Eurographics Symposium on Rendering (DL-only Track) (2021)
A. Bousseau and M. McGuire (Editors)

Appearance-Driven Automatic 3D Model Simplification

J. Hasselgren[1], J. Munkberg[1], J. Lehtinen[1,2], M. Aittala[1] and S. Laine[1]

[1]NVIDIA Research
[2]Aalto university

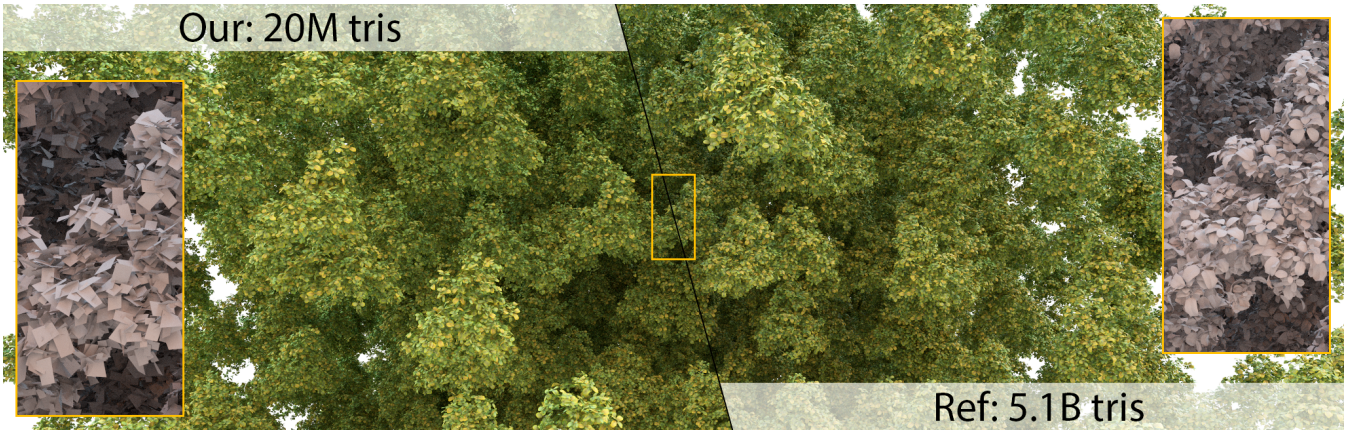


Figure 1: *We automatically approximate shape and appearance of detailed 3D scenes. Here, we approximate foliage with low-poly proxy geometry and textures, instance it 3000 times, and render the scene in a standalone path tracer with new lighting conditions. Compared to the reference, we accurately capture the appearance of the scene at just a fraction (0.4%) of the triangle count. The geometry is illustrated in the insets. The assets are from the Moana Island Scene, a publicly available data set courtesy of Walt Disney Animation Studios.*

Abstract

We present a suite of techniques for jointly optimizing triangle meshes and shading models to match the appearance of reference scenes. This capability has a number of uses, including appearance-preserving simplification of extremely complex assets, conversion between rendering systems, and even conversion between geometric scene representations.

We follow and extend the classic analysis-by-synthesis family of techniques: enabled by a highly efficient differentiable renderer and modern nonlinear optimization algorithms, our results are driven to minimize the image-space difference to the target scene when rendered in similar viewing and lighting conditions. As the only signals driving the optimization are differences in rendered images, the approach is highly general and versatile: it easily supports many different forward rendering models such as normal mapping, spatially-varying BRDFs, displacement mapping, etc. Supervision through images only is also key to the ability to easily convert between rendering systems and scene representations.

We output triangle meshes with textured materials to ensure that the models render efficiently on modern graphics hardware and benefit from, e.g., hardware-accelerated rasterization, ray tracing, and filtered texture lookups. Our system is integrated in a small Python code base, and can be applied at high resolutions and on large models. We describe several use cases, including mesh decimation, level of detail generation, seamless mesh filtering and approximations of aggregate geometry.

CCS Concepts

• ***Computing methodologies*** → *Mesh geometry models; Reflectance modeling;*

1. Introduction

Synthesizing images of objects with complex shapes and appearances is a central goal in computer graphics. The problem can be broken down into choosing suitable representations for shape and appearance, modeling the scene according to the chosen representations, and finally, rendering it efficiently.

Creating a shape and appearance model for a particular 3D scene is inherently an inverse problem: we seek a representation that will,

once fed through the renderer, result in an image that looks the way we want. Yet, most modeling tools turn the problem around: instead of providing the user with means to specify the image they want, they provide tools for editing the scene representation, leaving the modeler to iteratively proceed toward their goal.

The goal in this work is to automatically find shape and appearance representations that match, when rendered, a reference scene provided by the user. This approach is often called inverse rendering or analysis-by-synthesis. In contrast to algorithms like multi-view stereo that must make do with a small number of reference images, we focus on applications where it is possible to programmatically synthesize reference views of the target scene under arbitrary, controllable viewing and lighting conditions. Within this scope, we present inverse rendering techniques for

- **Geometric simplification (LOD).** Optimizing for the shape of a lower-resolution mesh to combat geometric aliasing and increase rendering efficiency.
- **Joint shape-appearance simplification.** Optimizing the shape and surface appearance model (mesh geometry, displacement maps, normal maps, spatially-varying BRDFs) to mimic the appearance of a more complex asset.
- **Simplification of aggregate geometry.** Dramatically simplifying complex foliage assets with little impact in visual quality.
- **Animation.** Joint optimization of shape and skinning weights on reduced geometry to match a target animation.
- **Conversion between rendering systems.** Optimizing the scene representation to match images rendered by a different system.
- **Conversion between shape representations.** Finding a mesh geometry and associated appearance model that captures the appearance of objects given by other shape representations, such as signed distance fields (SDF).

These kind of goals have been previously pursued with specialized algorithms (Section 2) that typically focus on a single task, and consider only specific parts of the object representation. Since our approach is based on inverse rendering and nonlinear optimization it easily generalizes over all the different regimes while allowing joint optimization of all aspects of the representation that affect the final appearance.

In all our applications, the search for the shape and appearance is driven by image-space error. This, combined with performing shape and appearance optimization together, has the significant benefit that the mechanisms of the forward rendering model can each specialize for the effects they capture best, "negotiating" how to achieve the desired outcome together. As an example, this leads to a natural division of labor between the geometry (mesh) and a normal map: geometric detail is allowed to move between the representations by, e.g., locally smoothing a mesh and baking geometric detail into the normal map or other parameters of a physically based shading model [Kar13]. Our source code is publicly available at `https://github.com/NVlabs/nvdiffmodeling`.

2. Previous Work

Mesh decimation For a detailed overview of this mature research topic, we refer the reader to the book by Luebke et al. [LWC*02]. Commonly used algorithms include *vertex decimation* [Sch97], *vertex clustering* [LT97] and *edge contraction* [GH97]. The error metric is typically geometry based. A notable exception is view-dependent simplification [LE97] which optimizes for silhouette quality.

Lindstrom and Turk present a framework for image-driven simplification [LT00]. They use rendered images to decide which portions of a mesh to simplify. Please refer to Corsini et al. [CLL*13] for a survey of perceptual metrics for triangle meshes. Similarly, our objective function is based on visual differences, but we leverage gradient-based optimization through differentiable rendering.

Cohen et al. [CVM*96] introduce *simplification envelopes* for generating a hierarchy of LODs for polygonal meshes. They build envelopes around the mesh to avoid self-intersections and can guarantee a distance tolerance between the original and simplified meshes. For our continuous level of detail application, we similarly use a sequence of meshes to represent the LODs. In contrast, we optimize for visual error in image space.

Cook et al. [CHPR07] introduce a decimation technique that stochastically removes a subset of the geometric elements and alter the remaining elements by, e.g., scale and contrast adjustments, in order to preserve the overall appearance. This technique is particularly suited for unstructured objects such as foliage. We similarly exploit the scene graph when approximating aggregate geometry, but instead of using heuristics for preserving visual appearance, we optimize for it directly.

Appearance prefiltering For a summary of surface appearance prefiltering techniques, please refer to Bruneton et al. [BN12]. Linear texture prefiltering methods are incorrect when applied to spatially varying BRDF (*bidirectional reflectance distribution function*) and surface normal maps. Common approaches instead filter the *normal distribution function* (NDF) [Fou92, Tok05, HSRG07, OB10], where the challenge is how to compactly represent the filtered NDF. Common representations include Gaussians [Tok05] with one or more lobes, moment statistics [OB10], and spherical harmonics [HSRG07].

Further work has considered, e.g., displacement mapping, and masking and shadowing effects [DHI*13, WZYR19]. Loubet and Neyret [LN17] prefilter meshes and materials by absorbing fine details into a volumetric microflake representation. Zhao et al. [ZWDR16] downsample volumetric materials by optimizing for rendered similarity to the original.

