Extracting Well-behaved Quad Models, Materials, and Lighting from Images

Maximilian Frühauf

Master Thesis
September 2023

Advisors
Dr. Hayko Riemenschneider
Dr. Christopher Schroers

Supervisor
Prof. Dr. Markus Gross

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

cgl
computer graphics laboratory

Abstract

We present a novel approach to extract quad-dominant surfaces from multi-view images, producing artist-quality meshes decomposed into an easily editable mesh, high-frequency details, and material properties. Unlike other works, our method produces intrinsically regular and quad-dominant geometry which can be arbitrarily subdivided with Catmull-Clark subdivision without creating artifacts. By using displacement mapping, we constrain high-frequency details to the coarse quad-dominant mesh and provide meshes that can be directly used in traditional graphics pipelines and game engines without modification. We introduce a differentiable formulation of re-meshing a triangular mesh to a quad surface by optimizing orientation and position fields and extracting a quad-dominant surface. Additionally, we leverage recent advances in differentiable rendering to extract a decomposed material and lighting representation. We obtain high-quality meshes by end-to-end optimizing the surface extraction, shading, and lighting process. Our method achieves state-of-the-art view interpolation and geometric accuracy while providing much higher-quality mesh topology. The extracted meshes by our method are well-suited to downstream processing and can render at interactive rates in rasterizers and path tracers.

Zusammenfassung

Wir stellen einen neuartigen Ansatz zur Extraktion von quaddominanten Oberflächen aus Bildern mit mehreren Ansichten vor, der künstlerisch hochwertige Meshes erzeugt, die in ein leicht bearbeitbares Mesh, hochfrequente Details und Materialeigenschaften zerlegt werden. Im Gegensatz zu anderen Arbeiten erzeugt unsere Methode eine inhärent regelmäßige und quad-dominante Geometrie, die mit Catmull-Clark-Unterteilung beliebig unterteilt werden kann, ohne Artefakte zu erzeugen. Durch die Verwendung von Displacement Mapping beschränken wir hochfrequente Details auf das grobe quaddominante Mesh und produzieren Meshes, die direkt in traditionellen Grafikpipelines und Game Engines ohne Modifikation verwendet werden können. Wir stellen eine differenzierbare Formulierung für das re-meshing eines Dreicksmesh zu einer Quad-Oberfläche vor, indem wir die Orientierungs- und Positionsfelder optimieren und eine Quad-dominante Oberfläche extrahieren. Darüber hinaus nutzen wir die jüngsten Fortschritte im differenzierbaren Rendering, um eine dekomponierte Material- und Beleuchtungsdarstellung zu extrahieren. Wir erhalten qualitativ hochwertige Meshes, indem wir die Oberflächenextraktion, die Schattierung und die Beleuchtung durchgängig optimieren. Unsere Methode erreicht vergleichbare Messwerte in den Punkten Ansichtsinterpolation und geometrische Genauigkeit und bietet gleichzeitig eine viel hochwertigere Meshtopologie im Vergleich zu anderen Methoden. Die mit unserer Methode extrahierten Meshes eignen sich gut für die nachfolgende Verarbeitung und können mit interaktiven Raten in Rasterisierern und Pfadverfolgern gerendert werden.

Acknowledgments

Working on this Thesis has been incredibly enjoyable, challenging, and rewarding at the same time. I want to thank my supervisors, Chris and Hayko, for their invaluable feedback and inspiring conversions. They were constantly available, motivated me to keep pushing in new directions, and generally made the process fun. Thanks as well to Markus for making it possible to do a thesis with Disney|Research while complying with ETH requirements. I also would like to thank the entire team there for the small and large tips and tricks they shared with me. Looking at you Morteza and Guilherme ;)

Many thanks to Eva and my friends for listening to me rambling on about the importance of quadrangular faces and always having an open ear during the ups and downs of the project. They were instrumental in keeping my spirits high and not taking myself too seriously. Also, to my parents for continually supporting me.

During the project, I made extensive use of Open Source software such as *PyTorch*, *Blender*, *Instant Meshes*, and many more, and I am thankful to their contributors for making these works publicly available. Finally, aany thanks to the blendswap.com artists for creating the amazing models in the NeRF Synthetic dataset, the NeRD authors and British Museum for providing the NeRD-real-world dataset, and the TanksAndTemple authors for providing the Family and Ignatius scenes to validate our results on.

Contents

List of Figures

List of Tables

$\boldsymbol{k}_{\mathrm{d}}$	Albedo
$\boldsymbol{k}_{\mathrm{s}}$	Specular color
$\boldsymbol{k}_{\mathrm{orm}}$	(Occlusion, Roughness, Metallic)
ω_o	Outgoing ray direction
ω_i	Incoming ray direction
$\mathcal{V}$	Triangle mesh vertices
$\mathcal{E}$	Triangle mesh edges
$\mathbf{n}$	Normal vector
$\mathbf{o}$	Representative orientation direction
$\mathcal{R}_4(\mathbf{o}, \mathbf{n})$	Set of 4 3D orientations around $\mathbf{n}$
$\mathcal{O}$	Orientation field
$\mathbf{k}$	Orientation field rotations
s_q	Position field / Quad-dominant edge length
$\mathbf{p}$	Position field offset
$\mathbf{t}_{ij}$	Integer offset pulling the edge $(i, j) \in \mathcal{E}$ together
$\mathcal{T}_4(\mathbf{p}, \mathbf{n}, \mathbf{o})$	Lattice of integer translations length of s_q from $\mathbf{p}$ tangent to $\mathbf{n}$ in direction $\mathbf{o}$
$\mathcal{P}$	Position field
$\mathcal{N}(i)$	1-ring neighbors of vertex i
G	Marching Cubes grid size
D	Domain size of Marching Cubes

Table 1.: *Notation used throughout this work.*

Introduction

Creating 3D content for games and movies is time-consuming and requires technical and artistic skills. Automatic 3D content creation has been at the forefront of industry and academic research, as it promises to drastically reduce production costs and allows artists to spend more time on creative tasks. Photogrammetry [SSS06; PV02] is a classical solution to this problem, where multiple real-world images of an object are converted into a 3D model. However, this is typically a multi-step process consisting of multi-view stereo [Sch+16; SF16], feature extraction & matching, geometric extraction, texture parametrization, material baking, and delighting. Each step has conflicting optimization goals and introduces errors propagating from one step to the next. Artists using these methods rely on many post-processing tools to work with these objects and integrate them into current production pipelines, resulting in significant overhead.

In this work, we frame the process as an inverse rendering problem and propose an end-to-end optimization from the captured images directly to the extracted geometry, guided by the reconstruction quality of the captured images. Recent methods in differentiable rendering have shown remarkable results in learning high-resolution object representations from images [Mil+20; Wan+21; Li+23], but typically entangle geometry, material, and lighting in a single neural network, making their output hard to process further or render in current movie and game production pipelines. On top of this, extracting a surface requires an iso-surfacing algorithm such as marching cubes [Nie03], which creates unfavorable geometric artifacts at low extraction resolution. Disentangling these surface parameters has recently been proposed on triangle-mesh-based surface representations [HHM22; Mun+22], making the extracted assets usable outside their optimization processes. However, these methods still produce meshes with artifacts such as sliver triangles, making them potentially unusable for downstream tasks.

Re-meshing proposes to extract better meshes by optimizing for a regular quadrilateral topology given a triangle mesh [BZK09; Jak+15; Hua+18] but introduces errors due to approximating it at a lower-resolution based only on local surface heuristics. Importantly, these quadrilateral

meshes are suitable for subdividing using Catmull-Clark subdivision surfaces [CC78], texturing, mesh-editing, and physics-based simulation. However, current re-meshing methods are not differentiable, meaning any error they introduce cannot be mitigated by changing the input surface accordingly. In contrast to Nvdiffrec [Mun+22], we extract high-quality quadrilateral meshes from images in an end-to-end differentiable manner, which are directly suitable for Catmull-Clark subdivision and compatible with current production pipelines.

We present a novel method to directly extract quad-dominant meshes with decomposed materials and lighting from multi-view images by incorporating ideas from re-meshing [Jak+15] with recent work in inverse rendering [Mun+22] for triangular meshes. We assume the object is lit under a single unknown environment with corresponding camera poses and masks separating the object and background for each image as in related work [Bos+21b; Bos+21a]. Our approach extracts quadrilateral meshes by iteratively optimizing an orientation- and position field and the underlying signed distance function (SDF), learning quad-dominant topology and vertex positions. At our method's core is a novel differentiable formulation of re-meshing a triangular mesh extracted from a SDF to a quad-dominant one by jointly learning orientation and position fields while reconstructing the surface. Further, we apply a differentiable formulation of Catmull-Clark subdivision and displacement to recover small-scale details below the size of individual quad faces. Inspired by Nvdiffrec [Mun+22], we also extract spatially-varying materials and environment lighting, resulting in high-quality meshes directly usable without modification in current production pipelines. Building on a highly optimized rasterizer with deferred shading [Lai+20], we compare our rendered reconstruction with the captured images and optimize the surface, position- and orientation field, material, and lighting parameters.

To summarize, our main contributions include:

- An end-to-end differentiable re-meshing formulation of a triangle input mesh to a quadrilateral mesh consisting of majorly quad faces acting as a drop-in extension of current mesh-extraction techniques. By self-learning the orientation- and position field used for re-meshing, we derive a simplified optimization rule while still providing high-quality meshes.
- A differentiable formulation of the Catmull-Clark subdivision process, taking surface deformations introduced by this process into account during the optimization.
- Representation of high-frequency details smaller than the size of individual faces with displacement mapping, implicitly constraining small-scale features to the displacement map. In contrast, larger ones are baked into the quadrilateral mesh. This provides a low-resolution mesh that artists can easily edit using current tools while keeping high-resolution details in the displaced surface.

Background & Related Work

Multi-View 3D reconstruction is classically done by finding correspondences between images to estimate depth maps and fuse these observations in a voxel grid. These methods are limited, however, by the quality of the initial matching and cannot rectify potential errors later. Therefore, they heavily rely on post-processing techniques to produce good results.

2.1. Material Models

Accurate estimation of material properties, the Spatially-Varying Bidirectional Reflectance Distribution Function (SVBRDF), usually relies on complicated lighting setups or special viewpoints to capture material properties.

Recently, neural implicit representations have been used to estimate material properties from a collection of images alone [Bos+21a; Zha+21b]. However, these methods either learn a material model from data or parametrize material parameters with neural networks, making them unusable in current production pipelines or game engines, which store textures as 2D images and use a fixed BRDF to represent a range of materials.

A different approach is proposed by Munkberg et al. [Mun+22], who use a two-stage system capable of extracting textures compatible with the commonly used metallic & roughness material model. Inspired by their approach, we build on their system throughout this work and describe their material model in the following paragraph.

The BRDF $f_r(x, \omega_o, \omega_i)$ is based on the physically-based (PBR) material model by Disney [Bur12], commonly used in production pipelines, making the extracted models directly compatible. The Material model is a mixture of a diffuse and GGX specular lobe [Wal+07]:

$$
\begin{aligned}
f_r(x, \omega_o, \omega_i) &= (1 - m)\boldsymbol{k}_{\mathrm{d}} f_d(\omega_o, \omega_i) + \boldsymbol{k}_{\mathrm{s}} f_s(\omega_i, \omega_o, r) \\
\boldsymbol{k}_{\mathrm{s}} &= \frac{1 - m}{25} + m\boldsymbol{k}_{\mathrm{d}}
\end{aligned}
\tag{2.1}
$$

$\boldsymbol{k}_{\mathrm{d}}, \boldsymbol{k}_{\mathrm{s}} \in [0, 1]^4$ are the diffuse and specular colors with an optional fourth alpha channel. These colors are weighted by the metallic factor $m \in [0, 1]$ blending between a fully plastic and metallic lobe, and material a roughness value $r \in [0, 1]$ controlling the spread of the GGX normal distribution function [Bur12]. Since all of these parameters are spatially varying (SVBRDF), they are stored in 2D texture maps $\boldsymbol{k}_{\mathrm{d}}, \boldsymbol{k}_{\mathrm{orm}}$, where $\boldsymbol{k}_{\mathrm{orm}}$ are the combined (o, r, m) channels and o is used to model occlusion discussed in Section 2.2.

Importantly, Nvdiffrec [Mun+22] uses a tangent space normal map to represent high-frequency shading details, which we do not use as our method focuses on extracting the same high-frequency information as geometric detail through displacement.

Learnable Texturing Since the object's surface is a learnable parameter during optimization, a parametrization of the material parameters in Equation 2.1 is needed, which can be learned stably. The key idea used throughout the literature is to use volumetric texturing to parametrize the material parameters by world-space position [Zha+21a; Zha+21b; Mun+22]. Specifically, using a hash-grid positional encoding [Mül+22] of the position x and a Multi-Layer-Perceptron (MLP), the material parameters are predicted as follows:

$$
\mathrm{MLP}(\text{hash-grid}(x)) \rightarrow (\boldsymbol{k}_{\mathrm{d}}, \boldsymbol{k}_{\mathrm{orm}}) \tag{2.2}
$$

However, since production renderers require 2D texture maps and a corresponding UV surface parametrization, Nvdiffrec performs a re-parametrization of the materials once the surface has converged. They freeze the current topology and use Xatlas [You23] to automatically compute a 2D parametrization on the entire mesh and sample the MLP-based material model on the mesh surface to initialize these 2D textures. In the second optimization stage, they continue optimizing the surface reconstruction but with all material parameters represented by 2D texture maps. This process eliminates texture seams introduced by discontinuities in the 2D texture parametrization generated by automatic UV unwrapping [Mun+22].

2.2. Lighting Models

Light transport in a scene is described by the rendering equation, commonly used in computer graphics [Kaj86]:

$$
L_o(x, \omega_o) = \int_{H^2} f_r(x, \omega_o, \omega_i) L_i(x, \omega_i) \cos\theta_i d\omega_i \tag{2.3}
$$

Here, the outgoing radiance from a point x in direction ω_o, $L_o(x, \omega_o)$ is the product of all incoming radiance weighted by the BRDF $f_s(x, \omega_o, \omega_i)$ and the $\cos\theta_i$ foreshortening term. Crucially, without strongly limiting assumptions, no closed-form solution exists to this integral. Additionally, its recursive nature, as incoming radiance $L_i(x, \omega_i)$ is the outgoing radiance from the

intersection in direction ω_i, makes this integral generally computationally infeasible to approximate directly.

Therefore, multiple sampling ideas have been proposed to efficiently compute $L_o(x, \omega_i)$, some of which we will review in the following paragraphs. Importantly to the task of inverse rendering, Equation 2.3 needs to be evaluated efficiently, as we require many sample points in the scene to compute meaningful gradients w.r.t. the object's surface location while approximating the integral with little variance at practical sample counts.

Spherical Gaussians & Spherical Harmonics A common approximation to Equation 2.3 is to represent both the BRDF and incoming light with Spherical Harmonics (SH) [Che+19] or Spherical Gaussians (SG) [Zha+21a; Bos+21a]. Both of these approximate the integral in Equation 2.3 with a finite sum of either Spherical Gaussians or Spherical Harmonics. Since the product of both can be integrated in closed form, the rendering equation can also be approximated by a closed-form solution, removing the need for costly sampling and providing stable gradients.

However, both formulations only consider direct lighting, effectively cutting off each ray in Equation 2.3 after its first intersection, and cannot model shadows. Crucially, since both perform a frequency decomposition of the BRDF and incoming lighting, many individual frequencies must be summed together to model high-frequency lighting. Since computational cost scales linearly with the number of SG or SH coefficients, these methods are usually limited to low- to medium-frequency lighting.

Pre-integrated Lighting (Split-Sum Approximation) Another approach to approximate Equation 2.3 commonly used in real-time rendering is the *split-sum* approximation [Kar13]:

$$\begin{aligned} L_o(x, \omega_o) &= \int_{H^2} f_r(x, \omega_o, \omega_i) L_i(x, \omega_i) \cos\theta_i d\omega_i = \int_{H^2} f_r(x, \omega_o, \omega_i) L_e(x, \omega_i) \cos\theta_i d\omega_i \\ &= (1-m)\boldsymbol{k}_\mathrm{d} \underbrace{\int_{H^2} f_d(x, \omega_o, \omega_i) L_i(x, \omega_i) \cos\theta_i d\omega_i}_{=:L_d(x,\omega_o)} + \boldsymbol{k}_\mathrm{s} \underbrace{\int_{H^2} f_s(x, \omega_o, \omega_i) L_i(x, \omega_i) \cos\theta_i d\omega_i}_{=:L_s(x,\omega_o)} \\ &= (1-m)\boldsymbol{k}_\mathrm{d} L_d(x, \omega_o) + \boldsymbol{k}_\mathrm{s} L_s(x, \omega_o) \end{aligned} \tag{2.4}$$

By considering only direct lighting ($L_i(x, \omega_o) = L_e(x, \omega_o)$) from an environment map, the integral can be broken up into the outgoing radiance due to diffuse $L_d(x, \omega_o)$ and specular $L_s(x, \omega_o)$ interactions.

By dropping the dependence on position, i.e., $L_e(x, \omega_i) = L_e(\omega_i)$, the diffuse term can be simplified further and approximated as a finite sum. However, this removes the ability to represent shadows with this approximation, as the visibility of the environment from a point x is no longer considered:

$$\begin{aligned} L_d(x, \omega_o) &= \frac{1}{\pi} \int_{H^2} L_i(x, \omega_i) \cos\theta_i d\omega_i \\ &\approx \frac{1}{\pi} \int_{H^2} L_i(\omega_i) \cos\theta_i d\omega_i \end{aligned} \tag{2.5}$$

Where we assume $f_d(x, \omega_i, \omega_o) = 1/\pi$ is the diffuse lambertian BRDF. Importantly, this term can now be pre-computed for a given environment map and stored for fast lookups. Details in Subsection A.1.1.

At the core of the split-sum approach is the following approximation of the specular radiance $L_s(x, \omega_o)$:

$$L_s(x, \omega_o) \approx \underbrace{\int_{H^2} f_s(\omega_o, \omega_i; r) \cos\theta_i d\omega_i}_{=:S_1(\omega_o, r)} \underbrace{\int_{H^2} L_e(\omega_i) D(\omega_h; r) \cos\theta_i d\omega_i}_{=:S_2(\omega_o, r)} \tag{2.6}$$

Here, the specular contribution is approximated by two integrals: The first $S_1(\omega_o, r)$ is the integral of the BRDF with a solid white environment light, and $S_2(\omega_o, r)$ is the integral of the environment map with the GGX normal distribution function $D(\omega_h; r)$. Since $S_1(\omega_o, r)$ only depends on the incoming direction ω_o and roughness r, it can be pre-computed and stored offline. The second term, $S_2(\omega_o, r)$, additionally depends on the environment map. Hence, it must be re-computed at every optimization step, as the environment lighting is an optimizable parameter. Details on the pre-computation of the individual terms are provided in Subsection A.1.1.

Importantly, no ray tracing needs to be performed, and Munkberg et al. [Mun+22] show that gradients are effectively computable. At each iteration, only two texture lookups need to be performed to compute $L_s(x, \omega_o)$: First, of S_1 in terms of $(\cos\theta_i, r)$ and the second of the filtered environment cube map in direction ω_o at roughness r represented by a MIP-pyramid. Diffuse lighting is computed by an additional lookup in direction ω_o on a low-resolution filtered version of the environment cube map.

Since the environment map is an optimizable parameter, the diffuse and specular filtered cube maps are re-computed at each iteration. Additionally, since the same BRDF as in Equation 2.1 is used during pre-filtering, the learned materials and environment lighting are compatible with current production systems using the same basic material model, regardless of the integration method used for approximating Equation 2.3.

A significant limitation of this approach is its inability to model shadows, causing them to be represented by the albedo color $\boldsymbol{k}_\mathrm{d}$. As a band-aid for this problem, Munkberg et al. use the o-channel of the specular lobe map $\boldsymbol{k}_\mathrm{orm}$ as an occlusion weight to directly modulate the outgoing radiance, resembling ambient occlusion [Mun+22].

$$L_o(x, \omega_o) = (1 - o)\left[(1 - m)\boldsymbol{k}_\mathrm{d} L_e(\omega_o) + \boldsymbol{k}_\mathrm{s} L_s(x, \omega_o)\right] \tag{2.7}$$

2.3. Surface Representations

Classical multi-view reconstruction methods follow either a depth-based or voxel-based approach. For depth-based methods, a point cloud is reconstructed by identifying correspondences between images, limited by the difficulty of finding accurate point matches [Sch+16]. Voxel-based methods side-step these limitations, as they optimize over the parameters of a voxel grid by enforcing photometric multi-view consistency [BDC01]. However, reconstruction is limited to low resolutions since memory use scales cubically with the reconstruction resolution.

Neural Implicit Representations have recently gained in popularity for novel view synthesis [Mil+20; Mül+22] and reconstruction [Wan+21] tasks. NeRF [Mil+20] learns a volumetric density field similar to the voxel-based approach described above, but fails to extract high-quality surfaces due to lacking surface constraints. NeuS [Wan+21; Wan+22] instead learns a Signed Distance Field (SDF), enabling high-quality surface reconstructions but cannot be used directly in traditional rendering pipelines, which require an explicit mesh as input. Naïvely, extracting an explicit mesh from these representations introduces additional errors and is not differentiable, resulting in either excessive geometry to accurately represent the scene or limited accuracy, as the extracted surface cannot be optimized further to reduce these extraction errors.

Explicit Surface Representations propose to reconstruct a 3D mesh from a collection of images directly. Most approaches optimize a reconstruction loss on a fixed mesh topology [Che+19; NJJ21]. Recent work explores differentiable meshing of an implicit field [She+21; She+23; LDG18; Che+22], allowing for changing topology during optimization. Even though extracting an explicit surface from an SDF is generally not globally continuous, in practice, local discontinuities do not obstruct the optimization when using modern momentum-based optimizers such as Adam [KB17].

The most similar pipeline for extracting meshes from images is by Munkberg et al. [Mun+22; HHM22]. They directly extract a surface from a per-vertex SDF using DMTet [She+21], resulting in a triangle mesh with many skinny, sliver triangles. Follow-up work by Shen et al. [She+23] addresses this problem of skinny triangles and outputs feature aligned topology.

