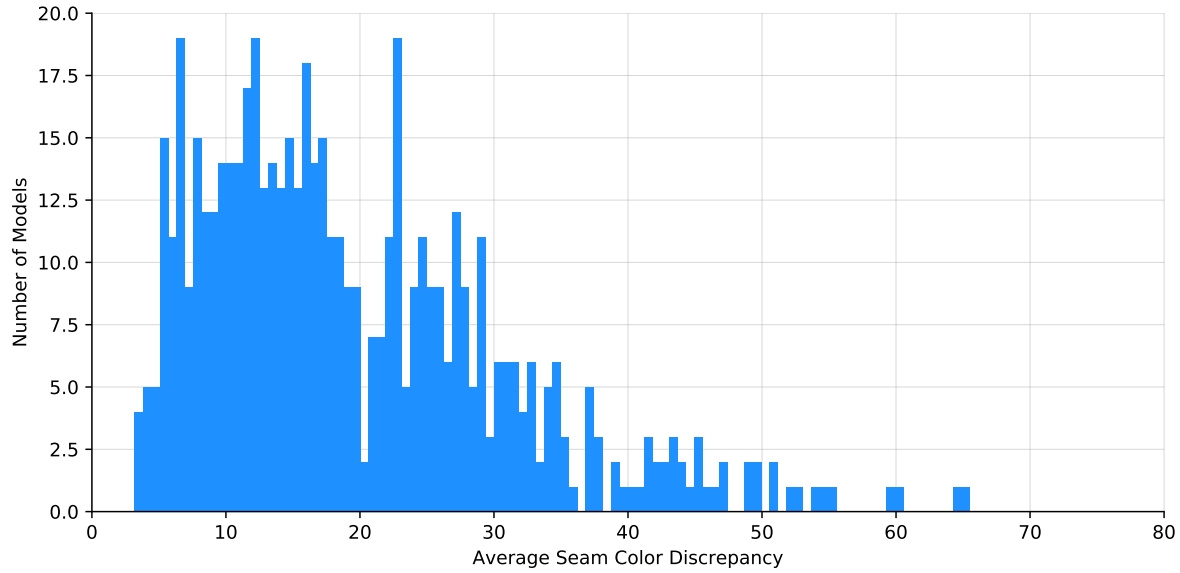


Figure 3.10: Distribution of average seam color discrepancy (measured in RGB space).

A scatter plot of the 1st, 50th and 99th percentile values of the models UV scaling, sorted by range from P_1 to P_{99} is reported in Fig. 3.9, showing how a significant amount of models exhibit a non-negligible spread in the UV scaling values and, consequently, non-uniform texel density across the surface. As already observed in Section 3.4.2, this is quite peculiar but also not particularly surprising, as models with less uniform sampling frequency distributions were likely generated from photographs that captured surface details at different scales (see for example Fig. 3.4).

Seam Color Discrepancy The distribution of average color discrepancies along texture seams is reported in Fig. 3.10. Overall, our dataset report fairly heterogeneous scores. There are two likely explanations for this. First, there can be differences in the uniformity of the object colors, whereby models with more varied colors are likely to exhibit more severe artifacts at texture seams. Second, there can be differences in the way texture seams are handled by the reconstruction tools during the texturing process: some tools may attempt to generate textures that are smoother across seams by optimizing for consistency in the texture data whereas others may prioritize fidelity to the photographic data.

3.5.3 Defects

The ability to deal with arbitrary input meshes without requiring preemptive cleaning is a key property of any robust implementation, whose importance has been increasingly recognized as testified by the emergence of more and more "real-world" datasets, and algorithms specifically designed to deal with them. RWTT is a further exemplification of this, and features meshes with all kinds of meshing and geometric defects, as well as defects related to texture mapping, such as local and global overlap of UV coordinates, and models that are only partially textured. An overview of the situation is reported in Fig. 3.11.

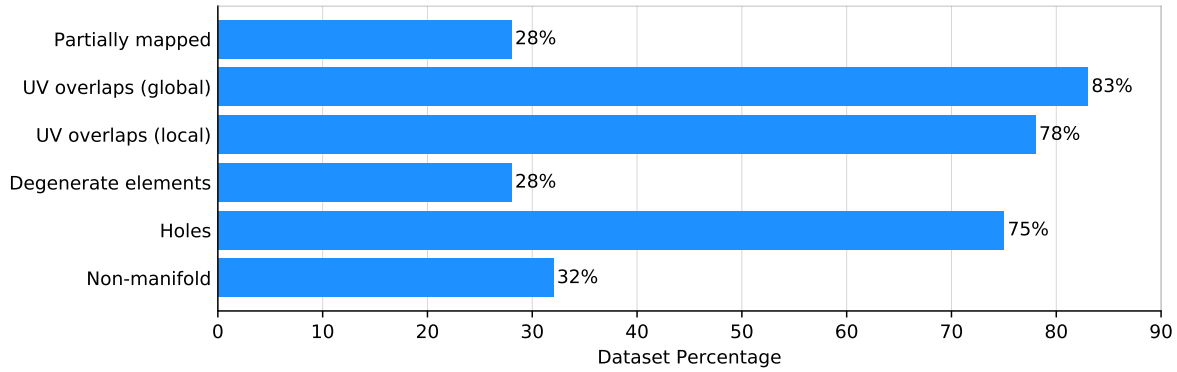


Figure 3.11: Common defects found in Real-World Textured Things models.

3.6 Recomputing UV Maps

Given the number of issues these UV maps exhibit, an obvious question is whether or not they are peculiar to the tools that generated the textures and parametrizations as part of the 3D reconstruction process, or if there is an intrinsic limitation in the current technology regarding the automatic UV mapping of complex and high-resolution models. Therefore, we have decided to evaluate the performance of existing state of the art UV mapping tools and see if they can produce maps of better quality than the ones found in our dataset.

We have performed experiments with commercial, open source and academic automatic UV mapping solutions. Maya [Aut19] is a commercial 3D modeling software that provides a simple automatic UV unwrapping procedure by projecting 3D faces to a fixed set of orthogonal planes. Blender [Ble22] also offers an automatic projection-based UV unwrapping functionality, but the set of projection planes is not fixed: instead, charts are computed by grouping together faces with similar normal vectors, and the projection plane of each chart is the one orthogonal to the average normal. Finally, we also experimented with OptCuts [LKK*18a], which is a recent academic work that automatically computes UV maps by alternating steps in which the distortion of the mapping is reduced, and steps in which seams can be added to the atlas charts to further reduce distortion, or removed if they are unnecessary.

For our experiments, we decided to set a time limit on the overall running time of 20 minutes. Since these tools cannot be conveniently automated to produce multiple textures, we have restricted the experiment only to models with single textures in order to ensure fair comparisons with the input data. We also restricted the experiments to models comprised of a unique connected component as the available implementation of OptCuts fails to initialize multi-component models. Moreover, to meet the topological requirements of OptCuts and ensure that the input mesh is two-manifold, we implemented a simple preliminary cleaning step that removes non-manifold vertices and edges, unreferenced vertices, and degenerate geometry. Note that, since OptCuts is an iterative algorithm, we saved the result obtained after 20 minutes of running time, rather than simply interrupting the task, if the algorithm was unable to reach numerical convergence within the given time frame.

The results of this experiment are reported in Fig. 3.12 as a 3D scatter plot of the texture occupancy, mapping conformality, and atlas solidity of the input data and the outputs produced by the three aforementioned implementations, with failure cases marked in red. Ideally, the scatter plots of the outputs should move towards the top corner at coordinates (1, 1, 1). Blender performs worse in terms of atlas solidity compared to the input UV maps, without managing

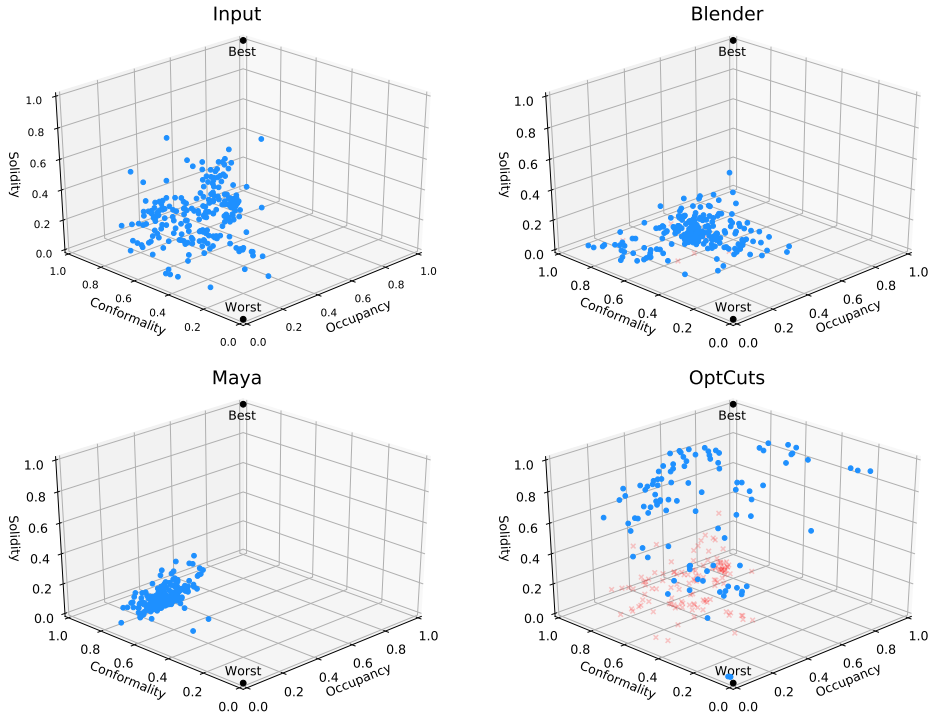


Figure 3.12: Performance of existing UV mapping tools on models generated with photo-reconstruction technology.

to improve under any of the remaining two metrics. Maya also generates heavily fragmented output atlases, but succeeds in significantly reducing the conformal distortion of the UV maps; note, however, that the results are affected by packing issues, as shown by the poor performance in terms of texture occupancy. OptCuts shows the best results despite being often interrupted before completing the atlas optimization, but is affected by a very high failure rate (54%). OptCuts failures are either due to numerical issues that prevent the correct initialization of the UV configuration to a bijective map, or to the inability to reduce the surface genus before starting. Finally, it is worth mentioning that, of the three implementations we tested, OptCuts is the only one that guarantees bijectivity of the output map; however, it is a much less efficient, practical and reliable solution.

Overall, our experimental evaluation shows that automatically computing UV maps for this category of 3D models is intrinsically challenging regardless of the method employed, and currently available methodologies are unable to consistently provide satisfactory results.

3.7 The TexMetro Tool

Alongside the dataset, we also release an open source command line tool – TexMetro – that can be used to compute all the measures described in Section 3.4 on arbitrary, user provided textured models. This tool can be used to assess and compare existing or future alternative photo-reconstruction tools (e.g. starting from the same set of images), as well as to assess the efficacy of existing or future methods aimed to improve the quality of 3D models.

The UV map, geometry, and connectivity of the input models are analyzed. The full set of statistics and measures are collected in a JSON file, while only a relevant subset of them is displayed on the standard output, consisting of salient information about the mesh geometry,

its parameterization and texture data. A description of the JSON file structure can be found in the documentation from the TexMetro code repository.

A few of the measures are dependent on the texture resolution (in texels): to compute these, TexMetro retrieves the resolution of the texture images referenced by the input model or, simply, accepts a set of texture sizes passed as command-line arguments.

The TexMetro tool is open source and is implemented using C++, Qt for interfacing, and OpenGL to accelerate texture overlap detection (which is done by rasterizing UV triangles over off-screen buffers).

3.8 Conclusions

In this chapter, we built a dataset of textured 3D models generated with automatic software, carefully selected and gathered from publicly available sources, and extracted relevant measures, including new ones specifically designed to capture the defects of this increasingly important and still neglected class of 3D models. We also introduced an open source tool, TexMetro (`https://github.com/cnr-isti-vclab/texmetro`) to collect measures and statistics on the geometry, parametrization and texture maps of user-provided models. The Real-World Textured Things dataset is available at `https://texturedmesh.isti.cnr.it`.

Our analysis revealed interesting traits that characterize these models. In particular, it is extremely common to find heavily fragmented texture atlases; this can be attributed to the fact that these parametrizations are generated automatically and tailored to the available pictures. On the other hand, experimental evaluation of current state-of-the-art automatic UV mapping tools, both commercial and academic, suggests that the issues that affect the UV maps of our dataset are not peculiar to the implementations of the software used to texture these meshes; rather, this seems to be a limitation of current automatic UV mapping tools and algorithms that either fail to produce better results or are not practical and robust enough to reliably undertake the task.

In the next chapters, we present methods designed to alleviate some of the issues we identified, in particular to reduce the texture map fragmentation and texturing artifacts (Chapters 4 and 5), and to optimize the representation of densely tessellated textured meshes for interactive rendering, storage and streaming (Chapter 6).

Chapter 4

Texture map optimization

We propose a method to improve an existing parametrization (UV-map layout) of a textured 3D model, targeted explicitly at alleviating typical defects afflicting models generated with automatic photo-reconstruction tools from real-world objects. This class of 3D data is becoming increasingly important thanks to the growing popularity of reliable, ready-to-use photogrammetry software packages. The resulting textured models are richly detailed, but their underlying parametrization typically falls short of many practical requirements, particularly exhibiting excessive fragmentation and consequent problems. Producing a completely new UV-map, with standard parametrization techniques, and then resampling a new texture image, is often neither practical nor desirable for at least two reasons: first, these models have characteristics (such as inconsistencies, high resolution) that make them unfit for automatic or manual parametrization; second, the required resampling leads to unnecessary signal degradation because this process is unaware of the original texel densities. In contrast, our method improves the existing UV-map instead of replacing it, balancing the reduction of the map fragmentation with signal degradation due to resampling, while also avoiding oversampling of the original signal. The proposed approach is fully automatic and extensively tested on a large benchmark of photo-reconstructed models; quantitative evaluation evidences a drastic and consistent improvement of the mappings.

4.1 Introduction

In this chapter, we present a technique that improves the existing parametrization of a given textured 3D model by reducing the amount of texture seams and texture charts, while at the same time striving to maintain the original parametrization as much as possible and minimize the resampling of the original texture images (Figure 4.1).

The objective of starting with an existing UV-map, and exploiting some of the information it brings, places our method in a different class, outside of the well studied – although still open – problem of *ex-novo* construction of a parametrization for a given 3D model. Before we proceed to present the details of our solution (Sec. 3.3), we illustrate the context that motivates this objective and the intended scope of our technique.

4.1.1 Context and motivations

As we discussed in Chapters 1 and 3, textured models generated with 3D photo-reconstruction techniques are becoming increasingly common, and come with a distinctive set of technical characteristics. The geometric resolution, texture resolution, and geometric accuracy are, generally, fully adequate for high-quality renderings; the textures in particular — directly

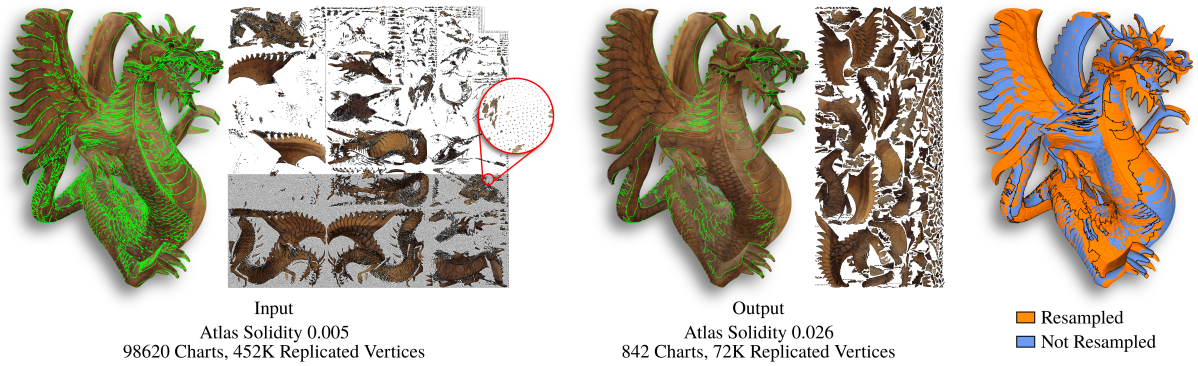


Figure 4.1: Our algorithm takes as input a textured 3D model (left), and iteratively merges texture charts to reduce the UV-map fragmentation (middle), leveraging the existing parametrization to minimize and possibly avoid resampling artifacts in the output map.

stemming from the original photographs — contribute greatly to the photo-realism of the renderings.

Our preliminary analysis, based on empirical evidence gathered from hundreds of data points, highlighted a recurring defect whereby the underlying UV-map is typically very low quality. This is likely because, in this class of meshes, the parametrization is produced as an integral part of the automatic texture reconstruction process. As a result, the UV-map often fragments the surface into an excessive number of "texture islands", many of them small, others larger but presenting long, twisted boundaries; overall, the models present an extremely large number of *texture seams* (also called texture cuts), as empirically evidenced by a recent study [MPCT20].

While texture seams are unavoidable in any textured mesh, their presence imposes a small toll on performance, memory resources, and quality of a rendering [YLT19] as summarized below. In normal meshes, the effect is usually negligible for most applications; in typical photo-reconstructed models, however, the disproportionate number of texture seams can result in significant strain on downstream applications, which can detract from usability, especially in resource critical contexts such as web-applications or video-games.

This situation is not accidental, and should not be attributed merely to an oversight or lack of engineering of current photo-reconstruction tools. Rather, it results from peculiar requirements and constraints of the UV-map construction inside the photo-reconstruction pipeline compared to generic authored 3D models.

UV-map construction for generic models When a generic mesh is to be textured, a UV-map needs be constructed starting from its "naked" geometry, for example by appropriately cutting the surface into disk-like regions, unfolding each of them into planar charts, and packing these charts on the texture rectangle. During this process, a blend of objectives is pursued, including limiting the geometric distortions (undistorted mappings are desirable because they imply an even texel distribution on the surface), as well as limiting the amount of cuts. Only after the UV-map is ready, the final texture is filled accordingly, by generating color values for each texel (for example, painting them, or extracting them from a high-resolution model [TCS03]).

With typical photogrammetry models, the above procedure is neither desirable, nor practical. First, in this case the final color information comes entirely from the photographs themselves; this means that an undistorted (area and angle preserving) mapping would not in any way be beneficial in this scenario: regardless of how well texel samples are distributed on the surface,

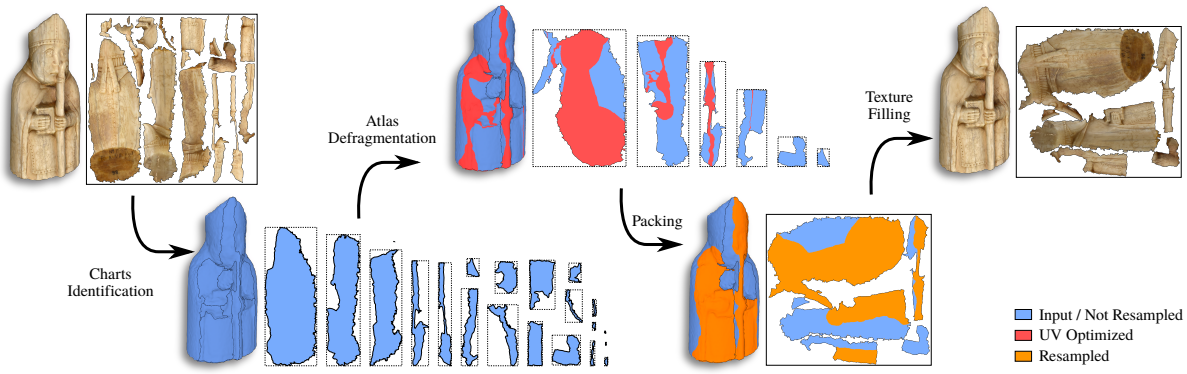


Figure 4.2: Overview of our method. We begin by extracting the charts from an input textured model, then iteratively merge them by optimizing a small area in neighborhood of the fused boundaries (in red). Then, we detect and repack the texture atlas, striving to limit the necessary resampling in the following texture filling phase (orange areas are resampled, blue areas are grid-preserving [RNLL10] and copied verbatim from the original).

any color information they contain will have to be extracted from the available samples, i.e. the pixels on the original images. Ideally, the texture samples should precisely match the input pixels. A related consideration is that rendering a textured mesh resamples the texture signal when the object is presented to the screen. If the texture data was already resampled from the original photographs then the original signal would be resampled twice, deteriorating it further. On top of this, an additional technical difficulty is that the mesh is typically high-resolution, extremely irregular, and affected by inconsistencies such as small holes, topological noise, or lack of manifoldness. These factors hinder both automatic algorithms and artist-assisted tools for UV-map creation. This is evidenced by the the experiments of our preliminary analysis reported in Chapter 3, which indicates that UV-creation tools, from either commercial software suites or research labs, fail to produce a satisfactory parametrization on more than 50% of photogrammetry models.

For these reasons, the UV-map creation phase of a photogrammetry 3D reconstruction follows a different approach, as follows.

UV-map construction for photogrammetry models In photo-reconstructed models, each part of the 3D surface is visible, by definition, from a portion of one or more photographs. Therefore, a natural mapping is already defined from each portion of the 3D surface into the 2D photo, defined by the view projection. The final texture is simply a collage of portions of the original photographs. The resulting UV-map features a local sampling density that reflects the actual availability of color data. For example, it assigns more texels to areas captured from close-up shots then to areas seen from afar. It also avoids any undue oversampling, or indeed any resampling of the original photographs.

As a bonus, this approach is extremely reliable, scalable, and fully automatic. The difficult task of creating a global parametrization is substituted by the much simpler task of just identifying the best photograph for each triangle, then clustering triangles according to the assignment, and finally packing the resulting regions in a final texture. This scales very well with the complexity of the mesh, and does not require any assumption on its consistency (e.g. manifoldness), although care must be taken to hide visible seams and ghosting artifacts emerging from small inconsistencies in the global registration of the available images [LI07, GWO*10].

The drawback is that, as noted, the mapping will present excessive fragmentation, because models are usually reconstructed from many photographs form different angles, and the assignment of triangles to photographs depends on occlusion and face normals, both of which vary on a per-triangle basis (especially for complex or noisy geometries).

Problems induced by texture fragmentation The penalty imposed by the excessive number of texture seams affecting typical photogrammetry models can be summarized as follows (see [YLT19] for a more detailed analysis):

- *Unused texture space*. The gaps unavoidably left by the imperfect packing of charts represent a waste of texture memory; in many photogrammetry models, this accounts for a significant percentage of the total texture memory on the GPU;
- *Replicated texels.* To minimize the visual artifacts at seams due to bilinear interpolation, color values must be replicated on the two sides of the texture in order to avoid the visual artifact known as "texture bleeding". This means that the number of *unique* texture samples can be much smaller than the number of stored texels;
- *MIP-mapping incompatibility.* The above problem is exacerbated by MIP-mapping, which requires a much wider gap around charts to avoid texture bleeding when minification filtering is used;
- *Incompressible residual visual artifacts.* Irrespective of any amount of texel replication, the seam will always be visible under extreme magnification, because of the inconsistent pattern of bilinear interpolation on the two sides of the cut (see [RNLL10]);
- *Vertex duplication at seams.* To represent texture seams, the seam vertices must be duplicated in the buffered mesh geometry, directly impacting the GPU memory usage, the per-vertex rendering workload, and also vertex-cache efficiency on the GPU;
- *Diminished texture-cache coherency.* Texture seams cause the pattern of access to texture to be less cache-coherent, affecting the efficiency of the hard-wired texture look-ups.

The magnitude of these drawbacks grows with the number of seams; therefore, by substantially alleviating the fragmentation, we expect to reduce each of them significantly. In Sec. 5.8, we offer an empirical assessment of the improvements obtained by our method, as measured on a large benchmark of real-world examples.

4.2 Overview

We want to improve an existing texture UV-map by pursuing two conflicting goals:

1. Mitigate its fragmentation, i.e. reduce the number of seams;
2. Preserve the original distortion properties, ideally avoiding altogether the need to resample the original texture in as many areas as possible.

In order to remove a given texture seam, the two sides of the cut must be aligned in texture space, which in the general case requires deforming their shape, thus affecting the distortion of the UV-map; moreover, the change propagates to the affected triangles and, in cascade, to the neighboring triangles. This means that our two goals are conflicting, and our algorithm strives to find a balance between them.

Algorithm 1

```
1: procedure GREEDY TEXTURE DEFRAGMENTATION
2:     Identify all candidate merge operations                ▷ Sec. 4.4.1
3:     Score each operation                                   ▷ Sec. 4.4.3
4:     Insert operations in priority queue Q
5:     repeat
6:         Extract top operation S from Q
7:         Perform S                                          ▷ Sec. 4.4.2
8:         if S failed then
9:             Roll back S
10:            Insert S in failed operation set F
11:        else                                               ▷ Sec. 4.4.4
12:            Update score of impacted operations in Q
13:            Move impacted op. from F to Q
14:            Update sorting of Q
15:        end if
16:    until Q is empty
17:    Pack the final atlas and synthesize a new texture      ▷ Sec. 4.4.5
18: end procedure
```

This problem statement leads to a combinatorial explosion of the number of potential solutions that can be considered; finding the global optimum would clearly be impractical. Instead, we resort to a greedy approach which works by iteratively performing a sequence of local *merge operations* (summarized in Figure 4.2 and Algorithm 1).

Each merge operation consists in attempting to fuse two neighboring texture charts into one larger chart, dissolving all seams originally separating them (thus helping towards Goal 1). Due to the need to reduce the explosion of the number potential fusion operations, we disregard partial fusion operations where only a part of the seams are fused. Dissolving a seam in general implies the modification of the UV positions of the involved vertices, which introduces distortions and, potentially, new overlaps. To counter this, we re-optimize the UV-map in a *limited area* around the modified UV positions (Sec. 4.4.2). This optimization step resembles the distortion minimization procedure of standard parametrization approaches, with the difference that we target the shape of the original UV triangles rather than the shape of the 3D mesh-triangles (according to Goal 2 above). If, after the optimization, this merge operation results in overlaps or excessive distortions it is discarded and rolled back.

As in any greedy approach, the order of the attempted operations is crucial, as different orderings can lead to different results. All potential operations are identified and scored according to their predicted outcome towards our two goals, which is estimated with a heuristic (Sec. 4.4.3). Then, they are attempted from most appealing to least appealing, progressively removing the existing seams as shown in Figure 4.3. A successful operation affects the scores of other potential operations, which are updated accordingly (a priority queue is used to quickly identify the next operation to attempt).

Once all merge operations have been attempted, we pack the updated charts in a new texture area. When placing each chart, we can detect triangles that have been only rigidly transformed but *not* deformed, and constrain the chart placement to be *grid-preserving* [RNLL10] w.r.t. the input position of such triangles. In doing so, we can avoid resampling a possibly significant

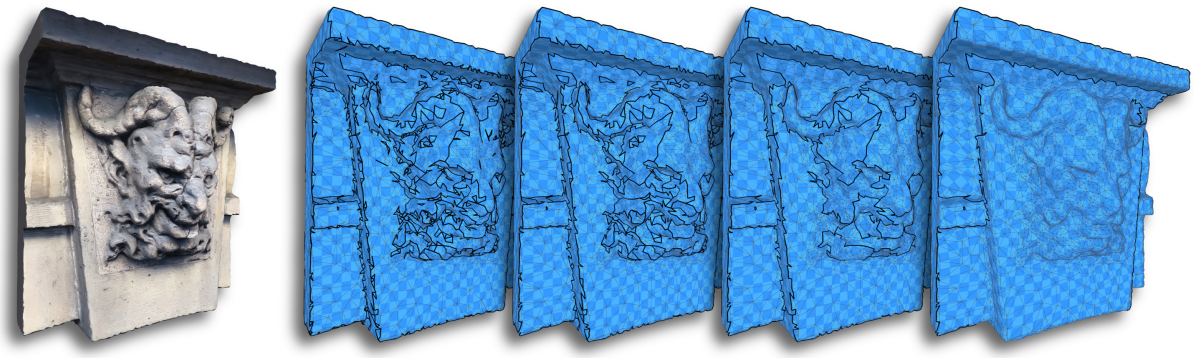


Figure 4.3: Our method produces a simplified cut layout for the existing UV-map by progressively dissolving texture seams.

fraction of the input signal when generating the new texture data.

Lastly, (Sec. 4.4.5) we synthesize a new texture image for the updated UV-map (which is usually much smaller than the original image), by resampling color values only where necessary, and copying pixels verbatim in the other parts.

Before detailing each phase of this algorithm we review relevant previous literature.

4.3 Related Work

Single-patch Mesh Parametrization There is a vast literature on the problem of computing mesh parametrizations, with applications in texture mapping among the most common motivations. A thorough discussion of this topic is beyond the scope of this section, and in the following we discuss only the more closely related issues; we refer the reader to Section 2.2 and the discussion in [FH05, SPR*07, HPS08] for an introduction.

Single patch mesh parametrization seeks the best UV image of a given disk-like discretized surface, and is typically formulated as an energy minimization problem. The energy functional measures the amount of distortion, defined as the discrepancy in shape (angles), area, or both (isometric distortion, as in our case), between the 3D mesh faces and corresponding texture triangles. The distortion is minimized over the possible assignments of UV positions to mesh vertices using numerical methods.

Several approaches have been proposed, differing for example on whether the UV boundary needs to be known a priori [EDD*95, Flo97], whether only shape distortion is minimized [DMA02, LPRM02, SC17], and whether local self-overlaps of charts (also known as fold-overs) are prevented [Tut63]. In some techniques, global self-overlaps of charts, i.e. self-intersecting chart boundaries, are prevented [ZMT05, SS15, JSP17] or automatically fixed [LPRM02, LVS18]. The complexity of the numerical methods employed also varies considerably, ranging form simple least-squares solvers (only the case for conformal methods), to iterative descent algorithms from non-linear optimization theory. Early approaches [HG00, SSGH01] move a single vertex at a time, and exploit multi-resolution structures to accelerate the process. The ARAP algorithm [SA07, LZX*08] uses a local-global approach where UV triangles are first individually aligned to their target shape in a local step, before solving a global Poisson system over UV vertices. Similarly, the SLIM algorithm [RPPSH17] iteratively minimizes a quadratic proxy of the deformation energy. Other works focus on identifying good descent directions to improve the convergence rate of the energy minimization by preconditioning the

gradient descent direction with a positive-definite Hessian approximation [SS15, SPSH*17] or the mesh Laplacian [KGL16].

For our purposes, we can in principle pick any existing single-patch optimization method, and employ it in the re-optmization process that follows a merge operation; we need to adapt it by changing its objective, simply by redefining the minimized functional to minimize the isometric distortion with respect to the original 2D UV triangle rather than the 3D triangle. Among all the possible choices, we choose to formulate our single-patch optimizer after [LZX*08], which matches our requirements. See Section 4.4.2 for detail. Other alternatives would be viable, for example to strike a different tradeoff between computational speed and result quality.

Global Mesh Parametrization Surfaces with a topology which is not disk-like, or presenting intrinsic curvature far from flat, must be cut before computing the flattening with single-patch parametrization techniques. Finding the optimal cut layout is at the core of the global mesh parametrization task, and is known to be NP-Hard [EHP04]; effective heuristics, however, are the subject of intensive research spanning multiple decades.

There is a huge number of automatic methods, approaching the problem for a variety of angles; their enumeration exceeds the scope of this chapter, and the reader is referred to the State of The Art sections of recent papers, such as [PTH*17]. In spite or recent advancements, the search for a fully automatic solution is still an open problem, and the industry practice tends to leave artists partially in control of the process, which is among the motivations for our work.

In general terms, these methods share with ours the need to balance the amount of cutting with the total distortion. Many methods seek this balance implicitly [GGH02, SCOGL02, SH02], while others explicitly combine the number of seams and the parametrization distortion into a single energy functional to be minimized [PTH*17, LKK*18a].

In order to approach this trade-off methodically, one needs first a measure of the two quantities that must be balanced. Several well-principled measures for distortion have been proposed in the context of single-patch mesh parametrization (see above), but measures for the amount of cutting have fewer predecessors. Many global parametrization methods implicitly minimize quantities like number of seam edges, or total length of seams (in either 3D or UV space), which is unsatisfactory as these quantities depend on either meshing resolution, texture resolution, or model scale. The definitions employed in [PTH*17] or [LKK*18a] are carefully designed for efficient numerical minimization, and present similar problems. In this chapter, we adopt the definition of atlas "solidity" introduced in Section 3.4.2 [MPCT20], which is scale and resolution independent.

Signal-Specialized UV Maps In a similar spirit to our work, a long standing idea in parametrization is that the content of the texture should be taken in account when optimizing a UV-map. For example, in [SWB98], an *importance map* is computed from an existing texture, and used to drive the texture coordinate optimization. In [SGSH02], the authors introduce a signal-stretch metric that also takes into account the signal approximation error introduced by the mapping, and [BTB02] presents a relaxation algorithm that evenly distributes frequency content across the parametrized image.

Our method does not experiment with signal specialization, considering each used pixel of the original texture equally important, and always aiming at their 1:1 preservation in the final textures. The two concepts are, however, not mutually exclusive; in the future, our approach

could be augmented with signal specialization, but a trade off will be necessary to combine this with our objective of limiting the texture area to be resampled.

Mesh repairing One of the motivations behind our work is that typical photogrammetry meshes come with inconsistencies hindering most automatic parametrization methods. One alternative to our proposed strategy would consist, then, in addressing these inconsistencies prior to a standard re-parametrization of the mesh. Automatic mesh repairing has a long history in literature [ACK13], but it is not a completely solved problem. In addition, this route would still need to globally parametrize a mesh featuring high geometric complexity and resolution, which is also not a solved problem. Finally, there would be the drawback related to the change in texel density and the need of resampling, as discussed. While mesh repairing can be used in conjunction with our method to improve model quality, it is not a prerequisite and we validate our approach on unmodified real-world models.

Alleviating the effect of seams Our strategy is to strive to reduce the number of seams in photogrammetry models. A class of orthogonal alternative strategies, surveyed in [YLT19], consists instead in attenuating some or all of their many detrimental *effects*, such as the need for texel duplication (e.g. [LH06]) and vertex duplication (e.g. [Yuk17, Tar16]), visual artifacts (e.g. [RNLL10, LFJG17]), or texture packing inefficiency (e.g. [BL08]). Many of these techniques require substantial changes in the downstream application; among the others, [LFJG17] most closely resemble our approach in that an existing UV-map and texture is modified to produce seamless transitions. There are, however, important differences: first, only the problem of visual artifacts is addressed in [LFJG17]; second, the targeted meshes typically feature a much smaller number of seams compared to photogrammetry models, and it is not clear how the method performs on more complex inputs.

Packing of texture charts Packing 2D charts efficiently inside the texture rectangle in an important step of any atlas-based global parametrization method. It is a form of polygon packing, a combinatorial optimization problem in the field of Computational Geometry that can only be addressed by heuristics. In the context of texture mapping, a popular approach is to rasterize the charts and pack them in a coarse 2D grid while greedily minimizing various cost functions such as the height of the packing horizon [LPRM02], or the wasted space left between charts [SWG*03, NS11]. Packing efficiency can be improved by detecting empty areas induced by complex outlines and reduce them by splitting charts, so to obtain more regular and compact boundaries [LVS18]; this approach can be extremely effective, but introduces additional cuts, in direct contrast to our objective.

For our purposes, we can adopt any existing packing algorithm, on the condition that we are able to adapt it by imposing the grid-preserving constraints outlined in Sec. 4.2. We do not claim any advancement on the packing algorithm itself; rather, our strategy is to surpass the packing efficiency of the input UV-map by virtue of feeding fewer and more regular charts to the packing algorithm.

4.4 Phases of the algorithms

Our algorithm works entirely on the UV-layout of the original 3D mesh, which is a 2D triangular mesh. With “vertex” and “edge”, we refer to a 2D vertices and 2D edges in this layout (we will use “mesh-vertex” and “mesh-edges” when we refer to the 3D mesh).

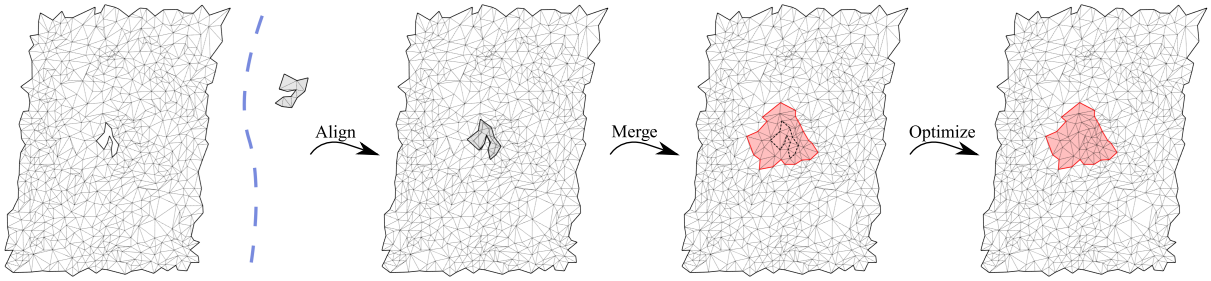


Figure 4.4: Steps required to perform a merge operation. The two charts are first aligned by rigidly moving the smaller one so to maximize the match between the edges to be fused (in this case, its entire boundary). Then, the edges are merged by displacing and fusing vertices, which can introduce distortions and overlaps. Finally, vertex positions are locally re-optimized in area proportional to the displacements (red shade), so that distortions with respect to the original shape, and possibly folds, are removed.

Most mesh-edges correspond to one edge each, but a mesh-edge on a texture seam corresponds to two distinct edges, one on each side of the cut, which we call "linked". A pair of linked edges can be "dissolved" by fusing together the vertices at their endpoints.

The UV-layout is partitioned into a number of connected components, called "charts". Two charts are said to be linked when they are bounded by (one or more) linked edges.

4.4.1 Enumeration of candidate merge operations

To enumerate the candidate operations for our greedy approach we consider, for every pair of linked charts, the merge operation that consists in fusing the two charts into one by dissolving all the linked edges on their boundary. In principle, smaller operations could also be considered, dissolving only some subset of the linked edges shared by the two charts; however, in order to reduce the size of the search space, those are not considered by our greedy strategy. This choice is, again, justified by the combinatorial complexity of the problem being faced.

Note that pairs of linked edges can also be found on the boundary of the same chart. For each connected group of such linked edges, we list an additional "intra-chart" candidate operation, which consists in dissolving them without reducing the number of charts. Therefore, multiple distinct intra-chart operations can be considered for a single chart, one for each set of connected linked edges.

Any merge operation that would result in a chart that is not genus 0 or becoming a closed surface is deemed invalid, and always discarded.

4.4.2 Performing a merge operation

A merge operation S dissolves a certain number of edge pairs, coming from two charts a and b. For intra-chart operations, in the following, we simply consider $b = a$. This operation (see Figure 4.4) is performed in three steps: aligning the chart boundaries, merging and minimizing distortion, and finally checking the consistency and quality of the updated UV-map; the process can ultimately either succeed or fail, according to a number of criteria.

Initial alignment

Operation S has the effect of fusing together a certain number of vertices, by first transforming them into a common 2D location and then topologically unifying them in the mesh structure. These displacements distort all affected triangles, against our second Goal. If the distortion surpasses a given threshold, the operation is rejected and rolled back. To increase the chance of success, we strive to attenuate the needed distortions with a number of countermeasures.

As a preliminary step of every operation, we apply a 2D rigid transformation $\mathbf{M}_S$ to the smallest of the two charts to minimize, in the least-squares sense, the distance between every pair of vertices to be fused. Finding $\mathbf{M}_S$ is a straightforward shape-matching problem that can be efficiently solved [SHR16], unless the operation is intra-chart, in which case the linked edges belong to the same UV-component and we simply set M_S as the identity transformation. Note that the rotation imposes a resampling of the smallest chart in the final image, which is traded for the reduction of the number and length of texture seams.

Once the linked edges are rigidly aligned, we move each pair of vertices to be fused into their average 2D position and merge them. In order to to mitigate the introduced distortions, we then optimize the position of a subset of vertices in the neighborhood of the dissolved seam, as follows.

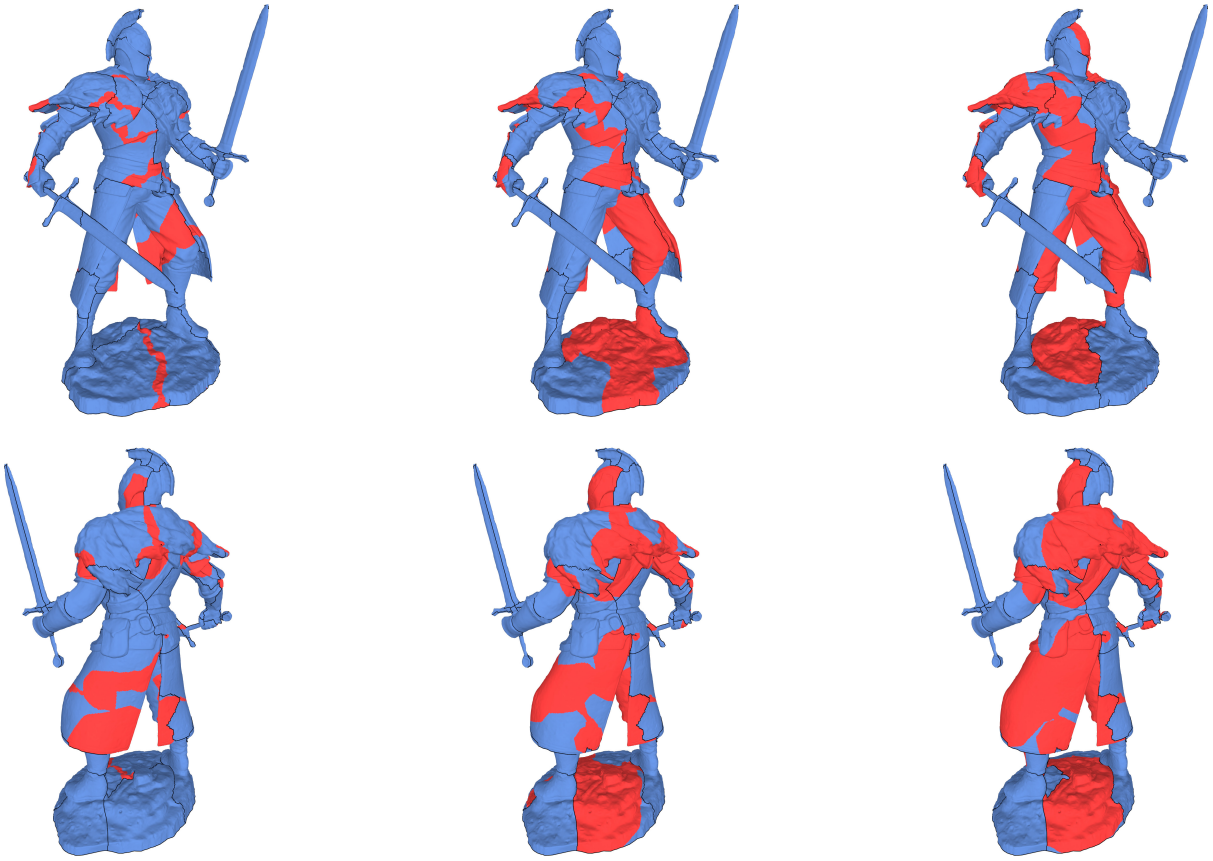
Local UV re-optimization

Intuitively speaking, distortions caused by small displacements of a vertex can be removed by changing only immediately adjacent triangles, while large displacements require optimizing a correspondingly wide band around the vertex. We follow this intuition by identifying all triangles with a 2D distance within α times the displacement of any fused vertex. The α parameter controls the trade-off between the size of the UV optimization area and residual distortion, and thus the likelihood for the operation to succeed and ultimately the removal of seams. In all our experiments we use a value of $\alpha = 5$, which we empirically identified as a good compromise (see Figure 4.5).

Our distortion minimization procedure closely follows the As-Rigid-As-Possible (ARAP) pararametrization approach [LZX*08], which targets isometric distortions. In our case we seek to minimize the isometric distortion with respect to the original UV layout: this simply means that we set the target shape and size of each triangle as the one in the original 2D layout rather than the one in the 3D mesh. Other single patch parametrization techniques could be adapted in a similar way. Iterative methods such as ARAP fit well with our approach because they can exploit the existence of a valid starting configuration.

In the optimization, we freeze (i.e. set as constant) the position of vertices outside the optimization area, while vertices inside the optimization area are free to move during the global phase of ARAP. This greatly reduces the size of the ARAP instance, dramatically speeding up the merge execution.

After the optimization, the value of the minimized energy, as defined in [LZX*08], is measured, and summed over the optimized area. If it surpasses a "local" distortion threshold, the operation is rejected because it is deemed to introduce too much distortion. Additionally, the accumulated distortion over the entire UV-map is also tracked, and operations are also rejected if they exceed a global distortion threshold (see Sec. 4.5 for the values employed in our experiments).



$\alpha = 5$ $\alpha = 10$ $\alpha = 15$

Figure 4.5: The impact of the parameter α on the size of the optimized area (highlighted in red) at the end of the defragmentation loop. In this example, using values larger than 5 increases the area undergoing UV re-optimization, for a small impact on the removal of seams (black lines).

Consistency Checks

Performing a merge can infringe consistency constraints in various ways.

A merge S can introduce *global* UV overlaps when, after the transformation M_S is applied, the boundaries of the two charts a and b collide (disregarding the edges that are fused by the operation). This condition is detected early, before UV re-optimization takes place.

Another occurrence of global UV overlaps can be produced by the UV re-optimization, when the boundary of the combined chart, inside the optimized area, is made to intersect. Similarly, the re-optimization can introduce *local* UV overlaps (triangle folds), in spite of these configurations being penalized by the minimized energy. These conditions are detected after UV re-optimization.

Note that we reject operations that introduce *new* overlaps, but tolerate the ones that were already present in the input map, which are detected and labelled in pre-processing for this purpose. Our scenario justifies this choice. Even if the final UV-map fixed a pre-existing overlap condition, there would be no distinct color data to fill the no-longer-overlapping regions, and the final texture resampling would just replicate the artifacts of the input texture.

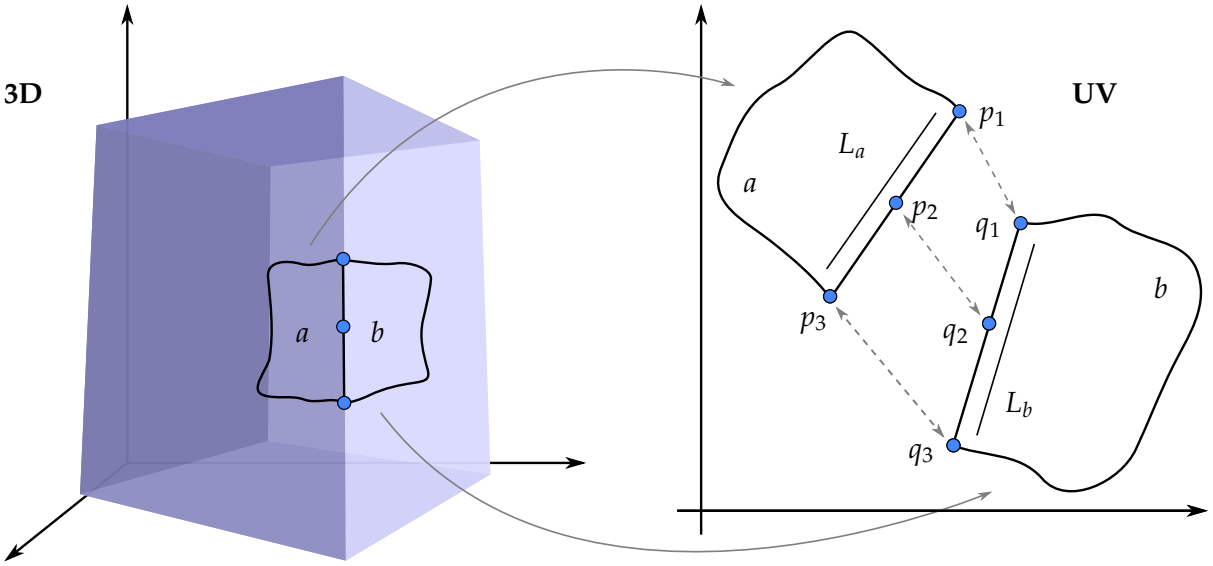


Figure 4.6: Quantities used to estimate the appeal score of a merge operation involving charts a and b.

