Jrender: An Efficient Differentiable Rendering Library based on Jittor

Hanggao Xin, Chenzhong Xiang, Wenyang Zhou, Dun Liang
Tsinghua University
Beijing, China
cjld@tsinghua.edu.cn

Abstract

Differentiable rendering has been proven as a powerful tool to bridge 2D images and 3D models. With the aid of differentiable rendering, tasks in computer vision and computer graphics could be solved more elegantly and accurately. To address challenges in the implementations of differentiable rendering methods, we present an efficient and modular differentiable rendering library named Jrender based on Jittor. Jrender supports surface rendering for 3D meshes and volume rendering for 3D volumes. Compared with previous differentiable renderers, Jrender exhibits a significant improvement in both performance and rendering quality. Due to the modular design, various rendering effects such as PBR materials shading, ambient occlusions, soft shadows, global illumination, and subsurface scattering could be easily supported in Jrender, which are not available in other differentiable rendering libraries. To validate our library, we integrate Jrender into applications such as 3D object reconstruction and NeRF, which show that our implementations could achieve the same quality with higher performance.

Keywords: differentiable rendering, real-time rendering, deep learning.

1. Introduction

For 2D image tasks, deep learning methods have a great success in image segmentation [37], image classification [28], image generation [42] and so on. Similarly, for tasks in 3D world, reconstruction [62], segmentation [17], classification [14], generation [15] of 3D models also have been improved by deep learning significantly. In these tasks, gradients could flow from 2D images or 3D models to neural network weights, guiding the neural networks to address these challenging problems. As an important building block of deep learning, differentiable rendering could generate images from 3D models and flow gradients between 2D images and 3D models. With the aid of differentiable rendering, neural networks could be guided by gradients both from 2D and 3D spaces, which brings an improvement for many applications. For example, with differentiable rendering, human faces [51, 13, 3], hand shapes and poses [11, 2] could be reconstructed or estimated from a single RGB image, which is unattainable for other deep learning methods.

However, it is not easy for researchers in the machine learning community to implement differentiable rendering algorithms. In deep learning, millions even billions of iterations are needed in the training process. To meet this requirement, the differentiable rendering part should be **efficient** to achieve real-time performance. And the system design of the differentiable rendering implementation should be **modular**. Concretely, for surface rendering, the vertex processing stage, the rasterization stage, the fragment shading stage, and the post-processing stage should be decoupled, and the support for volume rendering should also be isolated from the surface rendering pipeline. Besides that, the differentiable rendering library should be **versatile**. Abundant materials, lighting conditions, and shading effects are needed to guarantee the rendering quality. Finally, the differentiable rendering system should be **extendable**. New rendering algorithms, materials and shading effects should be easily implemented based on the original rendering system.

To solve these issues, we introduce an efficient differentiable rendering library named Jrender based on Jittor [18]. Jittor is a a new deep learning framework, which is better at training and inference for neural networks due to its special meta-operator fusion and unified graph execution. Besides that, Jittor provides useful model zoos such as JGAN [61], JNeRF [57], JDet, JSeg, JMedSeg, JSparse, and JPointCloudLib. Jrender is implemented by CUDA kernels and Jittor meta-operators to ensure the system is differentiable, efficient and robust. To make Jrender more efficient, we optimize the rendering pipeline and the rendering algorithms. Besides that, in Jittor's view, all forward and backward computations in the differentiable rendering system could be organized as a unified graph execution, which utilizes the optimization techniques such as operator fusion to make the system save the computation and memory costs at the same time. In Jrender, both differentiable surface rendering and

differentiable volume rendering are supported. PBR materials and shading effects such as soft shadows, ambient occlusions, subsurface scattering, and global illumination are also supported, which help Jrender outperform other differentiable libraries in rendering quality. Our differentiable rendering library is publicly available[1], and we hope this work could help researchers in computer vision and computer graphics communities to explore differentiable rendering in more applications.

2. Related Work

Offline differentiable rendering. Li et al. [29] first proposed a stochastic method to compute gradients of the physically-based rendered pixel with the input parameters such as lighting, geometry, and material conditions. By using Monte Carlo ray tracing in edge sampling, they could elegantly solve the continuous and discontinuous parts of the gradient integral together. Loubet et al. [33] proposed a similar way to handle the discontinuous parts in the gradient integral by using reparameterization instead of edge sampling, but the expensive Monte Carlo ray tracing is still needed in their method. Besides that, Zhang et al. [59] extended the previous methods from surface rendering to volume rendering with high quality. The greatest advantage of offline differentiable rendering methods is good quality. Gradients produced by the methods mentioned above are accurate and these methods are versatile for amazing rendering effects such as GI, SSS, etc. However, these works based on Monte Carlo ray tracing are too slow. Mitsuba [41, 40, 46, 20] is a highly optimized rendering library for offline differentiable rendering, but it is still too far from the real-time performance. For example, Mitsuba costs about 100 ms to render a simple scene with 512×512 resolution.

As a summary, the offline differentiable rendering based on ray tracing are theoretical accurate and could produce plausible results. But due to the high costs of ray tracing, the offline differentiable rendering methods are thousands times slower compared with the real-time differentiable rendering methods. Nowadays, nearly all applications [55, 19, 8, 1, 49, 16, 48, 36, 38]using differentiable rendering are based on deep learning, and they have tough demands on the performance of the differentiable rendering implementations, which could not be achieved by these offline differentiable rendering methods.

Real-time differentiable rendering. For a 3D mesh represented by triangles, the rasterization step is needed to find the closest triangle for each pixel in the image space. However, as the fundamental part of the real-time rendering pipeline, the rasterization step itself is not differentiable with all input parameters. To solve this issue, previous methods could be divided into two categories. Some researchers [32, 24, 13] try to approximate gradients of the traditional rasterization step, and the others [45, 30, 8] modify the rasterization process even the entire rendering pipeline directly to make the forward rendering process differentiable.

To approximate gradients, Loper et al. [32] first proposed a general-purpose differentiable rendering library named OpenDR by computing gradients with local differential filters. However, it only supports the local shading model(all shading should be bound as vertex attributes) and the gradients could only flow between neighboring pixels due to the filtering operations. Kato et al. [24] proposed a Neural 3D Mesh Renderer (NMR), which replaces the sudden change in rasterization with a linear interpolation change. In NMR, shading could be computed for each fragment, so NMR could support more shading models than OpenDR and is differentiable with respect to textures. By using linear interpolation for approximation, NMR has another advantage in that pixels beyond the boundary also flow gradients to vertices, which could handle the localness issue of OpenDR and avoid being trapped in the poor local minima. TF Mesh Renderer [13] is another way to overcome the discontinuity in the occlusion function. By introducing negative barycentric coordinates for triangles laying outside the pixel, they calculate gradients using barycentric coordinates from rasterization for each triangle-pixel pair. In this approximation, the local geometry of the boundary is treated as the planar. For scenes with complex geometry, TF Mesh Renderer will fail to handle the mutual occlusions between objects. As a summary, compared with other methods [23] to approximate gradients, NMR is a better choice for general purposes. We provide an optimized implementation for NMR in Jrender, which is about ten times faster than the previous official implementation [24].

