

G-Buffer Supported Neural Screen-space Refraction Baking for Real-Time Global Illumination

Ziyang Zhang
Waseda University
Tokyo, Japan

ziyangz5@toki.waseda.jp

Edgar Simo-Serra
Waseda University
Tokyo, Japan

ess@waseda.jp

Abstract

We propose a neural screen-space refraction baking method for global illumination rendering, with applicability to real-time 3D games. Existing neural global illumination rendering methods often struggle with refractive objects due to the lack of texture information in G-Buffers. While some existing approaches extend neural global illumination to refractive objects by predicting texture maps (UV maps), they are limited to objects with simple geometry and UV maps. In contrast, our method bakes refracted textures without these assumptions by directly encoding the world coordinates of refracted objects into the neural network instead of UV coordinates. Our experiments demonstrate that our approach performs better on refraction rendering than previous methods. Additionally, we investigate the differences in neural network performance when baking coordinates in different spaces, such as world space, screen space, and UV space, showing the best results yielded by baking in world-space coordinates.

1. Introduction

Global illumination rendering is a fundamental problem in computer graphics, playing a fundamental role in generating realistic images from 3D scenes. While offline global illumination can be well achieved by expensive path tracing algorithms [14], real-time global illumination still remains a challenging problem, critical for real-time 3D games. Traditionally, precomputing, or “baking”, the global illumination is a common practice to achieve real-time global illumination rendering. Recently, using neural networks to bake the global illumination has extended traditional precomputation methods’ capability to handle dynamic scenes and complex light transportation.

A popular approach for modern 3D rendering in games is deferred rendering, where information about objects in the screen is rasterized to G-buffers, containing information such

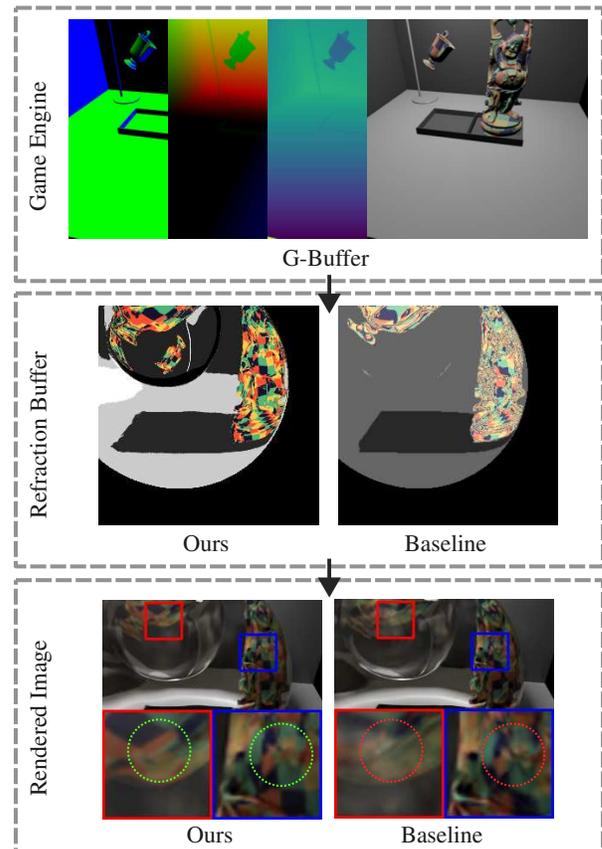


Figure 1. **Overview of our method.** To accurately bake the refracted texture, supporting the neural global illumination rendering, we precompute the refracted world coordinates, and then transform them into screen-space to perform color mapping. Compared to the baseline, CrystalNet [25], our method can generate high-quality refracted textures, and thus improve the rendering quality.

as diffuse colours, normals, etc. Afterwards, the shading is done directly in screen space using the information provided by G-buffers. Neural Global Illumination [6, 9, 11, 25] replaces the shading algorithm with a neural network, making

use of the G-buffers to provide the neural network with all the necessary information to predict the global illumination without the need to embed all geometric details and complex color information into the network itself. Using G-buffers also allows the neural network to avoid relying on complex spatial data such as point clouds, thereby greatly reducing computational complexity. Furthermore, G-buffers can be easily accessed from rasterization-based renderers, which makes neural global illumination accessible to modern game engines.

However, using G-buffers as input has a limitation: they do not capture information about objects visible through refraction in refractive objects. When rendering refractive objects, because of lacking information provided by G-buffers, the neural networks have to encode the complex texture color, and thus leading to inaccurate rendering. Zhang and Simo-Serra [25] proposed a method to predict the refracted texture as texture coordinates to reduce the frequency of the learning targets, which is more friendly to neural networks. However, the method assumes that the texture mapping is simple and smooth. In practice, texture mapping is often complex and discontinuous, which makes the UV mapping prediction method less effective. Furthermore, the texture coordinates are sensitive to small errors, which makes the prediction of texture coordinates more challenging.