4.4.3 Ranking of the candidate operations

Attempting merges, and in particular the UV re-optimization stage, is the most time consuming part of our pipeline. The global efficiency of our algorithm depends on the ability to identify sensible operations to be attempted first, and avoid spending times on operations that will fail and need to be reverted. In addition, the success or failure of an operation can depend on the operations that previously affected the same charts. For all these reasons, we devise a method to quickly evaluate an "appeal" score based on the geometric attributes of the charts involved (Figure 4.6), with higher scores assigned to operations estimated as more likely to succeed and more beneficial towards our two conflicting goals.

In order to reduce the fragmentation (Goal 1), we want the perimeters of the affected charts to be reduced. Therefore, we want to prioritize operations that dissolve a large fraction of the boundary of either chart. Specifically, an operation S, fusing edges in a with a total length L_a and edges in b with a total length L_b, has a potential benefit estimated as

$$\text{Benefit}(S) = \max\left(\frac{L_a}{\text{Perimeter}(a)}, \frac{L_b}{\text{Perimeter}(b)}\right). \tag{4.1}$$

The maximal value of 1 occurs for operations which place one chart completely inside a hole of the other, which, indeed, we want to attempt early in the greedy process, as they are likely to succeed, and simplify boundaries which otherwise risk causing global overlaps later.

Performing an operation S implies fusing a certain number of vertex pairs. The farther the vertices are after being rigidly aligned, the more they will have to be displaced, and the more severe distortions will likely be introduced by the operation, even after being minimized by the UV re-optimization. Therefore, we assign to S an estimated cost, computed as the average residual distance:

$$\text{Cost}(S) = \frac{1}{|V_S|} \cdot \sum_{(p,q)\in V_S} \|p - \mathbf{M}_S(q)\|_2, \tag{4.2}$$

where V_S is the set of vertex pairs to be fused, and $\mathbf{M}_S$ is the rigid transformation matrix associated to the operation S. Note that at this stage the transformation M_S is only used to

evaluate the matching error, although we cache it to avoid recomputing it later when the actual operation is executed.

Additionally, we observe that operations where either chart is small are in general computationally less expensive to attempt, and less likely to fail. Therefore, we favor operation S with a bonus given by

$$\text{SizeBonus}(S) = 1/\min(\text{Area}(a), \text{Area}(b)) \tag{4.3}$$

Finally, an operation S that has already failed $k_S > 0$ times is likely (although not guaranteed) to fail again when re-attempted. Therefore, we introduce a back-off penalty factor to make it increasingly less appealing:

$$\text{BackOff}(S) = 2^{-k_S}. \tag{4.4}$$

The final appeal score of an operation S is then given by

$$\text{Appeal}(S) = \frac{\text{Benefit}(S)}{\text{Cost}(S)} \cdot \text{SizeBonus}(S) \cdot \text{BackOff}(S) \tag{4.5}$$

4.4.4 Updating candidate operations

If an operation S fails, it is removed from queue of potential operations Q, and placed in a set of "failed operations" F.

If it succeeds, then we need to update all the candidate operations in Q involving each of the affected charts a and b (which will now affect the unified chart resulting from the execution of S), and also to reevaluate their appeal and update the ranking in Q.

In this case, we also reinsert any failed operations affecting a or b, moving them from F to Q (although with a diminished appeal score, due to the back-off factor). There are a variety of reasons that can make a previously failed operation succeed after a chart has been updated. One occurring fairly easily is that is that global overlaps become less frequent when chart boundaries are simplified by previous operations, e.g. by filling holes.

4.4.5 Atlas Repacking

When all merge operations have been attempted, the resulting charts must be packed to obtain the final texture layout.

The space of 2D transformations that can be applied to the original texture coordinates without requiring a resampling of the input only includes rotations by multiples of 90-degrees and translations by an integer number of pixels, and their horizontal and vertical reflections. These transformations, which have been referred to as "grid-preserving" in previous literature [RNLL10], can be exploited in conjunction with our defragmentation procedure to avoid resampling possibly large portions of the new texture atlas and match the input texture reconstruction exactly when rendering the output model (Figure 4.7). Namely, we can identify within each chart the maximal set of rigidly transformed UV-triangles that share the same transformation matrix, and constrain the packing algorithm to be grid-preserving with respect to those triangles.

We begin by analyzing the final charts to determine if they should be constrained by the packing algorithm. In each chart, we detect the triangles that can be rigidly transformed back into their input position, and record the corresponding rotation angle. Let θ be the angle

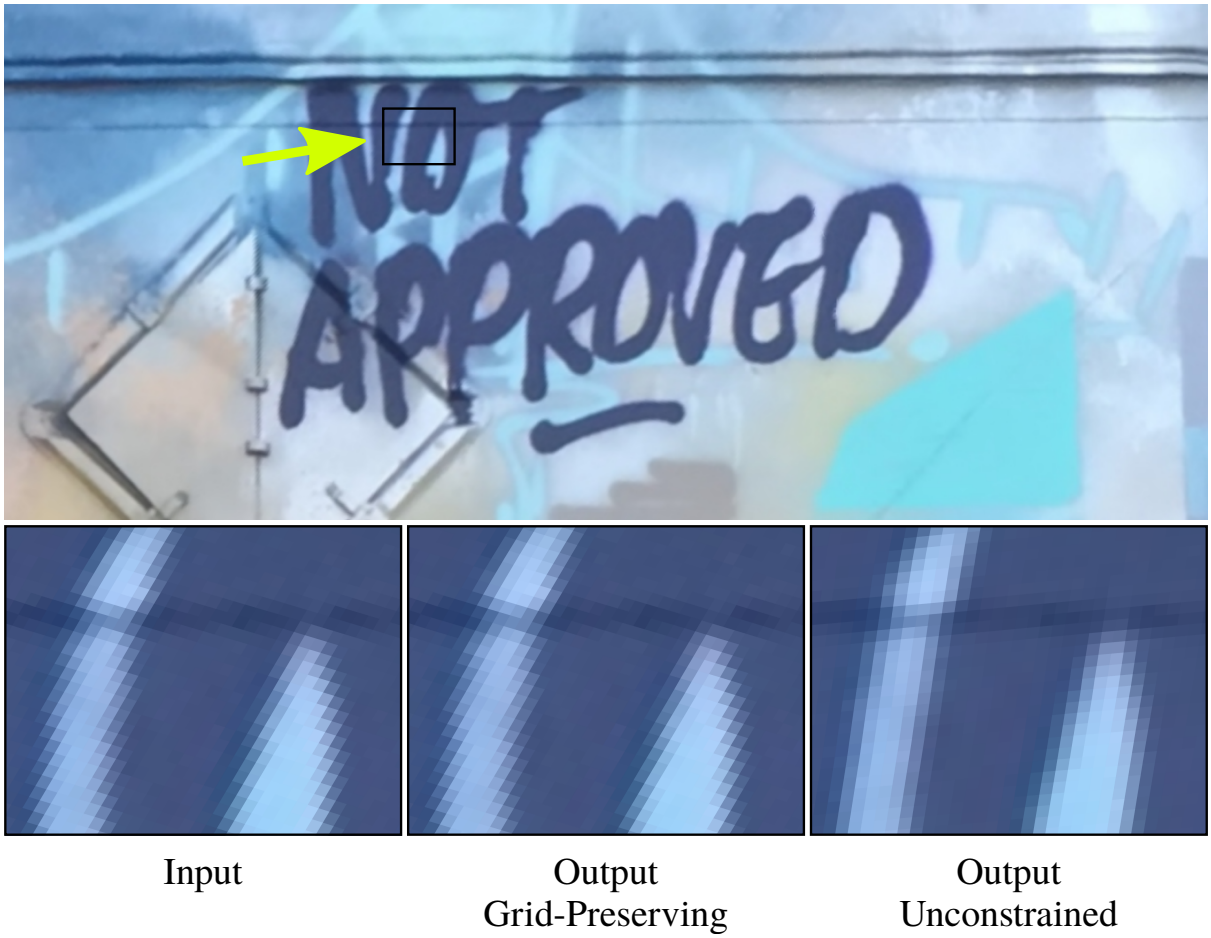


Figure 4.7: Exact correspondence between the input and output texels can be achieved by constraining a rigidly-transformed chart sub-region to be grid-preserving. First closeup: a rendering of the input. Second closeup: a rendering of the output with grid-preserving packing. Third close-up: when the packing is unconstrained, the texture is resampled.

shared by the subset of triangles maximizing the chart area (we have chosen to maximize 3D area, hence visibility during rendering, rather than 2D area). If this area is larger than a fixed percentage of the chart (we used 5%), then we label all triangles associated to θ as "preserved", and the chart as constrained. This means that we will only allow the packing algorithm to rotate this chart by $k(\pi/2) - \theta$. Finally, after packing, we enforce the grid-preserving constraint of all the preserved triangles by applying a sub-pixel correction offset to each chart, ensuring that the final positions of the relevant vertices is indeed an integer translation of the rotated input. As highlighted in Figure 4.8, the impact of such grid-preserving packing is heavily affected by the input fragmentation, the chart size distribution and the overall efficacy of the defragmentation step.

Our packing algorithm, following [LPRM02, SWG*03], is based on rasterizations on a coarse grid. We approximate the region occupied by a chart with the grid cells it partially or completely cover (accounting for an extra "safety border" of texels, needed to avoid bleeding).

We track the packing area and available space using multiple horizons [NS11]; for a given packing direction, we track both the upper horizon, and an inner horizon that maps the largest available gap below the upper horizon. Using multiple horizons allows placement of smaller charts between larger ones, which is particularly useful in presence of large charts (which our

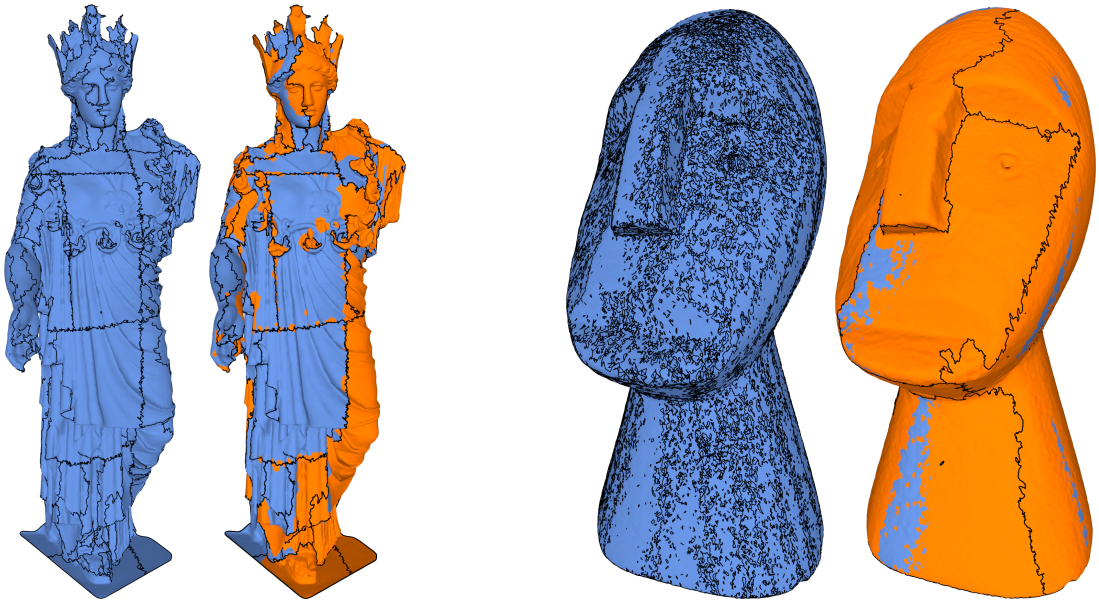


Figure 4.8: Two examples of texture defragmentation, with the resampled areas highlighted in orange. As expected, the resampled area increases as the models undergo a larger number of successful merge operations (right).

merging algorithm is designed to produce). Chart placement is performed in a greedy fashion, minimizing a cost function that penalizes wasteful placements. If the chart is placed using the inner horizon, we minimize the wasted space as [LPRM02, SWG*03], that is, the empty grid area left between a chart's bottom horizon and the current horizon profile. Otherwise, we place the chart in the position that minimize the horizon increase. The sequence of chart placements is usually determined by sorting charts from largest to smallest, as this provides reasonably good results without needing to enumerate all the possible permutations of the sequence. If we have fewer than 100 charts, we also attempt a number of randomized sequences, which we have found to help improve packing efficiency.

Finally, we detect the bounding square of the used texture space and we trim the texture to this size, which is usually smaller that the original size (see Results in Sec. 4.5). The final texture is simply constructed by either copying texels for the original texture verbatim, inside the triangles labeled as "preserved", and interpolating pixels in the other areas. In our prototype, we color all unused texels by extending and blending the colors of used texels, using push-pull algorithm in [GGSC96], in order to contain the bleeding artifacts at any MIP-map level.

4.4.6 Extension to Multiple Texture Sheets

Photogrammetry models frequently encode the surface color into multiple texture sheets, assigning different part of the model to different sheet. This can be done, for example, to circumvent the hard-coded upper bound on texture resolution, thus affording a larger number of texels for the model. Since our objective is to produce a UV-map with the same mapping resolution of the input, we adapted the packing algorithm to handle this occurrence. During the packing, we target a texture size equal to the maximal resolution of the sheet. when a chart cannot be accomodated in the current sheet, we initialize a new empty sheet.

4.5 Results

Implementation We have implemented our algorithm in C++ using the VCG Library and Eigen. The source code of this reference implementation is publicly available.

Experiments We evaluate the performance and robustness of our method over the dataset we presented in Chapter 3, that reflects the textured models produced by the current generation of photo-reconstruction tools. The dataset consists of 568 models of varying resolution from several different sources and constructed with different photogrammetry tools. As previously discussed, these 3D models exhibit complex geometry, dense and uneven meshing tessellation, inconsistencies such as non-manifoldness, non-coherent face orientation, degenerate faces, and so on; textures are high-resolution and in many cases consist of multiple image files.

We batch process the dataset, successfully producing a result from every input. Examples of processed models are reported in Figure 4.16.

We use the following settings: we allow alignment errors up to twice the seam length, and forbid operations that do not reduce the perimeter by 20% for at least one of the charts; the ARAP energy thresholds are 0.5 for the local optimization area, and 0.025 for the energy of the entire atlas (the latter compensates for the relaxed matching threshold used).

Processing time The average processing time is around 5 minutes on a PC equipped with an Intel Core i7 8750H and 32 GBs of RAM.

Defragmentation efficacy We empirically assess the efficacy by statistically examining a number of quantitative measurements and practical impacts in the produced output.

We measure the *Atlas Solidity* [MPCT20] value of each model before and after our defragmentation, and plot the distributions in Figure 4.9. Atlas Solidity is an aggregate measure characterizing the overall amount of seams in an atlas, defined as the inverse of the ratio between the total length of texture seams and the length of the minimal perimeter of a 2D shape encircling the same texture area, with a Solidity of 1 being the best and theoretical maximum. Other measures comprise the reduction in the total number of charts, and UV border length (Figure 4.10).

A direct practical benefit of defragmenting the UV-map is the reduction of vertex replications that are necessary to encode seams, which impacts the memory required to store and render the model (e.g. the GPU buffers). Distributions of the vertex replication coefficients, i.e. the ratios of the number of 2D vertices to 3D vertices, before and after optimizing the models with our method is reported in Figure 4.11. Another direct effect is the reduction of the overall number of texture samples while preserving the input texel density and mapping resolution, reported in Figure 4.12.

The different measures concur in showing a clear reduction of the fragmentation and of the associated practical problems, both on average and in each instance. For example, chart count decreases up to 99% and 72% on average, vertex duplication decreases up to 43% and 6.8% on average, and atlas solidity increases by up to 20 times and 21% on average.

The one exception is overall texture size. While the average reduction is 23.3%, there are a few cases where the the output texture is actually larger than the input texture. In our analysis, two factors contribute to this result. First, we always pack our textures using a padding size

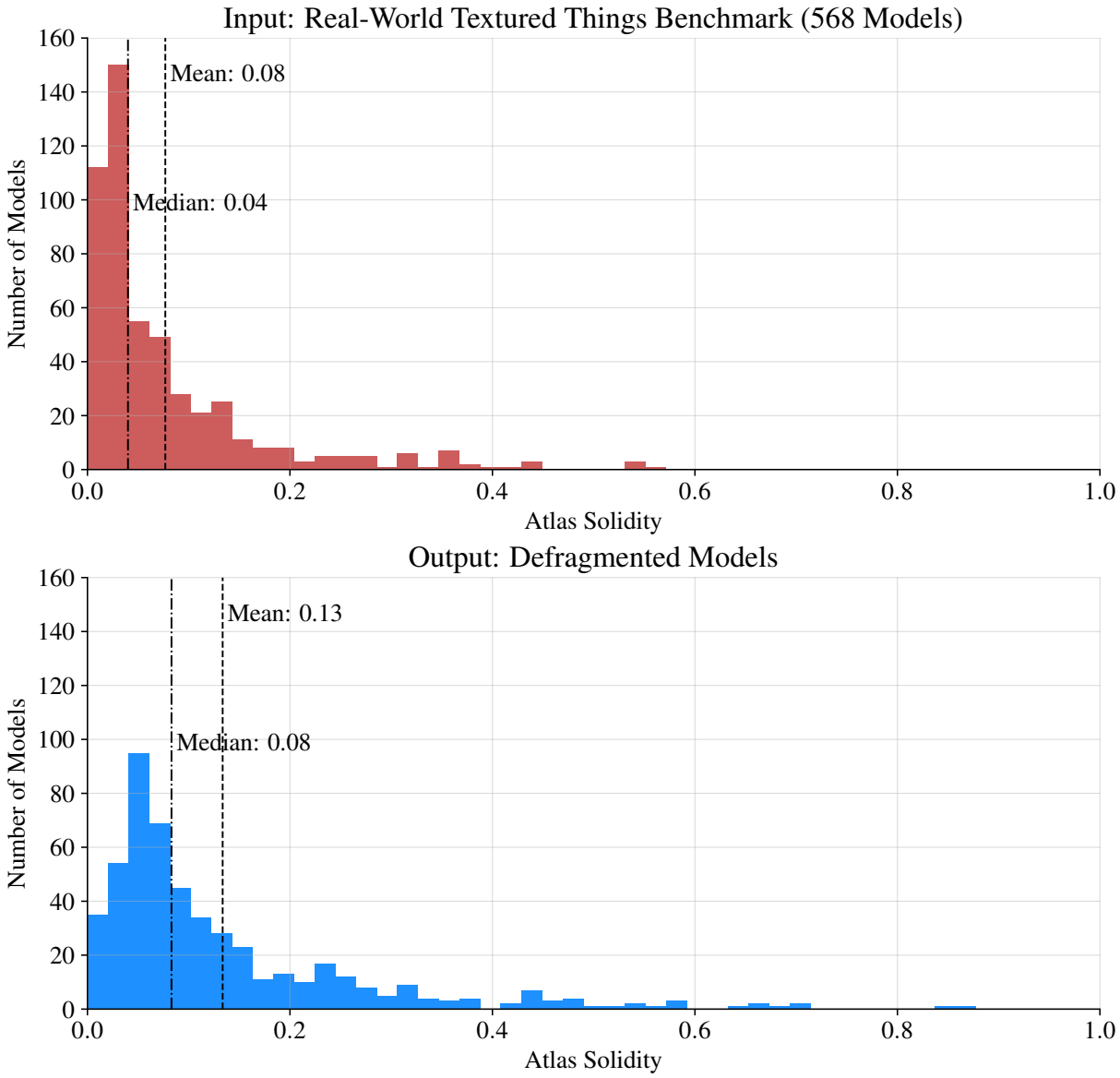


Figure 4.9: Distribution of Atlas Solidity values before and after processing the benchmark.

of 4 texels (meaning charts are at least 8 texels apart), which ensures correct filtering when MIP-mapping is used up to level two, while the several of the original images accommodate for no padding at all. Second, the larger produced charts can be harder to pack; if deemed necessary, this could be countered with strategies such as [LVS18, LFY*19], which reintroduce straight cuts for the sake of packing efficiency. We omitted 7 models from this analysis, as they are anomalous in the context of photogrammetric reconstructions, in that they contain either instanced objects or faces with random texture coordinates; these configurations mislead our system into trying to unnecessarily allocate texture space for these areas.

Introduced distortions As stated, our second objective is to keeping the mapping distortion faithful to the input UV-map low, for the reasons explained in Sec. 4.1.1. We assess the fulfillment of this goal by measuring the ARAP energy (which, as discussed, measures the loss of isometry with respect to the original UV layout,) as done in Figure 4.13.

A closely connected objective, as stated, is to avoid any resampling in areas of the original

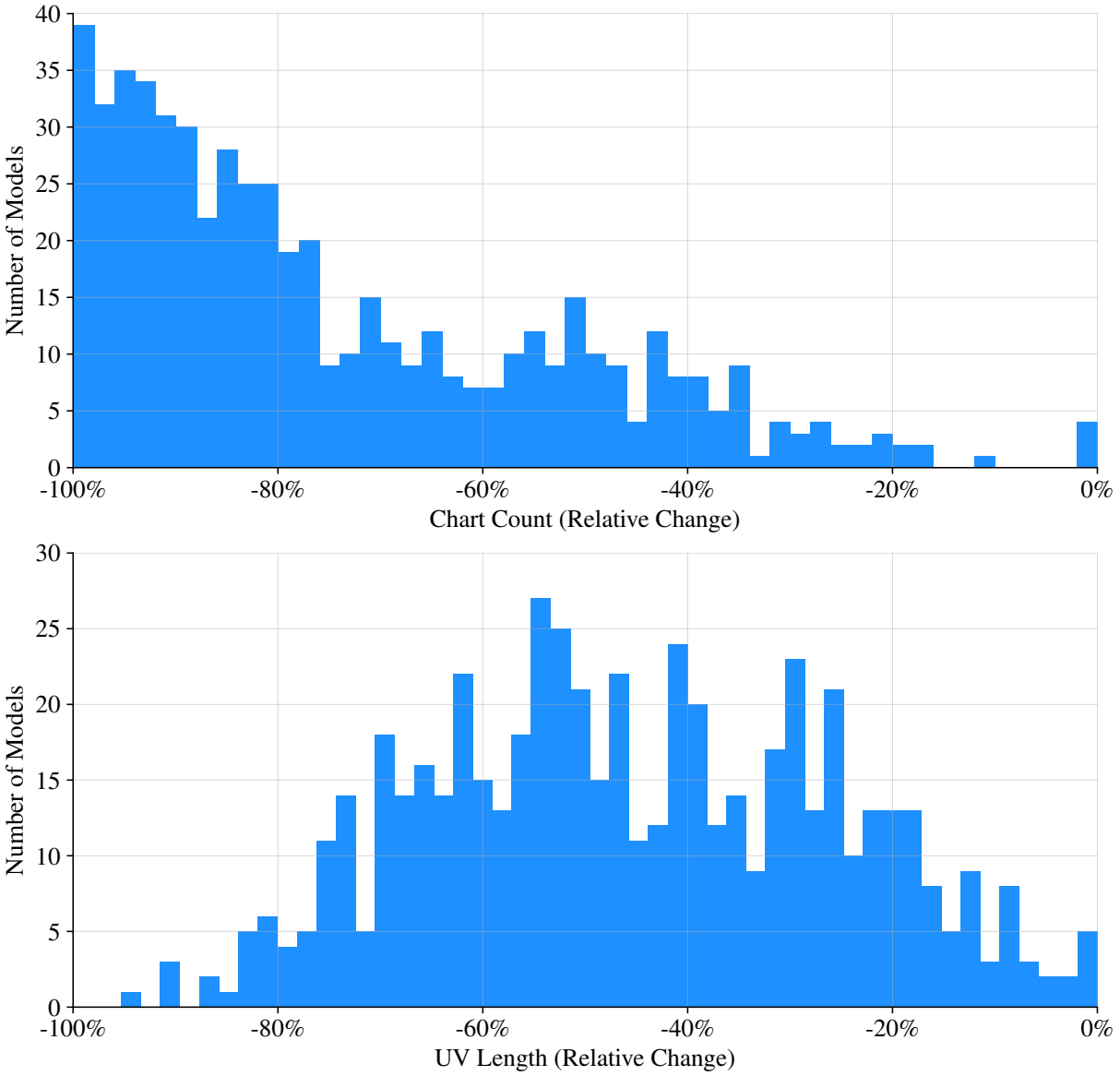


Figure 4.10: Aggregate chart count and UV border length reductions achieved by our method.

model, which is one main characteristic setting our method apart from any predecessor (to the best of our knowledge). We report statistics of the fulfillment of this objective in Figure 4.14. On average, 46.6% of the surface area was *not* resampled.

4.5.1 Ablation study

To validate our design choices, we have performed an ablation study measuring the individual impacts of the local optimization phase (for various values of the expansion parameter α), the greedy ordering of the operations (against a random order), and of each component of the Appeal function (4.5) used to order them.

The results, reported in Table 4.1, confirm that each stage of our algorithm yields a significant benefit, and that our choice of parameters strike good trade-off.

In particular: larger values of α, expanding the UV re-optimized area, improve the results in terms of atlas Solidity, but increase the need of texture resampling and the running time

		Average relative change over Dataset					Average relative change over Baseline					
	Time (s)	Resampled Area	Vertex Replication	Texture Size	Solidity	# Charts	Time (s)	Resampled Area	Vertex Replication	Texture Size	Solidity	# Charts
Baseline	322	+53.4 %	−6.8 %	−23.3 %	+121.0 %	−72.3 %						
$\alpha = 2.5$	295	+39.0 %	−6.1 %	−23.6 %	+89.5 %	−68.5 %	−9.5 %	−26.0 %	+0.7 %	+0.0 %	−10.9 %	+25.4 %
$\alpha = 10$	600	+64.6 %	−7.0 %	−23.2 %	+133.2 %	−73.5 %	+130.1 %	+27.8 %	−0.2 %	+0.4 %	+4.3 %	−7.1 %
NoRetry	319	+53.0 %	−6.7 %	−23.4 %	+117.2 %	−71.6 %	−0.1 %	−0.8 %	+0.1 %	−0.1 %	−1.0 %	+6.1 %
NoBC	406	+53.6 %	−6.7 %	−23.4 %	+117.7 %	−72.0 %	+28.2 %	+1.5 %	+0.1 %	+0.4 %	−1.3 %	+2.8 %
NoSB	363	+53.8 %	−6.8 %	−23.6 %	+122.6 %	−72.2 %	+14.7 %	+1.0 %	−0.1 %	−0.2 %	+0.8 %	+1.3 %
Random	607	+55.9 %	−6.5 %	−23.3 %	+115.5 %	−70.7 %	+83.7 %	+7.0 %	+0.3 %	+0.8 %	−0.4 %	+37.7 %

Table 4.1: Results of the ablation study comparing baseline results of our approach against variations of our algorithm reducing and increasing the UV optimization area expansion α (compared to our baseline value of 5), and altering the greedy sequence of merge operations. *NoRetry*: do not reattempt failed operations; *NoBC*: drop the Benefit/Cost ratio from the Appeal function (4.5), *NoSB*: drop the SizeBonus term from Appeal function (4.5); *Random*: rank the operations randomly.

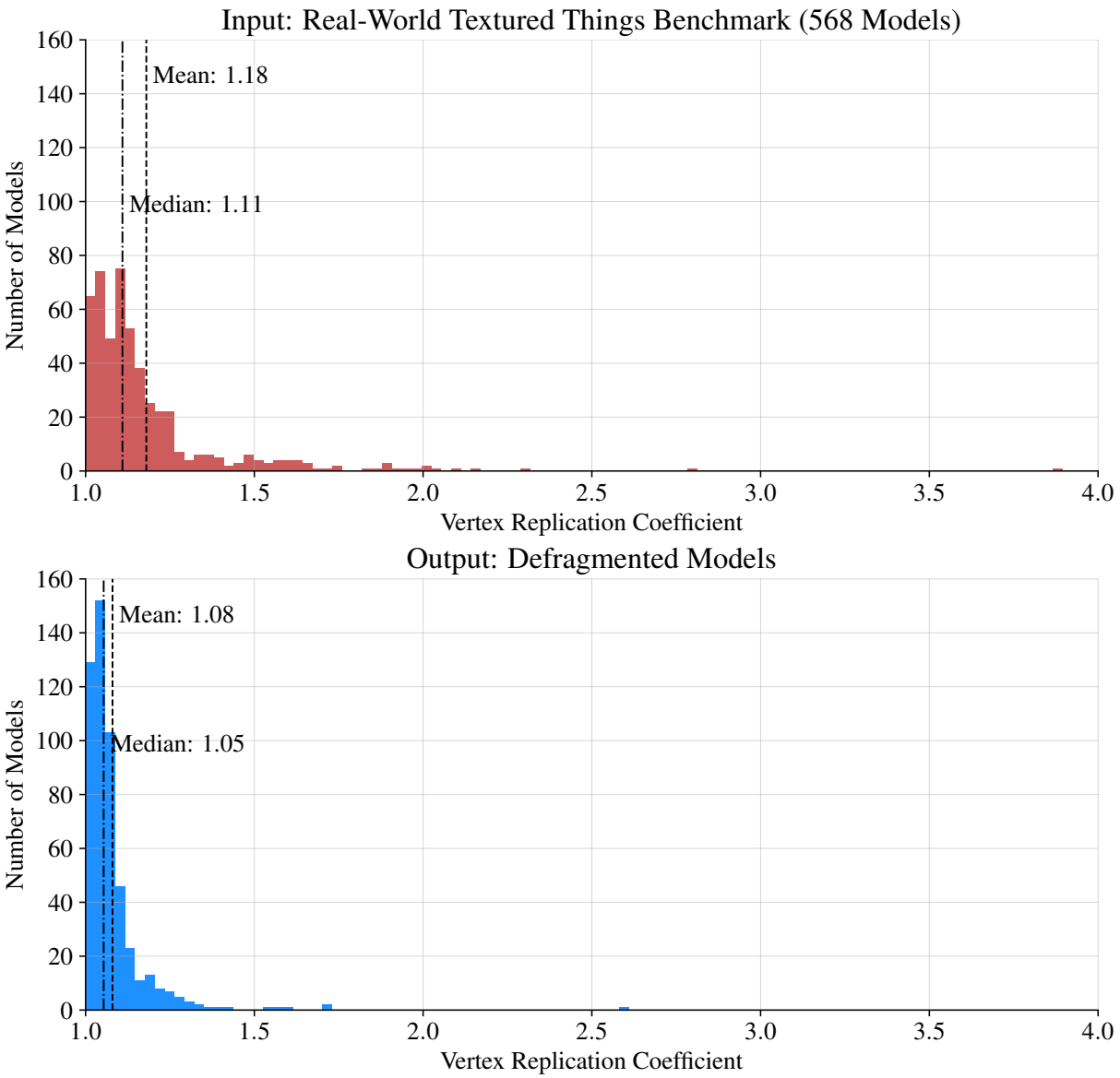


Figure 4.11: Vertex replication factors before and after processing the dataset.

considerably; the opposite is true for smaller values. The ordering of the operations is proved important, as adopting a random order produces consistently worse results and drastically increases the running time; also, dropping any component of the Appeal function clearly degrades the performance, with the possible exception of discarding SizeBonus, which leads to mixed results according to the other metrics. Finally, a designated experiment confirms that reattempting previously failed operations is unquestionably beneficial.

4.5.2 Comparison against computing an entirely new UV-map

In support of the premise of our work, we compare against traditional approaches which ignore the existing UV-map and rebuild it from scratch; an obstacle is that, as mentioned, previous experiments [MPCT20] already establish that a direct application of off-the-shelf software suites or existing implementations of automatic UV-mappers fails to produce satisfactory results in a majority of cases (due to various inconsistencies on the input models or their excessive resolution). We consider this as an argument in favor of our approach.

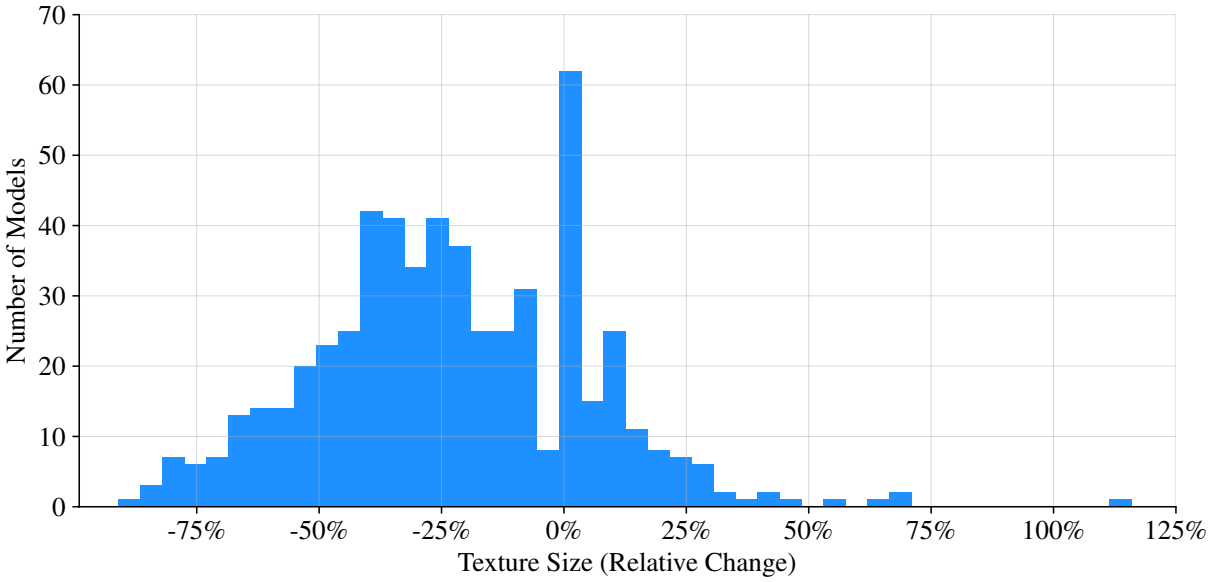


Figure 4.12: Efficacy of our method in reducing the size of texture images required to render the models.

A central point of our strategy is that we strive to preserve the shape of the existing UV-map triangles, rather than minimizing the distortions of the 3D-to-2D mapping, which can make a significant difference when the texture samples distribution and UV-map resolution over the object surface are not uniform (Figure 4.15, top). To validate this choice, we visually compare our method against the results obtained by applying a state-of-the-art UV-optimization algorithm to the original UV-map, reducing its distortion but without affecting the cut layout of the mesh. The produced UV-map is representative of the results that can be obtained by recomputing a UV-map ex-novo, at least away from cuts. For this experiment, we minimize the symmetric Dirichlet energy [SS15] (minimal for isometric mappings) with Composite Majorization [SPSH*17], using scaffolding [JSP17] to ensure the new texture atlas is overlap-free.

A visual comparison is offered in Figure 4.15. Our approach is visibly better at preserving highly detailed areas and high frequencies of the input texture image.

4.6 Conclusions

In this Chapter, we introduced a new automatic approach designed to attenuate texture fragmentation of an existing texture model, which is a defect systematically arising with photogrammetry-based 3D reconstruction techniques. The method was proved to be effective at reducing the fragmentation and the consequent drawbacks, making the models more usable by downstream applications. Compared to alternative approaches, including a global re-parametrization of the model, the main benefits are reliability and fidelity to the input image quality, thanks to the ability to preserve the original texel distribution and UV-map resolution. A re-sampling of the texture is avoided on approximately half of the object surface on average. The method was extensively tested and evaluated by batch processing the dataset we introduced in Chapter 3.

Limitations Our approach is not well suited for situations where there is a vast redundancy of input photographs, and where super-resolution approaches such as [LTH*17] could be

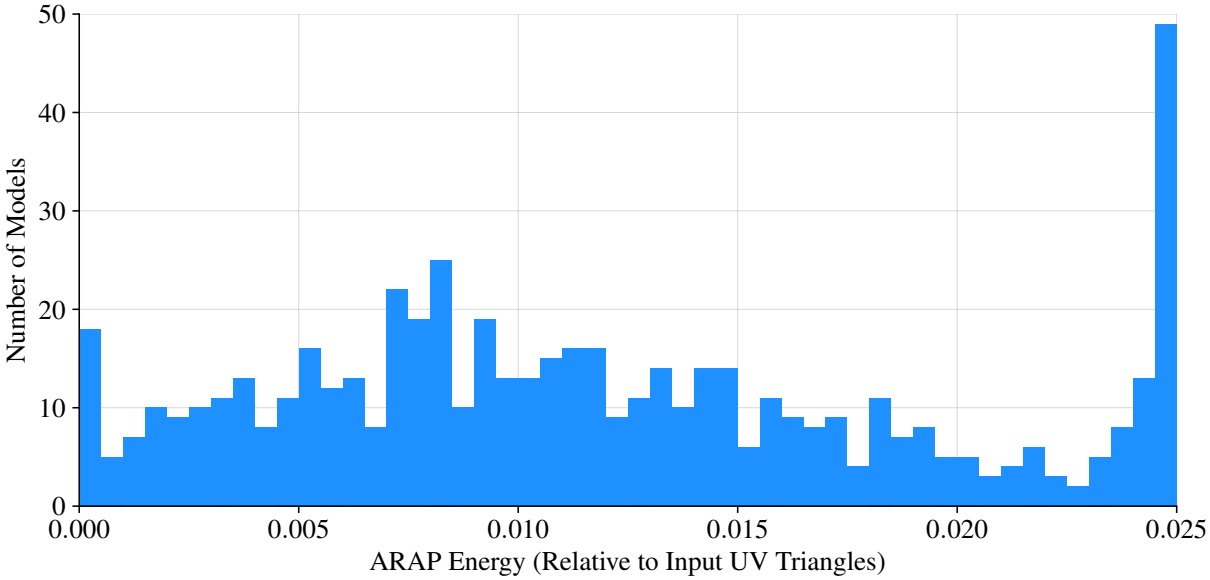


Figure 4.13: Distribution of ARAP energies after optimizing the texture atlases. Note the very low global cut-off at 0.025, and also how in most cases the energy values remain well below this threshold.

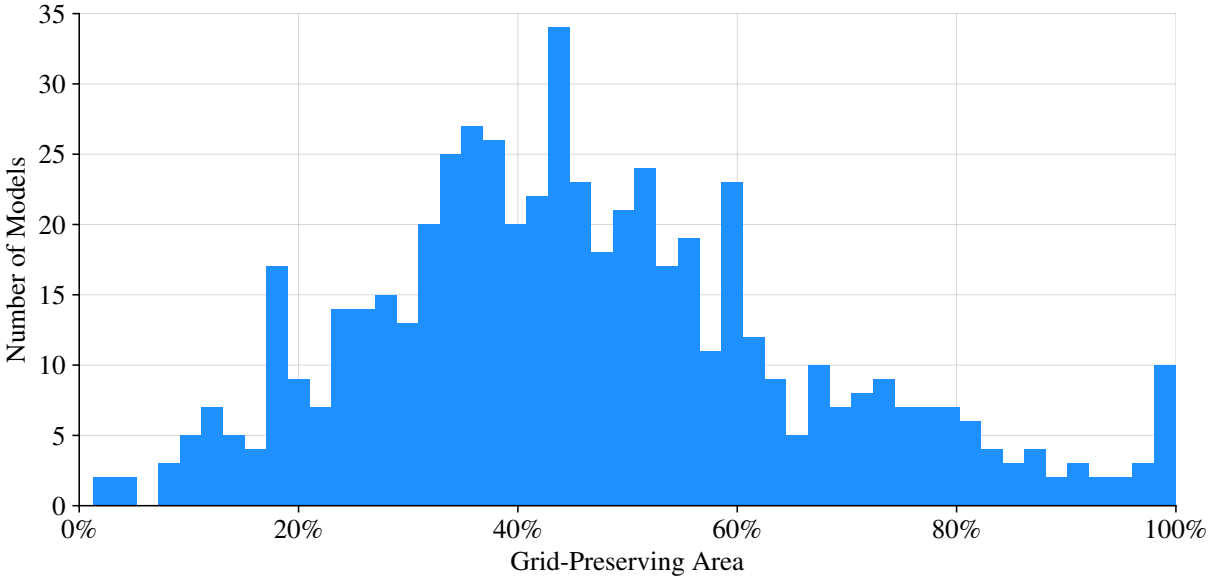


Figure 4.14: Distribution of fractional object areas (computed in 3D space) that do not require resampling the input texture and where the signal reconstruction exactly matches the input data.

exploited to increase the texture quality. Another scenario is when the texture resolution must also be downsized, which can be the case for example when very high-resolution photographs are available. In both cases, the benefit of inheriting the input texel distribution is diminished (although the reliability and robustness of the proposed method would still be an advantage).

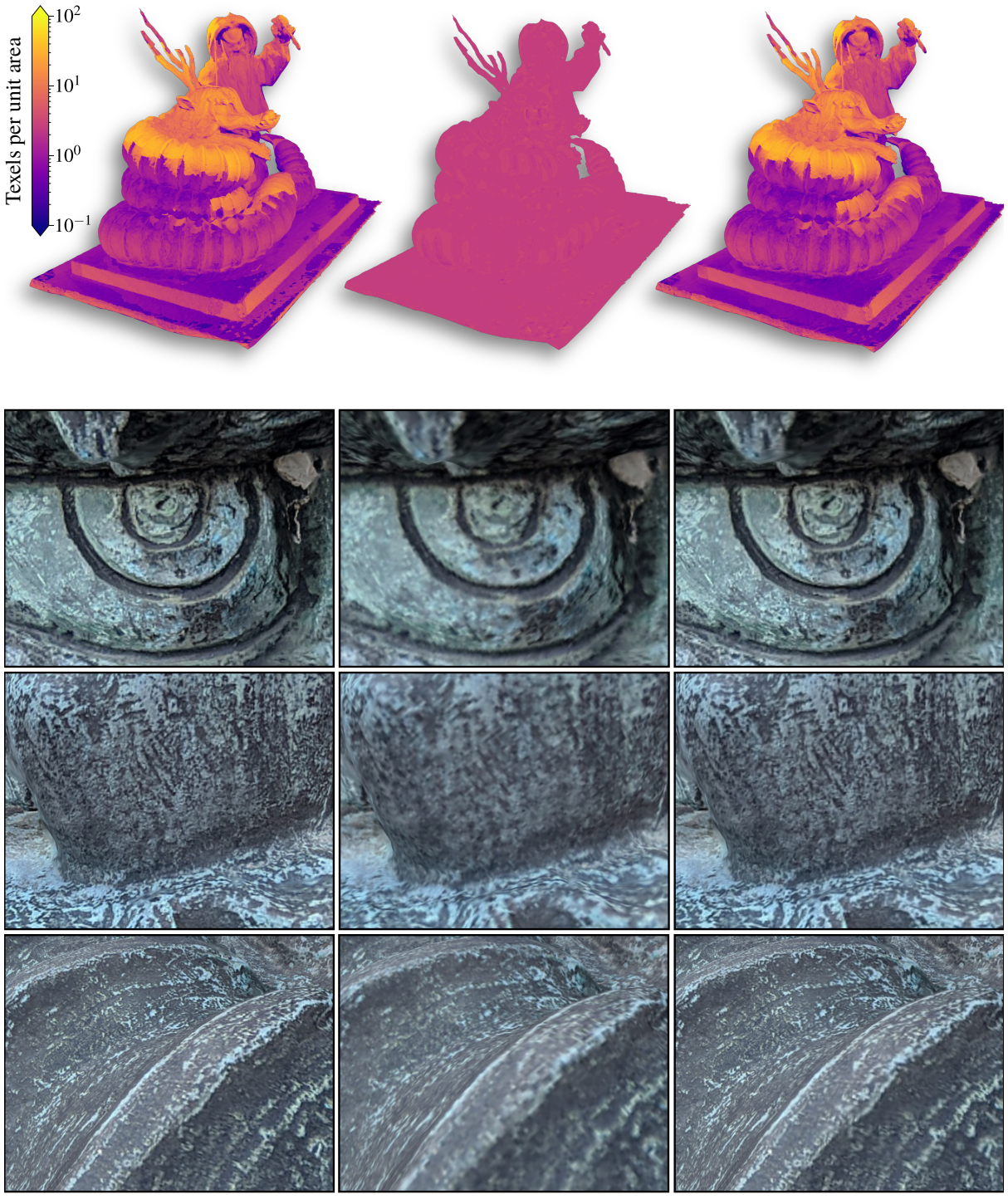


Figure 4.15: Comparison between generating a UV-map *ex-novo* and our method on a model exhibiting uneven texel distribution across the surface. Top: traditional parametrization approaches minimize distortion w.r.t. the 3D surface and are unable to preserve the input texel distribution. Bottom: close-ups of areas mapped at high-resolution reveal how this results in loss of high-frequency detail, while our approach produces a noticeably sharper texture.



Figure 4.16: A gallery of results obtained with our algorithm.

Chapter 5

Texture inpainting

We devise a technique designed to remove the texturing artifacts that are typical of 3D models representing real-world objects, acquired by photogrammetric techniques. Our technique leverages the recent advancements in inpainting of natural color images, adapting them to the specific context. A neural network, modified and trained for our purposes, replaces the texture areas containing the defects, substituting them with new plausible patches of texels, reconstructed from the surrounding surface texture. We train and apply the network model on locally reparametrized texture patches, so to provide an input that simplifies the learning process, because it avoids any texture seams, unused texture areas, background, depth jumps, and so on. We automatically extract appropriate training data from real-world datasets. We show two applications of the resulting method: one, as a fully automatic tool, addressing all problems that can be detected by analyzing the UV-map of the input model; and another, as an interactive semi-automatic tool, presented to the user as a 3D "fixing" brush that has the effect of removing artifacts from any zone the users paints on. We demonstrate our method on a variety of real-world inputs.

5.1 Introduction

Local texturing defects, i.e. problems in the texture image associated with the 3D mesh, are particularly frequent in photo-reconstructed 3D models and originate from a variety of issues and challenges in the acquisition process outlined in Section 2.1.

In this chapter, we present a technique to correct these texture inconsistencies *a posteriori*, i.e., after the model acquisition has been completed. By working on the reconstructed models, our method is oblivious to the acquisition process and can be used to counter a wide range of potential problems with texture images, regardless of their origin. The basic idea is that the defective sub-areas of the texture image are discarded and *inpainted* anew by seamlessly filling the gap, guided by the surrounding context. While the inpainted regions may drift from the real-world object represented by the model (which we do not want to assume to be still available), we want it to look plausible and realistic. Inpainting methods are attractive because the vast majority of the texture data is valid and contains reliable information that can be used to interpolate the texture content and cover localized artifacts.

