Supplementary Materials: Neural Rays for Occlusion-aware Image-based Rendering

Yuan Liu[1] Sida Peng[2] Lingjie Liu[3] Qianqian Wang[4] Peng Wang[1]
Christian Theobalt[3] Xiaowei Zhou[2] Wenping Wang[5]
[1]The University of Hong Kong [2]Zhejiang University [3]Max Planck Institute for Informatics
[4]Cornell University [5]Texas A&M University

1. Comparison between NeRF [4] and NeuRay

In this section, we will compare NeRF [4] with NeuRay on parameterizing different probabilities on a ray. Let us first review NeRF's definitions of different probabilities.

NeRF's parameterization. Given a ray $\boldsymbol{p}(z) = \boldsymbol{o} + z\boldsymbol{r}$ with $\{\boldsymbol{p}_i \equiv \boldsymbol{p}(z_i) | i = 1, ..., N; z_i < z_{i+1}; z_i \in \mathbb{R}^+\}$ sample points on the ray, NeRF computes the densities of these points by $d_i = \mathcal{F}(\boldsymbol{p}_i)$, where $\mathcal{F}$ is an MLP. Then, the alpha values on these points are computed by $\alpha_i = 1 - \exp(-\text{ReLU}(d_i)l_i)$, where $l_i = z_{i+1} - z_i$. In this case, the visibility from the ray origin $\boldsymbol{o}$ to the point $\boldsymbol{p}_i$ is

$$v_i = \prod_{j=1}^{i-1}(1 - \alpha_j). \tag{1}$$

The hitting probability, which means the ray is not occluded by any depth up to z_i and hits a surface in the range (z_i, z_{i+1}), is

$$h_i = v_i \alpha_i \tag{2}$$

This process is summarized by Fig. 1 (a), where all probabilities are based on the density d_i.

NeuRay's parameterization. In comparison, all probabilities in NeuRay are based on the occlusion probability $t(z)$. Given an **input ray**, we also sample points $\{\boldsymbol{p}_i \equiv \boldsymbol{p}(z_i) | i = 1, ..., N; z_i < z_{i+1}; z_i \in \mathbb{R}^+\}$. Based on $t(z)$, we compute the hitting probability $\tilde{h}_i$ (Eq.(7) in the main paper) and the visibility v_i

$$\tilde{h}_i = t(z_{i+1}) - t(z_i), \tag{3}$$

$$v_i \equiv v(z_i) = 1 - t(z_i). \tag{4}$$

We use $\tilde{h}$ here to indicate it was defined on an *input ray*. Then, noticing the relationship between the visibility, the hitting probability and the alpha value in Eq.(2), we can compute the alpha value $\tilde{\alpha}_i$ on $\boldsymbol{p}_i$ from $\tilde{h}_i$ and v_i by $\tilde{\alpha}_i = \tilde{h}_i / v_i = (t(z_{i+1}) - t(z_i))/(1 - t(z_i))$, which is the Eq.(9) in the main paper. $\tilde{\alpha}$ also means it is defined on an *input ray*. This process is summarized by Fig. 1 (b), which is inverse to the density definition in NeRF.

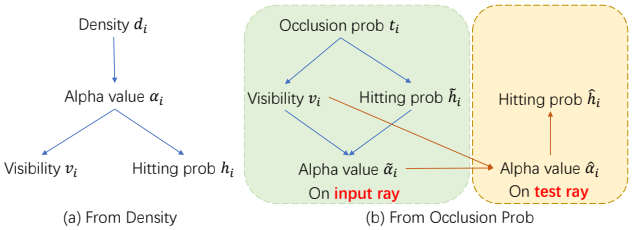


Figure 1. Comparison between the density in NeRF and the occlusion probability in NeuRay. (a) In NeRF, the network decodes the density to compute the alpha values, the hitting probability and the visibility. (b) In NeuRay, the network decodes the parameters of occlusion probability, which enables efficient computation of the visibility and the hitting probability. Then, we derive the corresponding alpha values from the hitting probability and visibility.