In this paper, we propose neural screen-space refraction baking, a method that predicts refracted world coordinates using a neural network, and then transforms these coordinates into screen-space to perform color mapping. Our method does not require models with high quality texture mappings and is therefore more robust to models with complex or irregular texture mappings than previous UV-coordinate-based methods [25]. Because our method only uses G-buffers and camera transformation matrices as inputs, it can be easily deployed in modern game engines. The overview of our method can be found in Fig. 1 and Fig. 2.

In summary, our contributions are as follows:

- A neural screen-space refraction baking method that generates high-quality refracted textures without requiring high-quality texture mapping.
- Experimental results showing that the quality of the generated refraction textures is highest when predicted in world coordinates, rather than in texture space or directly in screen space.

2. Related Work

2.1. Traditional Refraction Rendering

Early attempts addressed refraction rendering with arbitrary refractive shapes and were able to handle both distant lighting and nearby objects [22, 23]. However, these methods could handle only two refractive surfaces due to limitations in their approximations. Oliveira and Brauwers extended

refraction rendering to deformable objects [16], but their method was limited to distant lighting. A precomputation-based method using ray tracing was also proposed [10], but it is restricted to static objects due to the precomputation step.

Furthermore, in modern rasterization-based game engines, refraction rendering is often achieved using screen-space refraction [17, 21]. However, the methods mentioned above are generally limited to few refractions, simple refractive meshes, or static scenes. In contrast, our method can handle an arbitrary number of refractive objects in dynamic scenes.

Rough surface refraction is also an important topic in refraction rendering. Existing methods have explored this using voxel-based techniques [7], approximations of the BTDF with spherical Gaussians [4], among others. In this paper, we only study the refraction of smooth surfaces as the previous work [25] has shown that neural networks are often highly capable of capturing low-frequency global illumination effects, such as roughness.

2.2. Neural Global Illumination

Neural global illumination provides a powerful way to generate real-time global illumination effects without path tracing. In the earliest attempt [18], Ren *et al.* used a neural network to bake the global illumination with static geometries. Later works extended this to dynamic scenes by training encoder-decoder networks to represent scenes [8, 11]. Later approaches such as Active Exploration [6] explore the way to importance sample the scene to improve the training procedure.

In parallel, neural radiosity methods precompute the global illumination by predicting a residual to the rendering equation [12]. Such methods often use lightweight neural networks and thus have low computational costs. Later, several methods extend this to dynamic scenes [3, 20].

However, these methods are not designed to handle refraction. Many modern neural global illumination methods [6, 9, 11] use G-buffers as input to neural networks, providing the models with essential information, particularly for complex texture colors. However, G-buffers do not capture information about objects visible through refractive surfaces. Since neural networks are known to struggle with encoding high-frequency information, lacking such data in the G-buffer means the network no longer has access to accurate texture information, resulting in inaccurate rendering.

2.3. Neural Refraction Rendering

To solve the refraction issue for neural global illumination, a method proposed an architecture, GlassNet [26], to support order-independent-transparency and high quality transparency rendering in neural rendering. This method, however, only assumes the index of refraction is one. Later, CrystalNet [25] extended GlassNet to support full refractive

objects by predicting the UV coordinates of the refracted texture. However, it requires refracted objects to have simple and smooth texture mappings. Our method, in contrast, predicts the world coordinates of refracted objects and then converts them to screen-space coordinates, which does not require high-quality texture mapping.

3. Proposed Approach

3.1. Overview of Neural Global Illumination

In neural global illumination, the overall objective is to predict the radiance that arrives at the camera from scene surfaces using a neural network, trained on ground truths generated by path tracing. At position \mathbf{p} given a ray with the incident and outgoing directions, ω_i and ω_o , the outgoing radiance, L_o , is given by the rendering equation [14]:

$$L_o(\mathbf{p}, \omega_o) = \int_{\Omega} L_i(\mathbf{p}, \omega_i) f(\omega_o, \omega_i) |\mathbf{n} \cdot \omega_i| d\omega_i + L_e(\mathbf{p}, \omega_o) \quad (1)$$

where L_i is the incoming radiance, f is the BRDF, \mathbf{n} is the surface normal. The integral is over the hemisphere Ω of possible incident directions. L_e is the emitted radiance.