We are inspired in this by the recent advances of modern image inpainting frameworks, where a similar operation is successfully performed on natural images (photographs) by leveraging Machine Learning tools. However, it is not trivial to extend these results to the case of diffuse color textures of 3D models. At a first glance, this can be attempted using either of two

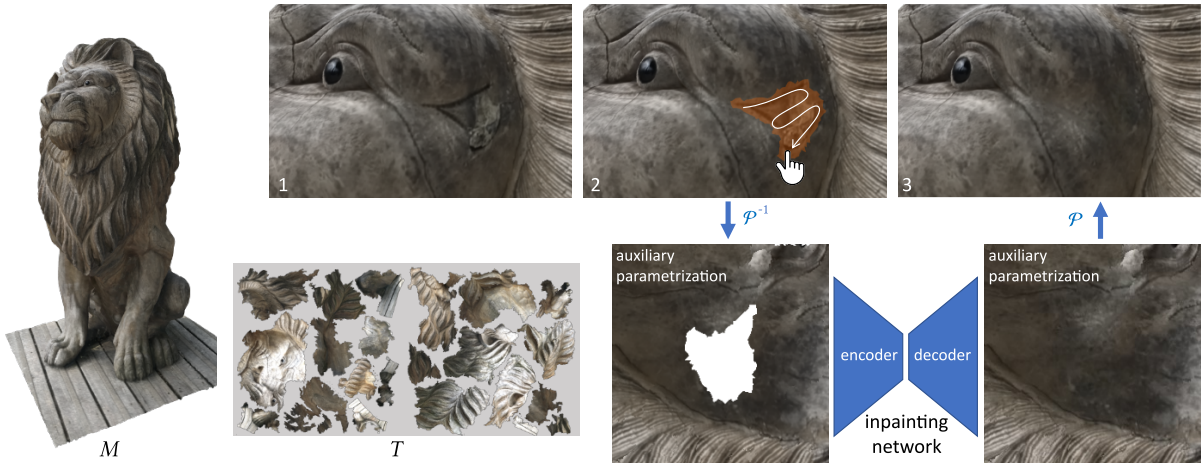


Figure 5.1: An example of our inpainting operation issued by an interactive stroke of a "fixing" brush performed over a photogrammetric 3D model M of a lion (2), originally featuring a texture defect (1). The system constructs a local parametrization $\mathcal{P}$ for a region surrounding the stroke, fills its domain with texture data from the model texture T, performs an inpainting operation on the region affected by the brush (big arrow), and transfers the the result back into the texture image, fixing the texture defect (3).



Figure 5.2: Example of our inpainting operation performed in screen space, over a 3D rendering, introducing artifacts.

strategies.

Inpainting in texture space. A simple way is to feed the texture image directly to the same inpainting procedure used for photographs. However, this does not work reliably, because of the semantics of texture image, which, in the context of photogrammetric models, invariably comes in the form of an atlas of texture islands (or charts), separated by gaps of unused space. Being unaware of the atlas discontinuities and texture distortions, any inpainting method applied to the global texture would produce visible seam artifacts. This is a critical issue, because photogrammetric models present heavily fragmented UV-maps [MPCT20], with intricate and dense texture seams.

Inpainting in image space. Another alternative that comes to mind is to apply image inpainting to the renderings, covering the artifacts that originate from the defective texture.

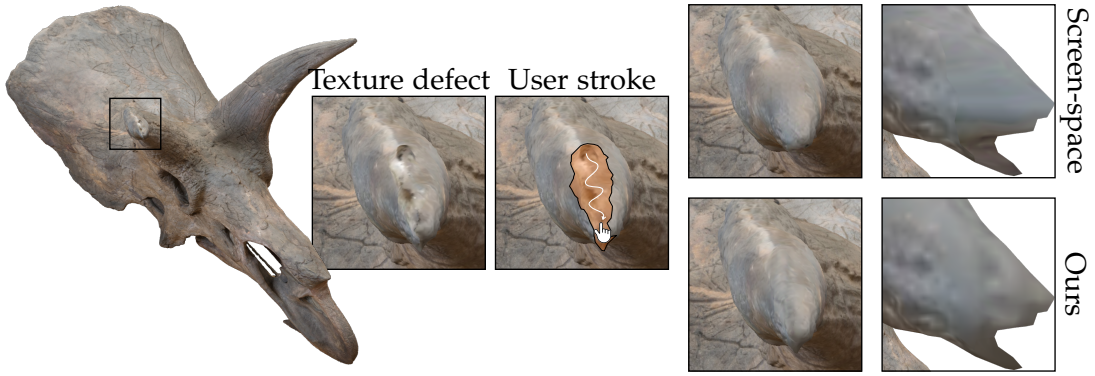


Figure 5.3: Comparison of screen space and our local texture-space inpainting using multiple views. Changing the observer's position reveals stretching artifacts near the object silhouette induced by the narrow viewing angle and the resulting perspective distortion affecting the projection of the inpainted screen-space pixels over the object surface.

The inpainted regions can then be re-baked into the texture by re-projection. This avoids the problem with texture seams and textures gaps, and is attractive because the texture inpainting procedure is fed synthetic images that are similar to the natural images that they are intended (and trained) for. Unfortunately, the inpainting method is unaware of the depth-jumps, and of the distinction between foreground and background, which needlessly complicates the semantics of the signal to be reconstructed. For example, the surrounding regions that happen to be close in image space can and will bleed into the inpainting area (see Figure 5.2). Moreover, screen-space inpainting is not robust to perspective distortion (see Figure 5.3) These problems is particularly severe in areas with high surface curvature or non-trivial topologies, where view projections are bound to come with occlusions.

5.1.1 Our solution

Our solution is to apply inpainting to an ad-hoc parametric domain that is defined on-the-fly for each inpainting operation. Given a region of the mesh M that presents a texturing defect, we create a local "auxiliary" parametrization of the affected faces and a region around them. We then synthesize a disposable texture for this region, inpaint over the defected region, and transfer inpainted texels back into the original texture (possibly, across seams).

This approach avoids all pitfalls from either the strategies above (depth-jumps, background, gaps in texture space, perspective distortions, texture seams); we can reach much lower texture distortions thanks to the localized nature of the parametrization. In short, this strategy presents to the inpainting algorithm a texture image with almost no distortion (either of the UV-map or view projection): every part of the texture to be inpainted, and its context that is used to guide the inpainting, appears as if seen orthogonally to the surface everywhere, which improves the learnability of the inpainting process.

5.1.2 Contributions

We introduce a robust specialized method to apply texture inpainting operations to 3D models.

Frameworks. We identify, implement, and test two distinct frameworks for using our new inpainting in practice (in Section 5.4) that can be used in isolation or in conjunction. One, which is fully automatic, targets texturing defects stemming from shortcomings inherited by

original UV-map, after a new better UV-map is available; the other framework is interactive, and allows a user to target any texturing defects by painting over them with a "fixing" brush.

Both frameworks are modular with respect to the specific inpainting method employed, and can seamlessly integrate and benefit from future advancements in the field. The idea is to improve the performance and applicability of any existing image- or texture-completion methods by running (and possibly training) them on a domain more friendly to inpainting, given by a novel specialized auxiliary parametrization, and by automatizing their application.

Specialized local parametrization. Given an area of the mesh to be inpainted, we need to produce a parametrization over the plane for it and its surrounding region, minimizing the mapping distortion; to reduce the distortion, we allow for carefully placed cuts to be introduced. This task is only partly similar to the extremely well-studied problem of producing an almost-isometric (global) parametrization for a given mesh. The differences in objectives are discussed in Section 5.5.1, and our construction procedure is described in Section 5.5.2.

Training data. The above process is also the basis for producing a dataset of images of a new category, which we use to train a network specialized for our task. As observed, this dataset is not the same as either natural images or textures, in spite of originating entirely from natural images. It can be described as pictures of the surface of natural objects transformed so that the image is (almost) orthogonal to the surface in every position. We describe a procedure to produce the training data in Section 5.6.2, and make the resulting data available for future investigations.

Automatic identification of UV-mapping induced defects. We present an algorithm in Section 5.7, to identify the areas requiring inpainting, considerably reducing the workload required by users to benefit from texture inpainting techniques.

To facilitate reproducibility and adoption of our work, we provide a reference implementation, completed with the inpainting network, the training data, the interactive texture inpainting system, and the automatic command-line inpainting tool.

5.2 Related work

Our technique has predecessors in automatic or semi-automatic systems designed for alleviating texture defects of models, often explicitly targeting photogrammetric models. Some, like [HDGN17], work on the original pictures used for the reconstruction, which, in contrast, we do not need to assume to be available; a few methods [TSPD16] apply inpainting to in *image space*, while others [FDGM18, SKCA20] (including ones integrated in commercial products [Ado22]) apply inpainting in *texture space*, incurring in the respective limitations we outlined in Section 5.1.

Several steps of our technique require facing challenges that have been studied in isolation.

5.2.1 Parametrizing photogrammetric models

Techniques for reconstructing a 3D model from a set of digital photos, based on Photogrammetry or Structure-From-Motion approaches [HZ04, SCD*06, RSN*14] are commonly used to create 3D textured assets for many different areas such as cultural heritage (for documentation, analysis, and preservation), the entertainment industry, digital art, personal leisure,

and many others. These models have often very complex UV-map layouts and many issues as documented, for example in [MPCT20]; while some recent approaches try improving these assets working on the parametrization [MCT21], there remains the problem that the content of the photo-reconstructed object often presents small artifacts that need to be corrected.

5.2.2 Image Inpainting

Image inpainting [BSCB00] refers to the process of reconstructing specific portions of an image while maintaining consistency. Inpainting is a low-level image processing task with many different applications, such as photo editing, object removal, and restoration of corrupted parts. There is a vast literature on this subject. Traditional approaches use either diffusion-based image-synthesis approaches [BSCB00, BBC*01] or reuse portion/patches of the current image [EF01, BSFG09]. Some methods work on the structure of the scene to get some kind of guidance by manually specified points of interest [DCOY03], or by perspective distortion [PSK06]. Other attempts to automatically recover these structures by exploiting tensor voting techniques [JT03], searching in tile space [KKDK12], or by recovering perspective planar surfaces [HKAK14]. Older patch-based approaches shared the common limitation that the synthesized portion can come only from the existing input image. A first method that exploited a very large collection of general images was proposed in [HE07] and others followed the same concept on more specialized problems like faces [MPK09, DDZ11].

In recent years, inpainting increasingly relies on Convolutional Neural Networks (CNNs) [XXC12, KSSH14, RXYS15]. Earlier network models suffered from limitations on image resolution and size of missing parts; fully convolutional models such as Context Encoders [PKD*16] and Generative Adversarial Networks [ISSI17] eased these limitations.

More recent inpainting CNN architectures [GYH21, HZW*21] have been proven to work well with images drawn from specific classes (e.g. faces, cars) for which approximating the data distribution is somewhat easier, but the inpainting of generic images remains difficult. A recent review on this subject can provide further information [EAAMA20].

5.2.3 Geometric deep learning and textured meshes

Geometric deep learning is a rapidly evolving field, and recently novel network architectures have been proposed to operate on 3D models enriched with texture data. TextureNet [HZY*19] introduces a 3D convolution operator based on local surface patches [BMRB16, MBBV15] that leverages a 4-RoSy direction field [JTPSH15, HZN*18] to parametrize a local neighborhood around each vertex and extract consistent feature descriptors on 3D surfaces of arbitrary topology. PFCNN [YLP*20] introduces a similar convolution operator that operates on aligned tangent spaces to match the behavior of 2D convolution over images. These methods apply convolutions to feature maps resampled to local grids, but still output vertex descriptors in the end; conversely, we need to *synthesize* high-resolution signals, which remains an interesting future direction and generalization of such patch-based approaches.

TM-Net [GWY*21] is a generative model that has been recently proposed to synthesize 3D textured objects. The model produces objects assembled from "parts" modeled as deformed squares tessellated at low resolution, with the texture data synthesized by a separate network and to a fixed UV layout. Contrary to this, we need to be able to handle arbitrary geometry and UV atlas layouts.

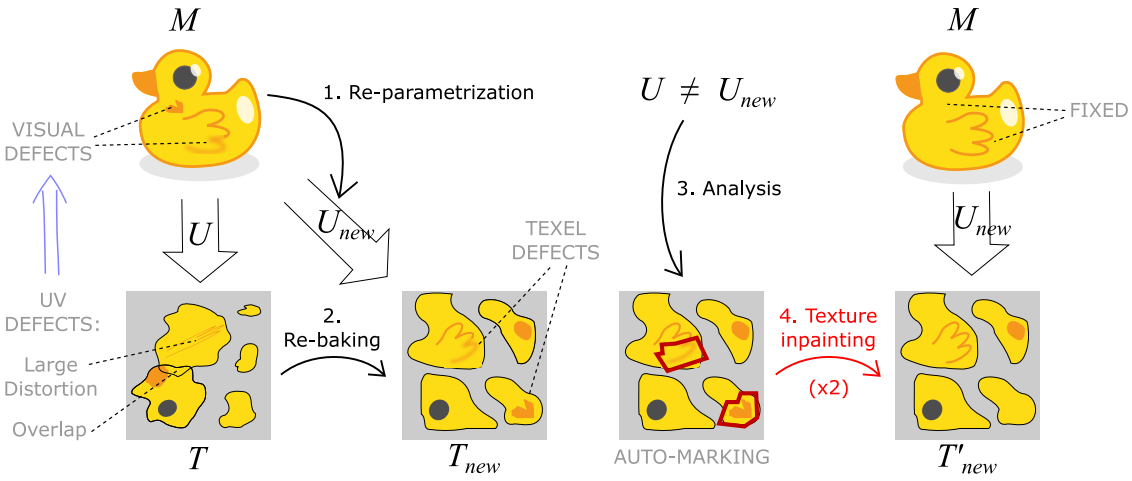


Figure 5.4: The fully automatic pipeline, showing the context of use of our texture inpainting technique (step 4). See Text.

5.2.4 Local parametrization

Constructing a global parametrization for a given surface is a long-standing open problem [FH05], in spite of recent progress such as, among others, [LKK*18b, ZLG*18, PTH*17, LYNF18]. In a sense, the problem is insurmountable because an ideal parametrization that is almost isometric and free from seams does not exist for a generic surface, and automatic construction techniques can only choose between different trade-offs, primarily between cuts (loss of continuity) and isometric distortions (loss of isometry). Where appropriate, this awareness has led to the development of solutions that, instead of seeking a static parametrization, construct, on the fly, a new local parametrization over a small surface area surrounding the region of interest. The limited size of the surface lowers the total amount of curvature that has to be flattened, reducing the need for cuts or distortions. This approach has been followed successfully for tasks such as surface analysis [PCCS11], 3D model inspection [PTW13], texturing implicit surfaces [SGW06], and many others. Our work also falls in this category, and we employ this approach for inpainting over texturing defects. To this end, we adapt our parametrization construction algorithm to the requirements of this particular scenario, which are detailed in Section 5.5.1.

5.3 Overview

Our method works on a photo-reconstructed 3D triangular high-resolution irregular mesh M, enriched with a texture image T connected to M via a UV-map U (i.e., a per-vertex assignment of texture coordinates). The input texture T is obtained by photographs shot at the real object as part of the acquisition process, and presents localized defects of various nature.

5.3.1 Texture inpainting operations

We correct the texture with a succession of individual *texture inpainting operations*, each targeting individual defective areas.

An inpainting operation acts on a subset of the mesh, i.e., it takes as an additional input a tagging of the mesh faces whose texture content is to be discarded and replaced by new content automatically generated according to the surrounding context. We perform this operation by

first parameterizing the affected area and its neighborhood into a square region S (Section 5.5), that serves as the domain of the inpainting operation (Section 5.6); specifically, we resample the texture T over S (Section 5.6.1), perform the inpainting operation on the resulting image, and resample the newly created texels over T.

5.3.2 Sources of texture defects

We need to distinguish between two classes of sources of texture defects.

Defects stemming from the parametrization. A class of texturing defects originates from issues in the UV-map U, resulting in the inadequate distribution of texels of T over M. Such issues include excessive angle or area distortions, and overlaps, either caused by flipped triangles, or (self-)overlaps of texture charts; all these cases are common in photogrammetric models, with our analysis in Chapter 3 reporting them in 83% of the datasets. This class also includes the case of minor color discontinuities appearing at texture seams, e.g. due to photograph misalignments. Another instance is resolution jumps in the texture domain, which are frequent because the resolution of a textured area is often linked to the pixel density of the source photograph, which depends on factors such as shooting distance or camera resolution.

Defects stored as incorrect colors in T. This class includes defects originating from all other reconstruction errors, including uneven lighting between photos, reflections, and other view-dependent lighting effects which end up stored in the diffuse map (for example, by failures of de-shading algorithms), wrong photo re-projection due to camera registration errors, unaccounted photographic radial distortions, inaccuracies in shape reconstructions, etc. Re-projection errors are particularly disruptive in areas where the texture source switches from one photograph to another (not necessarily in correspondence with a UV seam), and appear as either discontinuity artifacts or ghosting (when photographs are blended). Pixels from source photos can also be re-projected over parts M which are very far from the correct one, due to misaligned depth jumps.

Our texture inpainting operations can be used on both classes of defects, but within different frameworks (see next Section).

5.4 Frameworks of use

We propose two different frameworks to issue the sequence of texture inpainting operations (Section 5.4): a fully automatic one, targeting defects stemming from the parametrization, and an interactive one in which defects are corrected on demand, using an interactive 3D brush tool. The two frameworks can be used individually or in cascade. We discuss other potential frameworks of use in the conclusions.

5.4.1 Fully automatic pipeline

Problems stemming from the UV-map (Section 5.3.2) cannot be fixed by working on T alone because any reconstructed signal still cannot be represented using the same UV-map U. It is necessary to first produce a better UV-map U_{new} for M, either from scratch, discarding and substituting the original U [LZX*08], or incrementally, by modifying U locally [MCT21] (step 1, in Figure 5.4). The creation of U_{new} is completely orthogonal to our method, and any existing technique can be adopted. In our experiments, we use [JSP17].

Given the artifact-free U_{new}, we re-sample the original texture image T into a new texture T_{new} (step 2 in Figure 5.4). In doing so, the original shortcomings of U are transferred into the texels of T_{new}, where they can be fixed.

By analyzing and comparing U and U_{new}, we automatically mark all faces of M that presented sampling problems in U, but are no longer an issue in U_{new} (step 3 in Figure 5.4). We detail our algorithm to do so in Section 5.7. The marked faces are then divided into connected components, and inpainting operations are issued on each component separately, in cascade (step 4 in Figure 5.4).

5.4.2 Interactive mode

In the interactive framework, we apply the inpainting method on demand, allowing a user to simply paint the areas of M perceived as defective with a "fixing" brush, similar to the ones used commonly in 3D painting or sculpting interfaces (see Figure 5.1). From the user's perspective, each stroke of the brush triggers an inpainting operation over the covered triangles, and the results are shown back immediately, at the end of the stroke, by updating the texture.

The interactive mode sidesteps the need to automatically identify defects, and the difficulty of even defining what constitutes a defect in T (which is not trivial, in absence of a ground truth). On the other hand, it requires inpainting operations to be performed in interactive times, a feature that our method provides.

5.5 Auxiliary parametrization

Given a cluster of connected faces $M' \subset M$ that needs to be inpainted, our first task is to flatten it and its surrounding region into a squared 2D area S, which will serve as the input domain for the inpaint operation. We do so by building a local "auxiliary" parametrization function $\mathcal{P} : B \subset \mathbb{R}^2 \rightarrow \mathbb{R}^3$ that maps a region of the plane B into some subset of M encapsulating M' (see Figure 5.5). Area $A \subset B$ must be mapped into $\mathcal{P}(A) = M'$, and it will be filled by the inpainting procedure (red in the Figure 5.5); the rest of $B \setminus A$ must be mapped into a surrounding mesh area M', and provides the context for the inpainting procedure (green in Figure 5.5).

Note that in our notation the parametrization function goes from the manifold to the plane, rather than the inverse (which is more customary), because in our case $\mathcal{P}$ can be non-injective and therefore non-invertible.

The problem of constructing this parametrization (that is, determining A, B, the function $\mathcal{P}$, and its image on the mesh $\mathcal{P}(B)$) is related to the extensively studied field of surface parametrization, but there are differences in our objectives, as follows.

5.5.1 Objectives for the auxiliary parametrization

Low distortions. Like in most other texture-mapping related contexts, we want to minimize isometric distortions, that is, we want $\mathcal{P}$ to be as length preserving as possible. Our tolerance for distortions is particularly low in the A domain, where they would have the effect to deform the inpainted image while transferring it back to the final texture. Distortions in the $B \setminus A$ part of the domain are more tolerable, but must also be kept low, as they may hinder the learning process during both training and the application of the inpainting by disrupting the context.

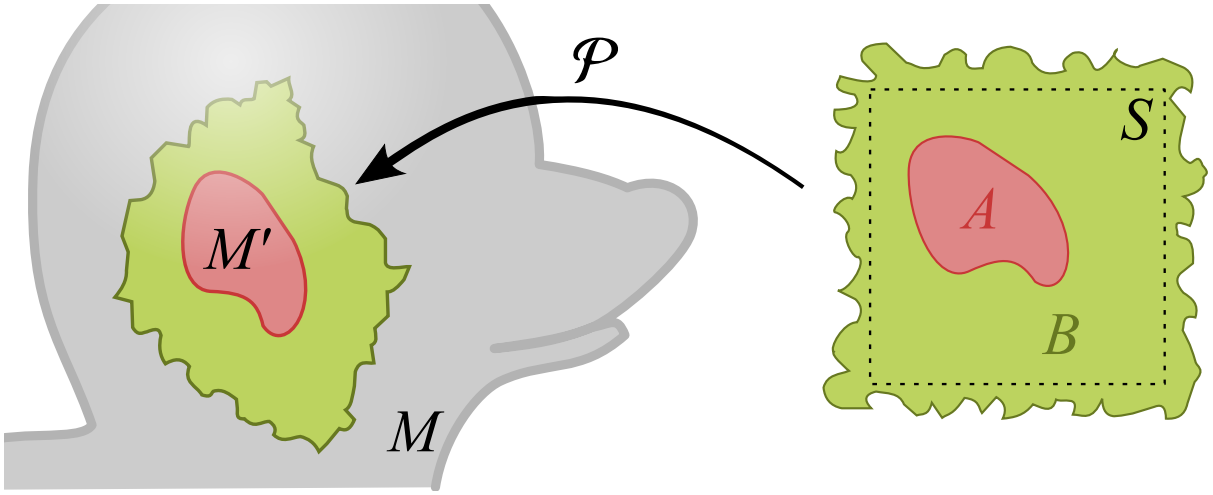


Figure 5.5: Our auxiliary parametrization $\mathcal{P}$ mapping a planar region B, which encapsulates a square S, onto the surface M (left). A central sub-region $A \subset B$ is mapped into $\mathcal{P}(A) = M'$, corresponding to the region to be inpainted (red), while the rest of $B \setminus A$ is mapped into an area of the mesh that will provide the context for the inpainting operation (green).

One motivation for using a localized parametrization (in place of the provided parametrization U_{new}) is that a smaller mesh can be parametrized with lower distortions.

Cuts scarcity. We require the region A to be free from texture seams, that is, to be mapped with continuity over $\mathcal{P}(A) = M'$ (the mesh region to inpaint). On the contrary, on the rest of the domain $B \setminus A$, we can introduce texture cuts, whenever they are needed to lower the distortion.

Undefined boundaries. Differently from the typical parametrization scenario, we do not know beforehand which part of the mesh $\mathcal{P}(B) \subset M$ must be parametrized. We require it to encapsulate M', that is, $\mathcal{P}(B) \supset M' = \mathcal{P}(A)$; we also need the parametric domain B to fully encapsulate a large *squared* region S around A; finally, we want A to be far from the boundaries of S, so that sufficient context is given to the inpainting procedure. Within these requirements, we are free to define $\mathcal{P}(B)$ and B in the best way to fulfill the other objectives.

Non-strict injectivity. In contrast to typical scenarios, we do not need P to be fully injective. Instead, we tolerate that different regions inside B (but not inside A) map into the same part of M. This will cause some pictorial detail over M to be repeated inside the context region of S. Allowing for repetitions is necessary toward the goal of filling the entire S in the parametric domain, in cases where M does not provide enough surface around $P(A)$ due to extreme intrinsic curvature or open boundaries. From an inpainting perspective, pattern repetitions in the context area are preferable to either leaving empty gaps inside S, or distorting the details more (see Figure 5.6).

5.5.2 Construction of auxiliary parametrization

To fulfill our objectives, we adopt the following steps.

1. We construct A and parametrization $\mathcal{P}_A : A \to M'$.

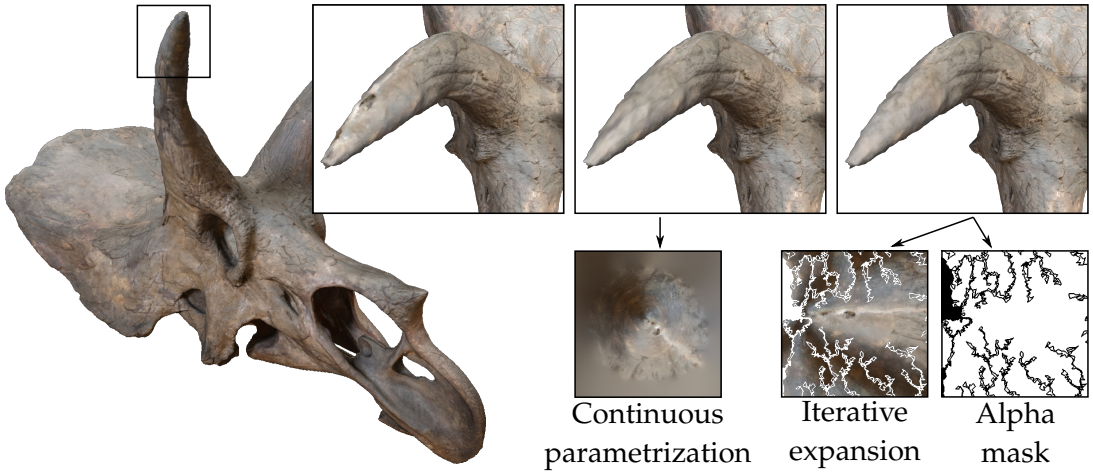


Figure 5.6: An inpainting operation covering a highly curved region. Using a *continuous* auxiliary parametrization, without cuts or repetitions, introduces severe distortions, which negatively impact the generation and transfer of inpainted texels, and result in stretching artifacts (middle inset). By allowing cuts, repetitions, and overlaps, we produce a better context for the inpainting operation (rightmost inset). Overlapping texels are marked in the alpha mask, are shown in white in the RGB image, and as black in the alpha masks.

2. We initialize B and $\mathcal{P} : B \to M$, to A and $\mathcal{P}_A$, and iteratively extend B, one triangle at a time, until a sufficiently large squared region S around A is fully covered.

In the first phase, any existing single-patch disk parametrization method can be employed. We opted for the local-global As Rigid As Possible (ARAP) parametrization [LZX*08], because of its robustness and performance in terms of isometric distortions. The mapping is not strictly guaranteed to be free from local overlaps (which would infringe the required injectivity), but we never encountered this issue. If need be, existing countermeasures can be applied during the global phase of ARAP [BCE*13].

The second phase follows the spirit of the incremental approach of [MZ12]. We initialize B as A, and at every step (see Figure 5.7) we select one boundary edge over B, according to a prioritization strategy (see below), and attempt an *expansion* step of B over that edge, adding to B the parametric domain t for one new triangle of M, and a new one vertex v to the boundary δB; the 2D position of v is determined as the minimizer of the total isometric energy [LZX*08] defined over t (keeping its two other vertices constant). After every expansion, we evaluate up to two potential *merge* steps of v (an operation called "edge retract" in [Hop96]) with either the boundary vertices v' and v'' located two edges away from v on the boundary, clockwise and counterclockwise; a merge operation consists in fusing together two consecutive boundary edges, and a potential operation exists if and only if these two edges are mapped by P into the same mesh edge (see Figure 5.7, right).

The details of this step are as follows.

Creating cuts. After a merge operation, the 2D position of the merged vertex is redefined as the minimizer of the combined, area-weighted, ARAP energy of all affected triangles in B. If the minimal ARAP energy value is larger than a tolerance value (we used 0.1), the merge operation is rejected. In this case, a seam is created in the final parametrization. Differently from similar approaches, we do not "freeze" these edges, and both can be elected in subsequent expansion steps, thus creating texture repetitions (see below).

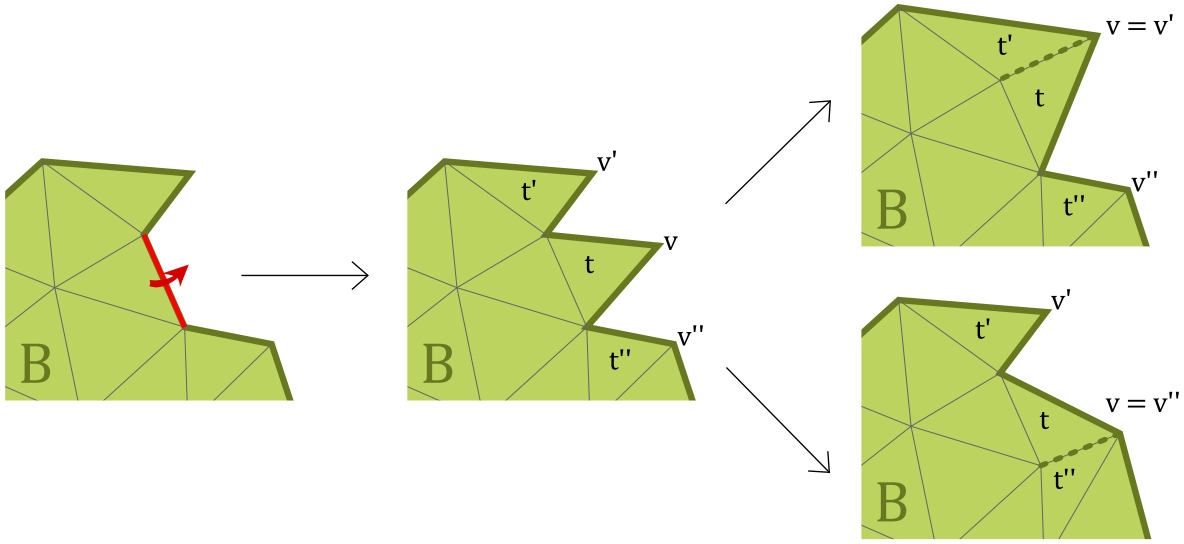


Figure 5.7: An expansion step, performed over one edge of δB (in red, left), expanding B and δB (middle), potentially followed by either one or the other (or both, in succession), fusion step (right). See text.

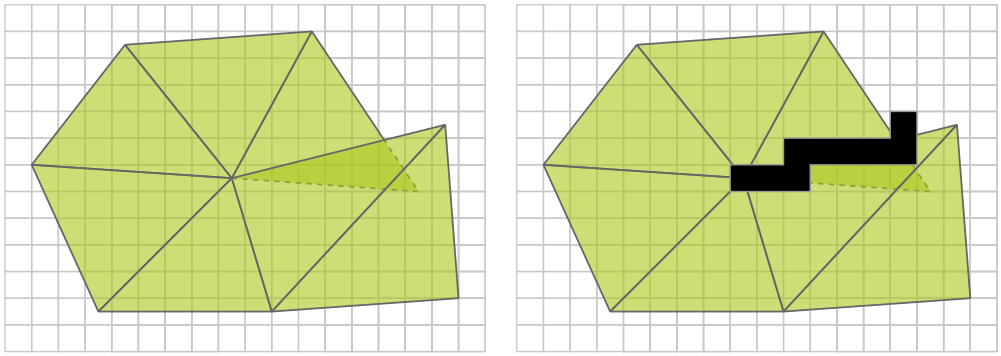
Crack-less overlaps avoidance. Different from similar approaches, we do not prevent the expansion or merge steps that would create an overlap of the patch. Instead, simply not elect for expansion any edge which is already *fully* in the interior of B. In other words, expansion of B is stopped after the overlap occurred. In this way, thus creating a narrow (up to one triangle wide) overlapping area, instead of a narrow empty area, at the seams of B. These overlaps will be dealt with in the subsequent patch creation phase.

Repetitions. We also allow expansions over triangles that are already represented inside B, resulting in potential repetitions, in S, of the textures of the triangles of M visited multiple times; while not ideal, repetitions are preferable to empty regions. For example, when unfolding the side area of a thin cylinder, B can wrap around M multiple times. As another example, when a merge operation is rejected (due to excessive distortions), both copies of the same mesh edge in parameter space can still be expanded in subsequent steps, repeating the triangle on the other side of the edge anew.

Definition and update of S. After the initialization phase, we set S as the square having 4 times the area of that Axis Aligned Bounding Box (AABB) of A, or its largest dimension (whichever is larger). After every step, we recenter S into the center of the AABB of B.

Edge prioritization strategy. At every step, we pick for expansion the edge in δB that is currently the farthest from the boundary of the rectangle S, so to encourage a uniform expansion of B over all S; more precisely, we score each vertex along δB with its distance from the nearest of the four sides of S, identify the highest scoring vertex v_{max}, and pick the edge on either side of v_{max} (we choose the side connecting v_{max} with the highest-scoring vertex).

Stopping criteria. We never pick one edge for expansion that is fully outside S, or already fully in the interior of B. When no edge in δB is eligible, we stop the loop. At that point, the entire surface of S will be covered at least once.



Overlapping triangles Alpha mask

Figure 5.8: The alpha channel of the texture patch is zeroed (black pixels, right) to annotate the discontinuities near the border edges of overlapping triangles.

5.6 Texture inpainting network

The next step is to inpaint the auxiliary texture, that is, to automatically generate new texel values to fill in the texels in the area marked as "defective", according to the context around that area.

We employ a close adaptation of a recently proposed Deep Convolutional inpainting network [LRS*18], although any other method can be plugged into our framework. This model uses an encoder-decoder architecture with skip-connections analogously to UNet [RPB15], and partial convolutions [LSW*18] to improve performance with masked image inputs. The network takes as input an image and a binary mask denoting valid and invalid pixels, and outputs a reconstructed RGB image.

We leave the network architecture and reconstruction loss mostly unaltered from previous art, but perform training on a dataset created specifically for our task. We implemented the model in PyTorch.

5.6.1 Texture re-resampling

Once we produce the auxiliary parametrization, we resample the original texture into S, and generate the binary mask by marking all pixels inside A.

We perform this task with a GPU-assisted rendering over S into a texture patch. The original texture is accessed with a standard bilinearly interpolated access to the original texture. The mask is drawn by rendering all the marked triangles in the patch.

As discussed, small areas of S can feature overlapping triangles (in correspondence with parametrization cuts), meaning that this rendering will present pixel overdraws (we render the triangles in inverse order of expansion, and disable depth-testing to ensure the last drawn textured triangle overwrites the others), which will potentially generate artificial color discontinuities. We generate an extra binary alpha channel to annotate such discontinuities, by drawing the visible edges of the overlapping triangles (Figure 5.8). The additional 1-bit channel is then fed to the network, together with the RGB channels; this is intended as a way to "inform" it of the artifact nature of the color discontinuity.

5.6.2 Training

One of the biggest drawbacks of data-driven approaches, and deep learning in particular, is the requirement of large amounts of data to drive the training process. Prior knowledge about the data itself can significantly affect the performance of neural networks, both in terms of network architecture, and what kind of data the network is exposed to during training.

Therefore, we choose to train the inpainting network using a task-specific dataset, that is, texture patches created from scanned objects, instead of using general-purpose datasets of natural images that are already available. This is motivated by the peculiarity of our inputs which, as discussed, differ significantly from generic photographs. Since texture patches lack features such as perspective distortion and depth jumps, the training process can be expected to require less data, and to converge more quickly, for the same quality. For the training, we generate a collection of around 50000 patches sampled from our Real-World Textured Things dataset (Chapter 3), extracting 100 patches from each model. We create training and validation sets with a random 85%-15% split.

Patch extraction. Given a textured model, we extract a number of patches by simulating strokes over the mesh and applying the auxiliary parametrization construction (Section 5.5.2), followed by the synthesis of the mask and the RGBA image (Section 5.6.1)

Training. The network is trained with batches of 24 samples at 256×256 pixels. We use 340 epochs of 40k images each, and fine-tune for 60 more epochs. Because texture patches contain not only RGB data, but also an additional channel of binary alpha values marking color discontinuities induced by the overdraw, we alter the first layer of the inpainting network to optionally accept 4-channels inputs. The output is still an RGB image and the reconstruction loss used to train the model simply ignores the extra input channel. The training was carried out on an NVIDIA RTX 3060 GPU with 12 GBs of VRAM; loss values and validation performance are reported in Figure 5.16, bottom.

5.7 Automatic UV-map defect identification

In our automatic framework (Section 5.4.1), triangles in M that require an inpainting process are identified by comparing the original UV-map U, where the original texture image is defined, with the new UV-map U_{new}, which is provided to the system and used to re-sample the final texture.

The rationale is that U_{new} is assumed to be constructed to minimize isometric distortions with the 3D mesh M, but balances this objective with a number of other requirements (such as sparsity of cuts, or different texture resolution for different parts of the mesh, etc). While U_{new} defines the *requested distribution* of texels on M, U defines *the actual availability* of texels.

5.7.1 Per triangle labeling

First, we analyze each triangle t of M and determine if that triangle requires inpainting, that is, if its image in T fails to provide an adequate source of texels for t. This is the case if any of the three problems below is detected.

Excessive distortions. Let u_0 and u_1 be 2D triangles that are the images of t in the two UV-maps U, U_{new}. The final texel re-sampling will consist in sampling a regular texel grid over

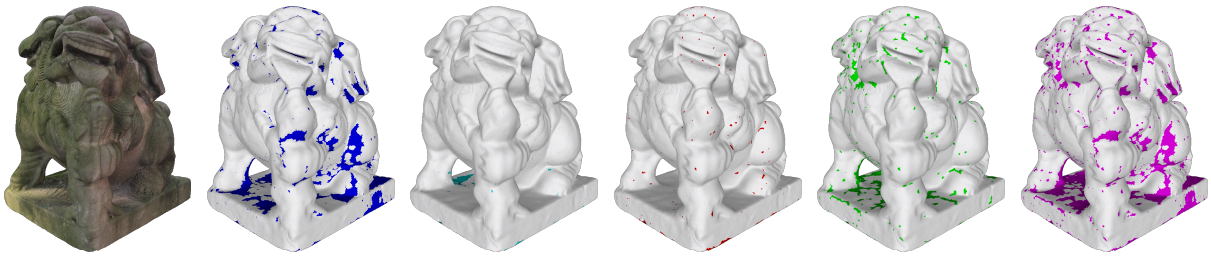


Figure 5.9: Segmentation and clustering of defective faces.

u_1 and, for each sample, copy the (interpolated) color value from the position in u_0 at the same barycentric coordinates. We employ the notions, well understood in the context of surface parametrization, to quantify the *distortions* of this mapping [FH05] (one difference being that our mapping is between 2D triangles). This analysis determines if the re-sampling will result in oversampling or undersampling, in any direction. Specifically, let F denote the linear map from u_1 to u_0 and J_F its Jacobian, that is,

$$J_F = [a_0|b_0][a_1|b_1]^{-1} \tag{5.1}$$

with a_0, b_0 being 2D column vectors denoting two arbitrarily chosen 2D edge-vectors in u_0, and a_1, b_1 being the corresponding ones in u_1; let σ_0 and σ_1 the singular-values extracted from the Singular Value Decomposition of J_F ($\sigma_0 \leq \sigma_1$) If σ_0 is larger than a given threshold, we mark the triangle as defective. As a default value for the threshold, we use 1.2, which corresponds to the choice to mark all and only the triangles region that will undergo an over-sampling of 20% in any one direction. If $\sigma_{0,1} < 1$, this denotes a situation where the original texture is undersampled to produce the final texture, and requires no intervention (because we assume that the sampling rate determined by U_{new} is the desired one).

Overlaps. We mark the triangles of M whenever u_0 overlaps with the image of any other triangle in U. This situation occurs when U is non-injective, unavoidably leading to texture artifacts when the same area of the original texture is copied multiple times over different areas of the final texture. Any detected overlap invalidates both overlapping triangles. To speed up detection, we use a regular grid paired with hashing as a spatial indexing structure.

Seam discontinuity. Another test we perform is to check whether or not the color signal stored in the original texture T happens to be mismatching on the two sides of any texture seam of U (in other contexts, such as CAD models or manually modeled game assets, this occurrence can be assumed to be by design, as texture seams often separate semantically different areas; in photogrammetric models, which are our target, it evidences a likely reconstruction defect, as discussed in Sec. 5.3.2). To detect the mismatch, we linearly sample every seam edge on both sides, collecting bi-linearly interpolated samples at the two corresponding locations of T, and measure their Euclidean (r, g, b) distance. If the difference, averaged over the edge, is larger than a threshold (we used 0.1 in a color intensity scale from 0 to 1), we mark any triangle sharing one vertex with that edge.

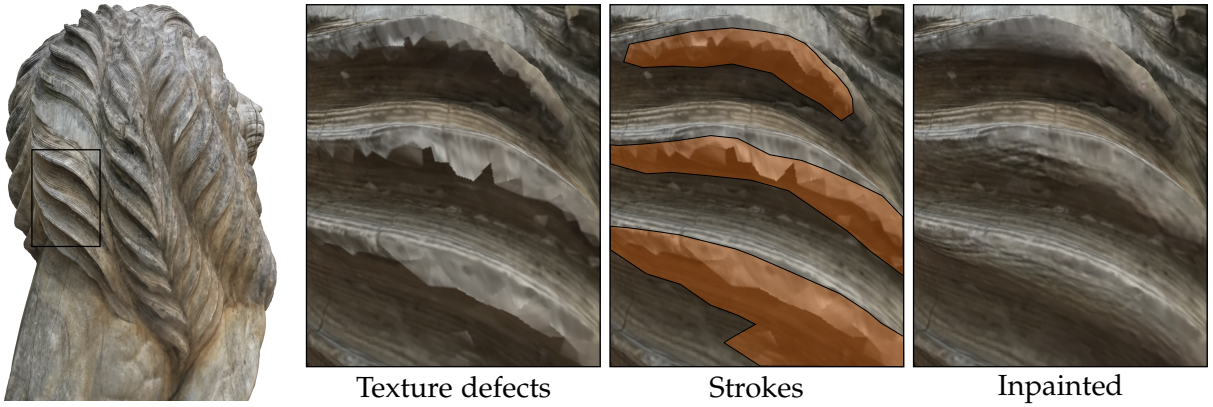


Figure 5.10: Interactive inpainting of a wooden statue. The interpolation over the artifacts correctly reproduces the wood grain.

5.7.2 Clustering defective faces

Once the triangles have been individually marked, we group them into connected components, each to be addressed by one inpainting operation (Figure 5.9). This may lead to an excessive number of patches, with jagged and discontinuous borders. To address this issue, we apply a patch border regularization using morphological operators, in a dilate-erode procedure. We apply two steps of dilation (marking any triangle sharing one vertex with an already marked one) followed by two steps of erosion (un-marking any triangle sharing one unmarked vertex).

5.8 Results and experiments

We empirically validate both our interactive and fully automatic frameworks, by performing a range of experiments on real-world data aimed at producing a qualitative (Sections 5.8.2, 5.8.3 and 5.8.6) and quantitative (Section 5.8.4) assessment, including direct comparisons with competing strategies.

5.8.1 Examples of results

Interactive inpainting sessions. Figures 5.10, 5.11, 5.18 show examples of results of various interactive inpainting sessions. The network correctly interpolates textures and recovers details that are consistent with the patch surroundings. In Figure 5.11, the isotropic content in the rock texture is correctly replicated, while in Figure 5.10 the inpainted areas correctly capture the anisotropy of the wood grain and fibers; in both cases, realistic looking missing detail is produced that seamlessly blend with the surroundings.

Automatic defect correction. Figure 5.12 shows the results of inpainting the automatic framework (Section 5.4.1), where texture defects are automatically identified and corrected. The procedure is reasonably quick, with the iterative inpainting accounting for most of the processing time. In the reported example (400k faces), the entire process takes around 10 minutes. While simple, our heuristic detects several visible artifacts, which are effectively corrected by the inpainting procedure. On occasion, the heuristic produces patches that are too large for the neural network to inpaint effectively, producing visible artifacts (see Figure 5.19).