Motivation of designing the hitting probability $\hat{h}$ as the form in Sec. 3.6. Our motivation is explained as follows. When we need to compute the hitting probability on the sample points $\boldsymbol{p}_i$ of a **test ray**, we need to derive the alpha values $\hat{\alpha}_i$ on these points and compute the hitting probability by $\hat{h}_i = \prod_{j=1}^{i-1}(1 - \hat{\alpha}_j)\hat{\alpha}_i$. Note we use $\hat{\alpha}$ to indicate that it is defined on the *test ray* and to distinguish it from the alpha value $\tilde{\alpha}$ defined on *input rays* and the alpha value α computed from *the constructed radiance field*. To compute $\hat{\alpha}_i$, we choose to use the weighted sum of alpha values $\tilde{\alpha}$ from all input rays with their visibility as weights

$$\hat{\alpha}_i = \frac{\sum_j \tilde{\alpha}_{i,j} v_{i,j}}{\sum_j v_{i,j}}. \tag{5}$$

This is the Eq.(10) in the main paper. By considering the visibility, only visible input rays can affect the alpha values $\hat{\alpha}$ of the point. Finally, the alpha value $\hat{\alpha}_i$ is used in the computation of $\hat{h}_i = \prod_{j=1}^{i-1}(1 - \hat{\alpha}_j)\hat{\alpha}_i$.

2. Discussion about visibility from density

In the main paper, we have discussed that parameterizing the visibility with volume density is computationally impractical, because it requires K_r times forward pass of the network $\mathcal{F}$ to compute visibility from one input view to one 3D point. For example, given only 1 test ray and 8 neighboring input views, we sample 128 points on this test ray. Then, to compute the visibility from every input view to every sample point, we need to consider 8×128 input rays. If we further sample 32 points on every input ray to accumulate volume density for visibility computation. It will require $8\times128\times32$ times evaluations of the network $\mathcal{F}$. Such computation complexity is not affordable for training on a single 2080 Ti GPU to achieve reasonable results.

3. Memorization interpretation in finetuning

$\ell_{consist}$ enables the NeuRay representation to memorize the predicted surfaces from the constructed radiance field in finetuning. An example is shown in Fig. 2.

1. In one training step as shown in Fig. 2 (a), NeuRay chooses view A as the pseudo test view and aggregates features from neighboring input views B, C and D to predict h_i of the test ray on the test view A.
2. Applying the consistency loss $\ell_{consist}$ forces the visibility feature $\boldsymbol{g}$ of the test ray of A to produce $\tilde{h}_i$ consistent with h_i. This can be interpreted that the h_i is memorized in $\boldsymbol{g}$, which also means the memorization of visibility, because both are computed from $t(z)$.
3. In subsequent another training step which uses view A, B and C to render the pseudo test view D in Fig. 2 (b), NeuRay already knows that the green point is invisible to A by checking the memorized visibility on A. Hence, NeuRay is able to predict an accurate h_i for view D.
4. The predicted h_i on the test ray of view D is also memorized on D and helps the prediction of hitting probabilities of other views in the subsequent training steps.

By memorizing the scene geometry in h_i, NeuRay is able to refine its scene representation to provide better occlusion inference during finetuning. In contrast, existing methods [1, 6, 7, 11] can only adjust their network parameters to improve their feature aggregation during finetuning, which are still oblivious to occlusions in the scene.

