Supplementary Material for NeRF Revisited: Fixing Quadrature Instability in Volume Rendering

Mikaela Angelina Uy[1] **George Kiyohiro Nakayama**[1] **Guandao Yang**[1,2]
Rahul Krishna Thomas[1] **Leonidas Guibas**[1] **Ke Li**[3,4]
[1]Stanford University [2]Cornell University [3]Simon Fraser University [4]Google
{mikacuy, w4756677, guandao, rt03mas, guibas}@stanford.edu, keli@sfu.ca

We conduct further experiments, analysis and discussions on our proposed reformulation, where we take a piecewise *linear* approximation to opacity and piecewise *constant* approximation to color that results in an integral that is a simple and closed-form expression. This allows us to address the drawbacks of current piecewise constant assumption in NeRFs such as ray conflicts during optimization and a non-invertible CDF that lead to imprecise importance samples and vanishing gradients. We provide additional results in Sec S.1: ablation study (Sec S.1.1), a video demo (Sec S.1.2), additional results on a real dataset (Sec S.1.3), additional qualitative results (Sec S.1.4), comparison with PL-DIVeR S.1.5, additional geometric extraction results S.1.6 and comparison with less number of samples S.1.7. We then provide a walkthrough of the piecewise constant derivation from (2) (Sec S.2), which is followed by the thorough step-by-step derivation of our piecewsie linear opacity in volume rendering and precise importance sampling (Sec S.3). We also include analyses on piecewise quadratic and higher order polynomials in Sec S.4. Finally, we end with additional implementation and experiment details (Sec S.5), Limitations (Sec S.6) and Societal Impact (Sec S.7).

S.1 Additional Results

S.1.1 Ablation Study

We conduct a further ablation study on our precise importance sampling. As described in Sec. 4 in the main paper, our piecewise linear opacity approximation allows to solve for a closed-form solution for inverse transform sampling leading to the formulation of our **precise** importance sampling. Unlike the vanilla piecewise constant opacity approximation, our approach results in an invertible CDF, and hence we do not need to define an invertible surrogate function G for inverse transform sampling as in piecewise constant opacity (see Eq. 8 of main paper), which does not necessarily result in samples from the actual ray distribution $p(s)$. We quantitatively ablate the effectiveness of our precise importance sampling (**Precise**) by replacing our formulation (Eq. 13 main paper) with the surrogate function G as in the vanilla constant setting (**Surrogate**). As shown in Table S1 (first and second row of each metric), our precise importance sampling consistently outperforms using the surrogate on all metrics across all 8 scenes in the Blender dataset (hemisphere). Moreover, we also show that our precise importance sampling enables us to use fewer samples for the fine network as it is able to sample correctly from the ray distribution. Hence, keeping the same total number of rendering samples, we are able to achieve a further boost in performance by using 128 coarse and 64 fine samples as shown in Table S1 (second and third row of each metric). We use N_c and N_i to denote the number of coarse and fine samples, respectively.

S.1.2 Video Demo

We also ahow a video demo of our results. Please see our project page (pl-nerf.github.io) for the video (best viewed in full screen). In the mic scene, we observe that the structure inside the mic is lost in the piecewise constant opacity approximation. Moreover, we see the blurriness along the sides of the body of the mic caused by ray conflicts from the perpendicular and grazing angle views during optimization of the vanilla NeRF model. For the chair scene, we see that our **PL-NeRF** is able to