Our work can be applied to prefiltering. In our experiments, we restrict the shading model to one diffuse lobe and one specular GGX lobe [WMLT07], and let optimization adjust the mesh shape and the material parameters so that the rendered result matches a highly supersampled reference. This model is indeed less expressive than many of those in prefiltering literature, but has the benefit that there is no change to a typical game engine or any runtime overhead. The approach is flexible, as it treats any target surface and material representation in a unified manner: only the final visual appearance is observed, and it does not matter what combination of, e.g., mesh shape, displacement, normal, and material parameters produced it.

Appearance and geometry capture Appearance capture can be framed as seeking a digital asset (e.g., an SVBRDF map and a mesh) whose renderings visually match some real-world object. This is conceptually similar to our setup, with the exception that our targets are other digital assets.

Much of the difficulty in appearance capture originates from the desire to limit the acquisition effort for the user. Exhaustively measuring real-world appearance under all lighting and view directions is prohibitively expensive for most purposes. When only a sparse sampling is available, typical approaches employ multistage processing involving, e.g., clustering [LKG*03] or multi-view stereo [NLGK18] to find a solution that reproduces the measurements and generalizes well to unseen conditions. Many approaches use special viewing configurations and lighting patterns [GTHD03, GCP*09] to recover more varied reflectance information in each measurement. For recent surveys, see Dong [Don19], Guarnera et al. [GGG*16], and Weinmann and Klein [WK15].

We are free to render our targets in as many viewing and lighting conditions as needed. This eliminates much of the complexity, and allows us to directly end-to-end optimize over the material parameters and vertex positions using a visual similarity loss.

Many appearance capture methods can be viewed as using simple differentiable renderers to match their predictions to the observations. For example, Gao et al. [GLD*19] and Guo et al. [GSH*20] optimize for SVBRDF maps that reproduce a handful of target photographs upon rendering. They assume planar geometry, and use neural network based regularization to encourage plausible generalization to unseen conditions. Zsolnai-Féher et al. [ZFWW20] use neural nets to capture material parameters from images. Sztrajman et al. [SKWW17] convert materials between different analytic BRDF representations by optimizing for their rendered visual similarity. Similar to us, their target images are renderings of the target asset. However, they do not consider geometry as part of the optimization.

Scene acquisition with neural networks NeRF [MST*20] represents the scene as a neural radiance field. The quality of the reconstructed views are impressive, but not yet suitable for interactive applications as the radiance field needs to be densely sampled at inference time, and the generated radiance field is static. Similar to our approach, they use many observations of the scene to train the model. In contrast, we generate triangle meshes and materials, which render in real-time in a standard 3D engine. Thies et al. [TZN19] achieve a similar goal by learning a latent texture map and an associated image-space decoder CNN that allows high-quality view interpolation on meshes reconstructed by multi-view stereo in fixed lighting environments.

Kato et al. [HKH18] generate triangular meshes from a single color image by starting from a sphere and optimizing for silhouette loss, ignoring materials and fine detail such as normal maps. Pixel2Mesh [WZL*18] extend this idea by using a graph convolutional neural network to represent the mesh, and a set of geometrical losses to ensure that the mesh is well-formed, including a Laplacian regularization term. BSP-Net [CTZ20] uses a different geometrical representation based on binary space partitioning, which leads improves quality when using low polygon meshes. Similar to these approaches, our shape optimization is also based on deforming an existing triangular mesh, but we rely on multiple image observations and use image loss, rather than geometric losses, to allow higher-quality reconstruction that includes materials and shading effects.

Differentiable rendering For an overview of current approaches, please refer to Laine et al. [LHK*20]. They also introduce a flexible set of differentiable rasterization primitives which can be used together with PyTorch or TensorFlow to build custom differentiable rendering pipelines. Several differentiable rendering packages [JSL*19, VKP*19, RRN*20] provide more built-in functionality but are less customizable.

Chen et al. [CLG*19, ZCL*20] optimize over vertex positions, colors, normals, light directions and texture coordinates through a variety of lighting models, purely from 2D observations. We extend this approach to handle more complex shading models (GGX), more complex geometry, normal maps, displacement maps, and transparency, for higher visual quality. Additionally, we optimize shape and appearance jointly instead of training in stages, and show additional use cases including mesh filtering and animation. Chen et al. focus primarily on predicting 3D objects from single images, while we use multiple image observations to capture the shape and appearance, which leads to higher-quality reconstructions.

Li et al. [LADL18] present a differentiable Monte Carlo ray tracer, and Mitsuba 2 [NDVZJ19] implements a full differentiable path tracer. In this paper, we use and extend the rasterization primitives from Laine et al. [LHK*20] for quicker iteration times and ease of customization. Combining our approach with a differential path tracer would enable additional applications, but at a significantly higher computational cost. Given recent progress in differentiable rendering performance [NDSRJ20], we hope to leverage a differentible path tracer in future work.

3. Our Method

Our goal is to jointly optimize shape and material parameters to match the visual appearance of images from a reference renderer. We follow the common practice of representing the shape as a triangle mesh and using a spatially varying BRDF for materials. This ensures that our optimized representation renders directly in modern game engines and can readily exploit hardware-accelerated rasterization, ray tracing, and filtered texture-lookups.

Figure 2 outlines our method. The *latent representation* consists of a triangle mesh and a set of textures describing spatially varying material parameters from a physically based shading model During optimization, we render the latent representation using a differentiable rendering pipeline: a sequence of mesh operations, a rasterizer, and a deferred shading stage. An image-space loss is then computed between the resulting image and a target image produced by a reference renderer under identical viewing and lighting conditions. Because the rendering pipeline is fully differentiable, we can compute the gradient of the loss with respect to parameters of the latent representation, i.e., vertex positions and texture contents, and consequently optimize these to improve the visual similarity.

We iterate over a large number of image pairs with randomized

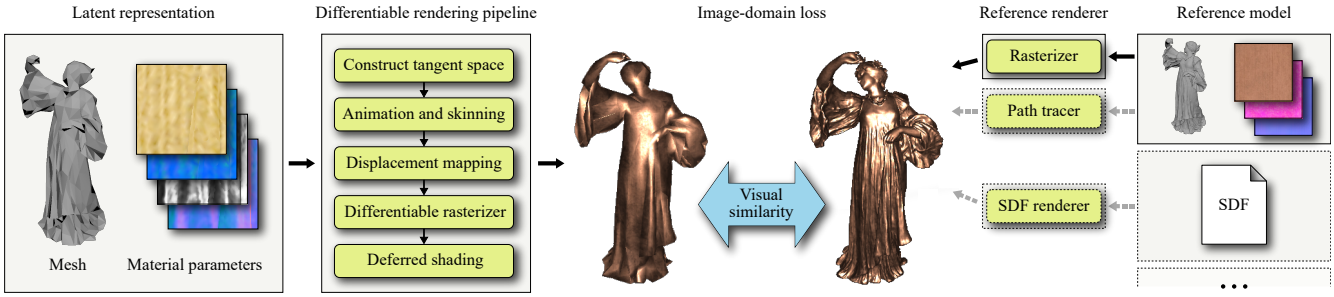


Figure 2: *An overview of our method. We render the latent representation (mesh and material parameters) in a differentiable rendering pipeline: a sequence of mesh operations followed by rasterization and deferred shading. Similarly, a target image is generated by the reference renderer under identical viewing and lighting conditions, and an image-domain loss is computed on the two images. During optimization, we iterate over a large number of image pairs with random viewing and lighting conditions. Using back-propagation and stochastic gradient descent, the latent representation is gradually morphed to produce images close to the reference.*

camera and a single randomized point light, similar to a virtual photo-goniometer. Using stochastic gradient descent, the latent representation is gradually morphed to match the appearance of the reference model.

More formally, let θ denote the parameters of our latent representation (e.g., mesh vertex positions and spatially varying material parameters). The rendered image $I_\theta(c,l)$ is a function of θ, camera, c, and light, l. The reference render is another function $I_{\text{ref}}(c,l)$, parameterized by the camera and light. Given an image space loss function L, we minimize the empirical risk

$$\underset{\theta}{\operatorname{argmin}}\ \mathbb{E}_{c,l}\left[L\big(I_\theta(c,l),I_{\text{ref}}(c,l)\big)\right] \tag{1}$$

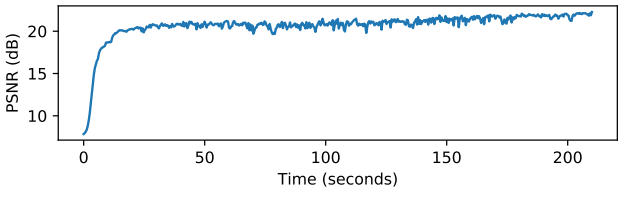
using stochastic gradient descent based on gradients w.r.t. the latent parameters, $\partial L/\partial\theta$, which are obtained through differentiable rendering.