4. Training details and architecture

Architecture. In the cost volume construction, $N_s = 3$ neighboring input views are used as source views and the height and width of the cost volume are $1/4$ of the height and width of the original image. The cost volume is constructed by a MVSNet [9] pretrained on the DTU training set. When training our renderer, we fixed the weight of this MVSNet due to GPU memory limitation but it is possible

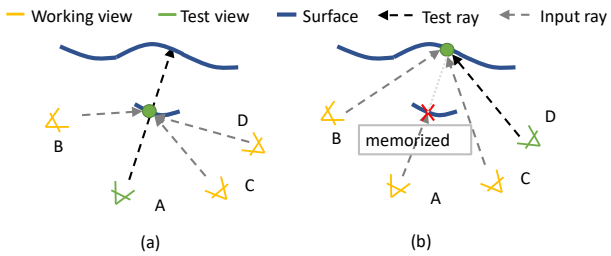


Figure 2. (a) During finetuning, when rendering the pseudo test view A using view B, C, and D, the predicted hitting probabilities and the corresponding visibility will be memorized on view A. (b) Afterwards, the memorized visibility on view A will be used for occlusion inference in rendering the pseudo test view D.

to train this part for better performance. The architecture is illustrated in Fig. 9 and Fig. 10. The image encoder is a ResNet [2] with 13 residual blocks which outputs a feature map with 32 channels. All convolution layers use ReLU as activation function and all batch normalization layers are replaced by instance normalization layers. Compared to IBRNet [7], our image encoder is more lightweight with less intermediate channel number and less residual blocks. The distribution decoder $\mathcal{F}$ contains 5 sub-networks, each of which consists of 3 fully-connected layers. These sub-networks are in charge of the prediction of μ_1, μ_2, σ_1, σ_2 and mixing weight w_0 respectively ($w_1 = 1 - w_0$). All fully-connected layers in $\mathcal{F}$ use ReLU as the activation function except for the final layer which use SoftPlus or Sigmoid as the activation function.

The architecture of feature aggregation networks follows design of IBRNet [7], which is illustrated in Fig. 10. Based on the aggregated feature, the alpha value α_i is computed by

$$\alpha_i = \mathcal{A}(\boldsymbol{f}_i), \tag{6}$$

where $\mathcal{A}$ is an alpha network. The color $\boldsymbol{c}_i$ is computed by

$$\boldsymbol{c}_i = \mathcal{B}(\boldsymbol{f}_i, \{\boldsymbol{f}_{i,j}; \boldsymbol{r}_{i,j} - \boldsymbol{r}; \boldsymbol{c}_{i,j}; v_{i,j}\}), \tag{7}$$

where $\mathcal{B}$ is a color blending network, $\boldsymbol{r}_{i,j}$ is the view direction from j-th input view to the point $\boldsymbol{p}_i$, and $\boldsymbol{c}_{i,j}$ is the color of this point projected on the input view. The alpha network $\mathcal{A}$ will learn to assign a large alpha value α_i to $\boldsymbol{p}_i$ when local image features $\{\boldsymbol{f}_{i,j}\}$ of all visible input views are consistent. Meanwhile, the color blending network $\mathcal{B}$ will learn to use the color $\boldsymbol{c}_{i,j}$ from visible views to produce the color $\boldsymbol{c}_i$ on $\boldsymbol{p}_i$.

Depth loss. In pretraining the generalization model on training scenes, we also force the mean μ_1 of the first logistics distribution to be similar to the input depth by a depth loss ℓ_{depth}