Another series of works modifying the rendering pipeline to be differentiable are started from VSS [45]. Rhodin et al. represented opaque objects with Gaussian density distribution to make the visibility function differentiable, and their method works well in generative pose estimation. However, due to the Gaussian density distribution representation, VSS is hard to handle geometry with high frequency such as sharp corners, and the rendered image is blurry compared with the traditional rendering methods. DIB-R [8] deals with non-differentiable rasterization by dividing the pixels into foreground pixels and background pixels. Gradients from foreground pixels are computed using barycentric coordinates, same as Kyle's [13] way, and gradients from background pixels are approximated by the alpha channel through a distance-based aggregation. However, the alpha channel is available only when a reference alpha mask exists, and the method would fail to handle complex visibility gradients or render with environment light-

[1] https://github.com/Jittor/jrender

ing [26]. The most generally used differentiable rendering method in this category is proposed by Liu et al. [30], named SoftRas. Instead of using the deterministic way to find the closest triangle, SoftRas treated each triangle as a probabilistic cloud, and the pixel color is computed as the sum of contributions from all mesh triangles(including the hidden ones). With the probabilistic cloud representation, all operations in SoftRas are differentiable. And the gradients from pixels could flow to blocked and far triangle vertices, which is more suitable for tasks needing to refine object vertices compared with other methods [32, 24]. Pytorch3d [44] provides a highly optimized implementation for SoftRas, but we will show that Jrender has a better performance for nearly all situations in Section 4.

Real-time differentiable volume rendering has a great success in NeRF [36], which represents the 3D scene volume as neural networks for the novel view synthesis task. For high performance and good quality, ray marching is used as the key technique to solve the volume rendering equation. Following works such as Instant-NGP [38] use hash strategy and acceleration structures to improve NeRF performance, but still use ray marching as the solution of volume rendering equation.

Applications using differentiable rendering. Differentiable rendering bridges the gap between the 3D world and 2D image, which could be integrated into deep learning tasks freely. As a result, differentiable rendering has been widely used in human face/body/hand reconstruction [22, 12, 9, 27, 60, 51], general object reconstruction [55, 19, 8, 31, 43, 39], 3D scene parameters estimation [1, 48, 5], novel view synthesis [36, 38], physically-based rendering [25, 7, 50], and mesh processing [49, 16, 58].

3. Rendering Pipeline

For surface differentiable rendering, the pipeline of Jrender is illustrated in Figure 1. To make the library modular for extensions, the rendering pipeline is divided into four stages: vertex processing, rasterization, fragment shading, and post-processing, and each step in the pipeline is not coupled. Besides optimized implementations for the calculation of gradients, we also provide various shading models(including PBR materials shading, soft shadows, global illumination, and subsurface scattering), which make the rendering quality of Jrender far better than previous differentiable rendering libraries such as OpenDR [32] and Pytorch3D [44]. In Section 3.5, we will explain the differentiable volume rendering support as an extension of the Jrender library.

As explained in Section 2, due to the poor performance of the offline differentiable rendering, most applications such as deep learning works could not be implemented for this heavy burden. And the rendering quality of real-time rendering methods is close to the ray tracing rendering quality for general scenes. So, Jrender library only focus on the real-time differentiable rendering. We believe that applications in computer vision and computer graphics communities could be implemented more easily and efficiently with our optimized differentiable rendering library.

3.1. Vertex processing

The vertex processing stage is aimed to project input 3D scene vertices to 2D image space. Given camera settings, vertices v_w in world space could be converted to the camera space by viewing transform matrix $\mathbf{View}$, then the camera space vertices could be projected to the image space as v_i by projection transform matrix $\mathbf{Proj}$, which is

$$v_i = \mathbf{Proj} * \mathbf{View} * v_w. \tag{1}$$

In Equation 1, the transform matrices $\mathbf{View}$ and $\mathbf{Proj}$ are only dependent on camera settings, and all operations involved in the vertex processing stage are differentiable. After the vertex processing, the primitive culling could be used as an optional stage, which could save some computational costs for further stages.

3.2. Rasterization

After projecting triangles into the image space, rasterization is used to determine the contribution of these triangles for each pixel. As explained in Section 2, rasterization used in the graphics API such as OpenGL is not differentiable due to the discontinuity from occlusions. To handle this issue, one general method is to approximate gradients of the original rasterization stage, and the other method is to modify the rasterization stage to make itself differentiable. In Jrender, we implement two representative methods named NMR [24] and SoftRas [30], which have been proven both efficient and robust for rasterization.

3.2.1 NMR

NMR [24] preserves the traditional rasterization step for the forward pass but approximates gradients for the backward pass to handle the discontinuity. The insight of NMR is to replace the sudden change due to occlusions with linear interpolations. With this approximation, the gradients could flow from the pixel color to lightning conditions, textures, geometry vertices, and camera models. Hiroharu et al. [24] integrated the neural 3D mesh renderer into applications such as single image 3D reconstruction and 3D mesh style editing, which have proven the efficiency and generality of NMR.

It is important to note that only the closet triangle could affect the pixel color due to z-culling in NMR. As a result, the gradients from pixels will not flow to the vertices of the

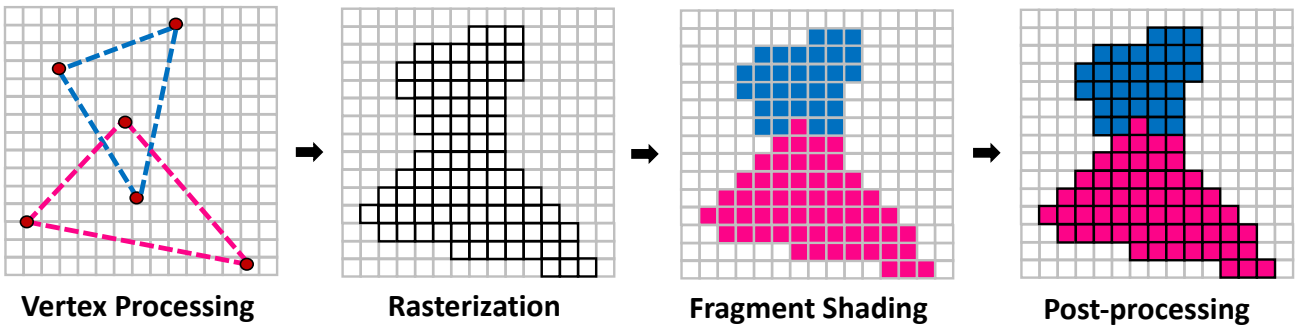

Figure 1. The overview of Jrender for surface differentiable rendering pipeline. As general rendering APIs, our method contains four stages: vertex processing, rasterization, fragment shading, and post-processing. Due to the efficient interpolation between Jittor and CUDA, our method could easily support various rasterization algorithms and shading models, which makes Jrender a general differentiable rendering library.

occluded triangles. This mechanism could save computation costs by discarding the occluded triangles, but gradients for occluded triangles could help the geometry optimization process in some applications.