Apart from the ability to place the camera and light, we consider the reference renderer as a black box. The only information communicated between the reference renderer and our latent representation are the target images that are used in the image-domain loss. Note, in particular, that this means that the reference renderer does not need to be differentiable, or even implemented in the same framework. This allows us to convert models across renderers, and even between different geometrical representations—e.g., from signed distance fields (SDF) to triangle meshes.

Note that the choice of reference rendering algorithm depends on the intended use of the final model. Our pipeline does not render shadows or other global effects, so if the reference renderings feature no such effects either—as can usually be arranged—the optimization will converge on materials that are as close as possible to the reference model. This is because both sides of the process in Figure 2 agree upon which effects in the final image are due to the model and which are due to the rendering algorithm. How the model is ultimately used in production is an orthogonal question, and at that point, shadows or path tracing can be enabled if desired. Alternatively, if the references are rendered with, say, ambient occlusion or path tracing enabled, the optimization process will bake these into the material parameters so that rendering *without* these effects produces a reasonable approximation. We demonstrate both approaches later in Sections 4.4 and 4.5.

As often in nonlinear optimization, good initial guesses may have a dramatic effect on the speed of convergence and eventual quality of the result. When a high-resolution mesh of the target object is available, we use off-the-shelf mesh decimation tools to produce the initial guess for the latent triangle mesh. In some cases, e.g., when baking foliage as billboard clouds, we draw on prior domain knowledge and explicitly specify a suitable initial mesh. However, we find that starting from a tessellated sphere often yields surprisingly good results. Similarly, if available, we may use texture maps from the reference scene as an initial guess for material parameters. If unavailable, we start from randomly initialized texture maps. We currently do not optimize the topology or texture coordinates of the latent representation. Hence, we require the initial mesh to have a reasonable level of tessellation and non-overlapping texture parameterization.

Our rendering pipeline combines the differentiable rasterization primitives by Laine et al. [LHK*20], differentiable shading in Cuda kernels, and mesh operations in PyTorch [PGC*17]. To encourage well-formed meshes, we include a Laplacian regularizer [Sor05] in our objective function. Each component is further detailed in Section 5 and the source code is provided for completeness. This setup is flexible and allows for differentiable rendering at high resolutions and high polygon counts at interactive rates. Please refer to Laine et al. for extensive performance comparisons of differentiable renderers. We obtain initial results with reasonable quality after a few seconds as shown in the plot below and in the accompanying video. For the final results, we run optimization at a resolution of 2048×2048 pixels, with a batch size of 8 for 10k steps, where each item uses a random camera and light position. This takes approximately 30 minutes on an NVIDIA A6000 GPU.



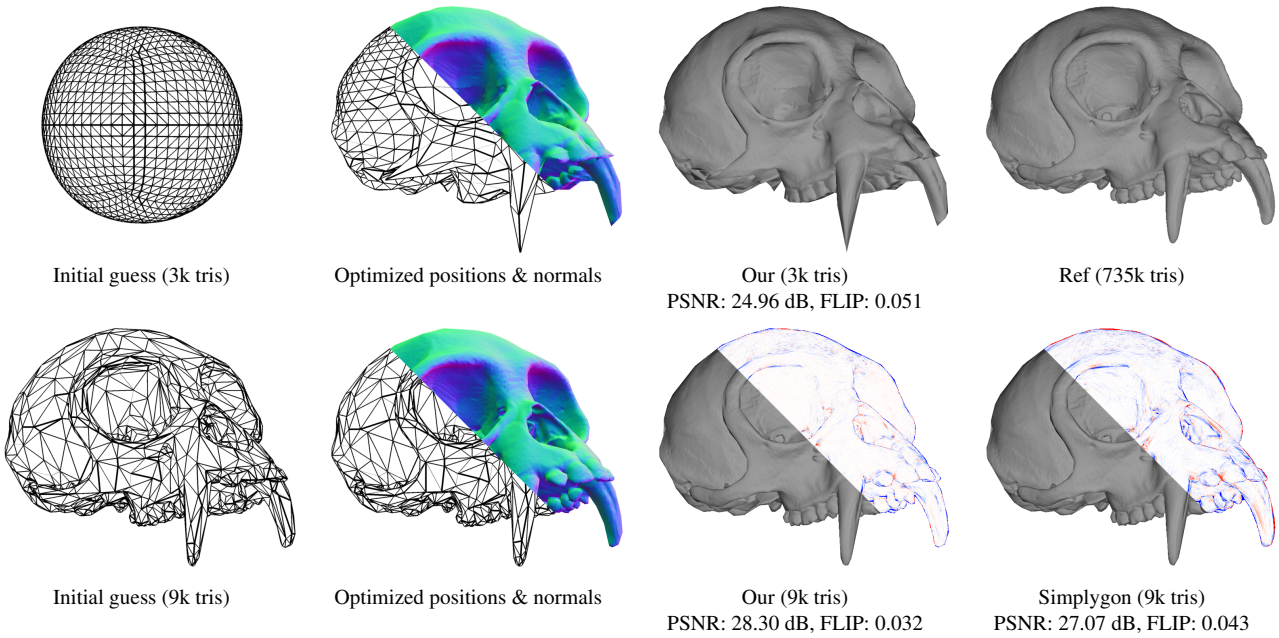


Figure 3: ***Top:*** *Starting from a sphere, we jointly optimize vertex positions and tangent space normal maps based on image observations of a reference mesh.* ***Bottom:*** *Starting from a reduced mesh with 9k triangles. We compare against the normal map baker in Simplygon by showing split images with difference images (red/blue = too bright/dim compared to reference), and through PSNR and FLIP [ANA*20] metrics. The mesh is courtesy of the Smithsonian 3D Digitization project [Smi18].*

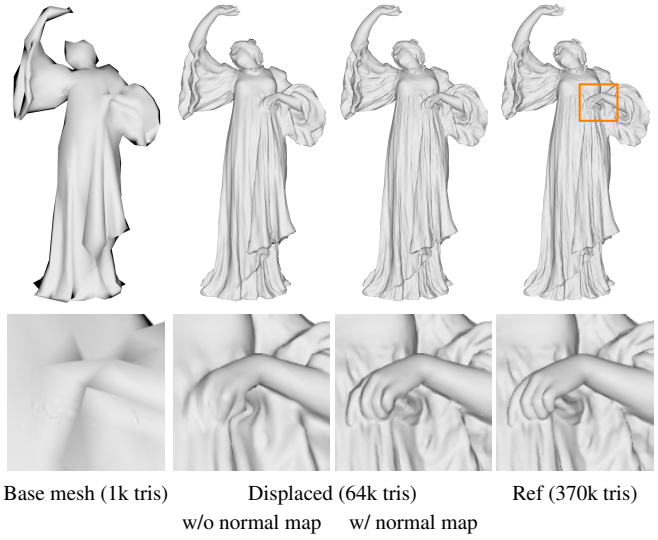


Figure 4: *We jointly optimize a base mesh, displacement map, and normal map to match the appearance of the dancer, courtesy of the Smithsonian 3D Digitization project [Smi18]. This is a complex optimization problem, with displacement constrained to the normal direction, and a coarsely tessellated base mesh. Still, the appearance after optimization matches the reference closely. Notably, some small details in the insets are baked into the normal map, even though they could be easily represented by displacement. We speculate that view parallax is minimal from the range of camera distances used during optimization, causing displacement and normal mapping to be equally viable. The initial decimated mesh was generated using the mesh simplifier in Autodesk Maya 2019.*

4. Applications

In this section, we present several use cases for our method: joint simplification of shape and appearance, prefiltering shape and appearance to reduce aliasing, geometric simplification of skinned character animation, approximation of aggregate geometry, and 3D mesh extraction from implicit surfaces. We defer discussion of implementation details until Section 5. Please refer to our interactive image viewer for detailed image comparisons.

4.1. Joint Shape-Appearance Simplification

Simplifying complex assets with minimal loss in visual fidelity is our most straightforward application. We present three variants that demonstrate joint optimization over different combinations of shape and appearance: normal map baking, joint simplification that also accounts for surface reflectance, and approximating complex meshes with displacement maps applied on a coarse base domain.