$$\ell_{depth} = \sum \|\mu_1 - z_{in}\|^2, \tag{8}$$

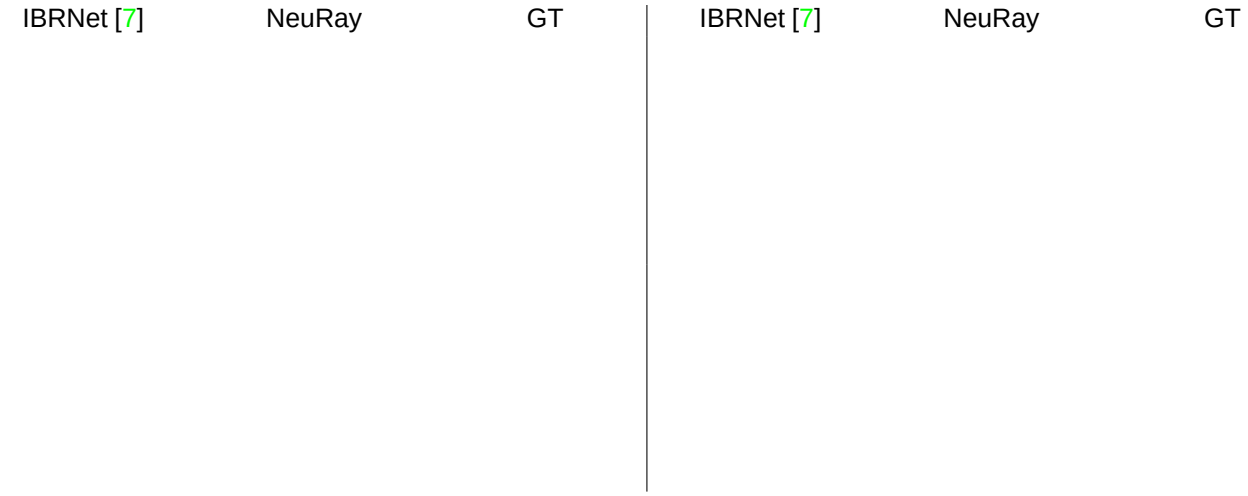


Figure 3. Qualitative results using only 2 neighboring working views to render the test view.

where the depth z_{in} is computed from the cost volume or the input estimated depth maps.

Training details. We use Adam [3] as the optimizer with the default setting ($_1 = 0.999$, $_2 = 0.9$). On every training step, 512 rays are randomly sampled. The initial learning rate for the generalization model is 2e-4 which decays to its half on every 100k steps. We train the generalization model with 400k steps on the training set, which takes about 3 days on a single 2080Ti GPU. In the netuning setting, the learning rate is initially set to 1e-4 and it also decays to its half after 100k training steps. Finetuning 200k steps on a NeRF synthetic dataset with resolution 800 800 cost 20 hours but netuning 200k steps on low resolution images with 400 400 can be faster, which costs 8 hours.

5. More comparison with baselines

Sparse working views on the LLFF dataset As discussed in the main paper, the reason that our method and IBRNet [7] perform very similar on the LLFF dataset is that the LLFF dataset contains very dense forward-facing views, which alleviates the feature inconsistency caused by occlusions. To demonstrate that, we decrease the number of working view N_w to render the novel images. We show the qualitative results with 2 working views in Fig. 3 and the PSNR on the LLFF dataset with different number of workings views in Table 3. The results show that out method consistently outperforms IBRNet and the performance gap enlarges with the decrease of working view number. The reason is that with the decrease of working views, every 3D point is visible to input views so that the feature inconsistency caused by occlusions becomes more severe. Hence, because feature aggregation of IBRNet is not very robust to feature inconsistency, its performance degenerates. In contrast, with the constructed NeuRay, our method is occlusion-aware and thus performs better with sparse views. Meanwhile, we also nd that undistortion by COLMAP [5] is not perfect and noticeable distortions still remain in the LLFF dataset, which makes the scene-speci c optimization of all methods struggles to improve.

Figure 4. Qualitative comparison with NeRF [4].

More qualitative comparison with NeRF. In Fig. 4, we show more comparison with NeRF [4], in which our method can recover details more clearly than NeRF with the same training steps on the given scene.

Dataset	Metrics	Cost volume	Depth maps
NeRF Syn.	PSNR	28.29	28.92
	SSIM	0.927	0.920
	LPIPS	0.080	0.096
DTU	PSNR	26.47	28.30
	SSIM	0.875	0.907
	LPIPS	0.158	0.130
LLFF	PSNR	25.35	25.85
	SSIM	0.818	0.832
	LPIPS	0.198	0.190

Table 1. Results in the generalization setting using NeuRay initialized by estimated depth maps [5] or constructed cost volumes [9]

	$N_w = 8$	$N_w = 12$	$N_w = 16$
PSNR	32.97	33.20	33.60

Table 2. PSNR on the Lego from the NeRF synthetic dataset with different working view numbers N_w.