However, both extract triangular meshes unsuitable for downstream processing, as their resolution is fixed to the one used during reconstruction. These do not consist of equilateral faces, making them unsuitable for further subdivision or changing the extracted level of detail after optimization.

2.4. Quad-dominant Re-meshing

Quad-remeshing has been explored extensively in the context of one-shot re-meshing of a given triangle mesh [Jak+15; Hua+18] into a quad-dominant mesh. The approach by Jakob et al. [Jak+15] first computes an orientation field assigning a local coordinate axis for each input vertex. Subsequently, each vertex is assigned local coordinates in a position field. The orientation field determines the directions of edges in the resulting quadrilateral mesh, and the position field the placement of the re-meshed vertices.

Quad faces on a surface mesh are ideally regularly sized and aligned in strips of a common direction on the surface. These two interrelated quantities, the direction of a quad face expressed as an orientation field, and its connectivity with neighboring faces and vertex positions with a position field, are particularly interesting. However, since isometric quad faces are symmetric under rotations of $90°$, the orientation field needs to have the same inherent symmetry, achieved by considering a "cross" of orientations spaced $90°$ apart from each other, instead of a single direction.

Specifically, these fields are expressed by a four-way rotationally symmetric (4-RoSy) orientation field and similarly a four-way symmetric (4-PoSy) position field. Intuitively, the 4-RoSy field defines a cross of 4 symmetric directions $90°$ to each other in the tangent place on the input surface, representing the direction to which an output quad face should align. Similarly, the 4-PoSy field defines a fixed-size lattice of positions in the tangent space of each vertex aligned to the orientation field.

This change in perspective from quad meshes to orientation and position fields allows us to break up the problem of extracting a quad mesh from an input triangulation into computing two such fields [Bom+13]. However, since re-meshing a triangular mesh with isometric quad faces is not always possible, singularities in the orientation and position fields can occur, representing faces of degree three or more than four. Since these singularities occur only at a minority of faces, the resulting mesh contains predominantly quad faces. Therefore, this class of meshes is named quad-dominant.

2.4.1. Orientation field

Orientations are modeled by a 4-RoSy field in the tangent plane of a normal vector $\mathbf{n} \in \mathbb{R}^3$. Specifically, we consider 4 uniformly spaced directions $\mathcal{R}_4(\mathbf{o}, \mathbf{n}, k)$ in this tangent plane around the *representative direction* $\mathbf{o} \in \mathbb{R}^3$ indexed by $k \in \mathbb{Z}$: .

$$\mathcal{R}_4(\mathbf{o}, \mathbf{n}, k) := \text{rot}\left(\mathbf{n}, k\frac{2\pi}{4}\right)\mathbf{o}, \quad k \in \mathbb{Z} \tag{2.8}$$

$$\mathcal{R}_4(\mathbf{o}, \mathbf{n}) := \{\mathcal{R}_4(\mathbf{o}, \mathbf{n}, 0), \mathcal{R}_4(\mathbf{o}, \mathbf{n}, 1), \ldots, \mathcal{R}_4(\mathbf{o}, \mathbf{n}, 3)\} \tag{2.9}$$

Where $\text{rot}\,(\mathbf{n}, \alpha) \in \mathbb{R}^{3\times3}$ denotes a rotation matrix about $\mathbf{n}$ by angle $\alpha \in \mathbb{R}$ and $\mathcal{R}_4(\mathbf{o}, \mathbf{n})$ is the set of symmetric rotations about $\mathbf{o}$ in the tangent plane of the normal $\mathbf{n}$. Collectively, we assign such a 4-RoSy field to each vertex $i \in \mathcal{V}$ in the triangular input mesh, creating the orientation field $\mathcal{O}$ for each representative direction $\mathbf{o}_i$ and normal $\mathbf{n}_i$:

$$\mathcal{O} := \{R_4(\mathbf{o}_i, \mathbf{n}_i) \mid i \in \mathcal{V}\} \tag{2.10}$$

An exemplary field is shown in Figure 2.1. For such a 4-RoSy field to represent a consistent direction over all edges $(i, j) \in \mathcal{E}$ of the triangular input mesh, it should vary smoothly over the edges. Intuitively, this is measured as the angle difference between neighboring vertices in the input mesh [Ray+08; BZK09; Jak+15]:

$$E(\mathcal{O}, \mathbf{k}) := \sum_{(i,j)\in\mathcal{E}} \angle(\mathcal{R}_4(\mathbf{o}_i\mathbf{n}_i, k_{ij})\mathcal{R}_4(\mathbf{o}_j\mathbf{n}_j, k_{ji}))^2 \tag{2.11}$$

Where the integer variables $k_{ij} \in \{0, 1, 2, 3\}$ are introduced for each edge $(i, j) \in \mathcal{E}$, tracking the 4-RoSy element minimizing the angle between neighboring vertices. For convenience, we stack these variables together in an integer vector $\mathbf{k} \in \{0, 1, 2, 3\}^{|\mathcal{E}|}$.

Th energy $E(\mathcal{O}, \mathbf{k})$ can then be *globally minimized* by mixed-integer optimization [BZK09]. Alternatively, [Jak+15] propose to adapt the Gauss-Seidel method for *locally minimizing* the same energy as follows:

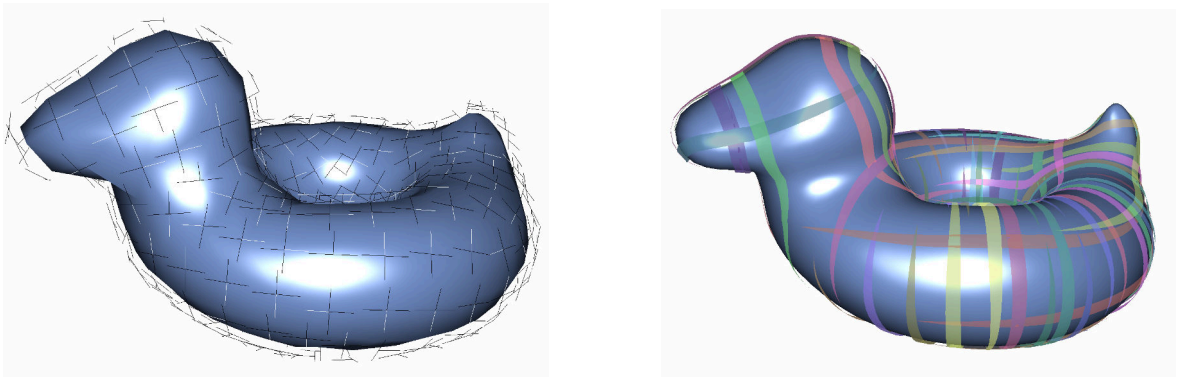

Figure 2.1.: *Smoothed orientation field. The left image shows the symmetry axes of each vertex with the representative direction in white. The right one shows a smooth visualization of the orientation field. Images are generated with Instant Meshes [Jak+15].*

Algorithm 1 4-RoSy field smoothing algorithm

for $i \in \mathcal{V}$ **do**
  $\mathbf{o}'_i = 0$
  for $(i, j) \in \mathcal{E}$ **do**
    $k_{ij}, k_{ji} \leftarrow \arg\min_{0 \leq k,l < 4} \angle \left(\mathcal{R}_4(\mathbf{o}_i, \mathbf{n}_i, k), \mathcal{R}_4(\mathbf{o}_j, \mathbf{n}_j, l)\right)^2$
    $\mathbf{o}'_i \leftarrow \mathbf{o}'_i + w_{ij} \mathcal{R}_4(\mathbf{o}_j, \mathbf{n}_j, k_{ij})$
    $\mathbf{o}'_i \leftarrow \mathbf{o}'_i / \|\mathbf{o}'_i\|_2$ ▷ Re-normalization
    $\mathbf{o}_i \leftarrow \mathbf{o}' - \langle \mathbf{n}_i, \mathbf{o}'_i \rangle \, \mathbf{o}'_i$ ▷ Project to $\mathbf{v}_i$'s tangent plane
  end for
end for

Where an additional weight $w_{ij} \in \mathbb{R}$ is associated with each edge (i, j) and chosen uniformly ($w_{ij} = 1$). Compared to mixed-integer optimization, the main advantage of the above formulation is that it performs only local updates by considering each vertex $i \in \mathcal{V}$ independently. This significantly reduces the search space for the integer rotations k_{ij}, as they are computed independently for each edge (i, j) instead of over all combinations for all edges. Specifically, only $4^2 = 16$ combinations of rotations for vertex i and j need to be considered for each edge. With this extrinsic energy, the orientation field aligns well with sharp features in the triangular input mesh. Also, since the minimization in Algorithm 1 can be performed in parallel for each vertex, this formulation is well suited to parallelization on the GPU.

2.4.2. Position field

To extract quad faces from the input mesh, Jakob et al. [Jak+15] aligns the position field to an optimized orientation field in the sense of Equation 2.11, which defining a smooth direction over the surface of the input mesh. By aligning a local 2D grid with edge length $s_q \in \mathbb{R}$ in the tangent space of each vertex with the 4-RoSy direction, a local 2D parametrization is formed, which already exhibits a quad structure.

Jakob et al. use this insight to formulate a purely local energy that smooths each vertex's local parametrization until a consistent quad structure is globally formed over the entire surface. By optimizing a similar energy to the orientation field described in Subsection 2.4.1, a purely local optimization scheme can be used to compute a globally consistent quad mesh from local symmetry fields on each vertex.

Given a fixed quad edge length s_q, the four-way position symmetry field (4-PoSy) of vertex $i \in \mathcal{V}$ is defined as a 2D lattice in its tangent plane with vertices at multiples of s_q. To align the gradient of the local parametrization with the given 4-RoSy orientation field, the orientation $\mathbf{o}$ is used as the representative direction of the lattice. Specifically, the lattice of integer translations in directions $\mathcal{R}_4(\mathbf{o}, \mathbf{n}, k)$ for $k \in 0 \dots 3$ around a vertex with normal $\mathbf{n}$ and orientation $\mathbf{o}$ is defined as follows:

$$\mathcal{T}_4(\mathbf{p}, \mathbf{n}, \mathbf{o}, \mathbf{t}) := \mathbf{p} + s_q \sum_{k=0}^{1} \mathbf{t}_k \mathcal{R}_4(\mathbf{o}, \mathbf{n}, k) \tag{2.12}$$

$$\mathcal{T}_4(\mathbf{p}, \mathbf{n}, \mathbf{o}) := \left\{ \mathcal{T}_4(\mathbf{p}, \mathbf{n}, \mathbf{o}, \mathbf{t}) \mid \mathbf{t} \in \mathbb{Z}^2 \right\} \tag{2.13}$$

Intuitively, the lattice $\mathcal{T}_4(\mathbf{p}, \mathbf{n}, \mathbf{o})$ is the set of symmetric vertices in four directions with origin at $\mathbf{p} \in \mathbb{R}^3$. Due to this notion of symmetric positions at length s_q intervals in each direction, the only remaining degree-of-freedom is the *fractional* translation in the 'horizontal" and "vertical" directions in $[0, s_p]$. Specifically, the symmetries in $\mathbf{p}$ are modelled by the integer translation $\mathbf{t} \in \mathbb{Z}^2$. So for every two integer translations $\mathbf{t}, \mathbf{t}' : \mathcal{T}_4(\mathbf{p}, \mathbf{n}, \mathbf{o}, \mathbf{t}) \equiv \mathcal{T}_4(\mathbf{p}, \mathbf{n}, \mathbf{o}, \mathbf{t}')$.

Considering the whole input mesh, one symmetry group $\mathcal{T}_4(\mathbf{p}, \mathbf{n}, \mathbf{o}, \mathbf{t})$ is created per vertex, collectively defining the orientation field $\mathcal{P}$ as follows:

$$\mathcal{P} := \{ \mathcal{T}_4(\mathbf{p}_i, \mathbf{n}_i, \mathbf{o}_i) \mid i \in \mathcal{V} \} \tag{2.14}$$

An exemplary position field is shown in Figure 2.2. Ideally, two vertices with an edge in $\mathcal{E}$ have coinciding lattices, meaning they are exactly a multiple of s_q away from each other in either lattice direction. Practically, this is rarely the case, but rather, they often align in a single vertex. On the position field $\mathcal{P}$, a smoothness energy similar to the one on the orientation field defined in Equation 2.11 measures as the position difference between neighboring lattices in the *embedding space*, so directly on the lattice positions as follows:

$$E(\mathcal{P}, \mathbf{t}) := \sum_{(i,j) \in \mathcal{E}} \| \mathcal{T}_4(\mathbf{p}_i, \mathbf{n}_i, \mathbf{o}_i, \mathbf{t}_{ij}) - \mathcal{T}_4(\mathbf{p}_i, \mathbf{n}_i, \mathbf{o}_i, \mathbf{t}_{ji}) \|_2^2 \tag{2.15}$$

Where $\mathbf{t}_{ij}, \mathbf{t}_{ji} \in \mathbb{Z}^2$ are integer translations for each edge (i, j) which are computed by a brute force search over both lattices. Since this energy is very similar to the one defined on the 4-RoSy orientation field, it can be optimized with the following similar strategy, again resembling Gauss-Seidel iterations:

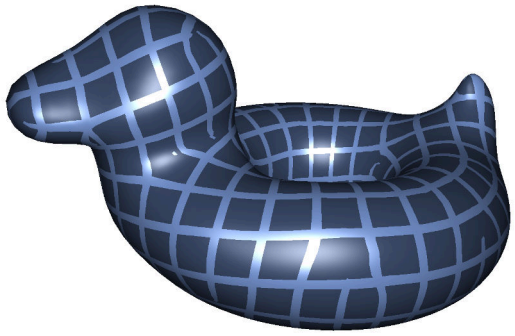

Figure 2.2.: *Smoothed position field. Field lines where $\mathbf{t}_{ij}$ are unit vectors are highlighted in light blue. The image is generated with Instant Meshes [Jak+15].*

Algorithm 2 4-PoSy field smoothing algorithm

for $i \in \mathcal{V}$ **do**
  $\mathbf{p}'_i = 0$
  for $(i, j) \in \mathcal{E}$ **do**
    $\mathbf{t}_{ij}, \mathbf{t}_{ji} \leftarrow \arg\min_{\mathbf{t},\mathbf{t}' \in Q_1,Q_2} \|\mathcal{T}_4(\mathbf{p}_i, \mathbf{n}_i, \mathbf{o}_i, \mathbf{t}) - \mathcal{T}_4(\mathbf{p}_j, \mathbf{n}_j, \mathbf{o}_j, \mathbf{t}')\|_2^2$
    $\mathbf{p}'_i \leftarrow \mathbf{p}'_i + w_{ij}\mathcal{T}_4(\mathbf{p}_j, \mathbf{n}_i, \mathbf{o}_i, \mathbf{t}_{ij})$
    $\mathbf{p}'_i \leftarrow \mathbf{p}'_i / \sum_{n=1}^{j} w_{ij_n}$ ▷ Laplacian update
    $\mathbf{p}_i \leftarrow \mathbf{p}' - \langle \mathbf{n}_i, \mathbf{p}'_i - \mathbf{v}_i \rangle \mathbf{n}_i$ ▷ Project to $\mathbf{v}_i$'s tangent plane
  end for
  $\mathbf{p}_i \leftarrow \text{round}(\mathbf{n}_i, \mathbf{o}_i, \mathbf{p}_i, \mathbf{v}_i)$ ▷ Rounding of $\mathbf{p}_i$ to the nearest lattice position
end for

Where the last rounding operation performs an integer translation computing the closest position $\mathbf{p}_i$ to the vertex position $\mathbf{v}_i$.

An important detail in the above optimization scheme is that the integer translations $\mathbf{t}_{ij}$ are re-computed with brute-force search for each edge (i, j), which might appear infeasible since the search is over two infinite s_p-dimensional lattices. Generally, such a search is equivalent to the Shortest Vector Problem (SVP) for which no efficient solution exists [Mic98]. However, Jakob et al. use that the desired fractional position $\mathbf{p}_i$ should be close to the vertex positions $\mathbf{v}_i, \mathbf{v}_j$, which reduces the search space tremendously. As for any edge (i, j) only the lattice vertices around the intersection point $\mathbf{q}_{ij}$ of the tangent planes of $\mathbf{v}_i, \mathbf{v}_j$ closest to either vertex position need to be searched over which is easy to compute and denoted as O_{ij}, Q_{ji}. The procedure is given in Subsection A.2.2.

2.4.3. Quad extraction

This smoothing of first the orientation- then position field results in a position field from which a quad-dominant mesh can be extracted where most edges have length s_q.

After position field optimization, most integer variables $\mathbf{t}_{ij}$ reference the same vertex location,

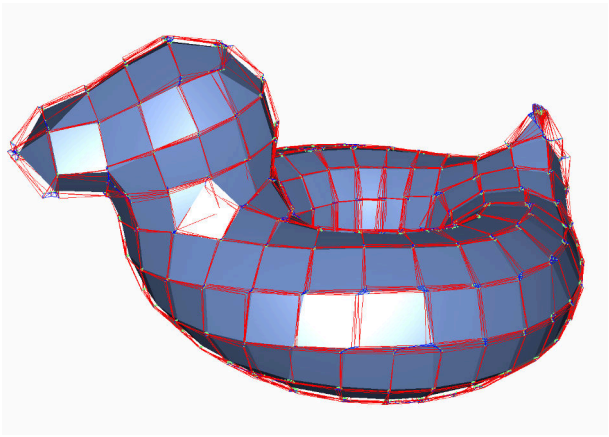

Figure 2.3.: *Edges in the quad-dominant mesh are extracted from vertices where $\mathbf{t}_{ij}$ is a unit vector (shown in red). Other vertices (blue) are collapsed to a single point, forming the quad-dominant vertices. The image is generated with Instant Meshes [Jak+15].*

with either $\mathbf{t}_{ij} = 0$ or $\mathbf{t}_{ij} = -\mathbf{t}_{ji}$. Crucially though, for all vertices where $\mathbf{t}_{ij} = [0, 1]^T$ or $\mathbf{t}_{ij} = [1, 0]^T$, their vertex positions differ by a single integer translation, representing an edge in the output mesh. So by collapsing all vertices where $\mathbf{t}_{ij} = -\mathbf{t}_{ji}$, the resulting graph only contains edges for which $\mathbf{t}_{ij}$ is a unit vector or equivalently a quad edge in the output mesh. An example is shown in Figure 2.3.

Each vertex $\mathbf{v}_i$ in the merged graph now corresponds to some cluster of representative positions $\{\mathbf{p}_{i_1}, \mathbf{p}_{i_2} \ldots, \mathbf{p}_{i_n}\}$ for which a single vertex position is computed as follows:

$$\mathbf{v}_i' \leftarrow \frac{1}{\sum_k^n \alpha_{i_k}} \sum_{k=1}^{n} \alpha_{i_k} \mathbf{p}_{i_k} \quad \alpha_i = \exp(-\beta^2 \|\mathbf{p}_i - \mathbf{v}_i\|_2^2) \tag{2.16}$$

So the quad-dominant vertex positions are just the weighted sum of the representative positions $\mathbf{p}_{i_1}, \ldots, \mathbf{p}_{i_n}$. The weight α_i with $\beta = s_q/3$ strongly weights vertices in the input mesh close to their representative positions.

2.4.4. Singularities

Jakob et al. [Jak+15] note that on the optimized orientation- and position fields singularities occur whenever the composition of all transformations over any simple cycle in $\mathcal{E}$ around a vertex $\mathbf{v}$, is not the identity. Specifically, since the input mesh is triangular, singularities can be detected by summing up k_{ij} or $\mathbf{t}_{ij}$ respectively over the edges of each triangle. If the result is not a multiple of four, a singularity in either the orientation-, respectively, the position field is present.

Orientation Singularities result in an irregular vertex in the final meshing of degree either more or less than four.

Position Singularities result in the insertion or removal of an isoline from the orientation field, resulting in the insertion of a T-junction, which can be meshed as a triangular or pentagonal face. Due to these singularities, the extracted meshes are only quad-dominant and not purely quadrangular. Conceptually, they split or merge two parallel edge loops (sequences of consecutive edges without a turn) while preserving their alignment.

2.5. Displacement

Unlike shading tricks, such as Bump- or Normal Mapping, Displacement mapping represents small-scale details using actual geometry [Coo84; CCC87]. During rendering, these geometric details can realistically interact with scene lighting, cast shadows and change the silhouette of the displaced object. Work by Wang et al. [YRS22] extends this notion to displacement on Signed Distance Functions represented by MLPs. They show that a key requirement to learning a scale decomposition of surface details using displacement is a tight link between the coarse and fine surface representation. While their formulation is not directly applicable to our work, since they optimize continuous SDFs instead of a discrete mesh representation, we interpret their results as motivation for our tightly coupled displacement decomposition based on subdivision surfaces.

End-to-end learning of Quad-Dominant Meshes

We present a method to directly extract an object 3D reconstruction as a quad-dominant mesh from multi-view images with background segmentation masks with known camera poses illuminated under a single unknown lighting condition. By integrating ideas from re-meshing with differentiable rendering, our quad-dominant mesh representation is suitable for Catmull-Clark subdivision and displacement, extracting high-frequency details and spatially varying materials (as 2D Textures) and lighting (as a high dynamic range image probe), all editable by an artist using standard tooling. We carefully design our optimization task to include a differentiable re-meshing formulation and subdivision, providing high-quality meshes directly compatible with current production pipelines. Unlike prior work, we directly optimize for an explicit mesh with quadrilateral faces, which are directly suitable for subdivision, resulting in artist-friendly meshes.

Concretely, starting from an initial surface, we adapt Instant Meshes [Jak+15] by formulating a differentiable re-meshing objective. At each iteration, we extract a quad-dominant mesh from a Signed Distance Function (SDF), subdivide it using Catmull-Clark subdivision, and apply fine-scale displacement. Then, we shade and rasterize the mesh following Nvdiffrec [Mun+22] to compute a reconstruction loss in image space. Finally, gradients are back-propagated through the rasterization, shading, subdivision, and quad-extraction processes to update the SDF, re-meshing field, texture, and lighting parameters. An overview of this pipeline is shown in Figure 3.1 with our core contributions highlighted in green. Each step is explained in detail in the following sections: Section 3.1 & Section 3.2 explain the SDF extraction and differentiable re-meshing, Section 4.1 the subdivision process, Section 4.2 displacement, and Section 4.3 the rasterization.