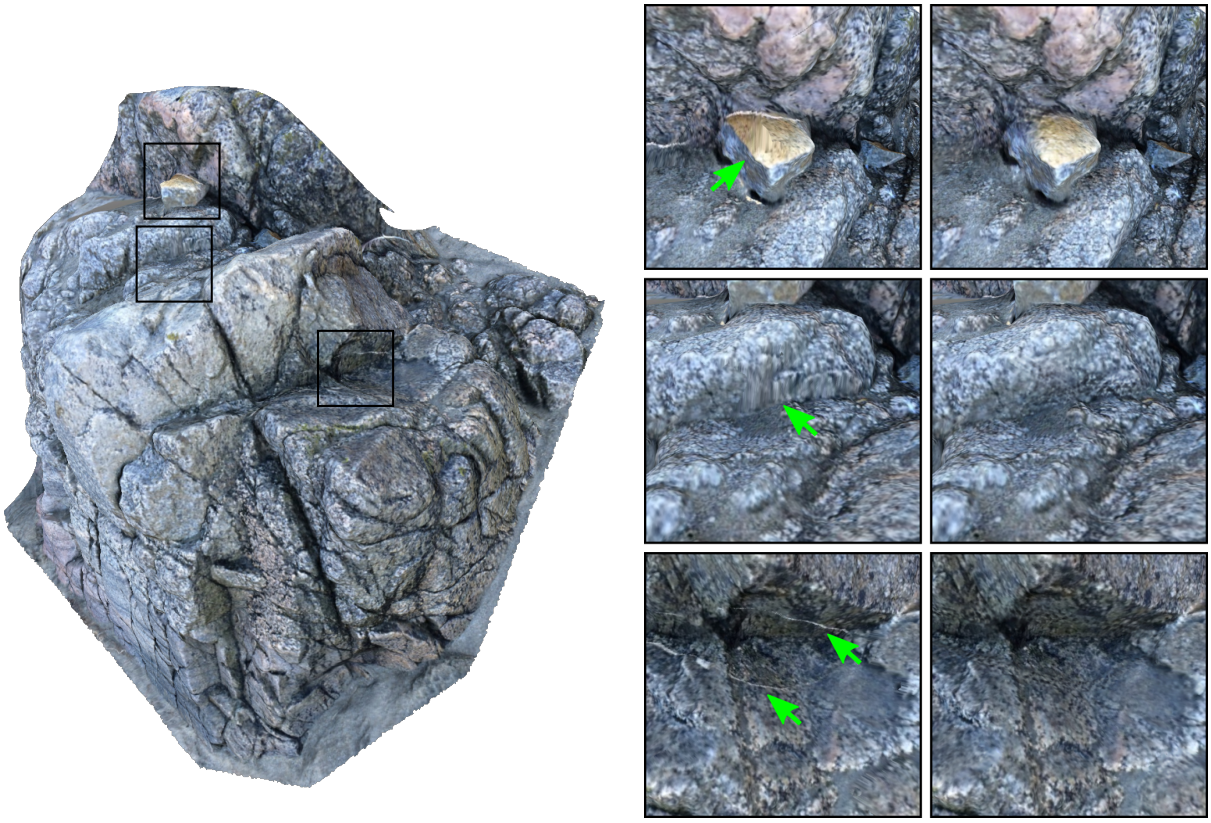


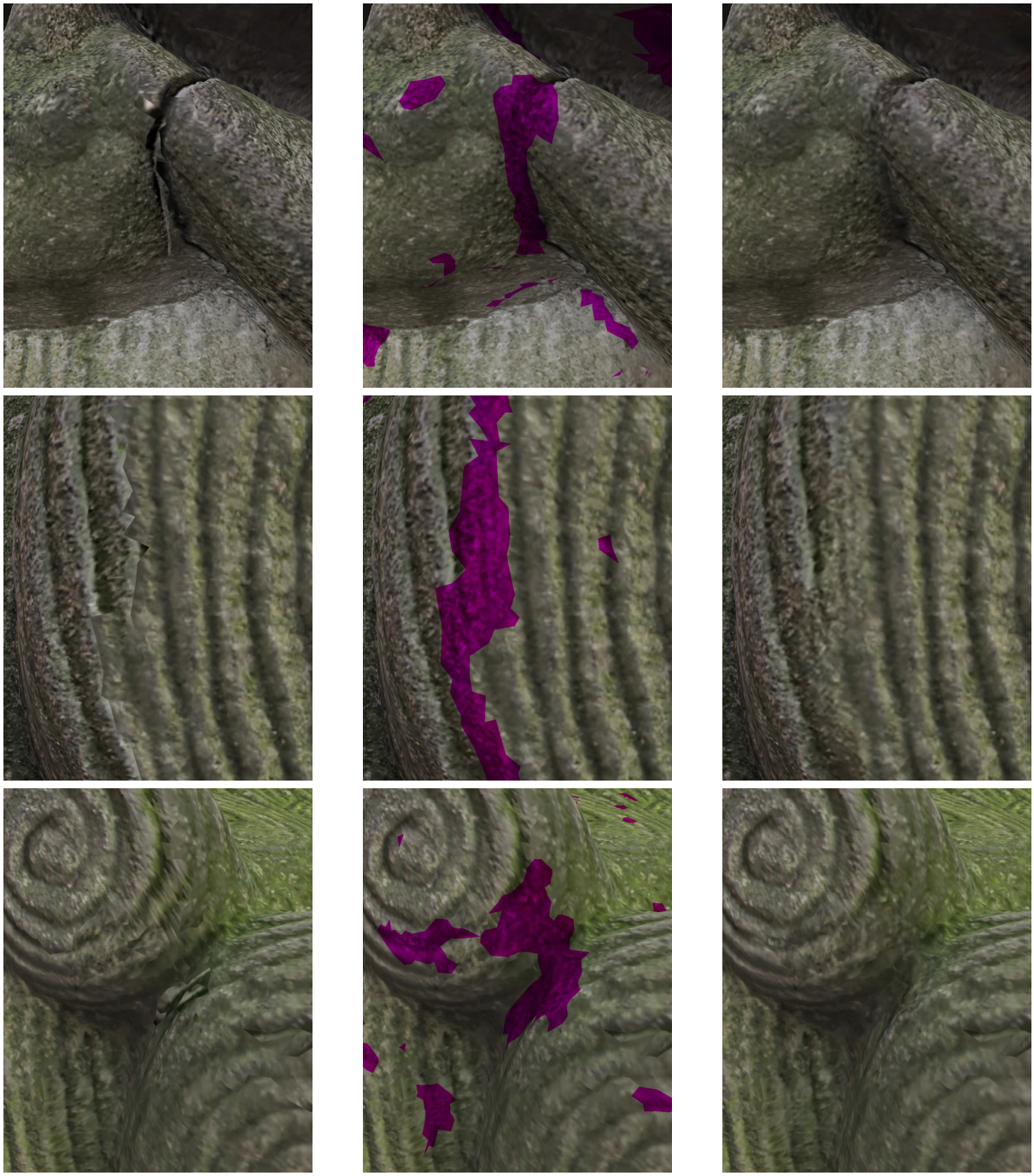
Figure 5.11: Interactive inpainting of a textured rock using the texture-space network. The network correctly reproduces the surface texture and remedies artifacts. Sharp discontinuities in the surface material (first row) are more difficult to treat and the network tends to produce overly smooth transitions.

5.8.2 Comparative evaluation

We compare our results against two alternative strategies: inpainting performed in screen-space, and inpainting performed in texture-space using texture synthesis.

Comparison with screen-space inpainting. For this comparison, we trained the same network from scratch on generic photographs taken from the Places365-Standard dataset [ZLK*18]. We used batches of 6 samples at a resolution of 512×512 pixels, as suggested in [LRS*18], and a learning rate of $2 \cdot 10^{-4}$ for 140 epochs (35k images each), followed by a fine-tuning with a learning rate of $5 \cdot 10^{-5}$ for 30 more epochs. Overall, the process took about 10 days.

Visual results can be seen in Figure 5.13. Experiments evidence a significant quality improvement in moving from screen-space inpainting to texture-space inpainting, (third to fourth column). The screen-space inpainting systematically produces visible artifacts and is unable to correctly capture the silhouette of the 3D geometry and its impact in the overall image lighting. As a result, the inpainted area always looks flat and disconnected from its surroundings. Conversely, out inputs constructed from the auxiliary parametrizations visibly improve the results both in texture richness and smoothness of blending the inpainted patch with the surroundings. Other insights on the differences between texture-space and screen-space



Original texture | Automatic detection | Inpainted

Figure 5.12: Inpainting of automatically detected defects. Top: UV-overlaps and texture seams. Middle: visible texture seam due to the color discrepancy induced by adjacent texture charts with different sampling resolutions. Bottom: UV-map distortion.

inpainting are given by the assessment of the respective training process, in Section 5.8.5.

An additional, inherent advantage of texture-space inpainting is that the results are independent of the observer's position. This is shown in Figure 5.3, where a texture defect is interactively selected for inpainting. With screen-space inpainting, switching to a different view reveals projection artifacts induced by perspective distortion; our method, instead, is only affected by the distortion of the underlying parametrization, which is usually drastically lower.



Figure 5.13: Visual comparison of inpainting results in different spaces. The framing is the same one used for screen-space inpainting.

Comparison with texture synthesis. For this comparison, we use a publicly available implementation of a State of the Art texture synthesis technique [Opa19, Emb19, EL99, WL00], and feed it the same input as that of our neural network (the input texture patch, and a binary mask marking the area to be inpainted). We have integrated this tool in our implementation; the GUI exposes it as a different inpainting mode, easing the comparison with the neural

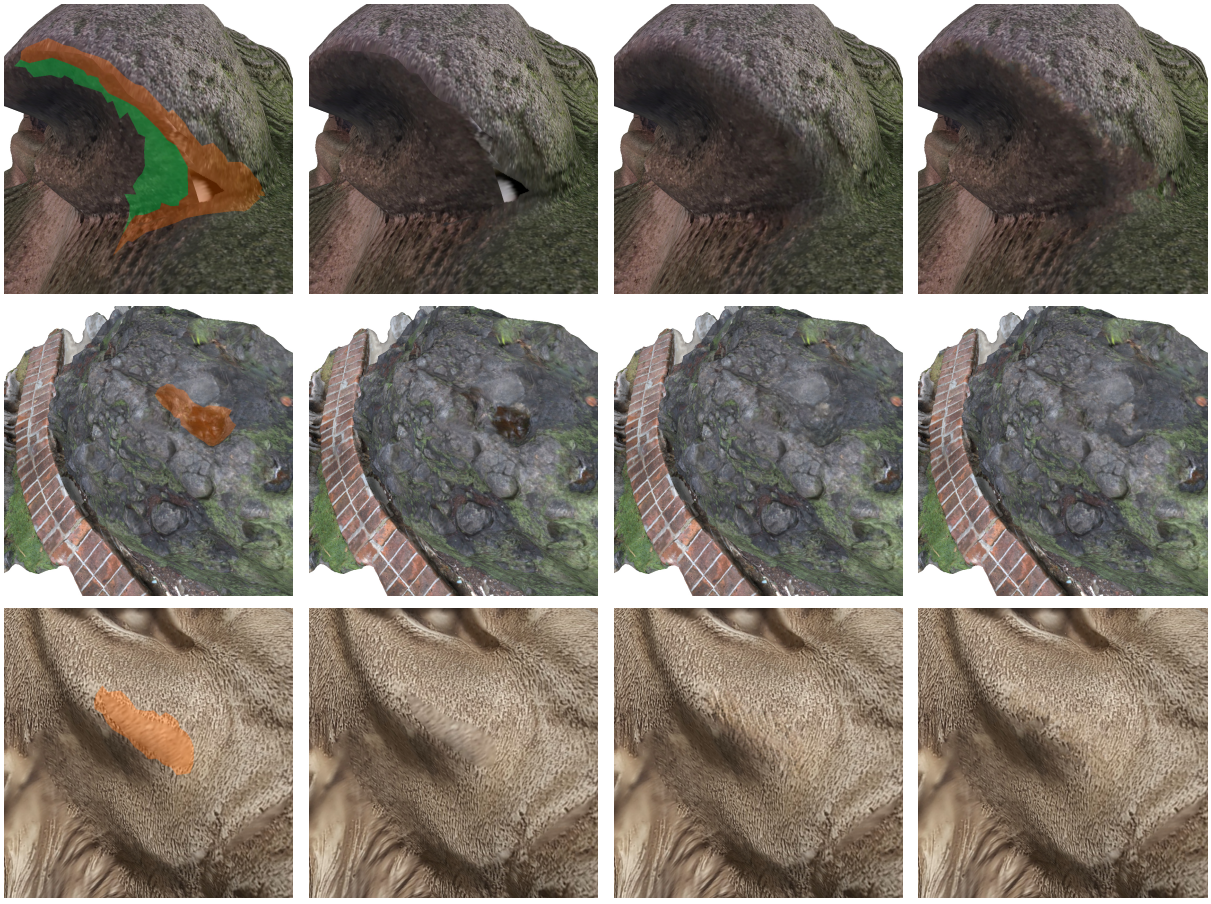


Figure 5.14: Inpainting with the texture-space network and texture synthesis produces results of similar quality. Texels interpolated with the neural network blend smoothly with the surrounding context, while texture synthesis occasionally produces sharp material transitions (first row).

model on the exact same inputs.

Examples of visual results can be seen in Figure 5.14. The results of the neural network are qualitatively similar to using texture synthesis, but sometimes the texture synthesis produces sharp transitions between surface textures (Figure 5.14, first row, where the texture synthesis hallucinates a sudden transition from moss to rock); when used to fix a blurred input, texture synthesis sometimes produces lower frequencies than desired (third row).

5.8.3 Comparison with inpainting in global texture space

Figure 5.15 compares our local strategy that leverages the auxiliary parametrization against a more trivial approach of inpainting directly in global texture space. For global texture inpainting, we use a neural network trained on texture patches without overlaps and alpha masking (as described in Section 5.8.4). We show an example where an inpainting operation is attempted near a seam of the global texture space, resulting in disconnected regions of the texture to require inpainting. Our approach bypasses the problem, resulting in a significant quality improvement in the produced result. Another drawback of the direct approach is that the entire texture must be fed to the inpainting model, drastically increasing the time and

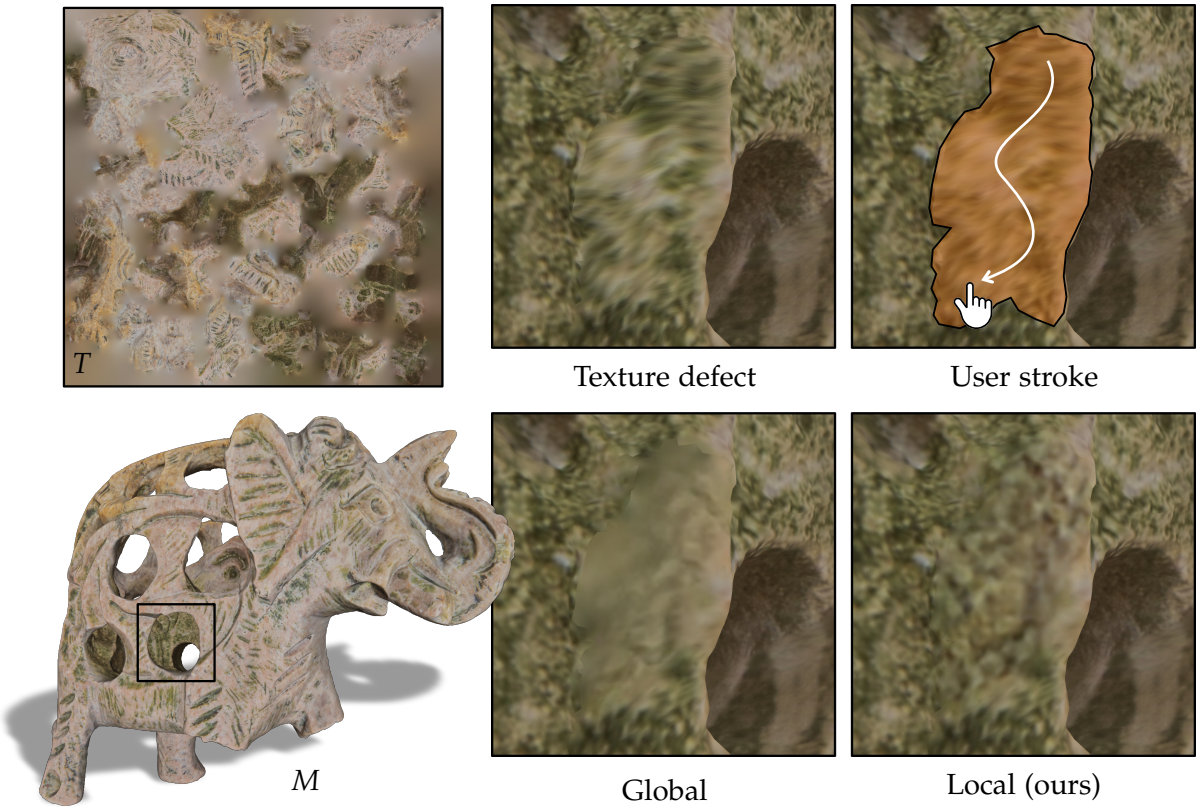


Figure 5.15: Comparison of inpainting in global texture space and our approach of building a local patch. Global inpainting causes the unused background texels to bleed into the texture charts. *Model: Broken Stone Elephant by Andrea Spognetta (CC BY-NC 4.0).*

memory requirements of the operation. For reference, inpainting a 4k texture with our CNN model allocates more than 16 GBs of memory and is an order of magnitude slower compared to inpainting our 512×512 patches.

5.8.4 Quantitative evaluation and ablation study

We also compare results obtained by allowing or disallowing triangles to overlap the auxiliary parametrization, and report quantitative measures that validate our choice to do so. Without overlaps, empty "cracks" regions appear in the image patch and are left for the network to fill.

We evaluate three different variations of inpainting in *texture-space*: feeding texture patches *without* overlaps to the screen-space network; feeding patches *without* overlaps to a network retrained on examples of the same kind; and our proposal of feeding patches with alpha-masked overlaps to the network trained on examples of the same kind.

For this quantitative analysis, we extract 721 texture patches (simulating user strokes from interactive inpainting sessions) from a fresh set of eight models *not* used in the training dataset, and measure the texture reconstruction quality using the original texture as ground truth reference. Table 4.1 reports the average Structural Similarity (SSIM) [WBSS04], restricted to the inpainted area, and the number of times each network produced the best score. The screen-space network is out-performed, further corroborating the argument that tailored training on texture-space patches is beneficial to our task; the inclusion of UV-overlaps and alpha masking further improves overall inpainting performance.

	Network trained on		
	Natural images	Textures	Textures with Alpha mask
Average SSIM	0.526	0.546	**0.551**
Winning rate	12.6%	39.1%	**48.3%**

Table 5.1: Validation of our inpainting approach using the SSIM score on texture patches extracted from unseen 3D models. *Natural images* is the screen-space model (trained on the Places2 dataset), *Texture* is the texture-space model without UV-overlaps and alpha masking, *Texture+Alpha* is our proposed model, trained on input domains with UV-overlaps and alpha-masking. SSIM scores are computed using [Ima21].

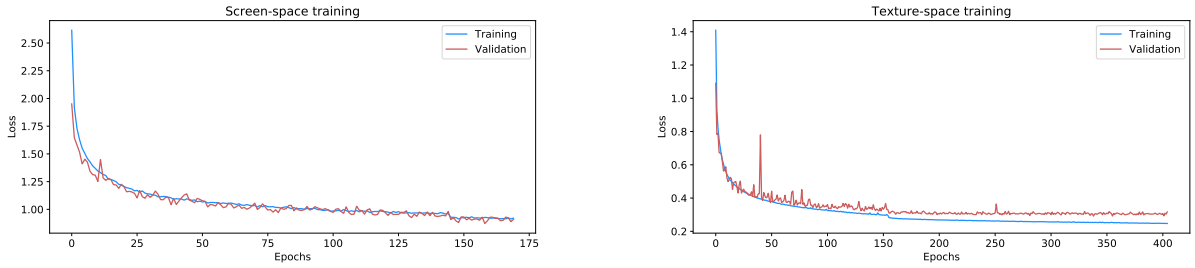


Figure 5.16: Training runs for the screen-space network (top) and the texture-space network trained on our patch-dataset (bottom). The screen-space model converges with a larger loss due to being trained and validated with samples at higher resolution (512×512 pixels instead of 256×256).

5.8.5 Effectiveness of the learning process

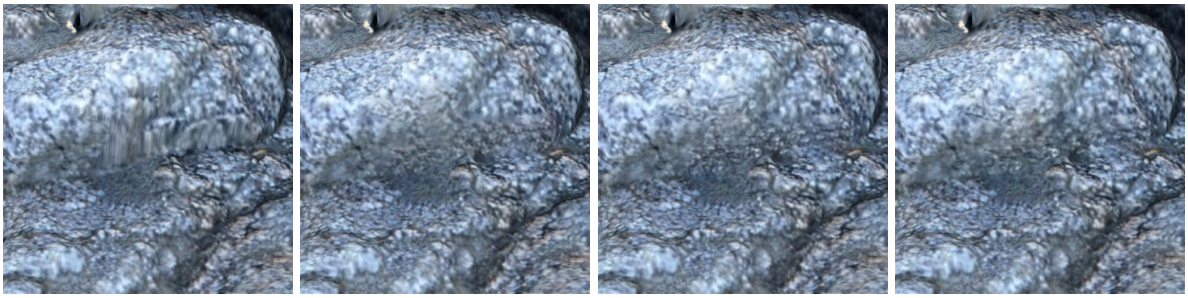
Figure 5.16 reports training and validation losses for both screen-space and texture-space networks; loss values are averaged and integrated over the image area, accounting for the resolution difference. The data indicate that both networks are successfully learning the respective domain, but a comparison suggests that the inpainting task is indeed made "easier" to learn in texture-space, using the auxiliary parametrization, as we conjectured, empirically validating the premise at the basis of our work.

Figure 5.17 qualitatively shows the progress of the texture-space network as the training is performed. At the beginning, the network learns to produce rich patches with varied detail; as the training advances, most of the improvement involves a better blending of the inpainted patch with the surroundings, by adjusting contrast and tone.

5.8.6 Comparison with real photographic data

Finally, we show a qualitative comparison of the ability of our inpainting framework to restore the original appearance of a textured object, using an actual photograph of the real-world object as ground truth.

In this experiment, we *exclude* a photograph from the texture reconstruction process, and compare it with the reconstructed texture before and after inpainting, as seen from the held-out view; the results are shown in Figure 5.18. In the first row, we show the texture obtained by excluding the view from the texturing process, which is affected by noticeable ghosting



Input 50 Epochs 150 Epochs Converged

Figure 5.17: Texture-space network improvement during training for stretching artifacts removal. The network progressively learns to produce patches both with richer texture detail and that blend better with the surrounding texels.

artifacts (first close-up) and missing color data (second close-up) compared to the ground truth photograph in the second row. The third and fourth rows show the regions selected for inpainting and the inpainted texture, with ghosting artifacts removed and plausible surface texture generated by the network in place of the missing data.

5.9 Conclusions

In this chapter, we presented a framework to apply inpainting techniques to textures of models coming from photogrammetric acquisitions. Our contribution is orthogonal to present and future advancements in inpainting techniques themselves, and provides a setup where most such techniques can be first trained and then applied. This allows for the direct inclusion of future works in a fast-advancing field such as inpainting, even if it is designed for natural images (and not textures). New inpainting models are likely to still benefit from the present work, i.e., from the idea of operating in a tailored re-parametrized texture domain, resorting to marked repetitions to limit both gaps and distortions.

Limitations. An issue of current Neural Network based inpainting, inherited by our system, is the limited size of the missing region in pixels. When the missing region is too large, the network can produce unexpected results, as shown in Figure 5.19. Using Neural Networks to inpaint high-resolution images and large missing areas is a currently developing research area, and recent works show promising results by combining and integrating patch-based strategies in the neural architecture to hallucinate high-frequencies.

Our work also inherits the inability of neural networks and texture synthesis approaches to always deal correctly with repetitive structures and patterns (e.g. grids, checkerboards).

Ethics and misuse. Altering the appearance of specific classes on captured models, e.g. cultural heritage objects or real people, raises ethical concerns; while the original texture defect does not faithfully document the reality, neither does our “fixed” texture, which has the additional peril of deceptively looking realistic. To counteract this problem, it may be helpful to visually annotate the inpainted regions.

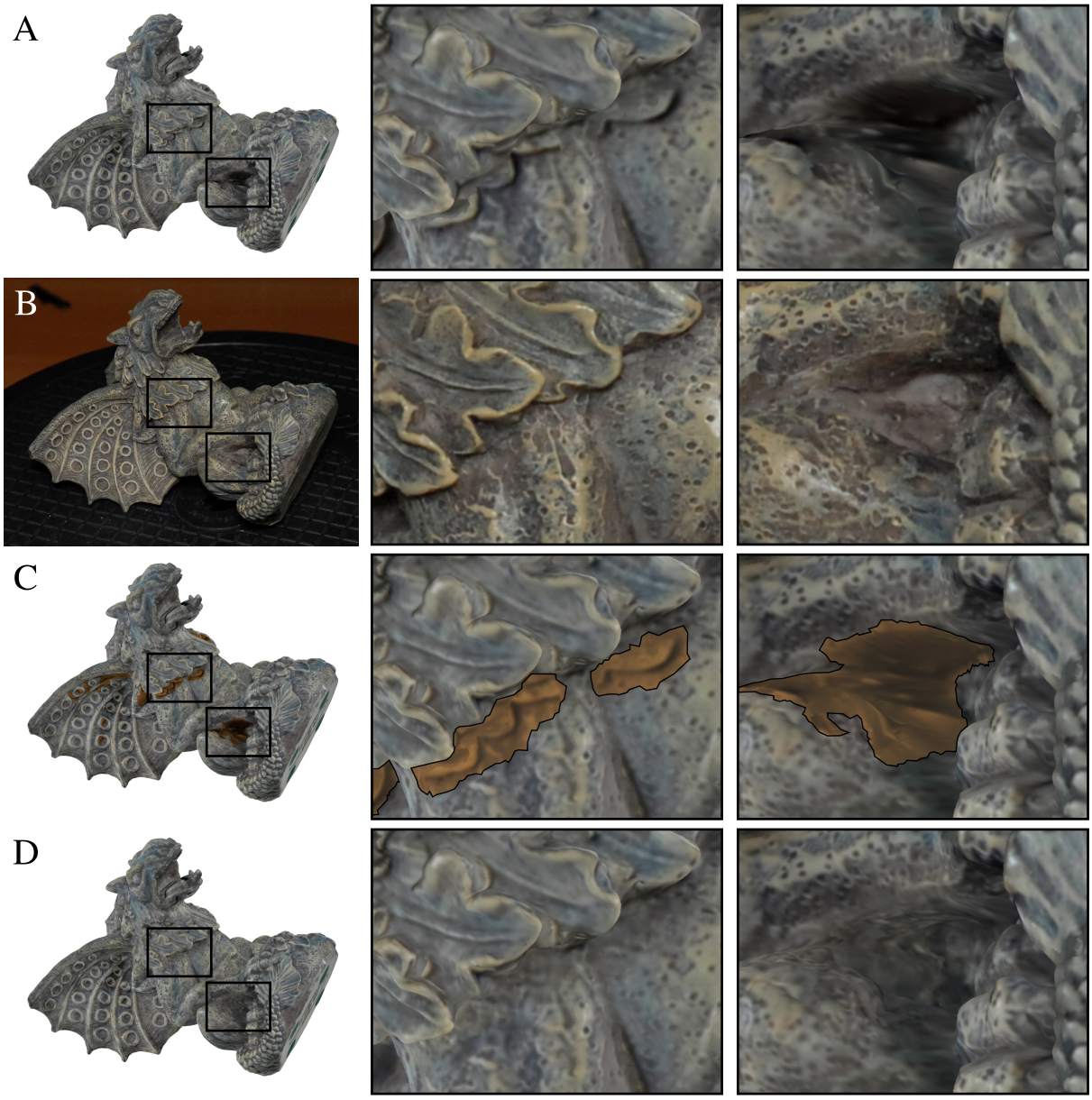


Figure 5.18: A qualitative assessment of the ability of our framework to restore the look-and-feel of a defective texture. From top to bottom: (A) a rendering of a 3D reconstructed model, featuring texture defects caused by missing photographic data; (B) a real photograph of the same object (*not* used in the texture reconstruction), showing the real aspect of the defected areas; (C) the manually selected region that marks the defected region; (D) the resulting inpainted texture. On the right: two close-ups. Comparing B and D suggests that, in this instance, the overall aspect of the object is successfully recovered.

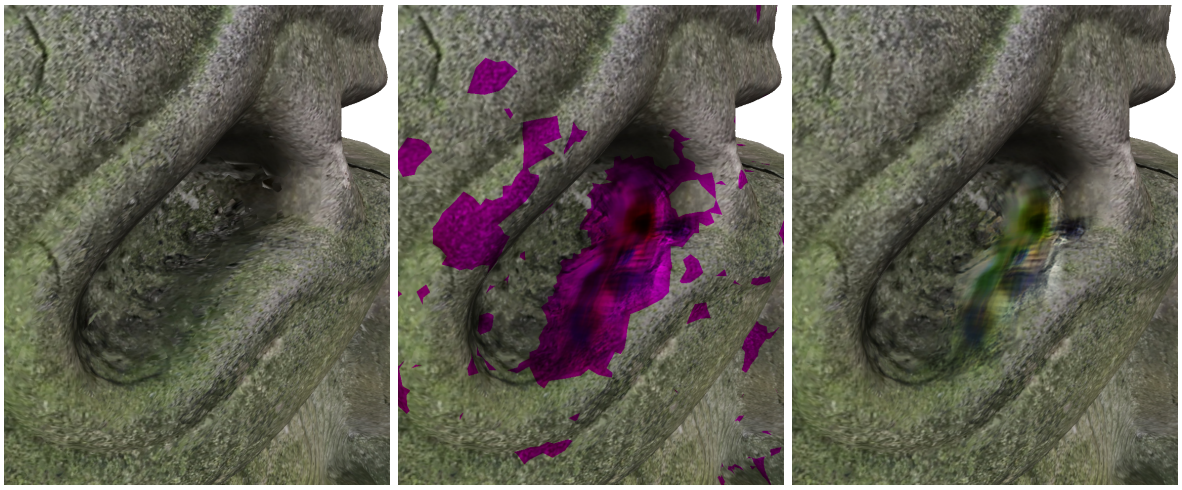


Figure 5.19: Artifacts produced by the neural network when attempting to inpaint large defective regions.

Chapter 6

Efficient representation of high-resolution geometry

High-quality photogrammetric 3D reconstructions are often characterized not only by the presence of texture data but also by extremely dense tessellations. The cost of rendering complex high-resolution geometry in real-time is often prohibitively expensive in terms of resources, hindering the adoption of these assets in interactive graphics applications. We introduce a method to optimize the geometric description of high-resolution triangle meshes by adopting a recently proposed displacement mapping scheme with dedicated GPU support, the "Micro-mesh" [NVI22b, NVI22a]. First, we propose a simplification scheme tailored to generating high-quality coarse base meshes, optimized for the subsequent refinement and displacement sampling. Then, we turn our attention to the resampling of texture data for this new representation, introducing a re-weighting scheme that allows us to minimize the parametrization distortion of the coarse base mesh according to the refined and displaced surface. Experimental results demonstrate that our algorithm outperforms alternative base mesh generation methods, in terms of representation accuracy and performance, on a selection of densely tessellated and textured meshes.

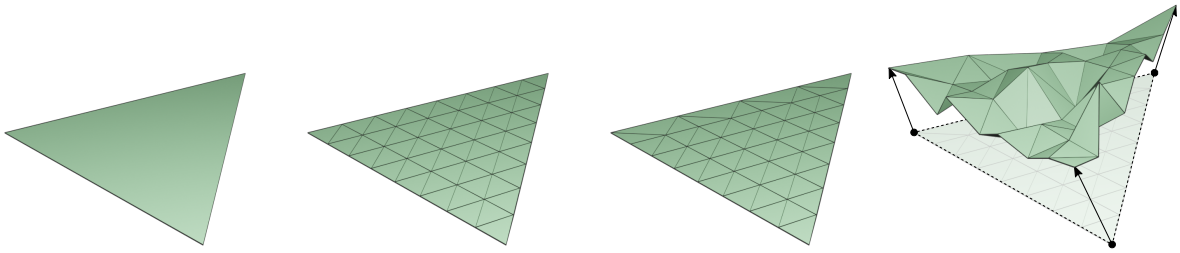
6.1 Introduction

Micro-meshes (from now on, μ-Meshes) are a recently introduced polygonal representation for 3D models, designed for efficient GPU-based rendering of high-resolution models; specialized hardware is now available to natively support it [NVI22b], unlocking unprecedented triangle counts in interactive applications such as videogames and virtual reality, and, ultimately, resulting into higher-quality real-time renderings.

The prominent advantage of μ-meshes is their ability to be efficiently rendered using either a rasterization-based or a ray-tracing-based approach. In addition, μ-meshes offer benefits such as inherent multi-resolution capabilities, removing or alleviating the need of storing multiple levels of a LOD-pyramid separately, and limited GPU memory footprint and GPU bandwidth usage, thanks to the adoption of specialized compression schemes.

This makes μ-meshes the ideal candidate to take the role of the main representation for 3D models, substituting relatively lower-resolution traditional meshes that are now the de-facto standard in real-time rendering applications.

Unfortunately, the construction of μ-meshes is an unexplored area, whereas traditional meshes



Base triangle | Regular subdivision | Optional edge decimation | Displacement

Figure 6.1: The μ-mesh geometric primitive. First, a base triangle is refined using regular 1-to-4 subdivision; then, border edges are optionally decimated to control the triangulation density of μ-mesh complexes while ensuring water-tightness; finally, the refined geometry is displaced along linearly interpolated displacement directions that are sampled at base vertices. Scalar micro-vertex displacements are assumed to be normalized in $[0, 1]$.

can be produced with a variety of approaches, including manual sculpting, 3D acquisition, and others. To allow digital artists to produce μ-meshes, new asset authoring pipelines would have to be devised or modified, incorporating new tools and approaches.

In this chapter, we propose an alternative route, exploring the problem of the automatic production of a μ-mesh starting from a traditional high-resolution mesh. This is intended as a way to ease the transition from the traditional representation to the new μ-mesh representation, and possibly as a tool for future asset creation pipelines designed to produce μ-meshes. Ultimately, our goal in the context of this thesis is to represent photo-reconstructed meshes: to this end, we also propose a parametrization tailored to this representation, that we use to convert photo-reconstructed textured models.

6.1.1 Problem statement

Our problem statement is as follows: given a high-resolution mesh representation M of a surface, we seek an alternative representation of the same surface that follows the μ-mesh constraints [NVI22b], namely we seek a constrained representation consisting of

1. A low-resolution surface representation acting as *base* or *control* mesh;
2. Micro-triangle topologies specified per base triangle as a 1-to-4 triangle subdivision level, possibly reduced by one at each edge by specifying three edge decimation flags;
3. A set of displacement directions specified per base vertex and linearly interpolated to produce micro-vertex displacement directions;
4. A set of per micro-vertex scalar displacements, representing the displacement map of the final refined micro-geometry.

Figure 6.1 shows the geometric primitive used to represent a complex μ-mesh surface.

Our objective is to produce an *accurate* and *efficient* μ-mesh representation of the input surface M, namely one that for a given base mesh and μ-mesh geometric complexity (triangle count), satisfies the following goals

(i) Isotropy of the micro-sampling;

(ii) Adaptivity of the micro-geometry, to faithfully reproduce small-scale surface detail;

(iii) Minimal displacement volume.

Goal (i) ensures that micro-vertices are distributed uniformly over the surface of the resulting μ-mesh, and directly maps to the isotropy of the base mesh triangulation, as the rules for producing the final micro-geometry are constrained to (recursive) 1-to-4 subdivisions of the base triangles. Goal (ii) avoids unnecessary dense micro-sampling where not needed, and can be achieved through adaptivity of the base mesh and grading of the subdivision levels, exploiting decimation flags to ensure water-tightness of the resulting micro-geometry. Goal (iii) has direct repercussions on the culling efficiency of graphics systems designed around the μ-mesh data structure, in particular those employing renderers based on ray-tracing, as base triangles can be culled before subdivision by performing intersection tests against prismoids enveloping the displacement volume spanned by a base-triangle and its three direction vectors (Figures 6.1 and 6.2).

6.2 Related work

Efficient representation of high-resolution geometry The visualization of high-resolution 3D meshes is a challenging task that requires significant resources in terms of computing power and memory storage, therefore significant efforts have been directed toward the design of efficient representations. The two main approaches to deal with extremely high-resolution data in interactive rendering systems are Level-of-detail meshes (LODs) and progressive encoding schemes.

Systems based on LODs [RB93, FS93] store several copies of the original geometry at lower resolutions and dynamically select the appropriate one according to the distance from the observer or similar metrics, but have several limitations: replicating data can be expensive if the mesh is densely tessellated, and transitions across detail levels produce visible "popping" artifacts.

Progressive schemes are adaptive multi-resolution representations that control the tessellation density dynamically across the surface, significantly reducing the transition artifacts. Pioneered by Hoppe with the Progressive Mesh data structure [Hop96], these schemes encode the sequence of operations performed by an edge-collapse decimation of the high-resolution mesh, and smoothly transition between resolutions by traversing the decimation sequence (forward to coarsen the mesh, and in reverse to refine it). Early approaches encoded transitions at triangle granularity [Hop96, Pup98, DFMP98], while more recent multi-resolution schemes adopt patch-granularity at a few thousand triangles to take advantage of the rendering speed of modern GPUs [GG*03, CGG*04, YSGM04, SM05].

More recently, alternative approaches designed around fast GPU tessellation schemes and displacement mapping have also been proposed [SS09, LD08, BS08]. These approaches represent the high-resolution surface with displacement values stored in a texture image (e.g. heightfield), and adaptively refine and displace a low-resolution base (or control) mesh according to the desired level. Typically, the connectivity and triangulation density of the resulting finely tessellated mesh are determined by simple subdivision schemes that can be efficiently implemented in hardware or dedicated shader code. The μ-Meshes representation belongs to this latter category.

Remeshing Remeshing algorithms optimize the representation of a surface by altering its connectivity (topology) and triangulation properties while preserving the original surface

appearance. The literature on remeshing methods is vast and spans several decades. For a comprehensive review, we refer the reader to a very recent survey [KPF*22].

Some methods optimize the surface connectivity with local topology-modifying operations [HDD*93] such as edge collapses, splits, flips, vertex smoothing [GH97, BK04, GZ05, GH98, Hop99, COM98, LT00, SAG03, HYB*17]. When remeshing is done to reduce the triangle count of the input mesh, faithfully reproducing the original surface appearance becomes critical, and state-of-the-art methods determine the decimation sequence and vertex spatial locations by approximating the geometric error produced by the mesh simplification. Most methods rely on vertex quadrics [GH97] (Section 2.4.1) to estimate the geometric error and compute optimal vertex placements for the decimated mesh. Quadrics have been later extended to incorporate additional appearance information other than the surface geometry such as colors [GH98] and texture coordinates [Hop99, GZ05]. Other methods incorporate texture-deviation terms to minimize texturing artifacts [COM98], or visual differences [LT00], to guide the simplification; however, these methods are quite costly as they require rendering texture patches or the entire object *during* the simplification process. An alternative to the error quadrics is to compute the bidirectional Hausdorff distance on local patches to guide the remeshing operations [HYB*17].

A common objective of remeshing methods is to produce isotropic triangulations. To this end, some methods remesh the surface by computing a Centroidal Voronoi Tessellation (CVT), finding the vertices of the new triangulation as the centroids of the Voronoi cells. In [AECdVDI05], the Voronoi tessellation is computed over a parametrization of the input surface, using curvature information to guide the adaptivity of the resulting triangulation by controlling the cell sizes; however, parametrization-based approaches require a consistent global parametrization with minimal distortion, which can be difficult to compute for complex high-resolution surfaces as discussed in Section 2.2. For this reason, other methods compute a supporting CVT for the triangulation in 3D instead [YLL*09, LWL*09].

[HYB*17] apply local remeshing operators and bound the minimum resulting angles to produce an isotropic triangulation. We partly adopt a similar strategy; however, in our case, we need to produce a *coarse* base mesh and develop mechanisms to ensure the bound on the isotropy of the resulting triangulation does not lock the decimation process.

Quad-meshing algorithms convert a triangle mesh into a quad mesh [PPT*11, JTPSH15] and can also be used to produce coarse triangulation. The literature on quad meshing is vast and reviewing it is beyond the scope of this chapter; we refer the interested reader to the survey [Cam17].

All the above methods are general and not designed around specific applications. In our case, we are interested in producing a coarse base mesh that produces a high-quality displaced μ-mesh, and we propose an end-to-end framework tailored to the task, of which the base mesh generation is the key component.

Mesh generation for displacement mapping Displacement mapping [Coo84] is a technique that allows to represent a high-resolution surface by combining a coarse base mesh with scalar or vector displacement data sampled on the surface. A highly tessellated surface can be produced by refining the coarse base mesh and displacing its vertices either along a prescribed direction (e.g. the vertex normal), or in arbitrary directions (vector displacements).

Displacement maps require *baking* [CMR*99] the displacement values onto the base mesh surface, typically storing the samples in a texture image; this operation quite common, and

is supported by common 3D modeling suites [Ble22, Hel21, Ado21, Mar22]. However, the configuration of the base mesh can greatly impact the quality of the final displaced surface. In this chapter, we *adopt* this representation scheme and concern ourselves with how to reliably produce *high-quality* representations using simple tessellation schemes and displacement maps.

Construction of base meshes for displacement mapping is not a trivial task, and approaches have been proposed based on the estimation of re-projectability of the high-resolution surface onto the control mesh [LMH00, CH02, PPT*11]. In Section 6.4 we discuss a simple and robust criterion to ensure that a base mesh is projected correctly onto the high-resolution surface.

Normal meshes [GVSS00] are similar to μ-Meshes in spirit, as they also represent the high-resolution surface by subdivision and scalar displacement. In a Normal Mesh, new vertices are displaced along the normal direction of the local frame of the subdivided surface *at the previous level*: this representation is non-linear, and creating the control mesh must satisfy a different set of constraints than ours.

Bijective shells [JSZP20] have been recently proposed as a geometric framework, citing attribute transfer and displacement map synthesis as central objectives. Their shell generation employs a similar approach to ours, based on local remeshing operations, but the input surface must satisfy strong assumptions to guarantee the existence of the bijection (manifold, self-intersection free) that are generally violated by photo-reconstructed meshes. Guaranteeing bijectivity between the coarse shell and the high-resolution mesh requires carrying out expensive geometric tests during the simplification process. Compared to [JSZP20], our generation method is faster and more robust in that it does not make any assumption on the topology and geometry of the input mesh.

Surface parametrization for displacement mapping Texturing displacement-mapped surfaces poses an interesting riddle. The displacement data is usually sampled from a texture image, thereby requiring a parametrization; however, the actual high-resolution geometry will be displaced from the base mesh, introducing distortion that the original parametrization is unable to account for [ZR19]. So how can we get a low-distortion parametrization for a displaced surface?

Few works deal with this problem in the literature [MW08, ZR19], and the proposed solutions compensate for the displacement deformation by using indirect textures: a 2D offset field is computed and stored together with the displacement map, and texture accesses to non-displacement data are offset so that distortion of the map to the displaced surface is minimized [MW08, ZR19].

In Section 6.10 we show how standard parametrization energy formulations can be adapted to account for the deformation induced by the μ-Mesh displacements. Contrary to the aforementioned works, the displacement map we generate is implicitly parametrized and sampled on a barycentric grid. This allows us to treat the displacement deformation as prior information and formulate a distortion energy that is re-weighted to account for the 3D surface deformation. We then minimize the re-weighted energy, solving for the base texture coordinates. Optimizing only the base coordinates reduces the degrees of freedom of the system, but our solution has the advantage of not requiring indirection maps or dedicated shader code.

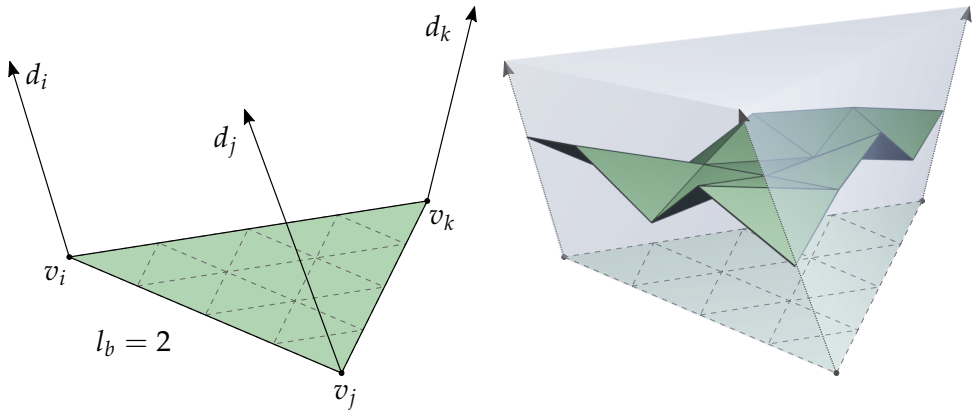


Figure 6.2: The output of our method, restricted to a single base face, encoding a bilinear prismoid whose volume envelopes the surface of the final μ-mesh.

6.3 Overview

We propose a method to generate a μ-mesh representation of a high-resolution input mesh $M = (V, F)$ where V denotes the mesh vertices and F the mesh faces. A μ-mesh is a base mesh $M_B = (V_B, F_B)$ augmented with additional attributes: a set of displacement directions D defined at base mesh vertices and interpolated across base faces, face subdivision levels S_B and three edge decimation flags E_B^i per face, and finally a set of scalar micro-displacements δ_μ, one per micro-vertex. Note that a μ-mesh implicitly defines a volume that contains the displaced micro-geometry and acts as a bounding proxy for intersection queries and collision detection (Figure 6.2, right). The lower and upper surfaces enveloping the tessellated and displaced surface share the same connectivity of the base mesh and their spatial coordinates can be obtained from the μ-mesh data as

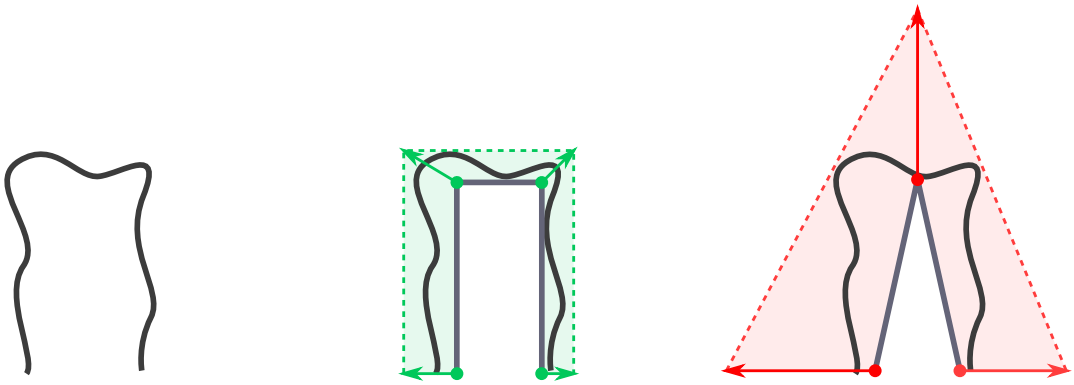
$$V_L = V_B + \delta_{min} D, \tag{6.1}$$

$$V_U = V_B + \delta_{max} D. \tag{6.2}$$

Input The input of our construction procedure is a standard but extremely high-resolution mesh. We primarily target meshes reconstructed from 3D scanners and photogrammetry, therefore we cannot make any assumption on the underlying mesh topology, the lack of geometric and topological artifacts, and, in general, the quality of the input mesh. As shown in Chapter 3, the connectivity of such meshes is, in general, highly irregular and the surface is prone to reconstruction artifacts such as inverted faces and other high-frequency artifacts, which our construction method must be able to handle to have any practical impact.

Output The output of our algorithm is a coarse geometric description of the resulting displacement volume and the set of attributes required to synthesize and displace the micro-geometry.

The displacement volume is encoded as a low-resolution base mesh M_B and a collection of per-vertex displacement directions $d_i \in D$, implicitly defining a closed volume within which the μ-mesh surface lies. In practice, the vertex positions of a base triangle and the corresponding displacement directions induce a "bilinear prismoid" whose top and bottom faces are triangles and the sides are bilinear surfaces, which also serves as culling primitive when building specialized BVHs that exploit the structure of μ-mesh [NVI22b].



Input Mesh Base Mesh A Base Mesh B

Figure 6.3: In this example, using base mesh A is drastically preferable to base mesh B, as it results in a smaller displacement volume, and smaller quantization and aliasing issues. One way to detect this situation is to observe that the top vertex of base mesh B has a much lower visibility score (Section 6.4.1).

Together with the base mesh data, our construction also produces per-triangle refinement levels $s_b \in S_B$, edge decimation flags to ensure the final μ-mesh is watertight, and scalar micro-vertex displacements.

6.3.1 Method

Our method to generate a high-quality μ-mesh from a high-resolution input mesh is organized into three steps. We first generate the coarse base mesh geometry $M_B = (V_B, F_B)$, making no assumptions on the topology or regularity of the input mesh; then, we refine and displace the base mesh to obtain the high-resolution μ-mesh; finally, we optimize the μ-mesh attributes to shrink the displacement volume around the input surface.

Base mesh We generate the base mesh starting from the high-resolution input, iteratively coarsening the mesh using edge-collapse operations. Our approach exploits the robustness of mesh simplification algorithms based on minimal-quadric edge collapses, adapted to produce a coarse base mesh tailored to our needs and the constraints imposed by the μ-mesh data structure.

Uniform micro-sampling The regularity of the base mesh refinement used to generate the micro-geometry produces significant anisotropy in the micro-vertex distribution when base triangles are far from equilateral. This is undesirable, as it results in unnecessary oversampling in the μ-mesh, negatively affecting both performance and storage requirements: on the one hand, the on-the-fly subdivision of the base mesh geometry will output an excessive amount of micro-geometry; on the other, micro-displacements for this "oversampled" micro-mesh will also increase the storage requirements. For this reason, we constrain the aspect ratio of the base mesh triangle produced by the simplification algorithm and introduce a cost term to penalize the creation of badly shaped triangles.