Method	$N_w = 8$	$N_w = 4$	$N_w = 2$
IBRNet [7]	26.87	25.41	21.72
NeuRay	27.06	26.43	24.86

Table 3. PSNR on the LLFF dataset with $N_w = 2; 4; 8$ working views. The performance of IBRNet [7] drops with the decrease of working views while NeuRay still performs well with only 2 working views. All models are already netuned on the scene for 200k steps.

Setting	Method	N = 100	N = 75	N = 50
Gen	NeuRay	28.41	28.06	26.31
	IBRNet	25.64	25.16	23.68
Ft	NeuRay	32.97	32.71	31.40
	IBRNet	30.37	28.57	25.68

Table 4. PSNR on the Lego with different input view numbers. With the decrease of view number, the performance decreases but NeuRay consistently outperforms IBRNet with different view numbers. "Gen" means the generalization setting while "Ft" means netuning on the scene.

6. Initialize from estimated depth maps

Besides initialization from the cost volumes, NeuRay could also be initialized from estimated depth maps by patch match stereo in COLMAP [5]. In Table 1, we show that initialization from estimated depth maps also produces good rendering results. Especially on the DTU dataset, COLMAP produces more accurate reconstruction than cost volumes so that the rendering quality is better.

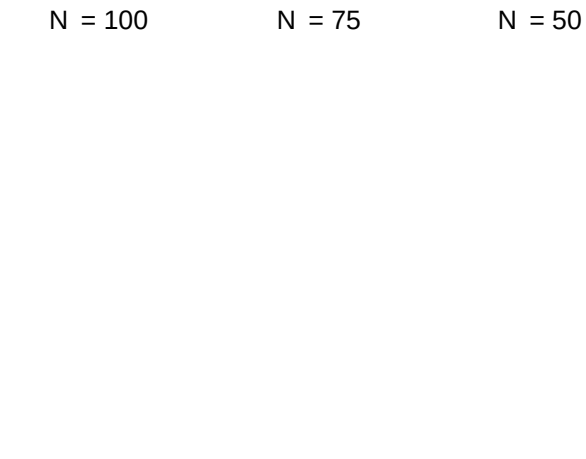


Figure 5. Qualitative results in the generalization setting with different input view numbers. Row 1 shows the results of IBRNet [7] and Row 2 shows the results of NeuRay.

7. View consistency with more working views

Can more working views improve the rendering quality? We observe that simply using more working views on our trained model does not lead to obvious improvements, because the model is netuned with 8 working views. However, netuning our model with more working views indeed improves the rendering quality. We netune our model with different numbers of working views 8, 12, 16 on the Lego from the NeRF synthetic dataset. The results are shown in Table 2. We can see that training with more working views improves the results. The reason is that adding more working views in training means using more views for the feature aggregation to reconstruct the scene geometry, which enforces the view consistency and improves the accuracy of estimated surfaces.

8. Results with sparse input views

To investigate how the performance degenerates as the decrease of the number of input views, we reduce the input views of the Lego of the NeRF synthetic dataset from 100 to 75 and 50 by the farthest point sampling on camera locations. The PSNR is reported in Table 4 and qualitative results are shown in Fig. 5. The performance degrades reasonably as the input views become sparser but is consistently better than IBRNet.

9. Estimation of depth maps

We can decode a depth map from NeuRay on a input view. Along every input ray, we evenly sample points and use the depth of the sample point with the largest as the depth of this input ray. In Fig. 6, we show the depth maps

Figure 6. Depth maps computed on the input views. Left are produced by initialized NeuRay and Right are produced by netuned NeuRay.

from the initialized and the netuned NeuRay representation. The results show that initialized NeuRay already produces a reasonable depth maps and optimizing NeuRay representation is able to enforce the consistency between views and rectify erroneous depth values. In light of this, netuning NeuRay may also be regarded as a process of depth fusion or a consistency check for MVS algorithm, which takes coarse depth maps as input and outputs re ned consistent depth maps. However, unlike commonly-used consistency check which simply discards erroneous depth, optimizing NeuRay is able to correct these erroneous depth.