To avoid doing expensive path tracing, we can train a neural network renderer, \mathcal{R} , to precompute the radiance inside the neural network, and predict L_o at a given position. \mathcal{R} accepts all parameters that are necessary for a regular renderer to compute L_o , including the world position, normal and texture. Those parameters are often provided as G-buffers rendered in a rasterization renderer. Providing those information helps the neural network to only focus on the illumination precomputation instead of the geometry and texture reconstruction. Usually, to further help the learning process, screen-space textures and direct lighting rendered in the rasterization renderer are also passed into the neural renderer. The process of rendering the scene with \mathcal{R} can be summarized as follows:

$$L_o(\mathbf{p}, \omega_o) \approx \mathcal{R}(\mathbf{p}, \omega_o, \mathbf{n}, m, c, L_d) \quad (2)$$

where m is the material information including roughness and texture color, c is the screenspace texture color, and L_d is the direct lighting.

3.2. Neural Refraction Baking

Benefiting from G-buffers and direct lighting, neural networks do not need to store all the geometry and texture information within the network itself, which is one of the crucial reasons why they can bake global illumination well with a relatively small number of parameters. Low-frequency global illumination, such as ambient occlusion, soft shadows, and color bleeding, can be well captured by these neural renderers given enough training data. However, neural

global illumination baking methods often perform poorly in high-frequency areas, especially on refraction effects, as the G-buffer can no longer provide enough information.

To formally define the problem, we can separate the incident radiance into two parts, L_i for the incoming radiance above a surface and L_i for the incoming radiance below the surface. All **reflection-related** terms are colored in **orange**, and all **refraction-related** terms are colored in **blue**. The rendering equation can be expanded as follows:

$$L_o(\mathbf{p}, \omega_o) = \int_{\Omega^+} L_i(\mathbf{p}, \omega_i) f_r(\omega_o, \omega_i) |\mathbf{n} \cdot \omega_i| d\omega_i + \int_{\Omega^-} L_i(\mathbf{p}, \omega_i) f_t(\omega_o, \omega_i) |\mathbf{n} \cdot \omega_i| d\omega_i \quad (3)$$

Baking the reflection scattering is relatively simple as it is possible to use G-buffers and direct rendering, and neural networks only need to encode global illumination rather than high-frequency details such as textures. However, on refractive surfaces, the neural network needs to encode all the high-frequency texture color inside the network as the lack of G-buffers. Under the expanded rendering equation, \mathcal{R} can be extended to:

$$\mathcal{R}(\mathbf{p}, \omega_o, \mathbf{n}, m, c, L_d) = \mathcal{R}(\cdot) + \mathcal{R}(\cdot, \hat{\gamma}) \quad (4)$$

where \cdot represents all the parameters in \mathcal{R} and $\hat{\gamma}$ represents all the information about objects refracted onto the refractive surfaces. Regular neural global illumination methods implicitly encode $\hat{\gamma}$ inside the network, which leads to blurry and inaccurate refraction effects, especially on high-frequency textures. To help the neural network to precompute L_i , it is possible to use another neural network, R-buffer (refraction buffer) generator, \mathcal{B} , to precompute the refracted texture information in a neural network friendly form. Previous works [25] uses \mathcal{B} to bake $\hat{\gamma}$ the refracted texture as texture coordinate maps, and then use texture mapping to reconstruct the color information as $\hat{\gamma}$. However, texture coordinates are highly sensitive to small errors, and the texture maps of objects can be very complex with severe discontinuity, causing the \mathcal{B} to be unable to reconstruct the refracted texture. Additionally, R-buffer often contains other information such as geometry normal, but we will focus on the texture color in this paper as it is the most challenging part to bake.

3.3. Screen-space R-Buffer Generation

To address the problem of baking refracted textures in texture coordinate form, we propose a precomputation method, screen-space neural refraction baking (SSNRB), that directly bakes refracted objects in world coordinates. First, we bake the refracted objects coordinates in world space, then convert the coordinates to screen-space coordinates. The refracted areas are then mapped using these screen-space coordinates, with framebuffers storing the texture information