Surface reproducibility with minimal displacement volume As discussed above, the μ-mesh surface is constrained within a displacement volume limited by two surfaces $M_L = (V_L, F_B)$

and $M_U = (V_U, F_B)$, sharing the same connectivity F_B, which are implicitly defined by the base mesh M_B, the set of displacement directions D and the micro-displacement values. Although the minimization of the displacement volume to produce direction vectors of minimal magnitude (under the assumption of normalized micro-displacements $\delta_i \in [0, 1]$) is described later as part of the discussion on μ-mesh optimization, we observe that some base mesh configurations go against this goal and should be avoided. In particular, consider the situation depicted in Figure 6.3 where an edge collapse produces a sharp vertex that requires a large displacement vector magnitude to envelope the original surface within the resulting displacement volume. To this end, we design a decimation cost function that combines a geometric error term with a measure of "ease of reprojectability" to smaller prismoid volumes and better reproduction of the original surface features.

μ-Mesh attributes After generating a high-quality base mesh, we proceed to compute the μ-mesh attributes required to produce the high-resolution representation of the input surface.

First, we determine refinement levels for base triangles, for which we will propose two options: one is to employ a constant refinement level, which paired with our adaptive mesh simplification scheme will produce an adaptive triangulation; another is to enforce approximately uniform micro-triangle areas, which will change the refinement level adaptively across the base mesh triangle. In this second case, care must be taken to ensure that the resulting topology is water-tight.

Next, we compute a set of displacement direction vectors. This is a critical step, and we will show how a careful choice of displacement direction can produce significantly better results than the naive choice of using vertex normals.

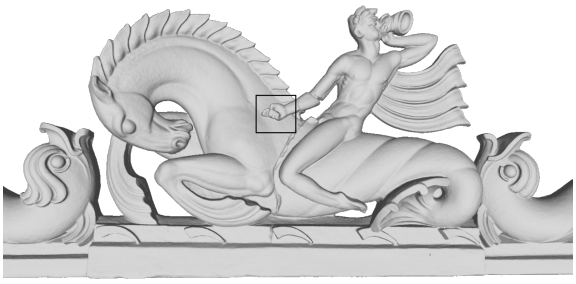
Once the μ-mesh topology and displacement vectors are fixed, we proceed to displace the micro-vertices. This is efficiently done with ray-casting supported by a bounding volume hierarchy (BVH) [Cla76] to accelerate ray-mesh intersection queries.

μ-mesh optimization Finally, we proceed to optimize the resulting μ-mesh to minimize the displacement volume according to the previously computed micro-displacement values. This ensures that, once the μ-mesh is converted to normalized micro-displacements, the volume bounding the micro-surface originating from each base triangle is minimized to improve the efficiency of the μ-mesh representation when using the prismoids as bounding primitives inside spatial data structures. Moreover, this is also helpful to reduce precision loss when the micro-displacements are encoded using integer representations.

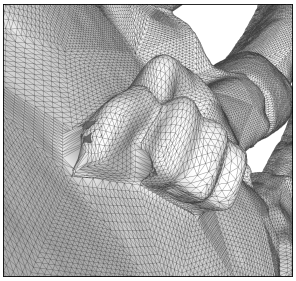
6.4 Determination of base displacement directions

Before detailing the base mesh construction, we introduce and discuss the choice of displacement direction vectors and the impact they have on the final displaced surface. The choice of base displacement directions to use is critical, as it implicitly defines the *displacement volume* that is spanned by the μ-mesh data structure, and within which the μ-mesh surface lies.

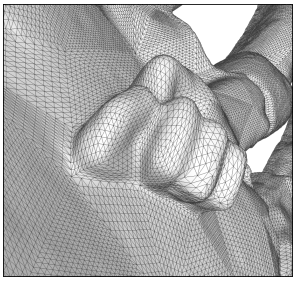
By construction, the displacement directions are generic 3D vectors and we are free to choose any orientation and magnitude; the μ-mesh data structure assumes normalized displacements, so the only practical constraint is that the magnitudes must be sufficiently large to reach the surface at any point. As we detail in Section 6.9, this can be easily achieved by adding a global



Input mesh (2M triangles)



(a) Vertex normals



(b) Optimal visibility

Figure 6.4: Choosing the wrong displacement directions can lead to visible geometric artifacts when the high-resolution surface is re-sampled by ray-casting along interpolated vectors. Left: results obtained with naive vertex normals (weighted average of the surrounding face normals). Right: results from the same base mesh using the optimal visibility directions computed as detailed in Section 6.4.1.

displacement to the base mesh and scaling the directions; in the following, we will assume the direction vectors are normalized and will focus only on their orientation in space.

Choosing the wrong direction vectors can result in noticeable artifacts, as exemplified in Figure 6.4 where the naive choice of using vertex normals as displacement directions produces self-intersections in the resulting μ-mesh. To avoid these situations, the displacement vectors should capture the local "extrusion" direction for base vertices, i.e. a set of vectors that such that their interpolated across a base triangle always points "outwards".

We observe that a sufficient condition for such direction vectors is that the normal vector n_f of a base face must have a positive dot-product with the directions d_i associated with its three vertices v_i. If that is the case, then it is easy to see that for any barycentric coordinates $w \in \mathbb{R}^3$ we have

$$n_f \cdot \sum_i w_i d_i = \sum_i w_i (n_f \cdot d_i) > 0. \tag{6.3}$$

Conversely, if some (but not all) d_j exists such that $n_f \cdot d_j \leq 0$ then the interpolated displacement directions degenerate within the triangle; i.e. there exists some w such that

$$n_f \cdot \sum_i w_i d_i = 0 \tag{6.4}$$

by continuity of the interpolation of the function $n_f \cdot d_i$ across the base triangle, which is negative when $i = j$ and positive when $i \neq j$. Equation 6.4 can be verified by two conditions: the interpolated direction either collapses to the zero vector or becomes parallel to the base triangle plane. Both cases are equally bad: in the first, the displacement volume collapses to a single point, strongly limiting the range of deformations that can be represented by displacing the base mesh; in the second, the interpolated direction becomes tangent to the base mesh (and nearly tangent in the neighborhood), and displacing along directions that are almost tangent to the base and input surfaces will introduce folds and self-intersection, as shown in Figure 6.4a.

Equation 6.3 gives a simple criterion for the selection of the displacement direction vectors: we seek a direction d_v at vertex v that has positive dot-product with all the normals n_f of its adjacent faces $f \in F_v$, i.e.

$$d_v \in \{d \in \mathbb{R}^3 \colon d \cdot n_f > 0 \; \forall f \in F_v\} = \mathbb{V}_v. \tag{6.5}$$

The set $\mathbb{V}_v$ appears in previous literature [KPW95, CMO97, AL08, JSZP20] and has been sometimes referred to as the *visibility* cone, as its vectors represent directions from which all the surrounding faces are visible without local occlusions. It is a (possibly empty) convex set that is obtained geometrically as the intersection of the half-spaces induced by the surrounding face planes and their normal vectors.

6.4.1 Finding the optimal visibility direction

Given a visibility cone $\mathbb{V}_v$ at vertex v, we define as the *optimal visibility direction* the unit cone vector that maximizes the minimum dot-product with any of the surrounding faces, similarly to [JSZP20].

Formally, the vector we seek is the solution of the following optimization problem

$$\mathcal{V}_v = \max_{d \in S^3} \min_{f \in F_v} d \cdot n_f, \tag{6.6}$$

where S^3 is the unit sphere and we will refer to the objective value $\mathcal{V}_v$ as the *visibility* of vertex v. If the visibility is negative, then the visibility cone is empty and, as discussed earlier, the interpolation of the displacement direction vectors will be locally degenerate regardless of the chosen directions. Otherwise, we obtain the "most outwards" direction relative to the surrounding faces.

We remark that the optimal visibility direction is unique only if the visibility is positive; in the case of negative visibility, we can construct configurations of normal vectors that admit more than one optimal solution (e.g. two sets of vectors symmetrically distributed around each pole of the unit sphere).

Finding a valid visibility direction is not trivial, and less so solving the optimization problem in Equation 6.6. [CMO97] resort to linear programming to find three vertices of the convex polyhedron obtained by intersecting $\mathbb{V}_v$ with a cube containing the origin, and interpolate these three vertices to find a visibility direction inside the cone, then normalize it; their method does not solve the problem to optimality and fails if the visibility cone is empty. [AL08] propose a brute-force solution which has quartic – $O(n^4)$ – complexity in the number of face normals, while [JSZP20] solve the optimization problem formulation directly with a constrained quadratic programming solver; both strategies are impractical to integrate into the iterative coarsening scheme that we use to generate a low-resolution base mesh.

We propose a novel (to our knowledge) iterative algorithm to quickly find the optimal positive visibility direction given a collection of normals, if one exists. Our algorithm is an adaption of Welzl's algorithm for computing the minimum enclosing disk [Wel91]; it is guaranteed to find the unique optimal solution if the vertex has positive visibility, and only a partial solution (i.e. a solution that is valid only for a subset of normals) otherwise. We will later use this information to guide the base mesh generation and avoid creating vertices with negative visibility.

Preliminaries It is well known that the minimum enclosing disk for a set of 2D points is either a point itself (if the point set is a singleton) or passes by either two or three points. A similar problem structure applies when seeking a minimum spherical cap enclosing a given set of unit vectors, as outlined in Figure 6.5, where the cap center is represented by a normalized vector and dot products (i.e. cosine distances) are used as distance metric on the sphere.

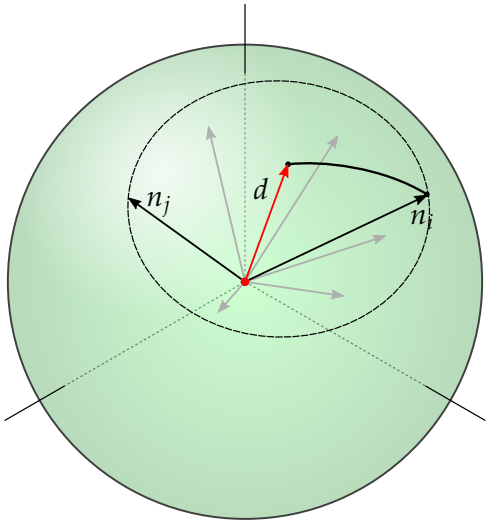


Figure 6.5: The minimum spherical cap enclosing a set of unit normal vectors can be represented as a direction d through the cap center and a support set of vectors that lie on the boundary of the cap. This problem setting shares similarities with the problem of finding the minimum enclosing disk on the plane, substituting the cosine distance between vectors for the euclidean distance between points [AL08].

Let $\mathcal{N}$ be the set of face normals for which we seek the optimal visibility direction. We represent a smallest-enclosing spherical cap as a direction d that points towards the cap's pole and a *support* set $\mathcal{S}_d$ of vectors that intersect the boundary of the cap; the cap radius is the dot product of d with any of the normals in the support. For $|\mathcal{N}| = 1$ and $|\mathcal{N}| = 2$ computing the smallest cap is trivial (either the support contains only a single normal or two, in which case the optimal visibility direction is the normalized average). For $|\mathcal{N}| = 3$ the support $\mathcal{S}_d$ can include two or three normals: if any cardinality 2 support includes the third normal, then we take the smallest corresponding cap; otherwise, the unnormalized optimal visibility direction can be found as [AL08]

$$d = (n_j - n_i) \times (n_k - n_i), \quad \mathcal{S}_d = \{n_i, n_j, n_k\}. \tag{6.7}$$

Optimal visibility for generic $\mathcal{N}$ sets In the general case, our algorithm proceeds iteratively by growing the smallest enclosing spherical cap until it contains all the normals in the set $\mathcal{N}$ or determines that a half-space containing them does not exist. We begin by initializing the smallest cap with any normal vector in $\mathcal{N}$. Then, at each iteration, we detect an outlier vector n_o and compute a new smallest enclosing disk for n_o and all the vectors currently in the support. This sub-problem is made easier by observing that n_o must necessarily lie on the border of the new smallest cap [Wel91], i.e. it must be necessarily included in the updated support at the next iteration; therefore, only a limited number of support normal combinations must be considered. If $S_d = \{n_i\}$ then we simply take $\{n_i, n_o\}$ as the new support; if $S_d = \{n_i, n_j\}$ then there are three supports that include n_o: two of cardinality 2 and one of cardinality 3; if $S_d = \{n_i, n_j, n_k\}$ then there are six supports that include n_o: three of cardinality 2 and three of cardinality 3. In all cases, we must pick the one that yields the smallest cap that encloses all the normals in S_d.

The algorithm stops as soon as it finds a spherical cap that encloses all the vectors, or if it detects that one such cap does not exist. The latter happens whenever either the new cap is smaller than the previous (meaning that the cap has "wrapped around" the previous visibility

Algorithm 2 OptimalVisibilityDirection (OptVis)

Input: Normal vectors $\mathcal{N} = \{n_0, \ldots, n_m\}$
Output: Optimal visibility direction d
1: Initialize the support $S = \{n_0\}$
2: **while** An outlier normal n_o exists **do**
3: $S' \leftarrow$ Support set of OptVis($\{n_i | n_i \in S\} \cup \{n_o\}$)
4: **if** $S' = \emptyset$ OR The visibility cap radius decreased **then**
5: Stop (the visibility cone is empty)
6: **end if**
7: $S \leftarrow S'$
8: **end while**
9: Return the optimal visibility direction for S as d

direction), or if it fails to find a new support that both includes n_o and all the previous support normals. Observe that the cap radius is monotonically increasing: at each iteration we start from the smallest cap for a given support, find a normal that is not included in the cap, and update the support so that it includes the vectors in the old support *plus* the new outlier: it is easy to see that the radius cannot decrease (i.e. the visibility score cannot increase) unless the previous cap was not minimal for the previous support set.

The pseudo-code for computing the optimal visibility direction is reported in Algorithm 2. We do not provide a formal complexity analysis here, although we remark that Welzl's algorithm works in expected linear time and our algorithm employs a similar mechanism. This algorithm is extremely quick in practice and computes valid directions in all instances where the intersection of the half-planes induced by the faces is non-empty and not near-degenerate, as shown in Figure 6.4 (bottom right).

With the mechanism to compute the displacement directions explained, we now proceed to discuss our approach for constructing μ-Meshes.

6.5 Base mesh generation by iterative coarsening

Our strategy to construct the base mesh is to perform a sequence of local operations on M, progressively turning it into M_B and M_T. We augment a greedy mesh decimation algorithm based on edge-collapses driven by quadric error metrics [GH97] by adapting the greedy cost function and adding decimation constraints to control the isotropy of the resulting micro-triangulation and ensure valid displacement directions always exist. The quadric initialization follows the usual scheme outlined in Section 2.4.1.

Our simplification strategy optimizes for the isotropy of the refined μ-mesh. We measure the isotropy of a triangle as its aspect ratio, which we define as the ratio of twice the inradius to the circumradius; this and similar or related definitions have been proposed in previous literature to measure the quality of a surface triangulation [Fie00]. The aspect ratio of a face f ranges from 0 for degenerate triangles to 1 in the case of perfectly equilateral triangles and is easily computed as

$$\mathrm{AR}(f) = \frac{16 \cdot \mathcal{A}(f)^2}{\mathcal{P}(f) \cdot |e_0| \cdot |e_1| \cdot |e_2|}, \tag{6.8}$$

where $\mathcal{A}(f)$ and $\mathcal{P}(f)$ denote the area and perimeter of face f respectively, and e_i the face side lengths.

Isotropy-guided μ-mesh creation Since the μ-mesh refinement is obtained by regular subdivision, the aspect ratio of a base face is inherited by all its micro-faces, except for ones adjacent to base edges that specify the decimation flags. Note that, in theory, one should seek to maximize the aspect ratio of the micro-faces *after* the displacement; however, this would result in a very complex system where the simplification, subdivision, and displacement steps are interconnected. Instead, we choose to optimize the isotropy of the base-mesh triangulation during the simplification process, under the assumption that if the base surface is a good approximation of the high-resolution input surface then the isotropy of the micro-surface triangulation will *not* be severely affected by the displacement process.

Greedy cost function Given a candidate edge $e = (u, v)$ to collapse and the new vertex placement v', the greedy cost function we use combines the geometric error term Q_e obtained by summing the two vertex quadrics with three weighted penalty terms AR_e, V_e and N_e; these terms are in the range [0, 1] and discourage the local loss of isotropy, visibility, and surface orientation respectively. The cost function we use, shown in logarithmic space to highlight its additive and multiplicative structure, is

$$\log(Q_e(v')) - \alpha \log(\mathrm{AR}_e(v')) - \beta \log(\mathrm{V}_e(v')) - \gamma \log(\mathrm{N}_e(v')). \tag{6.9}$$

The term AR_e is the minimum aspect ratio resulting from the edge collapse, guiding the simplification towards producing isotropic triangles. The term V_e estimates the local visibility after the collapse by computing the minimum visibility score of the resulting face normals *with respect to the existing visibility directions* (we use this approximation to avoid solving a visibility problem for all the vertices in the 1-ring of each candidate edge collapse, which slows down the decimation algorithm by a factor of 3). The visibility score is clamped to [0, 1] to rule out the introduction of negative-visibility vertices. Finally, the term N_e measures the maximum deviation of the face normals produced by the collapse as the minimum dot product normalized in the range [0, 1]. We let $\log(0) = -\infty$ and reject as unfeasible any collapse with a cost equal to ∞.

The weighting coefficients control the relative importance of the penalty terms; for our experiments, we use $\alpha = 0.5$, $\beta = 0.5$, and $\gamma = 0.1$.

6.5.1 Bounded anisotropy

To allow a stricter control on the isotropy of the μ-mesh triangulation, we also optionally bound the aspect ratio of the faces that are generated by the simplification process.

The aspect ratio bound is enforced by forbidding an edge collapse if it produces any face with an aspect ratio below a prescribed threshold, *unless* the aspect ratio of that triangle improves. Unfortunately, while this guarantees to produce a base mesh with the desired level of isotropy, it can also dramatically affect the decimation level. In particular, the resulting base mesh can contain clusters of triangles that become locked by the anisotropy bound early on and fail to get decimated to a satisfactory level; an example of this situation is reported in Figure 6.6. To counter this, we propose two workarounds: adaptive aspect ratio thresholding and smoothed edge collapses.

Adaptive aspect ratio thresholding To relax the aspect ratio constraints we introduce a tolerance parameter ϵ to allow faces that are already *below* the specified threshold to decrease their aspect ratio without preventing an edge collapse. Effectively, this changes the aspect ratio



1.5M Faces — 15K Faces, $\tau_{AR} = 0.4$

Input mesh — Simple bound — Smoothed collapses
Adaptive thresholding

Figure 6.6: Decimating a 1.5M faces input mesh to 15K faces with an aspect ratio threshold of 0.4. Using a simple bound produces locked triangle clusters early in the simplification, negatively affecting the adaptivity of the simplified base mesh. Conversely, smoothed edge collapses and adaptive thresholding are effective in escaping such "local minima", leading to a much better approximation of the input surface. Here we use $\epsilon = 0.1$ and $\lambda = 0.1$. *Mesh taken from [Thr22]*

threshold τ_{AR} from a global value to a local, per-triangle value. To find this value for a triangle t when evaluating the feasibility of an edge collapse, we keep a record of the maximum aspect ratio achieved by that triangle and compute the minimum between the global threshold and the adaptive local threshold

$$\tau_{AR}(f) = \min\left(\tau_{AR}, \mathrm{AR}_{max}(f) - \epsilon\right). \tag{6.10}$$

Smoothed edge collapses Isotropy of M_B is crucial for us, therefore we modify the edge collapse operation toward fulfilling this goal, by tweaking the position of the vertex generated by every collapse to make the resulting faces more regularly shaped, compatible with other objectives. Specifically, given a candidate edge $e = (u, v)$, we first compute the edge quadric Q_e as usual by taking the weighted sum of the two vertex quadrics Q_u, Q_v. Then, we compute the barycenter of the collapsed edge (i.e., the average position of the union of neighbors of u and v), and apply tangential smoothing [BK04] by projecting it onto the tangent planes at both u and v. Of the two smoothed positions, let p_s be the one yielding the lower geometric error as measured by the edge quadric Q_e, which we use to define a "pulling" quadric that measures the squared distance from it

$$Q_s(x) = \|x - p_s\|_2^2. \tag{6.11}$$

We then combine this pulling quadric with the original edge quadric using a weighting parameter λ

$$Q'_e = Q_e + \lambda Q_s, \tag{6.12}$$

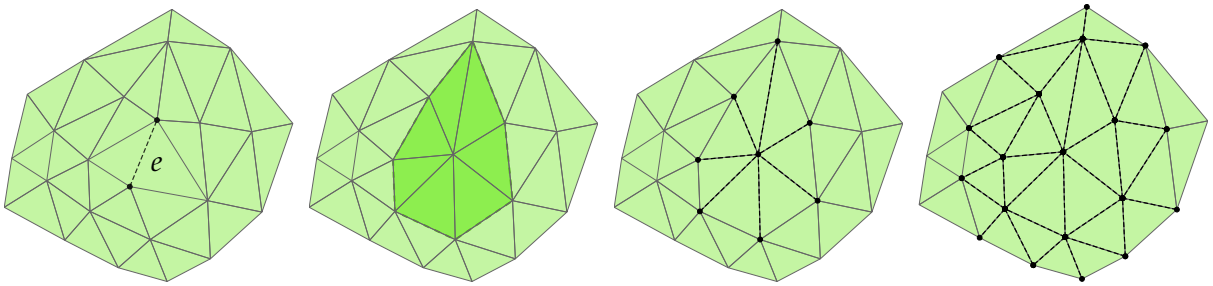


Figure 6.7: How an edge collapse affects other collapses in presence of aspect ratio and face normals cone bounds. (a) Collapse Operation. (b) Faces that change their normal and aspect ratio as a result. (c) Collapses that must be re-evaluated because the edge quadrics have changed. (d) Collapses that must be re-evaluated because the previous cost and feasibility have become obsolete due to changes to the aspect ratio, orientation, and visibility of the surrounding faces and vertices.

and set its minimizer as the vertex position produced by the edge collapse. In our experiments, we use $\lambda = 0.1$.

The combination of adaptive thresholding and smoothed edge collapses allows us to decimate triangle clusters that would otherwise be locked by the aspect ratio bound, as shown in Figure 6.6.

6.5.2 Collapse updates

The additional penalty terms included in the greedy cost function, and the presence of the aspect ratio constraints, affect the cost and feasibility of a larger number of candidate edge collapses compared to a traditional unconstrained simplification driven by the geometric error alone.

Consider a collapse operation acting on any edge e, as shown in Figure 6.7a. After collapsing the edge, the aspect ratio and normal vector of any face connected to the new vertex change, as well as the visibility score of any of its referenced vertices (Figure 6.7b).

With traditional edge collapses, one simply updated the quadratic error function stored at the new vertex, which only affects the cost function of collapses involving edges directly connected to the new vertex (Figure 6.7c). When including our penalty terms and feasibility constraints, we must also re-evaluate collapses that were previously deemed unfeasible and whose cost and outcome may change as a result of the current edge collapse. These involve any edge that is connected to at least one neighbor of the collapse vertices (Figure 6.7d), since their local neighborhood partly overlaps with the neighborhood of the current collapse, i.e. some of the neighboring faces will be altered by the current operation.

Unfortunately, this significantly increases the number of queue entries that must be updated after each collapse, approximately by a factor of 4, and similarly impacts the running time of the simplification process. Things are further complicated by the fact that we cannot re-use the previously computed position for these edges because of the smoothing we apply: this also depends on the local edge neighborhood, which will always include the vertex relocated by the collapse that triggers a queue update. To mitigate this effect when dealing with very high-resolution input meshes, we propose a faster decimation scheme that does not rely on a

global queue and is used as a pre-processing step to speed up the base mesh generation.

6.6 Proxy-mesh generation

The additional safeguards and error-checking mechanisms we include in the mesh decimation scheme to produce a high-quality base mesh have two negative effects on the overall performance of the mesh simplification algorithm:

- First, the aspect ratio constraints are input-sensitive, so they may still reject some initial vertex configurations, leaving high-resolution vertex clusters in the input mesh despite employing smoothed edge collapses;
- Second, they introduce a non-negligible slowdown compared to a standard quadric-based mesh simplification algorithm because their influence is not restricted to the 1-ring of the collapsed edge, but also on all the edges adjacent to any face in the 1-ring, for which the feasibility and cost of the collapse must be updated in the global greedy queue.

For these reasons, we propose to introduce a support mesh where we perform preliminary processing before decimating it with our tailored base-mesh generation algorithm. We will refer to this support mesh as "proxy" mesh, since it is used in place of the original high-resolution mesh as input for the base mesh generation step.

Pre-filtering To counter the first issue, we filter the input geometry to improve the compatibility of the input surface with the prior assumptions that back our introduction of additional constraints for the base mesh generation. We apply a combination of topological and shape filtering, to optimize the regularity of the triangulation and the smoothness of the surface while preserving salient and high-curvature shape features.

Fast mesh decimation To reduce the computation time, we adopt a randomized mesh simplification scheme that samples collapses randomly, subject to an adaptive error threshold. This approximate scheme exploits the redundancy in the geometric representation of high-resolution and extremely high-resolution meshes, trading simplification accuracy for speed to quickly decimate the proxy mesh to a default triangle count before feeding it to the constrained base mesh generation procedure.

As previous literature showed, there exist multiple sequences of local simplification operations that lead to the same decimated surface [Pup98]. This stems from the independence of local operations that do not share any primitive from their local support (e.g. for an edge collapse the union of the 1-ring of the edge vertices), so there exists a partial order among all possible local operations and any total order that is compatible with this partial order leads to the same decimated mesh topology. The randomized scheme we employ does not guarantee that we can recover the base mesh generated by the global greedy sequence when we produce a base mesh using the proxy mesh as input. However, since we are dealing with very high-resolution geometry, we expect (small) perturbations of the optimal collapse sequence to have a negligible effect on the final base mesh if the proxy mesh triangulation is kept at a reasonably high resolution. In our experiments, we keep the face count for the proxy mesh at 1M regardless of the original mesh resolution.

Algorithm 3 Proxy Mesh Decimation

Input: M, $n_{topology}$, n_{iso}, n_{aniso}, ϵ_{max}, f_{min}
Output: Decimated proxy mesh $M_P = (V_P, F_P)$

```
Initialize M_P ← M
Apply n_topology topology optimization passes
Apply n_iso Laplacian Smoothing passes
Apply n_aniso Anisotropic smoothing passes
Compute proxy mesh attributes from the filtered surface
ϵ ← Initial guess for the adaptive error threshold
while |F_P| > f_min do
    Sample k random edges and choose the edge e one with the smallest collapse cost ϵ_e
    if The collapse is feasible & ϵ_e ≤ ϵ then
        Collapse edge e
    end if
    Adjust the adaptive threshold ϵ or stop
end while
return M_P = (V_P, F_P) and the proxy vertex quadrics Q_P
```

6.6.1 Topological and shape filtering

First, we perform edge flipping to improve the triangulation isotropy and regularity of vertex valences in the input mesh. We apply a topological optimization pass over the entire mesh, flipping edges that result in a more regular diagonal split of the quad induced by the corresponding edge flap (pair of faces that share a common edge). We restrict flipping to near-planar quads, i.e. edge flaps where the normal vectors of the two faces are no more than 30 degrees apart, and flip the edge if it reduces the maximum quad angle.

Next, we filter the mesh geometry to improve smoothness and prepare the surface for downstream processing. The smoothing filter we use has two stages: the first stage is a simple laplacian smoothing that is used to filter out high-frequency noise and artifacts that may interfere with the cone-angle constraint; in the second stage, we employ an anisotropic smoothing filter to further smooth out high-frequency small-scale detail while preserving and sharpening salient shape features and creases that may otherwise be attenuated by an overly-aggressive isotropic smoothing.

For the second stage, we employ a simple yet effective strategy which consists in smoothing the input vertex quadrics and relocating each vertex at the minimizer of the locally smoothed quadric. This approach has been proposed before to achieve feature-preserving mesh denoising and smoothing [VNMC10, LTB19].

After pre-filtering the input mesh, we then employ a fast decimation scheme to quickly reduce the geometric complexity of the mesh in the range of 500K - 2M faces. In the downstream processing, the proxy surface completely replaces the input surface, meaning all the necessary attributes (e.g. the vertex quadrics) are taken from the proxy surface as well.

6.6.2 Fast mesh decimation with random collapses

The proxy mesh decimation procedure is outlined in Algorithm 3. The main idea is to replace the global ordering of the greedy edge collapse operations with a randomized sequence of collapse operations subject to an adaptive and monotonically increasing error threshold ϵ. The

algorithm takes as input the number of pre-filtering passes $n_{topology}$, n_{iso}, n_{aniso}, the maximum geometric error tolerance ϵ_{max} and the target face count f_{min} for the decimated proxy mesh.

We first pre-filter the input mesh as described in Section 6.6.1, and use the filtered mesh to compute the vertex error quadrics. Next, we compute an initial guess for the error threshold by picking 50 random edges and computing the smallest cost among the chosen sample. This initial guess is quite conservative, and the algorithm will increase the threshold as the decimation progresses to approximate the behavior of a global priority queue.

The algorithm then proceeds to decimate the proxy mesh to f_{min} faces by sampling k edges from the mesh and selecting the one with the smallest collapse cost as the next candidate collapse; in all our examples we use $k = 5$. If the cost is below the current threshold ϵ, the algorithm tests the feasibility of the collapse (e.g. topological and orientation consistency, but we can also incorporate the μ-mesh decimation constraints here) and proceeds with the collapse or discards the operation accordingly.

Error threshold adjustment and early stopping The error threshold adaption is controlled by tracking the number of successive iterations that fail to decimate an edge. If too many consecutive iterations are unsuccessful we double the error threshold and reset the counter; the increase continues until the threshold reaches the user-specified upper-bound ϵ_{max}. If the target face count is never reached, no valid collapse will be left on the mesh either because they are unfeasible or because they are above the maximum error tolerance. For this reason, the algorithm also has an upper bound on the number of failed iterations which, if reached, interrupts the proxy mesh decimation loop. In our experiments, we increase the error threshold after 20 failed consecutive iterations and interrupt execution after 500 failed iterations. In practice we never reach the interruption threshold as the target face count we use for the proxy decimation is not sufficiently low.

6.7 Determination of base-mesh refinement level

After computing the coarse base geometry, we generate the high-resolution micro-geometry that is used to re-sample the input surface. In this section, we outline two different strategies to refine the base mesh according to a user-specified target micro-face count $|F_\mu|$. Obviously, μ-meshes we can adopt a single global subdivision level l, producing a micro-face count that is equal to

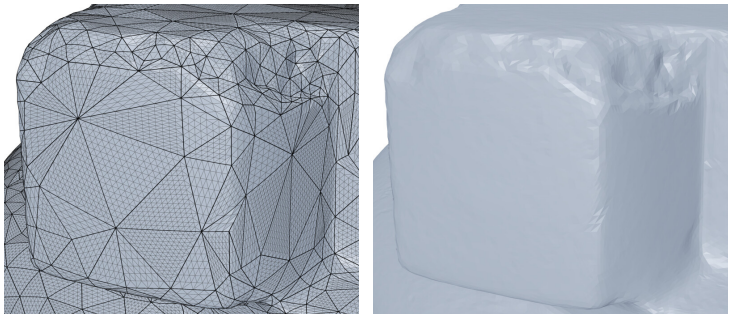
$$|F_\mu| = 2^{2l} \cdot |F_B|. \tag{6.13}$$

However, in this section we outline two refinement schemes that take advantage of the possibility to control the μ-mesh tessellation levels across the base mesh faces with the edge decimation flags, allowing a difference of ± 1 in the subdivision level of adjacent faces. In both cases, we first specify the subdivision levels freely and later adjust them to satisfy the "off-by-one" constraint, as detailed in Section 6.7.2.

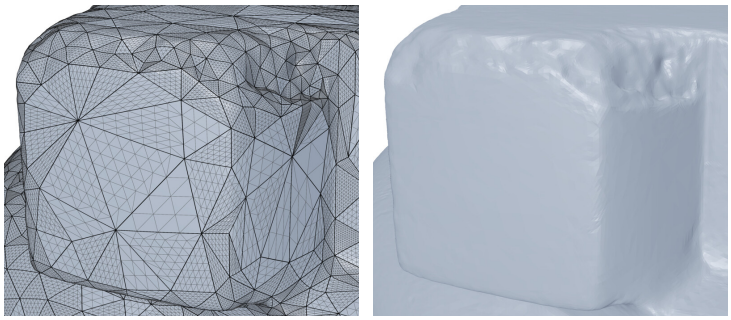
We propose two alternative strategies to select base-face subdivision levels that target either uniform micro triangle sizes or adaptive triangle sizes tailored to the input shape complexity. A comparison of the two is shown in Figure 6.8.

6.7.1 Uniform micro-face area

With this refinement strategy, we aim to generate micro-triangles of roughly equal area.



(a) Uniform refinement



(b) Adaptive refinement

Figure 6.8: Comparison between uniform and adaptive refinement. Uniform refinement is oblivious to the high-resolution surface, while adaptive refinement allocates more micro triangles to better reproduce salient features and small-scale surface detail.

As stated above, a regular subdivision of a triangle at level l produces 2^{2l} micro-triangles. This rule can be used to retrieve a *fractional* global subdivision level that would produce the desired number of micro-triangles, by solving equation 6.13 for l. Then, the continuous global subdivision l can be derived as

$$l = \frac{1}{2} \log_2 \left(\frac{|F_\mu|}{|F_B|} \right). \tag{6.14}$$

Once the global subdivision is found, we can find the target micro-triangle area a_m as

$$a_m = \frac{\mathcal{A}(M_B)}{|F_\mu|} = \frac{\mathcal{A}(M_B)}{2^{2l} \cdot |F_B|}, \tag{6.15}$$

where $\mathcal{A}(M_B)$ is the total area of the base mesh. Then, for a given base triangle b, its estimated number of micro-triangles is simply the ratio of its area a_b to the average micro-triangle area:

$$|F_\mu^b| = \frac{a_b}{a_m}. \tag{6.16}$$

From this, we set the fractional subdivision level s_b of a base triangle b as

$$s_b = \frac{1}{2} \log_2 |F_\mu^b| = l + \frac{1}{2} \log_2 \left(\frac{a_b}{a_B} \right), \tag{6.17}$$

which has a straightforward interpretation as the global subdivision level l *plus* a correction factor that depends on the base triangle area a_b relative to the average base triangle area $a_B = \mathcal{A}(M_B)/F_B$; this correction term accounts for base triangle area differences in producing micro-triangles of uniform area.

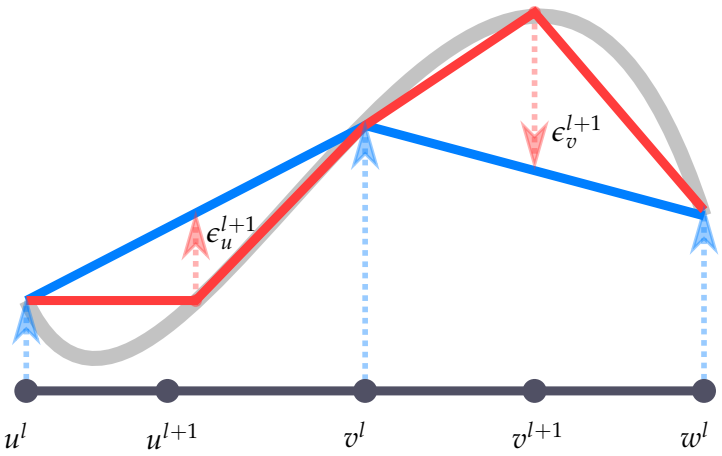


Figure 6.9: Error estimation for adaptive refinement. The current subdivision level l and the displaced micro-geometry are shown in blue. The displacements of the micro-vertices added at level $l+1$ are shown in red, and the error ϵ of *not up-sampling* is the distance of a new micro-vertex from the current μ-mesh surface.

Finally, s_b is rounded to the nearest integer to find the actual subdivision level used to refine the base mesh faces and produce the μ-mesh geometry.

6.7.2 Adaptive refinement

Adaptive refinement uses an iterative algorithm to determine subdivision levels of base triangles according to the displacement error incurred by the actual displacement of the micro-vertices.

This refinement strategy iteratively increases subdivision levels by measuring the geometric error that is induced by virtually up-sampling and displacing a base triangle, and then down-sampling it to its current level. The geometric error is measured as the average distance of the extra micro-vertices relative to the current micro-displaced surface, as shown in Figure 6.9.

The procedure to compute the adaptive subdivision levels is outlined in Algorithm 4 and is controlled by two parameters: the target number of micro-faces $|F_\mu|$ and the maximum tolerance for the displacement error ϵ. We use a priority queue to order the base triangles according to the geometric error induced by their current displacement level s_b; base subdivision levels are progressively increased until either the desired number of micro-faces is reached or the next displacement error is below the maximum tolerance.

6.7.3 Adjusting subdivision levels

Both uniform-area and adaptive refinement consider on base triangles in isolation and can produce T-junctions in presence of arbitrary jumps in the subdivision level of adjacent triangles. After the subdivision levels have been computed, we apply a correction pass to ensure the levels match across base edges.

The correction pass can be done quickly with a simple iterative algorithm that pools subdivision levels on edges and propagates the maximum to the adjacent faces until they all differ by at most one level. This strategy is extremely simple to implement and converges in just a few passes. After this, we set the edge decimation flags are set to obtain the final base mesh subdivision data.

Algorithm 4 Adaptive refinement

for all Base triangles **do**
    $s_b \leftarrow 0$
    Compute the displacement errors ϵ_b of down-sampling from level $s_b + 1$
    Initialize the priority queue Q with faces ordered by *decreasing* down-sampling error
end for
while $\mathcal{Q} \neq \emptyset$ or F_μ has been reached **do**
    Extract the next base triangle b.
    if ϵ_b is below the threshold **then**
        Stop
    else
        $s_b \leftarrow s_b + 1$
        Update ϵ_b by evaluating the down-sampling error from the next level
        Re-insert b in Q with the updated error.
    end if
end while

6.8 Determination of micro displacement values by raycasting

Once the base mesh geometry is defined, we find the micro-vertex coordinates by ray-casting against the original high-resolution surface. We use a BVH of axis-aligned bounding boxes to speed up the scalar displacement computations, which is significantly faster than generating the base mesh even for fine-grained micro-geometry tessellations.

To ensure the ray-casting algorithm is robust to self-intersections and other geometric artifacts, we adopt standard mitigation techniques such as only intersecting the high-resolution surface at faces that are coherently oriented with the ray direction and discarding intersections with different connected components. We also exploit the smoothness of the scalar displacement field to improve robustness to outlier and missed intersections. If we detect intersection outliers (e.g. rays that travel too far before hitting the input surface), we clear the corresponding scalar displacements and replace them by interpolating the displacement distances from valid neighbors.

6.8.1 Borders and tangential displacements

For open surfaces, correctly reproducing the input mesh border poses two significant challenges. First, it is much more probable that ray casting fails at border micro-vertices because the base mesh does not accurately reproduce the original border curve, likely resulting in ray misses and incorrect intersections. Second, even if the displacement rays intersect the input surface, the orthogonal displacement direction will still cause the μ-Mesh border to follow the base mesh curve instead of the input border curve.

To improve the robustness of ray casting at border micro vertices, we augment the input surface with a border "band" when computing displacement for border micro vertices. We experimented with two alternatives. The first solution is to use a *parallel* band that simply extends the input surface at the border to ensure that misalignments with the base mesh result in missed or wrong intersections. The second solution is to intersect against an *orthogonal* band that follows the original border curve by defining special *tangent* displacement directions at border vertices, oriented away from the mesh surface.

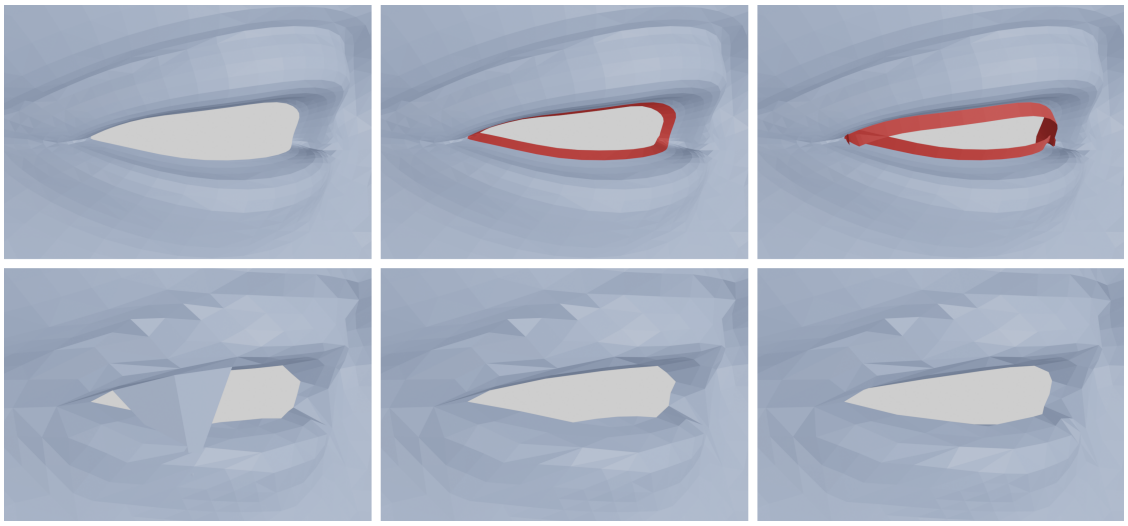


Figure 6.10: Accurately reproducing mesh borders can be challenging, due to the lack of robustness in the ray-casting process. Without augmenting the border with a support band, incorrect intersections can be difficult to detect resulting in displacement artifacts (left). Adding a horizontal prolongation band (middle) improves robustness, but accurately reproducing the border curve remains difficult. Using a vertical band and tangential displacements at mesh borders can more accurately reproduce the border curve (right).

We compare these solutions in Figure 6.10. In the first row, we show the input surface augmented with the border geometry in red, while in the second row, we show the corresponding displaced μ-meshes. Without introducing the additional geometry, the ray casting at border micro vertices is not robust. Adding a horizontal band avoids missed intersections, but the structure of the base mesh border remains visible despite a finely tessellated μ-mesh. Tangent displacements noticeably improve the smoothness of the border curve; however, for interior micro vertices that are near the border, the displacement direction is very close to being tangent to the surface, possibly leading to displacement artifacts as discussed in Section 6.4.

6.9 Determination of base displacement lengths

The μ-mesh data structure encodes the bottom side of the displacement volume at height zero and the top side at height one, i.e. $\delta_{min} = 0$ and $\delta_{max} = 1$ in Equations 6.1 and 6.2, which requires normalizing the scalar displacement field in the range $[0, 1]$. The prismoids originating from the base mesh and the displacement direction vectors can also be used as bounding primitive for intersection tests against the μ-mesh faces; therefore, the displacement direction lengths should be kept as short as possible to achieve a tight bounding of the displaced surface.

We discuss here two different solutions for normalizing the scalar displacements, by updating the base vertex positions and scaling the displacement directions.

The first one is a global min-max displacement and scaling that preserves the the displaced surface exactly. We find the minimum and maximum unnormalized displacements $\bar{\delta}_{min}$ and

$\bar{\delta}_{max}$ and update the base attributes as follows:

$$V'_B = V_B + \bar{\delta}_{min} D, \tag{6.18}$$

$$D' = (\bar{\delta}_{max} - \bar{\delta}_{min}) D. \tag{6.19}$$

Then, displacements are remapped inside the updated displacement volume as

$$\delta'_i = \frac{\delta_i - \bar{\delta}_{min}}{\bar{\delta}_{max} - \bar{\delta}_{min}}. \tag{6.20}$$

The drawback of using global scaling is that the prismoid heights are disentangled from the curvature of the displaced surface, while we would like to create shorter prismoids in low-curvature regions. Instead, a tighter displacement volume that adapts to the μ-mesh is two advantages: it improves the performance of primitive culling when using the prismoids within BVHs and improves precision when quantizing and compressing the scalar displacements.

The second normalization strategy we propose relies on local min-max displacement aggregation to determine the displacement direction lengths. We track the local minimum and maximum displacement at any base vertex v_j by aggregating the scalar displacement δ_i of any micro vertex u_i, from adjacent base faces, that maps to v_j with a positive barycentric coordinate. Then, we displace v_j and re-scale its displacement direction d_j analogously to Equations 6.18 and 6.19, but using the local information aggregated at vertex v_j instead of global minimum and maximum values:

$$v'_j = v_j + \delta^j_{min} d_j, \tag{6.21}$$

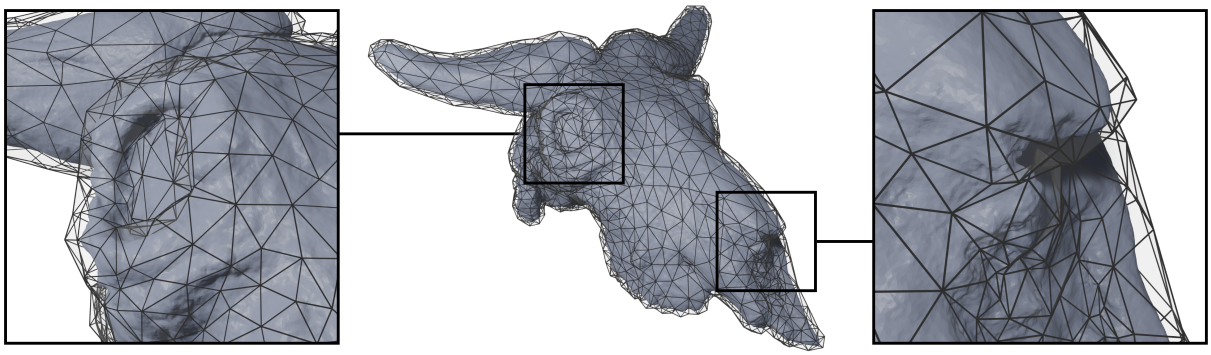
$$d'_j = (\delta^j_{max} - \delta^j_{min}) d_j. \tag{6.22}$$

Scaling the displacement directions independently instead of uniformly changes the orientation of the interpolated directions, therefore we cannot simply map the old displacement value to a new one that preserves the original micro vertex position as in Equation 6.20. We experimented both with iterating the displacement and normalization steps until reaching convergence (i.e., the global minimum and maximum displacements fall in the $[0, 1]$ range) and with projecting the old position onto the new displacement direction (bounding it within the updated displacement volume); we found the latter strategy to be more robust (and faster, as it does not require to iterate the ray-casting step), at the expense of introducing a negligible increase in the surface approximation error.

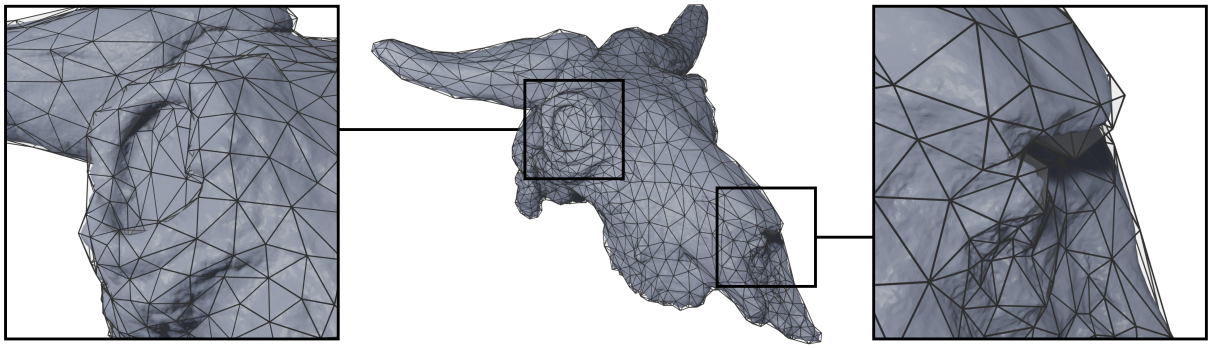
Figure 6.11 shows a comparison of the two normalization schemes, and reports the overall displacement volume measured by decomposing each base prismoid into 3 tetrahedra. As can be expected, the local scaling shrinks the displacement volume significantly and produces a much tighter cage around the μ-mesh surface.

