Joint UV Optimization and Texture Baking

JULIAN KNODT, LightSpeed Studios, USA
ZHERONG PAN, LightSpeed Studios, USA
KUI WU, LightSpeed Studios, USA
XIFENG GAO, LightSpeed Studios, USA

Level of detail (LOD) has been widely used in interactive computer graphics. In current industrial 3D modeling pipelines, artists rely on commercial software to generate highly detailed models with UV maps, and then bake textures for low-poly counterparts. In these pipelines, each step is performed separately, leading to unsatisfactory visual appearances for low polygon count models. Moreover, existing texture baking techniques assume the low-poly mesh has a small geometric difference from the high-poly, which is often not true in practice, especially with extremely low poly count models.

To alleviate the visual discrepancy of the low-poly mesh, we propose to jointly optimize UV mappings during texture baking, allowing for low-poly models to faithfully replicate the appearance of the high-poly even with large geometric differences. We formulate the optimization within a differentiable rendering framework, allowing the automatic adjustment of texture regions to encode appearance information. To compensate for view parallax when two meshes have large geometric differences, we introduce a Spherical Harmonic (SH) parallax mapping, which uses SH functions to modulate per-texel UV coordinates based on the view direction. We evaluate the effectiveness and robustness of our approach on a dataset composed of online downloaded models, with varying complexities and geometric discrepancies. Our method achieves superior quality over state-of-the-art techniques and commercial solutions.

CCS Concepts: • **Computing methodologies** → **Modeling and simulation**.

Additional Key Words and Phrases: Texture Baking, Differentiable Rendering, UV Optimization

ACM Reference Format:
Julian Knodt, Zherong Pan, Kui Wu, and Xifeng Gao. 2018. Joint UV Optimization and Texture Baking. In . ACM, New York, NY, USA, 19 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

The tradeoff between visual fidelity and high performance has been a central topic in interactive computer graphics for decades and will continue to be. Despite tremendous strides in graphics hardware, user requirements for highly realistic content always necessitates rendering more polygons than what is affordable for both high-end and low-end platforms. In light of that, level of detail (LOD) techniques have been widely used to render a low-poly model with fewer details when the model is small, distant, or unimportant, while rendering a high-resolution, detailed model when the viewer is close.

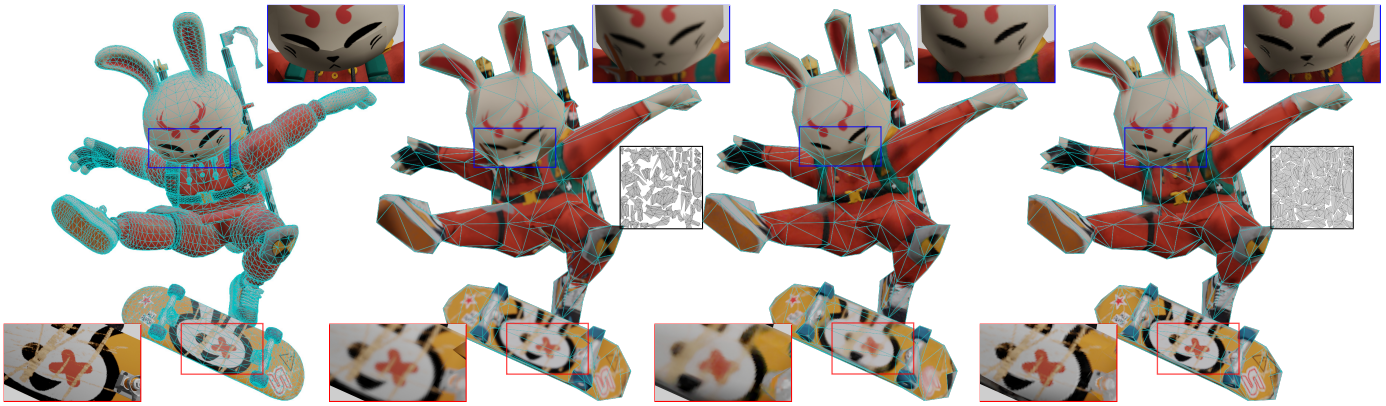
Transactions on Graphics, August 22nd, 2023, Bellevue, Washington

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

In industrial 3D modeling pipelines, artists create a high-poly model and rely on commercial software to generate low-poly counterparts to replicate the appearance. Typically, 3D artists first generate a low-poly mesh from its high-poly counterpart, then create a UV mapping for the low-poly mesh, and finally bake textures from the high-poly using commercial tools, such as Marmoset [Marmoset, 2022], Blender [Blender Online Community, 2018], Unreal Engine (UE) [Epic Games, 2022], Geogram [Lévy, 2019], InstaLOD [Nerurkar, 2021], Rapid Compact [DGG, 2018] and Substance Painter [Adobe, 2014]. This pipeline can often generate unsatisfactory results, requiring manual intervention due to several issues. First, the results of texture baking heavily rely on the outcome of UV mapping, but the quality of a UV mapping is not taken into account during texture baking, as they are performed independently. Without appearance-aware UV mappings, texels can be wasted on small or unimportant mesh regions, leaving salient regions ill-represented. Moreover, existing texture baking techniques assume there are small geometric differences between the low-poly and high-poly models, which is often not the case in practice. With large geometric discrepancies, both raycasting-based [Engel, 2019] and differentiable-rendering-based [Hasselgren et al., 2021, Li et al., 2018a, Liu et al., 2019, Loper and Black, 2014, Nimier-David et al., 2019] texture bakers suffer from view-parallax artifacts [Kaneko et al., 2001]. These artifacts manifest as strange copies from other parts of the model or blurry regions with unrecognizable features (Fig. 1). Given these issues, users must manually tune the interwoven factors of the model including the geometry, the UV parameterization, and the texture map, to achieve a satisfactory appearance.

Inspired by recent advances in differentiable rendering [Hasselgren et al., 2021], we propose a novel appearance baker that is robust to geometric differences. The novelty of our method lies in two aspects. First, our method jointly optimizes the texture map and UV coordinates guided by a visual difference metric. We alternatively update the texture content and UV parameterization, which effectively enlarges the solution space for high visual quality. Second, our method compensates for large geometric discrepancies using an optimized view parallax mapping. Specifically, we introduce *Spherical Harmonic (SH) Parallax mapping*, which uses a SH texture map to store coefficients for nonlinear Spherical Harmonic functions that shifts per-texel UV coordinates based on the view direction. The SH map is optimized jointly with the UV and texture maps to minimize the visual differences. The introduced SH map adds only a marginal overhead to the rendering pipeline but offers additional degrees of freedom to significantly improve visual similarity between the low-poly and the high-poly.

We have evaluated our method with 26 high-poly meshes and 78 meshes with different level-of-details, geometric discrepancies,



High-Poly UE NVDiffModeling Ours

Fig. 1. Compared with the high-poly (#Faces = 34K), low-polys (#Faces = 704) with textures baked using either the raycasting approach from UE [Epic Games, 2022] or the differentiable rendering approach from NVDiffModeling [Hasselgren et al., 2021] that exhibit various blurry artifacts (red) or inaccurate geometric details (blue), our method demonstrates the best baking quality. The PSNR/MS-SSIM values (higher is better) of the results generated by UE, NVDiffModeling, and ours are 21.17/0.921, 23.46/0.953, and 27.05/0.979, respectively. We also visualize the UV mesh for the original low-poly mesh (middle) and ours after optimization (far-right).

and appearance complexities. Compared with state-of-the-art approaches and commercial solutions, our method consistently delivers higher visual similarity at the same textures resolutions, in both PSNR and MS-SSIM. We attach the tested dataset and the results corresponding to all the figures in an anonymous github repository at https://anonymous.4open.science/r/tex_baking_supplementary/, including meshes, textures, and video clips of the final rendered results of each model in the dataset.

2 RELATED WORKS

In the following section, we review prior works on UV parameterization, texture baking, and bump maps that are related to our work.

2.1 UV Parameterization

UV parameterization maps 2D textures to 3D meshes and plays a central role in visual appearances. Given a mesh, generating a UV map typically involves three steps: 1) cutting open the mesh into one or more "charts", each of which can be flattened onto the 2D plane without introducing flips; 2) UV distortion minimization of all charts which involves reshaping and moving each chart; 3) packing optimized charts into an "atlas" [Sander et al., 2003, Young, 2017] according to texture resolutions desired by users. UV parameterization has been studied for decades, with early work [Lévy et al., 2002, Tutte, 1963] focusing on mapping from meshes isomorphic to disks to 2D planes. Recent work aims at improving the mapping quality by providing local injectivity [Rabinovich et al., 2017] or bijectivity [Jiang et al., 2017, Smith and Schaefer, 2015] guarantees. These works typically start from a valid initialization by Tutte's approach [Tutte, 1963] and then improve the quality by minimizing geometric distortion energies such as conformal mapping, Symmetric Dirichlet, AMIPS, or ARAP [Rabinovich et al., 2017, Smith and Schaefer, 2015]. Local injectivity and bijectivity guarantees are generally achieved via free-form boundary-checked optimizations [Smith and Schaefer, 2015] or scaffolding-based approaches [Jiang et al., 2017]. Among the abundant UV optimization techniques, few directly incorporate appearance attributes into the optimization process. The primary example of appearance aware optimization is [Sander et al., 2002], which optimizes the UV parameterization according to a measure of the complexity of given textures. We also note another recent work [Sun et al., 2022] with a similar goal of optimizing UV parameterization to improve texture efficiency. In this method, the shape of each parameterized 2D triangle is optimized by placing a camera facing each triangle to measure visual difference with the original parameterization, while our method focuses on a visual metric for the whole model from a specific distance. Unlike these works [Sander et al., 2002, Sun et al., 2022] that assume textures are constant throughout optimization, we jointly optimize both the texture content and the UV mapping to maximize measured visual similarity.

2.2 Texture Baking

Two techniques are widely used for texture baking: *Ray Casting* and *Differentiable Rendering*. The most widely used texture-baking technique is based on raycasting which has been the default approach in most popular 3D modeling software [Blender Online Community, 2018, Epic Games, 2022, Marmoset, 2022, Pixologic, 2022]. Given the high-poly and the low-poly model with its UV layout, this approach first casts a ray for each surface texel of the low-poly along its normal direction, then computes the interpolated attributes at the intersection point between the ray and the high-poly, and directly assigns the attribute to the texture of the low-poly. There are technical variants of how the origins and directions of the rays are determined [Engel, 2019], but regardless of the various choices of raycasting, ambiguity can happen for regions with significant geometric differences since multiple rays may intersect the high-poly at the same position, producing a baked color map with numerous artifacts. There have also have been recent works such as [Jiang

et al., 2020] which produce a bijective mapping between a high-poly mesh and a generated low-poly mesh, but they cannot handle the general case for an arbitrary high-poly and low-poly mesh. Recent advances in differentiable rendering [Hasselgren et al., 2021, Johnson et al., 2020, Laine et al., 2020, Li et al., 2018a, Liu et al., 2019, Loper and Black, 2014, Nimier-David et al., 2019, Patow and Pueyo, 2003] propose to resolve mapping ambiguity by changing the problem to a least-squares minimization of a visual similarity metric. Through auto-differentiation, attributes for the 3D scene can be automatically optimized by stochastic gradient descent [Kingma and Ba, 2015] to minimize visual metrics such as the ℓ_2 pixel-wise loss between rendered images of the low-poly and the high-poly. However, all existing works only consider optimizing the texture while fixing the given UV parameterization. As a result, the optimized textures can oftentimes be sub-optimal, where a large number of texels are wasted on small or unimportant regions, which may not even be visible from a distant view. Although optimizing textures via differentiable rendering is not a new concept [Hasselgren et al., 2021, Laine et al., 2020, Li et al., 2018a, Patow and Pueyo, 2003], our method is the first to jointly optimize UV and texture maps using differentiable rendering, leading to a much larger search space.

2.3 Bump Mapping

In interactive computer graphics, bump mapping refers to a collection of techniques representing small-scale geometry details, which enrich the geometric details of 3D models via pixel or texel perturbations. The distinctions between different bump mapping techniques lay in how they add details. *Normal mapping* [Cohen et al., 1998] changes the geometry's normal to create fine shading details. Since normals do not create a strong illusion of depth, *parallax mapping* [Kaneko et al., 2001] is often added to planar surfaces to add depth-parallax by shifting texture coordinates in the UV space, corresponding to the depth of elements. This idea was further improved by offset limiting [Welsh et al., 2004], which reduces artifacts at steep angles. Later, *steep parallax mapping* was introduced to perform multiple iterations of the UV shift by small steps, where a new depth is queried at each step of a ray-march through the depth map. Such ray-marching stops when it is below the object surface [McGuire and McGuire, 2005] and is then backtracked to the exact ray-object intersection point [Policarpo et al., 2005] or linear interpolation [Tatarchuk, 2006] is used as a fast approximation. These approaches need to perform multiple texture queries, which can cause significant overhead. Also these methods assume that the UV translation is always parallel to the original view direction, even when the UV parameterization has cuts and warps across triangular boundaries. Instead of modifying per-pixel attributes in the screen space, *displacement mapping* [Thonat et al., 2021] uses a height texture to displace vertices' positions in the vertex shader. While this is effective as a general-purpose method for recovering high-quality appearances, it generally requires significantly more computations since it requires the subdivision of the mesh to fit within each pixel, requiring a geometry shader to be employed. In this work, we borrow SH functions from irradiance environment maps [Ramamoorthi and Hanrahan, 2001] to encode geometric details while reducing the number of texture fetches needed.

3 PROBLEM STATEMENT

Given a high-poly model with known textures and a target low-poly model with an initial UV parameterization but without any texture, our goal is to optimize the UV mesh and textures of the low-poly, such that its expected visual appearance under all view directions is as close to the high-poly as possible. One key difference from conventional texture baking lies in the fact that we jointly optimize several new parameters related to texture mapping. This section briefly reviews the forward rendering procedure and introduces related notations.

Texture Mapping. The low-poly mesh $M \triangleq (V, E)$ is composed of a set of vertices $V \subset \mathbb{R}^3$ connected into a graph via edges E representing a triangulation. M is cut open along selected seam boundaries into a set of local charts that can be embedded into $\mathbb{R}^2$. The low-poly mesh is mapped through a UV parameterization function ψ to a planar mesh: $M_P \triangleq (U, E_{\text{uv}})$ with $U \subset \mathbb{R}^2$. Here ψ is typically chosen as a bijective function such that $\psi(V) = U$ and ψ is affine within each triangle.

Rasterization Pipeline. Following the definition introduced by NVDiffRast [Laine et al., 2020], the final color $I(p)$ of the pixel at screen coordinates $p = (x, y)$ in the forward rendering process can be written in the following form:

$$I(p) = \text{filter}\big(\text{shade}(d, C_p, N_p, \cdots)\big)(p), \tag{1}$$

where the filter is used to sample the color at the pixel center p from the continuous screen coordinates $p = (x, y)$ for antialiasing. The shading function is generic, and can be used for arbitrary rendering. It can takes arbitrary inputs, such as view direction d, color C_p, normal N_p at p. For our results, we use Lambertian shading:

$$\text{shade}(d, C_p, N_p) \triangleq (d \cdot N_p)\, C_p. \tag{2}$$

In the rasterization-based rendering pipeline, any visible 3D point v can be rasterized on the 2D screen space along the view direction d via a rasterization function R such that $p = R(v, d)$. Assuming R is locally invertible, given a view direction $d \in \text{SO}(3)$, each screen-space pixel p can be associated with a point $v = R^{-1}(p, d) \in \mathbb{R}^3$ on the input mesh. Given the screen coordinate p, the visible surface color C_p is defined as:

$$C_p \triangleq \text{fetch}(\psi(R^{-1}(p, d)), T_C) \tag{3}$$

where T_C is the diffuse texture, and the fetch function gets the texture data by interpolating neighboring texels. Similarly, we have $N_p \triangleq \text{fetch}(\psi(R^{-1}(p, d)), T_N)$ with T_N being the texture for the normal map. While we primarily demonstrate Lambertian shading in most experiments, our approach is able to handle other BSDF models. We demonstrate in Fig. 14 that we can optimize a mesh with specular texture under random lighting or with parallax mapping.

Differentiable Rendering. Recently, NVDiffmodeling [Hasselgren et al., 2021] proposed to resolve texture ambiguity during texture baking in a least-square sense. Differentiable rendering identifies T_C and T_N as the minimizer of the following expected visual similarity

metric among all view directions:

$$\underset{T_C,T_N}{\operatorname{argmin}} \underset{d\in\mathrm{SO}(3)}{\mathbb{E}} \left[\int_S \mathcal{L}_{\mathrm{diff}}(p,d)dp\right]$$
$$\mathcal{L}_{\mathrm{diff}}(p,d) \triangleq \|I(p,d,T_C,T_N) - I(p,d,T'_C,T'_N)\|,$$

where S is the screen space and T'_C, T'_N are the textures of the ground truth high-poly. Here we slightly abuse notation and denote $I(p,d,T'_C,T'_N)$ as the rendering function of the high-poly. We also stipulate that, if a 3D point v is occluded, then $p = R(v,d)$ is undefined and $I(p,d,T_C,T_N) = I(p,d,T'_C,T'_N)$. However, all existing differentiable rendering frameworks only consider optimizing the texture while fixing the given UV parameterization. As a potential drawback of such an approach, a large number of texels could be wasted on small or unimportant portions.

4 JOINT OPTIMIZATION FRAMEWORK

Our method differs from existing approaches in two aspects. First, we unify two avenues of prior research, distortion-minimizing [Rabinovich et al., 2017, Smith and Schaefer, 2015] and content-aware [Sander et al., 2002, Sun et al., 2022, Tewari et al., 2004] UV parameterization, using differentiable rendering [Hasselgren et al., 2021]. Second, we propose to optimize a view-parallax mapping [Kaneko et al., 2001] to compensate for potentially large geometric differences between the two meshes.