Normal map baking. As an initial example, we start from a sphere with 3k triangles and optimize shape and a tangent-space normal map to approximate a highly detailed reference mesh with 735k triangles. Besides the normals, the material is otherwise fixed as diffuse uniform gray. Our result is shown in the top row of Figure 3. While some high-frequency detail is missing, the result is nonetheless encouraging, considering that the optimization process is entirely automatic and uses no direct information about the reference model. In the bottom row of the figure, we repeat the experiment starting from a reduced version of the reference mesh with 9k triangles, as produced by Simplygon Free 8.3. As expected, this improves the results considerably, as it is now sufficient to fine-tune the geometry instead of discovering it from scratch, and we

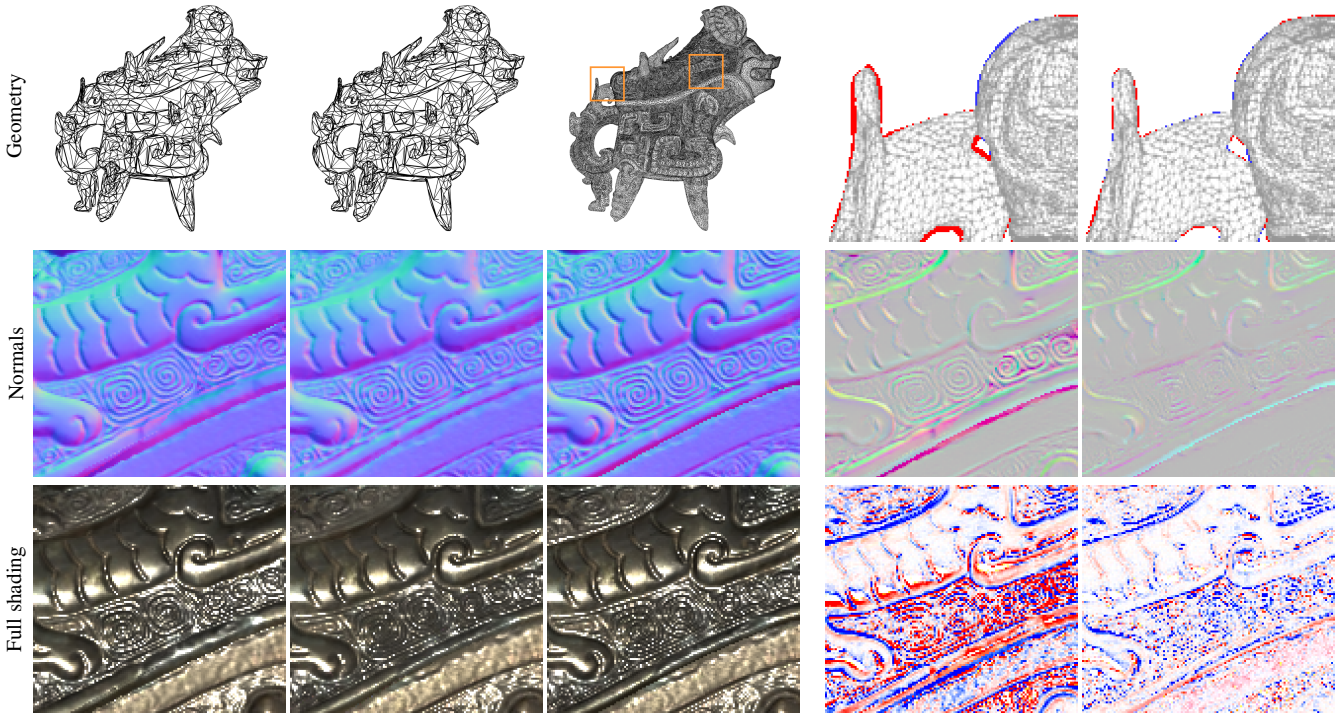


Figure 5: *Analysis of the Ewer sculpture mesh. **Top row:** Initial simplified geometry by Simplygon, our optimized geometry, and original geometry. Difference images on the right show the coverage errors of the simplified meshes overlaid on the reference mesh. Red indicates pixels covered by the simplified mesh but not the reference mesh, and blue indicates the opposite situation. As can be seen, optimizing the geometry improves the silhouettes considerably. **Middle row:** Closeups of normals generated by Simplygon, our optimized normals, and reference normals, followed by difference images. Note that our normals are not optimized directly against reference normals but discovered via optimizing the full shading result against the reference. **Bottom row:** Full shading results rendered at 1 spp with difference images (red/blue = too bright/dim compared to reference). Our result exhibits fewer overly bright pixels because the rendering is optimized to match the reference on average, thereby reducing aliasing-induced sparkling. Some of the improvement over Simplygon's output may be attributed to the use of a more versatile shading model, but that does not explain the improvement in, e.g., normals or silhouettes.*

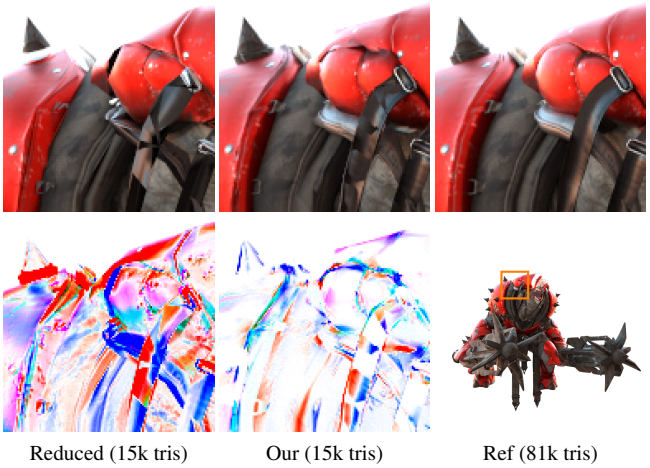


Figure 6: *A character from the Unreal Engine Paragon asset [Epi18]. We clear up some of the artifacts introduced by the automatically reduced mesh (generated in Autodesk Maya 2019), reattach geometry, and repair texture. The bottom row shows difference images (red/blue = too bright/dim compared to reference).*

slightly outperform the Simplygon normal map baker. Please refer to the supplemental material for a study of how quality is impacted by the triangle count of the initial mesh.

Displacement map baking. In addition to normal maps, displacement mapping is an increasingly popular technique for representing complex shapes in real-time settings [Epi20]. It achieves a compact representation by tessellating a coarse *base mesh* on the fly, and displacing the resulting vertices in the direction of the interpolated surface normal by amounts read from the displacement map texture.

Our approach enables using displacement maps for approximating geometry by simply implementing the tessellation and displacement steps in our forward rendering pipeline. Figure 4 shows a displacement mapped version of the dancer mesh, rendered with diffuse shading to make the geometrical impact more apparent. Our result is obtained by jointly optimizing the pre-tessellation shape of the base mesh, the normal map, and the displacement map. As shown in the insets, our process yields a natural "division of labor" between the representations: the base mesh models the overall shape, the displacement map models mid-scale detail, and the finest

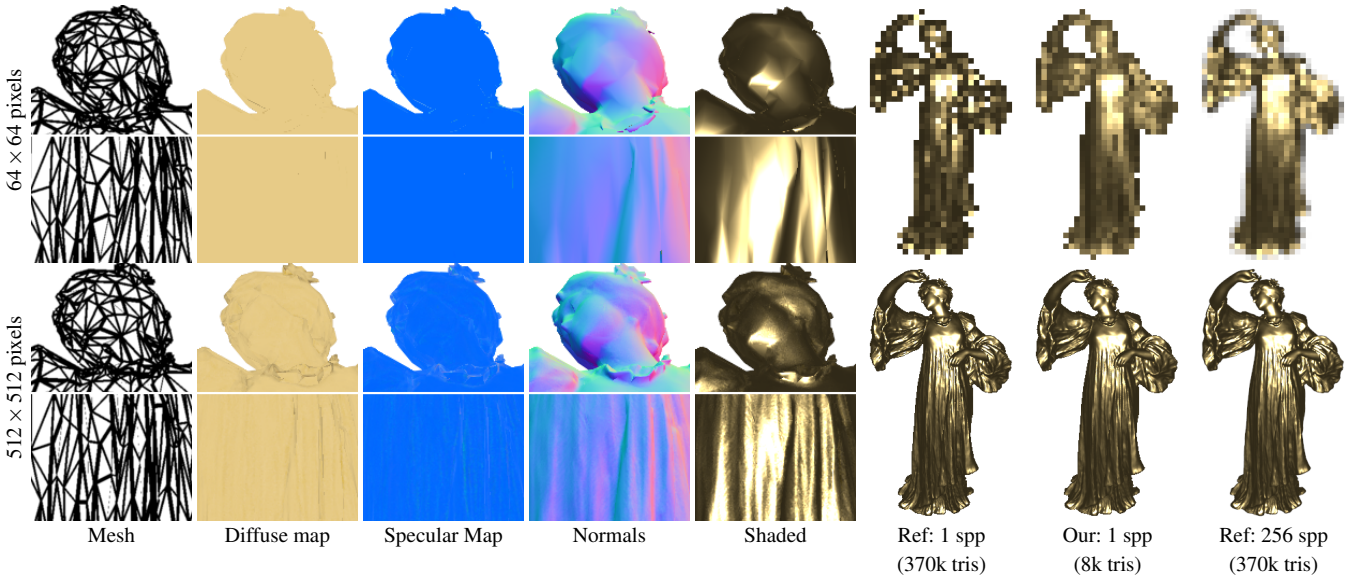


Figure 7: *We can jointly prefilter shape and appearance for a certain rendering resolution. The dancer in the **top row** is optimized for a resolution of* 64×64 *pixels, and for* 512×512 *pixels in the **bottom row**. Note in the top row how the normals are smoothed to band-limit shading. The three rightmost images show the models rendered at the intended resolution. We match the appearance of the super-sampled reference well with a single sample per pixel.*

detail that is not representable by the displaced surface is captured by the normal map.