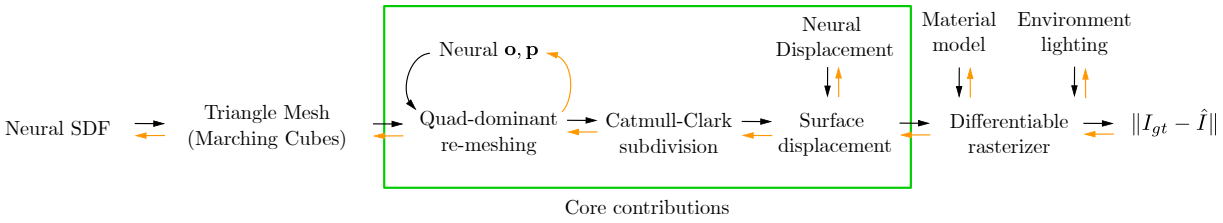

Figure 3.1.: *Overview of our Approach. We learn decomposed topology, materials, and lighting jointly using only posed 2D images as supervision. From a neural SDF, we extract an initial triangle mesh using differentiable Marching Cubes. By learning neural orientation-, and position fields* $\mathbf{o}, \mathbf{p}$*, we re-mesh this triangular mesh into one containing predominantly quad faces, suitable for Catmull-Clark subdivision. On top of this subdivided mesh, we apply displacement predicted by a neural displacement field to recover fine-scale details smaller than individual quad faces. Finally, we apply a differentiable material and lighting model and back-propagate the Image-space loss between the reference and rasterized images. Our output is a low-resolution quad-dominant mesh with spatially varying 2D textures storing material, lighting and surface displacement, which can be directly used in production renderers and game engines.*

3.1. Marching Cubes Mesh Extraction

Signed Distance Functions can represent the surface of an object by assigning the signed distance to the objects surface for each query position. Such an SDF $s(x) : \mathbb{R}^3 \to \mathbb{R}$ is defined to be positive inside the surface and negative outside, with the zero-level set $\mathcal{S} = \{x \in \mathbb{R}^3 \mid s(x) = 0\}$ giving the surface itself.

Recent work has shown that using a light-weight Multi-Layer Perceptron (MLP) to represent $s(x)$ combined with a multi-resolution hash-grid encoding proves to be effective both in terms of representational power and performance characteristics [Mül+22; Li+23]. For each input vertex $x \in \mathbb{R}^3$, the hash-grid encoder $\text{enc}(x) : \mathbb{R}^3 \to \mathbb{R}^{f \cdot d}$ predicts a feature vector $F_i \in \mathbb{R}^d$ at each level of the hierarchical encoding. By using feature grids on multiple scales and concatenating the interpolated feature vectors together, the final feature vector $F \in \mathbb{R}^{f \cdot d}$ is formed. For details, we refer the reader to [Mül+22].

The SDF $s(x)$ is then represented by composing the hash-encoding with a small coordinate MLP to map the feature vector F to the 1-dimensional output:

$$s(x) = \text{MLP}(\text{enc}(x), x) \tag{3.1}$$

Band-limited training of the hash-encoding is used to stabilize training of SDF in Equation 3.1. Each level of the multi-resolution hash-encoding is a grid of $d_i \in \mathbb{N}$ discrete features in each dimension spaced evenly in $[-D/2, D/2]^3$ with $D \in \mathbb{R}_{>0}$, which are either represented as a dense grid of features or by a spatial hash [Mül+22]. However, both can be treated similarly since they discretize the same domain at varying spatial resolutions. By the Nyquist Theorem, any such grid with spatial resolution d can represent functions $f(x)$ of frequency at most $d/2$. Therefore, a hash-encoding with l levels is at most able to represent functions with frequency $d_l/2$ accurately.

This fact has been repeatedly exploited in the literature for training MLP-based SDFs suffering from unstable optimization or local minima [Li+23; YRS22]. Their key insight is that band-limiting the internal representation helps regularize the learned approximation of $f(x)$. Intuitively, by enabling higher frequencies for approximating $f(x)$ at later stages in the optimization, the model is constrained to find the lowest frequency explanation fitting the data.

Specifically, we leverage this property by masking out high-resolution levels of the hash-encoding at the early steps of the optimization and only enabling them later:

$$s(x) = \text{MLP}(m(t)\text{enc}(x), x) \quad \text{with } m(t)_i = \begin{cases} 1 & \text{if } \lfloor i/d \rfloor < m_0 \vee t > t_{\lfloor i/d \rfloor} \\ 0 & \text{else} \end{cases} \tag{3.2}$$

Intuitively, $m(t)_i$ enables all d elements of a layer of the hash-encoding if it's either one of the first initially active $m_0 \in \mathbb{N}$ layers or if its activation time t_i has been reached. Throughout this work, we use a linear increase for the cutoff time t_i for each layer.

$$t_1 = t_0 + \delta t \quad t_2 = t_1 + \delta t \quad t_3 = \dots \tag{3.3}$$

We determined $t_0 = 3000, \delta t = 1500$ to work well experimentally, and use these throughout.

Surface extraction with Marching Cubes Following previous work by Nielson [LC87], we use the popular Marching Cubes algorithm to extract a discrete surface from the SDF. Given a 3D grid of $G \in \mathbb{N}$ vertices of size $D \in \mathbb{R}$ in each direction, we evaluate the SDF on G^3 uniformly spaced vertices in $[-D/2, D/2]^3$. Each cube in this grid has 8 vertices, for which the signs of $s(x)$ determine if the SDF zero-crossing is passing through a given cube. If on any edge between vertices $x_a, x_b \in \mathbb{R}^3$ a sign switch $\text{sign}(s(x_a)) \neq \text{sign}(s(x_b))$ is detected, the zero level-set of $s(x)$ has to intersect the cube's edge (x_a, x_b). Since the SDF $s(x)$ at each of the cube's vertices can be either positive or negative, there are a total of 256 possible cases of how the surface can pass through this cube, which can be further reduced to 15 after removing symmetries. By enumerating these cases and storing them in a lookup table, extraction can be efficiently parallelized [Nie03]. The location of each surface vertex is then determined by interpolating $s(x)$ for each edge of the cube if the SDF's sign changes along this edge. Specifically, the position of each vertex is along an edge with grid vertices $x_a, x_b \in \mathbb{R}^3$ is determined as follows:

$$x_{ab} = \frac{s(x_a)x_a + s(x_b)x_b}{s(x_a) + s(x_b)} \quad \text{if } \text{sign}(s(x_a)) \neq \text{sign}(s(x_b)) \tag{3.4}$$

Even though Marching Cubes triangulates each face of the extracted surface, since they result from the intersection of a cube with the SDF, some of these configurations already exhibit a quad-like structure, which we leverage during quad-dominant re-meshing in Section 3.2.

Differentiating Marching Cubes is required for back-propagating gradients to the underlying SDF when extracting a surface with Marching Cubes, as described in the previous paragraph. Conceptually, gradients on the Marching Cubes surface need to be passed back to the discrete sample points where the SDF was queried to extract the triangle surface in the first place. Back-propagating these gradients requires computing the following derivatives:

$$\begin{aligned}\frac{\partial x_{ab}}{\partial s(x_a)} &= \frac{s(x_a)}{(s(x_b) - s(x_a))^2}(x_b - x_a) \\ \frac{\partial x_{ab}}{\partial s(x_b)} &= \frac{s(x_b)}{(s(x_b) - s(x_a))^2}(x_a - x_b)\end{aligned} \tag{3.5}$$

However, both of these derivatives above are only defined if $s(x_a) \neq s(x_b)$. A similar singularity has been explored by [She+21], who empirically show that the case $s(x_a) = s(x_b)$ never happens in practice and does not negatively impact the optimization. This observation is grounded by the fact that for any edge (x_a, x_b), Marching Cubes will only extract a triangle if $\text{sign}(s(x_a)) \neq \text{sign}(s(x_b))$, implying the above condition. During a backward step, the updated SDF values are updated by some discrete step size, and the updated triangle surface is extracted. So Equation 3.5 is only evaluated on grid vertices with different signs in the first place, making it always well-defined. For all other vertices where $\text{sign}(s(x_a)) = \text{sign}(s(x_b))$ we do not propagate gradients: $\partial x_{ab}/\partial s(x_a) = \partial x_{ab}/\partial s(x_b) = 0$

During back-propagation, the SDF values around the zero-crossing where the surface is located could be arbitrarily small, i.e., $|s(x_b) - s(x_a)| < \epsilon$ for some small $\epsilon > 0$. Marching Cubes would still extract a surface, computing the gradients in Equation 3.5 and causing exploding gradients. We prevent these issues by clipping the denominator of Equation 3.5 to the experimentally determined value of 10^{-10}.

$$\begin{aligned}\frac{\partial x_{ab}}{\partial s(x_a)} &\approx \frac{s(x_a)}{\max\{(s(x_b) - s(x_a))^2, 10^{-10}\}}(x_b - x_a) \\ \frac{\partial x_{ab}}{\partial s(x_b)} &\approx \frac{-s(x_b)}{\max\{(s(x_b) - s(x_a))^2, 10^{-10}\}}(x_b - x_a)\end{aligned} \tag{3.6}$$

3.2. Differentiable Re-meshing

The key contribution of our approach is a differentiable re-meshing step of a triangle input mesh to a quad-dominant one consisting of mostly quad faces with few of different degree. By performing this re-meshing step as part of an end-to-end reconstruction pipeline, our learning algorithm can directly account for any errors introduced in the quad-dominant extraction process.

Given a triangle mesh of the surface extracted from a SDF using Marching Cubes (see Section 3.1), we estimate a suitable orientation direction and position offset on a fixed-size lattice for each vertex in the triangle mesh. Specifically, these position offsets are constrained to a lattice with 4 symmetric directions where each lattice point is at an integer multiple of a fixed offset from the initial vertex position as described by Jakob et al. [Jak+15]. Similarly, we perform iterative smoothing steps of first the orientation, then the position field by explicitly optimizing the Laplacian smoothness energy from [Jak+15]. A quad-dominant mesh is extracted from the position field by clustering all vertices in the input mesh with the same integer position offset and replacing each with its weighted centroid.

Our key observation is that we expect the surface location to change minimally between successive iterations of our surface optimization. Therefore, we expect the optimized orientation and position fields to stay similar between iterations. We exploit this by reusing the resulting smoothed fields from one iteration as initial values for smoothing in the next. Crucially, as the input topology might change during optimization, we learn these initial values with a Multi-Layer Perceptron (MLP) queried by the position of each vertex.

3.2.1. Explicit Optimization

Starting from an initial guess for the orientation field $\mathcal{O}$ and position field $\mathcal{P}$, we first optimize all orientations $\mathbf{o}_i$ for each vertex $i \in \mathcal{V}$ and then each vertex' fractional position offset $\mathbf{p}_i$ aligned to the optimal orientation. As the energy function described in Subsection 2.4.1 enforces similar directions among neighboring vertices, the resulting optimal orientation field $\mathcal{O}$ contains smoothly varying orientation vectors over the entire surface of the input mesh. These flows are analogous to the orientation of a quad face placed on the input surface. Hence, quad faces extracted from this field are expected to align to sharp features and vary smoothly over the surface, resulting in desirable quad strips, i.e. chains of quad faces with the same direction. For more details, see Subsection 2.4.1.

We use this optimized quad field to constrain the adjacency of each extracted quad face to follow these optimal directions by enforcing the first axis of the 2D lattice represented by the position field $\mathcal{P}$ to align to this direction $\mathbf{o}$. In detail, the basis vectors for the position lattice at each vertex i are:

$$[\mathbf{o}_i, \mathbf{o}_i \times \mathbf{n}_i] \tag{3.7}$$

Crucially, different from Jakob et al. [Jak+15], we do not perform hierarchical smoothing of these fields as our self-learning formulation described in Subsection 3.2.2 effectively performs a much larger number of smoothing iterations by reusing optimal results from previously optimized fields. Hence, we do not need to simplify the input mesh to form different hierarchy levels nor transfer optimal fields between hierarchy levels. Additionally, we even observe hierarchical smoothing to be less stable during optimization than our self-learning formulation, which we hypothesize to be due to the weaker coupling between optimal fields across iterations when transferring the optimized fields up and down a multi-resolution hierarchy. As in such a configuration, only the highest (lowest-resolution) hierarchy level couples the optimal fields between iterations. We ablate these findings in Section 5.4.

Orientation field optimization Given an initial orientation field $\mathcal{O}$, the optimization rule proposed by Jakob et al. successively applies six iterations of Algorithm 2, updating all orientations $\mathbf{o}_i$ for all vertices $i \in \mathcal{V}$. Empirically, they observe this choice of six iterations as an optimal tradeoff between runtime and smoothness [Jak+15]. We follow the same procedure and update all orientations at each iteration, smoothing the orientation field successively:

$$\underbrace{\mathbf{o} \leftarrow \text{Alg. } 1(\mathbf{o}, \mathbf{n}) \quad \mathbf{o} \leftarrow \text{Alg. } 1(\mathbf{o}, \mathbf{n}) \quad \dots \quad \mathbf{o}^* \leftarrow \text{Alg. } 1(\mathbf{o}, \mathbf{n})}_{\text{six times}} \tag{3.8}$$

Every Iteration of Algorithm 1 aligns the orientation of all vertices to the mean orientation of its neighbors, similar to a Laplacian update. Naïvely unrolling the optimization proposed by Jakob et al. is feasible since the number of vertices we optimize over is relatively small. However, when back-propagating through this process, special care has to be taken as described in Subsection 3.2.2. The process results in a smoothed orientation field $\mathbf{o}^*$, which we use to align the position field in the next paragraph.

Position field optimization A similar optimization rule to the one used on the orientation field is used for the position field. However, we enforce each vertex's fractional position $\mathbf{p}$ to align to the optimal orientation $\mathbf{o}^*$ optimized in the previous step by using $\mathbf{o}$ as one of its coordinate axes:

$$\underbrace{(\mathbf{p},\mathbf{t}) \leftarrow \text{Alg. } 2(\mathbf{p},\mathbf{o}^*,\mathbf{n}) \quad (\mathbf{p},\mathbf{t}) \leftarrow \text{Alg. } 2(\mathbf{p},\mathbf{o}^*,\mathbf{n}) \quad \dots \quad (\mathbf{p}^*,\mathbf{t}^*) \leftarrow \text{Alg. } 2(\mathbf{p},\mathbf{o}^*,\mathbf{n})}_{\text{six times}} \tag{3.9}$$

Similar to the optimization rule in Equation 3.8, we apply six iterations of Algorithm 2 to extract a sufficiently smooth position field. These smoothed fractional positions $\mathbf{p}^*$ for each vertex are used in Subsection 3.2.3 to extract a quad-dominant by introducing an edge whenever the associated integer variables $\mathbf{t}^*$ associated with edge are equal to one of the two 2D unit vectors.

3.2.2. Self-learning of the fields

During optimization, the surface location changes minimally between successive iterations since we enforce smooth SDF learning as detailed in Section 3.1. This drives our key assumption that the associated fields should also stay similar.

Jakob et al. [Jak+15] show that directly optimizing the orientation and position energies tends to get stuck in local minima and propose use a Multi-resolution approach instead to avoid these. However, this requires successively collapsing the vertices of the input mesh and transferring data up- and down this hierarchy.

We propose a much simpler but equally effective optimization strategy: Instead of re-building a hierarchy at each step, we only perform optimization on the input mesh itself or the highest resolution level used by Jakob et al. [Jak+15]. To avoid getting stuck in local minima during field optimization, we propose learning the initial values of the orientation and position fields using an MLP. So we add a secondary optimization loop learning the optimal orientation and position fields in a self-supervised manner similar to ideas in radiance cashing [Mül+21]. This allows us to learn the optimal fields in world-space coordinates required for transferring the field to meshes under changing topology.

Orientation Field Self-learning At each iteration, the **o**-MLP predicts an initial value for the orientation field at each vertex of the input mesh:

$$\hat{\mathbf{o}}_i \leftarrow \mathbf{o}\text{-MLP}(\mathbf{v}_i) \qquad \hat{\mathbf{o}}_i \leftarrow \hat{\mathbf{o}}_i - \mathbf{n}_i \langle \hat{\mathbf{o}}_i, \mathbf{n}_i \rangle \qquad \hat{\mathbf{o}}_i \leftarrow \hat{\mathbf{o}}_i / \|\hat{\mathbf{o}}_i\| \tag{3.10}$$

Above, the value $\hat{\mathbf{o}}_i$ is predicted for each vertex $\mathbf{v}_i$ and then projected to the tangent plane around the normal $\mathbf{n}_i$ of vertex $\mathbf{v}_i$. Intrinsically, $\hat{\mathbf{o}}_i$ is a 1 Degree-of-Freedom quantity since it is a normalized vector in the tangent plane of $\mathbf{v}_i$, essentially only parametrized by an angle around the normal. However, we experimentally found a full 3D representation of the normal to be easier to learn for the **o**-MLP.

To train this network, one needs to take the $\pi/2$-symmetry of the orientation vectors into account. Since for a given orientation $\mathbf{o}_i$, all rotations of multiples of $\pi/2$ around the vertex normal represent the same orientation, the loss needs to take this into account. So a desirable loss function comparing the prediction by the **o**-MLP, $\hat{\mathbf{o}}_i$ and the resulting smoothed orientation $\mathbf{o}_i$ needs to also be symmetric around rotations of $\pi/2$. One option is the widespread van-Mises distribution, which is rotationally invariant for angle differences of 2π [MJ09].

$$1 - \exp(\kappa(\cos\theta - 1)) \tag{3.11}$$

Where the parameter $\kappa \in \mathbb{R}_{>0}$ controls the sharpness of the distribution and we assume $\kappa = 1$ throughout this work. Work by Dielen et al. [Die+21] uses Equation 3.11 with a doubled wrapping frequency to learn a π-symmetric orientation field. However, since we use a single direction symmetric every $\pi/2$ rotation around the normal to parametrize the orientation field instead of their two independent directions with symmetries at π, we use four times higher wrapping frequency in Equation 3.11. So the resulting loss comparing the predicted orientation $\hat{\mathbf{o}}_i$ and the smoothed result $\mathbf{o}_i$ as follows:

$$\begin{aligned} \mathcal{L}_o(\mathbf{o}_i, \hat{\mathbf{o}}_i) &= 1 - \exp(\cos(4\theta) - 1) \\ &= 1 - \exp(8(\cos^4\theta - \cos^2\theta)) \end{aligned} \tag{3.12}$$

Where θ is the angle between $\mathbf{o}_i$ and $\hat{\mathbf{o}}_i$. See Subsection A.2.3 for a detailed derivation of Equation 3.12.

Position Field Self-learning is similar to the orientation field. We use a similar **p**-MLP to predict an initial position offset for each input vertex:

$$\hat{\mathbf{p}}_i \leftarrow \tanh(\mathbf{p}\text{-MLP}(\mathbf{v}_i)) \in \mathbb{R}^2 \tag{3.13}$$

For every vertex normal $\mathbf{n}_i$, we compute the tangent-space vectors with $\mathbf{s}_i, \mathbf{t}_i \in \mathbb{R}^3$ around each normal (details in Subsection A.2.4) and use these to interpolate position offset from the vertex position $\mathbf{v}_i$ scaled by the re-meshing edge length s_q.

Measuring the deviation of the initially predicted position $\hat{\mathbf{p}}_i$ and the smoothed one $\mathbf{p}_i$ is easiest in local space, since each $\mathbf{p}_i$ is aligned to its orientation direction $\mathbf{o}_i$. By predicting $\hat{\mathbf{p}}_i \in \mathbb{R}^2$ as a 2D dimensional vector in the tangent space of $\mathbf{v}_i$ with direction $\mathbf{o}_i$, we decouple the dependence of the predicted position from the direction $\mathbf{o}_i$. This makes learning the fractional position offset $\hat{\mathbf{p}}_i$ easier.

Since $\mathbf{p}_i \in \mathbb{R}^3$ is a 3D world-space vector, we first project it back to its tangent space forming $\mathbf{p}_i^{2D}$. The final position loss is then just an L2-norm between the predicted position and the smoothed result in local space:

$$\mathcal{L}_p(\hat{\mathbf{p}}_i, \mathbf{p}) = \|\hat{\mathbf{p}} - \mathbf{p}_i^{2D}\|_2^2 \qquad \mathbf{p}_i^{2D} = \begin{bmatrix} \mathbf{o}_i^T \\ (\mathbf{o}_i \times \mathbf{n}_i)^T \end{bmatrix} (\mathbf{p}_i - \mathbf{v}_i)/s_q \tag{3.14}$$

$\mathbf{p}_i^{2D}$ is the local 2D projection of $\mathbf{p}_i$ normalized with the quad edge length s_q.

Combined Loss To learn both of these fields jointly, we directly regress the smoothed orientations $\mathbf{o}_i$ and positions $\mathbf{p}_i$ at every step:

$$\mathcal{L}_{op} = \mathcal{L}_o(\hat{\mathbf{o}}_i, \text{sg}\,[\mathbf{o}_i]) + \mathcal{L}_p(\hat{\mathbf{p}}_i, \text{sg}\,[\mathbf{p}_i]) \tag{3.15}$$

We use the gradient stop operation $\text{sg}\,[\cdot]$ to force the **o**-MLP and **p**-MLP to track the smoothed fields, without introducing gradients back on the vertices. Effectively, we are posing the orientation and position learning task as a self-learning problem similar to [Mül+21].

Differentiation at the optimum Naïvely back-propagating through the unrolled orientation- and position field optimization as defined in Equation 3.8 and Equation 3.9 is unstable, as the fields potentially change drastically during this optimization, especially when the self-learning process detailed in Subsection 3.2.2 has not converged yet.