4.1 Objective Function

It has been shown in prior works [Rabinovich et al., 2017, Sander et al., 2002] that UV parameterization ψ plays an important role in high-quality texture baking. There are two lines of research that use different criteria of a "good" ψ. First, distortion-minimization algorithms such as [Rabinovich et al., 2017] assume that an ideal ψ should be as isometric as possible, i.e., $\nabla\psi$ should be as close to the identity as possible. Follow-up work [Jiang et al., 2017, Su et al., 2020] further ensures ψ is globally bijective, preventing different 3D points from being ambiguously mapped to the same UV coordinate. The other line of research, as in [Sander et al., 2002, Sun et al., 2022], proposes to make ψ content-aware. In doing so, an ideal ψ should assign larger areas to T_C with "richer" color information. However, the area of content-aware texture baking has received less attention than the isometric approach due to the inherent difficulty in measuring the texture information entropy. With the growth of differentiable rendering, measures of visual similarity, $\mathcal{L}_{\mathrm{diff}}$, are ideal candidates for content complexity measures, so we propose to jointly optimize UV coordinates U and texture color/normal maps T_C, T_N. Our augmented optimization problem takes the following form:

$$\underset{T_N,T_C,U}{\operatorname{argmin}} \underset{d\in\mathrm{SO}(3)}{\mathbb{E}} [\mathcal{L}_{\mathrm{diff}}] + \lambda_{\mathrm{I}\circ\mathrm{UV}} \underset{d\in\mathrm{SO}(3)}{\mathbb{E}} [\mathcal{L}_{\mathrm{I}\circ\mathrm{UV}}]$$
$$\text{s.t. } \psi \text{ is bijective,}$$

where $\mathcal{L}_{\mathrm{diff}}$ plays a guiding role that provides content to T_N, T_C, and $\mathcal{L}_{\mathrm{I}\circ\mathrm{UV}}$ makes UV coordinates U content aware. $\lambda_{\mathrm{I}\circ\mathrm{UV}}$ denotes a weight coefficient.

A priori, one might assume the image loss, $\mathcal{L}_{\mathrm{diff}}$, alone could recover content-aware UV coordinates, but we find that it does not provide sufficient gradient information to do so, as illustrated in Fig. 2 and ablated in 5.3. Thus we propose a multiplicative, content-aware image-based symmetric Dirichlet energy $\mathcal{L}_{\mathrm{I}\circ\mathrm{UV}}$ formulated as:

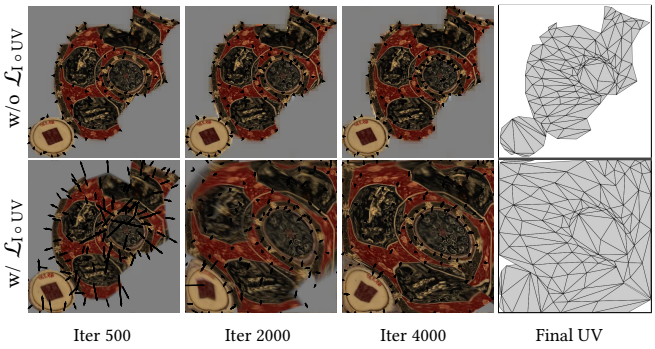


Fig. 2. As compared with our formulation (bottom row), the optimization without the content-aware term (top row) cannot provide sufficient gradient information to make the optimized texture content-aware. Arrows indicate gradient magnitude and direction.

$$\mathcal{L}_{\mathrm{I}\circ\mathrm{UV}}(d) \triangleq \int_M (\|\nabla\psi(v)\|^2 + \|\nabla\psi^{-1}(v)\|^2)\mathcal{L}_{\mathrm{diff}}dv,$$

where we adopt the convention that $\mathcal{L}_{\mathrm{diff}} = 0$ for invisible, occluded v. $\mathcal{L}_{\mathrm{diff}}$ within $\mathcal{L}_{\mathrm{I}\circ\mathrm{UV}}$ is a fixed, per-triangle image loss and acts as a weight for the symmetric Dirichlet Energy, $\|\nabla\psi(v)\|^2 + \|\nabla\psi^{-1}(v)\|^2$. The symmetric Dirichlet energy [Smith and Schaefer, 2015] penalizes deviations of $\nabla\psi$ from the identity and helps prevent ψ from producing local inversions. In practice, the integral over the mesh surface M is a summation of triangles if linear shape functions are used in discretization, and v can be any point on M. Note that ψ^{-1} is well-defined as we require ψ to be bijective. The original symmetric Dirichlet energy is not content-aware by itself. By multiplying with $\mathcal{L}_{\mathrm{diff}}$, $\mathcal{L}_{\mathrm{I}\circ\mathrm{UV}}$ will impose a higher distortion penalty if the surface point v induces a larger visual dissimilarity. Thus, $\mathcal{L}_{\mathrm{I}\circ\mathrm{UV}}$ becomes a content-aware UV mapping regularizer. We note that image loss is a per-pixel measure, whereas the symmetric Dirichlet energy is a per-triangle measure. In order to unify the two, we evaluate $\mathcal{L}_{\mathrm{I}\circ\mathrm{UV}}$ as a summation over triangles, where for each triangle, we compute a screen-space measure of the integral of $\mathcal{L}_{\mathrm{diff}}$. The visual similarity loss contributed by the given triangle is computed using rasterization, as in NVDiffmodeling [Hasselgren et al., 2021].

Note that if some v is occluded from d, then $\mathcal{L}_{\mathrm{diff}} = 0$, which will cause the triangle containing v to easily flip or become degenerate. To avoid this degenerate solution, we add the original symmetric Dirichlet energy $\mathcal{L}_{\mathrm{uv}}$ with the weight λ_{uv} to our objective to bias towards a distortion-minimizing ψ:

$$\mathcal{L}_{\mathrm{uv}} \triangleq \int_M \|\nabla\psi(v)\|^2 + \|\nabla\psi^{-1}(v)\|^2 dv.$$

Empirically, we find $\mathcal{L}_{\mathrm{I}\circ\mathrm{UV}}$ and $\mathcal{L}_{\mathrm{uv}}$ useful in different scenarios, where $\mathcal{L}_{\mathrm{I}\circ\mathrm{UV}}$ increases quality in a content-aware sense, while $\mathcal{L}_{\mathrm{uv}}$ ensures a minimum baseline isometric quality for ψ. Because geometric distortion is less important, we typically choose $\lambda_{\mathrm{I}\circ\mathrm{UV}}$ much larger than λ_{uv}.

Finally, to ensure that ψ is bijective, we need the UV mesh M_P to be free of intersections, for which there have been two approaches.

The scaffolding approaches [Jiang et al., 2017] triangulate the ambient space of M_P in the UV space and transforms the constraint into locally injective symmetric Dirichlet regularizations for both internal and ambient triangles. Alternatively, primal interior point method such as the incremental potential function [Li et al., 2021] formulates a barrier function bounding the distance between any $v_0, v_1 \in \partial M_P$ away from zero. We adopt the latter approach and transform the constraint into the following barrier term:

$$\mathcal{L}_{\text{IPC}} \triangleq -\sum_{\substack{v_a \neq v_b, v_a \neq v_c \\ v_a, v_b, v_c \in \partial T \cup \partial M_P \\ <v_b, v_c> \in E}} \text{clog}(\text{dist}(v_a, v_b, v_c))$$

$$\text{clog}(x) \triangleq \max(-(x - d_0)^2 \log(x/d_0), 0),$$

where T represents the rectangle texture space, $\mathcal{L}_{\text{IPC}}$ is a sum of the barrier distance function, dist($\bullet$), over all the vertex-edge pairs on $\partial T \cup \partial M_P$ and d_0 is an activation distance for efficiently pruning faraway geometric pairs from computation (we refer readers to [Li et al., 2021] for more details). We avoid the intersection between ∂M_P and ∂T since we need to constrain the UV mesh to remain inside the texture space T. The IPC energy allows us to impose bijectivity without the additional memory and computational cost of storing and re-triangulating a scaffolding mesh. Finally, we add a distance d_0, which is used to separate different charts by a distance of several texel widths, ensuring enough texels for constructing independent interpolation stencils for neighboring charts. This reduces seam artifacts on the resulting model.

4.2 Spherical Harmonic Parallax Mapping

When geometric discrepancies between the low-poly and the high-poly mesh are large, optimizing UV parameterization may lead to large UV regions being allocated to mismatched surfaces. This exacerbates blurriness in regions with high geometric differences, leading to an even worse final quality. To address this issue, we built *SH parallax mapping* to reduce the influence of geometric discrepancy, where the texture mapping function is formulated as the following operation:

$$C(v) = \text{fetch}\Big(\psi(v) + \frac{\text{fetch}(\psi(v), T_D)}{d_{\text{tangent(z)}}} \phi_{\text{SH}}\big(\psi(v), d\big), T_C\Big), \quad (4)$$

Where $d_{\text{tangent(z)}}$ is the value in the local normal axis when projecting the view direction in the coordinate system defined by the tangent, bitangent, and normal of a triangle. T_D is a scalar depth map, and the SH-based UV coordinate shift function ϕ_{SH} of order N is defined as:

$$\phi_{\text{SH}}(t, d) = \sum_{\ell=0}^{N} \sum_{m=-\ell}^{\ell} \text{fetch}(t, T_{\ell,m}) Y_\ell^m(d), \quad (5)$$

where t is the UV coordinates, Y_ℓ^m is the real Legendre polynomial of order ℓ, and $T_{\ell,m}$ is the texture storing the corresponding coefficients. Note that all coefficients are stored in one multi-channel texture so that a single texture lookup fetches all the coefficients. Our UV coordinate bias function unlocks a larger solution space, leading to higher visual similarity, and the cost of deploying equation (5) in a forward renderer is marginal using low-order SH functions. Prior work utilizing SH in irradiance mapping [Ramamoorthi and Hanrahan, 2001] directly computes the coefficients, but we optimize coefficients through auto-differentiation of the forward rendering pass with integrated SH parallax mapping.

4.3 Vertex Displacements Optimization

In addition to using parallax mapping, we found that constrained optimization of low-poly mesh vertices improves the silhouette mismatch between the low-poly and high-poly meshes. As we decrease the level of details, the silhouette mismatch between the high-poly and low-poly deteriorates further, which cannot be mitigated by any texture-based approach. To tackle this issue, we additionally optimize vertex displacements. Different from prior differentiable rendering works that perturb the vertex coordinates permanently at each iteration using a Laplacian smoothness term [Hasselgren et al., 2021, Luan et al., 2021, Nicolet et al., 2021], which may cause our low-poly meshes to deform in ways that do not minimize silhouette differences, we allow low-poly vertices to be perturbed only along their normal directions fetched from the normal map, and the amount of perturbations are additional optimization variables denoted as $\delta \in \mathbb{R}$. In other words, we only apply the displacements of our low-poly after the optimization procedure, moving $v_i \in V$ to $v_i + \delta_i n_i$, where all δ_i are assembled in a vector ΔV. In Fig. 13 and by comparing the columns Ours_{V} and $\text{NV}^\star$ of Table 11 in the Appendix, we demonstrate the quality degradation of prior works using Laplacian smoothing, and empirically show that our vertex optimization strategy leads to higher-quality results.

4.4 Optimization Technique

Put together, our optimization problem takes the following unconstrained form:

$$\underset{T_N, T_C, T_D, T_{l,m}, \Delta V, U}{\text{argmin}} \quad \mathcal{L} \triangleq \mathbb{E}_{d \in \text{SO}(3)} \left[\mathcal{L}_{\text{diff}}\right] + \lambda_{\text{uv}} \mathcal{L}_{\text{uv}} + \lambda_{\text{I} \circ \text{UV}} \mathbb{E}_{d \in \text{SO}(3)} \left[\mathcal{L}_{\text{I} \circ \text{UV}}\right] + \lambda_{\text{IPC}} \mathcal{L}_{\text{IPC}}, \quad (6)$$

where $\lambda_\bullet$ denotes a weight coefficient for the corresponding energy term. Unfortunately, the original Adam optimizer [Kingma and Ba, 2015] in the differentiable rendering framework [Hasselgren et al., 2021, Laine et al., 2020] does not work for our problem for two reasons. First, UV optimization involves non-Lipschitz barrier energy with a non-trivial feasible domain which, once exited is hard to return to. Using a constant or shrinking learning rate as in [Hasselgren et al., 2021, Laine et al., 2020] could potentially leave the feasible domain of these functions. Second, some functions are much more costly to compute than others. For example, evaluating the $\mathcal{L}_{\text{diff}}$ term only involves efficient rasterization-based rendering, but the $\mathcal{L}_{\text{IPC}}$ term involves costly geometric proximity querying, which scales with the number of vertices. To overcome these difficulties, we propose a feasibility-guaranteed alternating multi-stage optimizer, which is described below.

Line-Search Safety Guarantee. The unconstrained objective function is non-Lipschitz due to the barrier terms $\mathcal{L}_{\text{uv}}$ and $\mathcal{L}_{\text{IPC}}$, ensuring the UV mapping function ψ is bijective and M_P has no-flipped triangles. Unlike standard differentiable rendering framework that uses stochastic gradient descend for Lipschitz functions, we need to resort to strict line-search to ensure $\mathcal{L}_{\text{uv}}$ and $\mathcal{L}_{\text{IPC}}$ stay in their

feasible domain. We follow prior work [Li et al., 2020, Rabinovich et al., 2017] that first estimates the largest step size α_0 and then find a suitable final step size α satisfying the first Wolfe condition. To keep $\mathcal{L}_{\text{uv}}$ and $\mathcal{L}_{\text{I}\circ\text{UV}}$ well-defined, we compute α_0 to be the largest value such that the minimal singular value $\sigma_{\min}(\nabla\psi) \geq 0$, or the point at which triangle inversion would occur. This can be computed by solving a quadratic equation per triangle in parallel. To ensure that M_P and T remain intersection-free, we further compute α_1, the time-of-impact where a vertex-edge pair on $\partial T \cup \partial M_P$ intersects. Then, we define the final $\alpha \triangleq \min(\alpha_0, \alpha_1)$.

UV and Texture Initialization. Since our method requires a bijective initialization for the UV coordinates U, we employ *xatlas* [Young, 2017] to generate an initial bijective mapping. We also perform a normalization step of the initial UV parameterization, such that it is centered and fills the 2D bounding box $[0.01, 0.99]^2$, to ensure that the UV has space to deform. We initialize texture colors as gray, tangent normal values to $[0, 0, 1]$ which corresponds to using the mesh's face normal, and Spherical Harmonic coefficients to a small uniformly random value.

Mesh Normalization. Our method requires both meshes to be well-aligned. We assume the two meshes are both contained within the bounding box of the input high-poly mesh, so we uniformly transform the high-poly to be contained in the unit box $[-1, 1]^3$. We then apply the same transformation to the low-poly mesh. We find this strategy enough to yield high-quality textures. We have also experimented with using independent transformation to the two meshes, but this can oftentimes lead to misaligned silhouettes.

Alternating Optimization. After initialization, we alternate updating the textures and UV coordinates to achieve joint optimality. We first fix the UV coordinates U and update all the textures, $T_N, T_C, T_D, T_{l,m}, \Delta V$, using k_0 iterations of Adam [Kingma and Ba, 2015] with batch-sampled approximations of the rendering loss over view directions $d \in \text{SO}(3)$. This gives a good initial estimate of the spatially varying weight of the symmetric Dirichlet texture regularization term $\mathcal{L}_{\text{I}\circ\text{UV}}$. We then perform another k_1 iterations, with each iteration having f steps of the Adam optimizer for the textures and one iteration of L-BFGS [Liu and Nocedal, 1989] with line-search for UV coordinates by fixing the per-triangle image loss. When updating $\mathcal{L}_{\text{I}\circ\text{UV}}$, we do not minimize the final rendering loss and do not update the texture, and focus primarily on updating the UV parameterization. The gradient of the spatially adaptive symmetric Dirichlet term takes the following form:

$$\nabla\mathcal{L}_{\text{I}\circ\text{UV}} = \int_M \nabla\left[\|\nabla\psi(v)\|^2 + \|\nabla\psi^{-1}(v)\|^2\right]\mathcal{L}_{\text{diff}}dv + \int_M (\|\nabla\psi(v)\|^2 + \|\nabla\psi^{-1}(v)\|^2)\nabla\mathcal{L}_{\text{diff}}dv.$$

Although this alternating scheme is not theoretically justified, it achieves an ideal balance between efficacy and accuracy in practice. We have experimented with exact line-search scheme, where $\mathcal{L}_{\text{diff}}$ in $\mathcal{L}_{\text{I}\circ\text{UV}}$ are updated at each step of line search. However, this exact scheme leads to twice more expensive iteration cost, with minimal noticeable improvements in results. Our alternating optimization is terminated after a fixed iteration budget is reached or the gradient of the UV vertices and the change in diffuse texture is below a certain threshold. After this, the texture and UV coordinates are approximately jointly optimal. Finally, we use another k_2 iterations of Adam optimization to fine-tune textures, which can also be terminated early if textures have converged. Our final optimization is summarized in Algorithm 1.

Algorithm 1 Alternative Optimizer for Equation 6

Input: $k_0, f, k_1, k_2, k_3, \epsilon_0, \epsilon_1$, initial U and V
Output: $T_N, T_C, T_D, T_{l,m}, U, V$
Initialize T_N by setting all normals to $(0, 0, 1)$
Initialize T_C by setting all colors to gray $(0.5, 0.5, 0.5)$
Initialize T_D as 0
Initialize $T_{l,m} \sim \mathcal{U}(-\epsilon_0, \epsilon_0)$
Use k_0 iterations of Adam to update $T_N, T_C, T_D, T_{l,m}, \Delta V$
while $\Delta\|T_N\|, \Delta\|T_C\| > \epsilon_1$ and iters $< k_1$ **do**
    Use f iterations of Adam to update $T_N, T_C, T_D, T_{l,m}, \Delta V$
    Use 1 iteration of L-BFGS with line search to update U
Use k_2 iterations of Adam to fine-tune $T_N, T_C, T_D, T_{l,m}, \Delta V$
Set $V = V + \Delta V$
Return $T_N, T_C, T_D, T_{l,m}, U, V$

4.5 Implementation Details

We implement our algorithm in Python, using NVDiffrast [Laine et al., 2020] with CUDA for numerical computations and differentiable rendering. Details related to the implementation of our algorithms are described below.

Texture Padding and Scaling. We use d_0 in $\mathcal{L}_{\text{IPC}}$ to impose a padding distance between neighboring charts, which ensures charts are far enough apart for texture interpolation. We set $d_0 = 5\Delta t$, where Δt is the texel width. Further, we note that the low-poly and high-poly can have drastically different UV mapping functions ψ. Intuitively, our method performs the best when the two mapping functions ψ contain the same number of texels. To approximately enforce this constraint, we scale the UV coordinates relative to texture resolution to ensure the low-poly and high-poly take up the same number of texels in the UV space and we do not recalculate the scaling during optimization.

Numerical Stability. Our method relies on automatic differentiation for computing the gradients, which may introduce numerical issues in several places. As compared to analytic gradients which are used in [Li et al., 2020, Rabinovich et al., 2017], we find automatic differentiation requires additional treatment for numerical stability of non-Lipschitz barrier energy terms. First, for the symmetric Dirichlet function, we use a log exponential trick and let $\psi(v) = \exp(\log(\psi(v)))$. The log-exponential trick is known to better utilize the limited floating point precision and prevent catastrophic gradient explosion when $\phi(v)$ takes on values very close to zero. Second, we find that boundary computation can quickly consume floating point precision, as edges and points get extremely close. To remedy this, we propose a mixed precision scheme, computing all gradients using 32-bit floats to save hardware bandwidths while

switching to 64-bit floats to compute boundary edge-vertex overlaps, which do not require gradients.