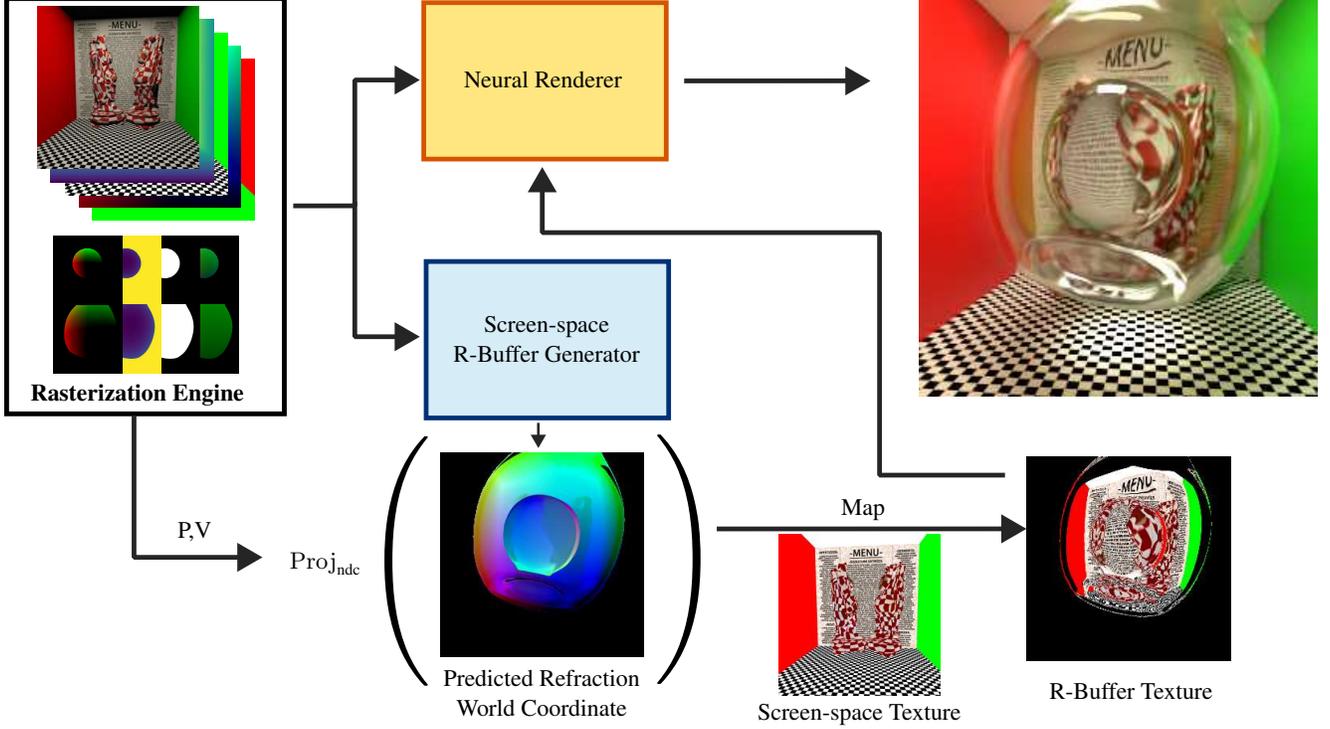


Figure 2. **Overall structure of our method.** We acquire the G-buffers and direct lighting from the rasterization renderer and pass them into the neural renderer and the screen-space R-buffer generator. The screen-space R-buffer generator precomputes the refracted world coordinates, which are then transformed into normalized device coordinates (NDC) and used to map the refracted texture color.

in the screen space. The overall structure of our rendering pipeline is shown in Fig. 2.

Formally, we can summarize our method based on Eq. (4) as follows:

$$\begin{aligned} \mathcal{R}(\mathbf{p}, \omega_o, \mathbf{n}, m, c, L_d, \hat{\gamma}) &= \mathcal{R}(\cdot, T(c, c)) \\ c &= \text{Proj}_{\text{ndc}}(\mathcal{B}(\cdot)_{\text{world}}) \end{aligned} \quad (5)$$

where \cdot represents all parameters of \mathcal{R} except for $\hat{\gamma}$, c is the refracted texture color obtained from the screen-space texture c , and $\mathcal{B}(\cdot)_{\text{world}}$ represents the baked world coordinates. The function T denotes texture mapping. The operator Proj_{ndc} is the transformation from world space to normalized device coordinates (NDC). This operation is trivial to access in a rasterization engine by multiplying the world-to-view transformation (V), the view-to-clip transformation (P), and then performing a perspective division:

$$\text{Proj}_{\text{ndc}}(\mathbf{p}_{\text{world}}) = \frac{PV\mathbf{p}_{\text{world}}}{(PV\mathbf{p}_{\text{world}})_w}$$

$$P = \begin{bmatrix} \frac{1}{\tan(\frac{fov}{2}) \cdot a} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan(\frac{fov}{2})} & 0 & 0 \\ 0 & 0 & \frac{f}{f-n} & -\frac{fn}{f-n} \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad V = \begin{bmatrix} R & \mathbf{t} \\ 0 & 1 \end{bmatrix} \quad (6)$$

where a is the aspect ratio, f and n are the far and near plane, fov is the field of view, and R and \mathbf{t} are the rotation and translation of the camera. As NDC coordinates range from $[-1, 1]$, the texture mapping function T can use bilinear interpolation based on the screen-space texture and the transformed coordinates to reconstruct the refracted texture color.

Our method uses \mathcal{B} to precompute the world coordinates instead of texture coordinates. Our method is more robust to texture map discontinuity and can handle objects with complex texture mapping as we do not precompute the texture coordinates. For example, the mesh shown in Fig. 3 has a complex texture mapping and discontinuity. However, in world space, the mesh is continuous and smooth, which is more friendly to neural networks.