While the above example is optimized for a single, fixed tessellation level, dynamic tessellation is often used for level of detail selection. We can additionally optimize a displacement mapped mesh to look good under multiple levels of tessellation. Please refer to the supplemental material for further results.

Simplification with complex materials. Next, we upgrade to a physically based shading model with one diffuse lobe and one isotropic GGX specular lobe [WMLT07] commonly used in modern game engines. Figure 5 shows a bronze sculpture of high geometric complexity, complex texture mapped materials, and normal maps. Here, we optimize jointly for shape and appearance under random views and point light directions. The specular term introduces higher frequencies and higher dynamic range, both of which make the optimization process more challenging. Hence, we use an image-domain loss robust to large floating-point values. Please refer to Section 5.3 for details. We compare against the reference and to the initial mesh reduced using Simplygon. Our method, optimizing based on visual differences, closely matches the true silhouette, finds accurate normals, and captures the shaded appearance of the reference. The specular highlights in our results are somewhat blurred, but it could be argued that this is preferable to the aliasing that the reference mesh shows when rendered at 1 spp.

Automatic cleanup. As a final example, Figure 6 demonstrates cleanup of the result of an unsuccessful mesh decimation operation performed in another software package. In this test, we reduced a game character from the Unreal Engine Paragon asset [Epi18] from 81k to 15k triangles in Autodesk Maya 2019. Note that the automatic reduction slightly decreases the volume of the mesh, detaches geometric elements, suffers from incorrect texturing, and produces some self-intersecting geometry. After optimization, we regain the volume and automatically clean up most of the geometry and texturing issues. We additionally show that our results generalize to different renderers by rendering the meshes in a path tracer with environment lighting and global illumination, rather than the rasterizer used during training. Please refer to the supplemental image viewer for an in depth comparison with results from both renderers.

4.2. Shape and Appearance Prefiltering

In the previous section, our goal was to create faithful representations of complex assets with reduced triangle counts. A closely related variant of this problem is to find efficiently renderable approximations to original assets that are so complex that they require a substantial amount of supersampling to produce alias-free images. We call this problem joint prefiltering of shape and appearance. The resulting optimized models have the property that they reproduce, when rendered at only one sample per pixel, the appearance of assets that require potentially hundreds of samples per pixel for alias-free reproduction. The only practical differences in the optimization are specifying a typically smaller target image resolution, and rendering the reference images with enough supersampling to ensure lack of aliasing; all previously demonstrated freedoms in choosing the shape and appearance models still apply. We address some technical challenges associated with high triangle counts in Section 5.2.

Figure 7 shows joint shape and appearance prefiltering on the golden statue, targeting rendering resolutions of 64×64 and 512×512 pixels. Note that the result prefiltered for the smaller resolution

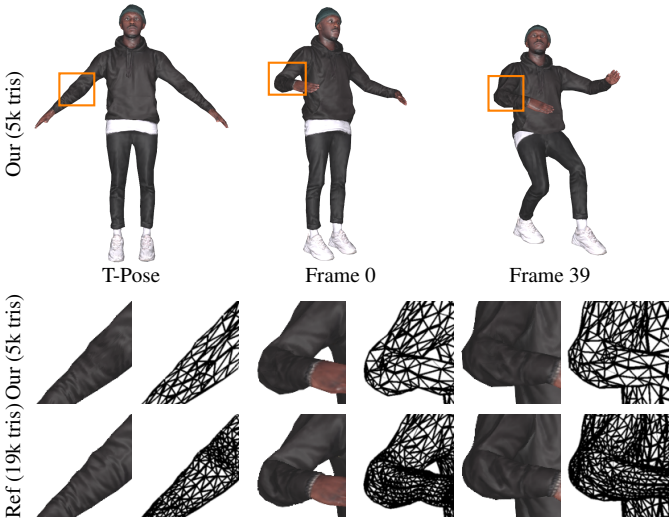


Figure 8: *Mesh decimation applied to an animated, rigged character from RenderPeople [Ren20]. We reduce the mesh and optimize vertex positions, skinning weights, normal maps, and material parameters over the animation.*

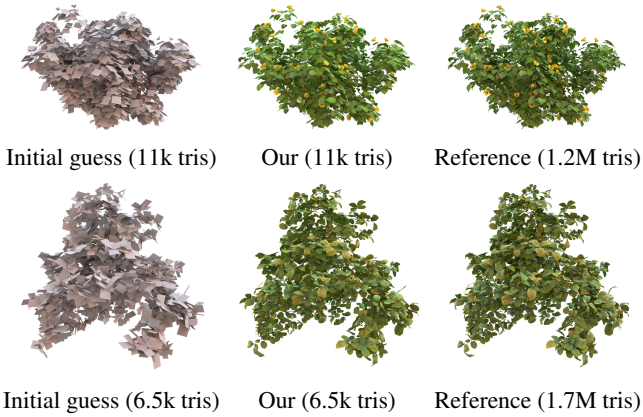


Figure 9: *Approximating aggregate geometry. We start from a low-polygon mesh and jointly optimize shape, material parameters, and transparency. The shaded results are rendered in a path tracer to illustrate that our results generalize across renderers.* ***Top row:*** *The leaves and flowers of the "isHibiscus" asset (1.2M triangles), approximated by 11k tris.* ***Bottom row:*** *The leaves from the "isGardenia" asset (1.7M triangles), approximated by 6.5k triangles. The models are taken from the Moana Island Scene [Wal18], a publicly available data set courtesy of Walt Disney Animation Studios.*

of 64×64 pixels has, as one would expect, considerably smoother normals that account, together with the specular map, for the effect of averaging present in supersampling. Also note the geometric smoothing, e.g., the flower on the statue's head. When rendered at the intended resolution, the low-resolution mesh matches the appearance of the reference well, with no apparent aliasing.

To obtain an appropriate amount of prefiltering at different rendering resolutions (i.e., rendering distances), it is not sufficient to optimize for one resolution only. For materials, this is easy to achieve by treating each mipmap level as an independent latent variable. This yields resolution-specific material representations that a trilinear texture lookup will automatically interpolate between. Even though linear interpolation between material parameters is not generally correct [BN12], the optimization process will find a representation that, assuming trilinear texture fetches will be used, matches the target images as well as possible.

For geometry, we can store multiple sets of vertex positions and choose between these based on distance to mesh, average projected edge length, or a similar heuristic. As with mipmapping, linear interpolation between levels can be used to eliminate popping artifacts. To simplify the experiments, we have opted to keep the topology fixed, so no reduction in triangle count is obtained in our tests—for geometric simplification, a mesh LOD scheme would need to be incorporated into the optimization. Please see the accompanying video for an example animation of continuous, distance-dependent prefiltering.

4.3. Animation and Skinning

Having so far focused on static scenes, we now study appearance-driven simplification of animated articulated characters over entire animation sequences. More precisely, given a high-resolution reference mesh animated by skeletal subspace deformation (SSD), we optimize over the bind-pose vertex positions, normals, SVBRDF, and skinning weights (bone-vertex attachments) of a simplified model in an attempt to replicate the appearance of the reference animation. In contrast to simplifying the character in the bind pose (T-pose) only, this holds promise for being able to strike compromises to distribute the error evenly among the frames by adjusting the geometry, skinning weights, and materials appropriately.

Implementation is straightforward: the only addition required is blending transformed vertex positions using the skinning weights, a simple linear operation. An example is shown in Figure 8, where we decimate a rigged animated mesh in a completely automated process. We include examples on rigged meshes from RenderPeople [Ren20] using skeletal animations from the CMU motion capture database [Lab20] in the accompanying video.

We assume the time-varying bone transformations are known, and treat them as constants during optimization. Joint optimization of both bone transformations and skinning weights [JT05] is a possible direction for future work. Furthermore, we assume that normal maps and SVBRDFs are constant in time. Modeling dynamic behavior like wrinkles opens another interesting future direction.

Above we use a reduced mesh as initial guess, but to thoroughly battle-test our ability to optimize skinning weights we instead start from a sphere and morph it into an animated figure with known skeletal animation, jointly optimizing shape, materials and skinning weights. Please refer to the video for results.