Parameter Settings. For the number of iterations, k_0, f, k_1, k_2, we set a budget of $k_0 = 500$, $k_1 = 800$, $f = 5$, and $k_2 = 500$. We find that 500 iterations are often enough to approximate the output of NVDiffmodeling initially, and it is enough to clean up any artifacts caused by shifting UV coordinates. To initialize Algorithm 1, we set SH coefficients uniformly by setting $\epsilon_0 = 1 \times 10^{-5}$, as we do initially do not want parallax mapping. We also set $\epsilon_1 = 1 \times 10^{-6}$. For weights on loss terms, we set $\lambda_{\text{UV}} = 1 \times 10^{-2}$, $\lambda_{\text{IPC}} = 1$, and $\lambda_{\text{I} \circ \text{UV}} = 5$. We use 3rd-order SHs as it is a good balance between efficiency and representability.

Fig. 3. Textured high-poly models used in our dataset (not to scale) from [Sketchfab, 2022] Each model has a single texture atlas of size 2048×2048, with varying geometric and texture complexities. Attributions for models are provided in the code repository, and are licensed under CC-Attribution.

5 EXPERIMENTS

We conduct various experiments to validate our proposed approach and demonstrate its robustness. We run all our experiments on a workstation with an 8-core Intel processor clocked at 2.5 GHz, 64 GB of memory, and a single NVIDIA Titan XP GPU with 12 GB of RAM. All computations are performed on the GPU and in a single CPU thread. We evaluate the appearance quality of the generated textures by sampling a set of 48 camera views uniformly spaced on a sphere of radius 2 centered around the model, which is resized into the unit box $[-1, 1]^2$, and compare the visual similarity using both PSNR and MS-SSIM [Wang et al., 2003], for both metrics higher values are better. For conciseness, variations of our approach are denoted by a combination of *UV*, *SH*, and *V*, which stand for UV optimization, SH parallax mapping, and vertex displacement, respectively. We always optimize the diffuse color and normal textures for all the variations of our method. For example, Ours_{UV} means we optimize T_C, T_N, and U, Ours_{SH} optimizes T_C, T_N, T_D, and $T_{l,m}$, Ours_{V} optimizes T_C, T_N, and ΔV, while $\text{Ours}_{\text{UV, SH}}$ optimizes T_C, T_N, U, T_D, and $T_{l,m}$, etc. *Ours* indicates that we optimize all variables as described in Algorithm 1.

In the following sections, we will describe the dataset preparation, evaluation of our approach, and ablation studies of our algorithm. While only selected images are provided in the main text, for the curious reader, we release our dataset and the generated results for each rendered model in the figures on GitHub at https://anonymous.4open.science/r/tex_baking_supplementary/, which includes the high-poly and low-poly meshes, the associated diffuse color and normal textures, as well as depth map, vertex displacement vector and spherical harmonic coefficients in PyTorch ".pt" format. We also include a video for each result showing the baked appearance.

5.1 Dataset

To comprehensively evaluate our approach, we collect a dataset of 3D models found in the wild and generate a set of LOD models. We first obtain high-poly meshes by collecting a set of 3D models from Sketchfab [Sketchfab, 2022] and then convert each of them into a mesh with a single UV atlas $\in [0, 1]^2$ and a 2048^2 texture using xatlas [Young, 2017] for parameterization and Blender [Blender Online Community, 2018] for texture transfer. For each high-poly mesh, we then produce 3 low-poly meshes with varying geometric differences from the reference mesh using Simplygon [Donya Labs AB, 2022].

When generating meshes using Simplygon, we set the screen-size parameters to 50, 100, and 300, corresponding to large, moderate, and small geometry differences, denoted as LOD-3, LOD-2, and LOD-1, respectively. Screen-size roughly corresponds to the mesh resolution, where lower values will lead to coarser approximations with fewer vertices and faces. We use the UV parameterization that Simplygon generates alongside the produced meshes. While our set of high-poly meshes has a broad variety of shapes, geometric complexities, and texture qualities, the low-polys have dramatically different geometry and topology properties from their high-poly counterparts, see Fig. 4, Fig. 5, Table 5 for detailed statistics and Fig. 3 for visual renderings. Since the original models are from online, we release attributions to the original artists but not the original models themselves.

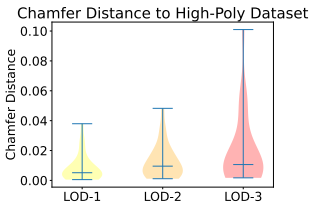
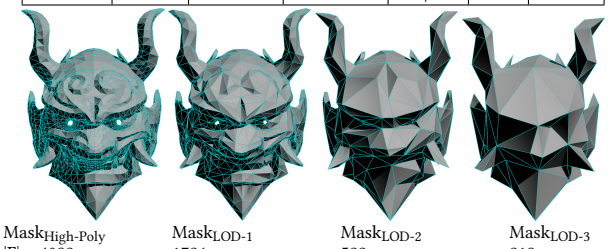


Fig. 4. Chamfer distance statistics between the LOD datasets and the high-poly meshes. For the computation of Chamfer distance, we center each mesh at the origin and resize it to fit in the unit box.

5.2 Evaluation

Comparisons with Competing Texture Bakers. We compare against the open-source ray-casting approach in Blender [Blender Online Community, 2018], the texture baker in a flagship 3D game engine,

Dataset	#Comp.	#Vertices	#Faces	Manifold	Genus	Holes
High-Poly	8/225/2k	502/8k/70k	723/9k/0.1M	5/26	0/0/0	0/29/174
LOD-1	1/41/312	244/1k/5k	306/1k/5k	24/26	0/3/39	0
LOD-2	1/19/82	43/354/1k	64/406/1k	26/26	0/2/28	0
LOD-3	1/6/40	28/114/349	32/137/368	26/26	0/0/12	0



$\text{Mask}_{\text{High-Poly}}$	$\text{Mask}_{\text{LOD-1}}$	$\text{Mask}_{\text{LOD-2}}$	$\text{Mask}_{\text{LOD-3}}$
$\|F\| = 4082$	1726	522	218
Chamfer	0.0031	0.0062	0.0083

Fig. 5. Statistics of our dataset: (top) topology properties of the high-poly and different LODs of the dataset, (bottom) visual rendering of LOD of the Mask model in our dataset. Within the table, we show the minimum/median/maximum of different properties of our dataset, including the number of components (#Comp.), the number of vertices and faces, the number of manifold meshes/the total number of models, the genus, and the number of holes contained in the models.

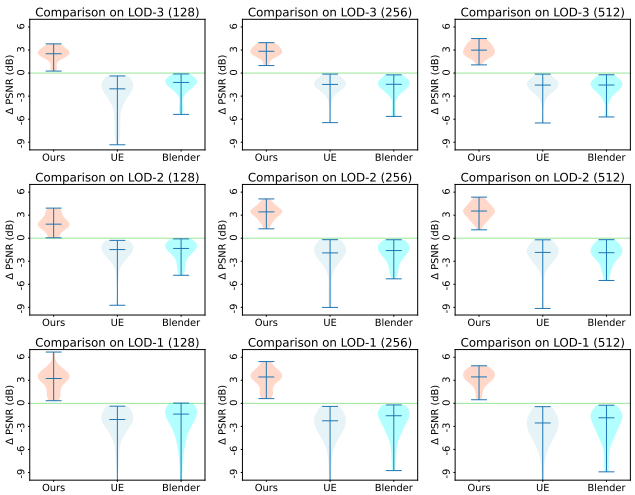


Fig. 6. We compare our approach with UV optimization, SH parallax, and vertex displacements (Red), relative to NVDiffModeling (Green Line), and include UE (Dark Blue) and Blender (Teal) for reference. We find that across all texture resolutions and mesh resolutions, our approach outperforms NVDiffModeling significantly. The horizontal green line denotes NVDiffModeling, which we use as the baseline, and the middle marker of the violin plot denotes the median improvement.

Unreal Engine (UE) [Epic Games, 2022], and the differentiable rendering approach NVDiffModeling [Hasselgren et al., 2021]. For all approaches, including ours, we use the same configuration and settings for all meshes. For Blender, we use the default setting for cage extrusion without additional resizing. For UE, we set the ray offset distance from the surface to be large (1×10^5), as their approach does not automatically rescale meshes before texture baking and operates in world space.

We batch process the competing approaches and our method on the three LOD datasets generated from Simplygon and produce new diffuse and normal textures with resolutions at 512^2, 256^2, and 128^2. As shown in Fig. 6. Our approach is noticeably better by 2-4 PSNR than competing approaches across all datasets. Our approach significantly outperforms traditional raycasting, being up to 10 PSNR better on some meshes and consistently better on every mesh in our dataset independent of texture resolution.

On the LOD-3 dataset, where the low- and high-polys have high geometric discrepancies, we achieve around 3 PSNR improvements compared to NVDiffModeling. On LOD-2 with lower geometric discrepancies, our approach performs even better, achieving around 3.25 PSNR than NVDiffModeling, except with a texture resolution of 128^2 where our performance is about 2 PSNR better. When the meshes are extremely similar on LOD-1, our approach consistently performs about 3.5 PSNR better than NVDiffModeling, independent of texture resolution. All the reported PSNR improvement is the median improvement over NVDiffModeling, which is outlier resilient and demonstrates a consistent increase. Across texture resolutions, we find our relative PSNR gain slightly decreases when the texture is tiny, e.g. at a resolution of 128^2, but increases as resolution increases. Intuitively, this is because all approaches need a minimal number of texels to encode the high-poly's appearance. Full statistics for each mesh in our dataset at each resolution are attached in Table 8, Table 9, and Table 10 of Section A in the Appendix.

We show some visual comparisons of baked models in Fig. 7. On a single model, we compare multiple levels of detail, alongside the ground truth model and competing methods, with close-up insets of different portions of each model. At LOD-1, our approach is similar to NVDiffModeling but is able to capture finer features around edge borders, as can be seen in the insets around the eyes. Blender also performs well, but UE has artifacts due to casting the ground onto the penguin and the penguin onto the ground. At LOD-2, our approach retains the visual acuity of LOD-1, whereas the other approaches become more blurred. Notably, around the eyes, UE and NVDiffModeling begin to blur, significantly degrading the quality. At the lowest LOD, LOD-3, our approach still appears sharp but has more errors around regions where the low-poly and high-poly mesh have begun to diverge significantly, which can be seen on the cracking ground which has started to vanish in NVDiffModeling. UE performs surprisingly well, appearing sharp on the eyes, but has artifacts since the feet are now a cylinder connected to the ground, which is extremely different as compared to the original model. As can be seen, across different LOD our approach preserves sharp features and remains faithful to the original model. Our advantages at low LOD as compared with NVDiffModeling are mainly due to our SH parallax mapping and vertex displacement. At higher LOD, our advantages are due to UV optimization.

We also examine different texture resolutions in Fig. 8 for LOD-2 of a simple geometric model with a complex texture. We can see that across all texture resolutions, ours maintains key details, whereas the other approaches appear blurry or pixelated. The optimized UV is highly efficient, taking up almost the entire 0-1 space, whereas the original UV does not utilize much texture space. At the lowest texture resolution, 128^2, our approach has some pixelation artifacts

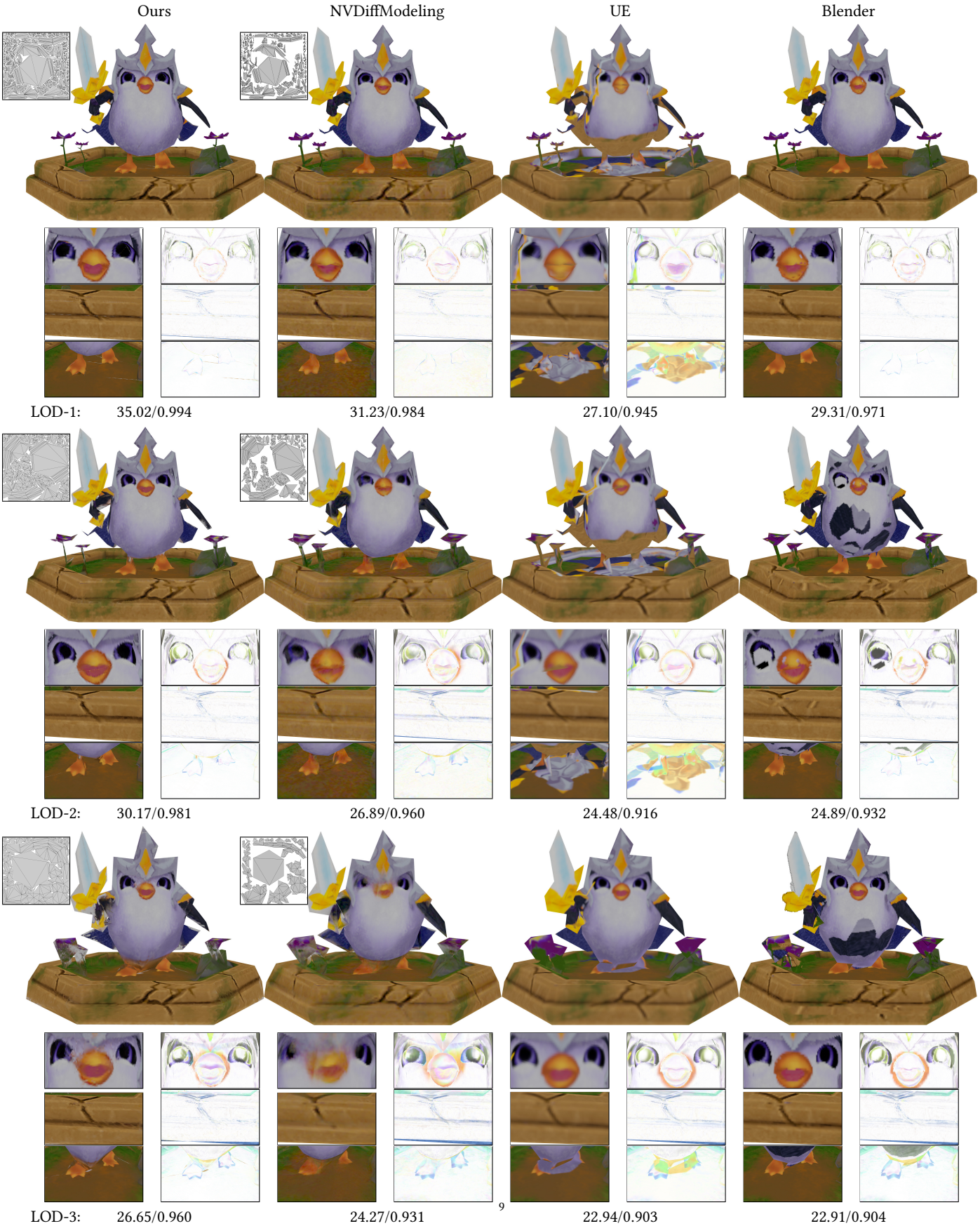


Fig. 7. Visualization of outputs of our texture baking approach on multiple levels of details of the Penguin model, where the number of faces for the high-poly, LOD-1, LOD-2, and LOD-3 are 18163, 2372, 664, and 292, respectively. Independent of LODs, our approach outperforms raycasting and differentiable rendering approaches, as shown by visual inspections and the corresponding PSNR/MS-SSIM values below each mesh. Note that UE and Blender have the same UV as NVDiffmodeling.

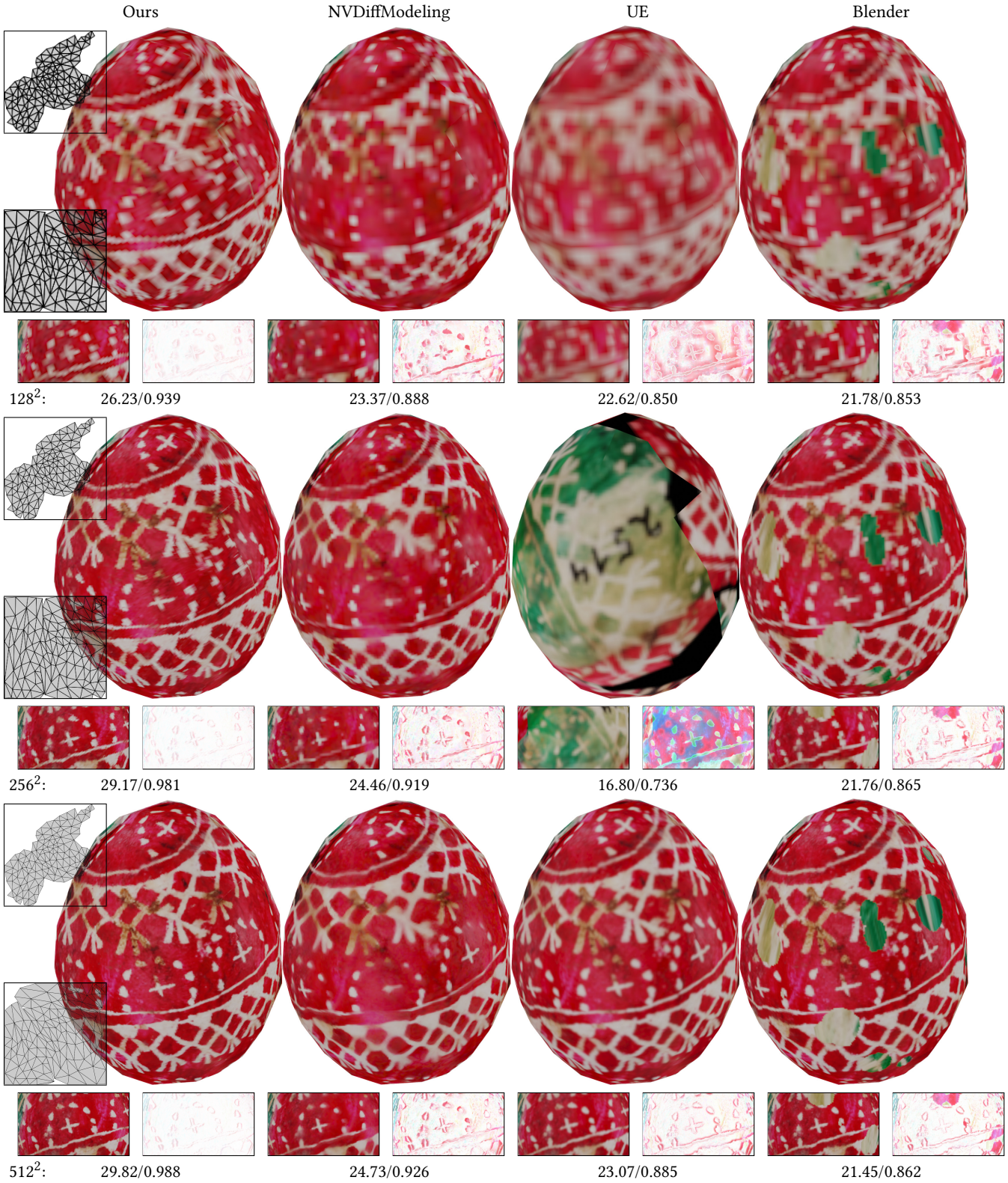


Fig. 8. Comparisons of the baked Egg LOD-2 models generated by raycasting, differentiable rendering, and our approaches at texture resolutions of 128, 256, and 512. In the left column we show the initial UV at the top, and our optimized UV at the bottom.

because there are insufficient texels to represent the original texture. Even so, our approach is able to recover some sharp features. NVDiffModeling loses many details, and some regions appear pixelated. UE blurs the features but maintains the general concept, whereas Blender instead leads to pixelization of features. At the middle texture resolution, 256^2, our approach reduces some of the blurrings from 128^2, capturing more precise details. NVDiffModeling begins to exhibit a higher quality but still misses many small details, and there is uneven blurring across the surface. UE and Blender both contain artifacts, but where Blender does not have artifacts it appears slightly pixelated. At the highest texture resolution, 512^2, our approach appears very sharp. NVDiffModeling is still unable to recover some details and appears blurry in multiple places. UE appears very sharp but has a high discrepancy against the ground truth. Blender, on regions without artifacts, appears very sharp and has good quality as well. Across all resolutions, we demonstrate that our approach has superior visual quality and captures more details.