Therefore, we adapt ideas from Vector Quantization (VQ) and skip gradient back-propagation through the orientation- and position field optimization process [OVK17]. Practically, after the respective field optimization, we apply the following update rule:

$$\mathbf{o}^* \leftarrow \hat{\mathbf{o}} + \text{sg}\,[\mathbf{o}^* - \hat{\mathbf{o}}] \qquad \mathbf{p}^* \leftarrow \hat{\mathbf{p}} + \text{sg}\,[\mathbf{p}^* - \hat{\mathbf{p}}] \tag{3.16}$$

In the above equation, $\text{sg}\,[\cdot]$ is the stop-gradient operator defined as the identity during the forward pass while having zero partial derivatives. In other words, during forward computation, the above equation evaluates to $\mathbf{o}^* \leftarrow \mathbf{o}^*$, $\mathbf{p} \leftarrow \mathbf{p}^*$ so the identity function. During backward computation, however, the stop-gradient term drops out, resulting in the following gradients:

$$\frac{\partial \mathbf{o}^*}{\partial \theta} = \frac{\partial \hat{\mathbf{o}}}{\partial \theta} \qquad \frac{\partial \mathbf{p}^*}{\partial \theta} = \frac{\partial \hat{\mathbf{p}}}{\partial \theta} \tag{3.17}$$

This causes our method to skip propagating gradients through the field optimization process and directly forward gradients to the learnt orientation and position fields.

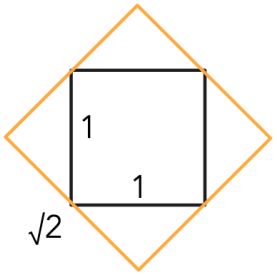

Figure 3.2.: *Example of the minimal re-meshing size of a quadratic grid cell (black). For a quad face (orange) to enclose the grid cell for all orientations, it must be $\sqrt{2}$ times larger.*

3.2.3. Quad-dominant face extraction

After orientation- and position field optimization, we extract a quad-dominant mesh from the integer position field as described by Jakob et al. [Jak+15]. Even though the extracted mesh might contain some non-quad faces at this point, due to singularities in the optimized fields, the number of quads significantly dominates the number of faces with all other degrees. Therefore, we call such a mesh quad-dominant. Details on this extraction process are provided in Subsection 2.4.3.

Lower edge length bound for face extraction In quad extraction, the quad edge length $s_q \in \mathbb{R}$ directly determines the resolution of the resulting quad-dominant mesh. Ideally, s_q should be as small as possible to achieve the highest quad-dominant mesh resolution. This has the side effect of tightly coupling the quad-dominant vertices extracted from the position field to the vertices in the triangular input mesh since each quad-dominant vertex is formed from a cluster of vertices in the triangle mesh. Therefore, each cluster should be as small as possible to back-propagate gradients with maximal spatial resolution from the quad-dominant vertices to the triangle mesh vertices.

Based on the voxel size used when extracting the triangular input mesh defined in Section 3.1, we can derive a lower bound for s_q as follows:

$$s_q = \frac{\sqrt{2}D}{G-1} \tag{3.18}$$

Where D, G are the domain size and grid resolution as defined in Section 3.1. Intuitively, the edge length of a grid of size G with extent D in each dimension is $\frac{D}{G-1}$. During re-meshing, the extracted quad faces need to be able to enclose their respective grid faces for all orientations in order to be able to align to arbitrary surface orientations prescribed by the orientation field. For this to always be the case, the covering quad-dominant face must be larger by $\sqrt{2}$. A visual example is shown in Figure 3.2. Throughout this work, we use the edge length in Equation 3.18.

Differentiable Subdivision, Displacement, and Rendering

4.1. Catmull-Clark Subdivision

The extracted quad-dominant mesh described in Section 3.2 can still contain some non-quad faces at singularities of the position field. Inspired by production renderers and game engines, we model fine-scale details smaller than the re-meshing edge length s_q by subdividing the quad-dominant mesh and displacing this subdivision surface. Using Catmull-Clark subdivision [CC78; DV21] compared to tesselation has two advantages: 1.) Smoothes small irregularities in the surface, making it easier to extract smooth geometries. 2.) Creates a quad-only surface after a single subdivision step by inserting vertex pairs of valence three and five.

Every subdivision step splits each face into four smaller ones and smooths the vertex positions to approximate a second-order smooth surface of the quad-dominant mesh iteratively. Since this refinement can be performed independently for each face, the process is well suited to parallel implementation as proposed by Dupuy et al. [DV21]. They provide a recursive refinement rule for computing a single level of Catmull Clark subdivision in parallel using a formulation based on half edges.

Specifically, our method uses iterative refinement to extract a Subdivision surface $\mathcal{S}_l$ for a user-defined number of levels $l \in \mathbb{N}$ from the quad-dominant mesh $\mathcal{Q}$, in this case the subdivision surface control cage.

$$\mathcal{Q} = \mathcal{S}_0 \rightarrow \mathcal{S}_1 \rightarrow \mathcal{S}_2 \rightarrow \cdots \rightarrow \mathcal{S}_l \tag{4.1}$$

We set $l = 2$ and iteratively increase it throughout the optimization from $l = 0$ initially.

Figure 4.1.: *Comparison of applying 4 subdivision levels on the Control Cage $\mathcal{S}_0$. Our result and the one by Dupuy et al. [DV21] are identical, while our implementation is end-to-end differentiable.*

Differentiable subdivision We implement a differentiable version of the parallel algorithm by Dupuy et al. [DV21] to lean control cage vertex positions through the subdivision process from image observations through differentiable rendering. We subdivide the control cage l times at each iteration to obtain the subdivision surface $\mathcal{S}_l$. To obtain gradients on the control mesh $\mathcal{S}_0$ with respect to $\mathcal{S}_l$, we implement the subdivision procedure using auto-differentiation in PyTorch with gather & scatter operations. Subdivision at each level is computed by Algorithm 3, where details for the individual algorithms are given in the original work by Dupuy et al. [DV21].

Algorithm 3 Catmull-Clark subdivision algorithm

for $d = 0.1, 2, \ldots l$ **do**
  $\mathcal{C}_{d+1} \leftarrow$ REFINECREASES($\mathcal{C}_d$) ▷ Algorithm 5 [DV21]
  $\mathcal{H}_{d+1} \leftarrow$ REFINETOPOLOGY($\mathcal{H}_d$) ▷ Algorithm 1 [DV21]
  $\mathcal{V}_{d+1} \leftarrow$ FACEPOINTS($\mathcal{S}_d, \mathcal{V}_d$) ▷ Algorithm 2 [DV21]
  $\mathcal{V}_{d+1} \leftarrow$ EDGEPOINTS($\mathcal{S}_d, \mathcal{V}_{d+1}$) ▷ Algorithm 3 [DV21]
  $\mathcal{V}_{d+1} \leftarrow$ NONMANIFOLDVERTEXPOINTS($\mathcal{S}_d, \mathcal{V}_{d+1}$) ▷ Algorithm 4
  $\mathcal{T}_{d+1} \leftarrow$ REFINEUVS($\mathcal{S}_d, \mathcal{T}_d$) ▷ Author's source code [DV21]
end for

Initially, we extract the set of half-edges $\mathcal{H}_0$, vertices $\mathcal{V}_0$, vertex creases $\mathcal{C}_0$, and vertex UVs $\mathcal{T}_0$ from the input subdivision cage $\mathcal{S}_0$.

Semi-sharp Vertex Creases $\mathcal{C}_l$ define the sharpness of any half-edge where zero sharpness corresponds to the resulting surface from Catmull-Clark subdivision and any non-zero value blends between the supersampled input surface. To keep the boundary location of the subdivision cage constant for all levels, we initialize all boundary half-edges with a sharpness of 16. A comparison of the subdivision surfaces created by our method compared to Dupuy et al. [DV21] is given in Figure 4.1.

Bow-type meshes The Algorithm proposed by Dupuy et al. [DV21] as presented only supports manifold meshes. However, our quad-dominant face extraction method can produce non-manifold meshes during optimization, which must also be subdivided consistently and correctly to back-propagate gradients to the control cage.

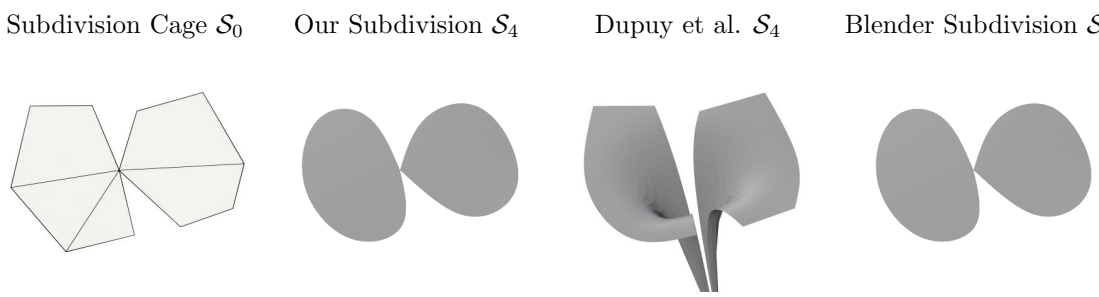

Figure 4.2.: *The method by Dupuy et al. [DV21] is only able to handle manifold meshes. When subdividing a bow-type mesh $\mathcal{S}_0$ (left) their method introduces spurious geometry causing non-consistent subdivision. Considering non-manifoldness, our results at the fourth subdivision level $\mathcal{S}_4$ are identical to Blender's.*

Specifically, our quad-dominant extraction method does not produce general non-manifold surfaces, but a subclass called bow-type meshes [Lie91]. These topologies are characterized by a single shared vertex between two otherwise manifold surfaces, called a fan, with an example given in Figure 4.2. Here, two fans are connected by a single vertex. When subdividing this control cage with the method by Dupuy et al., the vertex location of the non-manifold vertex is updated incorrectly, creating spurious geometry.

This behavior is caused by incorrectly computing the vertex valence, the number of edges of each vertex, as the algorithm proposed by Dupuy et al. only iterates through all edges incident to a vertex within the same fan. For non-manifold vertices, the correct valence needs to be computed as the number of incoming (or outgoing) edges for each vertex. We propose to take this case into account in Algorithm 4 where we detect non-manifold vertices by computing the vertex valence within each fan with the method proposed by Dupuy et al. and by counting the number of incoming edges. If these differ, the vertex must be part of two separate fans and, therefore, non-manifold. In this case, we do not refine the vertex position but copy it from the previous subdivision level, effectively freezing it in-place. Figure 4.2 shows the resulting mesh when freezing non-manifold vertices.

Gradient smoothing under Subdivision When applying subdivision during the entire surface optimization, we found that the surface would get such in local minima, e.g., failing to predict holes in the surface, thereby increasing its genus. As we did not observe this behavior without applying Catmull-Clark subdivision, we perform the initial surface optimization steps without subdivision until the silhouette of the surface has converged. Specifically, we do not apply subdivision during the first 3000 optimization steps, rendering the quad-dominant mesh directly, since we find that surface predictions have usually converged sufficiently well after this threshold. We apply Catmull-Clark subdivision only after this warmup period and observe improved convergence across datasets.

A geometric perspective for this failure to converge during early-stage optimization is that each subdivided vertex is the weighted average of neighboring vertices in the coarser subdivision level (respectively, the subdivision cage). So, gradients on these subdivided vertices contribute to multiple vertices in the coarser level. Therefore, the gradients at any vertex in the coarse level are also an average of the gradients on the fine level. These fine-level gradients point in very

Algorithm 4 Modified VERTEXPOINTS algorithm freezing non-manifold vertices

procedure NONMANIFOLDVERTEXPOINT($\mathcal{S}_d$: Input mesh, $\mathcal{V}_{d+1}$: vertices)
  $\mathcal{V}_{l+1} \leftarrow$ VERTEXPOINTS($\mathcal{S}_l, \mathcal{V}_{l+1}$) ▷ Algorithm 4 [DV21]
  $n_0, n_1, \ldots, n_{|\mathcal{V}_{d+1}|} = 0 \quad e_0, e_1, \ldots, e_{|\mathcal{V}_{d+1}|} = 0$
  for $h \in \mathcal{H}_d$ **do**
    $n \leftarrow$ VALENCE($\mathcal{S}_d, h$) ▷ Algorithm 6 [DV21]
    $v \leftarrow$ VERT(h)
    $n_v \leftarrow n_v + n$ ▷ Scatter half-edge valence to vertices
    $e_v \leftarrow e_v + 1$ ▷ Scatter number of incoming half edges to vertices
  end for
  for $v \in \mathcal{V}_d$ **do**
    if $e_v \neq n_v$ **then**
      $\mathcal{V}_{d+1}[v] \leftarrow \mathcal{V}_d[v]$ ▷ Freeze non-manifold vertices
    end if
  end for
end procedure

different directions if the reconstruction error is large, so the resulting average will no longer be a meaningful gradient direction. Therefore, we find the Catmull-Clark subdivision surface's gradients to be meaningful only if the reconstruction error is not too large.

Since we use subdivision only for recovering fine-scale surface details, we find that this limitation is not problematic in our problem formulation, as the subdivision is only applied later in the optimization once the surface location has converged sufficiently.

4.2. Displacement

Representing high-frequency details smaller than a single face is supported by applying learned displacement mapping to the quad-dominant base geometry. First, we subdivide the quad-dominant base geometry $\mathcal{Q}$ with a fixed number of subdivision steps to both increase mesh resolution and smooth the quad-dominant mesh, resulting in the subdivided surface $\mathcal{S}_l$.

Local object level displacement High-frequency details are modeled by perturbing this surface with a learned displacement field represented as a hash-encoded MLP, identical to the one used for the SDF (details in Section 3.1):

$$\mathbf{v} \leftarrow \mathbf{v} + \mathbf{d}(\mathbf{v}) \qquad \mathbf{d}(\mathbf{v}) = \frac{s_q}{2} \tanh(\text{MLP}(\text{hash-enc}(\mathbf{v}), \mathbf{v})) \tag{4.2}$$

Each vertex is perturbed by a vector in the range $[-s_q/2, s_q/2]^3$, where we choose the maximal displacement magnitude empirically as $s_q/2$. Intuitively, details above this threshold should be represented by changes to the quad-dominant mesh and only small shading details encoded by displacement vectors. This avoids an over-parametrized representation, un-ambiguously decomposing large surface features in the quad-dominant mesh and only representing smaller features by displacement.

Additionally, regularize on displacement magnitude to enforce a smooth surface prior:

$$\mathcal{L}_{\text{disp}} = \frac{1}{|\mathcal{S}_l|} \sum_{\mathbf{v} \in \mathcal{S}_l} \|\mathbf{d}(\mathbf{v})\|_1 \tag{4.3}$$

4.3. Implementation

Material Model and Lighting For differentiable rendering, we adapt Nvidffrec [Mun+22] and replace the iso-surfacing technique DMTet [She+21], which is at the core of their work, with our mesh extraction pipeline. So we apply shading and lighting in the same way as Nvdiffrec, resulting in our method achieving the same decomposition of materials into diffuse and specular components $\boldsymbol{k}_{\text{d}}, \boldsymbol{k}_{\text{orm}}$ respectively. Similarly, we also represent the environment light as a cube map and optimize its parameters as part of the optimization pipeline.

Triangulation As our method builds on Nvdiffrec, which uses a differentiable rasterizer built on OpenGL [Lai+20], only capable of rendering triangle meshes, we triangulate the displaced mesh. The displaced mesh is guaranteed to only contain quad faces, resulting from performing at least one step of Catmull-Clark subdivision on the quad-dominant mesh. Therefore, we triangulate the displaced mesh by splitting each quad face into two triangles. However, since the diagonal to split the quad along is ambiguous, we use the shorter of both diagonals as the split axis. Hence creating two triangles with the least distorted angels. Empirically, we observe stable optimization results using this method but would be interested in exploring a leaned triangulation criteria as in FlexiCubes [She+23] in the future.

Differentiable Rasterization We use the Nvdiffrast [Lai+20] differentiable rasterizer with deferred shading to render our shaded and triangulated models to images, which can then be compared to the posed ground-truth images. The choice to use rasterization for rendering is mainly driven by its simplicity and stable computation of shading gradients since no sampling needs to be performed. Additionally, modern GPUs natively support rasterization, making the process efficient to compute. However, this choice limits us to triangulating our displaced mesh, which is counter-intuitive to our goal of extracting quadrangular faces. In the future, we hope to exchange this rasterizer for one directly supporting quad-meshes, specifically displaced subdivision surfaces.

Second-Stage Optimization Current rendering pipelines store textures as 2D images mapped onto a 3D mesh, requiring a 2D parametrization over the mesh's surface. As we parametrize materials with a world-space MLP during optimization, we follow Nvdiffrec, and re-parametrize the model once surface and textures have converged by generating unique 2D texture coordinates for each face.

Since our parametrization needs to support quad-dominant meshes, we compute an automatic parametrization using the Smart UV project functionality in blender [Com18] instead of Xatlas [You23] as Nvdiffrec. After computing the parametrization, we initialize the mesh's textures

with the converged MLP results and continue optimizing over only the vertex positions, materials and lighting while keeping topology fixed. As shown by Nvdiffrec, this second-stage optimization is necessary to remove (u, v)-seams introduced by the parametrization and may also increase texture resolution, as high-resolution 2D textures are used for $\boldsymbol{k}_{\text{d}}, \boldsymbol{k}_{\text{orm}}$ and displacement $\mathbf{d}$.

4.4. Optimization

Beyond the losses described in this chapter, we use the following regularizers to extract mesh, material and lighting properties consistently.

Image Loss is computed as the coverage and texture loss separately after tone mapping each image, following Nvdiffrec [Mun+22].

$$\begin{aligned}
\Gamma(x) &= \begin{cases} 12.92x & x \leq 0.0031308 \\ (1+a)x^{1/2.4} - a & x > 0.0031308 \end{cases} \\
\mathcal{L}_{\text{mask}} &= \| I^{\text{a}}_{\text{gt}} \left(\Gamma(\log(1 + I^{\text{rgb}}_{\theta})) - I^{\text{rgb}}_{\text{gt}} \right) \|_1 \\
\mathcal{L}_{\text{tex}} &= \frac{1}{|I_{\text{gt}}|} \sum_{x,y} (I^{\text{a}}_{\theta}[x, y] - I^{\text{a}}_{\text{gt}}[x, y])^2
\end{aligned} \tag{4.4}$$

Since colors are represented in linear space in our physically based shading pipeline, a loss function robust to unbounded radiance values is required. Therefore, we tone map the rendered images I_θ using a logarithmic mapping and the sRGB transfer function $\Gamma(x)$ [Sto+96] before comparing it to the ground-truth images I_{gt}. To separate the texture and coverage losses, we denote the color channels of the rendered and ground-truth image respectively, by $I^{\text{rgb}}_{\theta}, I^{\text{rgb}}_{\text{gt}}$ with their associated alpha channels $I^{\text{a}}_{\theta}, I^{\text{a}}_{\text{gt}}$.

Light Regularizer Sine real-world datasets contain mostly neutral white lighting, we follow Nvdiffrec [Mun+22] and NeRFactor [Zha+21b] and regularize the lighting towards white:

$$\mathcal{L}_{\text{light}} = \frac{1}{3} \sum_{i=1}^{3} |\bar{c}_i - c_i| \tag{4.5}$$

Where $c_i, \bar{c}_i$ are the RGB channel intensities and average channel intensity, respectively.

Occlusion Regularizer As the lighting computation used by Nvdiffrec does not support shadows, they modulate the outgoing radiance by an occlusion mask stored in the first channel of the $\boldsymbol{k}_{\text{orm}}$ map. Since this effect is not physically accurate, its regularized to be small.

$$\mathcal{L}_{\text{occ}} = \frac{1}{|\boldsymbol{k}_{\text{orm}}|} \sum_{x,y} \boldsymbol{k}^{\text{o}}_{\text{orm}}[x, y]^2 \tag{4.6}$$

λ_{mask}	1	λ_{occ}	10^{-3}
λ_{tex}	1	λ_{mat}	$3 \cdot 10^{-2}$
λ_{op}	10^{-2}	λ_{δ}	10^{4}
λ_{light}	$5 \cdot 10^{-4}$		

Table 4.1.: *Loss weights used throughout this work unless stated otherwise.*

Material Regularizer Similar to NeRFactor, Nvdiffrec regularizes the predicted Albedo to be as smooth as possible by enforcing the albedo at a surface location $\mathbf{x}_{\text{surf}}$ to be similar at over small perturbations ϵ:

$$\mathcal{L}_{\text{mat}} = \sum_{\mathbf{x}_{\text{surf}}} |\boldsymbol{k}_{\text{d}}(\mathbf{x}_{\text{surf}}) - \boldsymbol{k}_{\text{d}}(\mathbf{x}_{\text{surf}} + \epsilon)| \tag{4.7}$$

SDF regularizer As formulated so far, only visible faces receive a gradient signal, leading to random geometry in the extracted mesh in unsupervised regions inside the mesh or in unobserved regions. This is undesirable in terms of accurately extracting the geometry and from a texturing perspective since all extracted faces occupy pixels in the final 2D texture map. Nvdiffrec penalizes all SDF values as follows, similar to Liao et al. [LDG18]:

$$\mathcal{L}_{\text{inside}} = \sum_{s_i, s_j \in \mathbb{S}_e} H(\sigma(s_i), \text{sign}(s_j)) + H(\sigma(s_j), \text{sign}(s_i)) \tag{4.8}$$

Here, $H(x, y)$ is the binary cross entropy function, $\sigma(x)$ is the sigmoid function, and $\mathbb{S}_e$ the set of all edges where $s_i \neq s_j$. Intuitively, this loss penalizes all sign changes in the SDF while biasing each towards $s_i \approx s_j$, effectively removing the extracted face if $\text{sign}(s_i) = \text{sign}(s_j)$.

Laplacian Regularizer In the second stage, when the topology is fixed, a Laplacian regularizer is used to regularize vertex movements. Given the uniformly weighted vertex differential $\delta_i = \mathbf{v}_i - \frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} \mathbf{v}_j$, where $\mathcal{N}(i)$ are the 1-ring neighbors of vertex $\mathbf{v}_i$, Nvdiffrec use the following regularizer:

$$\mathcal{L}_{\delta} = \frac{1}{n} \sum_{i=1}^{n} \|\delta_i - \delta_i'\|_2^2 \tag{4.9}$$

Where δ_i, δ_i' are the Laplacians of the vertex $\mathbf{v}_i$ in the currently optimized mesh, and the fixed input mesh (output from the first stage), respectively.