10. Additional ablation studies

We have conducted additional ablation studies on some of our network designs in Table 5.

1) Adding $\ell_{consist}$ in the generalization training. $\ell_{consist}$ is designed in scene-speci c netuning to memorize the geometry. Adding the loss in cross-scene generalization training produces similar results as the model without the loss.

2) Optimize G^0 only or network parameters only. Either leads to inferior results because only optimizing G^0 will disable re nement of feature aggregation while only optimizing networks will disable re nement of scene geometry in netuning.

3) Number of mixed distributions N_l. N_l limits the maximum number of semi-transparent surfaces to be represented on an input ray. However, we nd that increasing N_l from 2 to 3 does not bring signi cant improvements because there are few semi-transparent surfaces in most cases.

4) Finetune G instead of G^0. Finetuning G^0 with visibility encoder produces better results than netuning G (+0.5 PSNR). The reason is that the convolution layers in the visibility encoder associate nearby feature vectors of trainable G^0 and these nearby pixels usually have similar visibility.

Generalization		Finetuning	
Description	PSNR	Description	PSNR
Default NeuRay	28.41	Default NeuRay	32.97
Add $\ell_{consist}$	28.46	Optimize vis./net. only	32.25/31.90
$N_l = 3$	28.44	$N_l = 3$	33.00
		Finetune G	32.43

Table 5. Additional ablation studies on the Lego from the NeRF Synthetic dataset, reported in PSNR.

11. Direct rendering from NeuRay

Can we render a novel view image from the NeuRay representation without constructing a radiance eld?

Yes, we can render an image directly from NeuRay, which is called the *direct rendering* of NeuRay. Given a test ray $o + zr$ with sample points $\{p_i\}$, we will compute the output color for this ray by $\hat{C} = \sum_i c_i \hat{h}_i$. In Sec. 1, we already show how to compute $\hat{h}$. The only remaining problem is how to compute the color c_i of each sample point directly from NeuRay. To achieve this, we adopt a traditional Spherical Harmonics Fitting here.

We use a set of spherical harmonic functions as basis functions to t a color function $R : S^2 \to R^3$ on every point. Speci cally, we solve the following linear least squares problem

$$\min_{\gamma_i} \sum_j \tilde{h}_{i,j}(z_{i,j}; z_{i,j} + l_i) \| R(r_{i,j}; \gamma_i) - c_{i,j} \|^2 + \lambda_i \| \gamma_i \|; \tag{9}$$

where $\tilde{h}_{i,j}$ is the hitting probability of point p_i on the input ray emitted from the j-th input view, R the color function, γ_i the coef cients of spherical harmonic basis functions on this point, $r_{i,j}$ and $c_{i,j}$ are the viewing direction and color of the input ray emitted from the j-th view, respectively, λ_i a regularization term, and Γ a prede ned diagonal matrix. This linear least squares problem has a closed-form solution. After nding the solution γ_i to Eq.(9), the color c_i is computed by

$$c_i = R(r; \gamma_i). \tag{10}$$

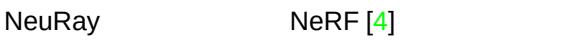
Note that in Eq. (9), the color difference is weighted by hitting probabilities so that occluded input rays will not interfere the output colors. Meanwhile, since we t a function R on the sphere, NeuRay is able to represent anisotropic colors (or radiance) at a point. NeX [8] and PlencOctree [10], also use spherical harmonics functions as basis

Method	Fern	Lego
NeuRay-DR	25.27	29.93
NeuRay-RF	25.93	32.97

Table 6. PSNR of Direct Rendering (DR) of NeuRay and rendering from the constructed radiance field (RF) with NeuRay.

(a) (b)

Figure 7. Comparison between the direct rendering of NeuRay (a) and the rendering of the constructed radiance field (b). Direct rendering of NeuRay produces meaningful results but artifacts occur on the edges. Note the direct rendering is not supervised.