By having methods to handle varying levels of mesh and texture resolutions, our approach is robust and maintains high quality even when the low- and high-polys are significantly different. A video clip for each generated result with a 360° comparison of the target and reference models for all the methods is provided in the anonymous github repo as mentioned earlier in this section.

Comparisons with Content-Aware UV Parameterizations. While existing content-aware UV optimization methods [Sander et al., 2002, Sun et al., 2022] cannot be used to bake textures between meshes with geometric differences, they can be employed to reduce the texture resolution for the same model. Specifically, we examine the application of generating lower resolutions of the textures for high-poly models for both 2D and 3D applications.

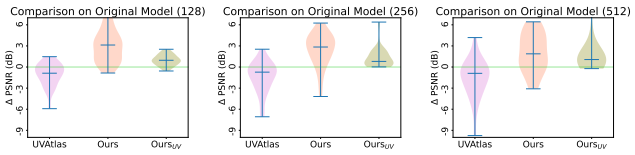


Fig. 9. We compare 4 methods: NVDiffModeling (green line), our method optimizing texture content, UV parameterization, SH and vertex displacement (Ours), our method without SH or vertex displacement (Ours$_{UV}$), and UVAtlas, where we first use UVAtlas [Sander et al., 2002] to generate a UV parameterization, and then using NVDiffModeling to generate texture contents. Our approaches (Ours and Ours$_{UV}$) achieve the best results over all comparisons on downsampled textures on the original model.

For 3D comparisons, we compare our method against a baseline of NVDiffModeling [Hasselgren et al., 2021] and Signal-Specialized Parameterization [Sander et al., 2002], where we first apply UVAtlas [Sander et al., 2002] to generate a UV parameterization, then bake textures using NVDiffModeling. To parameterize our approach while not using any signal-aware parameterizaton, we reparameterize each high-poly mesh using Blender's smart projection and set it such that it produces many charts with large margins, utilizing few texels. This often produces poor parameterizations but usually generates a bijective mapping. For those which are not bijective, we manually clean them, triangulating, removing degenerate faces, and reparameterizing them as necessary. As shown in Table 7 and Fig. 9, we find that UV optimization provides a performance boost and including SH and vertex displacement also helps reduce the visual differences, despite poor initialization as compared to NVDiffModeling, which uses Simplygon's parameterization. Surprisingly, we find that UVAtlas performs even *worse* than only NVDiffModeling, and we suspect this is because while UVAtlas may allocate relatively more texels to important regions, it does not use the full texture space, so the absolute number of texels is less than the number of texels used by the original parameterization. This can be seen in the output parameterizations as provided in the GitHub repo.

We also demonstrate the effectiveness of our method of compressing textures on 2D meshes, a common application in 2D and 2.5D games. For 2D models, it is trivial to generate a UV parameterization which is perfectly isometric to the mesh, but we posit that deforming the UV parameterization can increase visual quality for compressed textures. We simplify our approach by fixing an orthographic camera that can view the entire plane and do not optimize the normal map, SH parallax, or vertex displacements, as we are focused only on recovering a compressed texture on the plane. We also set $\mathcal{L}_{\text{UV}} = 1 \times 10^{-5}$. As shown in Table 6 and Fig. 10,

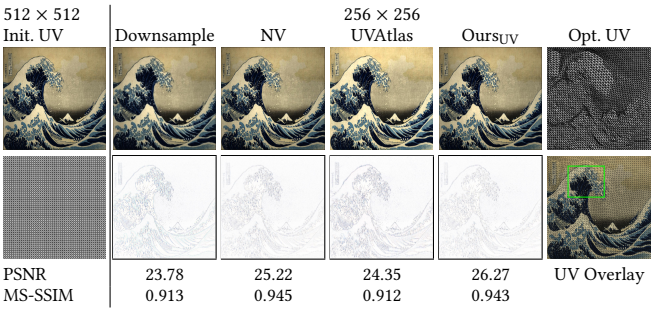


Fig. 10. Our method is able to compress textures on planar surfaces by giving regions with higher frequency detail a larger portion of UV space, such as the region highlighted by the green box on the rightmost image. In the bottom row, from left-to-right is the original UV, the pixel-wise difference between the reconstructed and the original, and the optimized UV overlaid on top of the original image.

our method is able to outperform direct bilinear downsampling, NVDiffModeling, and UVAtlas on a variety of images with different kinds, such as single icon images, paintings, pixel art, game textures, and maps, including those without high-frequency details, or those with extremely high information density. All the tested 2D data are provided in the aforementioned github repo. Similar to our approach on 3D meshes, across all 2D textures at varying target resolutions (three different resolutions are tested), our algorithm performs much better than UVAtlas, NVDiffModeling, and direct downsampling, achieving 1 dB, 3 dB, and 3.25 dB more on average over the 2D dataset at 64^2, 128^2, and 256^2 respectively.

Comparison with Other Parallax Mapping Variants. As an alternative to resolving view parallax, our framework can be used to optimize other parallax mapping variants, such as parallax mapping (Parallax) [Kaneko et al., 2001], parallax mapping with offset limiting (PMOL) [Welsh et al., 2004], steep parallax mapping

Table 1. A comparison of some existing approaches that handle parallax, in no particular order. Depth (h) and SH are evaluated at a specific texture UV coordinate, and v is the tangent space view direction transformed from the view direction d in the world space. We note that for approaches after the first two, it is possible to either include or not include the factor $\frac{1}{v_z}$. POM is based on SPM, but seeks to find a closer approximation to the correct position by using the linear interpolation between the last two steps.

Method	Formula	#Tex. Fetch
Parallax Mapping [Kaneko et al., 2001]	$\text{UV} + v_{\text{xy}} \frac{\text{h}}{v_z}$	1
PMOL [Welsh et al., 2004]	$\text{UV} + v_{\text{xy}}\text{h}$	1
SPM [McGuire and McGuire, 2005]	$\text{UV}_i + v_{\text{xy}} \frac{\text{h}}{k}, h > \frac{i}{k}, k \in \mathbb{Z}_+$	O(N)
POM [Tatarchuk, 2006]	SPM + Lerp	O(N)
Ours	$\text{UV} + \text{SH}(d) \frac{\text{h}}{v_z}$	1

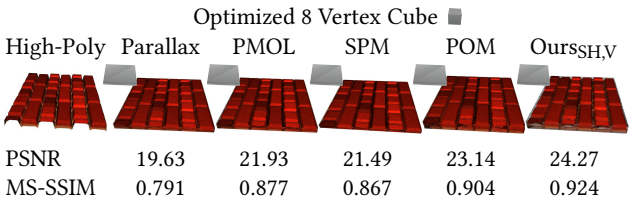


Fig. 11. We compare the baking results by optimizing different variants of parallax mapping on a brick wall, using a cube as the low-poly model. From left-to-right, Parallax Mapping [Kaneko et al., 2001], parallax mapping with offset limiting (PMOL) [Welsh et al., 2004], steep parallax mapping (SPM) [McGuire and McGuire, 2005], Parallax Occlusion Mapping (POM) [Tatarchuk, 2006], and our SH approach. Our proposed SH gives a convincing perception of depth, where the other variants do not appear as deep between bricks. We optimize each of the vertex positions of the cubes for all the variants, and note that the result of each approach is nearly identical.

(SPM) [McGuire and McGuire, 2005], and parallax occlusion mapping (POM) [Tatarchuk, 2006]. For SPM and POM, we multiply translation by a function of depth (depth/#layers), so that it can be optimized jointly alongside a depth map. Refer to Table 1 for different equations of each method. We use 3rd-order SH for our approach, 16 layers for SPM and POM, and a height scale of 1×10^{-2} for all parallax approaches. We use a minimum of 1000 iterations (k_1 in Algorithm 1) for optimizing Parallax, PMOL, and SH. We use 2500 iterations for SPM and POM. On the LOD-3 dataset, we demonstrate that our approach has the best visual quality for reconstructing textures amongst all the parallax variations, see Table 11. In Fig. 11, we also show the comparisons of our approach with other variants on a typical example, brick wall, where the high-poly has a complex geometry and the low-poly is a cube. Quantitatively, our approach achieves the best PSNR/MS-SSIM evaluations. Qualitatively, optimizing SH is able to capture occlusion, which is not recovered as clearly by optimizing other variants. In addition, our approach only requires a single texture fetch, whereas POM and SPM require multiple fetches for ray marching.

We also demonstrate how our approach compares visually on an example when the ground truth depth to a high-poly mesh is known in Fig. 12. Here, we apply different parallax variants to the same plane, and visualize the difference, and optimize our approach on the same plane. We produce a high-poly mesh by directly shifting vertices in the opposite direction of the normal by the given depth, and use the original plane to recover the input. In this case, our approach is still competitive with prior approaches, and provides a convincing perception of parallax.

Comparison with Displacement Mapping and Mesh Deformation. We compare our approach with two other approaches to reduce the geometric difference between the source and target meshes on the LOD-3 dataset. Specifically, we compare with 1) displacement mapping with 1 level of mesh subdivision that is optimized by NVDiffModeling, and 2) mesh deformation originally provided by NVDiffModeling. As illustrated in Table 11, among the three methods, our approach is competitive with 1 level of subdivision of the displacement mapping and performs much better than mesh deformation since NVDiffModeling employs Laplacian term that often leads to undesirable mesh deformations on most of the low-poly models. Note that the subdivision of the displacement mapping requires a significantly different pipeline in practice than our approach, including a geometry shader which is known to be slower than fragment or vertex shaders. Fig. 13 demonstrates a visual comparison of these approaches on the Knight model.

Extensions to Other Texture Maps. While we demonstrate that our approach works for generating diffuse and normal maps, we also demonstrate in Fig. 14 that it works for other kinds of texture maps, such as a specular map, which can be achieved by modifying the forward rendering equation to include a specular term. As demonstrated, any map can be generated as long as it is used in the forward rendering equation in a differentiable form. As another example, we have shown in Fig. 11 that we can optimize the depth map for traditional parallax mapping. We test optimizing a depth map on all models with various parallax approaches in Table 11.

Timings. We compare the timings of our approach with NVDiffModeling, and demonstrate the per-iteration cost of our approach, as shown in Table 2. First, we note that it is essentially cost-free to include vertex optimization on top of UV and parallax mapping optimizations. While NVDiffModeling is mostly independent of texture resolution, our approach's cost increases with texture size, due to larger memory usage for UV optimization. Our approach also requires more iterations for the UV parameterization to converge. We note that timings are always machine and implementation dependent, and since our approach is entirely implemented in Python, it does not have the implementation benefits and compiler optimizations that NVDiffModeling has, as it is mostly a wrapper around a CUDA library. We also omit comparisons with raycasting, because there is no "per iteration" cost for raycasting. Raycasting approaches are two orders of magnitude faster than any of the approaches listed though and are the most efficient.

Stage-wise Convergence. To justify the necessity of our three-stage pipeline, we plot the convergence history of $\mathcal{L}_{\text{diff}}$ and $\mathcal{L}_{\text{I}\circ\text{UV}}$ in Fig. 15, which is smoothed to filter out the noises. In these plots, the green and yellow vertical lines indicate the point where UV optimization starts and ends, respectively. Our first stage before the green line can significantly reduce the visual discrepancy and

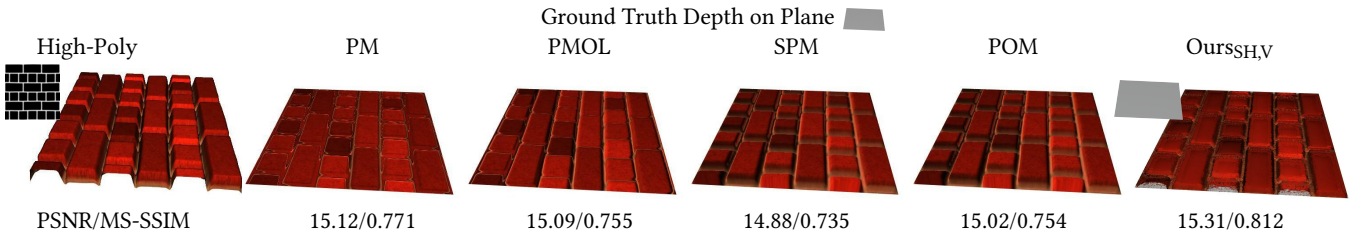


Fig. 12. We compare our approach against different parallax variants on the same textured mesh with a ground-truth depth map. Ours is competitive with the other approaches and has an illusion of depth. We do not show the geometry for each parallax variant, as we do not perform any optimization on those, only using the ground truth depth. We *do* optimize our own approach, as it does not work without optimization.

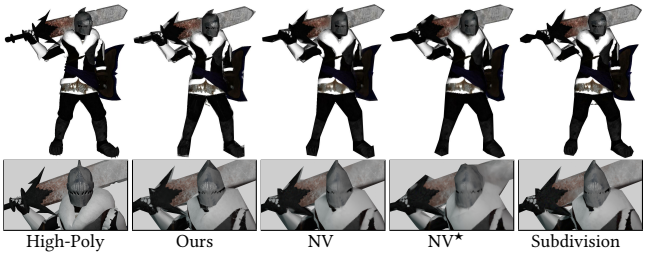


Fig. 13. We visualize the difference between our approach, NVDiffModeling (NV), NVDiffModeling with the vertex optimization enabled (NV★), and 1 level subdivision of the displacement mapping (subdivision) on the same low-poly mesh. We can see that our approach has much better visual quality than NV and NV★ where both blur features on the reconstructed model, and slightly better than subdivision, despite subdivision quadrupling the number of triangles.

Fig. 14. Our approach can also optimize any map that is used during forward rendering, such as the specular map. We visualize a wall-console with optimized specular (T_S), normal (T_N), diffuse (T_C), and depth maps (T_D). Any map can be optimized if it can be used in a differentiable forward rendering equation.

Table 2. We compare the time required to perform 1 iteration of optimization for each approach at different texture resolutions in seconds, as well as the number of iterations until convergence. †The number of iterations may change per scene and for each approach, so we enforce a minimum number of iterations.

Texture Baking Timing (wall-clock)			
	Method (Tex. Res)	ms/iter	Iters†
NV	NV (128)	245	1183
	NV (256)	242	1105
	NV (512)	244	1050
Ours	Ours$_{UV,SH}$ (128)	448	4504
	Ours$_{UV,SH}$ (256)	450	4504
	Ours$_{UV,SH}$ (512)	607	4504
	Ours (512)	608	4504
	Ours$_{SH,V}$ (512)	486	1000
	Ours$_{SH}$ (512)	507	1000

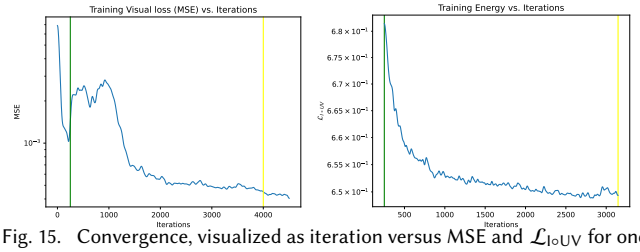


Fig. 15. Convergence, visualized as iteration versus MSE and $\mathcal{L}_{\text{lo}\cup\text{V}}$ for one model. UV optimization starts at the green line, and ends at the yellow line. We find this trend to be consistent across our dataset.

boostrap the follow up stages. When UV optimization starts, it contributes to another considerable decrease in $\mathcal{L}_{\text{diff}}$. Our third stage after the yellow line does not reduce the visual discrepancy as much, but it is needed to fine-tune the texture image and remove noises.

5.3 Ablation Studies

In this section, we evaluate the necessity and importance of various components in our method.

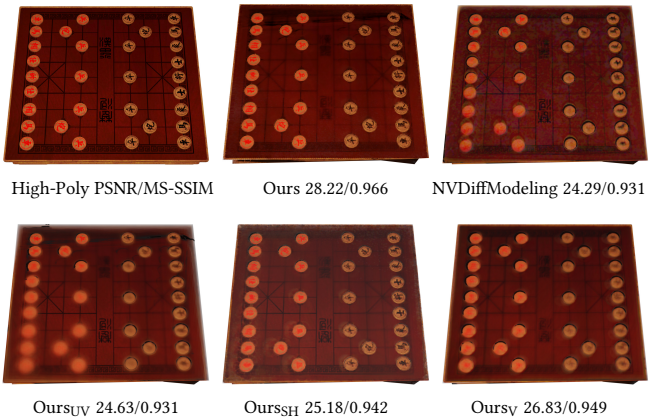


Fig. 16. We demonstrate the importance of each component (UV optimization, parallax mapping, and vertex displacement on a single model). Individually, each component of our approach helps minimize the visual difference between the low- and high-poly.

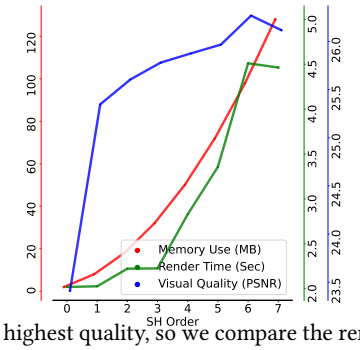
Vertex Displacement Optimization. In order to demonstrate the importance of our approach's vertex displacement, we compare our approach with and without in Table 8, Table 9, and Table 10. By including vertex displacement, our approach is able to perform better on most examples as compared to our approach with only SH and UV optimization. This is especially noticeable at LOD-3, as there is a more significant geometric difference, leading to a large increase for our approach.