3.2.2 SoftRas

SoftRas [30] is the representative method of modifying the rasterization stage to make itself differentiable. Different from the traditional method, SoftRas uses a stochastic way to determine the contribution of triangles to each pixel. By modeling triangles as probabilistic clouds with a blur radius, the contribution of each triangle is not binary for each pixel. Due to the usage of the probability map, the modified rasterization is completely differentiable. Liu et al. [30] also validated the SoftRas method in single-view mesh reconstruction and image-based shape-fitting tasks, and both achieved state-of-the-art results.

SoftRas method does not adapt the z-culling mechanism, and the triangles occluded by the closet triangle could also influence the pixel color by the aggregate function. So, the gradients from the pixel could flow to both the closet triangle and the occluded triangles. For applications optimizing the geometry vertices, gradients flowing to the occluded triangles will help the optimization converge fast.

3.3. Fragment shading

Various shading effects supporting is a key feature of Jrender. Compared with the local shading model in previous differentiable libraries [32, 44], Jrender supports PBR materials shading, ambient occlusions, soft shadows, global illumination, and subsurface scattering effects, which make the rendering results match the real world better. It is needed to mention that, due to the usage of Jittor, more shading models could be implemented easily as an extension of Jrender.

Figure 2. Rendering results of cow model with the GGX normal distribution function under different roughness settings. With the roughness increase, the highlight tends to disperse on the entire surface, which could help produce different material styles for applications.

3.3.1 PBR material

Compared with the naive shading models such as Phong and Lambertian, the PBR material is based on the microfacet theory [47] and could represent most materials in the real world. BRDF $f(\omega_i, \omega_o)$ of the PBR material is defined as:

$$f(\omega_i, \omega_o) = \frac{D(\omega_h)G(\omega_i, \omega_o)F(\omega_o)}{4\cos\theta_o \cos\theta_i}. \quad (2)$$

In Equation 2, $D(\omega_h)$ is the normal distribution function of microfacet mirrors on 3D surfaces, $G(\omega_i, \omega_o)$ is used to describe the shadowing and masking effects due to the occlusion between microfacets themselves, and the Fresnel term $F(\omega_o)$ determines the amount of reflected energy with respect to the view angle. Based on the definition above, the model appearance could be diverse from different user input parameters such as metallic, roughness, base reflectivity, etc. In Figure 2, we show the same model rendered with different roughness settings.

3.3.2 Ambient occlusion

Ambient occlusion is important in real-time rendering to approximate the indirect illumination. Occlusions between

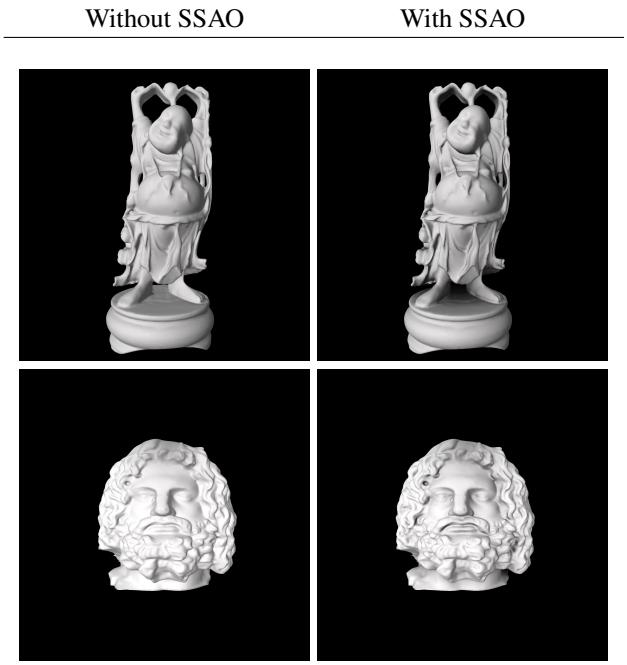

Figure 3. Comparisons between rendering results with/without SSAO. For cloth creases in Buddy and the beard in Statue, the results with SSAO could darken the illumination due to the occlusions, which makes the images more realistic.

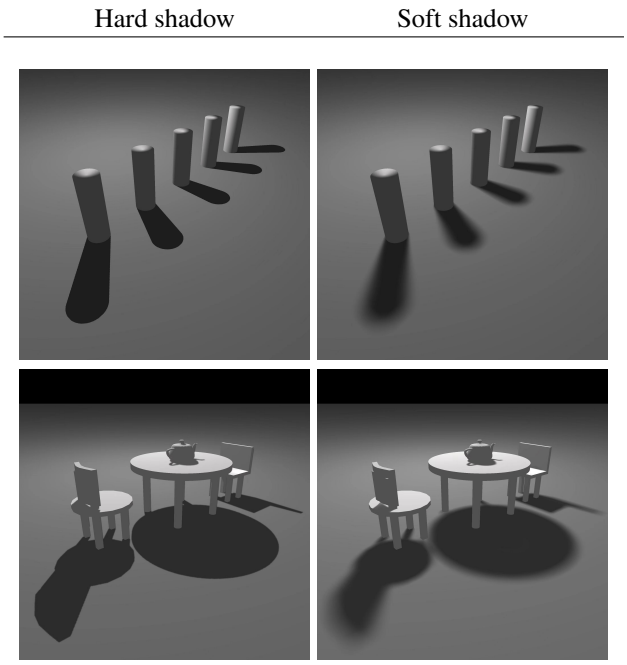

Figure 4. Comparisons between rendering results with hard shadows and soft shadows. Hard shadow effects from the shadow map are usually used with point light sources, and the soft shadows from VSM are used to enhance the image reality for the area or environment light sources.