Similar to CrystalNet [25], we use the total variation loss [19] to smooth the baked world coordinates. The total variation loss is defined as follows:

$$\mathcal{L}_{tv}(c) = \sum_{i,j} |c_{i+1,j} - c_{i,j}| + |c_{i,j+1} - c_{i,j}| \quad (7)$$

By doing so, we encourage the neural network to predict smooth world coordinates in order to further decrease the sensitivity to small errors in the final screen-space texture mapping.

3.4. Training

We choose the GlassNet architecture[26] as the structure of \mathcal{R} and \mathcal{B} because it can handle multiple refractive objects without the need for sorting. Following this architecture, the \mathcal{R} and \mathcal{B} will accept the G-buffers of opaque objects and transparent objects at the same time as shown in Fig. 2. The \mathcal{R} and \mathcal{B} are trained separately. By doing so, we can train \mathcal{B} with a larger dataset as generating R-buffers ground truth is much faster than generating the path tracing ground truth, as R-buffers ground truth only requires ray tracing the refracted rays.

We use Mitsuba 3 [13] to generate the path tracing ground truths using a sample per pixel (SPP) of 4096. The path tracing ground truth images are shown in Fig. 3. The training dataset consists of 1,000 to 1,500 images per scene for the neural renderer and 3,000 images per scene for the R-buffer generator. All training images have a resolution of 256×256 . We uniformly sample the camera position and locations of variable objects within pre-determined ranges in order to support variable scenes.

One common limitation of screen-space methods is that artifacts may occur when zooming into low-resolution screen-space textures as they do not contain enough information. We address this problem by rasterizing 1024×1024 screen-space textures and performing bilinear interpolation on them to generate the R-buffers.

4. Experiments

4.1. Overall Setup

We compare our method with GlassNet [26] and CrystalNet [25] as they are the closest works to our methods. Our experiments are under three different scenes: HEMISPHERE, BUDDHA, and BUNNY. All scenes are modified from existing resources [2, 27] and assets in the public domain. HEMISPHERE contains two light sources, two overlapping refractive objects—one of which is movable—and one rotating model (the ewer). BUDDHA and BUNNY are two scenes within the Cornell Box setup, each containing different objects with complex texture mapping, movable refractive objects, and a variable light source.

For evaluation metrics, we use L1 error, SSIM [15], LPIPS [24], and DISTS [5] for the rendering result evaluation. We use L1 error for the R-buffer evaluation (R.L1)

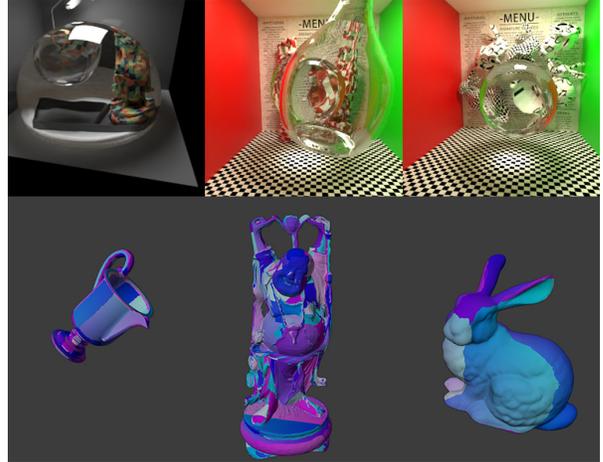


Figure 3. **Random selection from the dataset.** We also demonstrate the models with complex UV mapping we used in the dataset.

by comparing the R-buffer texture reconstructed using ray-traced ground truth texture/world coordinates with the R-buffer texture reconstructed from the neural network prediction. Additionally, we also demonstrate the L1 error and masked SSIM of the refraction area on the rendering results indicated as T.L1 and T.SSIM. To better evaluate the high-frequency texture reconstruction, we include the L2 error of the amplitude spectrum on frequency domain (T.Amp.) [25] on the refractive areas. The qualitative results are shown Fig. 4 and the quantitative results are shown in Tab. 1.

4.2. Rendering Results

As shown in Fig. 4, the renderer supported by our screen-space R-buffers performs much better than the baselines. We show the UV map of the mainly used models in Fig. 3. The UV maps of the models are complex, so that directly baking them into the R-buffer generator would not be feasible.

HEMISPHERE contains a large, movable refractive hemisphere that encloses a small glass egg. As the texture mapping is very complex, CrystalNet failed to reconstruct the refracted texture, especially on the newer model, which is behind multiple refractive objects. Our method, on the other hand, was able to reconstruct the refracted texture accurately and achieved better rendering quality.