4.4. Approximating Aggregate Geometry

Stochastic aggregate geometry, such as foliage, are particularly difficult to simplify: as the overall appearance emerges from the com-

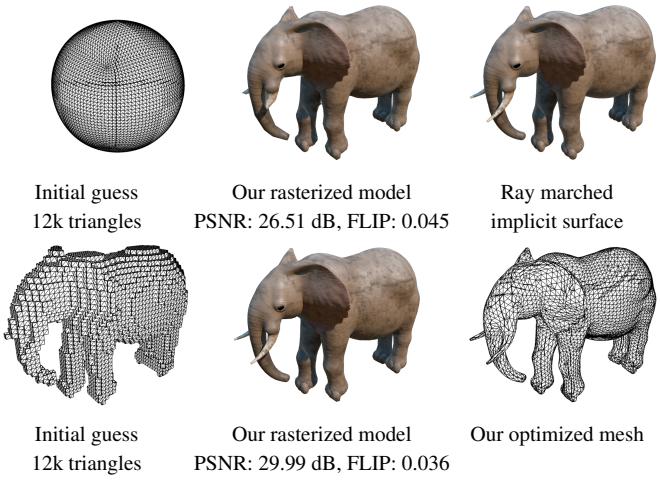


Figure 10: *We convert a ray marched implicit model, an adapted version of the ShaderToy "Elephant" ©Inigo Quilez, to a mesh with materials by optimizing for visual loss in a differentiable rasterizer. We use a tessellated sphere (12k triangles) as initial guess and jointly optimize shape and appearance. We also show the corresponding results using a better initial guess, produced by marching cubes. Note the improvements on sharp details, e.g., the tusks.*

bined effect of many small, disjoint components, techniques such as mesh decimation are ineffective.

Cook et al. [CHPR07] introduce a stochastic decimation technique that exploits a known scene graph, and randomly remove a subset of the geometric elements and alter the remaining elements, e.g., by scaling and contrast adjustments, to preserve the overall appearance of a scene. We approach the same problem from another angle, drawing inspiration from the billboard clouds of Décoret et al. [DDSD03]: instead of stochastically pruning the procedural scene graph, we replace the complex leaf geometries with textured quads. With the quads providing an initial guess, we then jointly optimize material parameters, shape, and *transparency* based on visual loss of rendered images. For this, we extended the differentiable rasterization primitives of Laine et. al [LHK*20] to support order-independent transparency through depth peeling [Eve01]. Please refer to Section 5.2 for details.

In Figure 9, we show two examples of simplification of aggregate geometry from the Disney Moana asset [Wal18]. In both cases, we create plausible approximations from only 0.8% and 0.4% of the triangles of the reference mesh, respectively. For this application, we used squared L_2 as objective function.

In Figure 1, we instance our approximation 3000× and render in a path tracer. The result closely resembles the reference with a massive reduction in geometric complexity. Our supplemental material includes a comparison with stochastic simplification [CHPR07]. Please also refer to our video for animated results.

4.5. Generalizations

All examples so far have used the same geometric representation for both the latent representation and the reference model. Moreover, the reference images have been produced by the same renderer as used for optimization. Enabled by supervising the optimization strictly in image space, we now demonstrate generalization across surface representations and rendering systems. While the full scope of potential applications is vast, we illustrate this by converting a ray marched implicit surface to a mesh. In the supplemental, we additionally include an example of transferring a path traced rendered model to a rasterizer.

Textured meshes from implicit surfaces. In Figure 10, we adapt an implicit surface pixel shader from ShaderToy [JmQ14] to isolate the main object and match the lighting and camera model of our renderer. We automatically convert this ray marched implicit surface to a triangle mesh with materials. We use a tessellated sphere as the initial guess for the geometry. For this example, we also add an ambient material term to our latent representation to better match the lighting of the implicit surface renderer, which uses custom ambient lighting. We also use a static light position, so shadowing is captured and baked into the material parameters in the optimization process. View-dependent shading effects, e.g., specular highlight, are still captured. Please refer to the supplemental material for additional results.

5. Implementation

This section covers the implementation details of our method. We first describe the differentiable mesh operations that are applied to the mesh before rendering. The second part details our differentiable renderer, and the final part provides details of the optimization process.

5.1. Mesh Operations

Referring to Figure 2, our pipeline includes differentiable mesh operations for tangent space computation, animation & skinning, and displacement mapping.

Tangent space To optimize tangent space normal maps on deforming geometry, the tangent frame must be differentiable and dynamically updated to reflect any change in vertex position. We compute smooth vertex normals and derive tangent and bi-tangent vectors from the vertex positions and texture coordinates [Mik08]. Using mesh-derived smooth normals is a not a limitation because creases or other sharp features can be handled by the normal map.

Animation & skinning We support skinning for Universal Scene Description (USD) [Pix16] meshes and rely on the USD API to evaluate skeleton animation. We optimize the skinning weights of animated meshes, and therefore implement a differentiable skinning operator according to:

$$v_i^s = \sum_{b \in \mathcal{B}} w_{ib} M_b v_i, \tag{2}$$

where $\mathcal{B}$ is the set of bones, M_b is the bone transform matrix for the current frame, and w_{ib} is the skinning weight of bone b influencing

vertex v_i. Weights are typically stored using a sparse indexed representation, but we implement the full dense skinning operator to support any vertex-bone association during optimization.

Displacement mapping Here, the latent representation consists of a coarse base mesh and a scalar displacement map. The mesh is subdivided, and the displacement map is used to displace the tessellated vertices along the interpolated normal direction. The tessellator uses edge-midpoint subdivision [WZL*18], where each triangle is split into four new triangles by inserting a new vertex along each edge. This operation changes topology, which is not a differentiable operation in our current renderer. This is simple to work around by using a tessellation criteria that does not depend on any parameters requiring gradients. For the example in Section 4.1, we select a constant tessellation factor and precompute the topology of the tessellated mesh before optimization. The position of each vertex created by the tessellation is recomputed every iteration to ensure that gradients are propagated correctly.

For displacement lookups we deploy the differentiable texture primitive of Laine et al. [LHK*20], and displace each vertex according to:

$$v_i^d = v_i + \text{tex2d}(t_i) \cdot n_i, \tag{3}$$

where v_i is the original tessellated vertex position, n_i is the interpolated normal, and t_i is the texture coordinate.

5.2. Differentiable Renderer

Our renderer is based on the differentiable rasterization primitives of Laine et al. [LHK*20]. We employ deferred shading based on a G-buffer that stores 3D position, normal, tangent, bi-tangent, and texture coordinates for each pixel.

For materials, we use a variant of the physically based model from Disney [Bur12] that is common in modern game engines [Kar13, LdR14]. This allows us to easily import game assets and lets us render our optimized meshes directly in existing engines without modifications. Material models for real-time rendering commonly combine a diffuse term with an isotropic, specular GGX lobe [WMLT07]. The parameters for the diffuse lobe k_d are provided in a four-component texture, where the optional fourth channel α represents transparency. The specular lobe is described by a roughness value r for the GGX normal distribution function and a metalness factor m which interpolates between plastic and metallic appearance by computing a specular highlight color according to $k_s = (1-m) \cdot 0.04 + m \cdot k_d$ [Kar13].

For appearance filtering, we want additional flexibility in suppressing the specular lobe. To that end, we add a parameter γ that scales the specular lobe according to: $k_s' = (1-\gamma)k_s$. Our specular parameters are thus represented by a three-component texture (γ, r, m) where each component may vary spatially. This representation is purposely chosen to resemble the commonly used ORM (occlusion, roughness, metalness) textures, where we have replaced the occlusion channel with γ.

Note that we have chosen to reparameterize a single, fixed BSDF model for appearance filtering to make results easily adoptable in game engines and to show that true prefiltering can be made a part of the asset pipeline. However, it would be straightforward to replace the BSDF with multi-lobe models [BN12], spherical harmonics representations [HSRG07], or a variant of the neural representation proposed by Müller et al. [MMR*19] for more powerful representations.

Antialiasing For the prefiltering applications in Section 4.2, we want to match the appearance of a highly supersampled target image to a 1 spp rendering of our latent representation. In practice, we experience instabilities when optimizing for low rendering resolutions due to approximations in silhouette gradient computations [LHK*20]. We work around this by rendering our latent representation using multisampled antialiasing (MSAA), i.e., we sample visibility at a higher rate but shade only once per pixel. This setup improves visibility gradients at silhouettes, but still enforces that our latent representation matches the shading of the supersampled reference with just a single shading sample.

This solution has the limitation that MSAA correctly integrates the visibility term, which is not expected for 1 spp rendering. This creates a small mismatch between our optimization setup and final rendering of the optimized model at 1 spp, but we have not found this to be an issue in practice. MSAA is only used during training, and all our result images and animations indicating 1 spp are rendered without it, i.e., they represent true 1 spp rendering without any antialiasing unless otherwise mentioned.

Order-independent transparency Transparency is required for the aggregate geometry application in Section 4.4. We implement order-independent transparency through *depth peeling* [Eve01], i.e., by rendering multiple passes where each pass peels off the front-most depth layer. We extend the rasterization primitive of Laine et al. [LHK*20] by adding a two-sided depth test to their fragment shader and passing the depth output of previous rasterization pass as a parameter to the next. While we rasterize the depth layers front-to-back, we perform blending in back-to-front order (starting with the background) as this works best with their antialiasing primitive [LHK*20]. Eight passes of depth peeling were used in our experiments.