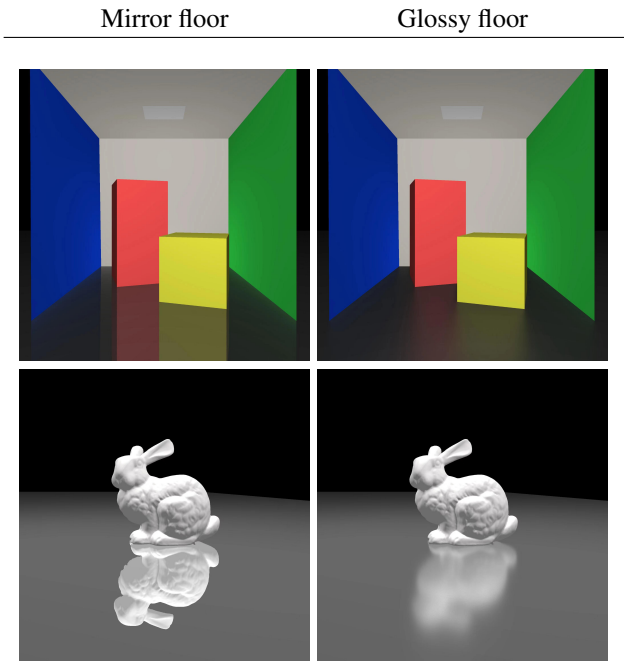

Figure 5. Rendering results with SSR for Cornell Box and Bunny. For the mirror surface, clear reflections of the objects could be rendered on the floor. And for the glossy surface, the blur reflections could also be rendered correctly with the guidance of SSR.

surfaces will darken the illumination of the scene, but this effect can not be caught by direct illumination. Ambient occlusion is used to approximate this occlusion effect and make the scene look more real, especially for objects with complex geometry such as corners and creases.

In Jrender, we provide an optimized implementation of SSAO [4] for the ambient occlusion effect. By sampling positions around the surface, the occlusion from local geometry could be approximated efficiently. As shown in Figure 3, objects rendered with SSAO have more realistic details on surfaces with complex geometry, which promotes the reality of rendering results with nearly no burden.

3.3.3 Soft shadow

Shadow rendering is also an important technique to enhance image quality. In the real world, the hard shadow is common under point lights, and the soft shadow makes a quiet difference for scenes illuminated by area lights or environment lights. Shadows are the visualization of occlusions, and the geometric relation between objects in 3D scenes might be confusing without shadows. But it is not naive to gain high-quality shadows in real-time rendering. As far as we know, Jrender is the only differentiable rendering library supporting shadow rendering.

In Jrender, we use shadow mapping for hard shadows and variance soft shadow mapping (VSSM) [56] for soft

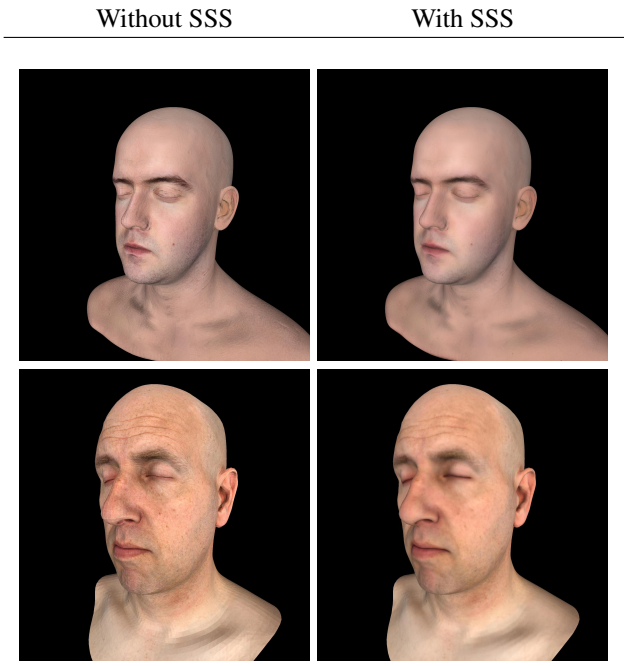

Figure 6. Rendering results with/without subsurface scattering (SSS) for two human head models. Without SSS, the appearance of human skin looks dry and plastic. With the improvement of SSS, the human skin looks more realistic, especially for the nose sides, ears, and eye sockets, which receive more illumination from other surfaces due to subsurface scattering.

shadows. In VSSM, Chebyshev's inequality is used to approximate the proportion of occlusions from the shading point view, which is efficient and could produce high-quality results as shown in Figure 4.

3.3.4 Global illumination

Global illumination is another key feature of Jrender, which could promote the rendering quality to a new level. Previous rendering libraries [32, 44] ignore the light scattering between objects and only support direct illumination. However, important features such as reflections and color bleeding could not be achieved only with direct illumination.

The most general method in real-time rendering for global illumination is screen space ray tracing (SSR) [35], which uses rays in screen space to trace against the depth buffer and achieves honorable global illumination effects. We show the rendering results with SSR in Figure 5, and we find that the global illumination is depicted well for both mirror surfaces and glossy surfaces.

3.3.5 Subsurface Scattering

For objects with translucency materials, the light entering into the object will scatter and absorb, then exit at a different position of the surface. This process is called subsurface scattering and is hard to accurately calculated in real-time rendering. But the subsurface scattering makes an important role in the appearance of translucency materials such as human skin, marble, leaves, candles, etc. Especially for applications using 3D virtual humans, the usage of subsurface scattering will improve the entire realism.

In Jrender, we achieve the subsurface scattering effects using Jorge et al.'s method [21]. They approximate the diffusion function by several Gaussian functions. By using this approximation, the contribution between each surface pair could be easily calculated as a sum of Gaussian functions. In Figure 6, we show the rendering results of human skin with/without subsurface scattering and the areas around the human nose and ears are quite different owing to the usage of subsurface scattering.

3.4. Post-processing

In the post-processing stage, per-pixel operations could be done to improve image quality. For example, to convert the illumination from the high dynamic range to the low dynamic range, the gamma correction is needed here to make the rendering results more plausible for human eyes. Another common technique used in the post-processing step to improve image quality is anti-aliasing. In Jrender, we used the fast approximate anti-aliasing (FXAA) technique to detect the jagged edges and smooth them. Compared with other anti-aliasing methods, FXAA is easy-implemented and efficient, which could also handle most aliasing issues.

3.5. Volume rendering

From Section 3.1 to Section 3.4, we have illustrated the way to render 3D meshes with versatile materials and fancy shading effects. In this section, we will explain the way Jrender used to render 3D volumes, which could be deemed as an extension of Jrender.

To solve the volume rendering equation with real-time performance, the ray marching technique is used to query and accumulate contributions from sample points. As an optimized implementation, the ray sampled with empty space or very small transmittance will be terminated early for performance. Please note that the early terminated ray will not affect the final appearance.