In the scene, BUDDHA, there are two overlapping refractive objects and two models of Buddha behind them. Because of the complex texture mapping, CrystalNet was not able to reconstruct the refracted texture accurately, leading to distortion in the rendered images. Our method, on the other hand, was not sensitive to the complexity of the texture mapping and was able to reconstruct the refracted texture accurately. Furthermore, in the scene BUNNY, as the checkerboard is much smaller than in the BUDDHA scene, the CrystalNet completely failed to reconstruct the

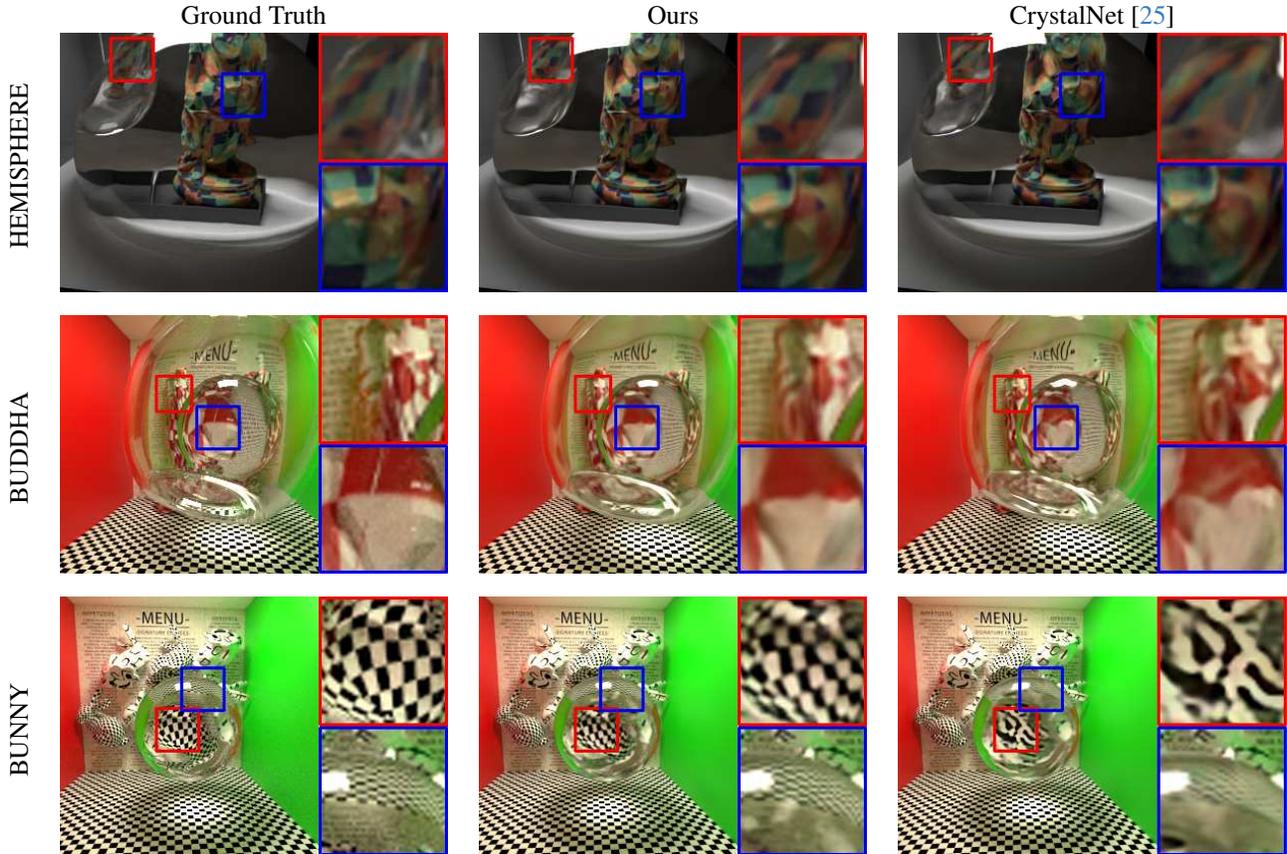


Figure 4. **Qualitative comparison.** Our method is able to render the refraction effects more accurately on meshes with complex texture mappings.