6.10 Barycentric ARAP

In this section, we show how to optimize the base mesh parametrization to account for the distortion induced by the micro expansion of the base mesh surface, assuming a parametrization has been computed for the base mesh. The low geometric complexity of the micromesh support greatly simplifies the task of finding a good UV cut layout for the parametrization, which we use as starting point for our UV distortion minimization.



Global normalization, Volume = 2.150450×10^6 units^3



Local normalization, Volume = 6.29932×10^5 units^3

Figure 6.11: Displacement normalization with global and local scaling of the direction vectors. Global rescaling guarantees to preserve the original micro-vertex locations, but is much less effective in reducing the displacement volume, whereas the local min-max scalar displacement aggregation produces a much tighter envelope of the micromesh surface.

We reformulate the ARAP [LZX*08] energy to express the distortion of triangles whose vertices are expressed as convex combinations of base mesh vertices, under the constraint that the base mesh vertices belong to the same base face. Note that, in principle, our construction is not restricted to ARAP and can be extended to any geometric energy.

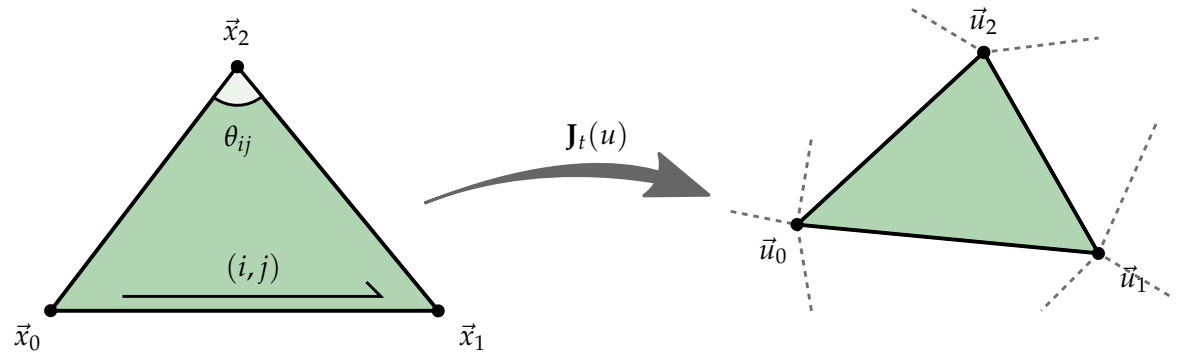
6.10.1 Background: ARAP parametrization on standard 3D meshes

We begin by recalling the local/global energy minimization scheme employed to minimize the ARAP energy on standard triangle meshes. For an in-depth description, we refer the reader to Section 2.2.7 and the original reference [LZX*08].

Let t be the index of a generic mesh triangle equipped with a 2D local isometric parametrization $x_t = (x_0^t, x_1^t, x_2^t)$ and, and let $u_t = (u_0^t, u_1^t, u_2^t)$ be the texture coordinates referenced by t. Moreover, let $\mathbf{J}_t(u)$ be the 2×2 Jacobian of the map from x_t to u_t, and $\mathbf{R}_t \in \mathcal{R}$ an auxiliary matrix encoding a 2D rotation.

The ARAP energy is formulated as

$$E(u, \mathcal{R}) = \sum_t \mathcal{A}(t) \|\mathbf{J}_t(u) - \mathbf{R}_t\|_F^2, \tag{6.23}$$



2D isometric projection (local) UV configuration (global)

Figure 6.12: Notation used for the ARAP energy formulation.

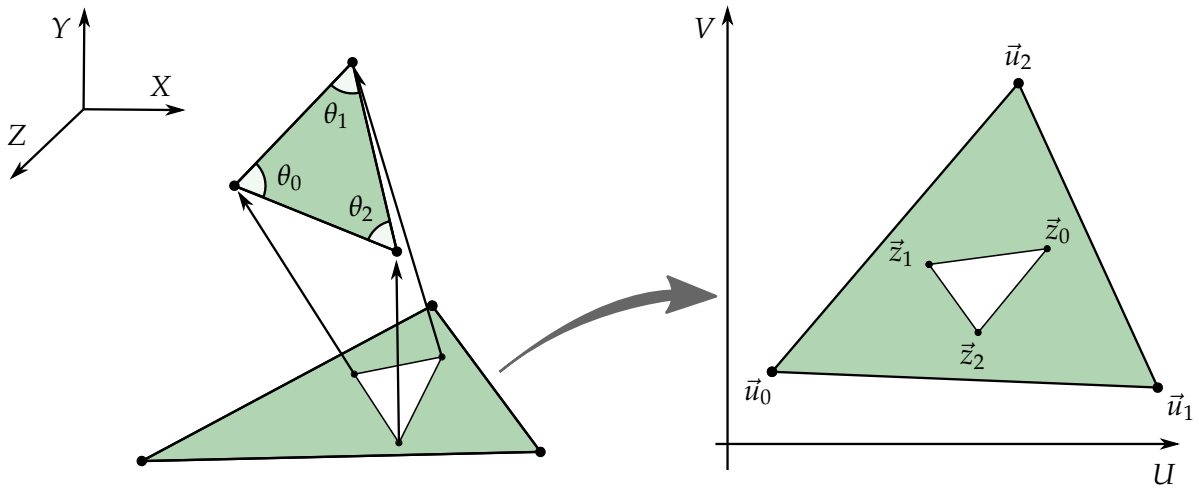


Figure 6.13: Notation for the barycentric ARAP energy formulation.

and equivalently, in terms of half-edges as

$$E(u, \mathcal{R}) = \frac{1}{2} \sum_{(i,j) \in he} \cot \theta_{ij} \| (u_i - u_j) - \mathbf{R}_{t(i,j)} (x_i - x_j) \|^2 \tag{6.24}$$

where $t(i,j)$ is the triangle to which half-edge (i,j) belongs and θ_{ij} is the opposite angle within $t(i,j)$ (Figure 6.12).

The ARAP energy is minimized with a local/global iterative scheme: in the *local* step, the parametric coordinates u are kept fixed, and the best fitting rotation matrices L are determined per-triangle by computing the SVD decomposition of $\mathbf{J}_t(u) = \mathbf{U}\Sigma\mathbf{V}^\top$ and setting $\mathbf{R}_t = \mathbf{U}\mathbf{V}^\top$; in the *global* step, the rotation matrices are kept fixed, yielding a quadratic energy in u (Equation 6.24) that can be minimized by setting the gradient to zero and solving the resulting sparse linear system.

6.10.2 Extension to micro-triangles

When a triangle is subdivided and displaced, micro-vertex attributes are inherited by interpolating base vertex attributes using barycentric weights. Consider a generic micro-triangle t in parameter space (Figure 6.13): its UV coordinates z_i are expressed as a linear combination of

the base UV coordinates u_0, u_1, u_2

$$z_i = \mathbf{U}_t w^i = \begin{pmatrix} | & | & | \\ u_0 & u_1 & u_2 \\ | & | & | \end{pmatrix} \begin{pmatrix} w_0^i \\ w_1^i \\ w_2^i \end{pmatrix}, \; i = 0, 1, 2 \tag{6.25}$$

where the base UV coordinates have been arranged as columns of a 2×3 matrix $\mathbf{U}_t$ multiplying the vector of the barycentric weights w^i.

In the local step, we compute the best-fitting rotation associated with each micro-triangle t. This is easily computed as the linear map between the isometric 2D projection of the *displaced* micro-triangle x_t and its interpolated parameter-space coordinates z_t.

When solving the *global* step, the energy term associated to a generic "micro" half-edge $(i, i+1)$ remains quadratic in u and can be written as

$$E(i, i+1) = \cot\theta_{i+2} \| (\mathbf{U}_t (w^i - w^{i+1}) - \mathbf{R}_t (x_i - x_{i+1}) \|^2, \tag{6.26}$$

with $\mathbf{R}_t$ now encoding the rigid alignment of the isometric 2D projection (x_0, x_1, x_2) of the *displaced* micro-triangle to its UV counterpart.

The gradient of the energy term (6.26) with respect to u_j is then

$$\nabla_{u_j} E(i, i+1) = 2 \cot\theta_{i+2} \left(\sum_{k=0}^{2} u_k^\top (w_k^i - w_k^{i+1}) - \mathbf{R}_t (x_i - x_{i+1}) \right) (w_j^i - w_j^{i+1}).$$
$$j = 0, 1, 2 \tag{6.27}$$

The global gradient can then be computed by accumulating the micro half-edge terms and set to zero to solve the global ARAP step with fixed L matrices.

We remark that the above derivation makes no assumptions on the regularity of the subdivision scheme used to generate the micro-geometry, and can be employed to optimize the parametrization of generic displacement-mapped meshes. In Figure 6.14, we show the efficacy of our formulation in mitigating the distortion induced by displacing the μ-mesh.

It is easy to see how a parametrization that is oblivious to the μ-mesh data structure can incur additional distortion when the micro-geometry is synthesized and displaced to produce the high-resolution surface. However, although our reformulation incorporates knowledge about the deformation induced by displacing the micro-geometry, the degrees of freedom of the system are still bounded by the number of base mesh primitives; this could result in an "overconstrained" system, where the optimization fails to produce a low-distortion map for the displaced surface. In practice, instead, we can drastically reduce the distortion and obtain a near-optimal map even with very coarse base meshes and UV atlas layouts. In the example of Figure 6.14, a 2M triangles input mesh has been decimated to 10K and then refined to obtain approximately 2M micro-triangles.

The initial UV coordinates have been computed with Blender [Ble22] and consolidated with [MCT21] to produce a very coarse texture atlas that is then optimized by solving an ARAP instance on the base mesh. The integrated base mesh ARAP energy is $1.31419 \cdot 10^{-2}$, increasing to $6.08024 \cdot 10^{-2}$ when computed on the displaced μ-mesh, and reduces to $1.56383 \cdot 10^{-2}$ when optimized with our reformulation.

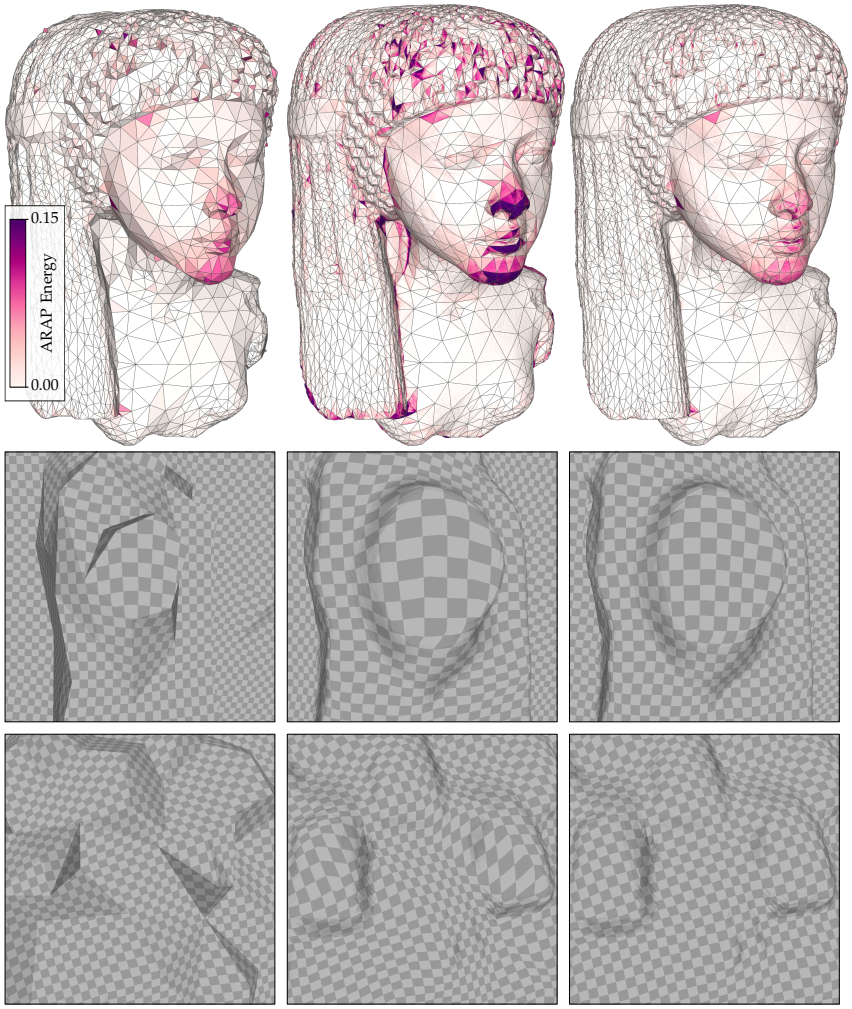


Figure 6.14: Using our barycentric ARAP formulation to minimize the distortion of the μ-mesh parametrization induced by interpolating base UV coordinates. We first minimize the ARAP energy on the base mesh (left), which still produces significant distortion when the parametrization coordinates are interpolated on the barycentric grid to get per micro-vertex UVs (middle). Reformulating the energy to account for the subsequent subdivision and displacement allows us to effectively optimize the mapping distortion of the final micromesh (right).

6.11 Results

We perform extensive quantitative and qualitative evaluations of our end-to-end μ-mesh generation framework, targeting high-resolution 3D scans and photogrammetry meshes.

Our approach inherits the robustness and efficiency of traditional decimation-based mesh simplification algorithms. All our experiments have been carried out on commodity hardware with 32GBs of RAM; as a reference, for a densely tessellated mesh of 34M faces the entire process takes less than 40 minutes, with the base mesh generation step taking up most of the time.

6.11.1 Compression and representation accuracy

We evaluate the efficiency of the μ-mesh representation at varying base mesh decimation rates by measuring the GPU storage requirements and geometric error with respect to the original input surface.

We estimate GPU occupancy by assuming the indexed triangle mesh representation uses 32-bit integers for the vertex indices and 32-bit floats for the vertex attributes. For a standard mesh, we assume the only vertex attribute is the 3D position so the total data size is obtained by summing the index (3 integers per face) and the vertex data (3 floats per vertex). For a μ-mesh, we measure the cost of storing the base mesh data as a standard mesh *plus* the cost of storing the displacement attributes, i.e. the per-vertex displacement directions (3 floats per vertex), the face subdivision data (1 byte per face, packing together 5 bits for the subdivision level and 3 bits for the edge decimation flags) and the micro vertex displacements (1 float per micro vertex).

To evaluate the geometric error, we compute the area-weighted average of the projection distance from the original surface to the μ-mesh surface, divided by the length of the input AABB diagonal to obtain a reference measure invariant to the object scale.

In Figure 6.15 we plot the data size and geometric error for varying decimation rates. For each data point, the corresponding μ-mesh is obtained by approximately restoring the original triangle count using uniform micro-triangle areas as the refinement strategy. Note that, for low base decimation rates, the overhead of storing the additional μ-mesh attributes increases the memory footprint. However, the data representation becomes more and more efficient at lower base mesh resolution while the geometric error remains approximately constant thanks to the densely tessellated and displacement-mapped micro-geometry.

Displacement quantization We can further reduce the storage cost by quantizing the micro vertex displacements. Since the values are normalized, we can encode them as fixed-width unsigned integers, further reducing the storage size. This can have a great impact on the storage requirements of finely tessellated μ-mesh from coarse base meshes, where most of the storage size is taken up by the scalar displacements themselves.

In Figure 6.16 we show the impact of bit quantization on the final displaced surface. As the visual comparison shows, we can drastically decrease the bit count needed to represent displacements without incurring in noticeable quantization artifacts: only at 4 bits per displacement (i.e. 16 uniformly distributed discrete steps between the lower and upper displacement surfaces) does the effect of quantization becomes distinctly visible. While the input surface complexity affects how detrimental the displacement quantization is, we remark that the displacement normalization with local scaling discussed in Section 6.9 is also beneficial in this regard, as a tighter lower and upper bound on the input surface naturally reduces the aliasing produced by more aggressive quantizations.

6.11.2 Level of details

One of the key advantages of representing high-resolution meshes with displacement maps is the inherent multi-resolution representation that can be achieved by controlling the subdivision of the control mesh before displacing it [SS09, LD08, BS08]. In Figure 6.17 we show the visual result of progressively reducing the resolution of the μ-mesh by decreasing the subdivision level of the base triangles.

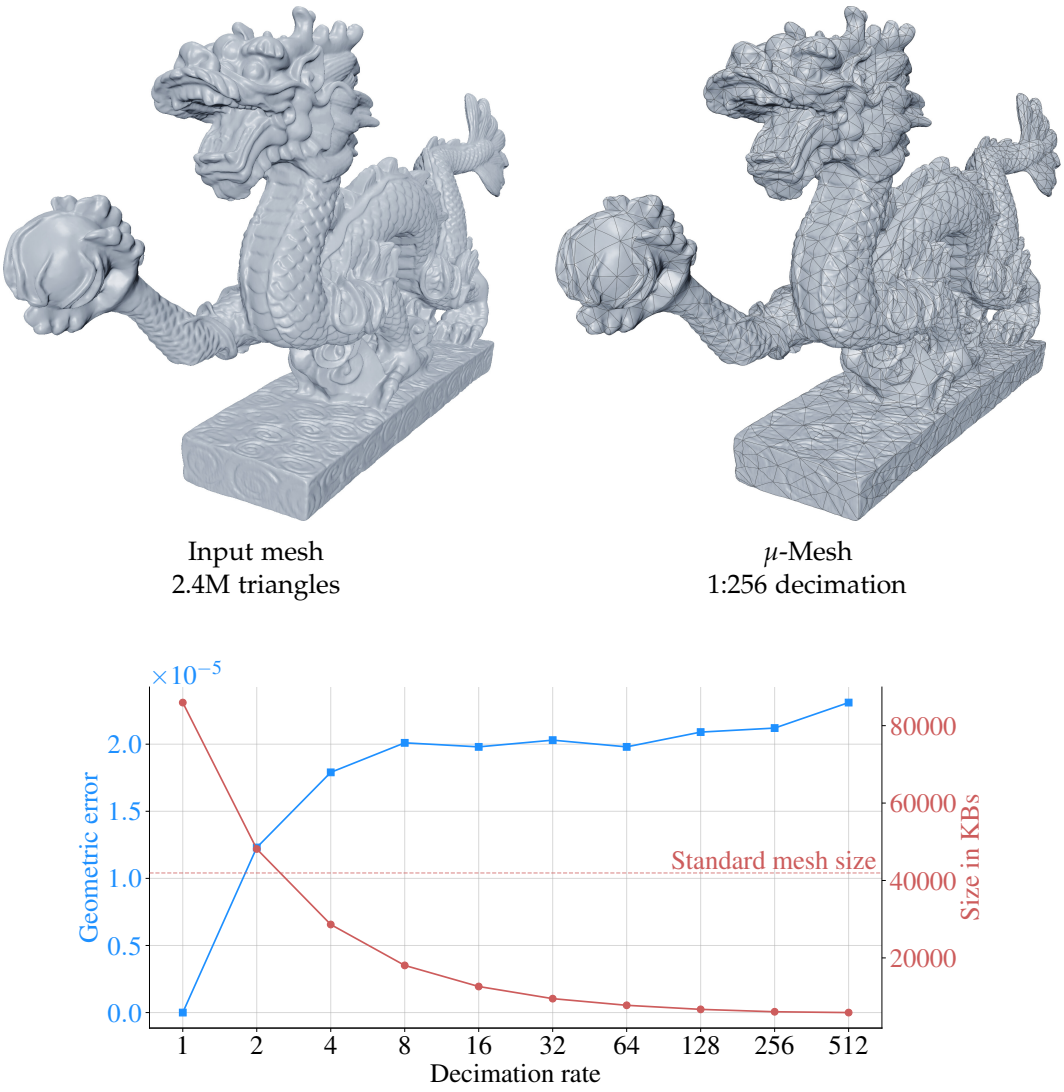


Figure 6.15: Used GPU RAM and introduced geometric error for various decimation rates of the base mesh.

6.11.3 Conversion of high-resolution models

In Figure 6.18 we show a collection of high-resolution meshes converted to the μ-Mesh representation with our method. The ability to efficiently and robustly produce μ-Mesh from high-resolution inputs is instrumental to our final objective of representing photogrammetry meshes with this encoding.

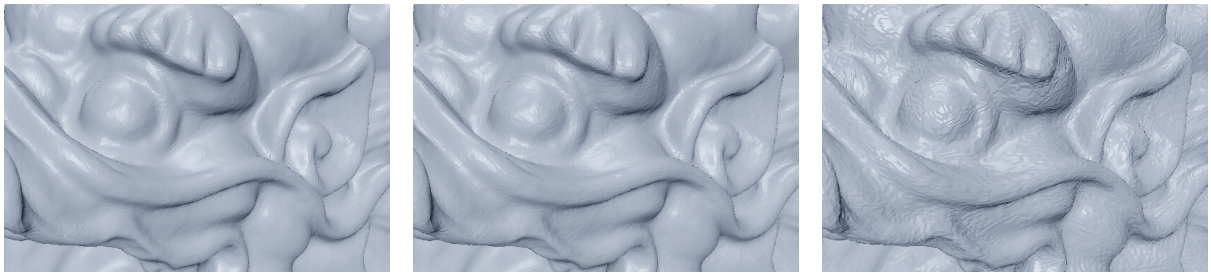
In Table 6.1 we report quantitative measures for the same models. We report the input, base, and micro-triangle count, the input size and compression ratio, the displaced μ-mesh isotropy and the geometric error, and the end-to-end processing time required to generate the μ-mesh. As discussed in previous sections, we measure the isotropy of the μ-mesh by computing the area-weighted average of the micro triangle aspect ratios resulting.

6.11.4 Textured models

In Figure 6.19 we show a selection of textured models from our Real-World Textured Things dataset [MPCT20]. We show a color map of the μ-Mesh ARAP energy ($\mu-$ARAP) before and



(a) Input (b) 32 bits (c) 16 bits



(d) 8 bits (e) 6 bits (f) 4 bits

Figure 6.16: Scalar displacement quantization. A high-quality base mesh approximation and tight scalar displacement volume allow for aggressive quantization of the micro-vertex displacements without losing representation accuracy.

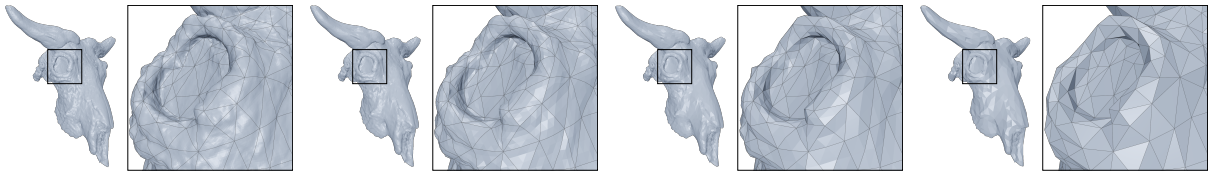


Figure 6.17: Controlling the level of detail of a μ-Mesh by progressively decreasing the subdivision level of base faces.

after optimizing our displacement-aware re-weighting (as described in Section 6.10) and the final textured model. For baking the texture image, we ray-cast along interpolated smooth micro-vertex normals to ensure the sampling direction is as orthogonal as possible to the displaced surface.

In Table 6.2 we report energy measures before and after optimizing base texture coordinates with our re-weighted formulation, as well as the optimized standard ARAP energy on the base mesh, which is used as initialization for the displacement-aware minimization. The energy decrease is quite noticeable despite significantly restricting the degrees of freedom of the system. The resolution of the base mesh ultimately determines the system variables, and coarser base meshes result in smaller gains in terms of distortion energy (rows 3 and 4 of Table 6.2). In terms of performance, our method is extremely fast as it operates on a reduced set of variables. When minimizing the re-weighted ARAP energy with the local-global method, the bottleneck is the *local* step and not the sparse linear solve required in the global step. The local step requires computing the SVD of a possibly huge number of small 2×2 matrices (one for each micro-triangle), but this process is trivial to parallelize to gain a noticeable speed-up. As can be seen from the table, the time required to minimize the re-weighted energy is extremely small.

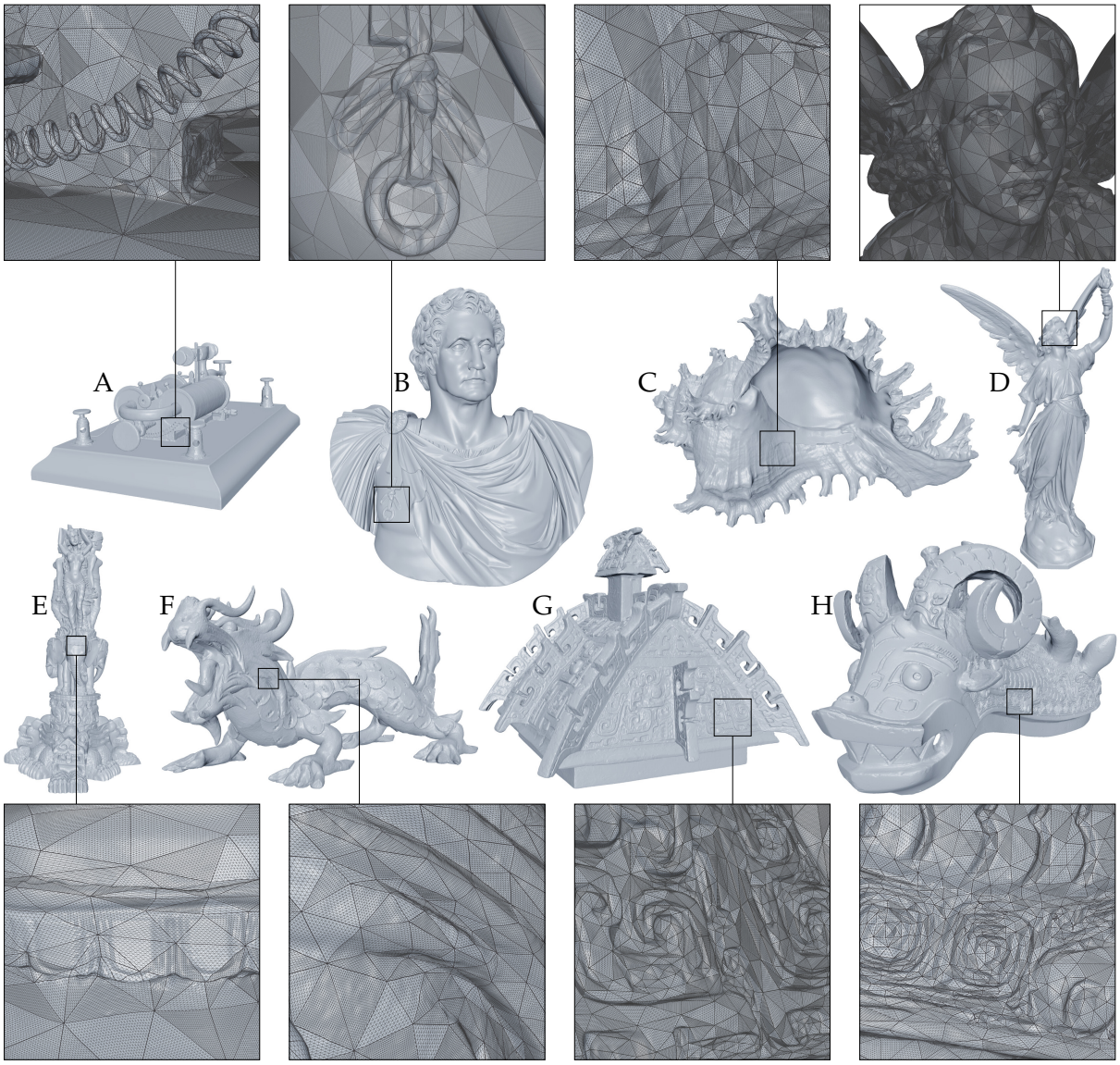
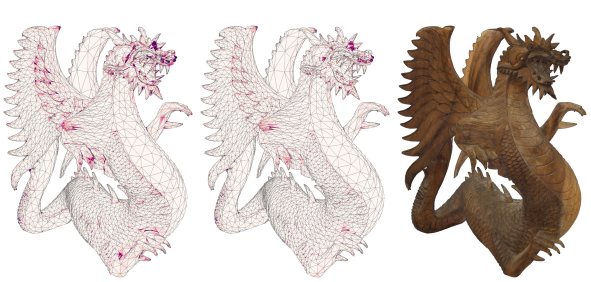


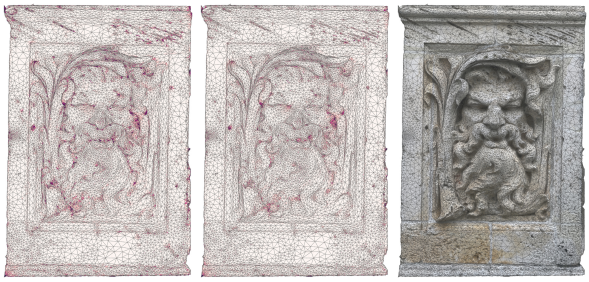
Figure 6.18: A selection of μ-Meshes generated from very high-resolution inputs. For quantitative measures regarding these models refer to Table 6.1.

Model	Figure 6.18	$\lvert F \rvert$	$\lvert F_B \rvert$	$\lvert F_\mu \rvert$	KBs (Input)	KBs (μ-Mesh)	Isotropy	Geometric error	Time
Telegraph	A	8 M	35 K	8.5 M	139648	20665	0.70	6.70×10^{-6}	13 m
GW Bust	B	26 M	30 K	27 M	458173	59171	0.73	1.83×10^{-6}	41 m
Murex	C	3.5 M	30 K	4 M	61867	10237	0.81	1.88×10^{-5}	9 m
Lucy	D	28 M	35 K	29. 5M	502128	64782	0.75	3.14×10^{-6}	42 m
XYZ Statuette	E	10 M	40 K	10.5 M	176776	25688	0.78	1.00×10^{-5}	15 m
XYZ Dragon	F	7 M	30 K	7.5 M	128153	18498	0.81	8.54×10^{-6}	13 m
Fangyi (2)	G	21.5 M	100 K	23 M	379236	56027	0.74	5.44×10^{-6}	27 m
Ewer	H	8 M	100 K	8 M	372011	23961	0.77	1.31×10^{-5}	14 m

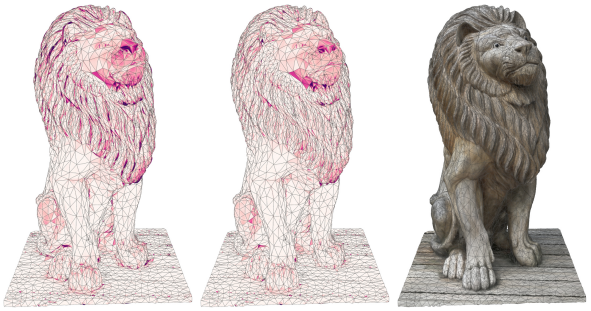
Table 6.1: Conversion of high-resolution 3D models to μ-meshes. Visual results are shown in Figure 6.18.



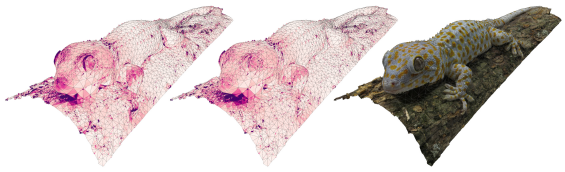
(a) RWT180 – Wooden Dragon



(b) RWT205 – Grotesque02



(c) RWT236 – Wooden Lion



(d) RWT295 – Tokay Gecko

Figure 6.19: Examples of textured μ-Meshes. The original models are taken from our Real-World Textured Things dataset [MPCT20] (Chapter 3). For each figure, we show color maps indicating the ARAP energy before and after the optimization with our re-weighted scheme, and the final textured result. Quantitative measures are reported in Table 6.2.

Model	Figure	$\lvert F_B \rvert$	$\lvert F_\mu \rvert$	ARAP Base	$\mu-$ARAP Initial	$\mu-$ARAP Optimized	Energy improvement	Time
Wooden Dragon	6.19a	20 K	3.3 M	0.007	0.035	0.021	39.9 %	20 s
Grotesque02	6.19b	30 K	6.5 M	0.012	0.033	0.025	39.3 %	28 s
Wooden Lion	6.19c	10 K	1.5 M	0.021	0.039	0.029	25.6 %	8 s
Tokay Gecko	6.19d	10 K	2.3M	0.035	0.089	0.069	22.4 %	10 s

Table 6.2: Displacement-aware parametrization of μ-Meshes. Textured results are shown in Figure 6.19

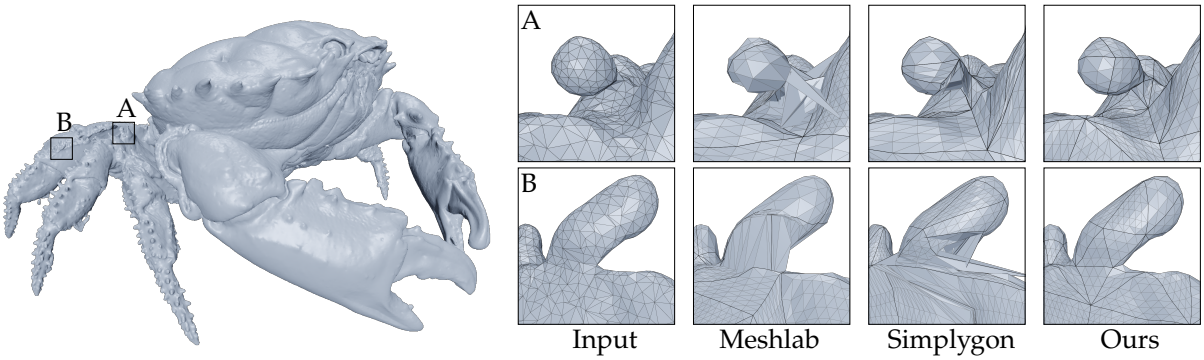
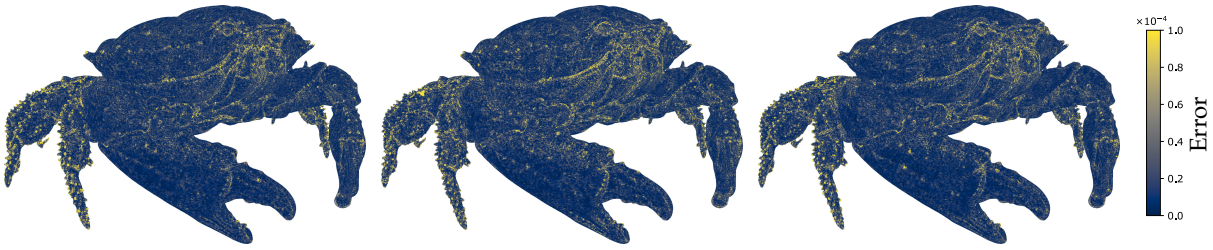


Figure 6.20: Comparison with other mesh simplification approaches. The two close-ups show failures due to the presence of base mesh vertices with negative visibility, resulting in displacement directions that are not coherently oriented.



(a) Meshlab – Error = 2.14×10^{-5} (b) Simplygon – Error = 2.20×10^{-5} (c) Ours – Error = 2.07×10^{-5}

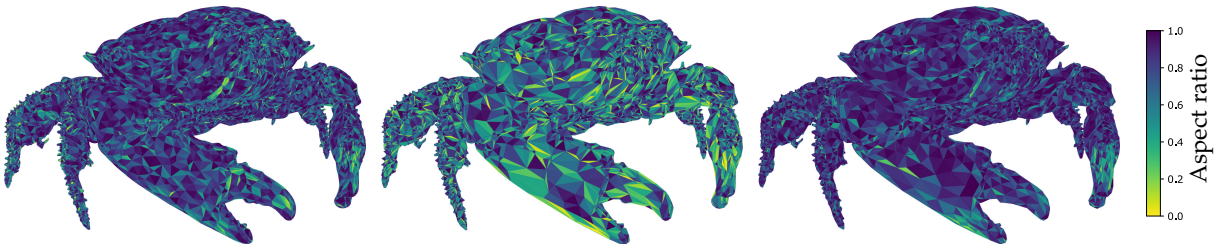
Figure 6.21: Geometric error for different base mesh generation approaches.

6.11.5 Comparison with standard mesh simplification

We compare our base mesh generation procedure with standard mesh simplification tools. For our comparison, we choose Meshlab [RCD*13] because of its popularity and Simplygon [Mic22] because it natively supports the generation of μ-meshes within its simplification and remeshing workflow. We compare the quality of the resulting base mesh and the isotropy of the micromesh tessellation.

In Figure 6.20 we show a qualitative comparison of the results obtained using as starting point the base meshes obtained with Meshlab, Simplygon, and our decimation strategies. The input mesh has 2.1M faces, which we decimate to 25K to produce the base mesh; to produce the final μ-mesh, we use our uniform tessellation and displacement procedure. We see that the base meshes obtained with Meshlab and Simplygon feature meshing artifacts and self-intersections due to the presence of negative visibility vertices, while our base mesh does not suffer from such issues and ensures that consistently oriented displacement directions can always be found.

We also compare the average geometric error and micro-sampling isotropy achieved by the same methods. As shown in Figures 6.21 and 6.22, our base mesh generation produces a better surface approximation and strategy significantly outscores the alternatives in terms of sampling isotropy, corroborating our core assumption that the isotropy of the base triangulation is a valid proxy for the final refined and displaced surface.



(a) Meshlab – Isotropy = 0.78 (b) Simplygon – Isotropy = 0.65 (c) Ours – Isotropy = 0.83

Figure 6.22: μ-Mesh isotropy for different base mesh generation approaches.

6.11.6 Comparison with Bijective shells [JSZP20]

In Figures 6.23 and 6.24 we show two comparisons with [JSZP20]. As we can see from Figure 6.24, the existence of a bijective map between the shell and the original mesh is not sufficient to ensure the correct reproduction of the high-resolution surface with linearly interpolated displacement directions. This is due to the nonlinearity of the volumetric bijective map, that in [JSZP20] is obtained by decomposing the shell prisms in tetrahedra (6 tetrahedra per prism) and assigning constant traversal directions for each individual tetrahedra. As a result, their shells can significantly drift from the original surface, making them unsuitable as base meshes in our setting where the displacement directions are simply linearly interpolated. Bijective shells also perform worse in terms of micro-sampling isotropy, and cannot decimate border edges.

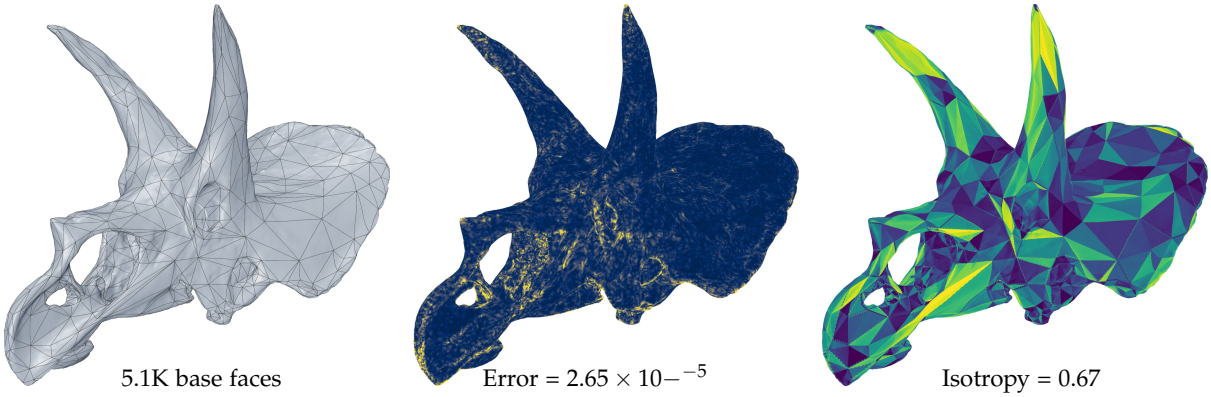
There are also other practical advantages to our solution. Our base mesh construction is more robust, as it can be safely applied to non-manifold and self-intersecting meshes whereas the generation of bijective shells outright rejects such inputs. It is also significantly faster: the construction of bijective shells can take hours when applied to meshes with a triangle count in the order of millions due to the expensive bijectivity checks that the method performs when optimizing the shell structure.

6.12 Conclusions

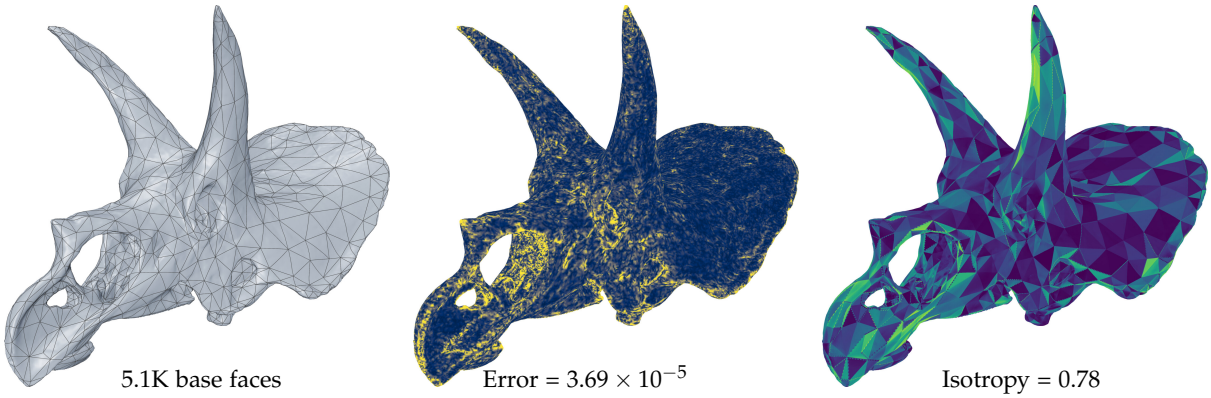
In this chapter, we presented a robust and reliable method to convert a high-resolution traditional mesh representation to a μ-mesh, a recently proposed efficient representation based on displacement maps. This encoding has hardware support [NVI22b], making it an ideal candidate for enabling high-performance rendering of extremely detailed geometries in real-time systems. We introduced a fast and reliable method to create a base control mesh that is optimized to produce a high-quality displaced surface, and demonstrated it on challenging models with millions of faces. Our construction method improves on general-purpose mesh simplification methods by introducing optimization targets that are designed around the subsequent refinement and displacement.

We also introduced a re-weighting scheme to optimize the distortion energy of the base mesh parametrization by accounting for the final surface displacement, which can be useful when converting textured and photogrammetric 3D models to this new representation.

Limitations Reproducing high-level anisotropic features with an isotropic and coarse base mesh is challenging. Without constraining the aspect ratio of base triangles, the decimation of such features would introduce stretched triangles to adapt to the feature, resulting in an



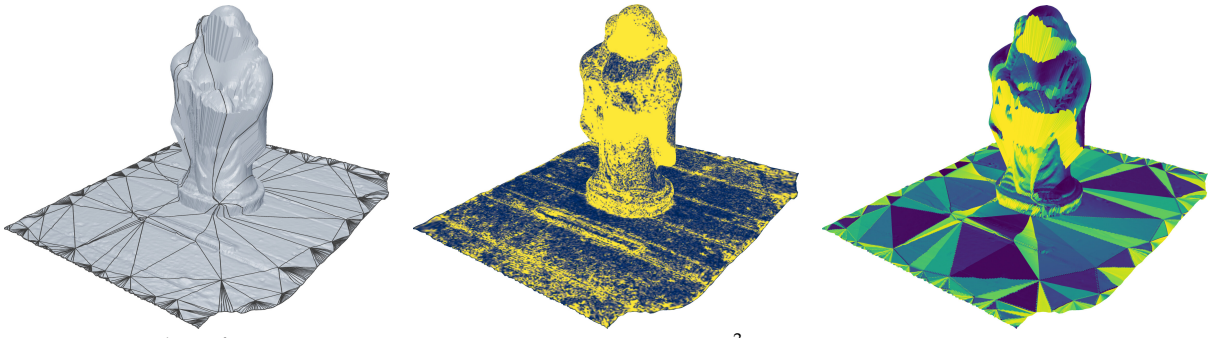
(a) Bijective shell – Processing time 12 minutes



(b) Ours – Processing time 2 minutes

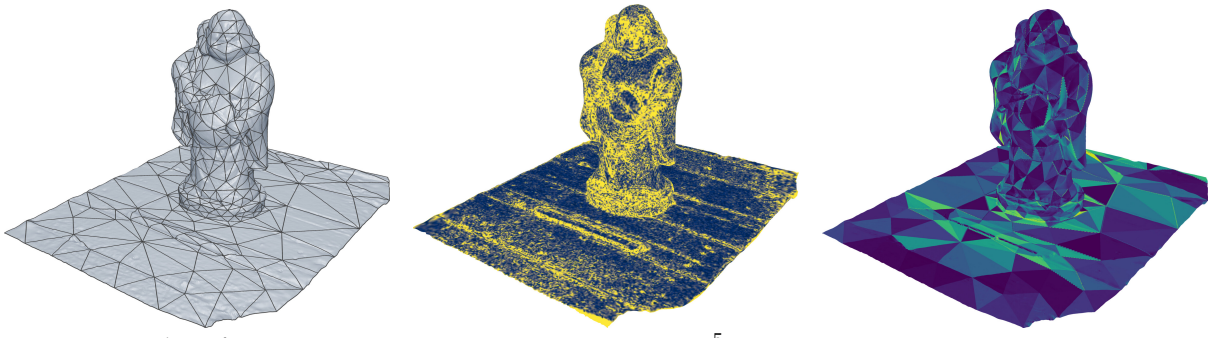
Figure 6.23: RWT161 (Triceratops horridus by Benoit Rogez) – Comparison with Bijective shell [JSZP20]. The input mesh has 580K faces.

over-sampled μ-mesh. On the other hand, constraining the aspect ratio of base triangles can produce, at very coarse decimation levels, significant geometric deviation from the high-resolution surface in the base mesh, resulting in aliasing artifacts. The investigation of alternative approaches to tackle this specific issue is an interesting direction for future work.



784 base faces | Distance = $9.73 \times 10-^{-3}$ | Isotropy = 0.57

(a) Bijective shell – Processing time 2 minutes



784 base faces | Distance = 6.37×10^{-5} | Isotropy = 0.78

(b) Ours – Processing time 40 seconds

Figure 6.24: RWT1 (The Thinker Sculpture by OpenMVS) – Comparison with Bijective shell [JSZP20]. The input mesh has 220K faces. This is a failure case for Bijective shell, outputting a mesh that is too coarse to correctly represent the original surface with the resulting base configuration.