NeuRay NeRF [4]

Figure 8. As an image-based rendering method, NeuRay is unable to correctly render the region that is not visible to all working views, even though such regions are visible on other images. In comparison, as a global scene representation, NeRF [4] can correctly render these regions.

functions to represent colors. In implementation, we use spherical harmonics up to 3 degree and thus $\mathbf{s} \in R^{16}$ and the diagonal elements of the regularization matrix $\mathbf{A}$ are $(0; 0.001; 0.005; 0.01)$ for degree 0 to 3.

Quality of direct rendering. To show the direct rendering of NeuRay can also produce reasonable renderings, we conduct experiments on the Lego and the Fern using the finetuned models. The qualitative results are shown in Fig. 7 and the quantitative results are shown in Table 6. The results demonstrate that though the direct rendering of NeuRay is not supervised by any rendering loss, it is still able to produce images with correct structures and details. Further supervising the direct rendering with a rendering loss may improve its rendering quality.

12. Limitations

As an image-based rendering method, our method needs to select a set of working views to render a novel view image. Hence, NeuRay is unable to render pixels that are invisible to all working views, though these pixels may be visible in other input views. We show an example in Fig. 8, in which the region is invisible to all working views. Meanwhile, our method is based on feature matching of input views to reconstruct the scene geometry. Such feature matching may struggle to find correct surfaces in textureless regions or cluttered complex regions like the Ficus. In this case, our method may not be able to keep the view consistency and correctly render, leading to rendering artifacts. Including more input views and working views or improving the feature matching may alleviate this problem. Another limitation is that speeding up rendering with NeuRay requires an accurate estimation of the surface locations so that such speeding up is only applicable to the finetuned model with accurate visibility. In the generalization setting, the predicted visibility is not accurate enough to maintain the rendering quality with very few sample points.

References

[1] Julian Chibane, Aayush Bansal, Verica Lazova, and Gerard Pons-Moll. Stereo radiance fields (SRF): Learning view synthesis for sparse views of novel scenes. CVPR, 2021. 2

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. CVPR, 2016. 2

[3] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014. 3

[4] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing scenes as neural radiance fields for view synthesis. In ECCV, 2020. 1, 3, 6

[5] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. Pixelwise view selection for unstructured multi-view stereo. In ECCV, 2016. 3, 4

[6] Alex Trevithick and Bo Yang. GRF: Learning a general radiance field for 3D scene representation and rendering. In ICCV, 2021. 2

[7] Qianqian Wang, Zhicheng Wang, Kyle Genova, Pratul Srinivasan, Howard Zhou, Jonathan T. Barron, Ricardo Martin-Brualla, Noah Snavely, and Thomas Funkhouser. IBRNet: Learning multi-view image-based rendering. CVPR, 2021. 2, 3, 4

[8] Suttisak Wizadwongsa, Pakkapon Phongthawee, Jiraphon Yenphraphai, and Supasorn Suwajanakorn. NeX: Real-time view synthesis with neural basis expansion. CVPR, 2021. 5

[9] Yao Yao, Zixin Luo, Shiwei Li, Tian Fang, and Long Quan. MVSNet: Depth inference for unstructured multi-view stereo. In ECCV, 2018. 2, 4