Parallax Mapping. We demonstrate the importance of SH Parallax mapping in Table 11. Across all versions of parallax mapping, we demonstrate that texture baking quality is improved through the use of parallax mapping universally across all models. By examining SH Parallax compared with other parallax mapping techniques, we find their improvements comparable, but SH parallax has higher final quality. We find adding parallax mapping to differentiable rendering as important for final quality, as even alternative approaches such as standard parallax mapping [Kaneko et al., 2001] provide a consistent improvement.

UV Optimization. The importance of UV optimization is highlighted during texture compression, as SH parallax mapping and vertex displacements are not used during the optimization. In Fig. 10, we show the produced UV parameterization easily identifies the original image features, and demonstrate that UV optimization has a non-negligible impact on the visual quality.

The importance of each of the individual components can be visually observed on the chess model as shown in Fig. 16.

SH Function Order. For our approach it is not a-priori clear what order of SH provides good quality, as there is a tradeoff between universality versus memory requirements and computational cost as shown in the inset. We would like to keep the number of coefficients as low as possible while providing the highest quality, so we compare the rendering quality of varying the order of SH coefficients versus quality on a single mesh, and show a comparison in Fig. 17. We find that there is a clear correlation between order and quality, and we select 3rd order SH as it still has significant visual quality improvements but is not expensive to include in the forward rendering equation.



Different UV Initializations. We also demonstrate that our approach works on arbitrary bijective UV mappings even those with poor initial distortions, as long as they are bijective. For example, given the Ogre model in Fig. 18, we initialize the UV parameterization of its low-poly mesh using the Tutte embedding [Tutte, 1963], and recenter and scale it within the bounds $[0.25, 0.75]$. We then bake the texture from the original model onto the remeshed version with NVDiffModeling and our approach, and observe that our approach can recover a higher quality UV embedding that can better represent the texture content. Then, we compare this to a different

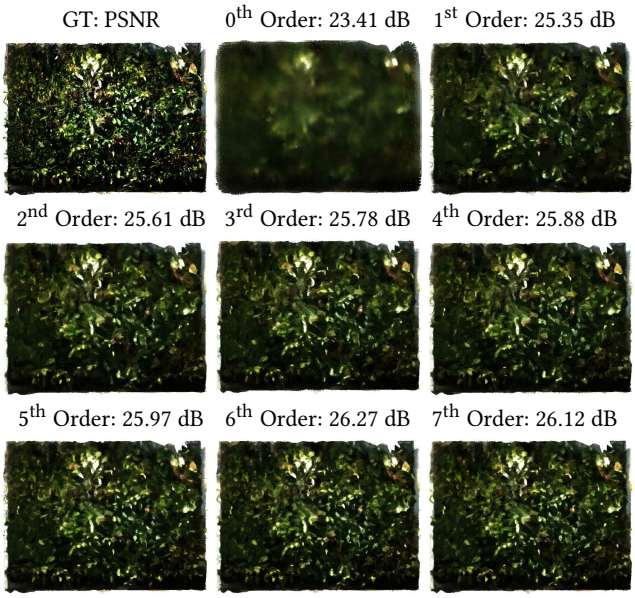


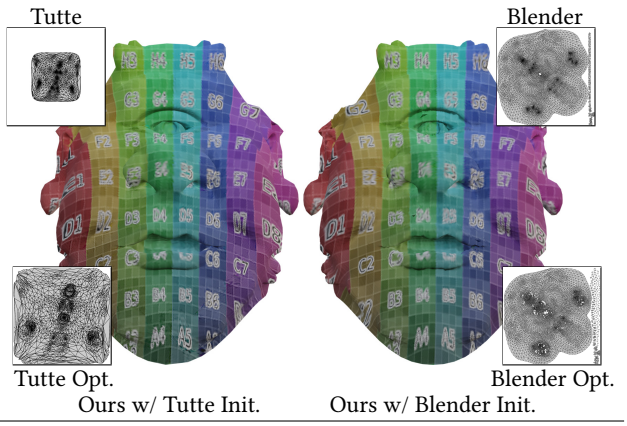
Fig. 17. We compare different orders of SH for our approach. The effect of increasing the order is negligible around order 5, but visible improvement can be seen in orders 0 to 4.

UV parameterization, using Blender's smart unwrapping, and compare it with and without our UV optimization. Our optimized Tutte embedding achieves better PSNR as compared to an unoptimized Blender UV unwrapping, but by also optimizing Blender's UV unwrapping we are able to achieve even better performance. In Fig. 18, we visualize the optimized Tutte embedding using our approach, as well as the parameterization output by Blender, optimized by our approach. Because there are many small charts in Blender's output, there is some difficulty in optimization, but these small triangles are able to expand and take up more texels, and within the large chart, areas with internal cuts are expanded.

Content-aware Symmetric Dirichlet $\mathcal{L}_{I \circ UV}$. We validate the necessity of including $\mathcal{L}_{\text{I} \circ \text{UV}}$ in our objective function by comparing our approach with and without it, as summarized Table 3, showing our approach performs much better with this term. We observe that with only the image loss $\mathcal{L}_{\text{diff}}$ to guide UV optimization, there is not sufficient gradient information to update the UV coordinates. We then demonstrate that our approach relies more on $\mathcal{L}_{\text{I} \circ \text{UV}}$ rather than $\mathcal{L}_{\text{UV}}$ to achieve improved performance. To this end, we test our approach on two models (Jpn. Lamp and Hand Fan) with Spherical Harmonic Parallax and one of the two terms: $\mathcal{L}_{\text{UV}}$ and $\mathcal{L}_{\text{I} \circ \text{UV}}$. We find that using $\mathcal{L}_{\text{IoUV}}$ as opposed to $\mathcal{L}_{\text{UV}}$ improves the output by 0.25 and 0.4 PSNR, respectively. Finally, we experimented with only optimizing UV using $\mathcal{L}_{\text{diff}}$, where we found that $\mathcal{L}_{\text{diff}}$ alone does not provide sufficient gradient information for the charts to expand, due to the discontinuous nature of the pixel gradients.

6 CONCLUSION & DISCUSSION

We demonstrate the efficacy of combining UV parameterization, view parallax mapping, vertex displacement, and texture-baking



Tutte Initialization		Blender Initialization	
Ours	32.17/0.994	Ours	33.22/0.995
$\text{Ours}_{\text{SH,V}}$	29.73/0.990	$\text{Ours}_{\text{SH,V}}$	31.65/0.992

Fig. 18. We demonstrate how our approach handles different UV initializations. The left of the figure is our optimized textured result with the Tutte embedding initializaiton which is optimized to be space filling. The right is with the Blender's UV unwrapping initialization which introduces additional seams. From the PSNR/MS-SSIM statistics table at the bottom, for the Tutte initialization, we see an increase of about 2.4 PSNR after optimizing UV, and for the Blender initialization, we see an increase of 1.6 PSNR. By optimizing the initial Tutte parameterization, our approach outperforms the initial Blender UV, and still improves when used on a better parameterization. Credits to Keenan Crane for the Ogre mesh.

Table 3. We ablate our approach with and without image-guided symmetric Dirichlet energy $L_{\text{I}\circ\text{UV}}$. Without this term, i.e. relying only on the image loss $\mathcal{L}_{\text{diff}}$, there is insufficient gradient information to guide the charts to expand, leading to inferior results.

Ablation of $L_{\text{I}\circ\text{UV}}$		
64×64	w/o $L_{\text{I}\circ\text{UV}}$	w/ $L_{\text{I}\circ\text{UV}}$
Cloud	32.19/0.956	**33.32/0.960**
Item Icons	21.69/0.907	**23.77/0.939**
Pixel Map	17.61/0.763	**18.66/0.808**
Painting	14.19/0.501	**14.77/0.524**
Sea Map	15.41/0.463	**16.43/0.593**
Rainforest	25.72/0.962	**27.82/0.972**
Texture 1	25.11/0.957	**28.36/0.967**
Texture 2	24.66/0.964	**29.13/0.967**
Texture 3	26.29/0.982	**30.93/0.982**
Hokusai	18.70/0.649	**19.52/0.674**

in a joint optimization formulation. Our approach builds on recent differentiable rendering advances to optimize a new parallax map and uses classic optimization techniques to ensure bijectivity of the UV parameterization. Over a dataset of complex 3D models, we demonstrate that combining these components leads to an increase in output visual quality, with little overhead.

Limitations. Our approach has a couple of limitations. First, the cost of optimizing UV coordinates is quite expensive, especially as compared to only baking textures without UV optimization. The bottleneck lies in the global intersection check and line search,

Table 4. Information about each 2D texture in our dataset.

2D Texture Information		
Name	Height	Width
Aliasing	1024	1024
Cloud	512	512
Item Icons	1024	1024
Pixel Map	2048	2048
Painting	1024	512
Sea Map	1024	2048
Mario	1024	1024
Rainforest	1024	1024
Texture 1	1024	1024
Texture 2	1024	1024
Texture 3	1024	1024
Hokusai	1024	2048

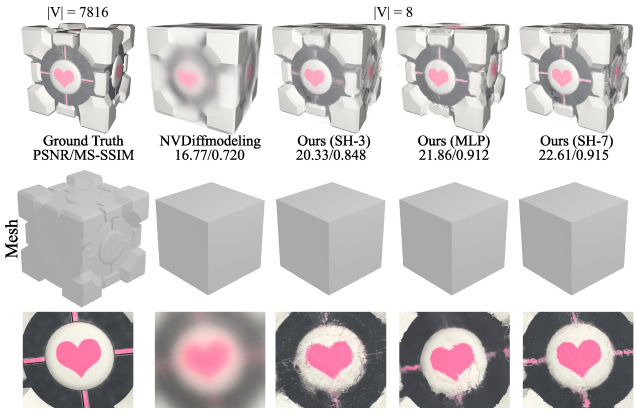


Fig. 19. We compare SH View Parallax and NVDiffmodeling on a challenging example. NVDiffmodeling cannot accurately capture the sharp details of the original model due to geometric differences. With only 3rd order SH, the low-poly model looks plausible from the front, but there are artifacts on the side. Using an MLP based approach reduces the number of artifacts on the side, but it takes significantly longer to train, and requires more resources to use in practice. We can increase the order of SH to 7 and find that it is able to outperform the MLP.

which we plan to speedup through re-engineering our implementation in CUDA. Second, our SH parallax mapping may introduce "cracks" if the low-poly and high-poly model are significantly different, see Fig. 19 for such an example. Due to bounds on frequency for our translation function, these cracks are inherent, as there will always be some region that it cannot fully capture, but they can be removed by introducing higher orders of SH or alternative representations such as neural nets. In practice, they will not be visible from a distance, and do not appear if the low-poly and high-poly models are similar. Thus, during practical use this drawback will be mitigated by switching LODs. Third, our method assumes that the cuts imposed by the original parameterization are fixed, and not optimized jointly with the learned texture map. It may be possible to incorporate OptCuts [Li et al., 2018b] or other cut reparameterizations with our work in order to optimize cuts, UV parameterization, and texture jointly.

Robustness & Failure Models. Our approach will never fail to output a model, even if it contains cracks in the output texture. If the input UV parameterization is bijective, the output will also be bijective as guaranteed by the safety of the line-search scheme. If the source and target mesh differ significantly, there may be issues in the optimized texture such as visible seams. For our test set where meshes are reasonably similar, cracks are never a prominent problem. We do not expect such problems when using mesh simplification tools to generate the output from the input mesh.

Downstream Applications. Our approach by default optimizes spherical harmonic view parallax maps, which might not have support in the downstream applications' rendering pipeline. To adapt our method to these applications, we propose a technique to "bake out" parallax mapping. Note that parallax mapping generates a view-dependent texture image, so the goal of "baking out" is to pick a single view direction d as a representative texel value. The differentiable rendering pipeline would choose to average the texel values over all view directions d, oftentimes leading to blurring data. Instead, we propose to follow the conventional approach and choose a single d along the normal direction, i.e. fetching the texel data along normal direction to yield a new "representative" texture to be used by a conventional rendering pipeline without parallex mapping functionality.

REFERENCES

Adobe. 2014. Substance Painter. https://substance3d.adobe.com/

Blender Online Community. 2018. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam. http://www.blender.org

Jonathan Cohen, Marc Olano, and Dinesh Manocha. 1998. Appearance-Preserving Simplification. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '98)*. Association for Computing Machinery, New York, NY, USA, 115–122.

DGG. 2018. Rapid Compact. https://www.rapidcompact.com/

Donya Labs AB. 2022. Simplygon 10. https://www.simplygon.com/

W. Engel. 2019. *GPU Zen 2: Advanced Rendering Techniques*. Black Cat Publishing Inc., United Kingdom, LE17 4AE, Lutterworth, 25 Church St.

Epic Games. 2022. Unreal Engine 5. https://www.unrealengine.com/en-US/unreal-engine-5

Jon Hasselgren, Jacob Munkberg, Jaakko Lehtinen, Miika Aittala, and Samuli Laine. 2021. Appearance-Driven Automatic 3D Model Simplification. In *Eurographics Symposium on Rendering*. Eurographics Association, Goslar, DEU, 13 pages.

Zhongshi Jiang, Scott Schaefer, and Daniele Panozzo. 2017. Simplicial Complex Augmentation Framework for Bijective Maps. *ACM Trans. Graph.* 36, 6, Article 186 (nov 2017), 9 pages.

Zhongshi Jiang, Teseo Schneider, Denis Zorin, and Daniele Panozzo. 2020. Bijective projection in a shell. *ACM Transactions on Graphics* 39, 6 (2020), 247–. https://doi.org/10.1145/3414685.3417769

Justin Johnson, Nikhila Ravi, Jeremy Reizenstein, David Novotny, Shubham Tulsiani, Christoph Lassner, and Steve Branson. 2020. Accelerating 3D Deep Learning with PyTorch3D. In *SIGGRAPH Asia 2020 Courses* (Virtual Event) *(SA '20)*. Association for Computing Machinery, New York, NY, USA, Article 10, 1 pages.

Tomomichi Kaneko, Toshiyuki Takahei, Masahiko Inami, Naoki Kawakami, Yasuyuki Yanagida, Taro Maeda, and Susumu Tachi. 2001. Detailed shape representation with parallax mapping. In *In Proceedings of the ICAT 2001*. IEEE Computer Society, USA, 205–208.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). Association for Computing Machinery, New York, NY, USA, 11 pages.

Samuli Laine, Janne Hellsten, Tero Karras, Yeongho Seol, Jaakko Lehtinen, and Timo Aila. 2020. Modular Primitives for High-Performance Differentiable Rendering. *ACM Trans. Graph.* 39, 6, Article 194 (nov 2020), 14 pages.

Bruno Lévy. 2019. Geogram. http://alice.loria.fr/index.php/software/4-library/75-geogram.html

Bruno Lévy, Sylvain Petitjean, Nicolas Ray, and Jérome Maillot. 2002. Least Squares Conformal Maps for Automatic Texture Atlas Generation. *ACM Trans. Graph.* 21, 3 (jul 2002), 362–371.

Minchen Li, Zachary Ferguson, Teseo Schneider, Timothy Langlois, Denis Zorin, Daniele Panozzo, Chenfanfu Jiang, and Danny M. Kaufman. 2020. Incremental Potential Contact: Intersection-and Inversion-Free, Large-Deformation Dynamics. *ACM Trans. Graph.* 39, 4, Article 49 (jul 2020), 20 pages.

Minchen Li, Danny M. Kaufman, and Chenfanfu Jiang. 2021. Codimensional Incremental Potential Contact. *ACM Trans. Graph.* 40, 4, Article 170 (jul 2021), 24 pages.

Minchen Li, Danny M. Kaufman, Vladimir G. Kim, Justin Solomon, and Alla Sheffer. 2018b. OptCuts: Joint Optimization of Surface Cuts and Parameterization. *ACM Trans. Graph.* 37, 6, Article 247 (dec 2018), 13 pages.

Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. 2018a. Differentiable Monte Carlo Ray Tracing through Edge Sampling. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 37, 6 (2018), 222:1–222:11.

D. C. Liu and J. Nocedal. 1989. On the limited memory BFGS method for large scale optimization. *Math. Programming* 45, 3, (Ser. B) (1989), 503–528.

Shichen Liu, Weikai Chen, Tianye Li, and Hao Li. 2019. Soft Rasterizer: A Differentiable Renderer for Image-Based 3D Reasoning. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE Computer Society, USA, 7707–7716.

Matthew M. Loper and Michael J. Black. 2014. OpenDR: An Approximate Differentiable Renderer. In *Computer Vision – ECCV 2014*, David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars (Eds.). Springer International Publishing, Cham, 154–169.

Fujun Luan, Shuang Zhao, Kavita Bala, and Zhao Dong. 2021. Unified Shape and SVBRDF Recovery using Differentiable Monte Carlo Rendering.

Marmoset. 2022. Marmoset Toolbag. https://marmoset.co/toolbag/

Morgan McGuire and Max McGuire. 2005. *Steep Parallax Mapping*. Technical Report. Brown University. Report at Brown University.

Manfred M. Nerurkar. 2021. InstaLOD. https://instalod.com

Baptiste Nicolet, Alec Jacobson, and Wenzel Jakob. 2021. Large Steps in Inverse Rendering of Geometry. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)* 40, 6 (Dec. 2021). https://doi.org/10.1145/3478513.3480501

Merlin Nimier-David, Delio Vicini, Tizian Zeltner, and Wenzel Jakob. 2019. Mitsuba 2: A Retargetable Forward and Inverse Renderer. *ACM Trans. Graph.* 38, 6, Article 203 (nov 2019), 17 pages.

Gustavo Patow and Xavier Pueyo. 2003. A Survey of Inverse Rendering Problems. *Computer Graphics Forum* 22, 4 (2003), 663–687.

Pixologic. 2022. ZBrush. https://pixologic.com/

Fábio Policarpo, Manuel M. Oliveira, and João L. D. Comba. 2005. Real-Time Relief Mapping on Arbitrary Polygonal Surfaces. *ACM Trans. Graph.* 24, 3 (jul 2005), 935.

Michael Rabinovich, Roi Poranne, Daniele Panozzo, and Olga Sorkine-Hornung. 2017. Scalable Locally Injective Mappings. *ACM Trans. Graph.* 36, 2, Article 16 (apr 2017), 16 pages.

Ravi Ramamoorthi and Pat Hanrahan. 2001. An Efficient Representation for Irradiance Environment Maps. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '01)*. Association for Computing Machinery, New York, NY, USA, 497–500.

Pedro V. Sander, Steven J. Gortler, John Snyder, and Hugues Hoppe. 2002. Signal-Specialized Parametrization. In *Proceedings of the 13th Eurographics Workshop on Rendering* (Pisa, Italy) *(EGRW '02)*. Eurographics Association, Goslar, DEU, 87–98.

P. V. Sander, Z. J. Wood, S. J. Gortler, J. Snyder, and H. Hoppe. 2003. Multi-Chart Geometry Images. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing* (Aachen, Germany) *(SGP '03)*. Eurographics Association, Goslar, DEU, 146–155.