4. Performance Analysis

Compared with previous differentiable rendering implementations, Jrender has a great advantage in performance for nearly all situations. We selected 10 meshes with different triangle numbers from ShapeNet [6] to test ours and previous implementations. For NMR, we test the performance of the NMR implementation in Jrender and the official NMR implementation [24]. For SoftRas, besides the Jrender implementation and the official implementation [30], we also test against the Pytorch3d [44], which

Table 1. Performance comparisons between Jrender and other differentiable rendering implementations including official NMR [24], official SoftRas [30] and Pytorch3D [44].

	Scene (#triangles)	bowl (280)	house (904)	draw. (1.8k)	chair (3.3k)	bag (5k)	sink (11k)	car (39k)	cam. (69k)	airp. (111k)	bed (248k)
Forward rendering + gradient time(ms)											
Jrender (SoftRas)	256 × 256	7.0	7.3	7.4	7.5	8.1	13.5	20.7	32.6	36.9	67.4
	512 × 512	7.1	7.6	8.1	8.5	15.2	17.2	22.1	36.1	38.3	69.3
	1024 × 1024	7.3	7.9	10.0	11.5	16.7	23.6	35.5	46.7	48.9	97.2
	2048 × 2048	7.9	14.7	21.5	29.1	41.2	64.1	69.1	94.1	128.4	294.1
	4096 × 4096	24.3	47.5	67.3	97.2	151	227.8	242.1	328.9	454.5	1092
Official SoftRas	256 × 256	7.2	8.4	9.7	11.3	14.2	27.4	53.6	114.9	168.7	689.7
	512 × 512	7.6	8.8	11.7	16.7	23.9	39.9	116.2	211.9	334.5	990.1
	1024 × 1024	9.2	13.7	26.9	44.1	71.7	127.4	404.9	714.3	1130	2942
	2048 × 2048	15.6	33.2	85.1	147	240.2	462.9	1550	2745	4361	11540
	4096 × 4096	43.7	109.9	312.1	556.3	921.7	1869	6027	11346	18520	38676
Pytorch3d (SoftRas)	256 × 256	12.8	13.1	13.2	13.5	13.6	18.8	28.6	36.2	48.8	76.4
	512 × 512	20.8	22.3	22.4	22.5	23.6	23.8	38.3	46.6	56.9	90.4
	1024 × 1024	48.3	50.1	51.7	53.3	56.8	57.8	82.9	95.9	124.1	215.7
	2048 × 2048	143.1	145.4	146.2	146.3	151.2	152.7	194.7	246.7	344.1	635.9
	4096 × 4096	558.7	560.6	562.4	565.9	571.4	589.9	701.4	923.1	1272	2399
Jrender (NMR)	256 × 256	7.5	11.5	11.8	12.5	13.2	15.2	19.4	22.3	27.4	36.8
	512 × 512	11.3	20.7	28.6	29.1	33.4	36.2	48.2	58.7	64.2	75.3
	1024 × 1024	32.1	69.4	92.1	95.7	102.5	109.7	114.7	127.5	166.2	223.1
	2048 × 2048	107.7	252.1	349.9	354.5	379.3	391.7	421.3	556.5	642.1	829.9
	4096 × 4096	431.9	1005	1415	1429	1556	1581	1841	1998	2417	3345
Official NMR	256 × 256	8.1	11.8	16.9	22.3	36.3	62.9	205.9	359.7	578.1	1307
	512 × 512	12.7	22.9	40.5	65.7	105.6	198.6	664.1	1227	2043	4549
	1024 × 1024	42.3	71.1	138.4	236.9	383.2	760.5	2581	4768	7926	17765
	2048 × 2048	139.7	268.8	540.5	934.6	1515	3062	10462	19041	33426	78745
	4096 × 4096	452.5	1087	1769	3781	6362	12814	48147	76942	141967	318589
Speedup factor											
Our vs Off. Soft.	256 × 256	1.0	1.2	1.3	1.5	1.8	2.0	2.6	3.2	4.6	10.2
	512 × 512	1.1	1.2	1.4	2.0	1.6	2.3	5.3	5.9	8.7	14.3
	1024 × 1024	1.3	1.7	2.7	3.8	4.3	5.4	11.4	15.2	23.1	30.3
	2048 × 2048	2.0	2.3	4.0	5.1	5.8	7.2	22.4	29.2	33.9	39.2
	4096 × 4096	1.8	2.3	4.6	5.7	6.1	8.2	24.9	34.5	40.7	35.4
Our vs Pytorch3d	256 × 256	1.8	1.8	1.8	1.8	1.7	1.4	1.4	1.1	1.3	1.1
	512 × 512	2.9	2.9	2.8	2.6	1.6	1.4	1.7	1.3	1.5	1.3
	1024 × 1024	6.6	6.3	5.2	4.6	3.4	2.4	2.3	2.1	2.5	2.2
	2048 × 2048	18.1	9.9	6.8	5.0	3.7	2.4	2.8	2.6	2.7	2.2
	4096 × 4096	23.0	11.9	8.4	5.8	3.8	2.6	2.9	2.8	2.8	2.2
Our vs Off. NMR	256 × 256	1.1	1.0	1.4	1.8	2.8	4.1	10.6	16.0	21.0	35.2
	512 × 512	1.1	1.0	1.4	2.3	3.2	5.5	13.7	20.9	31.8	60.4
	1024 × 1024	1.3	1.0	1.5	2.5	3.7	6.9	22.5	37.4	47.7	79.6
	2048 × 2048	1.3	1.1	1.5	2.6	4.0	7.8	24.8	34.2	52.0	94.8
	4096 × 4096	1.1	1.1	1.3	2.7	4.1	8.1	26.2	38.5	58.7	95.2

provides an optimized SoftRas implementation based on Pytorch.

For fairness, the same texture, lighting condition, and camera pose are used to render each 3D mesh with different resolutions, and the user-define hyper-parameters such as the sharpness scalars in SoftRas are set as default. Both Pytorch3D and Jrender support the coarse-to-fine rasterization architecture, and the same bin size is used to test against these libraries. All tests in this section are carried out on a single NVIDIA TITAN RTX GPU card with 24GB mem-

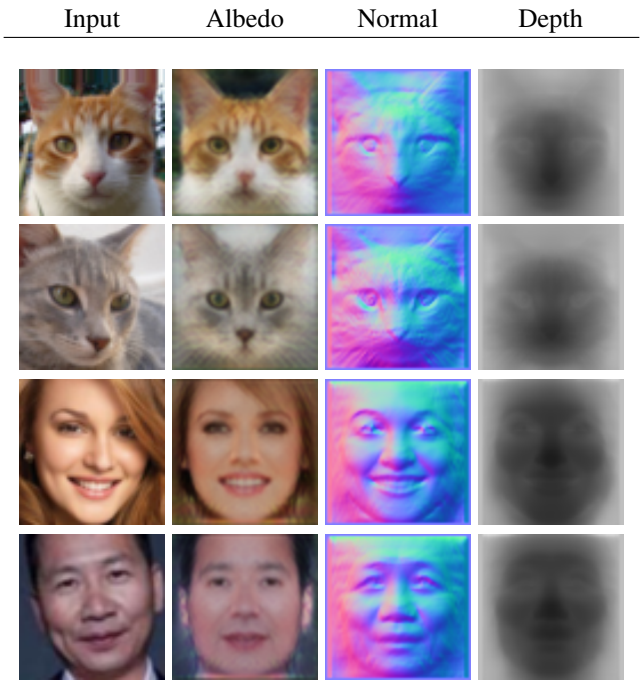

Figure 7. Reconstructed albedo maps, normal maps, and depth maps from our implementation. The first two test cases are from the Cat Head dataset and the last two test cases are from the CelebA face dataset.

ory.