Scene	Method	Metrics						
		L1 ↓	LPIPS ↓	SSIM ↑	DISTS ↓	T.L1 ↓	T.SSIM ↑	T.Amp. ↓
HEMISPHERE	GlassNet	0.02975	0.14080	0.84491	0.19662	0.14132	0.72818	0.06292
	CrystalNet	0.01567	0.03690	0.93126	0.09779	0.08054	0.87355	0.05091
	Ours	0.01524	0.03665	0.93237	0.09000	0.07897	0.87569	0.05027
BUDDHA	GlassNet	0.04617	0.14220	0.79107	0.13377	0.20934	0.55210	0.07672
	CrystalNet	0.03147	0.07762	0.85234	0.10585	0.15965	0.67129	0.06297
	Ours	0.03008	0.06496	0.86268	0.09871	0.14998	0.70082	0.06191
BUNNY	GlassNet	0.04272	0.08258	0.89602	0.10249	0.35754	0.41793	0.02402
	CrystalNet	0.02885	0.03526	0.92186	0.07637	0.28249	0.52467	0.02025
	Ours	0.02775	0.03396	0.93149	0.06951	0.24549	0.59784	0.02004

Table 1. **Quantitative comparison.** Our method achieves better performance in all scenes, especially in transparent areas. Best result is denoted in **bold**.

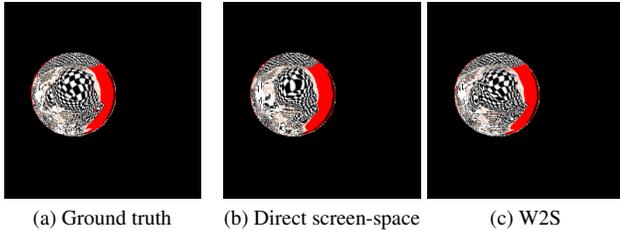


Figure 5. **R-buffer texture comparison between direct screen-space and W2S (world-to-screen-space).** The R-buffer texture generated by screen-space coordinates directly from neural networks has more distortion, while W2S method is more accurate.

refracted texture. Those scenes demonstrate the advantage of not using the UV map in the R-buffer generation process.

4.3. Ablation Study

We evaluate the quality of the generated R-buffer by comparing two approaches: directly predicting screen-space coordinates and predicting world coordinates, which are then converted to screen-space coordinates (W2S) using world-to-NDC transformation. As shown in Fig. 5, predicting the screen-space coordinates directly causes more distortion to the refracted textures and thus has higher error than the texture reconstructed by world coordinates. Quantitatively, the L1 reconstruction loss on BUNNY for the W2S method is 0.01911, while that of the direct screen-space coordinate method is 0.02392, demonstrating the advantage of the world-coordinate-conversion method. In conclusion, we consider predicting world coordinates and converting them to screen-space coordinates to be a better approach.

As we use the same neural network architecture as CrystalNet, the computational cost of our method is comparable to that of CrystalNet. The only additional step is the world-to-NDC transformation. As discussed in Sec. 3.3, terms related to this transformation already exist in the rasterization engine, and the transformation itself is simple to perform. Therefore, this additional step introduces minimal overhead in the rendering pipeline. In our experiments, our method even slightly outperforms CrystalNet in terms of computational cost. On RTX 4090, at a native resolution of 256×256 with a supersampler [1] performing $4 \times$ upsampling to 1024×1024 , our method can render at around 34.0 FPS, whereas CrystalNet is at 33.1 FPS. This is expected, as our method performs bilinear texture sampling on the screen-space texture only once, while CrystalNet requires multiple times on every refracted texture. Also, during the inference, our method does not require predicting the object indices, which is a computationally expensive operation in CrystalNet.

5. Limitation and Future Work

Our method is based on screen-space coordinates and thus is limited to the visible area of the screen when generating refraction buffers. However, in most cases, off-screen refraction happens on the edge of the refraction objects and thus does not have a clear view; therefore, the neural renderer itself can roughly predict the color information without causing too many artifacts. It is also possible to extend our method to handle refracted location off-screen by simultaneously predicting the world coordinates and texture coordinates. This would allow us to maintain a similar performance with CrystalNet on the off-screen refraction area and improve the performance on the on-screen refraction area.