Sketchfab. 2022. The best 3D viewer on the web. https://sketchfab.com/

Jason Smith and Scott Schaefer. 2015. Bijective Parameterization with Free Boundaries. *ACM Trans. Graph.* 34, 4, Article 70 (jul 2015), 9 pages.

Jian-Ping Su, Chunyang Ye, Ligang Liu, and Xiao-Ming Fu. 2020. Efficient Bijective Parameterizations. *ACM Trans. Graph.* 39, 4, Article 111 (aug 2020), 8 pages.

Haoran Sun, Shiyi Wang, Wenhai Wu, Yao Jin, Hujun Bao, and Jin Huang. 2022. Efficient Texture Parameterization Driven by Perceptual-Loss-on-Screen. *Computer Graphics Forum* 41, 7 (2022), 12 pages.

Natalya Tatarchuk. 2006. Practical Parallax Occlusion Mapping with Approximate Soft Shadows for Detailed Surface Rendering. In *ACM SIGGRAPH 2006 Courses* (Boston, Massachusetts) *(SIGGRAPH '06)*. Association for Computing Machinery, New York, NY, USA, 81–112.

Geetika Tewari, John Snyder, Pedro V. Sander, Steven J. Gortler, and Hugues Hoppe. 2004. Signal-Specialized Parameterization for Piecewise Linear Reconstruction. In *Geometry Processing* (Nice, France) *(SGP '04)*. Association for Computing Machinery, New York, NY, USA, 55–64.

Theo Thonat, Francois Beaune, Xin Sun, Nathan Carr, and Tamy Boubekeur. 2021. Tessellation-Free Displacement Mapping for Ray Tracing. 40, 6, Article 282 (dec 2021), 16 pages. https://doi.org/10.1145/3478513.3480535

W. T. Tutte. 1963. How to Draw a Graph. *Proceedings of the London Mathematical Society* s3-13, 1 (1963), 743–767.

Table 5. Different measures of complexity for each high-poly mesh and simplified mesh in our dataset.

LOD-1	# Comp.	# Vertices	# Faces
Dragon Jar	20	866	1182
Garden Seat	3	373	624
Helmet	224	3673	4038
Cat Statue	17	572	728
Chn. Chess	40	1076	1354
Hand Fan	28	521	518
Iron Cup	15	600	946
Cut Fish	24	703	864
Easter Egg	3	410	654
Baguette	8	244	306
Greek Vases	41	1181	1630
Gundam	275	4308	4614
Italian Car	57	1198	1468
Teacup	8	462	692
Jpn. Lamp	40	955	1042
Lego Fig.	36	760	872
Lemon	1	299	460
Mask	42	1326	1726
Golem	125	2799	3202
White Tree	312	4937	5094
Pengu	121	2188	2372
Pony (Car)	42	1042	1232
Sand Arena	222	3799	3990
Skate Bunny	154	2977	3334
Knight	184	2763	2872
Umbrella	188	3107	3248

LOD-2	# Comp.	# Vertices	# Faces
Dragon Jar	10	328	380
Garden Seat	1	122	194
Helmet	37	624	684
Cat Statue	6	169	214
Chn. Chess	19	269	270
Hand Fan	13	192	192
Iron Cup	8	243	304
Cut Fish	12	241	268
Easter Egg	1	135	214
Baguette	2	53	64
Greek Vases	28	487	560
Gundam	47	816	886
Italian Car	22	380	432
Teacup	5	143	192
Jpn. Lamp	32	611	660
Lego Fig.	14	262	304
Lemon	1	113	178
Mask	16	440	522
Golem	20	574	716
White Tree	82	1036	1018
Pengu	34	617	664
Pony (Car)	18	307	334
Sand Arena	50	808	836
Skate Bunny	34	641	704
Knight	32	560	578
Umbrella	69	1187	1210

LOD-3	# Comp.	# Vertices	# Faces
Dragon Jar	2	61	84
Garden Seat	3	81	98
Helmet	11	179	200
Cat Statue	4	62	80
Chn. Chess	3	42	44
Hand Fan	2	31	34
Iron Cup	1	120	160
Cut Fish	4	80	100
Easter Egg	3	89	114
Baguette	2	28	32
Greek Vases	16	219	226
Gundam	26	340	328
Italian Car	3	79	86
Teacup	3	92	104
Jpn. Lamp	12	349	368
Lego Fig.	4	89	108
Lemon	1	53	84
Mask	13	203	218
Golem	15	251	276
White Tree	40	343	304
Pengu	18	291	292
Pony (Car)	7	107	108
Sand Arena	20	225	218
Skate Bunny	16	251	264
Knight	7	207	222
Umbrella	18	342	364

High-Poly	# Comp.	# Vertices	# Faces
Dragon Jar	40	11388	20412
Garden Seat	746	12190	15000
Helmet	213	5293	6509
Cat Statue	13	9454	17536
Chn. Chess	244	18620	29660
Hand Fan	134	3675	4048
Iron Cup	440	15195	22258
Cut Fish	57	5081	8168
Easter Egg	8	4917	8964
Baguette	89	16951	28640
Greek Vases	273	23399	37524
Gundam	1776	17948	16106
Italian Car	445	7923	8876
Teacup	18	3311	5504
Jpn. Lamp	200	3758	4586
Lego Fig.	117	3112	3680
Lemon	17	502	723
Mask	102	3165	4082
Golem	225	5284	6677
White Tree	598	8312	9398
Pengu	655	14373	18163
Pony (Car)	331	6817	8338
Sand Arena	1960	24209	23931
Skate Bunny	1216	27241	34444
Knight	1298	70291	103366
Umbrella	222	2511	2495

Table 6. We compare various algorithms in optimizing a 2D planar mesh with various texture images, where we initialize the UV mapping to the identity mapping. We perform texture baking to a lower-resolution texture image, using our method, NVDiffModeling, bi-linear downsampling. For NVDiffModeling, we initialize the UV parameterization using UVAtlas with signal-specialized parameterization [Sander et al., 2002]. For our mesh, we only optimize the diffuse map and UV coordinates (i.e. no parallax or displacement maps are optimized), and all other methods only optimize the diffuse map. All input images are at least 512^2, and are not necessarily square. When performing extreme compression, such as to 64^2 images, our approach is able to consistently outperform other approaches. Our approach is even better when compressing to 256^2 images.

Texture Compression: $\mathbb{E}$[PSNR]$^\uparrow$/MSSSIM$^\uparrow$				
64 × 64	NVDiffModeling	Ours(UV)	Downsample	UVAtlas
Aliasing	10.10/0.311	10.10/0.315	9.73/0.312	**10.14/0.320**
Cloud	32.60/0.956	**33.32/0.960**	31.19/0.946	31.99/0.950
Item Icons	22.49/0.921	**23.77/0.939**	21.72/0.902	22.03/0.907
Pixel Map	18.22/0.786	**18.66/0.808**	17.74/0.756	18.04/0.770
Painting	14.64/0.514	**14.77/0.524**	14.16/0.457	14.37/0.468
Sea Map	16.17/0.560	**16.43/0.593**	16.03/0.553	15.80/0.491
Mario	17.59/0.798	**19.28/0.839**	16.31/0.749	15.85/0.709
Rainforest	27.05/0.966	**27.82/0.972**	27.10/0.964	26.45/0.961
Texture 1	25.82/0.952	**28.36/0.967**	25.43/0.956	26.00/0.928
Texture 2	25.50/0.956	**29.13/0.967**	25.08/0.960	26.02/0.928
Texture 3	26.93/0.973	**30.93/0.982**	26.66/0.980	28.01/0.969
Hokusai	19.15/0.656	**19.52/0.674**	17.42/0.539	18.68/0.615

Texture Compression: $\mathbb{E}$[PSNR]$^\uparrow$/MSSSIM$^\uparrow$				
128 × 128	NVDiffModeling	Ours_{UV}	Downsample	UVAtlas
Aliasing	10.45/0.372	**10.57/0.396**	9.98/0.342	10.52/0.386
Cloud	37.33/0.985	**39.31/0.990**	36.24/0.980	36.18/0.978
Item Icons	25.31/0.969	**28.24/0.983**	23.98/0.947	24.88/0.959
Pixel Map	20.00/0.889	**20.85/0.907**	19.08/0.838	19.68/0.872
Painting	16.64/0.747	**17.00/0.756**	15.59/0.638	16.02/0.679
Sea Map	17.44/0.745	**18.06/0.784**	17.02/0.700	16.83/0.647
Mario	21.60/0.921	**23.47/0.938**	21.21/0.913	19.26/0.863
Rainforest	30.17/0.985	**30.74/0.987**	29.61/0.980	28.73/0.980
Texture 1	26.84/0.980	**34.44/0.989**	25.86/0.976	28.00/0.954
Texture 2	26.64/0.980	**36.24/0.987**	25.52/0.976	28.58/0.953
Texture 3	27.77/0.984	**36.16/0.992**	26.97/0.988	31.41/0.982
Hokusai	21.46/0.816	**21.99/0.823**	20.58/0.771	20.82/0.769

Texture Compression: $\mathbb{E}$[PSNR]$^\uparrow$/MSSSIM$^\uparrow$				
256 × 256	NVDiffModeling	Ours_{UV}	Downsample	UVAtlas
Aliasing	12.04/0.674	**13.15/0.678**	11.52/0.642	12.24/0.588
Cloud	44.28/0.997	**46.88/0.998**	42.38/0.996	42.25/0.995
Item Icons	30.09/0.994	**35.67/0.998**	28.71/0.988	28.95/0.988
Pixel Map	22.89/0.966	**24.16/0.972**	22.11/0.944	22.26/0.950
Painting	20.87/0.938	**22.16/0.944**	19.68/0.897	19.51/0.887
Sea Map	20.05/0.925	**21.28/0.942**	19.46/0.906	18.62/0.827
Mario	26.19/0.979	**30.08/0.988**	25.25/0.972	23.20/0.952
Rainforest	32.23/0.994	**36.05/0.997**	32.44/0.993	31.75/0.992
Texture 1	29.33/0.995	**41.09/0.998**	27.95/0.993	33.89/0.976
Texture 2	30.06/0.993	**40.84/0.997**	28.65/0.993	37.10/0.973
Texture 3	30.23/0.993	**40.52/0.997**	28.98/0.996	38.34/0.989
Hokusai	25.41/0.946	**26.15/0.940**	23.78/0.913	24.35/0.912

Z. Wang, E.P. Simoncelli, and A.C. Bovik. 2003. Multiscale structural similarity for image quality assessment. In *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, Vol. 2. IEEE Computer Society, USA, 1398–1402 Vol.2.

Terry Welsh et al. 2004. Parallax mapping with offset limiting: A perpixel approximation of uneven surfaces. *Infiscape Corporation* 1 (2004), 1–9.

Jonathon Young. 2017. xatlas. https://github.com/jpcy/xatlas.

A ADDITIONAL RESULTS

Table 7. Experiments with the same setting as Table 6 but instead we compress textures on 3D models, using their provided UV coordinates as initialization.

Resolution : $2048^2 \to 128^2$	Original Model: $\mathbb{E}$[PSNR]$^\uparrow$/MSSSIM$^\uparrow$			
	UVAtlas	Ours	Ours_{UV}	NVDiffModeling
Dragon Jar	26.15/0.916	28.96/0.958	26.60/0.919	26.14/0.916
Garden Seat	22.17/0.807	25.61/0.913	23.33/0.838	22.28/0.810
Helmet	27.60/0.970	30.36/0.986	28.72/0.976	26.94/0.973
Cat Statue	29.95/0.934	32.43/0.969	31.24/0.953	31.11/0.953
Chn. Chess	34.88/0.981	37.28/0.989	35.58/0.982	36.14/0.984
Hand Fan	31.07/0.982	38.00/0.997	36.82/0.995	36.99/0.995
Iron Cup	23.11/0.922	26.64/0.965	24.15/0.932	23.39/0.924
Cut Fish	35.51/0.988	43.55/0.997	38.67/0.993	37.24/0.992
Easter Egg	25.02/0.900	34.85/0.985	30.11/0.963	27.59/0.942
Baguette	40.10/0.995	39.27/0.996	40.29/0.996	40.12/0.996
Greek Vases	30.87/0.979	34.94/0.993	31.72/0.982	29.85/0.982
Gundam	20.21/0.949	20.23/0.951	20.10/0.948	20.08/0.947
Italian Car	37.95/0.996	44.74/0.999	43.24/0.998	41.30/0.998
Teacup	29.45/0.925	37.27/0.979	34.33/0.960	32.55/0.950
Jpn. Lamp	29.28/0.937	33.88/0.979	31.26/0.959	30.45/0.953
Lego Fig.	35.69/0.996	39.40/0.999	39.44/0.998	39.02/0.998
Lemon	42.32/0.986	45.24/0.993	44.88/0.992	44.00/0.989
Mask	44.19/0.996	47.31/0.998	46.04/0.997	43.80/0.995
Golem	36.51/0.973	39.28/0.987	38.14/0.981	37.43/0.976
White Tree	17.32/0.834	17.77/0.809	17.61/0.826	17.26/0.823
Pengu	34.38/0.975	39.13/0.990	35.38/0.976	34.54/0.978
Pony (Car)	38.03/0.992	44.31/0.998	40.47/0.995	39.43/0.994
Sand Arena	39.75/0.996	43.30/0.999	43.53/0.998	42.46/0.998
Skate Bunny	28.79/0.986	32.63/0.994	30.91/0.990	29.62/0.988
Knight	28.77/0.980	28.54/0.983	29.41/0.982	28.33/0.977
Umbrella	29.81/0.969	35.31/0.990	29.84/0.962	28.34/0.949

Resolution : $2048^2 \to 256^2$	Original Model: $\mathbb{E}$[PSNR]$^\uparrow$/MSSSIM$^\uparrow$			
	UVAtlas	Ours	Ours_{UV}	NVDiffModeling
Dragon Jar	28.93/0.961	33.39/0.988	29.99/0.969	29.43/0.966
Garden Seat	24.57/0.898	29.27/0.971	26.51/0.936	25.03/0.909
Helmet	29.45/0.982	33.65/0.995	31.20/0.990	29.27/0.982
Cat Statue	31.89/0.964	36.27/0.991	34.13/0.983	33.90/0.981
Chn. Chess	35.88/0.987	38.77/0.993	36.99/0.990	36.85/0.989
Hand Fan	32.23/0.987	39.67/0.998	39.08/0.997	38.87/0.997
Iron Cup	24.44/0.938	29.33/0.984	25.62/0.953	25.08/0.942
Cut Fish	39.11/0.994	45.45/0.998	42.69/0.997	42.14/0.996
Easter Egg	29.33/0.961	40.31/0.997	36.17/0.992	34.06/0.987
Baguette	40.27/0.996	39.35/0.996	40.38/0.997	40.32/0.996
Greek Vases	33.24/0.989	37.85/0.997	34.23/0.992	33.67/0.991
Gundam	20.34/0.952	21.76/0.967	20.23/0.952	20.21/0.951
Italian Car	40.89/0.997	45.62/1.000	49.90/1.000	47.96/0.999
Teacup	32.67/0.956	40.33/0.993	37.53/0.983	36.12/0.976
Jpn. Lamp	30.66/0.954	36.14/0.988	33.12/0.976	32.05/0.969
Lego Fig.	39.09/0.998	38.27/0.998	42.73/0.999	42.46/0.999
Lemon	44.24/0.990	48.82/0.998	49.35/0.998	46.95/0.995
Mask	47.73/0.998	49.33/0.999	49.38/0.999	46.84/0.998
Golem	38.17/0.984	41.26/0.994	39.93/0.991	38.77/0.986
White Tree	17.57/0.843	17.88/0.818	17.68/0.844	17.22/0.838
Pengu	37.14/0.987	41.88/0.996	38.85/0.990	38.69/0.991
Pony (Car)	41.56/0.996	46.62/0.999	44.65/0.998	42.38/0.997
Sand Arena	42.92/0.998	43.67/0.999	46.78/0.999	40.39/0.997
Skate Bunny	31.17/0.992	34.87/0.997	34.19/0.997	33.72/0.996
Knight	28.87/0.983	28.59/0.984	29.73/0.986	28.69/0.980
Umbrella	33.46/0.988	37.82/0.996	34.93/0.992	32.42/0.983

Resolution : $2048^2 \to 512^2$	High-Poly: $\mathbb{E}$[PSNR]$^\uparrow$/MSSSIM$^\uparrow$			
	UVAtlas	Ours	Ours_{UV}	NVDiffModeling
Dragon Jar	33.09/0.988	40.27/0.998	36.21/0.995	33.85/0.990
Garden Seat	27.78/0.959	33.12/0.989	30.38/0.980	28.43/0.966
Helmet	32.18/0.992	38.69/0.998	34.99/0.997	32.75/0.994
Cat Statue	34.69/0.986	41.25/0.998	39.37/0.997	37.76/0.994
Chn. Chess	36.86/0.992	41.42/0.997	39.34/0.995	38.88/0.994
Hand Fan	33.65/0.991	40.62/0.999	43.17/0.999	43.39/0.999
Iron Cup	26.07/0.957	32.54/0.994	27.91/0.975	27.32/0.964
Cut Fish	43.55/0.997	46.61/0.999	48.25/0.999	47.10/0.999
Easter Egg	34.87/0.989	44.21/0.999	43.23/0.999	41.05/0.998
Baguette	40.87/0.997	40.31/0.997	41.17/0.997	40.95/0.997
Greek Vases	36.46/0.995	40.53/0.999	37.88/0.997	37.50/0.997
Gundam	20.63/0.956	26.12/0.990	21.49/0.967	20.98/0.961
Italian Car	43.45/0.999	45.27/1.000	54.37/1.000	39.26/0.998
Teacup	36.39/0.979	43.59/0.998	40.90/0.995	39.48/0.991
Jpn. Lamp	32.42/0.970	38.38/0.995	35.11/0.987	34.13/0.983
Lego Fig.	42.12/0.999	41.89/0.999	44.80/0.999	44.99/0.999
Lemon	46.88/0.995	52.51/1.000	53.78/1.000	51.36/0.999
Mask	51.00/0.999	50.23/1.000	49.81/0.999	49.86/0.999
Golem	40.06/0.992	43.28/0.998	42.17/0.996	40.58/0.993
White Tree	17.41/0.843	17.86/0.815	17.71/0.847	17.32/0.842
Pengu	40.15/0.994	43.57/0.998	42.81/0.997	42.85/0.997
Pony (Car)	44.68/0.998	46.26/0.999	48.01/1.000	45.05/0.998
Sand Arena	44.71/0.999	43.93/0.999	48.32/0.999	42.79/0.998
Skate Bunny	34.57/0.997	37.70/0.999	39.93/0.999	39.61/0.999
Knight	29.34/0.987	28.88/0.986	29.94/0.989	29.06/0.983
Umbrella	37.55/0.995	36.47/0.995	38.02/0.996	36.60/0.995

Table 8. Comparison of texture baking on LOD-1 using NVDiffModeling, our approach, our approach with SH and UV optimization, Unreal, and Blender.