As shown in Table 1, Jrender has a much better performance than other implementations. With the increase in triangle counts and resolutions, the official NMR and official SoftRas are quickly slower than Pytorch3D and Jrender. Due to the coarse-to-fine mechanism, Pytorch3D and Jrender are more suitable for large scenes and high resolutions. And the advantages of Jittor such as just-in-time compilation, automatic operator fusion, and higher computational graph processing efficiency make Jrender more efficient both in memory accessing and computing, which makes Jrender a better performance.

5. Applications

5.1. Unsupervised 3D objects reconstruction

As explained in Section 1, differentiable rendering is the bridge between 3D objects and 2D images. With the aid of differentiable rendering, Wu et al. [51] proposed an unsupervised method to reconstruct 3D objects from single-view images, which outperforms other methods and inspires many other important works.

In Wu's method, the only input is single-view images. In the training process, the neural networks are trained to factor out the 3D scene features such as albedo, depth, camera pose, and the lighting condition for each input image. Based on the symmetric assumption, the 3D scene features could be passed to the differentiable rendering part to reconstruct 2D images. As a result, gradients from losses defined in image space could flow to the predicted 3D scene features, and gradients of predicted 3D scene features 3D will finally pass to the neural networks weights.

	SIDE ($\times 10^{-2}$) $\downarrow$	MAD (deg.) $\downarrow$
Official implementation	0.793 ± 0.140	16.51 ± 1.56
Our implementation	$\mathbf{0.769 \pm 0.136}$	$\mathbf{15.99 \pm 1.49}$

Table 2. Comparisons of our implementation and the official version. The reconstruction errors on the BFM dataset with SIDE and MAD metrics (lower SIDE and MAD indicate better results).

As the validation, we re-implement Wu's method based on Jrender. Compared with the official implementation, ours is much more efficient and could reproduce the results presented in their paper as shown in Figure 7. In Table 2, we show the SIDE and MAD errors of our implementation and the official implementation, and ours is even better than the official one.

With the acceleration of Jrender, the performance of our implementation is 1.3 times faster than the official version. Concretely, the differentiable rendering part is 6.2 times faster than the official implementation, but the neural networks part is the bottleneck for the entire process, which takes up more than 70% computations.

5.2. Neural radiance fields

Neural radiance fields (NeRF) [36] are new representations of 3D scenes, which have great success in the novel view synthesis task. NeRF represents the 3D scenes as volumes instead of meshes, and the volumes are stored in neural networks.

In NeRF, ray marching is still the core technique to solve the volume rendering equation, and the only difference is that queries for volumes are calculated by the neural networks. We integrate the volume rendering part of Jrender into NeRF, and we find that our results could achieve the same quality as the official NeRF. As shown in Figure 8, our results of both synthesized scenes and real scenes are nearly as same as the official results, which validates the accuracy of differentiable volume rendering in Jrender.

Due to the just-in-time compilation mechanism and automatic operator fusion in Jittor, NeRF based on Jrender is 1.4 times faster than the official version [36]. We also found that the training speed could gain another 30% improvement by using float16 type instead of float32 type. Based on Jrender, researchers have done much effort to implement the NeRF library named JNeRF [57], which provides more optimized NeRF models for the computer graphics community.

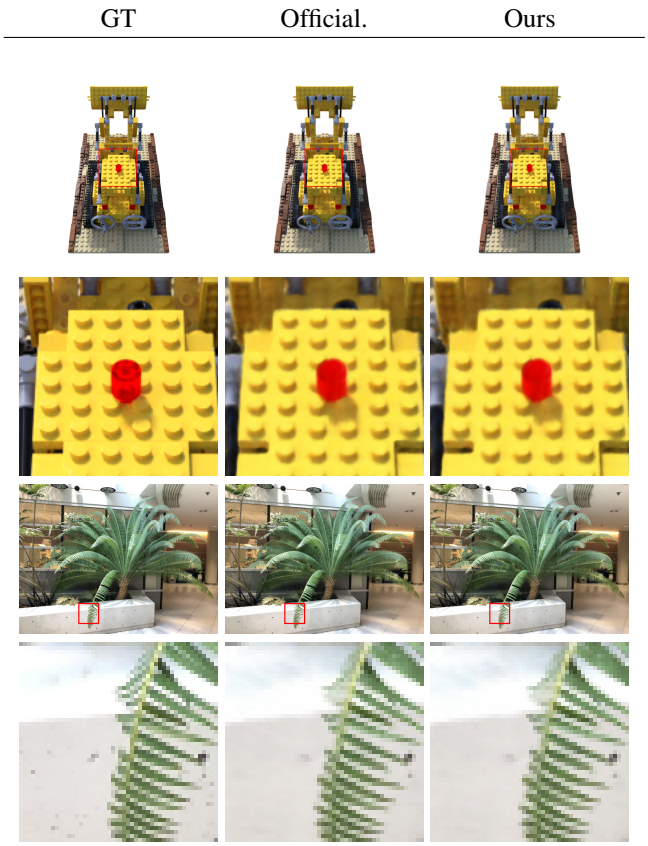

Figure 8. Comparisons between results of our implementation based on Jrender and the official version. For both the synthesized scene(Lego) and the real scene(Fern), our results could achieve the same quality as the official results.

6. Conclusion and Discussions

Conclusion. As a summary, we present an efficient and modular differentiable rendering library Jrender. With the reasonable and extendable system design, Jrender supports various rasterization methods, fancy shading effects, and versatile representatives of 3D scenes, and has a better performance and higher quality. We hope Jrender could accelerate and inspire more research in computer graphics and computer visions.