5.3. Optimization

Objective function Our renderer uses physically based shading and produces images with high dynamic range. Therefore, the objective function must be robust to the full range of floating-point values. Following recent work in HDR image denoising [MH20], our image space loss, L_{image}, computes the L_1 norm on tone mapped colors. As tone map operator, we transform linear radiance values, x, according to $x' = \Gamma(\log(x+1))$, where $\Gamma(x)$ is the sRGB transfer function [SACM96]:

$$\begin{aligned} \Gamma(x) &= \begin{cases} 12.92x & x \leq 0.0031308 \\ (1+a)x^{1/2.4} - a & x > 0.0031308 \end{cases} \\ a &= 0.055. \end{aligned} \tag{4}$$

In addition, we use a Laplacian regularizer [Sor05] on the triangle mesh in our latent representation. This is important in the beginning of optimization to keep the mesh surface intact when

gradients are large. The uniformly-weighted differential δ_i of vertex v_i is given by $\delta_i = v_i - \frac{1}{|N_i|}\sum_{j \in N_i} v_j$, where N_i is the one-ring neighborhood of vertex v_i. We follow Laine et al. [LHK*20] and use a Laplacian regularizer term given by

$$L_\delta = \frac{1}{n}\sum_{i=1}^{n} \left\| \delta_i - \delta_i' \right\|^2, \tag{5}$$

where δ_i' is the uniformly-weighted differential of the input mesh (i.e., our initial guess). When the input mesh is a poor approximation, e.g., a sphere, we use an *absolute* regularizer and set $\delta_i' = 0$.

Our combined objective function is:

$$L_{\text{opt}} = L_{\text{image}} + \lambda_t L_\delta, \tag{6}$$

where λ_t is the regularization weight that depends on the current optimization iteration t. We gradually reduce λ_t during optimization according to $\lambda_t = (\lambda_{t-1} - \lambda_{min}) \cdot 10^{-kt} + \lambda_{min}$. Here, $k = 10^{-6}$, and λ_{min} is chosen as 2% of the initial weight, λ_0. The uniform Laplacian regularizer depends on tessellation, whereas image-domain loss does not. Hence, the image loss must be balanced against the Laplacian loss as our applications include meshes with greatly varying triangle counts. The initial weight, λ_0, can either be specifed by the user or by a simple heuristic: We evaluate the Laplacian error at the start of optimization, and set $\lambda_0 = 0.25\, L_{\text{image}}/L_\delta$, which has worked well for most of our examples. We optimize the latent representation using Adam [KB15] with default parameters. Please refer to the supplemental material for an example of how the regularizer improves mesh quality and the detailed learning rate settings.

6. Conclusions and Future Work

We have demonstrated that a wide spectrum of modeling tasks, including simplification, conversion, and prefiltering, can be achieved in a common inverse rendering framework that supports various shape and appearance models. While we show improvements in individual examples such as mesh decimation, we believe the main strength of our approach lies in the ability to jointly optimize over shape, appearance, and animation parameters. Additionally, coupling automatic differentiation and optimization makes extensions to new applications and latent representations easy.

Our method is subject to the same limitations as most other methods based on differentiable rasterization and cannot account for effects such as refractive materials, specular reflections or subsurface scattering. In practice, we note that our results generalize across renderers, as shown in Figure 1. During optimization we never modify the topology of the initial guess. If the initial guess is too crudely tessellated to reasonably capture the surface of the reference model, we note that texture details can be blurred due to view-parallax effects.

There are wide opportunities for future work. To keep a clear focus for this paper, we have only evaluated applications within the limitations of triangle meshes and a commonly used material model, so that the optimized representation can be used unmodified in existing renderers. However, an obvious line of extensions is to use our framework to augment traditional fixed-function graphics models with learned representations, effectively creating hybrids between traditional graphics and the currently popular fully learned renderers [TZN19].

We also envision that appearance-based optimization could be used in semi-automated modeling tools. When visualizing the optimization process, it is often evident where problems occur due to, e.g., under-tessellated regions or insufficient mesh genus, and an artist could clean up such areas given interactive modeling tools. Our approach is fast enough to apply in these scenarios.

References

[ANA*20] ANDERSSON P., NILSSON J., AKENINE-MÖLLER T., OSKARSSON M., ÅSTRÖM K., FAIRCHILD M. D.: FLIP: A Difference Evaluator for Alternating Images. *Proceedings of the ACM on Computer Graphics and Interactive Techniques 3*, 2 (2020), 15:1–15:23. 5

[BN12] BRUNETON E., NEYRET F.: A Survey of Non-linear Pre-filtering Methods for Efficient and Accurate Surface Shading. *IEEE Transactions on Visualization and Computer Graphics 18*, 2 (2012), 242–260. 2, 8, 10

[Bur12] BURLEY B.: Physically Based Shading at Disney. In *SIGGRAPH Courses: Practical Physically Based Shading in Film and Game Production* (2012). 10

[CHPR07] COOK R. L., HALSTEAD J., PLANCK M., RYU D.: Stochastic Simplification of Aggregate Detail. *ACM Trans. Graph. 26*, 3 (2007). 2, 9

[CLG*19] CHEN W., LING H., GAO J., SMITH E., LEHTINEN J., JACOBSON A., FIDLER S.: Learning to predict 3d objects with an interpolation-based differentiable renderer. In *Advances in Neural Information Processing Systems 32*. 2019, pp. 9609–9619. 3

[CLL*13] CORSINI M., LARABI M.-C., LAVOUÉ G., PETRÍK O., VÁSA L., WANG K.: Perceptual metrics for static and dynamic triangle meshes. *Computer Graphics Forum 32*, 1 (2013), 101–125. 2

[CTZ20] CHEN Z., TAGLIASACCHI A., ZHANG H.: BSP-Net: Generating Compact Meshes via Binary Space Partitioning. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2020). 3

[CVM*96] COHEN J., VARSHNEY A., MANOCHA D., TURK G., WEBER H., AGARWAL P., BROOKS F., WRIGHT W.: Simplification envelopes. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques* (1996), SIGGRAPH '96, pp. 119–128. 2

[DDSD03] DÉCORET X., DURAND F., SILLION F., DORSEY J.: Billboard clouds for extreme model simplification. *ACM Trans. Graph. 22*, 3 (2003). 9

[DHI*13] DUPUY J., HEITZ E., IEHL J.-C., POULIN P., NEYRET F., OSTROMOUKHOV V.: Linear Efficient Antialiased Displacement and Reflectance Mapping. *ACM Trans. Graph. 32*, 6 (2013). 2

[Don19] DONG Y.: Deep appearance modeling: A survey. *Visual Informatics 3*, 2 (2019), 59–68. 3

[Epi18] EPIC GAMES: Epic games paragon assets, 2018. https://www.unrealengine.com/en-US/paragon. 6, 7

[Epi20] EPIC GAMES: Unreal engine 5: Nanite, 2020. https://www.unrealengine.com/en-US/blog/a-first-look-at-unreal-engine-5. 6

[Eve01] EVERITT C.: Interactive Order-Independent Transparency, 2001. 9, 10

[Fou92] FOURNIER A.: *Filtering Normal Maps and Creating Multiple Surfaces*. Tech. rep., 1992. 2

[GCP*09] GHOSH A., CHEN T., PEERS P., WILSON C. A., DEBEVEC P.: Estimating Specular Roughness and Anisotropy from Second Order Spherical Gradient Illumination. *Computer Graphics Forum 28*, 4 (2009), 1161–1170. 3

[GGG*16] GUARNERA D., GUARNERA G. C., GHOSH A., DENK C., GLENCROSS M.: BRDF Representation and Acquisition. In *Proceedings of the 37th Annual Conference of the European Association for Computer Graphics: State of the Art Reports* (2016), pp. 625–650. 3

[GH97] GARLAND M., HECKBERT P. S.: Surface Simplification Using Quadric Error Metrics. In *SIGGRAPH '97: Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques* (1997), pp. 209–216. 2

[GLD*19] GAO D., LI X., DONG Y., PEERS P., XU K., TONG X.: Deep Inverse Rendering for High-Resolution SVBRDF Estimation from an Arbitrary Number of Images. *ACM Trans. Graph. 38*, 4 (2019). 3

[GSH*20] GUO Y., SMITH C., HAŠAN M., SUNKAVALLI K., ZHAO S.: MaterialGAN: Reflectance Capture Using a Generative SVBRDF Model. *ACM Trans. Graph. 39*, 6 (2020). 3

[GTHD03] GARDNER A., TCHOU C., HAWKINS T., DEBEVEC P.: Linear Light Source Reflectometry. *ACM Trans. Graph. 22*, 3 (2003), 749–758. 3

[HKH18] HIROHARU KATO Y. U., HARADA T.: Neural 3D Mesh Renderer. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2018). 3

[HSRG07] HAN C., SUN B., RAMAMOORTHI R., GRINSPUN E.: Frequency Domain Normal Map Filtering. In *ACM SIGGRAPH 2007 Papers* (2007), pp. 28–40. 2, 10