Resolution : $2048^2 \to 128^2$	LOD-1: $\mathbb{E}$[PSNR]$^\uparrow$/MSSSIM$^\uparrow$				
	NVDiffModeling	$\text{Ours}_{\text{UV,SH}}$	Ours	UE	Blender
Dragon Jar	24.57/0.887	27.71/0.956	**29.11/0.968**	22.48/0.838	13.88/0.766
Garden Seat	21.91/0.804	23.92/0.889	**26.27/0.939**	20.87/0.730	21.29/0.774
Helmet	25.55/0.960	26.84/0.972	**26.96/0.969**	22.84/0.920	22.77/0.928
Cat Statue	28.73/0.945	30.18/0.974	**31.95/0.981**	27.28/0.909	27.86/0.926
Chn. Chess	31.41/0.977	32.29/0.986	**34.85/0.987**	30.19/0.960	30.22/0.965
Hand Fan	29.82/0.986	31.65/0.992	**33.51/0.993**	25.97/0.961	26.29/0.964
Iron Cup	23.15/0.923	24.56/0.939	**26.39/0.966**	21.59/0.891	22.28/0.907
Cut Fish	31.67/0.985	33.21/0.992	**35.53/0.994**	28.78/0.964	29.60/0.971
Easter Egg	25.55/0.920	29.85/0.975	**32.23/0.984**	24.36/0.879	25.58/0.914
Baguette	34.28/0.993	34.62/0.994	**35.72/0.995**	33.68/0.986	34.05/0.991
Greek Vases	28.00/0.977	29.21/0.986	**31.82/0.991**	26.72/0.963	27.61/0.972
Gundam	19.47/0.930	19.74/0.939	**19.81/0.941**	18.29/0.886	18.88/0.901
Italian Car	28.39/0.983	30.36/0.991	**32.91/0.994**	24.34/0.946	25.71/0.959
Teacup	28.26/0.928	30.22/0.959	**32.61/0.975**	25.98/0.886	26.49/0.903
Jpn. Lamp	26.62/0.928	28.19/0.957	**30.32/0.968**	25.07/0.890	25.31/0.902
Lego Fig.	30.73/0.992	32.48/0.995	**34.30/0.996**	24.42/0.967	24.93/0.972
Lemon	38.56/0.987	39.42/0.995	**39.53/0.991**	36.26/0.980	37.64/0.985
Mask	31.77/0.990	32.31/0.994	**34.97/0.995**	31.41/0.984	31.60/0.988
Golem	30.46/0.971	30.79/0.980	**32.98/0.983**	28.94/0.948	29.30/0.960
White Tree	17.02/0.818	16.98/0.805	**17.42/0.792**	14.70/0.756	12.26/0.755
Pengu	30.08/0.968	31.09/0.981	**33.47/0.984**	26.93/0.930	28.59/0.953
Pony (Car)	32.05/0.982	**32.89/0.988**	32.87/0.981	18.83/0.901	24.17/0.942
Sand Arena	32.71/0.991	33.42/0.994	**35.89/0.995**	30.59/0.982	31.82/0.988
Skate Bunny	26.43/0.979	27.83/0.985	**29.44/0.989**	24.03/0.951	24.10/0.956
Knight	26.22/0.973	26.06/0.970	**27.86/0.980**	24.93/0.957	22.79/0.949
Umbrella	26.44/0.943	27.05/0.956	**29.57/0.972**	18.91/0.841	21.12/0.882

Resolution : $2048^2 \to 256^2$	LOD-1: $\mathbb{E}$[PSNR]$^\uparrow$/MSSSIM$^\uparrow$				
	NVDiffModeling	$\text{Ours}_{\text{UV,SH}}$	Ours	UE	Blender
Dragon Jar	26.51/0.937	30.61/0.984	**31.95/0.988**	23.37/0.882	24.23/0.911
Garden Seat	24.08/0.893	27.34/0.966	**29.08/0.976**	22.36/0.820	23.09/0.862
Helmet	26.90/0.974	29.29/0.987	**29.78/0.987**	23.33/0.936	23.27/0.943
Cat Statue	29.78/0.968	31.47/0.988	**33.44/0.992**	28.06/0.938	28.68/0.951
Chn. Chess	32.02/0.984	33.22/0.992	**36.08/0.993**	30.55/0.968	30.79/0.973
Hand Fan	30.38/0.989	32.46/0.994	**34.41/0.996**	25.96/0.963	26.38/0.966
Iron Cup	24.35/0.939	26.64/0.968	**28.38/0.983**	22.34/0.908	23.18/0.926
Cut Fish	32.54/0.990	33.61/0.994	**36.05/0.996**	29.05/0.967	29.98/0.976
Easter Egg	28.79/0.967	32.04/0.991	**34.22/0.994**	26.94/0.939	28.23/0.959
Baguette	34.34/0.993	35.15/0.996	**35.81/0.995**	33.75/0.987	33.72/0.988
Greek Vases	28.68/0.983	29.99/0.990	**32.87/0.995**	27.08/0.970	28.07/0.978
Gundam	19.57/0.935	20.33/0.949	**20.75/0.955**	18.53/0.894	18.46/0.891
Italian Car	29.32/0.988	31.20/0.993	**33.60/0.996**	24.56/0.951	26.11/0.963
Teacup	30.02/0.955	32.04/0.986	**33.57/0.990**	27.03/0.915	27.77/0.930
Jpn. Lamp	27.58/0.951	29.61/0.977	**31.50/0.981**	25.74/0.913	26.10/0.923
Lego Fig.	31.29/0.993	32.85/0.995	**34.65/0.997**	24.28/0.968	24.92/0.973
Lemon	39.13/0.993	39.95/0.998	**40.24/0.997**	36.24/0.984	38.18/0.991
Mask	31.93/0.992	32.52/0.995	**35.18/0.996**	31.53/0.987	31.73/0.991
Golem	30.73/0.980	31.15/0.988	**33.56/0.990**	28.87/0.955	29.38/0.969
White Tree	16.96/0.827	17.04/0.809	**17.58/0.809**	14.69/0.767	12.25/0.762
Pengu	30.82/0.980	32.00/0.990	**34.56/0.991**	27.05/0.938	29.08/0.965
Pony (Car)	32.90/0.988	33.21/0.990	**33.95/0.986**	18.82/0.909	24.15/0.947
Sand Arena	33.05/0.993	33.98/0.995	**36.28/0.996**	30.78/0.984	32.09/0.990
Skate Bunny	27.77/0.986	29.67/0.991	**30.86/0.993**	24.30/0.957	24.58/0.962
Knight	26.64/0.978	26.85/0.981	**28.21/0.984**	24.95/0.961	22.85/0.953
Umbrella	28.45/0.971	29.66/0.981	**31.16/0.983**	18.98/0.861	21.35/0.907

Resolution : $2048^2 \to 512^2$	LOD-1: $\mathbb{E}$[PSNR]$^\uparrow$/MSSSIM$^\uparrow$				
	NVDiffModeling	$\text{Ours}_{\text{UV,SH}}$	Ours	UE	Blender
Dragon Jar	28.79/0.970	32.30/0.992	**33.30/0.993**	24.38/0.925	25.50/0.947
Garden Seat	25.88/0.938	29.06/0.981	**30.47/0.984**	23.74/0.889	24.23/0.909
Helmet	28.03/0.982	30.79/0.992	**32.55/0.994**	23.71/0.947	23.62/0.952
Cat Statue	30.53/0.978	32.18/0.993	**34.36/0.995**	28.68/0.957	28.98/0.961
Chn. Chess	32.54/0.987	33.83/0.995	**36.91/0.995**	30.86/0.973	31.22/0.979
Hand Fan	30.74/0.990	32.82/0.995	**34.58/0.996**	25.94/0.965	26.46/0.968
Iron Cup	25.61/0.956	29.00/0.986	**30.51/0.991**	23.00/0.925	23.83/0.942
Cut Fish	32.89/0.992	33.80/0.995	**35.98/0.996**	29.16/0.970	30.09/0.978
Easter Egg	30.40/0.981	32.55/0.994	**34.56/0.996**	28.77/0.967	29.33/0.973
Baguette	34.56/0.994	35.64/0.997	**36.40/0.996**	33.74/0.988	33.64/0.991
Greek Vases	29.03/0.985	30.37/0.992	**33.40/0.996**	27.19/0.973	28.05/0.980
Gundam	20.03/0.945	22.73/0.973	**23.23/0.977**	18.54/0.896	18.55/0.903
Italian Car	29.70/0.989	31.68/0.994	**33.69/0.996**	24.54/0.952	26.23/0.964
Teacup	31.01/0.973	32.40/0.991	**33.97/0.994**	27.59/0.936	28.33/0.949
Jpn. Lamp	28.28/0.965	30.59/0.986	**32.33/0.988**	26.18/0.931	26.44/0.936
Lego Fig.	31.51/0.994	33.12/0.996	**34.53/0.997**	24.18/0.968	24.93/0.973
Lemon	39.60/0.996	40.10/0.999	**40.44/0.998**	36.40/0.989	38.63/0.996
Mask	32.00/0.992	32.68/0.995	**35.05/0.996**	31.57/0.988	31.76/0.991
Golem	30.93/0.985	31.54/0.992	**33.81/0.993**	28.81/0.961	29.45/0.975
White Tree	17.18/0.831	17.07/0.811	**17.64/0.817**	14.68/0.772	12.25/0.765
Pengu	31.23/0.984	32.55/0.993	**35.02/0.994**	27.10/0.945	29.31/0.971
Pony (Car)	33.00/0.990	33.57/0.992	**35.31/0.991**	18.81/0.914	24.08/0.948
Sand Arena	33.22/0.993	34.47/0.996	**36.47/0.996**	30.81/0.985	32.16/0.990
Skate Bunny	28.49/0.988	30.50/0.993	**31.91/0.995**	24.40/0.960	24.71/0.964
Knight	26.76/0.979	27.08/0.983	**28.39/0.986**	24.91/0.963	22.86/0.954
Umbrella	29.57/0.981	31.76/0.990	**33.04/0.991**	19.00/0.875	21.40/0.916

Table 9. Comparison of texture baking on LOD-2 using NVDiffModeling, our approach, our approach with SH and UV optimization, Unreal, and Blender.

Resolution : $2048^2 \to 128^2$	LOD-2: $\mathbb{E}$[PSNR]$^\uparrow$/MSSSIM$^\uparrow$				
	NVDiffModeling	$\text{Ours}_{\text{UV,SH}}$	Ours	UE	Blender
Dragon Jar	22.97/0.868	24.26/0.916	**26.89/0.954**	21.32/0.821	21.58/0.837
Garden Seat	20.49/0.742	22.21/0.844	**24.16/0.904**	19.73/0.684	19.86/0.711
Helmet	22.88/0.930	24.59/0.957	**26.63/0.971**	20.90/0.894	21.19/0.904
Cat Statue	26.19/0.915	27.09/0.949	**28.75/0.966**	24.89/0.886	25.24/0.896
Chn. Chess	25.65/0.946	25.88/0.952	**29.33/0.970**	25.26/0.929	25.07/0.929
Hand Fan	26.80/0.967	28.49/0.980	**30.24/0.986**	23.88/0.933	24.50/0.946
Iron Cup	21.38/0.894	21.76/0.902	**23.76/0.936**	19.67/0.851	20.37/0.869
Cut Fish	27.80/0.970	28.48/0.978	**30.59/0.983**	26.02/0.943	26.52/0.953
Easter Egg	23.37/0.888	25.88/0.953	**26.23/0.939**	22.62/0.850	21.78/0.853
Baguette	30.03/0.985	30.21/0.987	**31.80/0.990**	29.74/0.978	29.95/0.984
Greek Vases	24.26/0.947	25.34/0.964	**26.63/0.966**	23.63/0.936	23.63/0.938
Gundam	18.71/0.902	**19.17/0.919**	18.76/0.902	17.51/0.855	17.90/0.864
Italian Car	25.18/0.964	**27.30/0.979**	26.79/0.972	22.55/0.926	20.35/0.920
Teacup	26.09/0.923	**26.94/0.946**	26.19/0.926	23.63/0.876	24.11/0.891
Jpn. Lamp	24.15/0.898	25.39/0.934	**25.65/0.919**	23.28/0.866	23.46/0.876
Lego Fig.	25.94/0.975	**27.59/0.982**	27.54/0.977	22.68/0.950	22.92/0.953
Lemon	29.71/0.974	29.86/0.981	**31.60/0.982**	28.92/0.965	29.34/0.970
Mask	26.07/0.968	26.83/0.976	**28.13/0.976**	25.79/0.960	25.80/0.964
Golem	25.37/0.941	25.75/0.956	**26.99/0.953**	24.85/0.917	24.96/0.927
White Tree	16.82/0.795	16.79/0.785	**17.06/0.778**	14.57/0.746	12.48/0.743
Pengu	26.51/0.948	27.57/0.968	**28.12/0.957**	24.56/0.906	24.84/0.924
Pony (Car)	26.59/0.960	27.19/0.966	**28.08/0.963**	17.86/0.878	22.97/0.927
Sand Arena	27.86/0.976	29.05/0.984	**29.86/0.982**	26.72/0.966	26.88/0.969
Skate Bunny	23.03/0.948	**24.82/0.966**	24.37/0.959	21.37/0.918	21.42/0.922
Knight	23.77/0.951	24.68/0.963	**25.04/0.958**	22.79/0.934	21.42/0.927
Umbrella	23.35/0.906	**25.43/0.951**	23.76/0.909	18.21/0.814	19.70/0.849

Resolution : $2048^2 \to 256^2$	LOD-2: $\mathbb{E}$[PSNR]$^\uparrow$/MSSSIM$^\uparrow$				
	NVDiffModeling	$\text{Ours}_{\text{UV,SH}}$	Ours	UE	Blender
Dragon Jar	23.59/0.893	25.82/0.954	**28.71/0.976**	21.69/0.848	21.80/0.856
Garden Seat	21.64/0.809	24.18/0.921	**26.24/0.955**	20.46/0.744	20.57/0.765
Helmet	23.40/0.940	25.50/0.967	**27.57/0.978**	21.08/0.905	21.35/0.914
Cat Statue	26.48/0.928	27.72/0.967	**29.52/0.980**	25.14/0.903	25.38/0.908
Chn. Chess	25.74/0.952	26.23/0.964	**29.47/0.974**	25.31/0.935	25.11/0.933
Hand Fan	27.00/0.969	28.82/0.983	**30.71/0.989**	23.70/0.934	24.48/0.947
Iron Cup	21.99/0.906	24.17/0.940	**25.51/0.962**	20.07/0.864	20.71/0.880
Cut Fish	28.04/0.973	28.79/0.981	**30.92/0.986**	26.07/0.946	26.61/0.956
Easter Egg	24.46/0.919	26.73/0.970	**29.17/0.981**	16.80/0.736	21.76/0.865
Baguette	30.03/0.985	30.32/0.988	**32.40/0.991**	29.84/0.980	29.84/0.980
Greek Vases	24.41/0.950	25.57/0.968	**28.65/0.982**	23.59/0.938	23.53/0.939
Gundam	18.74/0.906	19.53/0.927	**19.97/0.937**	17.55/0.858	17.50/0.856
Italian Car	25.52/0.967	27.75/0.981	**29.80/0.988**	22.53/0.928	20.23/0.922
Teacup	26.56/0.942	27.85/0.971	**28.73/0.976**	23.90/0.897	24.38/0.908
Jpn. Lamp	24.53/0.913	26.24/0.954	**28.08/0.965**	23.57/0.882	23.69/0.888
Lego Fig.	26.10/0.976	27.83/0.983	**29.72/0.988**	22.60/0.950	22.89/0.954
Lemon	29.75/0.977	29.94/0.985	**31.92/0.988**	28.88/0.966	29.35/0.972
Mask	26.13/0.969	27.01/0.978	**29.48/0.984**	25.81/0.962	25.82/0.965
Golem	25.43/0.946	25.95/0.962	**28.10/0.971**	24.80/0.921	24.97/0.932
White Tree	16.79/0.801	16.87/0.792	**18.01/0.819**	14.53/0.751	12.59/0.750
Pengu	26.77/0.956	27.99/0.975	**30.15/0.980**	24.55/0.913	24.90/0.930
Pony (Car)	26.86/0.965	27.77/0.973	**30.19/0.981**	17.85/0.884	22.92/0.930
Sand Arena	27.98/0.977	29.31/0.985	**31.39/0.989**	26.68/0.966	26.92/0.969
Skate Bunny	23.35/0.952	25.39/0.970	**26.79/0.977**	21.34/0.921	21.42/0.925
Knight	23.97/0.955	24.85/0.966	**26.34/0.973**	22.71/0.936	21.41/0.928
Umbrella	24.13/0.930	26.04/0.961	**27.57/0.968**	18.24/0.832	19.78/0.866

Resolution : $2048^2 \to 512^2$	LOD-2: $\mathbb{E}$[PSNR]$^\uparrow$/MSSSIM$^\uparrow$				
	NVDiffModeling	$\text{Ours}_{\text{UV,SH}}$	Ours	UE	Blender
Dragon Jar	23.85/0.902	26.21/0.961	**29.20/0.981**	21.64/0.857	21.59/0.859
Garden Seat	22.36/0.843	25.04/0.945	**27.19/0.967**	20.78/0.783	20.58/0.785
Helmet	23.67/0.944	26.01/0.971	**27.99/0.980**	21.10/0.911	21.40/0.918
Cat Statue	26.53/0.930	27.88/0.970	**29.69/0.982**	25.13/0.910	25.22/0.910
Chn. Chess	25.78/0.954	26.39/0.969	**30.21/0.981**	25.29/0.938	25.02/0.933
Hand Fan	27.03/0.969	29.14/0.984	**30.73/0.989**	23.57/0.933	24.63/0.949
Iron Cup	22.42/0.914	25.05/0.956	**27.14/0.977**	20.30/0.875	20.80/0.888
Cut Fish	28.08/0.974	28.93/0.982	**30.85/0.986**	26.05/0.947	26.59/0.957
Easter Egg	24.73/0.926	26.76/0.972	**29.82/0.988**	23.07/0.885	21.45/0.862
Baguette	30.05/0.986	30.39/0.988	**32.28/0.991**	29.84/0.980	29.86/0.984
Greek Vases	24.48/0.951	25.72/0.969	**28.67/0.983**	23.42/0.937	23.39/0.939
Gundam	18.93/0.911	20.62/0.944	**21.26/0.954**	17.53/0.859	17.34/0.862
Italian Car	25.64/0.967	28.05/0.983	**29.83/0.988**	22.38/0.928	20.14/0.922
Teacup	26.82/0.952	28.11/0.978	**28.89/0.981**	23.95/0.906	24.38/0.913
Jpn. Lamp	24.69/0.920	26.67/0.963	**28.49/0.973**	23.64/0.891	23.71/0.892
Lego Fig.	26.12/0.975	28.20/0.985	**29.87/0.989**	22.53/0.950	22.85/0.953
Lemon	29.77/0.978	29.96/0.986	**31.77/0.988**	28.85/0.966	29.34/0.972
Mask	26.17/0.969	27.10/0.978	**29.29/0.984**	25.81/0.963	25.80/0.965
Golem	25.45/0.949	26.18/0.966	**27.99/0.972**	24.74/0.922	24.93/0.932
White Tree	16.72/0.800	16.82/0.795	**17.82/0.818**	14.49/0.752	12.65/0.753
Pengu	26.89/0.960	28.23/0.978	**30.17/0.981**	24.48/0.916	24.89/0.932
Pony (Car)	26.93/0.967	27.89/0.974	**30.20/0.982**	17.79/0.886	22.83/0.929
Sand Arena	28.01/0.977	29.43/0.986	**31.17/0.988**	26.61/0.966	26.86/0.969
Skate Bunny	23.46/0.953	25.62/0.972	**27.04/0.979**	21.17/0.921	21.29/0.926
Knight	24.02/0.956	24.98/0.968	**26.37/0.974**	22.58/0.936	21.40/0.928
Umbrella	24.41/0.936	26.46/0.965	**27.91/0.972**	18.18/0.839	19.68/0.868

Table 10. Comparison of texture baking on LOD-3 using NVDiffModeling, our approach, our approach with SH and UV optimization, Unreal, and Blender.