Limitations and Future Works. Compared with other differentiable rendering libraries, Jrender could support more fantastic rendering effects. But Jrender still can not support some rendering effects such as hair rendering [54] , glints rendering [53] or vegetation rendering [10]. As an evolved open-source project, we will support many other differentiable rendering methods such as nvdiffrast [26], fancy real-time shading effects mentioned above, and the SDF rendering. Besides that, Jrender will also provide useful API and data structures to efficiently implement rendering methods based on SH [52] and probes [34] in the future.

References

[1] D. Azinovic, T.-M. Li, A. Kaplanyan, and M. Nießner. Inverse path tracing for joint material and lighting estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2447–2456, 2019. 2, 3

[2] S. Baek, K. I. Kim, and T.-K. Kim. Pushing the envelope for rgb-based dense 3d hand pose estimation via neural rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1067–1076, 2019. 1

[3] M. Bao, M. Cong, S. Grabli, and R. Fedkiw. High-quality face capture using anatomical muscles. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10802–10811, 2019. 1

[4] L. Bavoil and M. Sainz. Screen space ambient occlusion. *NVIDIA developer information: http://developers. nvidia. com*, 6(2), 2008. 5

[5] D. Beker, H. Kato, M. A. Morariu, T. Ando, T. Matsuoka, W. Kehl, and A. Gaidon. Monocular differentiable rendering for self-supervised 3d object detection. In *European Conference on Computer Vision*, pages 514–529. Springer, 2020. 3

[6] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015. 6

[7] C. Che, F. Luan, S. Zhao, K. Bala, and I. Gkioulekas. Towards learning-based inverse subsurface scattering. In *2020 IEEE International Conference on Computational Photography (ICCP)*, pages 1–12. IEEE, 2020. 3

[8] W. Chen, H. Ling, J. Gao, E. Smith, J. Lehtinen, A. Jacobson, and S. Fidler. Learning to predict 3d objects with an interpolation-based differentiable renderer. *Advances in Neural Information Processing Systems*, 32, 2019. 2, 3

[9] Y. Deng, J. Yang, S. Xu, D. Chen, Y. Jia, and X. Tong. Accurate 3d face reconstruction with weakly-supervised learning: From single image to image set. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 0–0, 2019. 3

[10] Z. Fan, H. Li, K. Hillesland, and B. Sheng. Simulation and rendering for millions of grass blades. In *Proceedings of the 19th symposium on interactive 3D graphics and games*, pages 55–60, 2015. 9

[11] L. Ge, Z. Ren, Y. Li, Z. Xue, Y. Wang, J. Cai, and J. Yuan. 3d hand shape and pose estimation from a single rgb image. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10833–10842, 2019. 1

[12] B. Gecer, S. Ploumpis, I. Kotsia, and S. Zafeiriou. Ganfit: Generative adversarial network fitting for high fidelity 3d face reconstruction. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1155–1164, 2019. 3

[13] K. Genova, F. Cole, A. Maschinot, A. Sarna, D. Vlasic, and W. T. Freeman. Unsupervised training for 3d morphable model regression. In *Proceedings of the IEEE Conference*

on Computer Vision and Pattern Recognition, pages 8377–8386, 2018. 1, 2

[14] D. Griffiths and J. Boehm. A review on deep learning techniques for 3d sensed data classification. *Remote Sensing*, 11(12):1499, 2019. 1

[15] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, and M. Bennamoun. Deep learning for 3d point clouds: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 43(12):4338–4364, 2020. 1

[16] J. Hasselgren, J. Munkberg, J. Lehtinen, M. Aittala, and S. Laine. Appearance-driven automatic 3d model simplification. *arXiv preprint arXiv:2104.03989*, 2021. 2, 3

[17] Y. He, H. Yu, X. Liu, Z. Yang, W. Sun, Y. Wang, Q. Fu, Y. Zou, and A. Mian. Deep learning based 3d segmentation: A survey. *arXiv preprint arXiv:2103.05423*, 2021. 1

[18] S.-M. Hu, D. Liang, G.-Y. Yang, G.-W. Yang, and W.-Y. Zhou. Jittor: a novel deep learning framework with meta-operators and unified graph execution. *Science China Information Sciences*, 63(12):222103:1–21, 2020. 1

[19] E. Insafutdinov and A. Dosovitskiy. Unsupervised learning of shape and pose with differentiable point clouds. *Advances in neural information processing systems*, 31, 2018. 2, 3

[20] W. Jakob, S. Speierer, N. Roussel, and D. Vicini. Dr. jit: a just-in-time compiler for differentiable rendering. *ACM Transactions on Graphics (TOG)*, 41(4):1–19, 2022. 2

[21] J. Jimenez, V. Sundstedt, and D. Gutierrez. Screen-space perceptual rendering of human skin. *ACM Transactions on Applied Perception (TAP)*, 6(4):1–15, 2009. 6

[22] A. Kanazawa, M. J. Black, D. W. Jacobs, and J. Malik. End-to-end recovery of human shape and pose. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7122–7131, 2018. 3

[23] H. Kato, D. Beker, M. Morariu, T. Ando, T. Matsuoka, W. Kehl, and A. Gaidon. Differentiable rendering: A survey. *arXiv preprint arXiv:2006.12057*, 2020. 2

[24] H. Kato, Y. Ushiku, and T. Harada. Neural 3d mesh renderer. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3907–3916, 2018. 2, 3, 6, 7

[25] P. Khungurn, D. Schroeder, S. Zhao, K. Bala, and S. Marschner. Matching real fabrics with micro-appearance models. *ACM Trans. Graph.*, 35(1):1–1, 2015. 3

[26] S. Laine, J. Hellsten, T. Karras, Y. Seol, J. Lehtinen, and T. Aila. Modular primitives for high-performance differentiable rendering. *ACM Transactions on Graphics (TOG)*, 39(6):1–14, 2020. 3, 9

[27] G.-H. Lee and S.-W. Lee. Uncertainty-aware mesh decoder for high fidelity 3d face reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6100–6109, 2020. 3

[28] S. Li, W. Song, L. Fang, Y. Chen, P. Ghamisi, and J. A. Benediktsson. Deep learning for hyperspectral image classification: An overview. *IEEE Transactions on Geoscience and Remote Sensing*, 57(9):6690–6709, 2019. 1

[29] T.-M. Li, M. Aittala, F. Durand, and J. Lehtinen. Differentiable monte carlo ray tracing through edge sampling. *ACM Transactions on Graphics (TOG)*, 37(6):1–11, 2018. 2

[30] S. Liu, T. Li, W. Chen, and H. Li. Soft rasterizer: A differentiable renderer for image-based 3d reasoning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7708–7717, 2019. 2, 3, 4, 6, 7

[31] S. Liu, S. Saito, W. Chen, and H. Li. Learning to infer implicit surfaces without 3d supervision. *Advances in Neural Information Processing Systems*, 32, 2019. 3