Chapter 7

The inverse barycentric mapping problem

Here we analyze the problem of finding the triangular barycentric coordinates of an interpolated ray hitting a given point. This task, which we term the inverse barycentric displacement problem*, is general and useful in geometry processing and computer graphics. Concrete applications of the solution include the construction of displacement maps and texture baking. We derive the set of complete, closed-form solutions, and discuss the number and existence of solutions. We close with a discussion of a few example applications.*

7.1 Introduction

Consider a 3D embedded triangle $\mathcal{T}$ with arbitrary displacement vectors $\mathbf{d}_i$ defined at its corners $\mathbf{v}_i$ (Figure 7.1, *left*). Let a ray start at a point $\mathbf{v}$ on $\mathcal{T}$, where $\mathbf{v} = u\mathbf{v}_0 + v\mathbf{v}_1 + w\mathbf{v}_2$ is obtained by barycentric interpolation, and travel in the direction of the (corresponding) interpolated displacement direction $\mathbf{d}$. Traveling along this ray, a displaced point $\mathbf{q}$ is defined at parametric position t:

$$\mathbf{q} = \mathbf{v} + t\,\mathbf{d}. \tag{7.1}$$

t is the *scalar displacement value* defined at $\mathbf{v}$, and may be stored in a scalar displacement texture covering $\mathcal{T}$. Usually, we are given an interpolated position $\mathbf{v}$ and direction $\mathbf{d}$ on $\mathcal{T}$ and we want to find $\mathbf{q}$ as a correspondence on a given *target* surface (dotted line in Figure 7.1, *right*). For example, $\mathbf{q}$ can be found by ray-casting against a high-resolution mesh. In the following, we will use the term *direct* displacement to refer to the problem of finding $\mathbf{q}$ (equivalently, t) for given $\mathbf{v}$ and $\mathbf{d}$.

In this appendix, we are concerned with the *inverse* of the *direct* displacement problem, where we are given a target point $\mathbf{q}$ and we want to find a position $\mathbf{v}$ on $\mathcal{T}$ such that the ray defined on $\mathbf{v}$ hits $\mathbf{q}$. The solution is interesting, useful, and non-trivial.

Rationale The inverse problem is useful in several contexts. For example, it makes it possible to bake textures for $\mathcal{T}$ by processing positions $\mathbf{q}$ directly on the target surface, sidestepping the need for ray-casting which typically requires expensive iterations over the numerous elements of a densely tessellated target surface. The design of good support meshes to be displaced can

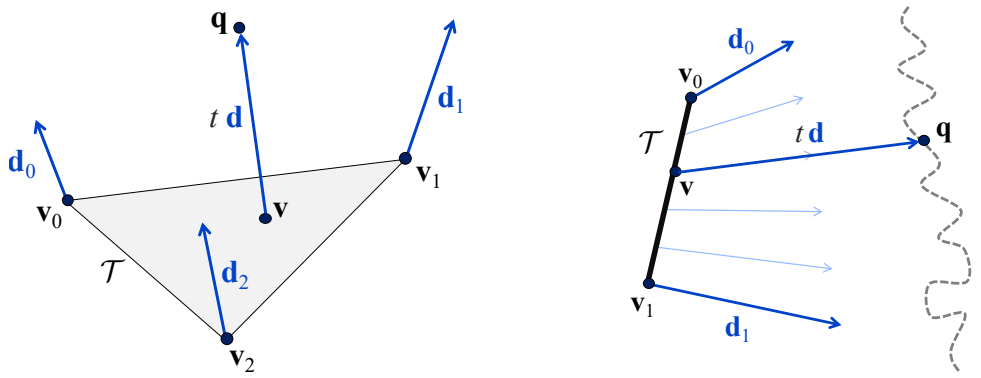


Figure 7.1: *left* - The inputs of the displacement problem (direct or inverse.) *right* - A 2D depiction (note that in 3D, the 3 corner rays do not meet at a point, which would otherwise make the problem trivial.)

also benefit from the ability to quickly identify where on a potential $\mathcal{T}$, if anywhere, a given point $\mathbf{q}$ on the target surface would be mapped.

The inverse mapping problem is not trivial and, despite its potential usefulness, to our knowledge has never been solved in closed form. The nontriviality is confirmed by the observation that the solution does not necessarily exists, nor it is necessarily unique, and certain configurations, for example when all three rays meet into $\mathbf{q}$, even admit infinite solutions for $\mathbf{v}$, as $\mathbf{q}$ can be reached from any point on $\mathcal{T}$.

Related work Traditionally, in contexts such as texture-baking, displacement-map construction (for curved surface representations or polygonal meshes), inverse subdivision surfaces, and others, existing solutions are designed to leverage solely on the solution of the direct problem. This may be due, at least in part, to the perceived intricacy of the inverse problem. In Section 7.4, we exemplify two scenarios where our closed-form solution to the inverse problem offers a compelling alternative to traditional solutions. The description of each scenario starts with a brief overview of the respective Related Work.

7.2 Problem definition

Given a target point $\mathbf{q}$, we want to find a point $\mathbf{v}$ on $\mathcal{T}$ such that the ray shot from $\mathbf{v}$ in the interpolated direction $\mathbf{d}$ hits $\mathbf{q}$ (7.1).

Let the corners of $\mathcal{T}$ be in positions $\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2$, and let the three directions defined at corners be $\mathbf{d}_0$, $\mathbf{d}_1$, $\mathbf{d}_2$. A point $\mathbf{v}$ is identified by its non-negative barycentric coordinates $\alpha_0, \alpha_1, \alpha_2$ with

$$\sum_i \alpha_i = 1 \tag{7.2}$$

so that:

$$\mathbf{v} = \sum_i \alpha_i \, \mathbf{v}_i \tag{7.3}$$

$$\mathbf{d} = \sum_i \alpha_i \, \mathbf{d}_i. \tag{7.4}$$

Substituting (7.3, 7.4) in (7.1), gives

$$\mathbf{q} = \sum_i \alpha_i \, \mathbf{v}_i + t \sum_i \alpha_i \, \mathbf{d}_i \tag{7.5}$$

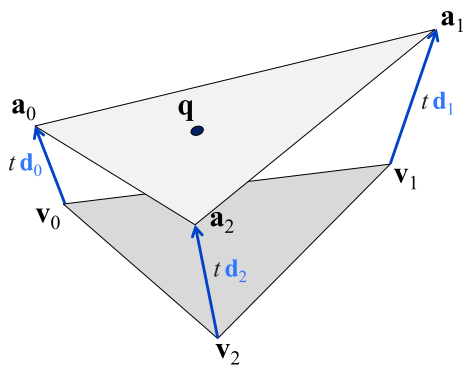


Figure 7.2: Construction for the geometric solution.

where the three scalars α_i and the scalar t are unknowns.

This seemingly simple geometric setup leads to a system, (7.5), of three scalar equations, quadratic with the four variables, subject to the linear constraint (7.2). This system is not trivial to solve directly.

7.3 Geometric derivation of the solution

We can rewrite Equation (7.5) as

$$\begin{aligned}\mathbf{q} &= \sum_i \alpha_i \left(\mathbf{v}_i + t\,\mathbf{d}_i\right) \\ &= \sum_i \alpha_i\,\mathbf{a}_i \end{aligned} \tag{7.6}$$

with $\mathbf{a}_{0,1,2}$ denoting the position of the vertices of $\mathcal{T}$ displaced by t along the respective direction:

$$\mathbf{a}_i = \mathbf{v}_i + t\,\mathbf{d}_i \tag{7.7}$$

Equation (7.6) implies that the target point $\mathbf{q}$ must necessarily lie on the plane $\mathcal{P}$ passing by the three positions $\mathbf{a}_i$ (see Figure reffig:1), irrespective of the choice of $\mathbf{v}$; observe that $\mathcal{P}$ is a function solely of t (and different choices of t results in general in planes with different orientations). Moreover, it implies that the barycentric coordinates of $\mathbf{q}$ in triangle $\mathbf{a}_0, \mathbf{a}_1, \mathbf{a}_2$ are the same as the sought barycentric coordinates $\alpha_0, \alpha_1, \alpha_2$ of $\mathbf{v}$ in $\mathcal{T}$.

Our solution is therefore to first find value(s) of t so that $\mathbf{q}$ lies on $\mathcal{P}$, and then simply find $\alpha_0, \alpha_1, \alpha_2$ (and thus $\mathbf{v}$) with fixed t as the barycentric coordinates of $\mathbf{q}$ inside triangle $\mathbf{a}_0, \mathbf{a}_1, \mathbf{a}_2$. A geometric view of this two-step solution process is shown in Figure 7.2.

The condition for $\mathbf{q}$ to be on $\mathcal{P}$ can be expressed as

$$0 = (\mathbf{a}_0 - \mathbf{q}) \cdot \mathbf{n}, \tag{7.8}$$

that is, that the vector connecting $\mathbf{q}$ to $\mathbf{a}_0$ is orthogonal to $\mathbf{n}$, with $\mathbf{n}$ denoting a vector orthogonal to $\mathcal{P}$ ($\mathbf{n}$ is, again, as a function of t):

$$\mathbf{n} = (\mathbf{a}_1 - \mathbf{a}_0) \times (\mathbf{a}_2 - \mathbf{a}_0). \tag{7.9}$$

Rewriting (7.8) in terms of $\mathbf{v}$, $\mathbf{d}$ and t gives:

$$\begin{aligned} 0 &= (\mathbf{a}_0 - \mathbf{q}) \cdot (\mathbf{a}_1 - \mathbf{a}_0) \times (\mathbf{a}_2 - \mathbf{a}_0) \\ &= (\mathbf{v}_0 + t\,\mathbf{d}_0 - \mathbf{q}) \cdot (\mathbf{v}_1 + t\,\mathbf{d}_1 - \mathbf{v}_0 - t\,\mathbf{d}_0) \times (\mathbf{v}_2 + t\,\mathbf{d}_2 - \mathbf{v}_0 - t\,\mathbf{d}_0) \\ &= ((\mathbf{v}_0 - \mathbf{q}) + t\,\mathbf{d}_0) \cdot ((\mathbf{v}_1 - \mathbf{v}_0) + t\,(\mathbf{d}_1 - \mathbf{d}_0)) \times ((\mathbf{v}_2 - \mathbf{v}_0) + t\,(\mathbf{d}_2 - \mathbf{d}_0)) \end{aligned} \tag{7.10}$$

This gives a polynomial that is cubic in t.

7.3.1 Deriving the polynomial coefficients

To ease the derivation of the coefficients of the polynomial in (7.10), it is convenient to group the constant sub-expressions in (7.10) as columns of two 3×3 matrices:

$$\mathbf{P} = [\,\mathbf{v}_0 - \mathbf{q}\,|\,\mathbf{v}_1 - \mathbf{v}_0\,|\,\mathbf{v}_2 - \mathbf{v}_0\,], \tag{7.11}$$

$$\mathbf{D} = [\quad \mathbf{d}_0 \quad |\,\mathbf{d}_1 - \mathbf{d}_0\,|\,\mathbf{d}_2 - \mathbf{d}_0\,] \tag{7.12}$$

We term $\mathbf{P}$ the *position matrix*, as it depends on the positions of vertices the of $\mathcal{T}$ and the target point $\mathbf{q}$; we term $\mathbf{D}$ the *displacement matrix*, as it depends on the displacement directions defined at the vertices of $\mathcal{T}$. We will refer to their columns as $\mathbf{P}_0$, $\mathbf{P}_1$, $\mathbf{P}_2$ (and likewise for $\mathbf{D}$).

We can now rewrite (7.10) as

$$0 = (\mathbf{P}_0 + t\,\mathbf{D}_0) \cdot (\mathbf{P}_1 + t\,\mathbf{D}_1) \times (\mathbf{P}_2 + t\,\mathbf{D}_2) \tag{7.13}$$

That is (recalling that $\det(\mathbf{a}|\mathbf{b}|\mathbf{c})$ is $\mathbf{a} \cdot (\mathbf{b} \times \mathbf{c})$):

$$0 = \det(\mathbf{P} + t\,\mathbf{D}). \tag{7.14}$$

Extracting t gives (see the Appendix at the end of this chapter for a derivation):

$$0 = A\,t^3 + B\,t^2 + C\,t + D \tag{7.15}$$

with

$$A = \det(\mathbf{D}) \qquad D = \det(\mathbf{P}) \tag{7.16}$$

$$\begin{aligned} B = {} & \det(\mathbf{P}_0|\mathbf{D}_1|\mathbf{D}_2) + \\ & \det(\mathbf{D}_0|\mathbf{P}_1|\mathbf{D}_2) + \\ & \det(\mathbf{D}_0|\mathbf{D}_1|\mathbf{P}_2) \end{aligned} \qquad \begin{aligned} C = {} & \det(\mathbf{D}_0|\mathbf{P}_1|\mathbf{P}_2) + \\ & \det(\mathbf{P}_0|\mathbf{D}_1|\mathbf{P}_2) + \\ & \det(\mathbf{P}_0|\mathbf{P}_1|\mathbf{D}_2) \end{aligned} \tag{7.17}$$

7.3.2 Finding all valid solutions (if any).

Solving (7.15) with the standard cubic formula returns up to three real solutions for t, which we denote as t_0, t_1, and t_2.

For a given choice of t_i, the corresponding triangle $\mathbf{a}_0, \mathbf{a}_1, \mathbf{a}_2$ is constructed (7.7), then, α_i are trivially extracted as the barycentric coordinates of $\mathbf{q}$ in that triangle. If any $\alpha_i < 0$, then $\mathbf{v}$ is outside the triangle $\mathcal{T}$, and the corresponding solution t_i must be discarded.

This procedure fails when the triangle $\mathbf{a}_0, \mathbf{a}_1, \mathbf{a}_2$ is degenerate, i.e. when it collapses into a line or a point (meaning that the used value of t_i satisfies (7.8), but only in the trivial sense that $\mathbf{n}$ is the zero vector). This condition can be detected by checking that $\mathbf{n}$ (7.9) vanishes. If the

target point $\mathbf{q}$ happens to belong to this line or point, then we have infinite solutions for α_i – and therefore, with (7.4), for $\mathbf{v}$; otherwise, the solution t_i must be discarded.

Depending on the context, negative values of t_i, or values larger than one, may also be discarded as invalid (when "backward" negative displacement, or displacements shooting "beyond $\mathbf{d}$", are to be disallowed).

Each non-discarded t_i, if any, results in a valid solution to the inverse projection problem. Multiple such solutions can exist. A possible strategy to employ in such cases is to consistently favor the smallest value of $|t_i|$, which has the benefit of making $\mathbf{v}$ vary with continuity as $\mathbf{q}$ varies (barring degenerate cases).

Special cases It is interesting to analyze a few special cases of the general formulation above, although an implementation does not need to be aware of them.

When $\mathbf{q}$ is on $\mathcal{T}$, or the plane containing $\mathcal{T}$, then the position matrix $\mathbf{P}$ is singular (its three columns are co-planar), so D is zero, and $t = 0$ is always a solution of (7.15), as expected (observe that it is not necessarily the only solution).

When the displacement vectors $\mathbf{d}_i$ are all parallel to each other (even if with different lengths), then the direction matrix $\mathbf{D}$ is singular and rank 1 (as the determinant and rank of $\mathbf{D}$ are the same as $[\mathbf{d}_0|\mathbf{d}_1|\mathbf{d}_2]$). This makes both A and B zeros (as taking any two vectors from $\mathbf{D}$ zeroes the determinant), making the polynomial (7.15) linear, and the solution for t unique. This is also expected, from a geometric point of view (observe however that this still does not make all planes $\mathcal{P}$ parallel to each other, for all t: that additionally requires all displacement vectors to be of the same length).

A pathological case is then all $\mathbf{d}_i$ are in the same plane as $\mathcal{T}$. In that case, A, B, and C are zeroed (as each matrix is made by co-planar column vectors), making the polynomial (7.15) degree 0. Unless D is also 0, when $\mathbf{q}$ is also on the plane of $\mathcal{T}$, there is no solution for t, valid or otherwise, as expected.

7.4 Example applications

As we argued in Section 7.1, the presented inverse problem can find applications in several contexts in Geometry Processing. In the following, we briefly exemplify this with two use cases.

7.4.1 Texture baking via rendering

Texture baking is a common technique, pioneered by early research such as [CMR*99, LMH00] and supported today by common 3D modeling suites [Ble22, Hel21, Ado21, Mar22]. A baked texture (sometimes referred to as transfer map, mesh map, redetail texture, or detail-recover texture) is a synthesized texture that records data originally stored in a high-resolution mesh $\mathcal{M}_H$ into the texels of a texture image intended for a given low-resolution mesh $\mathcal{M}_L$. The texels can be filled with displacement scalars, or any information stored on $\mathcal{M}_H$, such as normals, colors, pre-lit values, or any other per-vertex attributes, per face attributes, or even an original texture; $\mathcal{M}_L$ comes with a UV-map, which for simplicity we will assume to be free from seams, and per-vertex displacement directions; the latter can be per-vertex normal directions, or can even be explicitly optimized for this specific task in reprocessing [TCS03].

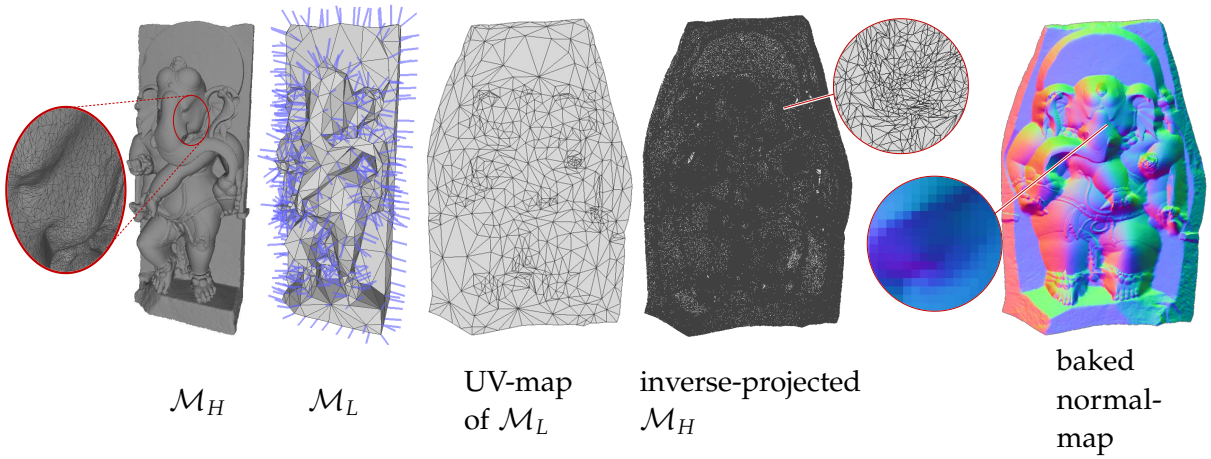


Figure 7.3: Texture baking via inverse displacement: an example of a result. The input consists of a hi-resolution mesh $\mathcal{M}_H$ (324K faces) and a low-resolution mesh $\mathcal{M}_L$ (1000 faces), provided with per-vertex displacement directions (in blue), and a UV-map (shown). We inverse project every vertex of $\mathcal{M}_H$ into $\mathcal{M}_L$, moving it into the corresponding u, v position, and rasterize it using a normal rendering pass, obtaining the baked normal map. *Used input data:* $\mathcal{M}_H$ is the *Ganesa* model from our dataset (Section 3.3.1) [MPCT20]; $\mathcal{M}_L$ is obtained by automatic simplification, using MeshLab; displacement directions on $\mathcal{M}_L$ are standard per-vertex normal directions (area-weighted average of per-face normal); the UV-map of $\mathcal{M}_L$ is constructed with [LZX*08].

Direct approach Traditionally, the baking is performed leveraging on the *direct* problem: for each texel, a ray is cast from the corresponding position on $\mathcal{M}_L$, toward the interpolated direction, and the information is extracted and stored from whichever point on $\mathcal{M}_H$ is hit.

Inverse approach (new) Using the inverse problem, we can iterate over vertices of $\mathcal{M}_H$, find a position on $\mathcal{M}_L$ by inverse projection, and reposition that vertex in the corresponding location in the 2D texture space. This way, $\mathcal{M}_H$ is morphed into the 2D parametric space of $\mathcal{M}_L$, and a simple rasterization-based rendering suffices to produce the baked texture. Alternatively, depth values t can also be rendered to bake displacement maps. See Figure 7.3 for one example.

Comparison The relative merits of the two solutions, traditional (direct) and novel (inverse), depend on the context, and a full analysis exceeds the scope of this paper; for our intents, the observation suffices that the two solutions offer a different set of advantages. In the following, we outline only a few general considerations.

Efficiency can be compared, neglecting for the sake of simplicity the impact of spatial indexing structures. A clear advantage of the inverse approach is that the final rendering pass can exploit the extremely optimized standard rasterization rendering pipeline; this is in practice extremely fast (milliseconds at most), and we can safely disregard its impact on computation time. Let R_H and R_L be the resolutions of $\mathcal{M}_H$ and $\mathcal{M}_L$ (number of primitives, either vertices or faces), and R_T be the resolution of the baked texture (number of texels), with, typically

$$R_T > R_H \gg R_L \tag{7.18}$$

(to prevent loss of information, R_T needs to surpass R_H, e.g. by a factor 2 according to Nyquist–Shannon sampling theorem; considerably more if $\mathcal{M}_H$ comes with its own texture

to be re-baked). For a given triangle, both the inverse and direct problems are solved in constant time, although direct ray-casting necessitates fewer operations. In the direct method, a ray from each texel must be tested against each primitive in $\mathcal{M}_H$, leading to a $O(R_T R_H)$ complexity; in the inverse approach, all vertices of $\mathcal{M}_H$ must be inverse-projected over each triangle of $\mathcal{M}_L$, costing $O(R_H R_L)$, which is a much lower asymptotic complexity. Naturally, both approaches can be sped up considerably by employing spatial indexing structures (down to $O(R_T \log(R_H))$ and $O(R_H \log(R_L))$ respectively), which reduces the gap.

Ease of use favors the inverse-projection approach, as fragment-shaders, e.g. the same ones used in the rendering, can be directly re-used in the final pass to reproduce any complex lighting or texturing effect to bake the results in form of textures (including shadowing, wireframe, or any other mesh rendering technique).

Accuracy favors the direct approach, as in the inverse approach the linear interpolation inside the triangles of the flattened $\mathcal{M}_H$ introduces a small approximation error, which decreases with the resolution of $\mathcal{M}_H$. Also, the inverse approach requires care to work in presence of texture seams over $\mathcal{M}_L$.

7.4.2 Perfect reprojectability test

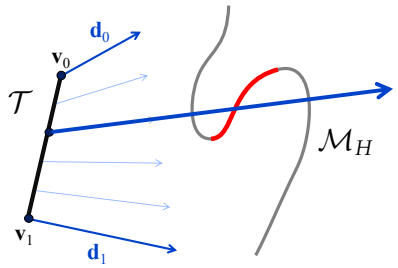


Figure 7.4: 2D depiction of a case where a given surface $\mathcal{M}_H$ is *not* reprojectable from base triangle $\mathcal{T}$, as one interpolated ray intersects $\mathcal{M}_H$ three times, including with one with mismatching orientation (red bold line).

In the context of constructing displacement-mapped surfaces, a recurrent concern is whether or not a given base mesh, with an associated set of per-vertex displacement directions, can correctly represent a given continuous target surface, often discretized as a high-resolution mesh $\mathcal{M}_H$ [CMO97, PPT*11, JSZP20]. A (scalar) displacement map is limited in that it can only encode different heights for each point; in other words, it is a warped "2.5D" height field, and, as such, it can only express a limited range of shapes. It is in general not trivial to determine if a given target surface $\mathcal{M}_H$ can be represented by a displacement map encoded over a base triangle $\mathcal{T}$.

Direct approach A brute force solution using the direct approach is to cast rays from a sampling of $\mathcal{T}$ and check whether or not any given ray hits multiple valid targets on $\mathcal{M}_H$, or, equivalently, whether or not it hits any back-facing face on $\mathcal{M}_H$ (see Figure 7.4). This relies on the sampling to be sufficiently dense, and is work intensive, making it impractical for example in contexts where the test must be performed to inform the construction of an arrangement of $\mathcal{T}$.

Existing alternatives In [JSZP20], the displacement is redefined as a continuous piecewise linear mapping, to make it easier to invert, making it possible to precisely test for repro-

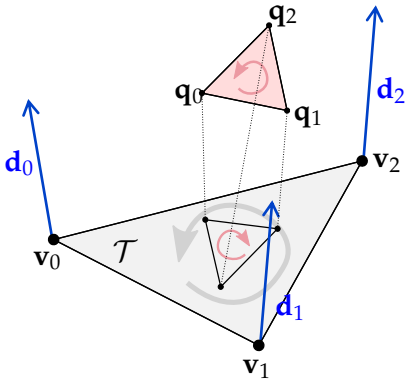


Figure 7.5: Exact reprojectability test for a triangle $(\mathbf{q}_0, \mathbf{q}_1, \mathbf{q}_2)$, over a base-triangle $\mathcal{T}$. In this instance, the test fails because the projection of $(\mathbf{q}_0, \mathbf{q}_1, \mathbf{q}_2)$ of $\mathcal{T}$ has a different orientation than $\mathcal{T}$.

jectability; while the test is exact, that framework changes the definition of the displacement map, drifting away from the traditionally intended semantics (vertices are displaced not along a single straight line, but along a broken line). In many previous works, the test is only approximated, so to make it faster. For example, in [PPT*11], the test is substituted by using a "fitmap", which is a scalar field defined in preprocessing that describes, for each position p on $\mathcal{M}_H$, the approximate maximal size of a base-triangle centered around p that can still be correctly reprojected; while this is can be an efficient way to inform the construction of an arrangement of base-triangles, the test is only an approximation.

Inverse approach (new) To perfectly test for reprojectability, we can test if any triangle in $\mathcal{M}_H$ is mapped, by the inverse projection, into a triangle inside $\mathcal{T}$ having an orientation opposite to that of $\mathcal{T}$. Given a triangle $(\mathbf{q}_0, \mathbf{q}_1, \mathbf{q}_2) \in \mathcal{M}_H$, we first inverse-project $\mathbf{q}_0$, $\mathbf{q}_1$ and $\mathbf{q}_2$, obtain three sets of barycentric coordinates $\boldsymbol{\alpha}^0$, $\boldsymbol{\alpha}^1$ and $\boldsymbol{\alpha}^2$ on $\mathcal{T}$, with $\boldsymbol{\alpha}^i = (\alpha_0^i, \alpha_1^i, \alpha_2^i)$ – see Figure 7.5.

The consistency of orientation of the back-projected triangle on $\mathcal{T}$ is then determined by the sign the 2D cross product of the 2D triangle defined by any two barycentric coordinates:

$$(\alpha_0^1 - \alpha_0^0)(\alpha_1^1 - \alpha_1^0) - (\alpha_0^2 - \alpha_0^0)(\alpha_1^2 - \alpha_1^0) > 0 \tag{7.19}$$

This test is exact; note that it does not explicitly require the 3D normal of either $\mathcal{T}$ or $(\mathbf{q}_0, \mathbf{q}_1, \mathbf{q}_2)$.

7.5 Conclusions

In this chapter, we presented a complete solution to what we term the "inverse displacement problem", a basic geometric sub-problem that commonly arises in contexts such as the construction or baking of scalar displacement maps over meshes, subdivision surfaces, and other representations. Our solution is given in closed form, and can be made efficient to evaluate; while typical problem instances admit a single valid solution, our procedure also correctly identifies cases admitting no solution, infinite solutions, or a set of either two or three solutions to choose from.

We observe that the vast and long-spanning Literature on displacement-map construction, and adjacent fields, always strive to formulate their algorithms to employ the direct problem only (to the best of our knowledge). We conjecture this may be due, in some part, to the perceived intricacy of the inverse problem, which, in spite of its apparent geometric simplicity, defies at

first glance straightforward solutions. The present work intends to offer an alternative and settle the question.

7.5.1 Problem variants

We conclude by considering the effect of a few common variations of the direct problem on the inverse problem.

In some scenarios, the displacement vectors $\mathbf{d}_i$ are assumed to be the same length, or all unit. Our solution makes no assumption about this and is directly usable. As far as we can tell, this additional assumption does not offer any opportunity to ease the inverse problem.

Another variation is when the interpolated displacement vector $\mathbf{d}$ is re-normalized after interpolation, defining the ray direction as $\mathbf{d}' = \mathbf{d}/||\mathbf{d}||$. This change does not affect our construction at all, provided that our final scalar displacement value t is adjusted, after the computation is over, to $t' = t\,||\mathbf{d}||$.

A different scenario is when the base triangle $\mathcal{T}$ is substituted by a (flat or not flat) quadrilateral $\mathcal{Q}$. For example, in many techniques, a displacement map is applied over a quad-mesh rather than a triangle-mesh [PPT*11, GA16]. The direct displacement problem is not affected much, except that $\mathbf{v}$ and $\mathbf{d}$ are now bilinearly interpolated between four corners, rather than being linearly interpolated between three. The inverse problem, however, becomes considerably more intricate. The presented solution is now unusable: equations from 7.2 to 7.7 can be immediately adapted (index i now ranging from 0 to 3), but neither $\mathcal{P}$ or $\mathbf{n}$ are defined any longer (as the four $\mathbf{a}_i$ are not necessarily co-planar, not even when $\mathcal{Q}$ is flat and $\mathbf{d}_i$ are unitary). In this variant, the inverse problem is (to our knowledge) still open.

Derivation of (7.15) from (7.14)

For any pair of 3×3 matrices $\mathbf{M} = [\mathbf{M}_0|\mathbf{M}_1|\mathbf{M}_2]$ and $\mathbf{N} = [\mathbf{N}_0|\mathbf{N}_1|\mathbf{N}_2]$, we have

$$\begin{aligned}\det(\mathbf{M}+\mathbf{N}) = &\det(\mathbf{M}_0|\mathbf{M}_1|\mathbf{M}_2) + \det(\mathbf{M}_0|\mathbf{M}_1|\mathbf{N}_2) + \\ &\det(\mathbf{M}_0|\mathbf{N}_1|\mathbf{M}_2) + \det(\mathbf{M}_0|\mathbf{N}_1|\mathbf{N}_2) + \\ &\det(\mathbf{N}_0|\mathbf{M}_1|\mathbf{M}_2) + \det(\mathbf{N}_0|\mathbf{M}_1|\mathbf{N}_2) + \\ &\det(\mathbf{N}_0|\mathbf{N}_1|\mathbf{M}_2) + \det(\mathbf{N}_0|\mathbf{N}_1|\mathbf{N}_2).\end{aligned} \tag{7.20}$$

This is an application of the general rule according to which the determinant of a sum of two $n \times n$ matrices is given by the sum of the 2^n determinants of the matrices obtained by any swapping of columns between the first and the second matrix; for the $n = 3$ case, this can also be immediately verified by rewriting $\det(\mathbf{a}|\mathbf{b}|\mathbf{c})$ as $\mathbf{a} \cdot (\mathbf{b} \times \mathbf{c})$.

Applying (7.20) to (7.14) with $\mathbf{M} = \mathbf{P}$ and $\mathbf{N} = t\,\mathbf{D}$, and remembering that, for any scalar k,

$$\det(k\,\mathbf{a}|\mathbf{b}|\mathbf{c}) = \det(\mathbf{a}|k\,\mathbf{b}|\mathbf{c}) = \det(\mathbf{a}|\mathbf{b}|k\,\mathbf{c}) = k\,\det(\mathbf{a}|\mathbf{b}|\mathbf{c}) \tag{7.21}$$

gives (7.15).

Chapter 8

Conclusions

In this thesis, we have analyzed common defects and shortcomings typical of textured 3D meshes generated with photogrammetry-based reconstruction techniques and proposed methods to address and correct some of them. While our contributions mainly target 3D data produced by photo-reconstruction reconstruction software, all the approaches we developed operate *a posteriori*, without prior knowledge of or affecting the reconstruction process. As a result, our algorithms are not restricted to this specific class of models and can be applied to generic textured 3D datasets.

We began our investigation by carrying out an extensive quantitative analysis of the issues and shortcomings of 3D textured meshes generated with photogrammetry software. Photogrammetry is a powerful technique that facilitates the creation of high-resolution textured models, but our findings highlighted recurring defects and peculiar features that complicate the adoption of such models in geometry processing algorithms and interactive systems. This investigation originated from first-hand practical experience working with photogrammetry models, and the large-scale quantitative analysis of Chapter 3 — across a number of different photo-reconstruction tools — corroborated the intuition that these recurring defects originate from intrinsic inaccuracies and challenges of the photogrammetric reconstruction task, and are not tied to specific implementations or lack of engineering in these tools.

As part of our analysis, we collected and released a large corpus of textured 3D meshes, which has been invaluable in the development and testing of the algorithms we presented in subsequent chapters. We also developed a tool to compute geometric and texturing quality measures on texture-mapped 3D objects. Our aspiration is that the Real-World Textured Things dataset (RWTT) and the TexMetro tool will help researchers and developers in future efforts to improve the quality of photo-reconstruction algorithms and implementations, as well as new tools targeted at post-processing the resulting models.

The first issue we tackled was the reduction of fragmentation in the texture maps reconstructed from photographic collections. A compact texture atlas brings unquestionable benefits to both processing algorithms and interactive applications. On the geometry processing side, a coarser texture atlas helps, e.g., simplification and remeshing tasks: these operations must preserve the texture-space connectivity of the atlas layout to avoid introducing severe texturing artifacts by compromising the integrity and injectivity of the underlying parametrization. An indirect effect is that improved simplification of textured models is also beneficial to multi-resolution encoding schemes, which are often adopted to optimize the distribution and rendering performance of high-resolution photo-reconstructed 3D data. On the rendering side,

there are benefits related to a more compact representation with fewer vertex replications to encode the texture seams, as well as improved coherency of texture access, which can hinder rendering performance in presence of extreme, but sometimes occurring nonetheless, texture fragmentation. To this end, we presented in Chapter 4 a method designed to reduce the atlas fragmentation while striving to preserve the original texture signal. Our method optimizes for mapping accuracy and texture sample distribution relative to the existing input texture map and was demonstrated to be a practical and robust solution: we field-tested it on the entire RWTT, witnessing drastic improvements in all of the targeted quality metrics.

Next, we turned our focus on removing texturing defects produced by inaccuracies in the reconstruction due to approximation errors or simply a lack of available photographic data. In Chapter 5 we presented a framework to fix texturing defects by performing inpainting operations on textured meshes directly. We took advantage of modern inpainting methods based on Convolutional Neural Networks and applied those to a specialized inpainting domain induced by a parametrization localized to the inpainting area. Texturing artifacts can sometimes be characterized analytically, by looking at the parametrization distortion or color discontinuities, but a general detection scheme for more challenging cases such as ghosting or blurring is more difficult to define. For this reason, we proposed both an interactive tool to manually fix texture defects with an inpainting or "fixing" brush, as well as an automatic tool to detect and inpaint a restricted class of texturing defects that can be characterized by analyzing the input parametrization and texture map. In training the network, we used the RWTT dataset to train the inpainting model on our specialized domain, achieving superior quality compared to image-space and global texture-space approaches.

Finally, we turned our attention to the problem of representing high-resolution textured mesh as μ-meshes, a recently introduced efficient representation of densely tessellated triangle meshes that can take advantage of hardware support [NVI22b]. μ-Meshes encode high-resolution geometry by combining a coarse base mesh, simple refinement rules, and displacement maps that are sampled on the barycentric grid and do not require an explicit parametrization of the base surface. In Chapter 6 we have presented a method designed to produce a high-quality base mesh geometry targeting isotropy of the sampling induced by the refinement rules, and optimized for the subsequent synthesis of the displacement-mapped surface. Our base mesh creation method improves over naive mesh simplification strategies that are oblivious to the computation of the displacement map, as well as state-of-the-art methods that address the problem. We also formulated a parametrization energy that is *displacement-aware* and can significantly reduce the mapping distortion induced by the surface deformation introduced by the displacement map. This helped improve the texturing quality and reduce the mapping distortion in presence of large deformations of the base mesh geometry, and was made possible because a μ-mesh displacement map does not require explicit UV mapping of the base mesh, breaking the circular dependency between displacement and parametrization distortion that affects traditional displacement mapping. Our solution makes no assumptions on the topology and triangulation quality of the input mesh, and can be used to produce compact representations of high-resolution 3D photo-reconstructions, facilitating the creation of assets for systems based on this new mesh representation.

8.1 List of publications

The content of Chapters 3, 4 and 5 largely mirrors the corresponding publications in [MCT21], [MPCT20] and [MCT23] respectively. The following is the complete list of published works

that emerged from the research presented in this thesis.

[MPCT20] Maggiordomo A., Ponchio F., Cignoni P., Tarini M.: Real-world textured things: A repository of textured models generated with modern photo-reconstruction tools. *Computer Aided Geometric Design 83* (2020), 101943

[MCT21] Maggiordomo A., Cignoni P., Tarini M.: Texture defragmentation for photo-reconstructed 3d models. *Computer Graphics Forum 40*, 2 (2021), 65–78

[MT22] Maggiordomo A., Tarini M.: Seamless Compressed Textures. In *Eurographics 2022 - Posters* (2022), Sauvage B., Hasic-Telalovic J., (Eds.), The Eurographics Association

[MFTT22] Maggiordomo A., Farronato M., Tartaglia G., Tarini M.: A method based on 3d affine alignment for the quantification of palatal expansion. *PLOS ONE 17*, 12 (12 2022), 1–14

[MCT23] Maggiordomo A., Cignoni P., Tarini M.: Texture inpainting for photogrammetric models. *Computer Graphics Forum,* to appear. (2023)

8.2 Future work

The work presented in this thesis is by no means definitive, and several future work directions are worth exploring.

The texture defragmentation algorithm we introduced in Chapter 4 is, to the best of our knowledge, the first approach to target specifically the goals set in Section 3.3; we think it is very likely that a more effective or more elegant strategy can be employed to reduce the fragmentation or the resampled areas more aggressively. For example, the local optimization itself could be designed to pursue the objective to avoid re-sampling inside each optimized triangle. This objective is "all-or-nothing" in nature, making its optimization difficult; recently, however, advanced numerical methods have been successfully employed over objectives posing a similar challenge [PTH*17, LKK*18a].

Our texture inpainting operator (Chapter 5) can be potentially employed in other frameworks for texture repairing beyond our interactive and automatic strategies presented in Section 5.4. An interesting future research direction is investigating robust methods to detect defective areas that need inpainting through a separated CNN, specifically trained for the purpose. This would enable the complete automation of the inpainting process, freeing artists from the lengthy process of inspecting the textured models, but is not trivial because there is no natural domain where to perform the detection; possibilities include the global texture or synthetic renderings. Another interesting direction is exploring the feasibility of integrating the inpainting process in reconstruction tools, exploiting information provided by the scene reconstruction to identify regions requiring inpainting (e.g., by providing a confidence value for each texel).

When converting open meshes to the μ-mesh representation, accurately preserving the border remains a challenging task. A promising direction to improve the representation power of displacement-mapped μ-meshes is to explore higher dimensional vector displacements (2D and 3D), to better reproduce borders and sharp salient shape features and reduce aliasing. An attractive feature of μ-mesh is that they do not require an explicit parametrization to represent the displacement map. Still, we have developed a method to optimize the parametrization to

accommodate the color data, which often accounts for a significant amount of extra storage. While our solution requires UV-mapping the base mesh, another option is to discard the UV-map altogether and adopt a *Mesh Colors*-like approach [YKH10, Yuk17]. It will be interesting to investigate how this will affect the visual quality of textured μ-Meshes when displacements induce significant deformation of the base mesh surface.

Bibliography

[3Df22] 3DFLOW SRL: 3DF Zephyr: The complete photogrammetry solution. https://www.3dflow.net/, 2022.

[ABK98] AMENTA N., BERN M., KAMVYSSELIS M.: A new voronoi-based surface reconstruction algorithm. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1998), SIGGRAPH '98, Association for Computing Machinery, p. 415–421.

[ACDL00] AMENTA N., CHOI S., DEY T. K., LEEKHA N.: A simple algorithm for homeomorphic surface reconstruction. In *Proceedings of the Sixteenth Annual Symposium on Computational Geometry* (New York, NY, USA, 2000), SCG '00, Association for Computing Machinery, p. 213–222.

[ACK01] AMENTA N., CHOI S., KOLLURI R. K.: The power crust. In *Proceedings of the Sixth ACM Symposium on Solid Modeling and Applications* (New York, NY, USA, 2001), SMA '01, Association for Computing Machinery, p. 249–266.

[ACK13] ATTENE M., CAMPEN M., KOBBELT L.: Polygon mesh repairing: An application perspective. *ACM Computing Surveys (CSUR) 45*, 2 (Mar. 2013).

[Ado21] ADOBE: Bake mesh maps. Substance 3D Documentation, https://helpx.adobe.com/substance-3d-painter/using/baking.html, jul 2021.

[Ado22] ADOBE: Substance 3D designer, 2022.

[AECdVDI05] ALLIEZ P., ÉRIC COLIN DE VERDIÈRE, DEVILLERS O., ISENBURG M.: Centroidal voronoi diagrams for isotropic surface remeshing. *Graphical Models 67*, 3 (2005), 204–231. Special Issue on SMI 2003.

[Agi22] AGISOFT LLC: Agisoft metashape: photogrammetric processing of digital images and 3D spatial data generation. https://www.agisoft.com, 2022.

[AL08] AUBRY R., LÖHNER R.: On the 'most normal' normal. *Communications in Numerical Methods in Engineering 24*, 12 (2008), 1641–1652.

[APK08] ALLENE C., PONS J.-P., KERIVEN R.: Seamless image-based texture atlases using multi-band blending. In *2008 19th International Conference on Pattern Recognition* (2008), pp. 1–4.

[APL14] AIGERMAN N., PORANNE R., LIPMAN Y.: Lifted bijections for low distortion surface mappings. *ACM Transactions on Graphics (TOG) 33*, 4 (2014), 69.

[ASS*09] AGARWAL S., SNAVELY N., SIMON I., SEITZ S. M., SZELISKI R.: Building rome in a day. In *2009 IEEE 12th International Conference on Computer Vision* (2009), pp. 72–79.

[ASSS10] Agarwal S., Snavely N., Seitz S. M., Szeliski R.: Bundle adjustment in the large. In *Computer Vision – ECCV 2010* (Berlin, Heidelberg, 2010), Daniilidis K., Maragos P., Paragios N., (Eds.), Springer Berlin Heidelberg, pp. 29–42.

[Aut19] Autodesk, Inc: Autodesk Maya 2019. `https://www.autodesk.com/products/maya/overview`, 2019.

[Aut22] Autodesk, Inc: ReCap Pro: Turn the physical into digital. `https://www.autodesk.com/products/recap/`, 2022.

[Bau02] Baumberg A.: Blending images for texturing 3d models. In *Proceedings of the British Machine Vision Conference* (2002), BMVA Press, pp. 38.1–38.10. doi:10.5244/C.16.38.

[BBC*01] Ballester C., Bertalmio M., Caselles V., Sapiro G., Verdera J.: Filling-in by joint interpolation of vector fields and gray levels. *IEEE Transactions on Image Processing 10*, 8 (2001), 1200–1211.

[BCE*13] Bommes D., Campen M., Ebke H.-C., Alliez P., Kobbelt L.: Integer-grid maps for reliable quad meshing. *ACM Trans. Graph. 32*, 4 (jul 2013).

[BCGB08] Ben-Chen M., Gotsman C., Bunin G.: Conformal flattening by curvature prescription and metric scaling. *Computer Graphics Forum 27*, 2 (2008), 449–458.

[Ben19] Bentley Systems, Inc: ContextCapture: 3D reality modeling software. `https://www.bentley.com/en/products/brands/contextcapture/`, 2014–2019.

[BHW11] Brown M., Hua G., Winder S.: Discriminative learning of local image descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence 33*, 1 (2011), 43–57.

[BK03] Boykov Y., Kolmogorov V.: Computing geodesics and minimal surfaces via graph cuts. In *Proceedings Ninth IEEE International Conference on Computer Vision* (2003), pp. 26–33 vol.1.

[BK04] Botsch M., Kobbelt L.: A remeshing approach to multiresolution modeling. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing* (New York, NY, USA, 2004), SGP '04, Association for Computing Machinery, p. 185–192.

[BKR17] Bi S., Kalantari N. K., Ramamoorthi R.: Patch-based optimization for image-based texture mapping. *ACM Trans. Graph. 36*, 4 (jul 2017).

[BL08] Burley B., Lacewell D.: Ptex: Per-face texture mapping for production rendering. In *Computer Graphics Forum* (2008), vol. 27, pp. 1155–1164.

[Ble22] Blender Development Team: Blender, 2022.

[BMRB16] Boscaini D., Masci J., Rodolà E., Bronstein M.: Learning shape correspondence with anisotropic convolutional neural networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems, Barcelona, Spain* (2016), NIPS'16, Curran Associates Inc., p. 3197–3205.

[Boi84] Boissonnat J.-D.: Geometric structures for three-dimensional shape representation. *ACM Trans. Graph. 3*, 4 (oct 1984), 266–286.

[Bro70] Broyden C. G.: The Convergence of a Class of Double-rank Minimization Algorithms 1. General Considerations. *IMA Journal of Applied Mathematics 6*, 1 (03 1970), 76–90.

[BS08] Boubekeur T., Schlick C.: A Flexible Kernel for Adaptive Mesh Refinement on GPU. *Computer Graphics Forum* (2008).

[BSCB00] Bertalmio M., Sapiro G., Caselles V., Ballester C.: Image inpainting. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques* (USA, 2000), SIGGRAPH '00, ACM Press/Addison-Wesley Publishing Co., p. 417–424.

[BSFG09] Barnes C., Shechtman E., Finkelstein A., Goldman D. B.: Patchmatch: A randomized correspondence algorithm for structural image editing. *ACM Trans. Graph. 28*, 3 (jul 2009).

[BTB02] Balmelli L., Taubin G., Bernardini F.: Space-optimized texture maps. In *Computer Graphics Forum* (2002), vol. 21, pp. 411–420.

[Cam17] Campen M.: Partitioning surfaces into quadrilateral patches: A survey. *Computer Graphics Forum 36*, 8 (2017), 567–588.

[Cap22] Capturing Reality: RealityCapture: Mapping and 3D modeling photogrammetry software. `https://www.capturingreality.com/`, 2022.

[CBSS17] Claici S., Bessmeltsev M., Schaefer S., Solomon J.: Isometry-aware preconditioning for mesh parameterization. *Comput. Graph. Forum 36*, 5 (aug 2017), 37–47.

[CCCS08] Callieri M., Cignoni P., Corsini M., Scopigno R.: Masked photo blending: Mapping dense photographic data set on high-resolution sampled 3d models. *Computers & Graphics 32*, 4 (2008), 464–473.

[CDF*04] Csurka G., Dance C., Fan L., Willamowski J., Bray C.: Visual categorization with bags of keypoints. In *Workshop on statistical learning in computer vision, ECCV* (2004), vol. 1, Prague, pp. 1–2.

[CFG*15] Chang A. X., Funkhouser T., Guibas L., Hanrahan P., Huang Q., Li Z., Savarese S., Savva M., Song S., Su H., Xiao J., Yi L., Yu F.: *ShapeNet: An Information-Rich 3D Model Repository*. Tech. Rep. arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

[CGG*04] Cignoni P., Ganovelli F., Gobbetti E., Marton F., Ponchio F., Scopigno R.: Adaptive tetrapuzzles: Efficient out-of-core construction and visualization of gigantic multiresolution polygonal models. *ACM Trans. Graph. 23*, 3 (aug 2004), 796–803.

[CGG*05] Cignoni P., Ganovelli F., Gobbetti E., Marton F., Ponchio F., Scopigno R.: Batched multi triangulation. In *Proceedings IEEE Visualization* (Conference held in Minneapolis, MI, USA, October 2005), IEEE Computer Society Press, pp. 207–214.

[CH02] Collins G., Hilton A.: Mesh Decimation for Displacement Mapping. In *Eurographics 2002 - Short Presentations* (2002), Eurographics Association.