Resolution : $2048^2 \rightarrow 128^2$	LOD-3: $\mathbb{E}$[PSNR]$^\uparrow$/MSSSIM$^\uparrow$				
	NVDiffModeling	Ours$_{UV,SH}$	Ours	UE	Blender
Dragon Jar	21.84/0.847	22.04/0.869	**24.42/0.927**	20.48/0.807	20.54/0.814
Garden Seat	19.39/0.704	20.27/0.784	**21.80/0.839**	18.65/0.647	18.50/0.655
Helmet	20.62/0.888	20.96/0.898	**22.88/0.929**	19.25/0.859	19.27/0.865
Cat Statue	24.43/0.896	**24.89/0.921**	23.68/0.905	23.13/0.866	23.41/0.874
Chn. Chess	24.12/0.926	24.65/0.939	**27.85/0.958**	23.74/0.914	23.46/0.913
Hand Fan	25.49/0.956	26.54/0.968	**27.83/0.976**	23.52/0.933	23.83/0.936
Iron Cup	19.17/0.846	19.70/0.857	**21.68/0.900**	16.71/0.776	17.33/0.794
Cut Fish	25.43/0.951	25.96/0.956	**27.28/0.955**	23.30/0.917	24.15/0.931
Easter Egg	22.19/0.861	23.86/0.921	**25.88/0.951**	14.86/0.738	17.66/0.763
Baguette	28.24/0.980	28.44/0.982	**30.69/0.987**	24.96/0.954	28.13/0.977
Greek Vases	22.54/0.927	23.27/0.943	**25.52/0.959**	20.41/0.904	21.84/0.915
Gundam	18.04/0.871	18.35/0.881	**18.69/0.897**	14.31/0.805	17.23/0.835
Italian Car	23.03/0.942	23.79/0.950	**26.28/0.969**	13.69/0.860	17.64/0.893
Teacup	23.75/0.897	**24.35/0.922**	24.01/0.918	22.48/0.865	22.59/0.864
Jpn. Lamp	21.24/0.842	21.78/0.869	**24.52/0.913**	16.45/0.780	20.39/0.816
Lego Fig.	23.55/0.957	24.54/0.964	**26.72/0.974**	21.30/0.935	21.74/0.939
Lemon	25.70/0.952	25.82/0.957	**28.58/0.971**	18.11/0.890	25.35/0.947
Mask	23.48/0.942	24.18/0.954	**26.28/0.964**	22.50/0.921	22.82/0.934
Golem	22.93/0.910	23.27/0.924	**25.04/0.934**	21.80/0.864	22.54/0.896
White Tree	16.46/0.771	16.42/0.762	**17.29/0.781**	14.30/0.725	12.10/0.720
Pengu	24.08/0.922	24.37/0.933	**26.32/0.951**	22.77/0.887	22.96/0.899
Pony (Car)	23.77/0.931	24.56/0.942	**27.57/0.962**	17.46/0.859	21.46/0.903
Sand Arena	25.48/0.960	26.43/0.970	**28.10/0.976**	24.45/0.950	24.51/0.951
Skate Bunny	21.05/0.916	22.33/0.937	**23.95/0.952**	19.64/0.886	19.49/0.888
Knight	22.02/0.924	22.84/0.939	**24.47/0.953**	20.69/0.903	19.94/0.898
Umbrella	21.13/0.859	22.10/0.897	**24.18/0.928**	17.45/0.791	18.57/0.811

Resolution : $2048^2 \rightarrow 256^2$	LOD-3: $\mathbb{E}$[PSNR]$^\uparrow$/MSSSIM$^\uparrow$				
	NVDiffModeling	Ours$_{UV,SH}$	Ours	UE	Blender
Dragon Jar	22.08/0.859	23.76/0.919	**25.09/0.943**	20.06/0.813	20.34/0.816
Garden Seat	19.91/0.745	21.11/0.832	**23.66/0.914**	18.67/0.665	18.28/0.656
Helmet	20.83/0.896	22.10/0.925	**23.89/0.946**	19.15/0.869	19.37/0.871
Cat Statue	24.58/0.905	25.32/0.939	**26.73/0.959**	23.22/0.876	23.35/0.878
Chn. Chess	24.24/0.928	24.91/0.947	**28.18/0.965**	23.75/0.915	23.44/0.913
Hand Fan	25.57/0.957	26.89/0.972	**28.15/0.979**	23.36/0.933	23.83/0.937
Iron Cup	19.34/0.852	19.79/0.862	**21.85/0.909**	16.53/0.774	17.07/0.788
Cut Fish	25.55/0.953	26.65/0.966	**28.62/0.975**	24.08/0.928	24.09/0.932
Easter Egg	22.71/0.881	24.71/0.946	**26.67/0.968**	21.57/0.839	17.42/0.760
Baguette	28.25/0.980	28.52/0.982	**30.66/0.987**	28.12/0.975	28.02/0.972
Greek Vases	22.62/0.929	23.55/0.948	**25.77/0.964**	21.83/0.915	21.73/0.914
Gundam	18.07/0.873	18.61/0.894	**19.04/0.908**	16.91/0.829	16.92/0.829
Italian Car	23.20/0.945	24.72/0.962	**26.73/0.974**	20.90/0.906	17.56/0.894
Teacup	24.02/0.912	24.66/0.941	**26.10/0.954**	22.34/0.872	22.73/0.876
Jpn. Lamp	21.42/0.854	22.38/0.893	**25.19/0.935**	20.45/0.819	20.41/0.819
Lego Fig.	23.61/0.958	25.10/0.969	**26.98/0.976**	21.34/0.936	21.70/0.940
Lemon	25.70/0.952	25.89/0.961	**28.60/0.975**	24.94/0.942	25.35/0.947
Mask	23.56/0.944	24.57/0.960	**26.33/0.967**	22.81/0.933	22.96/0.935
Golem	22.98/0.914	23.62/0.935	**25.29/0.945**	22.39/0.889	22.52/0.897
White Tree	16.42/0.773	16.38/0.757	**17.42/0.793**	14.27/0.725	12.18/0.720
Pengu	24.21/0.930	25.03/0.951	**26.64/0.959**	22.71/0.891	22.95/0.903
Pony (Car)	23.85/0.935	24.35/0.939	**27.49/0.963**	17.40/0.861	21.32/0.903
Sand Arena	25.54/0.961	26.61/0.972	**28.11/0.976**	24.38/0.950	24.53/0.951
Skate Bunny	21.20/0.919	22.81/0.944	**24.32/0.957**	19.46/0.886	19.34/0.889
Knight	22.10/0.927	23.10/0.945	**24.69/0.958**	20.52/0.902	19.95/0.898
Umbrella	21.31/0.868	22.78/0.916	**25.00/0.945**	17.36/0.796	18.46/0.815

Resolution : $2048^2 \rightarrow 512^2$	LOD-3: $\mathbb{E}$[PSNR]$^\uparrow$/MSSSIM$^\uparrow$				
	NVDiffModeling	Ours$_{UV,SH}$	Ours	UE	Blender
Dragon Jar	22.13/0.862	24.00/0.927	**25.17/0.946**	19.85/0.809	20.03/0.811
Garden Seat	20.11/0.761	22.24/0.892	**24.51/0.936**	18.21/0.655	17.80/0.644
Helmet	20.98/0.900	22.51/0.934	**23.92/0.948**	19.11/0.871	19.30/0.873
Cat Statue	24.61/0.907	25.69/0.948	**26.74/0.959**	23.06/0.876	23.16/0.876
Chn. Chess	24.29/0.931	25.23/0.954	**28.22/0.966**	23.68/0.913	23.40/0.911
Hand Fan	25.59/0.957	26.92/0.972	**27.89/0.976**	23.24/0.932	23.93/0.939
Iron Cup	19.49/0.857	21.17/0.898	**23.25/0.937**	16.33/0.770	16.83/0.782
Cut Fish	25.58/0.954	26.85/0.969	**28.62/0.976**	24.01/0.928	24.01/0.931
Easter Egg	22.83/0.886	24.75/0.949	**26.83/0.971**	21.18/0.834	17.24/0.756
Baguette	28.25/0.980	28.66/0.983	**30.32/0.986**	28.11/0.975	28.03/0.977
Greek Vases	22.65/0.930	23.66/0.950	**25.72/0.964**	21.88/0.917	21.59/0.912
Gundam	18.14/0.875	19.03/0.905	**19.64/0.920**	16.89/0.830	16.85/0.836
Italian Car	23.21/0.945	24.96/0.965	**26.68/0.974**	20.74/0.905	17.50/0.894
Teacup	24.10/0.919	25.28/0.956	**26.38/0.963**	22.61/0.892	22.69/0.879
Jpn. Lamp	21.47/0.858	22.79/0.910	**25.32/0.940**	20.53/0.822	20.31/0.816
Lego Fig.	23.63/0.958	25.13/0.969	**27.00/0.977**	21.37/0.936	21.67/0.939
Lemon	25.71/0.953	25.90/0.961	**28.34/0.974**	24.92/0.941	25.34/0.946
Mask	23.62/0.944	24.65/0.961	**26.22/0.966**	22.78/0.931	22.96/0.934
Golem	22.99/0.915	23.86/0.942	**25.17/0.945**	22.48/0.892	22.50/0.896
White Tree	16.39/0.769	16.44/0.767	**17.45/0.797**	14.22/0.722	12.30/0.724
Pengu	24.27/0.931	25.30/0.957	**26.65/0.960**	22.94/0.903	22.91/0.904
Pony (Car)	23.88/0.937	24.70/0.943	**28.38/0.972**	17.39/0.864	21.14/0.898
Sand Arena	25.54/0.961	26.69/0.973	**27.96/0.976**	24.32/0.949	24.34/0.949
Skate Bunny	21.21/0.919	23.05/0.948	**24.39/0.958**	19.30/0.888	19.20/0.888
Knight	22.11/0.927	23.28/0.948	**24.68/0.958**	20.38/0.900	19.91/0.897
Umbrella	21.38/0.870	23.09/0.924	**25.38/0.952**	17.54/0.805	18.34/0.813

Table 11. From left to right, we compare our full method, our method with only SH and vertex displacement, our method with only vertex displacement, 1 level of subdivision using NVDiffmodeling, NVDiffmodeling with Laplacian smoothing, NVDiffmodeling, parallax mapping [Kaneko et al., 2001], parallax mapping with offset limiting [Welsh et al., 2004], our 3rd order Spherical Harmonics, steep parallax mapping [McGuire and McGuire, 2005], and parallax occlusion mapping [Tatarchuk, 2006]. We note subdivision introduces additional vertices, either as pre-processing or at runtime, thus does not pose a fair computational comparison, though it is still a common tool for rendering low-poly models. Our approach remains competitive with 1 level of subdivision which effectively increases the number of triangles 4 times. POM and SPM are also optimized with the term $\frac{1}{v_z}$, which slightly outperforms NVDiffModeling.

Resolution: 512×512	LOD-3: $\mathbb{E}$[PSNR]$^\uparrow$/MSSSIM$^\uparrow$										
	Ours	Ours$_{SH,V}$	Ours$_{V}$	NV Subdiv.	NV*	NV	Parallax	PMOL	Ours$_{SH}$	SPM	POM
Dragon Jar	25.07/0.944	24.60/0.933	23.14/0.890	**25.58/0.941**	18.74/0.757	22.13/0.862	23.13/0.901	22.99/0.897	23.71/0.913	23.06/0.897	23.04/0.897
Garden Seat	24.53/0.935	24.10/0.922	21.99/0.841	**24.92/0.928**	18.49/0.685	20.12/0.762	21.78/0.868	21.41/0.846	22.21/0.874	21.70/0.864	21.65/0.862
Helmet	23.92/0.948	**24.06/0.949**	22.79/0.928	24.06/0.950	19.53/0.876	20.98/0.900	21.53/0.915	21.57/0.916	22.37/0.925	21.59/0.917	21.57/0.917
Cat Statue	26.74/0.959	26.55/0.955	25.59/0.928	**27.40/0.953**	23.27/0.864	24.61/0.907	25.29/0.940	25.17/0.934	25.35/0.929	25.23/0.937	25.25/0.937
Chn. Chess	**28.22/0.966**	27.94/0.963	26.83/0.949	27.67/0.961	23.69/0.918	24.29/0.931	24.54/0.937	24.64/0.939	25.18/0.942	24.69/0.941	24.68/0.941
Hand Fan	27.89/0.976	**27.95/0.977**	26.97/0.969	26.96/0.970	22.55/0.916	25.59/0.957	26.12/0.964	26.25/0.965	26.89/0.969	26.03/0.962	26.03/0.962
Iron Cup	23.25/0.937	23.90/0.943	22.01/0.909	**23.93/0.934**	18.46/0.833	19.49/0.857	19.98/0.870	20.22/0.876	21.25/0.901	20.14/0.876	20.16/0.877
Cut Fish	28.62/0.976	28.45/0.975	27.58/0.965	**28.68/0.976**	22.73/0.920	25.58/0.954	25.95/0.960	25.98/0.960	26.73/0.964	25.95/0.960	25.93/0.960
Easter Egg	26.83/0.971	26.70/0.969	24.74/0.929	**31.12/0.987**	23.92/0.906	22.83/0.886	24.49/0.942	24.25/0.934	24.73/0.941	24.55/0.944	24.52/0.943
Baguette	30.32/0.986	30.21/0.986	29.69/0.983	**33.00/0.992**	27.89/0.978	28.25/0.980	28.36/0.981	28.31/0.981	28.52/0.980	28.35/0.981	28.34/0.981
Greek Vases	25.72/0.964	25.42/0.960	24.54/0.946	**26.41/0.971**	21.75/0.914	22.65/0.930	22.91/0.936	22.87/0.936	23.39/0.939	22.88/0.936	22.88/0.936
Gundam	**19.64/0.920**	19.34/0.915	18.50/0.891	18.87/0.906	17.20/0.831	18.14/0.875	18.32/0.882	18.37/0.883	18.75/0.895	18.41/0.885	18.41/0.885
Italian Car	**26.68/0.974**	26.53/0.973	24.58/0.956	25.60/0.969	21.91/0.930	23.21/0.945	23.74/0.952	23.94/0.954	24.91/0.962	24.04/0.956	24.07/0.957
Teacup	26.38/0.963	26.19/0.958	25.18/0.936	**29.03/0.968**	22.13/0.864	24.10/0.919	24.56/0.937	24.77/0.941	25.32/0.949	24.86/0.944	24.86/0.943
Jpn. Lamp	**25.32/0.940**	25.17/0.937	24.13/0.914	25.19/0.935	21.73/0.861	21.47/0.858	21.75/0.873	21.81/0.875	22.57/0.893	21.81/0.877	21.80/0.876
Lego Fig.	**27.00/0.977**	26.62/0.975	25.00/0.964	26.34/0.977	21.82/0.941	23.63/0.958	23.98/0.961	24.10/0.962	24.91/0.965	24.17/0.963	24.18/0.963
Lemon	28.34/0.974	28.18/0.972	27.96/0.970	**31.81/0.986**	23.50/0.929	25.71/0.953	25.72/0.954	25.74/0.955	25.77/0.954	25.77/0.957	25.78/0.958
Mask	26.22/0.966	26.09/0.963	25.48/0.954	**27.21/0.972**	23.95/0.946	23.62/0.944	23.87/0.948	23.84/0.948	24.63/0.952	23.85/0.949	23.86/0.949
Golem	25.17/0.945	24.84/0.938	24.16/0.923	**26.10/0.954**	21.50/0.876	22.99/0.915	23.18/0.925	23.16/0.925	23.66/0.923	23.16/0.925	23.16/0.925
White Tree	17.45/0.797	**17.52/0.800**	17.31/0.809	16.92/0.814	16.46/0.770	16.39/0.769	16.59/0.772	16.48/0.770	16.89/0.776	16.56/0.762	16.63/0.765
Pengu	26.65/0.960	26.53/0.958	25.81/0.945	**26.95/0.962**	22.63/0.891	24.27/0.931	24.56/0.941	24.60/0.942	25.07/0.945	24.55/0.941	24.55/0.941
Pony (Car)	**28.38/0.972**	28.30/0.971	27.59/0.968	26.91/0.965	26.66/0.960	23.88/0.937	24.04/0.937	24.17/0.941	24.98/0.944	24.20/0.942	24.16/0.942
Sand Arena	27.96/0.976	27.78/0.974	27.23/0.969	**28.04/0.978**	23.58/0.944	25.54/0.961	25.70/0.963	25.75/0.964	26.41/0.966	25.69/0.963	25.69/0.963
Skate Bunny	**24.39/0.958**	24.10/0.955	22.86/0.940	24.01/0.959	17.64/0.848	21.21/0.919	21.75/0.929	21.75/0.929	22.61/0.937	21.74/0.929	21.72/0.929
Knight	**24.68/0.958**	24.60/0.956	23.87/0.946	24.55/0.960	20.28/0.897	22.11/0.927	22.43/0.933	22.40/0.933	23.32/0.940	22.42/0.933	22.41/0.933
Umbrella	**25.38/0.952**	25.12/0.949	23.95/0.932	24.49/0.943	21.71/0.890	21.38/0.870	21.98/0.896	21.93/0.892	22.66/0.904	21.94/0.894	21.94/0.894