[32] M. M. Loper and M. J. Black. Opendr: An approximate differentiable renderer. In *European Conference on Computer Vision*, pages 154–169. Springer, 2014. 2, 3, 4, 6

[33] G. Loubet, N. Holzschuch, and W. Jakob. Reparameterizing discontinuous integrands for differentiable rendering. *ACM Transactions on Graphics (TOG)*, 38(6):1–14, 2019. 2

[34] Z. Majercik, T. Müller, A. Keller, D. Nowrouzezahrai, and M. McGuire. Dynamic diffuse global illumination resampling. In *ACM SIGGRAPH 2021 Talks*, pages 1–2. 2021. 9

[35] M. McGuire and M. Mara. Efficient gpu screen-space ray tracing. *Journal of Computer Graphics Techniques (JCGT)*, 3(4):73–85, 2014. 6

[36] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021. 2, 3, 8

[37] S. Minaee, Y. Y. Boykov, F. Porikli, A. J. Plaza, N. Kehtarnavaz, and D. Terzopoulos. Image segmentation using deep learning: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 2021. 1

[38] T. Müller, A. Evans, C. Schied, and A. Keller. Instant neural graphics primitives with a multiresolution hash encoding. *arXiv preprint arXiv:2201.05989*, 2022. 2, 3

[39] M. Niemeyer, L. Mescheder, M. Oechsle, and A. Geiger. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3504–3515, 2020. 3

[40] M. Nimier-David, S. Speierer, B. Ruiz, and W. Jakob. Radiative backpropagation: an adjoint method for lightning-fast differentiable rendering. *ACM Transactions on Graphics (TOG)*, 39(4):146–1, 2020. 2

[41] M. Nimier-David, D. Vicini, T. Zeltner, and W. Jakob. Mitsuba 2: A retargetable forward and inverse renderer. *ACM Transactions on Graphics (TOG)*, 38(6):1–17, 2019. 2

[42] X. Ning, F. Nan, S. Xu, L. Yu, and L. Zhang. Multi-view frontal face image generation: a survey. *Concurrency and Computation: Practice and Experience*, page e6147, 2020. 1

[43] F. Petersen, A. H. Bermano, O. Deussen, and D. Cohen-Or. Pix2vex: Image-to-geometry reconstruction using a smooth differentiable renderer. *arXiv preprint arXiv:1903.11149*, 2019. 3

[44] N. Ravi, J. Reizenstein, D. Novotny, T. Gordon, W.-Y. Lo, J. Johnson, and G. Gkioxari. Accelerating 3d deep learning with pytorch3d. *arXiv preprint arXiv:2007.08501*, 2020. 3, 4, 6, 7

[45] H. Rhodin, N. Robertini, C. Richardt, H.-P. Seidel, and C. Theobalt. A versatile scene model with differentiable visibility applied to generative pose estimation. In *Proceedings*

of the IEEE International Conference on Computer Vision, pages 765–773, 2015. 2

[46] D. Vicini, S. Speierer, and W. Jakob. Differentiable signed distance function rendering. *ACM Transactions on Graphics (TOG)*, 41(4):1–18, 2022. 2

[47] B. Walter, S. R. Marschner, H. Li, and K. E. Torrance. Microfacet models for refraction through rough surfaces. *Rendering techniques*, 2007:18th, 2007. 4

[48] H. Wang, S. Sridhar, J. Huang, J. Valentin, S. Song, and L. J. Guibas. Normalized object coordinate space for category-level 6d object pose and size estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2642–2651, 2019. 2, 3

[49] J. Wu, J. B. Tenenbaum, and P. Kohli. Neural scene de-rendering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 699–707, 2017. 2, 3

[50] L. Wu, G. Cai, R. Ramamoorthi, and S. Zhao. Differentiable time-gated rendering. *ACM Transactions on Graphics (TOG)*, 40(6):1–16, 2021. 3

[51] S. Wu, C. Rupprecht, and A. Vedaldi. Unsupervised learning of probably symmetric deformable 3d objects from images in the wild. In *CVPR*, 2020. 1, 3, 8

[52] H. Xin, Z. Zhou, D. An, L.-Q. Yan, K. Xu, S.-M. Hu, and S.-T. Yau. Fast and accurate spherical harmonics products. *ACM Transactions on Graphics (TOG)*, 40(6):1–14, 2021. 9

[53] L.-Q. Yan, M. Hašan, W. Jakob, J. Lawrence, S. Marschner, and R. Ramamoorthi. Rendering glints on high-resolution normal-mapped specular surfaces. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2014)*, 33(4), 2014. 9

[54] L.-Q. Yan, W. Sun, H. W. Jensen, and R. Ramamoorthi. A bssrdf model for efficient rendering of fur with global illumination. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia 2017)*, 36(6), 2017. 9

[55] X. Yan, J. Yang, E. Yumer, Y. Guo, and H. Lee. Perspective transformer nets: Learning single-view 3d object reconstruction without 3d supervision. *Advances in neural information processing systems*, 29, 2016. 2, 3

[56] B. Yang, Z. Dong, J. Feng, H.-P. Seidel, and J. Kautz. Variance soft shadow mapping. In *Computer Graphics Forum*, volume 29, pages 2127–2134. Wiley Online Library, 2010. 5

[57] G.-W. Yang, Z.-N. Liu, D.-Y. Li, and H.-Y. Peng. Jnerf: An efficient heterogeneous nerf model zoo based on jittor. *Computational Visual Media*, 9(2):401–404, 2023. 1, 8

[58] K. Yin, J. Gao, M. Shugrina, S. Khamis, and S. Fidler. 3dstylenet: Creating 3d shapes with geometric and texture style variations. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12456–12465, 2021. 3

[59] C. Zhang, L. Wu, C. Zheng, I. Gkioulekas, R. Ramamoorthi, and S. Zhao. A differential theory of radiative transfer. *ACM Transactions on Graphics (TOG)*, 38(6):1–16, 2019. 2

[60] X. Zhang, Q. Li, H. Mo, W. Zhang, and W. Zheng. End-to-end hand mesh recovery from a monocular rgb image. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2354–2364, 2019. 3

[61] W.-Y. Zhou, G.-W. Yang, and S.-M. Hu. Jittor-gan: A fast-training generative adversarial network model zoo based on jittor. *Computational Visual Media*, 7:153–157, 2021. 1

[62] M. Zollhöfer, P. Stotko, A. Görlitz, C. Theobalt, M. Nießner, R. Klein, and A. Kolb. State of the art on 3d reconstruction with rgb-d cameras. In *Computer graphics forum*, volume 37, pages 625–652. Wiley Online Library, 2018. 1