Combined Loss The combined loss for the first- and second stages with all respective weighting terms is as follows:

$$\begin{aligned} \mathcal{L}_{\text{shading}} &= \lambda_{\text{mask}}\mathcal{L}_{\text{mask}} + \lambda_{\text{tex}}\mathcal{L}_{\text{tex}} + \lambda_{\text{light}}\mathcal{L}_{\text{light}} + \lambda_{\text{occ}}\mathcal{L}_{\text{occ}} + \lambda_{\text{mat}}\mathcal{L}_{\text{mat}} \\ \mathcal{L}_{\text{first}} &= \mathcal{L}_{\text{shading}} + \lambda_{\text{op}}\mathcal{L}_{\text{op}} \\ \mathcal{L}_{\text{second}} &= \mathcal{L}_{\text{shading}} + \lambda_{\delta}\mathcal{L}_{\delta} \end{aligned} \tag{4.10}$$

Where the loss weights are given in Table 4.1.

Results

This section evaluates our method for artist-friendly extraction of quad-dominant meshes. Firstly, we compare our approach to the state-of-the-art method Nvidffrec [Mun+22] on synthetic data, showing its effectiveness at view interpolation and geometry reconstruction. Secondly, to emphasize the mesh quality of our method, we additionally measure the reconstruction accuracy and mesh quality on a diverse set of meshes and compare them between our method, Marching Cubes [Nie03] and Marching Tetrahedra [She+21]. Lastly, we extract meshes from real-world data, comparing the reconstruction accuracy with Nvdiffrec.

All experiments were run on a single RTX 4090 with 24GB of VRAM.

5.1. View Interpolation

Motivation and Experimental Setting The main application of our method is extracting quad-dominant meshes from posed images. Therefore, we compare the reconstruction quality of our method on synthetic data where ground truth is available.

Specifically, since our method reconstructs meshes with suitable geometry for subdivision, our experimental setting focuses on extracting a low-resolution mesh that can be further processed in off-the-shelf applications while representing high-frequency details using displacement mapping. This has the advantage of providing an artist with a well-behaved and low-resolution geometry representation, which can be easily manipulated while maintaining the representational power of a high-resolution mesh through displacement mapping modeling small-scale details.

Dataset We evaluate our method on the NeRF Synthetic dataset [Mil+20], which is popular for view interpolation since it consists of static synthetic scenes with known camera poses and ground truth geometry, rendered in Blender [Com18]. The dataset consists of eight scenes with 100 training images rendered from camera locations distributed randomly in a hemisphere around each object at 800x800px resolution. For testing, a separate set of 200 images is used (for details, see [Mil+20]).

Baseline As a baseline method, we compare to Nvdiffrec [Mun+22; HHM22], which focuses on extracting decomposed triangular, instead of quad-dominant meshes, from posed images.

Evaluation Metrics We evaluate the visual quality of the surface reconstructions by comparing the Peak Signal to Noise Ratio (PSNR), Structural Similarity Index Measure (SSIM), and Learned Perceptual Image Patch Similarity (LPIPS) using TorchMetrics [Nic+22] on the test images for each scene.

Geometric quality is evaluated by computing minimal and maximal face angles and the number of position and orientation singularities in the learned orientation and position fields. In detail, to compare the reconstruction accuracy and intrinsic quality of our meshes, we compute the following metrics:

- Chamfer Distance (CD): We measure the Chamfer Distance between the predicted and ground truth meshes after scaling each object's bounding-box longest axis to 10 units, following [FSG17]. Then, we sample $2.5 \cdot 10^6$ points on all triangles visible in at least one test view, as both geometry inside each object and invisible regions in the training views cannot be supervised by image-based reconstruction. The final Chamfer distance is computed as in Equation 5.2. However, due to the different scale used in the comparisons with Nvdiffrec in this section, the Chamfer Distance computed here is not directly comparable to the one in Section 5.2.
- Minimal and maximal face angles: By computing the minimal and maximal angle within each face of the reconstructed meshes, we measure the intrinsic quality of the meshes. Ideally, these should be as similar as possible such that very slim, "sliver" faces are avoided since these cause artifacts when subdividing the mesh or performing other processing.

Results Table 5.1 shows per-scene breakdowns of the view-interpolation metrics on the NeRF Synthetic dataset. Qualitative results comparing to our baseline Nvdiffrec are shown in Figure 5.1. More visual examples of our method's extraction quality are in Figure 5.2.

Regarding view-interpolation scores, our method is consistently on par with our baseline Nvdiffrec regarding PSNR, sometimes even improving on its scores. Perceptually-based scores such as (SSIM and LPIPS) in Table 5.1 are even closer between both methods, highlighting that our method can learn a consistent object representation suitable for artist use.

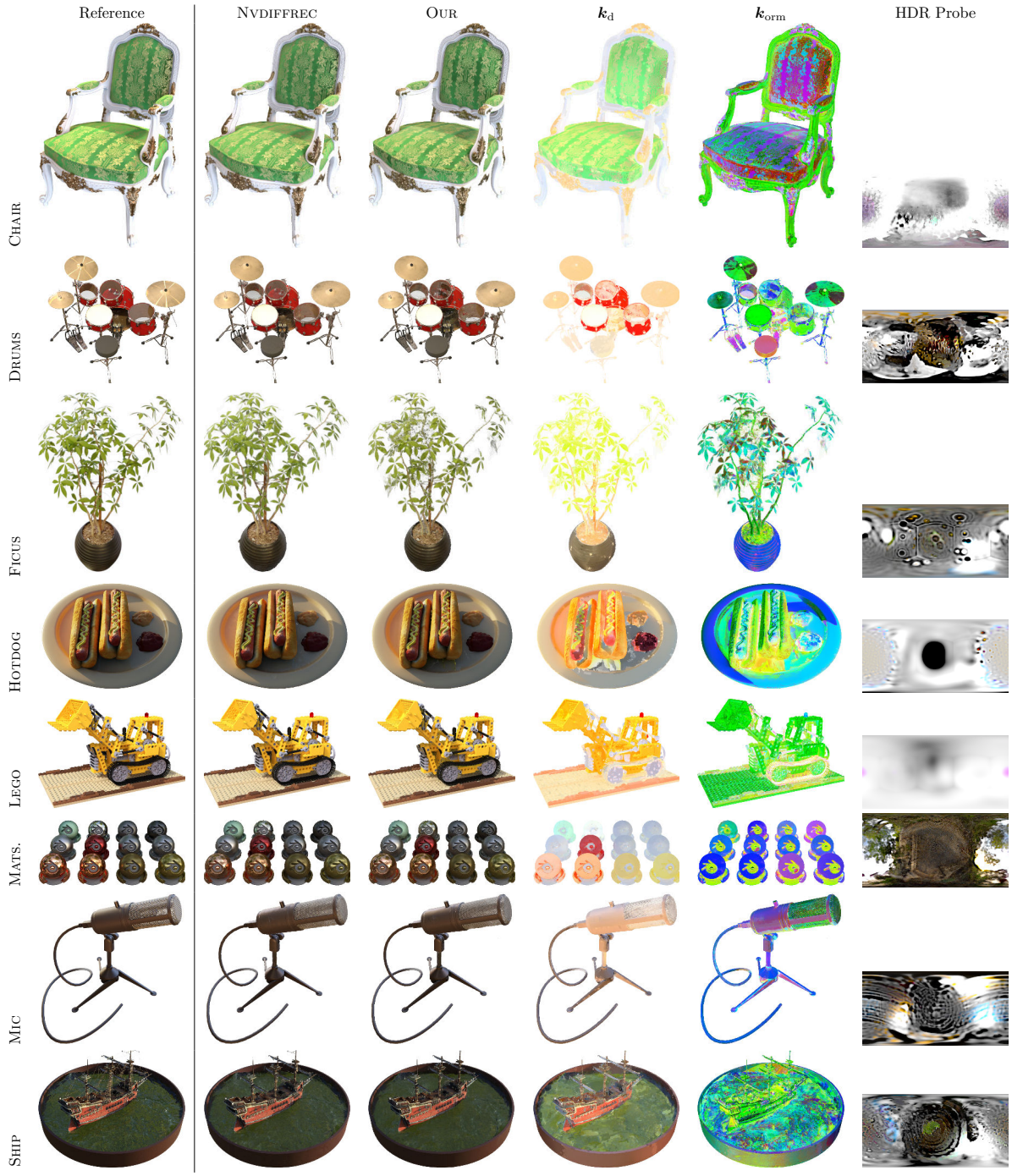

Figure 5.1.: *Qualitative comparison with Nvdiffrec [Mun+22] on the NeRF synthetic dataset. Visually, the results between both methods are very similar, except on the* FICUS *scene, where ours fails to reconstruct the small leaves.*

PSNR (dB) ↑	Chair	Drums	Ficus	Hotdog	Lego	Mats.	Mic	Ship	Avg.
MipNeRF	**35.08**	**25.56**	**33.44**	**37.38**	**35.46**	**30.63**	**36.38**	**30.46**	**33.05**
NeRF	33.00	25.01	30.13	36.18	32.54	29.62	32.91	28.65	31.00
Nvdiffrec	31.60	24.10	30.88	33.04	29.14	26.74	30.78	26.12	29.05
Our	31.42	24.42	24.84	32.97	28.64	26.71	30.22	26.24	28.18

SSIM ↑	Chair	Drums	Ficus	Hotdog	Lego	Mats.	Mic	Ship	Avg.
MipNeRF	**0.980**	**0.934**	**0.981**	**0.982**	**0.978**	**0.959**	**0.991**	**0.885**	**0.961**
NeRF	0.967	0.925	0.964	0.974	0.961	0.949	0.980	0.856	0.947
Nvdiffrec	0.969	0.916	0.970	0.973	0.949	0.923	0.977	0.833	0.939
Our	0.965	0.925	0.937	0.972	0.945	0.937	0.974	0.836	0.936

LPIPS ↓	Chair	Drums	Ficus	Hotdog	Lego	Mats.	Mic	Ship	Avg.
MipNeRF	**0.041**	0.104	0.045	**0.038**	0.053	**0.054**	**0.024**	**0.177**	**0.067**
NeRF	0.046	**0.091**	**0.044**	0.121	**0.050**	0.063	0.028	0.206	0.081
Nvdiffrec	0.045	0.101	0.048	0.060	0.061	0.100	0.040	0.191	0.081
Our	0.045	0.092	0.085	0.058	0.065	0.084	0.043	0.188	0.082

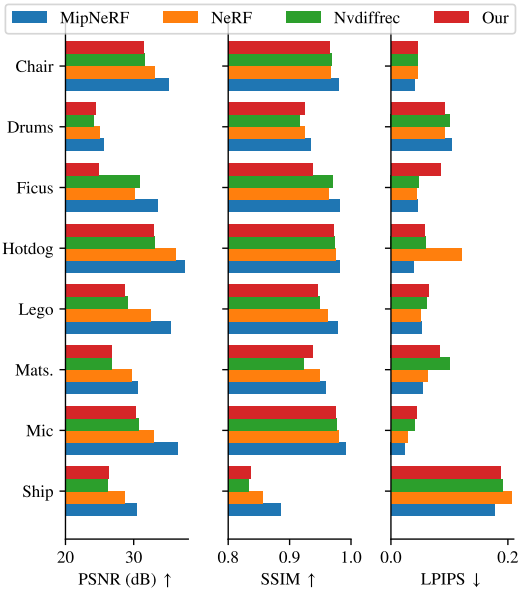

Table 5.1.: *Image quality metrics on all 200 test images in the NeRF Synthetic dataset. Results from NeRF are from Table 4 of the original paper [Mil+20]. Results for MipNeRF are from re-runs of the original paper [Bar+21] performed by [Mun+22]. Nvdiffrec results are taken from the original paper [Mun+22].*

CD ↓	Chair	Drums	Ficus	Hotdog	Lego	Mats.	Mic	Ship
nvdiffrec	0.0567	0.0344	**0.0141**	0.0292	**0.0249**	0.0170	**0.0013**	0.4306
second-qmesh	0.0244	0.0219	0.0485	**0.0239**	0.0275	**0.0154**	0.0051	0.1962
second-disp	**0.0227**	**0.0183**	0.0327	0.0250	0.0300	0.0157	0.0030	**0.1859**

Visible CD ↓	Chair	Drums	Ficus	Hotdog	Lego	Mats.	Mic	Ship
Nvdiffrec	0.0173	0.0358	**0.0067**	**0.0256**	**0.0065**	0.0085	**0.0013**	**0.0600**
Quad dominant	0.0027	0.0111	0.0436	0.0321	0.0163	**0.0054**	0.0055	0.0607
Displaced	**0.0022**	**0.0100**	0.0311	0.0353	0.0174	0.0055	0.0032	0.0601

# Faces ↓ (10^3)	Chair	Drums	Ficus	Hotdog	Lego	Mats.	Mic	Ship
Nvdiffrec	101	67	39	56	114	63	22	187
Quad dominant	**17**	**16**	**14**	**14**	**28**	**14**	**5**	**26**
Displaced	271	259	224	223	448	230	78	407

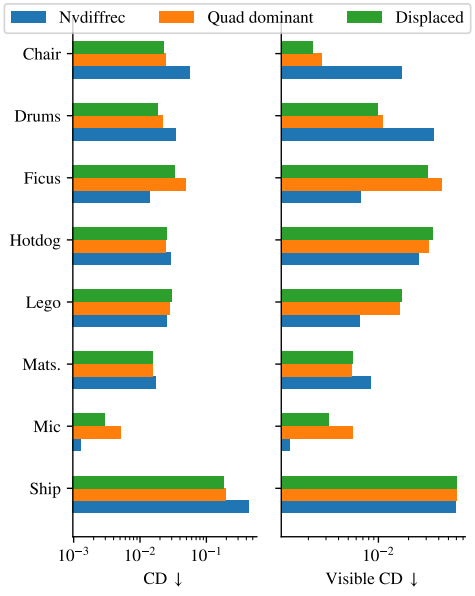

Table 5.2.: *Chamfer scores and face counts on the NeRF synthetic dataset. Results for Nvdiffrec are re-run based on publicly available code. Note that our method can primarily reconstruct opaque geometry larger than the re-meshing edge length s_q, causing our method to under-perform on the* FICUS *scene due to the small-scale leaf geometry present.*

Compared to NeRF-based methods such as MipNeRF [Bar+21] and the original NeRF [Mil+20] we observe a larger difference in scores. Following Munkberg et al. [Mun+22], we speculate that this is due to volume-based methods being able to optimize PSNR more effectively than mesh-based techniques using an opaque surface. Similarly, recent work in differentiable SDF rendering, which extracts a similar opaque surface, also shows these limitations [Wan+22].

To work around this, Nvdiffrec supports rendering transparency during second-stage optimization, where mesh topology is frozen and only vertex position, lighting, and materials are optimized. Allowing for partially transparent materials improves view interpolation quality, especially on scenes with small geometric features, such as FICUS. Also, since the shading model used in Nvdiffrec does not model transparent surfaces explicitly, these are represented by semi-transparent geometry used on the DRUMS, FICUS, MIC, and SHIP scenes. All results in this section show the result after second-stage optimization.

The FICUS scene is a challenging case for our method. Since our method re-meshes an extracted triangle mesh by increasing the edge length of the extracted surface, we cannot recover the small-scale details in the leaf geometry on this scene causing these leaves to be represented by a blob as shown in Figure 5.2. Even when allowing for partially transparent materials during second-stage optimization, view-interpolation results are worse than Nvdiffrec as shown in Table 5.1 and visually in Figure 5.1.

Even though our method does not focus on extracting accurate geometry but rather artist-friendly meshes, we still report Chamfer Distance between the reference and our meshes in Table 5.2. In their original work, Nvdiffrec measure the Chamfer Distance by sampling points from all faces in the predicted and ground truth meshes. However, since significant parts of the scenes in the NeRF synthetic dataset are not observed in the training views, their metric is biased towards predicting accurate geometry in un-observable regions of the objects, resulting in an inaccurate measurement of extracted quality. For consistency with their results, we compute the Chamfer Distance on the entire mesh as Nvdiffrec, but also provide an alternative version only computed on visible faces in Table 5.2.

Generally, Chamfer scores are close between our method and Nvdiffrec, with our method showing consistently better scores in scenes with simple geometry, which is well-represented by our quad-dominant meshes. However, as described previously, we consider the FICUS scene a challenging case for our method due to its small-scale leaves. Unsurprisingly, here Chamfer Distance is much larger in this scene compared to Nvdiffrec. Our method, on the other hand, does not sufficiently recover the individual leaves and predicts a single blob instead as shown in Figure 5.2.

To quantify the intrinsic topological quality of the reconstructed meshes, we compare Minimal and Maximal face angles in Figure 5.3. Generally, these show that our method can extract faces with much more concentrated angles around 90 degrees. The meshes extracted with Nvdiffrec using DMTet have an almost uniform distribution in edge lengths, showing that they contain many sliver triangles. These triangles cause artifacts when subdivided and cannot be used when mesh deformation is required. Our method extracts much more uniform and aligned faces as expanded upon in Section 5.2.

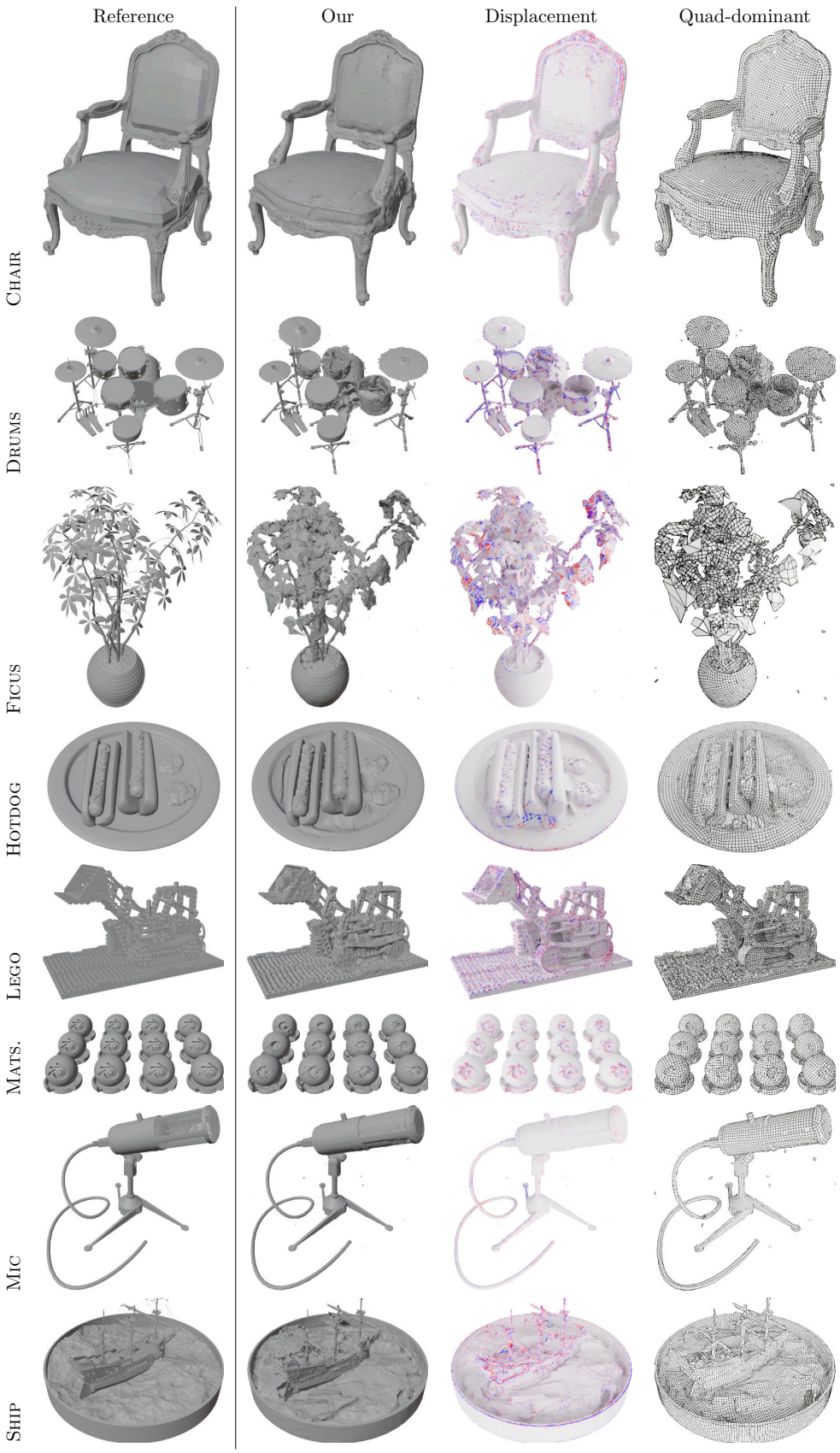

Figure 5.2.: *Mesh extraction results on the NeRF Synthetic dataset. We show our shaded models alongside the displacement magnitude and Quad-dominant meshes. Quad-dominant meshes capture the base geometry, while high-frequency details are represented by displacement on the subdivided quad-dominant mesh.*

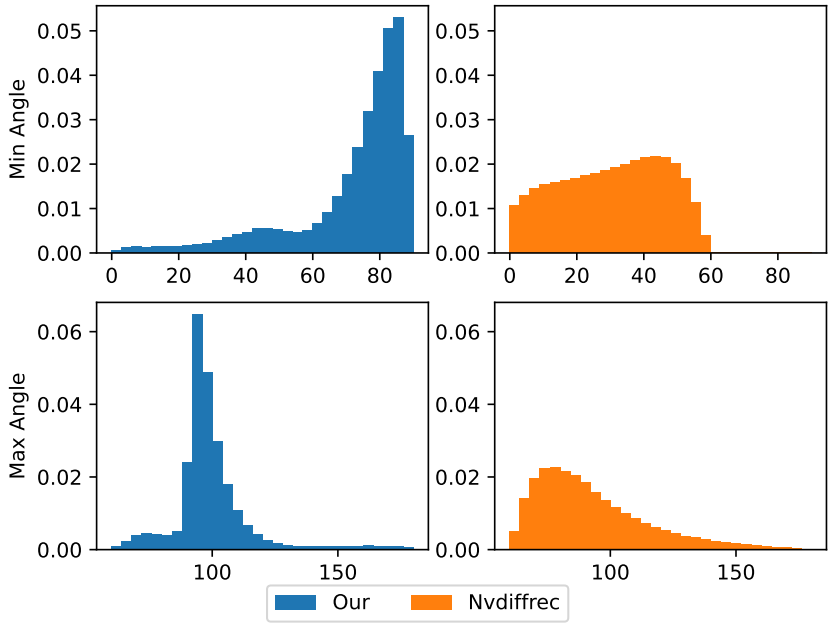

Figure 5.3.: *Quantitative comparison of the minimal and maximal angle within each face on the NeRF synthetic dataset. Our method extracts faces with angles concentrated around 90 degrees, while angles for Nvdiffrec have a much larger range, resulting in sliver triangles.*

5.2. Mesh Reconstruction

Motivation and Experimental Setting Our differentiable re-meshing formulation is at the core of our method for extracting quad-dominant meshes. To compare its ability to represent high-quality meshes, we compare different iso-surfacing methods in an idealized setting by directly comparing the reconstructed geometry to the ground truth objects. Specifically, we supervise each with a depth and mask image of the ground truth object at each iteration from a randomly sampled camera pose. We directly compute the difference between our rendered views and the ground truth on these views, effectively supervising the surface geometry.

$$\begin{aligned} \mathcal{L}_{\text{mask}} &= \|I_{\text{gt}}^{\text{mask}} - \hat{I}^{\text{mask}}\|_1 \\ \mathcal{L}_{\text{depth}} &= \|I_{\text{gt}}^{\text{depth}} - \hat{I}^{\text{depth}}\|_2 \\ \mathcal{L}_{\text{mesh}} &= \mathcal{L}_{\text{mask}} + \lambda_{\text{depth}}\mathcal{L}_{\text{depth}} \end{aligned} \tag{5.1}$$

Where the loss in Equation 5.1 replaces the losses in Equation 4.10 and we use $\lambda_{\text{depth}} = 10$ throughout our experiments following [She+23].