References

- [1] Namhyuk Ahn, Byungkon Kang, and Kyung-Ah Sohn. Fast, accurate, and lightweight super-resolution with cascading residual network. In *Proceedings of the European conference on computer vision (ECCV)*, pages 252–268, 2018. 7
- [2] Benedikt Bitterli. Rendering resources, 2016. <https://benedikt-bitterli.me/resources/>. 5
- [3] Arno Coomans, Edoardo A Dominci, Christian Döring, Joerg H Mueller, Jozef Hladky, and Markus Steinberger. Real-time neural rendering of dynamic light fields. In *Computer Graphics Forum*, page e15014. Wiley Online Library, 2024. 2
- [4] Charles De Rousiers, Adrien Bousseau, Kartic Subr, Nicolas Holzschuch, and Ravi Ramamoorthi. Real-time rendering of rough refraction. *IEEE Transactions on Visualization and Computer Graphics*, 18(10):1591–1602, 2011. 2
- [5] Keyan Ding, Kede Ma, Shiqi Wang, and Eero P. Simoncelli. Image quality assessment: Unifying structure and texture similarity. *CoRR*, abs/2004.07728, 2020. 5
- [6] Stavros Diolatzis, Julien Philip, and George Drettakis. Active exploration for neural global illumination of variable scenes. *ACM Transactions on Graphics (TOG)*, 41(5):1–18, 2022. 1, 2
- [7] Elmar Eisemann and Xavier Décorêt. Fast scene voxelization and applications. In *Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 71–78, 2006. 2
- [8] SM Ali Eslami, Danilo Jimenez Rezende, Frederic Besse, Fabio Viola, Ari S Morcos, Marta Garnelo, Avraham Ruderman, Andrei A Rusu, Ivo Danihelka, Karol Gregor, et al. Neural scene representation and rendering. *Science*, 360(6394):1204–1210, 2018. 2
- [9] Duan Gao, Haoyuan Mu, and Kun Xu. Neural global illumination: Interactive indirect illumination prediction under dynamic area lights. *IEEE Transactions on Visualization and Computer Graphics*, 2022. 1, 2
- [10] Olivier Gènevaux, Frédéric Larue, and Jean-Michel Dischler. Interactive refraction on complex static geometry using spherical harmonics. In *Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 145–152, 2006. 2
- [11] Jonathan Granskog, Fabrice Rousselle, Marios Papas, and Jan Novák. Compositional neural scene representations for

- shading inference. *ACM Transactions on Graphics (TOG)*, 39(4):135–1, 2020. [1](#), [2](#)
- [12] Saeed Hadadan, Shuhong Chen, and Matthias Zwicker. Neural radiosity. *ACM Transactions on Graphics (TOG)*, 40(6):1–11, 2021. [2](#)
- [13] Wenzel Jakob, Sébastien Speierer, Nicolas Roussel, and Delio Vicini. Dr.jit: A just-in-time compiler for differentiable rendering. *Transactions on Graphics (Proceedings of SIGGRAPH)*, 41(4), 2022. [5](#)
- [14] James T Kajiya. The rendering equation. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 143–150, 1986. [1](#), [3](#)
- [15] Artur Loza, Lyudmila Mihaylova, Nishan Canagarajah, and David Bull. Structural similarity-based object tracking in video sequences. In *2006 9th International Conference on Information Fusion*, pages 1–6, 2006. [5](#)
- [16] Manuel M Oliveira and Maicon Brauwere. Real-time refraction through deformable objects. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, pages 89–96, 2007. [2](#)
- [17] Matt Pharr and Randima Fernando. *GPU Gems 2: Programming techniques for high-performance graphics and general-purpose computation (gpu gems)*. Addison-Wesley Professional, 2005. [2](#)
- [18] Peiran Ren, Jiaping Wang, Minmin Gong, Stephen Lin, Xin Tong, and Baining Guo. Global illumination with radiance regression functions. *ACM Trans. Graph.*, 32(4):130–1, 2013. [2](#)
- [19] Leonid I Rudin, Stanley Osher, and Emad Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: nonlinear phenomena*, 60(1-4):259–268, 1992. [4](#)
- [20] Rui Su, Honghao Dong, Jierui Ren, Haojie Jin, Yisong Chen, Guoping Wang, and Sheng Li. Dynamic neural radiosity with multi-grid decomposition. In *SIGGRAPH Asia 2024 Conference Papers*, pages 1–12, 2024. [2](#)
- [21] Unity Technologies. *Refraction in the High Definition Render Pipeline*, 2021. Accessed: 2025-03-24. [2](#)
- [22] Chris Wyman. An approximate image-space approach for interactive refraction. *ACM transactions on graphics (TOG)*, 24(3):1050–1053, 2005. [2](#)
- [23] Chris Wyman. Interactive image-space refraction of nearby geometry. In *Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, pages 205–211, 2005. [2](#)
- [24] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 586–595, 2018. [5](#)
- [25] Ziyang Zhang and Edgar Simo-Serra. Crystalnet: Texture-aware neural refraction baking for global illumination. In *Computer Graphics Forum*, page e15227. Wiley Online Library, 2024. [1](#), [2](#), [3](#), [4](#), [5](#), [6](#)
- [26] Ziyang Zhang and Edgar Simo-Serra. Neural scene baking for permutation invariant transparency rendering with real-time global illumination. *arXiv preprint arXiv:2405.19056*, 2024. [2](#), [5](#)
- [27] Kun Zhou, Xi Wang, Yiyang Tong, Mathieu Desbrun, Baining Guo, and Heung-Yeung Shum. Texturemontage: Seamless texturing of arbitrary surfaces from multiple images. *ACM Transactions on Graphics*, 24(3):1148–1155, 2005. [5](#)