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

Metrics	Method	N_c	N_i	Avg.	Chair	Drums	Ficus	Hotdog	Lego	Mat.	Mic	Ship
PSNR↑	Surrogate	64	128	30.25	31.96	24.69	29.05	35.64	31.32	29.10	32.60	27.59
	Precise	64	128	30.87	32.85	24.96	29.61	**36.54**	32.27	29.35	**33.21**	28.22
	Precise	128	64	**31.10**	**32.92**	**25.07**	**30.18**	36.46	**32.90**	**29.52**	33.08	**28.71**
SSIM↑	Surrogate	64	128	0.940	0.962	0.914	0.957	0.973	0.953	0.943	0.978	0.841
	Precise	64	128	0.946	**0.969**	0.921	0.961	**0.977**	0.962	0.947	**0.982**	0.848
	Precise	128	64	**0.948**	**0.969**	**0.923**	**0.965**	**0.977**	**0.966**	**0.948**	0.981	**0.857**
LPIPS↓	Surrogate	64	128	5.50	3.85	8.52	4.15	2.70	2.94	4.04	2.30	15.5
	Precise	64	128	4.77	2.94	7.54	3.85	**2.27**	2.25	3.39	**1.59**	14.3
	Precise	128	64	**4.39**	**2.85**	**7.10**	**3.03**	2.28	**1.81**	**3.21**	1.73	**13.1**

Table S1: **Ablation Study.** Reported LPIPS scores are multiplied by 10^2. We use N_c and N_i to denote the number of coarse and fine samples, respectively.

achieve shaper textures on the back of the chair. Moreover, we see that there is an instability to the samples as illustrated by the inconsistencies in the noise – the gold specs on the green texture (please view in full screen). This is also evident when we vary the camera-to-scene distance. The chair model was trained on multi-distance Blender data to highlight the difference when camera distances vary.

S.1.3 Real Dataset Results

	PSNR↑	SSIM↑	LPIPS↓
Const. (Vanilla)	27.96	0.909	8.58
Linear (Ours)	**28.43**	**0.918**	**7.73**

Table S2: **DTU Quantative Results** Metrics computed from the average of 15 scenes from DTU dataset. The reported LPIPS score is multiplied by 10^2.

We further evaluate our **PL-NeRF** on a real dataset - DTU (1). We train and evaluate our approach on the 15 test scenes used in (6), and report the standard metrics (PSNR, SSIM, LPIPS). We follow the protocol used in (3) for the Real Forward Facing scene where $\frac{1}{8}$ of the views were held out for testing while the rest are used for training. Table S2 shows the quantitative results averaged over the 15 scenes, where our **PL-NeRF** outperforms the constant (3) baseline.

S.1.4 Additional Qualitative Results

We additionally show qualitative results from DTU dataset in Fig. S1. More qualitative results are also shown in Fig. S2.

Blender		Avg.	Chair	Drums	Ficus	Hotdog	Lego	Mat.	Mic	Ship
PSNR↑	DIVeR	30.78	32.01	**24.72**	30.1	**35.94**	29.03	29.31	32.10	**29.08**
	PL-DIVeR	**30.88**	**32.92**	24.7	**30.23**	**35.94**	**33.42**	**32.06**	**33.08**	28.99
SSIM↑	DIVeR	0.956	0.959	**0.917**	**0.963**	0.974	0.965	**0.977**	0.978	0.870
	PL-DIVeR	**0.947**	**0.969**	0.916	**0.963**	**0.966**	**0.966**	**0.977**	**0.981**	**0.871**
LPIPS↓	DIVeR	3.39	2.79	6.13	2.34	1.92	**1.46**	**1.77**	2.16	**7.77**
	PL-DIVeR	**3.28**	**2.85**	**6.01**	**2.12**	**1.83**	1.49	**1.77**	**1.73**	7.82

Table S3: **Quantitative Results of DIVeR v.s. PL-DIVeR** Reported LPIPS scores are multiplied by 10^2

S.1.5 PL-DiVER

We plug our method into DIVeR by using their voxed-based representation and feature integration, and dropping in our piecewise linear opacity formulation for volume rendering (PL-DIVeR). Results are shown in Table S3 demonstrating that our approach is on-par if not better across the different scenes in the Blender dataset. We highlight that this shows the improvement of using our piecewise linear opacity formulation, which is a drop-in replacement to existing methods.