Dataset We evaluate the reconstruction quality on the dataset by Myles et al. [MPZ14], which is a collection of objects from the AIM@Shape database and other community repositories. The dataset consists of 116 objects ranging from highly detailed CAD scans to noisy scanned surfaces. Since some of these are non-watertight or contain skinny geometry poorly represented by iso-surfacing methods, NCD [Che+22] proposes a method to remove these objects, which we use to select 4 representative objects[1].

[1]We select the following scenes: filigree100k, gargoyle100k, gearbox, grayloc

Baseline We compare our mesh extraction quality to two *differentiable* surface extraction techniques, Marching Cubes [Nie03] and DMTet [She+21], which provide a direct comparison in Figure 5.4. However, we do not compare to another closely related work, FlexiCubes [She+23], even though we follow a similar evaluation strategy since they neither provide the exact shapes they evaluate on nor source code for their method at the time of writing. Therefore, our results are not directly comparable to theirs.

When reconstructing meshes, the geometric resolution of the output surface is a key factor determining representable detail level and, therefore, a lower bound on the minimal reconstruction error of the base surface. Even though Marching Cubes and DMTet predict SDF on a uniform voxel grid, their output surface resolution differs due to using cubes or tetrahedral primitives, respectively. To match these two strategies, we follow FlexiCubes and increase the grid resolution for DMTet across comparisons to match the reconstructed triangle count with Marching Cubes. Since our method takes a Marching Cubes surface as input and re-meshes it with a $\sqrt{2}$ longer edge length, we use an input grid resolution of $\lceil \sqrt{2}G \rceil$ for evaluating our method, where $G \in \mathbb{N}$ is the grid resolution used in Marching Cubes.

Evaluation Metrics We evaluate the reconstructed meshes for reconstruction accuracy and intrinsic quality of the respective mesh. For a quantitative comparison, we follow NDC [Che+22] and compute Chamfer Distance (CD), F-Score (F1), Edge Chamfer Distance (ECD), and Edge F-Score (EF1) w.r.t. to the ground truth mesh. However, all these metrics are computed on point clouds sampled from meshes and only measure extrinsic surface accuracy, ignoring intrinsic mesh topology. To measure the reconstructed meshes' intrinsic properties, i.e., the topological quality, we also compare minimal and maximal angles and face aspect and radius ratios and provide qualitative comparisons.

We center each mesh such that its centroid is at the coordinate origin and its bounding box is in $[-1, 1]^3$ following NDC [Che+22]. Additionally, we observe our method predicting invisible geometry inside the supervised surface, which produces strong outliers in the metrics presented below. To give a fair comparison of the supervised reconstruction quality, before comparing to the ground truth surface, we remove all internal geometry by performing connected-component analysis and only keeping the largest one. Then, we randomly sample 10^5 points on the surface of each mesh, obtaining a point cloud $P = \{p_1, p_2, \ldots, p_{10^5}\}$ with $p_i \in \mathbb{R}^3$ and evaluate the reconstruction accuracy with the following metrics:

- Chamfer distance (CD): We compute the symmetric Chamfer Distance between the ground truth point cloud P_{gt} and the predicted one P_{pred}:

$$CD(P_{\text{pred}}, P_{\text{gt}}) = \frac{1}{|P_{\text{pred}}|} \sum_{p \in P_{\text{pred}}} \min_{q \in P_{\text{gt}}} \|p - q\|_2^2 + \frac{1}{|P_{\text{gt}}|} \sum_{p \in P_{\text{gt}}} \min_{q \in P_{\text{pred}}} \|p - q\|_2^2 \tag{5.2}$$

- F-score (F1): Following NDC [Che+22] we define precision and recall as:

$$\begin{aligned}\text{Precision}\left(P_{\text{pred}}, P_{\text{gt}}\right) &= \frac{1}{|P_{\text{pred}}|}\sum_{p\in P_{\text{pred}}}\max_{q\in P_{\text{gt}}}\mathbf{1}_{\|p-q\|<\epsilon}\\ \text{Recall}\left(P_{\text{pred}}, P_{\text{gt}}\right) &= \frac{1}{|P_{\text{gt}}|}\sum_{p\in P_{\text{gt}}}\max_{q\in P_{\text{pred}}}\mathbf{1}_{\|p-q\|<\epsilon}\\ \text{F1}\left(P_{\text{pred}}, P_{\text{gt}}\right) &= \frac{2\text{Precision}\left(P_{\text{pred}}, P_{\text{gt}}\right)\text{Recall}\left(P_{\text{pred}}, P_{\text{gt}}\right)}{\text{Precision}\left(P_{\text{pred}}, P_{\text{gt}}\right)+\text{Recall}\left(P_{\text{pred}}, P_{\text{gt}}\right)}\end{aligned} \tag{5.3}$$

  Where $\mathbf{1}_{\text{condition}}$ is an indicator function returning 1 if the condition is true and 0 otherwise. Throughout this evaluation, we use $\epsilon = 0.005$ as the threshold for point samples as in NDC [Che+22].

- Edge chamfer distance (ECD): To better evaluate the preservation of sharp features, NDC computes Edge Chamfer Distance as the Chamfer Distance over points near sharp edges and corners. A point in the set $E(P)$ is considered "sharp" if, in a ball of radius 0.004 around it, there exists a sufficiently different normal vector, or more precisely:

$$E(P) = \{p_i \in P \mid \min_{p_j\in P:\|p_i-p_j\|\leq 0.004}\langle n_i, n_j\rangle < 0.2\} \tag{5.4}$$

  Where $n_i, n_j \in \mathbb{R}^3$ are the face normals of the samples p_i, p_j. The edge chamfer distance is then just the chamfer distance over edges:

$$ECD(P_{\text{pred}}, P_{\text{gt}}) = CD(E(P_{\text{pred}}), E(P_{\text{gt}})) \tag{5.5}$$

- Edge F-score (EF1): Similarly, the Edge F-Score is simply the F-Score between the sharp edge points:

$$\text{EF1}(P_{\text{pred}}, P_{\text{gt}}) = \text{F1}(E(P_{\text{pred}}), E(P_{\text{gt}})) \tag{5.6}$$

Since our method focuses on extracting meshes with high topological quality, we evaluate the intrinsic quality of our meshes by comparing the minimal and maximal angles per face and compute the aspect and radius ratio of each face using PyVista [SK19].

- Minimal and maximal face angle: For every face $F \in \mathcal{F}$ we define the minimal and maximal angles as follows:

$$\angle_{\text{min}}(F) = \min_{v_i,v_j\in F}\angle(v_i, v_j) \qquad \angle_{\text{max}}(F) = \max_{v_i,v_j\in F}\angle(v_i, v_j) \tag{5.7}$$

- Radius ratio (RR): Intuitively, this metric measures the ratio of a circle circumcising each face to the one inscribed in it. For triangles, this metric is well-defined, but for quadrilaterals, the circumradius and inradius are not always defined, so the following alternative is used:

$$RR_t = \frac{R}{2r} \qquad RR_q = \frac{|q|_2 h_{\text{max}}}{\min_i \mathcal{A}_i} \tag{5.8}$$

  Where $R, r \in \mathbb{R}$ denote the circumradius and inradius of the triangle t respectively. In the quadrilateral case, $|q|_2$ is the sum of squared edge lengths, h_{max} is the longest diagonal and edge lengths, and $\min_i \mathcal{A}_i$ is the smallest area of the 4 triangles extractable from q.

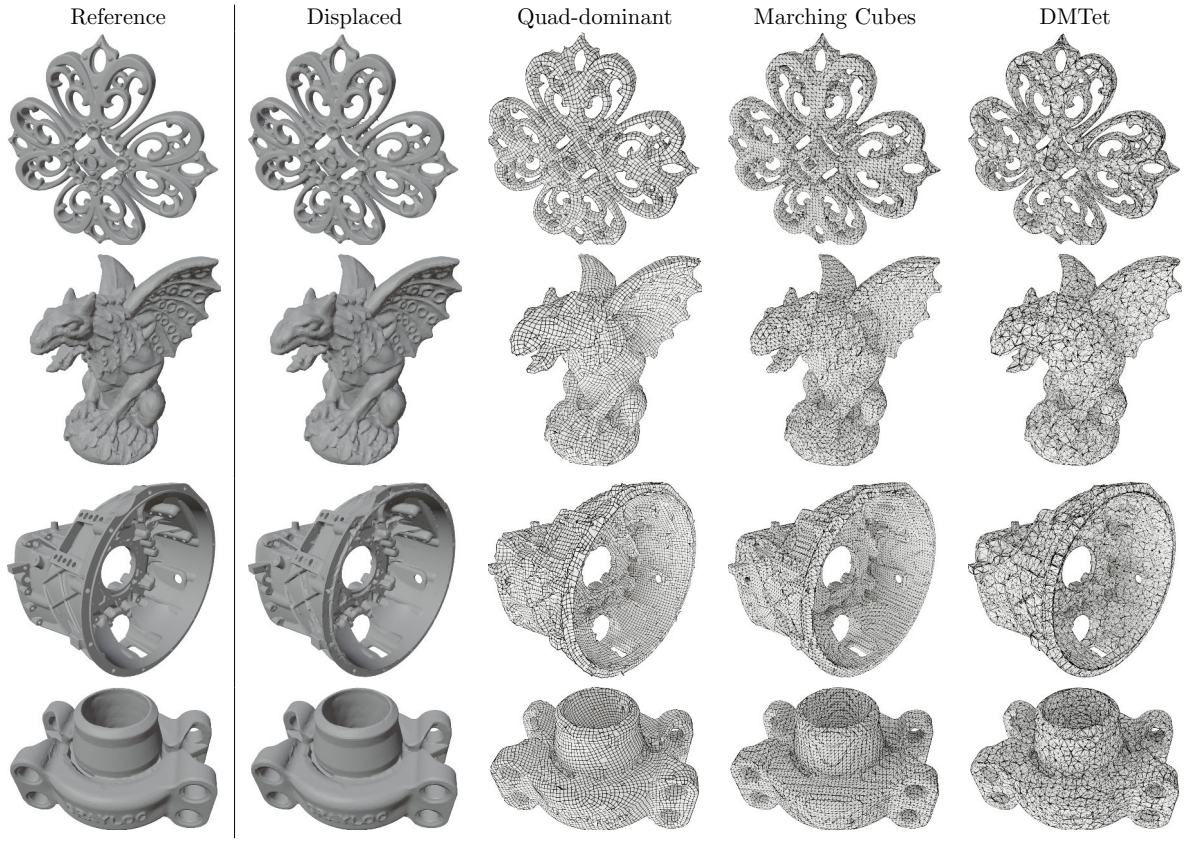

Figure 5.4.: *Visual comparison of different mesh extraction techniques. Our displaced and quad-dominant meshes, Marching Cubes, and DMTet [She+21] are generated by directly supervising geometry with depth and mask losses on the ground truth geometry. Our quad-dominant meshes show clear alignment of the quad faces to object features and extraction of surface details visible in the displaced surfaces.*

- Aspect ratio (AR): This measures the ratio of the longest edge to the radius of the inscribed circle in the case of triangles. For quads, the ratio of lengths to area is measured.

$$AR_t = \frac{|t|_\infty}{2\sqrt{3}r} \qquad AR_q = \frac{|q|_1|q|_\infty}{4\mathcal{A}} \tag{5.9}$$

  For a triangle t, $|t|_\infty$ denotes the maximal edge length, and r is the inradius of t. $|q|_1$ denotes the quad perimeter and $|q|_\infty$ denotes the largest edge length. $\mathcal{A}$ is the area of q.

Results We quantitatively compare both the extrinsic quality of the mesh reconstruction of Marching Cubes, DMTet, and our method in Table 5.3. In extrinsic metrics comparing the accuracy of the reconstructions with the ground truth, we see that our method cannot extract surfaces as accurately as Marching Cubes and DMTet as visible when comparing Chamfer Distances between the methods. However, in terms of F-score, which measures the accuracy of the surface in terms of a classification problem, this difference becomes much smaller, making our method comparable to both Marching Cubes and DMTet.

Since the major benefit of our method is not the exact reconstruction of surfaces but the extraction of meshes with clean topology and isometric faces aligned to the object's features, we

64^3	CD (10^{-5}) ↓	F1 ↑	ECD (10^{-2}) ↓	EF1 ↑	AR > 4 (%) ↓	RR > 4 (%) ↓	#V (10^3)	#F (10^3)
Marching Cubes (64^3)	1.23	0.83	0.16	0.23	11.58	11.58	16.75	33.75
DMTet (80^3)	<u>1.20</u>	<u>0.84</u>	**0.12**	<u>0.29</u>	20.94	21.40	18.00	35.75
Quad-Dominant (91^3)	1.37	0.82	0.17	0.22	<u>2.27</u>	**3.56**	12.25	12.25
Displaced (91^3)	**1.16**	**0.85**	<u>0.13</u>	**0.34**	**0.78**	<u>3.99</u>	51.50	51.50

Table 5.3.: *Quantitative results on Mesh Reconstruction on 4 objects from the dataset collected by Myles et al. [MPZ14]. We report the following metrics: CD: Chamfer Distance, F1: F1 Score, ECD: Edge Chamfer Distance, EF1: Edge F1 Score, #V: Number of vertices, #F: Number of faces.*

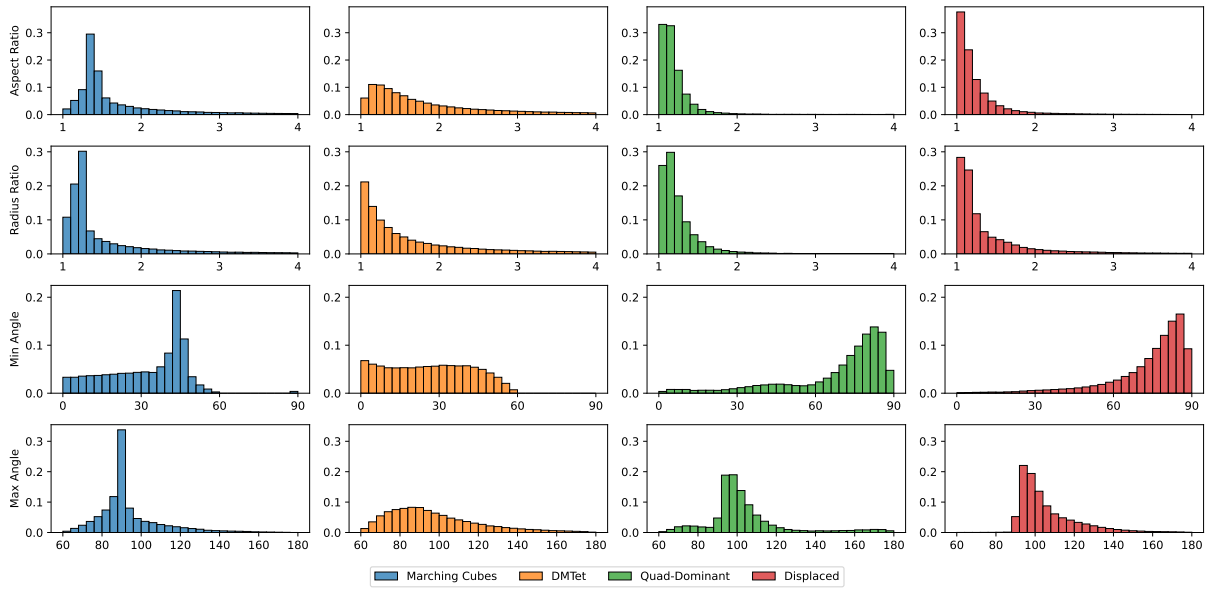

Figure 5.5.: *Quantitative comparison of the minimal and maximal angle within each face for all four scenes selected from the dataset collected by Myles et al. [MPZ14].*

also compare the intrinsic mesh quality of all methods in terms of aspect ratio outliers (AR > 4) and radius ratio outliers (RR > 4). The threshold of 4 is empirically chosen to represent acceptable mesh quality [She+23]. Since our method extracts mostly isometric quad faces, we improve over the baselines of Marching Cubes and DMTet drastically without requiring further regularization as in [She+23].

Similarly, by comparing the distributions of minimal and maximal angle and aspect and radius ratios over our dataset in Figure 5.5, we see the radius and angle ratios are more concentrated around the lower bound of 1 for our method than either Marching Cubes or DMTet. In terms of face angles, our meshes contain a majority of faces with minimal and maximal face angles of 90°, indicative of the fact that the majority of the faces our method extracts are quad faces. When additionally applying one step of Catmull-Clark subdivision, individual faces become even more isometric due to the subdivision process approximating a bi-cubic smooth surface as seen in the right-most column in Figure 5.5.

Qualitatively, Figure 5.4 shows the reconstructed meshes for each extraction method, showing

Scene	#Train	#Test	Train Res.
HEAD	66	66	512x512 px
GOLD	119	119	512x512 px

Scene	#Train	#Test	Train Res.
FAMILY	135	33	960x540 px
IGNATIUS	234	56	960x540 px

Table 5.4.: *Dataset statistics for the scenes used from the NeRD real world [Bos+21a] and Tanks and Temple [Kna+17] datasets respectively.*

PSNR (dB) ↑	HEAD	GOLD	Avg
Nvdiffrec	**27.22**	**24.34**	**25.78**
Our	<u>26.72</u>	<u>21.46</u>	<u>24.09</u>

SSIM ↑	HEAD	GOLD	Avg
Nvdiffrec	**0.934**	**0.884**	**0.909**
Our	<u>0.931</u>	<u>0.862</u>	<u>0.896</u>

LPIPS ↓	HEAD	GOLD	Avg
Nvdiffrec	<u>0.096</u>	**0.114**	**0.105**
Our	**0.083**	<u>0.127</u>	<u>0.105</u>

Table 5.5.: *Quantitative metrics on the NeRD real-world dataset [Nie+20].*

that both Marching Cubes and DMTet fail to align faces with the ground truth geometry while maintaining comparable mesh accuracy.

Additionally, since our quad-dominant meshes almost entirely consist of quad faces, they are suitable for Catmull-Clark subdivision, increasing the geometry resolution and subsequent displacement, further improving reconstruction quality. Since subdivision is only applied on the extracted quad-dominant surface and not the SDF grid, this increase in resolution does not come at a significant computational increase while providing a more accurate surface reconstruction. Therefore, our meshes are artist-friendly since the low-resolution quad-dominant surface is regular enough to be easily manipulated while offering independent control over displacement in the form of a texture map. Since subdivision and displacement are usually only applied at render-time, only the low-resolution quad-dominant meshes need to be edited by artists [Bur+18].

5.3. Real-World Reconstruction

To validate the real-world applicability of our method, we evaluate it on a diverse set of images captured in indoor and outdoor environments, from medium-scale objects to large-scale statues. However, since ground-truth geometry is generally not available, we compare to our baseline Nvdiffrec [Mun+22] based on the view interpolation metrics alone as defined in Section 5.1. Geometric quality is only evaluated qualitatively based on Figure 5.7 and Figure 5.6.

Datasets We provide reconstructions on two scenes from the NeRD dataset [Bos+21a] consisting of diverse camera poses around a single object. Both objects are highly reflective and have a relatively simple base shape with small surface scale details. Following Nvdiffrec [Mun+22], we downsample all images to 512x512px and use them for both training and testing. Considering larger scale objects, we also evaluate on two statues of the Tanks and Temple dataset [Kna+17] captured outdoors in daylight. Details on both datasets are given in Table 5.4.