[JmQ14] JEREMIAS P., QUILEZ I.: Shadertoy: Learn to Create Everything in a Fragment Shader. In *SIGGRAPH Asia 2014 Courses* (2014). 9

[JSL*19] JATAVALLABHULA K., SMITH E., LAFLECHE J.-F., FUJI TSANG C., ROZANTSEV A., CHEN W., XIANG T., LEBAREDIAN R., FIDLER S.: Kaolin: A PyTorch Library for Accelerating 3D Deep Learning Research. *arXiv:1911.05063* (2019). 3

[JT05] JAMES D., TWIGG C.: Skinning mesh animations. *ACM Trans. Graph. 24*, 3 (2005). 8

[Kar13] KARIS B.: Real shading in unreal engine 4. *SIGGRAPH 2013 Course: Physically Based Shading in Theory and Practice* (2013). 2, 10

[KB15] KINGMA D. P., BA J.: Adam: A Method for Stochastic Optimization. In *Proceedings of the 3rd International Conference for Learning Representations* (2015). 11

[Lab20] LAB C. G.: CMU Graphics Lab Motion Capture Database, 2020. http://mocap.cs.cmu.edu/. 8

[LADL18] LI T.-M., AITTALA M., DURAND F., LEHTINEN J.: Differentiable Monte Carlo Ray Tracing through Edge Sampling. *ACM Trans. Graph. (Proc. SIGGRAPH Asia) 37*, 6 (2018), 222:1–222:11. 3

[LdR14] LAGARDE S., DE ROUSIERS C.: Moving frostbite to physically based rendering 3.0. *SIGGRAPH 2014 Course: Physically Based Shading in Theory and Practice* (2014). 10

[LE97] LUEBKE D., ERIKSON C.: View-Dependent Simplification of Arbitrary Polygonal Environments. In *SIGGRAPH '97: Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques* (1997), pp. 199–208. 2

[LHK*20] LAINE S., HELLSTEN J., KARRAS T., SEOL Y., LEHTINEN J., AILA T.: Modular primitives for high-performance differentiable rendering. *ACM Transactions on Graphics 39*, 6 (2020). 3, 4, 9, 10, 11

[LKG*03] LENSCH H. P. A., KAUTZ J., GOESELE M., HEIDRICH W., SEIDEL H.-P.: Image-Based Reconstruction of Spatial Appearance and Geometric Detail. *ACM Trans. Graph. 22*, 2 (2003), 234–257. 3

[LN17] LOUBET G., NEYRET F.: Hybrid mesh-volume LoDs for all-scale pre-filtering of complex 3D assets. *Computer Graphics Forum 36*, 2 (2017), 431–442. 2

[LT97] LOW K.-L., TAN T.-S.: Model Simplification Using Vertex-Clustering. In *Proceedings of the 1997 Symposium on Interactive 3D Graphics* (1997), pp. 75–82. 2

[LT00] LINDSTROM P., TURK G.: Image-driven simplification. *ACM Transactions on Graphics 19*, 3 (2000), 204–241. 2

[LWC*02] LUEBKE D., WATSON B., COHEN J. D., REDDY M., VARSHNEY A.: *Level of Detail for 3D Graphics*. Elsevier Science Inc., USA, 2002. 2

[MH20] MUNKBERG J., HASSELGREN J.: Neural denoising with layer embeddings. *Computer Graphics Forum 39* (2020), 1–12. 10

[Mik08] MIKKELSEN M.: Simulation of wrinkled surfaces revisited, 2008. 9

[MMR*19] MÜLLER T., MCWILLIAMS B., ROUSSELLE F., GROSS M., NOVÁK J.: Neural importance sampling. *ACM Trans. Graph. 38*, 5 (2019). 10

[MST*20] MILDENHALL B., SRINIVASAN P. P., TANCIK M., BARRON J. T., RAMAMOORTHI R., NG R.: NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *ECCV* (2020). 3

[NDSRJ20] NIMIER-DAVID M., SPEIERER S., RUIZ B., JAKOB W.: Radiative Backpropagation: An Adjoint Method for Lightning-Fast Differentiable Rendering. *ACM Trans. Graph. 39*, 4 (2020). 3

[NDVZJ19] NIMIER-DAVID M., VICINI D., ZELTNER T., JAKOB W.: Mitsuba 2: A Retargetable Forward and Inverse Renderer. *ACM Trans. Graph. 38*, 6 (2019). 3

[NLGK18] NAM G., LEE J. H., GUTIERREZ D., KIM M. H.: Practical SVBRDF Acquisition of 3D Objects with Unstructured Flash Photography. *ACM Trans. Graph. 37*, 6 (2018). 3

[OB10] OLANO M., BAKER D.: LEAN Mapping. In *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2010), pp. 181–188. 2

[PGC*17] PASZKE A., GROSS S., CHINTALA S., CHANAN G., YANG E., DEVITO Z., LIN Z., DESMAISON A., ANTIGA L., LERER A.: Automatic differentiation in PyTorch. In *NIPS-W* (2017). 4

[Pix16] PIXAR ANIMATION STUDIOS: Universal Scene Description Website, 2016. http://www.openusd.org. 9

[Ren20] RENDERPEOPLE: Renderpeople, 2020. https://renderpeople.com/3d-people/. 8

[RRN*20] RAVI N., REIZENSTEIN J., NOVOTNY D., GORDON T., LO W.-Y., JOHNSON J., GKIOXARI G.: Accelerating 3D Deep Learning with PyTorch3D. *arXiv:2007.08501* (2020). 3

[SACM96] STOKES M., ANDERSON M., CHANDRASEKAR S., MOTTA R.: A Standard Default Color Space for the Internet - sRGB, 11 1996. URL: https://www.w3.org/Graphics/Color/sRGB.html. 10

[Sch97] SCHROEDER W.: A topology modifying progressive decimation algorithm. In *In VIS '97: 8th conference on Visualization* (1997), pp. 205–212. 2

[SKWW17] SZTRAJMAN A., KŘIVÁNEK J., WILKIE A., WEYRICH T.: Image-based Remapping of Material Appearance. In *Proc. 5th Workshop on Material Appearance Modeling* (2017), pp. 5–8. 3

[Smi18] SMITHSONIAN: Smithsonian 3D Digitization, 2018. https://3d.si.edu/. 5

[Sor05] SORKINE O.: Laplacian mesh processing. In *Eurographics 2005 - State of the Art Reports* (2005). 4, 10

[Tok05] TOKSVIG M.: Mipmapping normal maps. *Journal of Graphics Tools 10*, 3 (2005), 65–71. 2

[TZN19] THIES J., ZOLLHÖFER M., NIESSNER M.: Deferred Neural Rendering: Image Synthesis Using Neural Textures. *ACM Trans. Graph. 38*, 4 (2019). 3, 11

[VKP*19] VALENTIN J., KESKIN C., PIDLYPENSKYI P., MAKADIA A., SUD A., BOUAZIZ S.: TensorFlow Graphics: Computer Graphics Meets Deep Learning. 3

[Wal18] WALT DISNEY ANIMATION STUDIOS: Moana island scene (v1.1), 2018. http://technology.disneyanimation.com/islandscene/. 8, 9

[WK15] WEINMANN M., KLEIN R.: Advances in Geometry and Reflectance Acquisition (Course Notes). In *SIGGRAPH Asia 2015 Courses* (2015). 3

[WMLT07] WALTER B., MARSCHNER S. R., LI H., TORRANCE K. E.: Microfacet Models for Refraction through Rough Surfaces. In *Proceedings of the 18th Eurographics Conference on Rendering Techniques* (2007), pp. 195–206. 2, 7, 10

[WZL*18] WANG N., ZHANG Y., LI Z., FU Y., LIU W., JIANG Y.-G.: Pixel2Mesh: Generating 3D Mesh Models from Single RGB Images. In *ECCV* (2018). 3, 10

[WZYR19] WU L., ZHAO S., YAN L.-Q., RAMAMOORTHI R.: Accurate Appearance Preserving Prefiltering for Rendering Displacement-Mapped Surfaces. *ACM Trans. Graph. 38*, 4 (2019). 2

[ZCL*20] ZHANG Y., CHEN W., LING H., GAO J., ZHANG Y., TORRALBA A., FIDLER S.: Image GANs meet Differentiable Rendering for Inverse Graphics and Interpretable 3D Neural Rendering. *arXiv:2010.09125* (2020). 3

[ZFWW20] ZSOLNAI-FEHÉR K., WONKA P., WIMMER M.: Photorealistic Material Editing Through Direct Image Manipulation. *Comput. Graph. Forum 39*, 4 (2020), 107–120. 3

[ZWDR16] ZHAO S., WU L., DURAND F., RAMAMOORTHI R.: Downsampling Scattering Parameters for Rendering Anisotropic Media. *ACM Trans. Graph. 35*, 6 (2016). 2