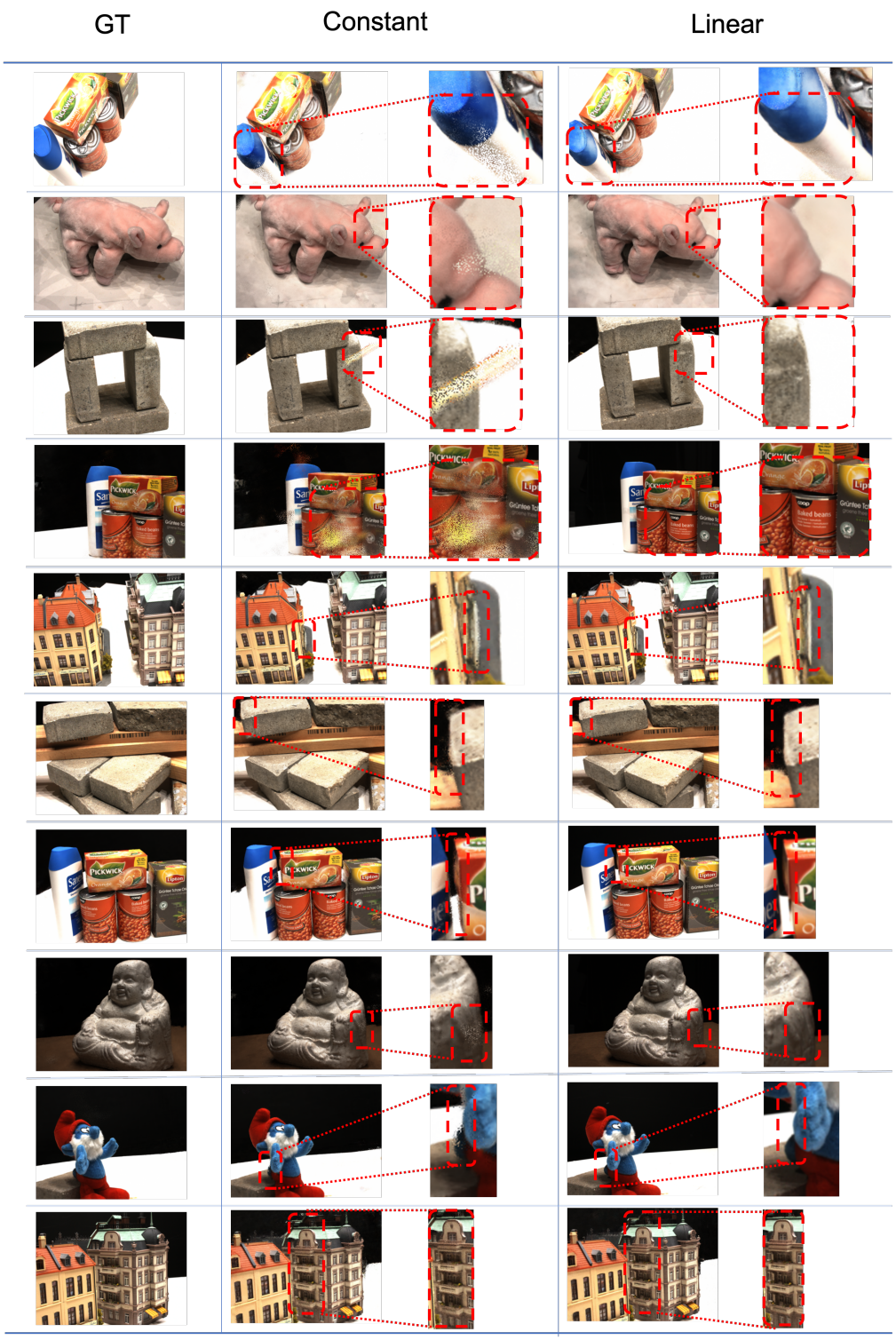

Figure S1: **DTU Qualitative Result** Visualizations of rendered DTU dataset test views. Because of the issue of grazing angle and binning inaccuracy of the piecewise constant opacity assumption, the vanilla NeRF (**constant**) exhibits blurry geometry, and rendering artifacts in the zoomed-in views (middle column). On the other hand, the piecewise linear opacity assumption in our model (**linear**) alleviates these issues (right column). Overall, the rendered views exhibit sharper surface boundaries and more faithful reconstruction compared to the constant model.