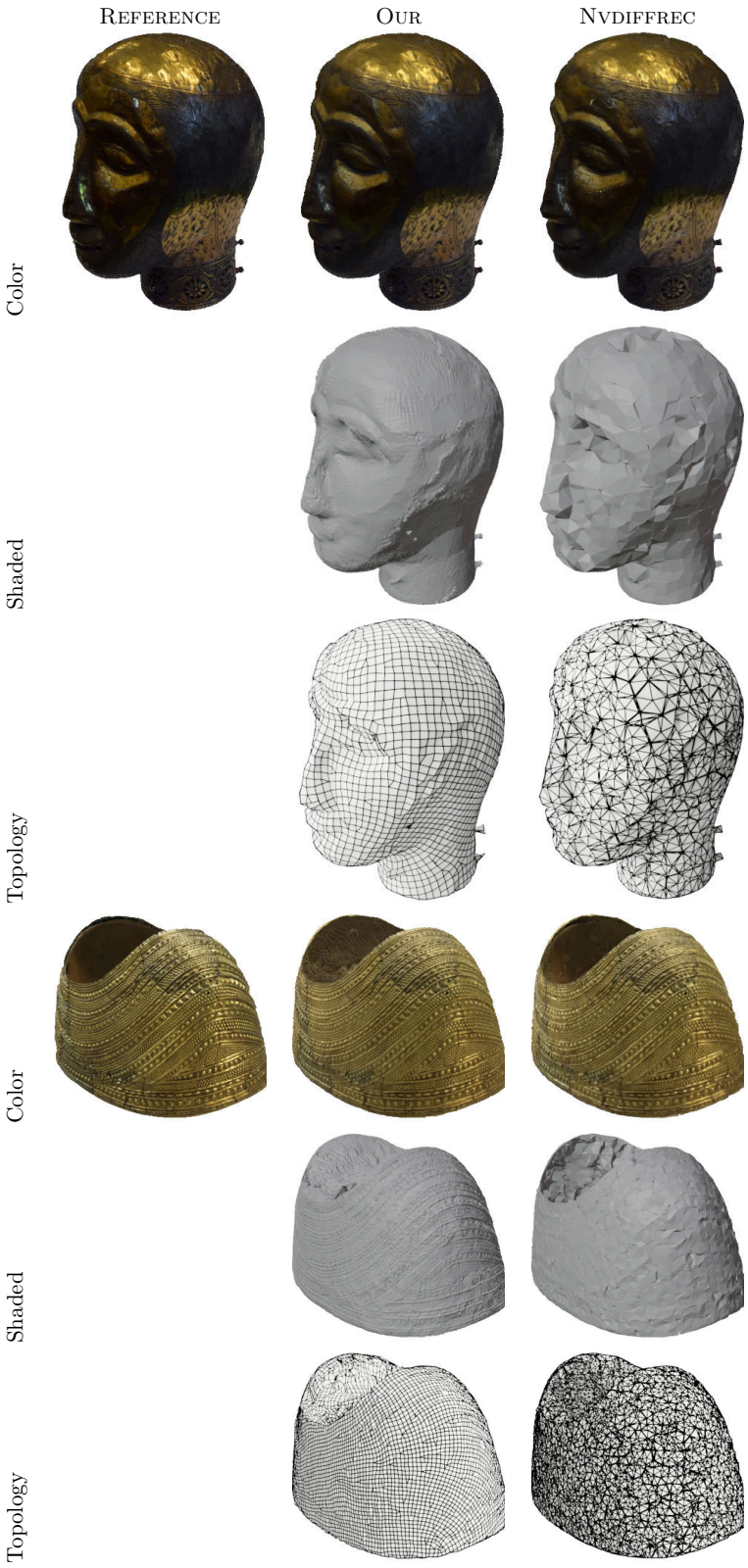

Figure 5.6.: *Qualitative results on the NeRD dataset [Bos+21a]. We show the reconstructed colors, shaded mesh, and topology for our's and Nvdiffrec's method. In the shaded view, we show the displaced mesh for our method and the DMTet mesh for Nvdiffrec.*

Figure 5.7.: *Qualitative results on the Tanks and Temple dataset [Kna+17].*

Results In Figure 5.6 and Figure 5.7, we show qualitative reconstruction results for each dataset described in the previous paragraph. In general, we again observe comparable reconstruction performance between Nvdiffrec and our method, while for our method, quad-dominant faces align well with geometric features, as shown in Figure 5.6.

On the GOLD CAPE scene, our decomposition into a quad-dominant mesh and displacement representing small-scale details is particularly noticeable in the embossed surface. Here, the quad faces are too large to model individual bumps, which are purely described by surface displacement. This is akin to how an artist would model such an object, by defining the overall shape as a hand-crafted mesh and then applying displacement to add small-scale details.

However, this scene also highlights a failure case of our method. As we initialize object reconstruction from a sphere, instead of randomly as Nvdiffrec, we fail to extract the hole of this crown-like object correctly. This is most likely due to our method not receiving enough image supervision showing this empty space and, therefore, being unable to overcome this local minimum in the extracted surface location.

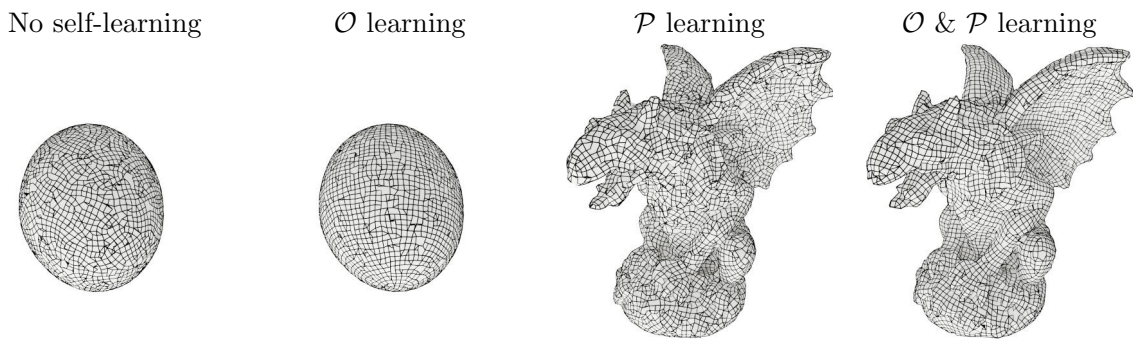

Figure 5.8.: *Leaning no field or just the orientation or just the position field causes the surface to not converge at all or produces sub-par face alignment to the surface geometry respectively. Only learning both the orientation and position fields both reconstructs the surface well and causes faces to align to object features.*

91^3	CD (10^{-5}) ↓	F1 ↑	AR > 4 (%) ↓	RR > 4 (%) ↓	S(RoSy) ↓	S(PoSy) ↓
No self-learning	1549.95	0.02	5.33	9.07	<u>195.00</u>	**0.00**
$\mathcal{O}$ learning	1548.18	0.02	4.18	5.70	**5.00**	**0.00**
$\mathcal{P}$ learning	**1.26**	**0.82**	<u>2.94</u>	<u>5.46</u>	1160.75	<u>2.50</u>
$\mathcal{O}$ & $\mathcal{P}$ learning	<u>1.38</u>	**0.82**	**2.27**	**3.56**	660.75	5.50

Table 5.6.: *Quantitative results for ablating our $\mathcal{O}$, $\mathcal{P}$ self-learning formulation. We measure reconstruction quality using CD: Chamfer Distance and F1: F1 Score. Intrinsic face quality is measured using AR>4: Aspect ratio outliers and RR>4: Radius ratio outliers. Orientation singularities $S(RoSy)$ and position field singularities $S(PoSy)$ measure the field quality.*

5.4. Ablations

In this section, we ablate our method regarding the need for self-learning orientation and position fields and the use of a multi-resolution hierarchy by performing reconstructions on the dataset described in Section 5.2. We use the same setup as presented in Section 5.2 throughout this section.

Ablating our self-learning formulation with qualitative results in Figure 5.8 and quantitative ones in Table 5.6, we show that both learning the orientation and position fields is crucial to extracting high-quality meshes. Figure 5.8 shows that learning the position field is crucial to represent the object at all, as without it the optimization cannot progress from the initial sphere shape. We hypothesize that this behavior is due to the position field changing significantly each iteration, if randomly initialized, causing a constantly changing surface location and therefore not allowing for consistent gradients to be back-propagated to the SDF. Additionally, not learning the orientation field causes inferior face alignment to object features and as expected drastically increases the number of orientation singularities as well as worse intrinsic face quality as shown in Table 5.6. Only learning both the orientation and position field, faithfully captures the input geometry while providing high-quality topology.

Discussion and future work

6.1. Discussion

This work proposes extending current 3D reconstruction systems with a quad-dominant re-meshing step. We show improved topology by automatically providing more uniform faces and aligning them to surface features to provide consistent quad-strips similar to ones created by human artists. Due to the differentiability of our formulation, our re-meshing step can be applied as an upgrade mechanism to current iso-surfacing algorithms such as Marching Cubes [Nie03], or Marching Tetrahedra [She+21]. By introducing a differentiable Catmull-Clark subdivision, we provide a refinement strategy dependent only on the extracted surface, not the number of voxels used during iso-surfacing. This allows for optimization over high-resolution meshes to extract high-frequency detail with displacement while storing these details in a lightweight displacement field and still providing an efficient representation for artists to edit.

Our method can achieve comparable results to the state-of-the-art method Nvdiffrec [Mun+22] regarding visual- and geometric accuracy. However, by comparing the extracted meshes qualitatively and considering the intrinsic mesh quality, we show that our method can reconstruct more isometric faces and provide consistent topology connecting individual faces, a property otherwise only found in manually created meshes.

We believe that the main advantage of 3D reconstruction from multi-view images is providing a high-quality starting point for artists to adapt the extracted meshes to their particular application, for instance, by changing mesh topology or editing materials. Therefore, quad-dominant meshes are a step in this direction, as they are an in-between representation between traditional triangle-soup models and hand-crafted pure-quad models used in character animation while being efficient to extract and optimize over. Additionally, our quad-dominant meshes contain significantly fewer faces while achieving similar scores to state-of-the-art results, resulting in lighter-weight meshes with similar representational power.

However, our approach does not come without limitations. Firstly, since we re-mesh a triangular input mesh, the edge length of re-meshed faces needs to be larger than the triangular input faces. As an arbitrarily oriented quad face needs to be placed on top of each connected cluster of input triangles, the re-meshing size is lower-bounded by $\sqrt{2}$ times the voxel size used to extract the input triangle mesh. This reduces the extracted mesh resolution, making it harder to represent small-scale structures, smaller than a single quad face but above the displacement size.

Additionally, since our method requires at least a couple of connected triangles to extract a single quad face and propagate gradients to the underlying SDF effectively, we cannot use randomly initialized per-vertex SDF values as in Nvdiffrec. We enforce this constraint of using a sufficiently smooth surface during re-meshing by representing the SDF with an MLP initialized to a sphere. However, this introduces difficulties representing objects with large genus, as there is a binary transition between predicting a closed surface to one with a hole. Also, this surface representation is prone to getting stuck in local minima in weakly-supervised concave surface regions, as only few ground truth images contribute to shading gradients there as visible in the shovel of the LEGO scene in Figure 5.2.

Like other NeRF-style methods, we fit a new model for each object and do not reuse information between them. Lifting this limitation and only fine-turning the extracted models on a newly captured set of images could improve the method's applicability and decrease the per-object reconstruction times.

6.2. Future work

Throughout this work, we have recognized the importance of high-quality meshes and believe their automatic extraction from images has great potential throughout the production process. Below, we discuss some ideas to improve upon the method as presented.

6.2.1. General improvements

Optimized implementation Our method would greatly benefit from a fused GPU implementation in CUDA, speeding up the quad-dominant mesh extraction process, currently our pipeline's most time-consuming step. Most operations in the field optimization process are trivially parallel as formulated. Care must be taken during graph extraction and clustering, as the greedy clustering algorithm processes edges in a fixed order. Currently, we perform the majority of these operations on the CPU due to ease of implementation within the time constraints of this work.

High level field optimization The extracted quad meshes could be further improved by reducing the number of singularities present in the position field by taking higher-level optimization goals into account, such as the ones presented in QuadriFlow [Hua+18]. For instance, by completely removing triangular faces, quad-focussed texturing systems such as Ptex [BL08] could remove the need for UV-unwrapping, as well as removing seams introduced by Catmull-Clark subdivision on non-quad faces.

Improved SDF representation We currently parametrize the SDF representing the object's surface with an MLP and hash-encoding initialized to a sphere with genus zero. However, if the object to reconstruct is of a higher genus, a binary transition from predicting a genus zero surface to introducing a hole has to happen, which is prone to getting stuck in local minima. Since this state transition is non-continuous, we cannot hope that local gradients provide sufficient information in this case. Experimentally, we still see correct behavior, though, which we attribute to its use of momentum to surpass these local optima. Nvdiffrec, solves this problem by initializing the SDF randomly at each voxel, effectively starting with a high genus surface [Mun+22]. In a similar setting, NeuS proposes to perform a ray-accumulation-based formulation of rendering an SDF [Wan+21], which could be valuable to bring to explicit meshes.

Improved shading model Since we inherit our shading and rendering model from Nvdiffrec, we also inherit its limitations. Namely, our method does not consider global illumination or shadows causing inaccurate material extraction, especially apparent when re-lighting the extracted meshes under a different environment light probe. One solution to this problem is performing Monte Carlo (MC) ray tracing and applying denoising to reduce the considerable variance introduced by this process [HHM22]. Another promising approach could be to explore Neural Radiance Caching [Mül+21] to estimate indirect lighting contributions to consider the actual scene geometry during denoising.

6.2.2. Extensions to our method

Interactive re-meshing The optimal direction for placing a quad strip depends not only on the geometry of the reconstructed object but also on factors such as the downstream use of the mesh, for instance, for animation. Artists take these constraints into account when hand-creating meshes. Hence, our method could be extended to accept artist guides for the layout of quad faces on the reconstructed surface. Jakob et al. [Jak+15] have a similar idea but cannot update the input mesh when the surface topology changes. Since our method is fully differentiable, an interactive extension of our method could consider artist guidelines as optimization constraints while optimizing the input remaining surface representation to best fit the provided multi-view images through photometric consistency.

Visible surface extraction Our method is limited by the resolution of the Marching Cubes mesh we extract from the SDF. Since we use a voxel grid to query SDF values, memory consumption scales cubically with the mesh extraction resolution. By instead only rendering visible parts of the mesh for each iteration using an algorithm like Sphere Tracing, memory consumption could be drastically reduced. To get an explicit quad-dominant mesh from the zero-crossing of the SDF extracted by Sphere Tracing, a partial section of the mesh would have to be reconstructed. By directly predicting a suitable orientation- and position field only for the visible sections of a surface, a partial quad-dominant mesh could be built, reaching eventual consistency as the multi-view images cover more sections of the mesh.

Adaptive re-meshing size Currently, we use a fixed size for re-meshing the triangular input mesh, causing a fixed resolution of the quad-dominant mesh. Meanwhile, Artists vary the quad size and isometry over the surface to achieve consistent edge flows and achieve suitable surface characteristics. By adopting a variable re-meshing size s_q, similar effects could be achieved while mitigating the visible artifacts caused by traditional Level-of-detail approaches [She+23] since smoothing the re-meshing fields provides consistent edge flows.

Temporal re-meshing from videos Current dynamic NeRFs represent object motion by learning a static model of the scene and a time-varying displacement field to deform this static representation into a moving object [Pum+21; Wan+22; Zha+22]. Our approach could replace this static object representation to enforce a high-quality quad-dominant mesh simultaneously optimized to represent the observed object motion while extracting high-quality mesh. Since the object motion is explicitly related to the static base geometry by the displacement network, the topology of the static mesh could also be optimized to best fit the observed motion, automatically orienting faces to minimize distortion when the object is in motion. This approach could be used to learn deformation-aware meshes with topology to accommodate the observed motion.

Conclusion

Creating high-quality 3D models is one of the most time-consuming processes of creating any CG feature movie or video game, costing artists hundreds of hours. To accelerate this process, photogrammetry is a popular approach, coming with challenges of extracting meshes with excessive face counts and entangling material properties, requiring involved post-processing. Recent methods in inverse rendering alleviate some of these shortcomings, such as NeRFactor [Zha+21b] providing factored material- and lighting, or Nvdiffrec [Mun+22] additionally extracting triangular meshes. In this work, we aimed to take this approach one step further and directly extract quad-dominant meshes from multi-view images alone, directly providing artists with high-quality meshes with edge-aligned faces and comparatively low face counts.

Our key contribution is a differentiable formulation of optimizing an orientation- and position field while back-propagating gradients back to an initially extracted surface as a triangle mesh. This allows our method to use a variety of iso-surfacing algorithms to extract an initial guess for the object's surface and apply our re-meshing technique to extract a quad-dominant mesh from them. The re-meshing step drastically improves the extracted meshes' topological quality at the cost of some representational power.

Secondly, we propose to apply the well-known Catmull-Clark subdivision procedure as a suitable method for subdividing quad-dominant meshes in a fully differentiable context. By applying displacement mapping, we extract fine-scale details, previously only achievable at excessive voxel grid resolutions. By formulating both differentiably, we can account for the changes in surface location introduced by both Catmull-Clark subdivision and re-meshing and directly optimize the extracted mesh to accurately represent the captured object.

Compared with the state-of-the-art 3D reconstruction techniques, our method extracts superior topology with the option of changing mesh level detail after post-reconstruction by choosing a desirable subdivision level and applying displacement. Even though, our method is currently not able to outperform related works in terms of reconstruction accuracy, we see great potential in extracting high-quality meshes and provide many possible directions for future work.

Bibliography

[Bar+21] Jonathan T. Barron et al. “Mip-NeRF: A Multiscale Representation for Anti-Aliasing Neural Radiance Fields”. In: *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2021 IEEE/CVF International Conference on Computer Vision (ICCV). Montreal, QC, Canada: IEEE, Oct. 2021, pp. 5835–5844. ISBN: 978-1-66542-812-5.

[BDC01] A. Broadhurst, T.W. Drummond, and R. Cipolla. “A Probabilistic Framework for Space Carving”. In: *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*. Eighth IEEE International Conference on Computer Vision. Vol. 1. Vancouver, BC, Canada: IEEE Comput. Soc, 2001, pp. 388–393. ISBN: 978-0-7695-1143-6.

[BL08] Brent Burley and Dylan Lacewell. “Ptex: Per-Face Texture Mapping for Production Rendering”. In: *Computer Graphics Forum* 27.4 (June 2008), pp. 1155–1164. ISSN: 01677055, 14678659.

[Bom+13] David Bommes et al. “Quad-Mesh Generation and Processing: A Survey: Quad-Mesh Generation and Processing”. In: *Computer Graphics Forum* 32.6 (Sept. 2013), pp. 51–76. ISSN: 01677055.

[Bos+21a] Mark Boss et al. “NeRD: Neural Reflectance Decomposition from Image Collections”. In: *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2021 IEEE/CVF International Conference on Computer Vision (ICCV). Montreal, QC, Canada: IEEE, Oct. 2021, pp. 12664–12674. ISBN: 978-1-66542-812-5.

[Bos+21b] Mark Boss et al. *Neural-PIL: Neural Pre-Integrated Lighting for Reflectance Decomposition*. Oct. 27, 2021. arXiv: 2110.14373 [cs]. URL: http://arxiv.org/abs/2110.14373 (visited on 02/28/2023). preprint.

[Bur+18] Brent Burley et al. “The Design and Evolution of Disney’s Hyperion Renderer”. In: *ACM Transactions on Graphics* 37.3 (June 30, 2018), pp. 1–22. ISSN: 0730-0301, 1557-7368.

[Bur12] Brent Burley. “Physically Based Shading at Disney”. In: *SIGGRAPH Talk* (2012).

[BZK09] David Bommes, Henrik Zimmer, and Leif Kobbelt. “Mixed-Integer Quadrangulation”. In: *ACM Transactions on Graphics* 28 (July 27, 2009).

[CC78] E. Catmull and J. Clark. “Recursively Generated B-spline Surfaces on Arbitrary Topological Meshes”. In: *Computer-Aided Design* 10.6 (Nov. 1, 1978), pp. 350–355. ISSN: 0010-4485.

[CCC87] Robert L. Cook, Loren Carpenter, and Edwin Catmull. “The Reyes Image Rendering Architecture”. In: *ACM SIGGRAPH Computer Graphics* 21.4 (Aug. 1, 1987), pp. 95–102. ISSN: 0097-8930.

[Che+19] Wenzheng Chen et al. “Learning to Predict 3D Objects with an Interpolation-Based Differentiable Renderer”. In: *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. 862. Red Hook, NY, USA: Curran Associates Inc., Dec. 8, 2019, pp. 9609–9619.

[Che+22] Zhiqin Chen et al. “Neural Dual Contouring”. In: *ACM Transactions on Graphics* 41.4 (July 22, 2022), 104:1–104:13. ISSN: 0730-0301.

[Com18] Blender Online Community. *Blender - a 3D Modelling and Rendering Package*. manual. Stichting Blender Foundation, Amsterdam: Blender Foundation, 2018.

[Coo84] Robert L. Cook. “Shade Trees”. In: *ACM SIGGRAPH Computer Graphics* 18.3 (Jan. 1, 1984), pp. 223–231. ISSN: 0097-8930.

[Die+21] Alexander Dielen et al. “Learning Direction Fields for Quad Mesh Generation”. In: *Computer Graphics Forum* 40.5 (Aug. 2021), pp. 181–191. ISSN: 0167-7055, 1467-8659.

[DV21] J. Dupuy and K. Vanhoey. “A Halfedge Refinement Rule for Parallel Catmull-Clark Subdivision”. In: *Computer Graphics Forum* 40.8 (Dec. 2021), pp. 57–70. ISSN: 0167-7055, 1467-8659.

[FSG17] Haoqiang Fan, Hao Su, and Leonidas Guibas. “A Point Set Generation Network for 3D Object Reconstruction from a Single Image”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Honolulu, HI: IEEE, July 2017, pp. 2463–2471. ISBN: 978-1-5386-0457-1.

[HHM22] Jon Hasselgren, Nikolai Hofmann, and Jacob Munkberg. *Shape, Light, and Material Decomposition from Images Using Monte Carlo Rendering and Denoising*. Oct. 4, 2022. arXiv: 2206.03380 [cs]. URL: http://arxiv.org/abs/2206.03380 (visited on 03/28/2023). preprint.

[Hua+18] Jingwei Huang et al. “QuadriFlow: A Scalable and Robust Method for Quadrangulation”. In: *Computer Graphics Forum* 37.5 (Aug. 2018), pp. 147–160. ISSN: 01677055.

[Jak+15] Wenzel Jakob et al. “Instant Field-Aligned Meshes”. In: *ACM Transactions on Graphics* 34.6 (Nov. 4, 2015), pp. 1–15. ISSN: 0730-0301, 1557-7368.