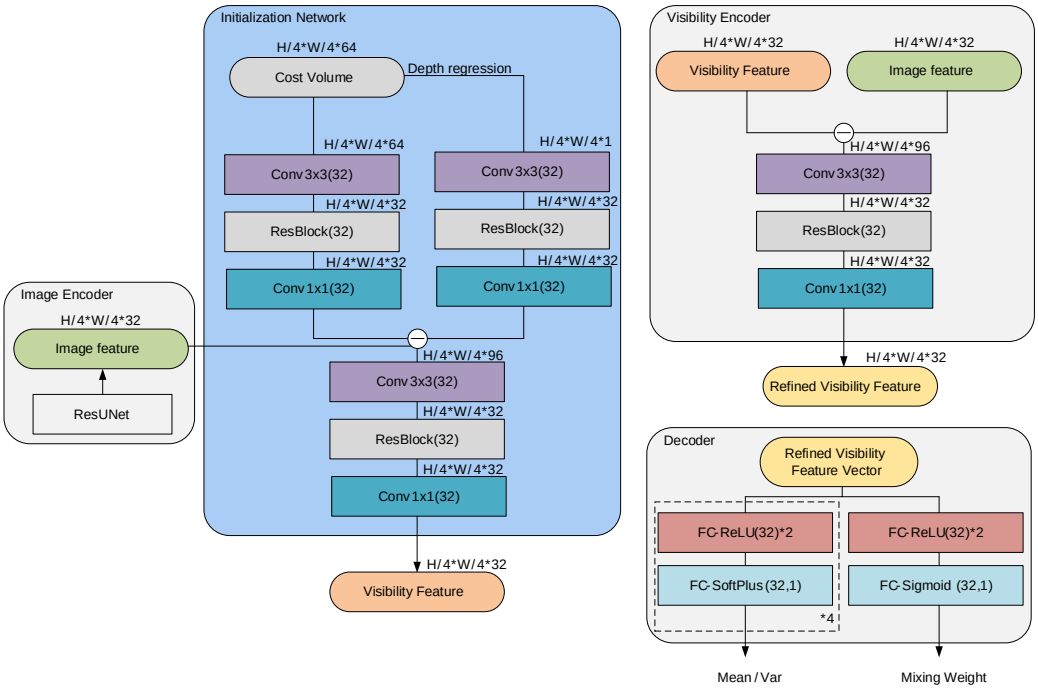


Figure 9. Architecture details of networks used in NeuRay.

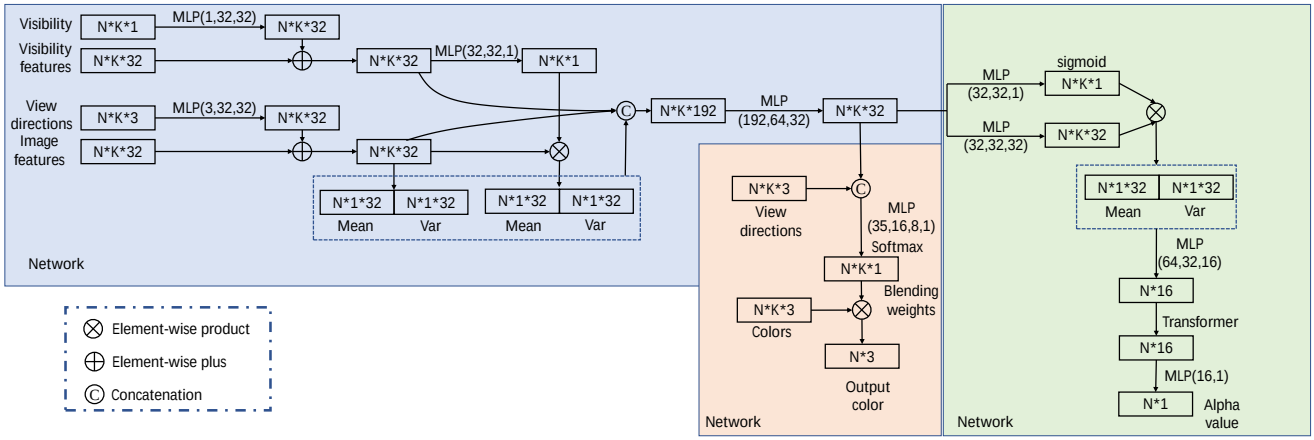


Figure 10. Architecture details of feature aggregation networks $(M; A; B)$. N is the number of sample points on a test ray and K is the number of working views.

[10] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. PlenOctrees for real-time rendering of neural radiance fields. In *ICCV*, 2021. 5

[11] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. PixelNeRF: Neural radiance fields from one or few images. In *CVPR*, 2021. 2