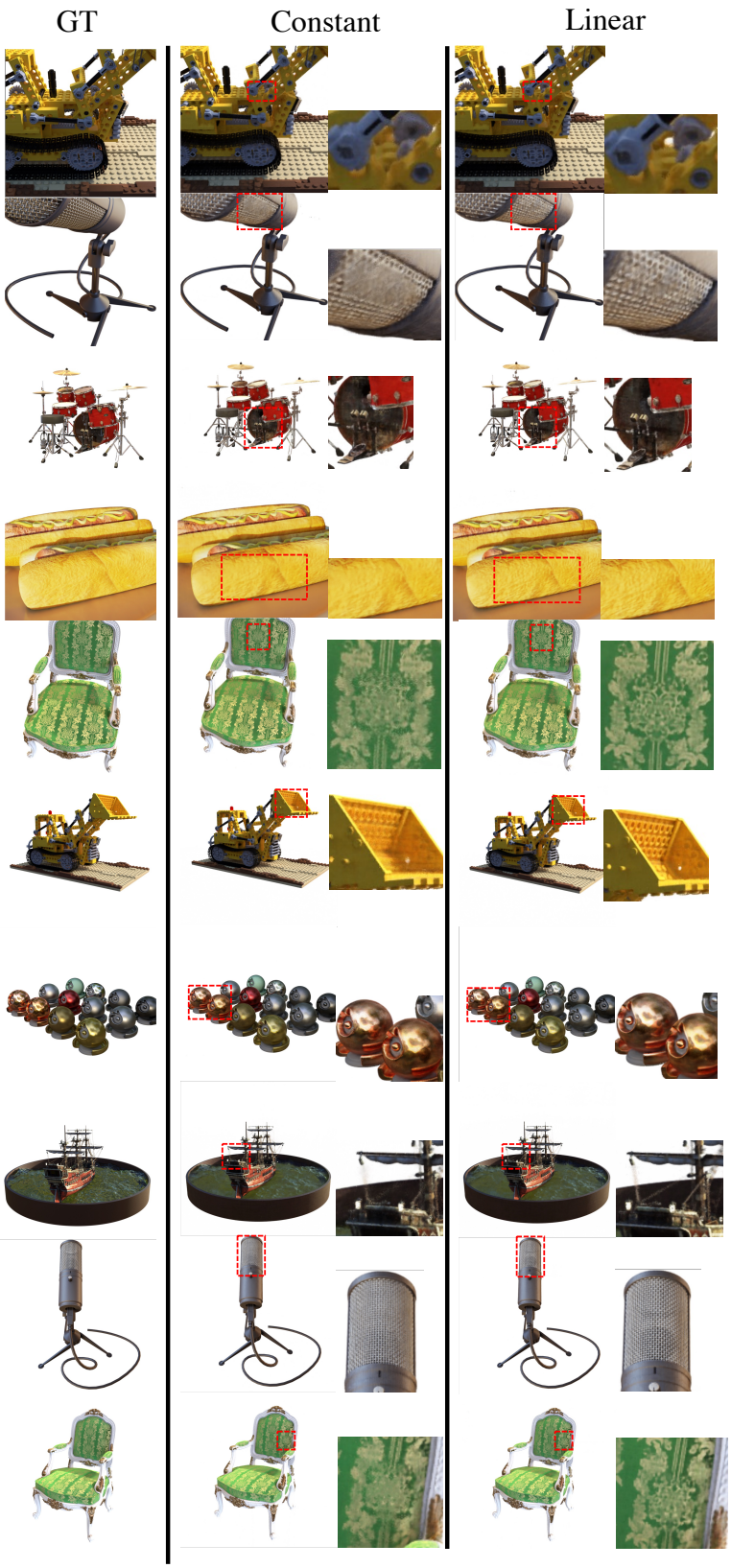

Figure S2: **Blender Qualitative Result** Additonal visualizations of rendered Blender dataset test views. We see that our **PL-NeRF** is able to achieve sharper and crisper texture (chair and hotdog surface), better capture fine geometric detail (hole in lego, rope in ship) and avoid blurriness caused by conflicting rays, e.g. grazing angle views as shown in the mic.