[Kaj86] James T Kajiya. “The Rendering Equation”. In: *SIGGRAPH* 20.4 (1986).

[Kar13] Brian Karis. “Real Shading in Unreal Engine 4”. In: *SIGGRAPH Course* (2013).

[KB17] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. Jan. 29, 2017. arXiv: 1412.6980 [cs]. URL: http://arxiv.org/abs/1412.6980 (visited on 09/24/2023). preprint.

[Kna+17] Arno Knapitsch et al. “Tanks and Temples: Benchmarking Large-Scale Scene Reconstruction”. In: *ACM Transactions on Graphics* 36.4 (Aug. 31, 2017), pp. 1–13. ISSN: 0730-0301, 1557-7368.

[Lai+20] Samuli Laine et al. *Modular Primitives for High-Performance Differentiable Rendering*. Nov. 6, 2020. arXiv: 2011.03277 [cs]. URL: http://arxiv.org/abs/2011.03277 (visited on 04/21/2023). preprint.

[LC87] William Lorensen and Harvey Cline. “Marching Cubes: A High Resolution 3D Surface Construction Algorithm”. In: *ACM SIGGRAPH Computer Graphics* 21 (Aug. 1, 1987), p. 163. ISSN: 0897912276.

[LDG18] Yiyi Liao, Simon Donne, and Andreas Geiger. “Deep Marching Cubes: Learning Explicit Surface Representations”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). Salt Lake City, UT: IEEE, June 2018, pp. 2916–2925. ISBN: 978-1-5386-6420-9.

[Li+23] Zhaoshuo Li et al. “Neuralangelo: High-Fidelity Neural Surface Reconstruction”. In: *IEEE Conference on Computer Vision and Pattern Recognition CVPR* (2023).

[Lie91] Pascal Lienhardt. “Topological Models for Boundary Representation: A Comparison with n-Dimensional Generalized Maps”. In: *Computer-Aided Design* 23.1 (Jan. 1, 1991), pp. 59–82. ISSN: 0010-4485.

[Mic98] D. Micciancio. “The Shortest Vector in a Lattice Is Hard to Approximate to within Some Constant”. In: *Proceedings 39th Annual Symposium on Foundations of Computer Science (Cat. No.98CB36280)*. Proceedings 39th Annual Symposium on Foundations of Computer Science (Cat. No.98CB36280). Nov. 1998, pp. 92–98.

[Mil+20] Ben Mildenhall et al. *NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis*. Aug. 3, 2020. arXiv: 2003.08934 [cs]. URL: http://arxiv.org/abs/2003.08934 (visited on 09/23/2022). preprint.

[MJ09] K.V. Mardia and P.E. Jupp. *Directional Statistics*. Wiley Series in Probability and Statistics. Wiley, 2009. ISBN: 978-0-470-31781-5.

[MPZ14] Ashish Myles, Nico Pietroni, and Denis Zorin. “Robust Field-Aligned Global Parametrization”. In: *ACM Transactions on Graphics* 33.4 (July 27, 2014), pp. 1–14. ISSN: 0730-0301, 1557-7368.

[Mül+21] Thomas Müller et al. “Real-Time Neural Radiance Caching for Path Tracing”. In: *ACM Transactions on Graphics* 40.4 (Aug. 31, 2021), pp. 1–16. ISSN: 0730-0301, 1557-7368.

[Mül+22] Thomas Müller et al. “Instant Neural Graphics Primitives with a Multiresolution Hash Encoding”. In: *ACM Transactions on Graphics* 41.4 (July 2022), pp. 1–15. ISSN: 0730-0301, 1557-7368.

[Mun+22] Jacob Munkberg et al. “Extracting Triangular 3D Models, Materials, and Lighting From Images”. In: *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). New Orleans, LA, USA: IEEE, June 2022, pp. 8270–8280. ISBN: 978-1-66546-946-3.

[Nic+22] Nicki Skafte Detlefsen et al. *TorchMetrics - Measuring Reproducibility in PyTorch*. Feb. 2022.

[Nie+20] Michael Niemeyer et al. “Differentiable Volumetric Rendering: Learning Implicit 3D Representations Without 3D Supervision”. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). Seattle, WA, USA: IEEE, June 2020, pp. 3501–3512. ISBN: 978-1-72817-168-5.

[Nie03] G.M. Nielson. “On Marching Cubes”. In: *IEEE Transactions on Visualization and Computer Graphics* 9.3 (July 2003), pp. 283–297. ISSN: 1077-2626.

[NJJ21] Baptiste Nicolet, Alec Jacobson, and Wenzel Jakob. “Large Steps in Inverse Rendering of Geometry”. In: *ACM Transactions on Graphics* 40.6 (Dec. 2021), pp. 1–13. ISSN: 0730-0301, 1557-7368.

[OVK17] Aaron Oord, Oriol Vinyals, and Koray Kavukcuoglu. “Neural Discrete Representation Learning”. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS’17. Red Hook, NY, USA: Curran Associates Inc., Dec. 7, 2017, pp. 6309–6318. ISBN: 978-1-5108-6096-4.

[Pum+21] Albert Pumarola et al. “D-NeRF: Neural Radiance Fields for Dynamic Scenes”. In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). Nashville, TN, USA: IEEE, June 2021, pp. 10313–10322. ISBN: 978-1-66544-509-2.

[PV02] Marc Pollefeys and Luc Van Gool. “From Images to 3D Models”. In: *Commun. ACM* 45 (July 1, 2002), pp. 50–55.

[Ray+08] Nicolas Ray et al. “N-Symmetry Direction Field Design”. In: *ACM Transactions on Graphics* 27 (Apr. 1, 2008).

[Sch+16] Johannes L. Schönberger et al. “Pixelwise View Selection for Unstructured Multi-View Stereo”. In: *Computer Vision – ECCV 2016*. Ed. by Bastian Leibe et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2016, pp. 501–518. ISBN: 978-3-319-46487-9.

[SF16] Johannes L. Schonberger and Jan-Michael Frahm. “Structure-from-Motion Revisited”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Las Vegas, NV, USA: IEEE, June 2016, pp. 4104–4113. ISBN: 978-1-4673-8851-1.

[She+21] Tianchang Shen et al. “Deep Marching Tetrahedra: A Hybrid Representation for High-Resolution 3D Shape Synthesis”. In: *Advances in Neural Information Processing Systems*. Vol. 34. Curran Associates, Inc., 2021, pp. 6087–6101.

[She+23] Tianchang Shen et al. "Flexible Isosurface Extraction for Gradient-Based Mesh Optimization". In: *ACM Transactions on Graphics* 42.4 (July 26, 2023), 37:1–37:16. ISSN: 0730-0301.

[SK19] C. Bane Sullivan and Alexander Kaszynski. "PyVista: 3D Plotting and Mesh Analysis through a Streamlined Interface for the Visualization Toolkit (VTK)". In: *Journal of Open Source Software* 4.37 (May 2019), p. 1450.

[SSS06] Noah Snavely, Steven M. Seitz, and Richard Szeliski. "Photo Tourism: Exploring Photo Collections in 3D". In: *ACM Transactions on Graphics* 25.3 (July 1, 2006), pp. 835–846. ISSN: 0730-0301.

[Sto+96] Michael Stokes et al. *A Standard Default Color Space for the Internet - sRGB*. Nov. 5, 1996. URL: https://www.w3.org/Graphics/Color/sRGB.html (visited on 09/25/2023).

[Wal+07] Bruce Walter et al. "Microfacet Models for Refraction through Rough Surfaces". In: *ESGR* (2007).

[Wan+21] Peng Wang et al. "NeuS: Learning Neural Implicit Surfaces by Volume Rendering for Multi-view Reconstruction". In: *NeurIPS* (2021).

[Wan+22] Yiming Wang et al. *NeuS2: Fast Learning of Neural Implicit Surfaces for Multi-view Reconstruction*. Dec. 10, 2022. arXiv: 2212.05231 [cs]. URL: http://arxiv.org/abs/2212.05231 (visited on 08/18/2023). preprint.

[You23] Jonathan Young. *Xatlas*. 2023.

[YRS22] Wang Yifan, Lukas Rahmann, and Olga Sorkine-Hornung. "Geometry-Consistent Neural Shape Representation with Implicit Displacement Fields". In: *ICLR 2022* (2022).

[Zha+21a] Kai Zhang et al. "PhySG: Inverse Rendering with Spherical Gaussians for Physics-based Material Editing and Relighting". In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). Nashville, TN, USA: IEEE, June 2021, pp. 5449–5458. ISBN: 978-1-66544-509-2.

[Zha+21b] Xiuming Zhang et al. "NeRFactor: Neural Factorization of Shape and Reflectance Under an Unknown Illumination". In: *ACM Transactions on Graphics* 40.6 (Dec. 2021), pp. 1–18. ISSN: 0730-0301, 1557-7368. arXiv: 2106.01970 [cs].

[Zha+22] Fuqiang Zhao et al. "Human Performance Modeling and Rendering via Neural Animated Mesh". In: *ACM Transactions on Graphics* 41.6 (Dec. 2022), pp. 1–17. ISSN: 0730-0301, 1557-7368.

Appendix

A.1. Pre-integrated lighting

A.1.1. Cube Map Filtering

Diffuse Irradiance Following Karis [Kar13], the diffuse irradiance is approximated independently of the current position. Integration of this term could either be done by Monte Carlo (MC) integration or by pre-filtering the environment map. Munkberg et al. [Mun+22] chose the latter approach outlined below:

$$\begin{aligned} L_d(x) &\approx \int_{H^2} L_e(\omega_i) \cos\theta_i d\omega_i \\ &\approx \sum_{x,y} \text{area}(x, y, N) L_e(\omega_i) \max\{\cos\theta_i, 0\} \end{aligned} \tag{A.1}$$

With the area of a pixel (x, y) in one of the 8 $N \times N$ cube map tiles defined as follows:

$$\begin{aligned} \text{area}(x, N) &= \arctan\left(\frac{|x - N/2| + 1}{N/2}\right) - \arctan\left(\frac{|x - N/2|}{N/2}\right) \\ \text{area}(x, y, N) &= \text{area}(x, N) * \text{area}(y, N) \end{aligned} \tag{A.2}$$

Since this sum is now over all pixels (x, y) in the environment map, all terms in the above equation where $\cos\theta_i < 0$ need to be removed.

A.1.2. Specular Irradiance

Unpacking the above, we see that the first term $S_1(\omega_o, r)$ is the integral of the specular BRDF weighted by a $\cos$ term integrated over the hemisphere. This is equivalent to the incoming radiance under constant white lighting (i.e. $L_e(x, \omega_i) = 1$). To compute this term numerically we can apply MC integration by sampling an incoming direction ω_i from the pdf. $p(\omega_i) = D(\omega_h, r)\cos\theta_h J(\omega_h) = D(\omega_h, r)\cos\theta_h \frac{1}{4\omega_o^T \omega_h}$:

$$\begin{aligned} S_1(\omega_o, r) &= \int_{H^2} f_s(\omega_o, \omega_i; r)\cos\theta_i d\omega_i \\ &\approx \frac{1}{N}\sum_{i=1}^{N} \frac{f_s(\omega_o, \omega_i; r)\cos\theta_i}{p(\omega_i)} \\ &= \frac{1}{N}\sum_{i=1}^{N} \frac{D(\omega_h)F(\omega_h, \omega_o)G(\omega_i, \omega_o)}{4\cos\theta_i\cos\theta_o}\cos\theta_i \frac{4\omega_o^T\omega_h}{D(\omega_h, r)\cos\theta_h} \\ &= \frac{1}{N}\sum_{i=1}^{N} \frac{F(\omega_h, \omega_o)G(\omega_i, \omega_o)}{\cos\theta_o\cos\theta_h}\omega_o^T\omega_h \end{aligned} \tag{A.3}$$

The next observation we make is that by multiplying and dividing by Schlick's Fresnel:

$$\hat{F}(\omega_h, \omega_o) = F_0 + (1 - F_0)\underbrace{(1 - \omega_o^T\omega_h)^5}_{=:F_c} = F_0 + (1 - F_0)F_c \tag{A.4}$$

We find that the remaining sum can be split again:

$$\begin{aligned} S_1(\omega_o, r) &\approx \frac{1}{N}\sum_{i=1}^{N} \frac{F(\omega_h, \omega_o)G(\omega_i, \omega_o)}{\cos\theta_o\cos\theta_h}\omega_o^T\omega_h \\ &\approx \frac{1}{N}\sum_{i=1}^{N} \frac{F_0 + (1 - F_0)F_c}{\hat{F}(\omega_h, \omega_o)} \frac{F(\omega_h, \omega_o)G(\omega_i, \omega_o)}{\cos\theta_o\cos\theta_h}\omega_o^T\omega_h \\ &\approx \frac{F_0}{N}\sum_{i=1}^{N} \underbrace{\frac{G(\omega_i, \omega_o)}{\cos\theta_o\cos\theta_h}\omega_o^T\omega_h}_{=:G_{\text{vis}}(\omega_i, \omega_o)} + \frac{1 - F_0}{N}\sum_{i=1}^{N} \frac{F_c G(\omega_i, \omega_o)}{\cos\theta_o\cos\theta_h}\omega_o^T\omega_h \\ &= \frac{F_0}{N}\sum_{i=1}^{N} G_{\text{vis}}(\omega_i, \omega_o) + \frac{1 - F_0}{N}\sum_{i=1}^{N} F_c G_{\text{vis}}(\omega_i, \omega_o) \\ &= \frac{F_0}{N}\underbrace{\sum_{i=1}^{N}(1 - Fc)G_{\text{vis}}(\omega_i, \omega_o)}_{=:\hat{S}_1^{\text{scale}}} + \underbrace{\frac{1}{N}\sum_{i=1}^{N} F_c G_{\text{vis}}(\omega_i, \omega_o)}_{=:\hat{S}_1^{\text{bias}}} \end{aligned} \tag{A.5}$$

Since the specular BRDF f_s is anisotropic at high angles of incidence, which is not easily modeled within our approximation, we assume that the incident direction is always aligned with the surface normal, i.e.:

$$n = \omega_i = \omega_o \tag{A.6}$$

According to [Kar13], this approximation is the largest source of error, as it prevents us from rendering long-tailed reflections at high angles of incidence. One last observation we can now make is that the outgoing direction ω_o in $S_1(\omega_o, r)$ can be expressed in spherical coordinates for some polar- and azimuthal angle $\theta \in [0, \pi/2]$, $\phi \in [0, 2\pi]$:

$$\omega_o = [\sin\theta\cos\phi, \sin\theta\sin\phi, \cos\theta]^T \tag{A.7}$$

As we assume anisotropic reflections in Equation 2.1, the BRDF is rotationally symmetric around ϕ, so we can assume $\phi = 0$. Therefore, ω_o is uniquely determined by it's polar angle θ:

$$\omega_o = [\sin\theta, 0, \cos\theta]^T = [\sqrt{1-\cos^2\theta}, 0, \cos\theta]^T \tag{A.8}$$

This finally gives us that $S_1(\omega_o, r)$ just has two Degrees of Freedom, ω_o's polar angle parametrized by $\cos\theta$ and the surface roughness, which are both in the range $[0, 1]$. So $S_1(\omega_o, r) = S_1(\cos\theta_o, r)$

In practice we compute both terms $\hat{S}_1^{\text{scale}}, \hat{S}_1^{\text{bias}}$ in Equation A.5 separately and store them in a lookup table $(\cos\theta_o, r) \rightarrow (\hat{S}_1^{\text{scale}}, \hat{S}_1^{\text{bias}})$. Since these terms do not depend on any scene parameters now, we can pre-compute this lookup table once offline and just query it at run-time without back-propagating through their estimators.

Environment Map pre-filtering The $S_2(\omega_o, r)$ term is a function of the environment map and, therefore, must be re-computed whenever this map changes. So, efficient and low variance estimation of this term is critical. Karis et al. [Kar13] do this by pre-multiplying the normal normal distribution function (NDF) $D(\omega_h)$ with the environment samples. This has the effect of blurring the environment map as roughness increases.

$$S_2(\omega_o, r) \approx \int_{H^2} D(\omega_h, r) L_e(x, \omega_i) \cos\theta_i d\omega_i \tag{A.9}$$

This additional dependence on the surface roughness can be efficiently represented using the levels of a MIP-map, where at higher MIP levels, larger roughness values are used for integration. In [Mun+22] this is implemented by successively down-sampling the base resolution environment map using average pooling in a $(2, 2)$ window and computing Equation A.10 for each MIP level with a linearly interpolated roughness value $r \in [r_{\text{min}}, r_{\text{max}}]$.

To compute the integral in Equation A.10, we could either use MC integration or utilize the fact that the environment map is a piece-wise constant function and integrate it in solid angles. Munkberg et al. [Mun+22] chose the latter approach, trading off lower noise at the cost of introducing some bias.

$$\begin{aligned} S_2(\omega_o, r) &\approx \int_{H^2} D(\omega_h, r) L_e(x, \omega_i) \cos\theta_i d\omega_i \\ &\approx \sum_x \text{area}(x) D(\omega_h, r) L_e(\omega_i) \cos\theta_i \end{aligned} \tag{A.10}$$

A.2. Symmetry fields

A.2.1. Rounding to lattice positions

For a regular lattice with normal $\mathbf{n}$, orientation $\mathbf{o}$ and position $\mathbf{p}$ the following operation rounds a position $\mathbf{p}'$ to the nearest lattice point:

$$\text{round}(\mathbf{n}, \mathbf{o}, \mathbf{p}, \mathbf{p}') := \mathbf{p} + \rho\left(\mathbf{o}\lfloor\gamma_1 + 1/2\rfloor + \mathbf{o}'\lfloor\gamma_2 + 1/2\rfloor\right) \tag{A.11}$$

with $\mathbf{o}' := \mathbf{n} \times \mathbf{o}$ and $\gamma_1 := \left(\mathbf{p}' - \langle \mathbf{p}, \mathbf{o}\rangle\right)/\rho$, $\gamma_2 := \left(\mathbf{p}' - \langle \mathbf{p}, \mathbf{o}'\rangle\right)/\rho$ and ρ is the target edge length.

A.2.2. Intersection point computation

A position $\mathbf{q}_{ij}$ that minimizes the distance to vertices $\mathbf{v}_i, \mathbf{v}_j$ while being constrained to the tangent planes of both vertices can be obtained by solving the following optimization problem:

$$\min_{\mathbf{q}_{ij}} \|\mathbf{v}_i - \mathbf{q}_{ij}\|_2^2 + \|\mathbf{v}_j - \mathbf{q}_{ij}\|_2^2 \tag{A.12}$$

$$\text{s.t.} \quad \langle \mathbf{n}_i, \mathbf{q}_{ij}\rangle = \langle \mathbf{n}_i, \mathbf{v}_i\rangle \text{ and } \langle \mathbf{n}_j, \mathbf{q}_{ij}\rangle = \langle \mathbf{n}_j, \mathbf{v}_j\rangle \tag{A.13}$$

Which has the unique solution:

$$\mathbf{q}_{ij} = \frac{1}{2}(\mathbf{v}_i + \mathbf{v}_j) - \frac{1}{4}(\lambda_i \mathbf{n}_i + \lambda_j \mathbf{n}_j) \tag{A.14}$$

$$\lambda_i = \frac{2\left\langle \mathbf{n}_i + \langle \mathbf{n}_i, \mathbf{n}_j\rangle \mathbf{n}_j, \mathbf{v}_j - \mathbf{v}_i\right\rangle}{1 - \langle \mathbf{n}_i, \mathbf{n}_j\rangle^2 + \epsilon} \tag{A.15}$$

$$\lambda_j = \frac{2\left\langle \mathbf{n}_j + \langle \mathbf{n}_i, \mathbf{n}_j\rangle \mathbf{n}_i, \mathbf{v}_i - \mathbf{v}_j\right\rangle}{1 - \langle \mathbf{n}_i, \mathbf{n}_j\rangle^2 + \epsilon} \tag{A.16}$$

Where the parameter $\epsilon = 10^{-4}$ ensures that if $\langle \mathbf{n}_i, \mathbf{n}_j\rangle \approx 1$, $\mathbf{q}_{ij}$ approximates the arithmetic mean of $\mathbf{v}_i$ and $\mathbf{v}_j$.

A.2.3. Wrapped van-Mises Distribution

The wrapped version of the van-Mises distribution can be efficiently computed by applying the double angle formula repeatedly ($\cos(2x) = 2\cos^2(x) - 1$).

$$\begin{aligned} 1 - \exp(\cos(4\theta) - 1) &= 1 - \exp(2\cos^2(2\theta) - 2) \\ &= 1 - \exp(2(2\cos^2\theta - 1)^2 - 2) \\ &= 1 - \exp(2(4\cos^4\theta - 4\cos^2\theta + 1) - 2) \\ &= 1 - \exp(8(\cos^4\theta - \cos^2\theta)) \end{aligned} \tag{A.17}$$

Given two unit vectors $\mathbf{a}, \mathbf{b}$, the equivalence $\cos\theta = \langle \mathbf{a}, \mathbf{b}\rangle$ allows for efficiently computing the powers of $\cos\theta$ in the above equation.

A.2.4. Local Coordinates

A local coordinate system can be fit around a normal $\mathbf{n}$ with:

$$\mathbf{s} = \mathbf{t} \times \mathbf{n} \qquad \mathbf{t} = \begin{cases} \begin{bmatrix} \mathbf{n}.z \\ 0 \\ -\mathbf{n}.x \end{bmatrix} / \sqrt{\mathbf{n}.x^2 + \mathbf{n}.z^2} & \text{if } |\mathbf{n}.x| > |\mathbf{n}.y| \\ \begin{bmatrix} 0 \\ \mathbf{n}.z \\ -\mathbf{n}.y \end{bmatrix} / \sqrt{\mathbf{n}.y^2 + \mathbf{n}.z^2} & \text{else} \end{cases} \tag{A.18}$$

The vectors $\mathbf{t}, \mathbf{s}$ then lie in the tangent space around $\mathbf{n}$.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

EXTRACTING WELL-BEHAVED QUAD MODELS, MATERIALS, AND LIGTHING FROM IMAGES

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
FRÜHAUF	MAXIMILIAN

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Zürich,	

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.