[CL96] Curless B., Levoy M.: A volumetric method for building complex models from range images. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1996), SIGGRAPH '96, Association for Computing Machinery, p. 303–312.

[Cla76] Clark J. H.: Hierarchical geometric models for visible surface algorithms. *Commun. ACM 19*, 10 (oct 1976), 547–554.

[CMO97] Cohen J., Manocha D., Olano M.: Simplifying polygonal models using successive mappings. In *Proceedings of the 8th Conference on Visualization '97* (Washington, DC, USA, 1997), VIS '97, IEEE Computer Society Press, p. 395–ff.

[CMR*99] Cignoni P., Montani C., Rocchini C., Scopigno R., Tarini M.: Preserving attribute values on simplified meshes by resampling detail textures. *The Visual Computer 15*, 10 (1999), 519–539.

[COM98] Cohen J., Olano M., Manocha D.: Appearance-preserving simplification. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1998), SIGGRAPH '98, Association for Computing Machinery, p. 115–122.

[Coo84] Cook R. L.: Shade trees. *SIGGRAPH Comput. Graph. 18*, 3 (jan 1984), 223–231.

[CPLW19] Chu L., Pan H., Liu Y., Wang W.: Repairing man-made meshes via visual driven global optimization with minimum intrusion. *ACM Transactions on Graphics (TOG) 38*, 6 (2019).

[CPT04] Criminisi A., Perez P., Toyama K.: Region filling and object removal by exemplar-based image inpainting. *IEEE Transactions on Image Processing 13*, 9 (2004), 1200–1212.

[CRS98] Cignoni P., Rocchini C., Scopigno R.: Metro: Measuring error on simplified surfaces. *Computer Graphics Forum 17*, 2 (1998), 167–174.

[CVHC08] Campbell N. D. F., Vogiatzis G., Hernández C., Cipolla R.: Using multiple hypotheses to improve depth-maps for multi-view stereo. In *Computer Vision – ECCV 2008* (Berlin, Heidelberg, 2008), Forsyth D., Torr P., Zisserman A., (Eds.), Springer Berlin Heidelberg, pp. 766–779.

[CZMK16] Choi S., Zhou Q.-Y., Miller S., Koltun V.: A large dataset of object scans, 2016.

[DCOY03] Drori I., Cohen-Or D., Yeshurun H.: Fragment-based image completion. *ACM Trans. Graph. 22*, 3 (jul 2003), 303–312.

[DCS*17] Dai A., Chang A. X., Savva M., Halber M., Funkhouser T., Niessner M.: ScanNet: Richly-annotated 3D reconstructions of indoor scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (July 2017).

[DDZ11] Deng Y., Dai Q., Zhang Z.: Graph laplace for occluded face completion and recognition. *IEEE Transactions on Image Processing 20*, 8 (2011), 2329–2338.

[DEGN99] Dey T. K., Edelsbrunner H., Guha S., Nekhayev D. V.: Topology preserving edge contraction. *Publications de l'Institut Mathématique*, 86 (1999), 23–45.

[DFMP98] De Floriani L., Magillo P., Puppo E.: Efficient implementation of multi-triangulations. In *Proceedings Visualization '98 (Cat. No.98CB36276)* (1998), pp. 43–50.

[DG03] Dey T. K., Goswami S.: Tight Cocone: A Water-tight Surface Reconstructor . *Journal of Computing and Information Science in Engineering 3*, 4 (12 2003), 302–307.

[DMA02] Desbrun M., Meyer M., Alliez P.: Intrinsic Parameterizations of Surface Meshes. *Computer Graphics Forum* (2002).

[DMK03] Degener P., Meseth J., Klein R.: An adaptable surface parametrization method. In *The 12th International Meshing Roundtable 2003* (Sept. 2003).

[DMR18] DeTone D., Malisiewicz T., Rabinovich A.: Superpoint: Self-supervised interest point detection and description. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR) workshops* (2018), pp. 224–236.

[DS95] Dey T., Schipper H.: A new technique to compute polygonal schema for 2-manifolds with application to null-homotopy detection, 1995.

[EAAMA20] Elharrouss O., Almaadeed N., Al-Maadeed S., Akbari Y.: Image inpainting: A review. *Neural Processing Letters 51*, 2 (2020), 2007–2028.

[EDD*95] Eck M., DeRose T., Duchamp T., Hoppe H., Lounsbery M., Stuetzle W.: Multiresolution analysis of arbitrary meshes. In *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1995), SIGGRAPH '95, ACM, pp. 173–182.

[EF01] Efros A. A., Freeman W. T.: Image quilting for texture synthesis and transfer. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, Los Angeles, California* (New York, NY, USA, 2001), SIGGRAPH '01, Association for Computing Machinery, p. 341–346.

[EHP04] Erickson J., Har-Peled S.: Optimally cutting a surface into a disk. *Discrete & Computational Geometry 31*, 1 (2004), 37–59.

[EL99] Efros A. A., Leung T. K.: Texture synthesis by non-parametric sampling. In *Proceedings of the Seventh IEEE International Conference on Computer Vision (ICCV), Kerkyra, Greece* (1999), vol. 2, pp. 1033–1038.

[Emb19] Embark Studios: Texture synthesis, 2019.

[Epi19] Epic Games: Unreal engine 4 marketplace. `https://www.unrealengine.com/marketplace/`, 2004 – 2019.

[FB81] Fischler M. A., Bolles R. C.: Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM 24*, 6 (jun 1981), 381–395.

[FCSS10] Furukawa Y., Curless B., Seitz S. M., Szeliski R.: Towards internet-scale multi-view stereo. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2010), pp. 1434–1441.

[FDGM18] Fayer J., Durix B., Gasparini S., Morin G.: Texturing and inpainting a

complete tubular 3D object reconstructed from partial views. *Computers and Graphics 74* (Aug. 2018), 126 – 136.

[FH05] Floater M. S., Hormann K.: Surface parameterization: a tutorial and survey. In *Advances in multiresolution for geometric modelling*, Dodgson N. A., Floater M. S., Sabin M. A., (Eds.). Springer Verlag, 2005, pp. 157–186.

[FH15] Furukawa Y., Hernández C.: Multi-view stereo: A tutorial. *Foundations and Trends® in Computer Graphics and Vision 9*, 1-2 (2015), 1–148.

[Fie00] Field D. A.: Qualitative measures for initial meshes. *International Journal for Numerical Methods in Engineering 47*, 4 (2000), 887–906.

[Fir16] Firman M.: RGBD datasets: Past, present and future. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops* (June 2016).

[Fle70] Fletcher R.: A new approach to variable metric algorithms. *The Computer Journal 13*, 3 (01 1970), 317–322.

[FLG14] Fuhrmann S., Langguth F., Goesele M.: MVE - A Multi-View Reconstruction Environment. In *Eurographics Workshop on Graphics and Cultural Heritage* (2014), Klein R., Santos P., (Eds.), The Eurographics Association.

[FLG15] Fu X.-M., Liu Y., Guo B.: Computing locally injective mappings by advanced mips. *ACM Trans. Graph. 34*, 4 (July 2015), 71:1–71:12.

[FLM92] Faugeras O. D., Luong Q. T., Maybank S. J.: Camera self-calibration: Theory and experiments. In *Computer Vision — ECCV'92* (Berlin, Heidelberg, 1992), Sandini G., (Ed.), Springer Berlin Heidelberg, pp. 321–334.

[Flo97] Floater M. S.: Parametrization and smooth approximation of surface triangulations. *Computer aided geometric design 14*, 3 (1997), 231–250.

[FP10] Furukawa Y., Ponce J.: Accurate, dense, and robust multiview stereopsis. *IEEE Transactions on Pattern Analysis and Machine Intelligence 32*, 8 (2010), 1362–1376.

[FS93] Funkhouser T. A., Séquin C. H.: Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1993), SIGGRAPH '93, Association for Computing Machinery, p. 247–254.

[GA16] Guidi G., Angheleddu D.: Displacement mapping as a metric tool for optimizing mesh models originated by 3d digitization. *J. Comput. Cult. Herit. 9*, 2 (may 2016).

[GAKC14] Goldlücke B., Aubry M., Kolev K., Cremers D.: A super-resolution framework for high-accuracy multiview reconstruction. *International Journal of Computer Vision 106* (2014), 172 – 191.

[GEB16] Gatys L. A., Ecker A. S., Bethge M.: Image style transfer using convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2016).

[GG*03] P., Ganovelli F., Gobbetti E., Marton F., Ponchio F., Scopigno R.: Bdam: Batched dynamic adaptive meshes for high performance terrain visualization. *Computer Graphics Forum 22*, 3 (sept 2003), 505–514.

[GGC*21] Griwodz C., Gasparini S., Calvet L., Gurdjos P., Castan F., Maujean B., De Lillo G., Lanthony Y.: AliceVision Meshroom: An open-source 3d reconstruction pipeline. In *Proceedings of the 12th ACM Multimedia Systems Conference, Istanbul, Turkey* (New York, NY, USA, 2021), MMSys '21, Association for Computing Machinery, p. 241–247.

[GGH02] Gu X., Gortler S. J., Hoppe H.: Geometry images. *ACM Trans. Graph. 21*, 3 (jul 2002), 355–361.

[GGSC96] Gortler S. J., Grzeszczuk R., Szeliski R., Cohen M. F.: The lumigraph. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques* (1996), SIGGRAPH '96, pp. 43–54.

[GH97] Garland M., Heckbert P. S.: Surface simplification using quadric error metrics. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques* (USA, 1997), SIGGRAPH '97, ACM Press/Addison-Wesley Publishing Co., p. 209–216.

[GH98] Garland M., Heckbert P.: Simplifying surfaces with color and texture using quadric error metrics. In *Proceedings Visualization '98 (Cat. No.98CB36276)* (1998), pp. 263–269.

[GLS15] Galliani S., Lasinger K., Schindler K.: Massively parallel multiview stereopsis by surface normal diffusion. In *2015 IEEE International Conference on Computer Vision (ICCV)* (2015), pp. 873–881.

[Gol70] Goldfarb D.: A family of variable-metric methods derived by variational means. *Mathematics of Computation 24*, 109 (1970), 23–26.

[Goo19] Google Scholar: "Metro: Measuring error on simplified surfaces" [CRS98] citation page. `https://scholar.google.it/scholar?cites=4029217030792744566`, 2019.

[GSC*07] Goesele M., Snavely N., Curless B., Hoppe H., Seitz S. M.: Multi-view stereo for community photo collections. In *2007 IEEE 11th International Conference on Computer Vision* (2007), pp. 1–8.

[GVSS00] Guskov I., Vidimče K., Sweldens W., Schröder P.: Normal meshes. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques* (USA, 2000), SIGGRAPH '00, ACM Press/Addison-Wesley Publishing Co., p. 95–102.

[GWO*10] Gal R., Wexler Y., Ofek E., Hoppe H., Cohen-Or D.: Seamless montage for texturing models. *Computer Graphics Forum 29*, 2 (2010), 479–486.

[GWY*21] Gao L., Wu T., Yuan Y.-J., Lin M.-X., Lai Y.-K., Zhang H.: Tm-net: Deep generative networks for textured meshes. *ACM Trans. Graph. 40*, 6 (dec 2021).

[GYH21] Guo X., Yang H., Huang D.: Image inpainting via conditional texture and structure dual generation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), Virtual* (2021), pp. 14134–14143.

[GZ05] Garland M., Zhou Y.: Quadric-based simplification in any dimension. *ACM Trans. Graph. 24*, 2 (apr 2005), 209–239.

[Har97] Hartley R.: In defense of the eight-point algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence 19*, 6 (1997), 580–593.

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

[HDGN17] Huang J., Dai A., Guibas L., Niessner M.: 3dlite: Towards commodity 3d scanning for content creation. *ACM Trans. Graph. 36*, 6 (nov 2017).

[HE07] Hays J., Efros A. A.: Scene completion using millions of photographs. *ACM Trans. Graph. 26*, 3 (jul 2007), 4–es.

[Hel21] Help A.: Transfer maps. Autodesk Knowledge Network, `https://knowledge.autodesk.com/support/maya/learn?s=Transfer+Maps`, dec 2021.

[HG00] Hormann K., Greiner G.: MIPS: An efficient global parametrization method. In *Curve and Surface Design: Saint-Malo 1999*, Laurent P.-J., Sablonnière P., Schumaker L. L., (Eds.), Innovations in Applied Mathematics. Vanderbilt University Press, Nashville, TN, 2000, pp. 153–162.

[HGC99] Hormann K., Greiner G., Campagna S.: Hierarchical parametrization of triangulated surfaces. In *Proceedings of Vision, Modeling, and Visualization 1999* (Erlangen, Germany, Nov. 1999), Girod B., Niemann H., Seidel H.-P., (Eds.), infix, pp. 219–226.

[Hig89] Higham N. J.: Matrix nearness problems and applications. *Applications of Matrix Theory 22* (1989).

[HK06] Hornung A., Kobbelt L.: Robust Reconstruction of Watertight 3D Models from Non-uniformly Sampled Point Clouds Without Normal Information. In *Symposium on Geometry Processing* (2006), Sheffer A., Polthier K., (Eds.), The Eurographics Association.

[HKAK14] Huang J.-B., Kang S. B., Ahuja N., Kopf J.: Image completion using planar structure guidance. *ACM Trans. Graph. 33*, 4 (jul 2014).

[HLS07] Hormann K., Lévy B., Sheffer A.: Mesh parameterization: Theory and practice. In *SIGGRAPH 2007 Course Notes* (San Diego, CA, Aug. 2007), no. 2, ACM Press, pp. vi+115.

[Hop96] Hoppe H.: Progressive meshes. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1996), SIGGRAPH '96, Association for Computing Machinery, p. 99–108.

[Hop99] Hoppe H.: New quadric metric for simplifying meshes with appearance attributes. In *Proceedings Visualization '99 (Cat. No.99CB37067)* (1999), pp. 59–510.

[HPN*16] Hua B., Pham Q., Nguyen D. T., Tran M., Yu L., Yeung S.: SceneNN: A scene meshes dataset with annotations. In *Fourth International Conference on 3D Vision*

(3DV) (Conference held in Stanford, CA, USA, 2016), IEEE Computer Society, pp. 92–101.

[HPS08] Hormann K., Polthier K., Sheffer A.: Mesh parameterization: Theory and practice. In *ACM SIGGRAPH ASIA 2008 Courses* (2008), SIGGRAPH Asia '08.

[HSDF15] Heinly J., Schönberger J. L., Dunn E., Frahm J.-M.: Reconstructing the world in six days. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), pp. 3287–3295.

[HYB*17] Hu K., Yan D.-M., Bommes D., Alliez P., Benes B.: Error-bounded and feature preserving surface remeshing with minimal angle improvement. *IEEE Transactions on Visualization and Computer Graphics 23*, 12 (2017), 2560–2573.

[HZ03] Hartley R., Zisserman A.: *Multiple View Geometry in Computer Vision*, 2 ed. Cambridge University Press, USA, 2003.

[HZ04] Hartley R., Zisserman A.: *Multiple View Geometry in Computer Vision*, 2 ed. Cambridge University Press, 2004.

[HZN*18] Huang J., Zhou Y., Niessner M., Shewchuk J. R., Guibas L. J.: Quadriflow: A scalable and robust method for quadrangulation. *Computer Graphics Forum 37*, 5 (2018), 147–160.

[HZW*21] Hui S., Zhou S., Wan X., Wang J., Deng Y., Wu Y., Gong Z., Wang J.: Auxiliary loss adaption for image inpainting. *arXiv preprint arXiv:2111.07279* (2021).

[HZY*19] Huang J., Zhang H., Yi L., Funkhouser T., Niessner M., Guibas L. J.: Texturenet: Consistent local parametrizations for learning from high-resolution signals on meshes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, California* (June 2019).

[Ima21] ImageMagick:, 2021.

[IS15] Ioffe S., Szegedy C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[ISSI17] Iizuka S., Simo-Serra E., Ishikawa H.: Globally and locally consistent image completion. *ACM Trans. Graph. 36*, 4 (jul 2017).

[itS19] itSeez3D: itSeez3D: #1 mobile 3D scanner app for ipad. `https://itseez3d.com/`, 2019.

[JSP17] Jiang Z., Schaefer S., Panozzo D.: Simplicial complex augmentation framework for bijective maps. *ACM Trans. Graph. 36*, 6 (nov 2017).

[JSZP20] Jiang Z., Schneider T., Zorin D., Panozzo D.: Bijective projection in a shell. *ACM Trans. Graph. 39*, 6 (nov 2020).

[JT03] Jia J., Tang C.-K.: Image repairing: Robust image synthesis by adaptive nd tensor voting. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), Madison, Wisconsin* (USA, 2003), CVPR'03, IEEE Computer Society, p. 643–650.

[JTPSH15] Jakob W., Tarini M., Panozzo D., Sorkine-Hornung O.: Instant field-aligned meshes. *ACM Trans. Graph. 34*, 6 (nov 2015).

[Ju04] Ju T.: Robust repair of polygonal models. *ACM Transactions on Graphics (TOG) 23*, 3 (Aug. 2004), 888–895.

[KBH06] Kazhdan M., Bolitho M., Hoppe H.: Poisson Surface Reconstruction. In *Symposium on Geometry Processing* (2006), Sheffer A., Polthier K., (Eds.), The Eurographics Association.

[KGL16] Kovalsky S. Z., Galun M., Lipman Y.: Accelerated quadratic proxy for geometric optimization. *ACM Trans. Graph. 35*, 4 (jul 2016).

[KH13] Kazhdan M., Hoppe H.: Screened poisson surface reconstruction. *ACM Trans. Graph. 32*, 3 (jul 2013).

[KKDK12] Kopf J., Kienzle W., Drucker S., Kang S. B.: Quality prediction for image completion. *ACM Trans. Graph. 31*, 6 (nov 2012).

[KMJ*19] Koch S., Matveev A., Jiang Z., Williams F., Artemov A., Burnaev E., Alexa M., Zorin D., Panozzo D.: ABC: A big cad model dataset for geometric deep learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2019).

[KPF*22] Khan D., Plopski A., Fujimoto Y., Kanbara M., Jabeen G., Zhang Y. J., Zhang X., Kato H.: Surface remeshing: A systematic literature review of methods and research directions. *IEEE Transactions on Visualization and Computer Graphics 28*, 3 (2022), 1680–1713.

[KPW95] Kim D.-S., Papalambros P. Y., Woo T. C.: Tangent, normal, and visibility cones on bézier surfaces. *Computer Aided Geometric Design 12*, 3 (1995), 305–320.

[KSS06] Kharevych L., Springborn B., Schröder P.: Discrete conformal mappings via circle patterns. *ACM Trans. Graph. 25*, 2 (apr 2006), 412–438.

[KSSH14] Köhler R., Schuler C., Schölkopf B., Harmeling S.: Mask-specific inpainting with deep neural networks. In *German conference on pattern recognition* (2014), Springer, pp. 523–534.

[KSW21] Karis B., Stubbe R., Wihlidal G.: Nanite – a deep dive. In *Advances in Real-Time Rendering in Games. ACM SIGGRAPH 2021 Courses)* (2021), SIGGRAPH '21.

[LB07] Lempitsky V., Boykov Y.: Global optimization for shape fitting. In *2007 IEEE Conference on Computer Vision and Pattern Recognition* (2007), pp. 1–8.

[LC87] Lorensen W. E., Cline H. E.: Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1987), SIGGRAPH '87, Association for Computing Machinery, p. 163–169.

[LD08] Lorenz H., Döllner J.: Dynamic mesh refinement on gpu using geometry shaders.

[LFJG17] Liu S., Ferguson Z., Jacobson A., Gingold Y.: Seamless: Seam erasure and seam-aware decoupling of shape from mesh resolution. *ACM Trans. Graph. 36*, 6 (nov 2017).

[LFY*19] Liu H.-Y., Fu X.-M., Ye C., Chai S., Liu L.: Atlas refinement with bounded packing efficiency. *ACM Trans. Graph. 38*, 4 (2019).

[LGCP05] Levoy M., Gerth J., Curless B., Pull K.: The stanford 3D scanning repository. `https://graphics.stanford.edu/data/3Dscanrep/`, 2005.

[LH87] Longuet-Higgins H.: A computer algorithm for reconstructing a scene from two projections. In *Readings in Computer Vision*, Fischler M. A., Firschein O., (Eds.). Morgan Kaufmann, San Francisco (CA), 1987, pp. 61–62.

[LH06] Lefebvre S., Hoppe H.: Perfect spatial hashing. *ACM Trans. Graph. 25*, 3 (2006), 579–588.

[LI07] Lempitsky V., Ivanov D.: Seamless mosaicing of image-based texture maps. In *2007 IEEE Conference on Computer Vision and Pattern Recognition* (2007), pp. 1–6.

[LKK*18a] Li M., Kaufman D. M., Kim V. G., Solomon J., Sheffer A.: Optcuts: Joint optimization of surface cuts and parameterization. *ACM Trans. Graph. 37*, 6 (dec 2018).

[LKK*18b] Li M., Kaufman D. M., Kim V. G., Solomon J., Sheffer A.: Optcuts: Joint optimization of surface cuts and parameterization. *ACM Trans. Graph. 37*, 6 (2018), 1–13.

[LKL02] Lee Y., Kim H. S., Lee S.: Mesh parameterization with a virtual boundary. *Computers & Graphics 26*, 5 (2002), 677–686.

[LMH00] Lee A., Moreton H., Hoppe H.: Displaced subdivision surfaces. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques* (USA, 2000), SIGGRAPH '00, ACM Press/Addison-Wesley Publishing Co., p. 85–94.

[Low04] Lowe D. G.: Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision 60* (2004), 91 – 110.

[LPC*00] Levoy M., Pulli K., Curless B., Rusinkiewicz S., Koller D., Pereira L., Ginzton M., Anderson S., Davis J., Ginsberg J., Shade J., Fulk D.: The digital Michelangelo project: 3D scanning of large statues. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques* (USA, 2000), SIGGRAPH '00, ACM Press/Addison-Wesley Publishing Co., pp. 131–144.

[LPK09] Labatut P., Pons J.-P., Keriven R.: Robust and efficient surface reconstruction from range data. *Computer Graphics Forum 28*, 8 (2009), 2275–2290.

[LPRM02] Lévy B., Petitjean S., Ray N., Maillot J.: Least squares conformal maps for automatic texture atlas generation. In *ACM transactions on graphics (TOG)* (2002), vol. 21, ACM, pp. 362–371.

[LQ05] Lhuillier M., Quan L.: A quasi-dense approach to surface reconstruction from uncalibrated images. *IEEE Transactions on Pattern Analysis and Machine Intelligence 27*, 3 (2005), 418–433.

[LRS*18] Liu G., Reda F. A., Shih K. J., Wang T.-C., Tao A., Catanzaro B.: Image inpainting for irregular holes using partial convolutions. In *The European Conference on Computer Vision (ECCV)* (2018).

[LSW*18] Liu G., Shih K. J., Wang T.-C., Reda F. A., Sapra K., Yu Z., Tao A., Catanzaro B.: Partial convolution based padding. In *arXiv preprint arXiv:1811.11718* (2018).

[LT00] Lindstrom P., Turk G.: Image-driven simplification. *ACM Trans. Graph. 19*, 3 (jul 2000), 204–241.

[LTB19] Legrand H., Thiery J.-M., Boubekeur T.: Filtered quadrics for high-speed geometry smoothing and clustering. *Computer Graphics Forum 38*, 1 (2019), 663–677.

[LTH*17] Ledig C., Theis L., Huszar F., Caballero J., Cunningham A., Acosta A., Aitken A., Tejani A., Totz J., Wang Z., Shi W.: Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (July 2017), pp. 105–114.

[LVS18] Limper M., Vining N., SHEFFER A.: Box cutter: Atlas refinement for efficient packing via void elimination. *ACM Trans. Graph. 37*, 4 (2018).

[LWL*09] Liu Y., Wang W., Lévy B., Sun F., Yan D.-M., Lu L., Yang C.: On centroidal voronoi tessellation—energy smoothness and fast computation. *ACM Trans. Graph. 28*, 4 (sep 2009).

[LYNF18] Liu L., Ye C., Ni R., Fu X.-M.: Progressive parameterizations. *ACM Trans. Graph. 37*, 4 (jul 2018).

[LZX*08] Liu L., Zhang L., Xu Y., Gotsman C., Gortler S. J.: A local/global approach to mesh parameterization. In *Proceedings of the Symposium on Geometry Processing* (Goslar, DEU, 2008), SGP '08, Eurographics Association, p. 1495–1504.

[Mar22] Marmoset: Baking in toolbag. `https://marmoset.co/toolbag/baking/`, nov 2022.

[MBBV15] Masci J., Boscaini D., Bronstein M. M., Vandergheynst P.: Geodesic convolutional neural networks on riemannian manifolds. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV) Santiago, Chile) Workshops* (December 2015).

[MCT21] Maggiordomo A., Cignoni P., Tarini M.: Texture defragmentation for photo-reconstructed 3d models. *Computer Graphics Forum 40*, 2 (2021), 65–78.

[MCT23] Maggiordomo A., Cignoni P., Tarini M.: Texture inpainting for photogrammetric models. *Computer Graphics Forum,* to appear. (2023).

[MFTT22] Maggiordomo A., Farronato M., Tartaglia G., Tarini M.: A method based on 3d affine alignment for the quantification of palatal expansion. *PLOS ONE 17*, 12 (12 2022), 1–14.

[Mic22] Microsoft: Simplygon: The standard in 3d games content optimization. `https://www.simplygon.com/`, dec 2022.

[MMM13] Moulon P., Monasse P., Marlet R.: Global fusion of relative motions for robust, accurate and scalable structure from motion. In *2013 IEEE International Conference on Computer Vision* (2013), pp. 3248–3255.

[MMPM16] Moulon P., Monasse P., Perrot R., Marlet R.: OpenMVG: Open multiple view geometry. In *International Workshop on Reproducible Research in Pattern Recognition* (2016), Springer, pp. 60–74.

[Mor78] Moré J. J.: The levenberg-marquardt algorithm: Implementation and theory. In *Numerical Analysis* (Berlin, Heidelberg, 1978), Watson G. A., (Ed.), Springer Berlin Heidelberg, pp. 105–116.

[MPCT20] Maggiordomo A., Ponchio F., Cignoni P., Tarini M.: Real-world textured things: A repository of textured models generated with modern photo-reconstruction tools. *Computer Aided Geometric Design 83* (2020), 101943.

[MPK09] Mohammed U., Prince S. J. D., Kautz J.: Visio-lization: Generating novel facial images. *ACM Trans. Graph. 28*, 3 (jul 2009).

[MT22] Maggiordomo A., Tarini M.: Seamless Compressed Textures. In *Eurographics 2022 - Posters* (2022), Sauvage B., Hasic-Telalovic J., (Eds.), The Eurographics Association.

[MTAD08] Mullen P., Tong Y., Alliez P., Desbrun M.: Spectral conformal parameterization. In *Proceedings of the Symposium on Geometry Processing* (2008), Eurographics Association, pp. 1487–1494.

[MW08] McGuire M., Whitson K.: Indirection mapping for quasi-conformal relief texturing. In *Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2008), I3D '08, Association for Computing Machinery, p. 191–198.

[MZ12] Myles A., Zorin D.: Global parametrization by incremental flattening. *ACM Trans. Graph. 31*, 4 (2012), 1–11.

[Nis04] Nister D.: An efficient solution to the five-point relative pose problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence 26*, 6 (2004), 756–770.

[Noc80] Nocedal J.: Updating quasi-newton matrices with limited storage. *Mathematics of Computation 35*, 151 (1980), 773–782.

[NS06] Nister D., Stewenius H.: Scalable recognition with a vocabulary tree. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)* (2006), vol. 2, pp. 2161–2168.

[NS11] Nöll T., Strieker D.: Efficient packing of arbitrary shaped charts for automatic texture atlas generation. *Computer Graphics Forum 30*, 4 (2011), 1309–1317.

[NVI22a] NVIDIA: Micromesh graphics primitive for micro triangles. `https://developer.nvidia.com/rtx/ray-tracing/micro-mesh`, nov 2022.

[NVI22b] NVIDIA: Nvidia Ada GPU architecture. `https://www.nvidia.com/it-it/geforce/ada-lovelace-architecture`, nov 2022.

[NW06] Nocedal J., Wright S.: *Numerical optimization*, 2. ed. ed. Springer series in operations research and financial engineering. Springer, New York, NY, 2006.

[Opa19] Opara A.: More like this, please! texture synthesis & remixing from a single example. Conference on procedural content generation for games (Everything Procedural), Breda, Netherlands, 2019. Talk.

[PCCS11] Pietroni N., Corsini M., Cignoni P., Scopigno R.: An interactive local flattening operator to support digital investigations on artwork surfaces. *IEEE Transaction on Visualization and Computer Graphics (Proceedings of Visualization 2011, Providence, Rhode Island) 17*, 12 (December 2011), 1989–1996.

[PCI*07] Philbin J., Chum O., Isard M., Sivic J., Zisserman A.: Object retrieval with large vocabularies and fast spatial matching. In *2007 IEEE Conference on Computer Vision and Pattern Recognition* (2007), pp. 1–8.

[Pix19] Pix4D SA: Pix4Dmapper: Professional photogrammetry software for drone mapping. `https://www.pix4d.com/product/pix4dmapper-photogrammetry-software/`, 2011–2019.

[PKD*16] Pathak D., Krahenbuhl P., Donahue J., Darrell T., Efros A. A.: Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 2536–2544.

[PP93] Pinkall U., Polthier K.: Computing discrete minimal surfaces and their conjugates. *Experimental mathematics 2*, 1 (1993), 15–36.

[PPT*11] Panozzo D., Puppo E., Tarini M., Pietroni N., Cignoni P.: Automatic construction of quad-based subdivision surfaces using fitmaps. *IEEE Transactions on Visualization and Computer Graphics 17*, 10 (2011), 1510–1520.

[PS00] Portilla J., Simoncelli E. P.: A parametric texture model based on joint statistics of complex wavelet coefficients. *International Journal of Computer Vision 40* (2000), 49–71.

[PSK06] Pavić D., Schönefeld V., Kobbelt L.: Interactive image completion with perspective correction. *The Visual Computer 22*, 9 (2006), 671–681.

[PTH*17] Poranne R., Tarini M., Huber S., Panozzo D., Sorkine-Hornung O.: Autocuts: Simultaneous distortion and cut optimization for UV mapping. *ACM Trans. Graph. 36*, 6 (nov 2017).

[PTW13] Pal K., Terras M., Weyrich T.: Interactive exploration and flattening of deformed historical documents. *Computer Graphics Forum 32*, 2pt3 (2013), 327–334.

[Pup98] Puppo E.: Variable resolution triangulations. *Computational Geometry 11*, 3 (1998), 219–238.

[RB93] Rossignac J., Borrel P.: Multi-resolution 3d approximations for rendering complex scenes. In *Modeling in Computer Graphics* (Berlin, Heidelberg, 1993), Falcidieno B., Kunii T. L., (Eds.), Springer Berlin Heidelberg, pp. 455–465.

[RCD*13] Ranzuglia G., Callieri M., Dellepiane M., Cignoni P., Scopigno R.: Meshlab as a complete tool for the integration of photos and color with high resolution 3D geometry data. In *CAA 2012 Conference Proceedings* (2013), Pallas Publications - Amsterdam University Press (AUP), pp. 406–416.

[RCMS99] Rocchini C., Cignoni P., Montani C., Scopigno R.: Multiple Textures Stitching and Blending on 3D Objects. In *Eurographics Workshop on Rendering* (1999), Lischinski D., Larson G. W., (Eds.), The Eurographics Association.

[RDS*15] Russakovsky O., Deng J., Su H., Krause J., Satheesh S., Ma S., Huang Z., Karpathy A., Khosla A., Bernstein M., Berg A. C., Fei-Fei L.: ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV) 115*, 3 (2015), 211–252.

[RMC16] Radford A., Metz L., Chintala S.: Unsupervised representation learning with deep convolutional generative adversarial networks, 2016.

[RNLL10] Ray N., Nivoliers V., Lefebvre S., Lévy B.: Invisible seams. *Computer Graphics Forum 29*, 4 (2010), 1489–1496.

[RPB15] Ronneberger O., P.Fischer, Brox T.: U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)* (2015), vol. 9351 of *LNCS*, Springer, pp. 234–241. (available on arXiv:1505.04597 [cs.CV]).

[RPPSH17] Rabinovich M., Poranne R., Panozzo D., Sorkine-Hornung O.: Scalable locally injective mappings. *ACM Transactions on Graphics 36*, 2 (Apr. 2017), 16:1–16:16.

[RSB*20] Rüther H., Schröder R., Bhurtha R., Wessels S., McDonald B.: Kua ruins heritage spatial documentation metadata dataset, 1 2020.

[RSN*14] Remondino F., Spera M. G., Nocerino E., Menna F., Nex F.: State of the art in high density image matching. *The Photogrammetric Record 29*, 146 (2014), 144–166.

[RXYS15] Ren J. S., Xu L., Yan Q., Sun W.: Shepard convolutional neural networks. *Advances in Neural Information Processing Systems 28* (2015), 901–909.

[SA07] Sorkine O., Alexa M.: As-rigid-as-possible surface modeling. In *Proceedings of the Fifth Eurographics Symposium on Geometry Processing* (2007), SGP '07, pp. 109–116.

[SAG03] Surazhsky V., Alliez P., Gotsman C.: *Isotropic Remeshing of Surfaces: a Local Parameterization Approach*. Research Report RR-4967, INRIA, Oct. 2003.

[SAPH04] Schreiner J., Asirvatham A., Praun E., Hoppe H.: Inter-surface mapping. In *ACM SIGGRAPH 2004 Papers* (New York, NY, USA, 2004), SIGGRAPH '04, ACM, pp. 870–877.

[SC17] Sawhney R., Crane K.: Boundary first flattening. *ACM Trans. Graph. 37*, 1 (2017).

[SC18] Sharp N., Crane K.: Variational surface cutting. *ACM Trans. Graph. 37*, 4 (jul 2018).

[SCD*06] Seitz S., Curless B., Diebel J., Scharstein D., Szeliski R.: A comparison and evaluation of multi-view stereo reconstruction algorithms. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)* (2006), vol. 1, pp. 519–528.

[SCOGL02] Sorkine O., Cohen-Or D., Goldenthal R., Lischinski D.: Bounded-distortion piecewise mesh parameterization. In *Proceedings of IEEE Visualization* (2002), IEEE Computer Society, pp. 355–362.

[SDMR20] Sarlin P.-E., DeTone D., Malisiewicz T., Rabinovich A.: Superglue: Learning feature matching with graph neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2020).

[SF16] Schönberger J. L., Frahm J.-M.: Structure-from-motion revisited. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 4104–4113.

[SGSH02] Sander P. V., Gortler S. J., Snyder J., Hoppe H.: Signal-specialized parametrization. In *Proceedings of the 13th Eurographics Workshop on Rendering* (2002), EGRW '02, Eurographics Association, pp. 87–98.

[SGW06] Schmidt R., Grimm C., Wyvill B.: Interactive decal compositing with discrete exponential maps. In *ACM SIGGRAPH 2006 Papers* (New York, NY, USA, 2006), SIGGRAPH '06, Association for Computing Machinery, p. 605–613.

[SH02] Sheffer A., Hart J. C.: Seamster: inconspicuous low-distortion texture seam layout. In *Proceedings of IEEE Visualization* (2002), IEEE Computer Society, pp. 291–298.

[Sha70] Shanno D. F.: Conditioning of quasi-newton methods for function minimization. *Mathematics of Computation 24*, 111 (1970), 647–656.

[SHR16] Sorkine-Hornung O., Rabinovich M.: Least-squares rigid motion using svd, 2016. Technical note.

[SJZP19] Shen H., Jiang Z., Zorin D., Panozzo D.: Progressive embedding. *ACM Trans. Graph. 38*, 4 (jul 2019).

[SKCA20] Saint A., Kacem A., Cherenkova K., Aouada D.: 3dbooster: 3d body shape and texture recovery. In *Computer Vision – ECCV 2020 Workshops* (Cham, 2020), Bartoli A., Fusiello A., (Eds.), Springer International Publishing, pp. 726–740.

[Ske19] Sketchfab: Sketchfab: Your 3D content on web, mobile, AR, and VR. `https://sketchfab.com/`, 2011–2019.

[SKPSH13] Schüller C., Kavan L., Panozzo D., Sorkine-Hornung O.: Locally injective mappings. *Computer Graphics Forum (proceedings of EUROGRAPHICS/ACM SIGGRAPH Symposium on Geometry Processing) 32*, 5 (2013), 125–135.

[SLMB05] Sheffer A., Lévy B., Mogilnitsky M., Bogomyakov A.: Abf++: fast and robust angle based flattening. *ACM Transactions on Graphics (TOG) 24*, 2 (2005), 311–330.

[SM05] Sander P. V., Mitchell J. L.: Progressive Buffers: View-dependent Geometry and Texture for LOD Rendering. In *Eurographics Symposium on Geometry Processing 2005* (2005), Desbrun M., Pottmann H., (Eds.), The Eurographics Association.

[Smi19] Smithsonian Institution: Smithsonian 3D Program. `https://3d.si.edu/`, 2013–2019.

[SMKF04] Shilane P., Min P., Kazhdan M., Funkhouser T.: The Princeton shape benchmark. In *Proceedings of the Shape Modeling International (SMI 2004)* (Washington, DC, USA, 2004), SMI '04, IEEE Computer Society, pp. 167–178.

[SPR*07] Sheffer A., Praun E., Rose K., et al.: Mesh parameterization methods and their applications. *Foundations and Trends® in Computer Graphics and Vision 2*, 2 (2007), 105–171.

[SPS*17] Schönberger J. L., Price T., Sattler T., Frahm J.-M., Pollefeys M.: A vote-and-verify strategy for fast spatial verification in image retrieval. In *Computer Vision – ACCV 2016* (Cham, 2017), Lai S.-H., Lepetit V., Nishino K., Sato Y., (Eds.), Springer International Publishing, pp. 321–337.

[SPSH*17] Shtengel A., Poranne R., Sorkine-Hornung O., Kovalsky S. Z., Lipman Y.: Geometric optimization via composite majorization. *ACM Trans. Graph. 36*, 4 (jul 2017).

[SS09] Schwarz M., Stamminger M.: Fast gpu-based adaptive tessellation with cuda. *Computer Graphics Forum 28*, 2 (2009), 365–374.

[SS15] Smith J., Schaefer S.: Bijective parameterization with free boundaries. *ACM Trans. Graph. 34*, 4 (jul 2015).

[SSGH01] Sander P. V., Snyder J., Gortler S. J., Hoppe H.: Texture mapping progressive meshes. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques* (2001), SIGGRAPH '01, Association for Computing Machinery, pp. 409–416.

[SSP08] Springborn B., Schröder P., Pinkall U.: Conformal equivalence of triangle meshes. *ACM Trans. Graph. 27*, 3 (aug 2008), 1–11.

[SSS*08] Sinha S. N., Steedly D., Szeliski R., Agrawala M., Pollefeys M.: Interactive 3d architectural modeling from unordered photo collections. In *ACM SIGGRAPH Asia 2008 Papers* (New York, NY, USA, 2008), SIGGRAPH Asia '08, Association for Computing Machinery.

[SSW*21] Sun J., Shen Z., Wang Y., Bao H., Zhou X.: Loftr: Detector-free local feature matching with transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2021), pp. 8922–8931.

[SWB98] Sloan P.-P. J., Weinstein D. M., Brederson J.: Importance driven texture coordinate optimization. *Computer Graphics Forum 17*, 3 (1998), 97–104.

[SWG*03] Sander P. V., Wood Z. J., Gortler S. J., Snyder J., Hoppe H.: Multi-chart geometry images. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing* (2003), SGP '03, Eurographics Association, p. 146–155.

[SZ03] Sivic, Zisserman: Video google: a text retrieval approach to object matching in videos. In *Proceedings Ninth IEEE International Conference on Computer Vision* (2003), pp. 1470–1477 vol.2.

[SZ15] Simonyan K., Zisserman A.: Very deep convolutional networks for large-scale image recognition, 2015.

[SZFP16] Schönberger J. L., Zheng E., Frahm J.-M., Pollefeys M.: Pixelwise view selection for unstructured multi-view stereo. In *Computer Vision – ECCV 2016* (Cham, 2016), Leibe B., Matas J., Sebe N., Welling M., (Eds.), Springer International Publishing, pp. 501–518.

[Tar16] Tarini M.: Volume-encoded uv-maps. *ACM Trans. Graph. 35*, 4 (2016).

[TCS03] Tarini M., Cignoni P., Scopigno R.: Visibility based methods and assessment for detail-recovery. In *Proceedings of IEEE Visualization* (2003), IEEE Computer Society, pp. 457–464.

[Thr22] Three D Scans: Three D Scans - Free 3D scan archive, 2022.

[TM03] Turk G., Mullins B.: Large geometric models archive, 2003.

[TM*08] Tuytelaars T., Mikolajczyk K., et al.: Local invariant feature detectors: a survey. *Foundations and trends® in computer graphics and vision 3*, 3 (2008), 177–280.

[TMHF00] Triggs B., McLauchlan P. F., Hartley R. I., Fitzgibbon A. W.: Bundle adjustment — a modern synthesis. In *Vision Algorithms: Theory and Practice* (Berlin, Heidelberg, 2000), Triggs B., Zisserman A., Szeliski R., (Eds.), Springer Berlin Heidelberg, pp. 298–372.

[TSPD16] Thonat T., Shechtman E., Paris S., Drettakis G.: Multi-view inpainting for image-based scene editing and rendering. In *2016 Fourth International Conference on 3D Vision (3DV)* (2016), pp. 351–359.

[Tur19] TurboSquid: TurboSquid: 3D models for professionals. `https://www.turbosquid.com/`, 2019.

[Tut60] Tutte W. T.: Convex representations of graphs. *Proceedings of the London Mathematical Society 3*, 1 (1960), 304–320.

[Tut63] Tutte W. T.: How to draw a graph. *Proceedings of the London Mathematical Society 3*, 1 (1963), 743–767.

[Uni19] Unity Technologies: Unity asset store. `https://assetstore.unity.com/`, 2019.

[VNMC10] Vieira A. W., Neto A. A., Macharet D. G., Campos M. F. M.: Mesh denoising using quadric error metric. In *2010 23rd SIBGRAPI Conference on Graphics, Patterns and Images* (2010), pp. 247–254.

[VSJ07] Velho L., Sossai Jr. J.:. *The Visual Computer 23*, 9 (2007), 621 – 629.

[WACS11] Wu C., Agarwal S., Curless B., Seitz S. M.: Multicore bundle adjustment. In *CVPR 2011* (2011), pp. 3057–3064.

[WBSS04] Wang Z., Bovik A., Sheikh H., Simoncelli E.: Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing 13*, 4 (2004), 600–612.

[Wel91] Welzl E.: Smallest enclosing disks (balls and ellipsoids). In *New Results and New Trends in Computer Science* (Berlin, Heidelberg, 1991), Maurer H., (Ed.), Springer Berlin Heidelberg, pp. 359–370.

[WL00] Wei L.-Y., Levoy M.: Fast texture synthesis using tree-structured vector quantization. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques, New Orleans, Louisiana* (USA, 2000), SIGGRAPH '00, ACM Press/Addison-Wesley Publishing Co., p. 479–488.

[WMG14] Waechter M., Moehrle N., Goesele M.: Let there be color! large-scale texturing of 3d reconstructions. In *Computer Vision – ECCV 2014* (Cham, 2014), Fleet D., Pajdla T., Schiele B., Tuytelaars T., (Eds.), Springer International Publishing, pp. 836–850.

[WSI07] Wexler Y., Shechtman E., Irani M.: Space-time completion of video. *IEEE Transactions on Pattern Analysis and Machine Intelligence 29*, 3 (2007), 463–476.

[WSK*15] Wu Z., Song S., Khosla A., Yu F., Zhang L., Tang X., Xiao J.: 3D ShapeNets: A deep representation for volumetric shapes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2015), pp. 1912–1920.

[Wu13] Wu C.: Towards linear-time incremental structure from motion. In *2013 International Conference on 3D Vision - 3DV 2013* (2013), pp. 127–134.

[XXC12] Xie J., Xu L., Chen E.: Image denoising and inpainting with deep neural networks. In *Advances in neural information processing systems* (2012), pp. 341–349.

[YKH10] Yuksel C., Keyser J., House D. H.: Mesh colors. *ACM Trans. Graph. 29*, 2 (apr 2010).

[YLL*09] Yan D.-M., Lévy B., Liu Y., Sun F., Wang W.: Isotropic remeshing with fast and exact computation of restricted voronoi diagram. *Computer Graphics Forum 28*, 5 (2009), 1445–1454.

[YLL*18] Yao Y., Luo Z., Li S., Fang T., Quan L.: Mvsnet: Depth inference for unstructured multi-view stereo. In *Proceedings of the European conference on computer vision (ECCV)* (2018), pp. 767–783.

[YLL*19] Yao Y., Luo Z., Li S., Shen T., Fang T., Quan L.: Recurrent mvsnet for high-resolution multi-view stereo depth inference. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2019), pp. 5525–5534.

[YLP*20] Yang Y., Liu S., Pan H., Liu Y., Tong X.: Pfcnn: Convolutional neural networks on 3d surfaces using parallel frames. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Virtual* (June 2020).

[YLT19] Yuksel C., Lefebvre S., Tarini M.: Rethinking texture mapping. *Computer Graphics Forum 38*, 2 (2019), 535–551.

[YLY*18] Yu J., Lin Z., Yang J., Shen X., Lu X., Huang T. S.: Generative image inpainting with contextual attention, 2018.

[YLY*19] Yu J., Lin Z., Yang J., Shen X., Lu X., Huang T.: Free-form image inpainting with gated convolution, 2019.

[YSGM04] Yoon S.-E., Salomon B., Gayle R., Manocha D.: Quick-vdr: interactive view-dependent rendering of massive models. In *IEEE Visualization 2004* (2004), pp. 131–138.

[Yuk17] Yuksel C.: Mesh color textures. In *Proceedings of High Performance Graphics* (2017), HPG '17.

[ZBK18] Zhu Y., Bridson R., Kaufman D. M.: Blended cured quasi-newton for distortion optimization. *ACM Trans. Graph. 37*, 4 (jul 2018).

[ZDJF14] Zheng E., Dunn E., Jojic V., Frahm J.-M.: Patchmatch based joint view selection and depthmap estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2014).

[ZJ16] Zhou Q., Jacobson A.: Thingi10k: A dataset of 10,000 3D-printing models, 2016.

[ZK14] Zhou Q.-Y., Koltun V.: Color map optimization for 3d reconstruction with consumer depth cameras. *ACM Trans. Graph. 33*, 4 (jul 2014).

[ZLG*18] Zhao H., Li X., Ge H., Lei N., Zhang M., Wang X., Gu X.: Conformal mesh parameterization using discrete calabi flow. *Computer Aided Geometric Design 63* (2018), 96–108.

[ZLK*18] Zhou B., Lapedriza A., Khosla A., Oliva A., Torralba A.: Places: A 10 million image database for scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence 40*, 6 (2018), 1452–1464.

[ZLY*20] Zeng Y., Lin Z., Yang J., Zhang J., Shechtman E., Lu H.: High-resolution image inpainting with iterative confidence feedback and guided upsampling, 2020.

[ZMT05] Zhang E., Mischaikow K., Turk G.: Feature-based surface parameterization and texture mapping. *ACM Trans. Graph. 24*, 1 (jan 2005), 1–27.

[ZR19] Zirr T., Ritschel T.: Distortion-free displacement mapping. *Computer Graphics Forum 38*, 8 (2019), 53–62.