Blender		Avg.	Chair	Drums	Ficus	Hotdog	Lego	Mat.	Mic	Ship
CD↓	Vanilla NeRF	10.43	5.162	**6.842**	29.94	7.555	7.474	6.833	5.214	11.44
	PL-NeRF	**10.10**	**4.676**	7.754	**29.58**	**7.004**	**6.825**	**6.061**	**5.213**	**10.44**

Table S4: **Geometry Extraction Error** Distance between the surface of the GT to the predicted meshes. Scores are $\times 10^3$

S.1.6 Geometric Extraction

We also show quantitative results in geometric extraction improvement of PL-NeRF compared to the original Vanilla NeRF. Table S4 reports the distance between the surface of the ground truth model to the predicted meshes by sampling point clouds via ray casting. We see that our piecewise linear approach achieves a lower error compared to Vanilla NeRF on almost all the scenes in the Blender dataset.

S.1.7 Comparison with Less Samples

We run both our PL-NeRF and Vanilla NeRF with 64 coarse and 64 fine samples results in an average of (30.09, 0.939, 0.056) and (29.86, 0.937, 0.059) for (PSNR, SSIM, LPIPS), respectively, on the Blender dataset. This shows that under less number of samples our piecewise linear opacity formulation is better than the original piecewise constant opacity assumption.

S.2 Volume Rendering: Walkthrough of Piecewise Constant Derivation from (2)

From (2), under the approximation that both opacity and color are piecewise constant, for $s \in [s_i, s_{i+1}]$, where $\tau_i = \tau(s_i)$ and $\tau(s) = \tau_i \forall s \in [s_i, s_{i+1}]$, the probability of the interval P_i is derived as follows:

$$
\begin{aligned}
P_i &= \int_{s_i}^{s_{i+1}} \tau(s)T(s)\mathrm{d}s \\
&= \int_{s_i}^{s_{i+1}} \tau_i \exp\left(-\int_0^s \tau(u)\mathrm{d}u\right)\mathrm{d}s \\
&= \int_{s_i}^{s_{i+1}} \tau_i \exp\left(-\int_0^{s_i} \tau(u)\mathrm{d}u\right) \exp\left(-\int_{s_i}^s \tau(u)\mathrm{d}u\right)\mathrm{d}s \\
&= \int_{s_i}^{s_{i+1}} \tau_i T(s_i) \exp\left(-\int_{s_i}^s \tau_i \mathrm{d}u\right)\mathrm{d}s \\
&= \tau_i T(s_i) \int_{s_i}^{s_{i+1}} \exp\left(-\tau_i(s - s_i)\right)\mathrm{d}s \\
&= \tau_i T(s_i) \frac{\exp\left(-\tau_i(s - s_i)\right)}{-\tau_i}\Big|_{s_i}^{s_{i+1}} \\
&= \tau_i T(s_i)\Big(1 - \exp\left(-\tau_i(s_{i+1} - s_i)\right)\Big).
\end{aligned}
$$

Moreover, under the piecewise constant assumption, transmittance T is derived and given by:

$$
\begin{aligned}
T(s_i) &= \exp\left(-\int_0^{s_i} \tau(u)\mathrm{d}u\right) \\
&= \prod_{j=1}^{i} \exp\left(-\int_{s_{j-1}}^{s_j} \tau(u)\mathrm{d}u\right) \\
&= \prod_{j=1}^{i} \exp\left(-\int_{s_{j-1}}^{s_j} \tau_{j-1}\mathrm{d}u\right) \\
&= \prod_{j=1}^{i} \exp\left(\tau_{j-1}(s_j - s_{j-1})\right).
\end{aligned}
$$

This is the formulation that is used in most, if not all, NeRF works, which has the drawbacks that we raised such as ray conflicts during NeRF optimization and non-invertible CDF causing imprecise importance sampling and vanishing gradients when defining a loss w.r.t. the samples. We propose to approach this issue by deriving the volume rendering equation under a piecewise linear approximation to opacity, which we detail in the next sections.

S.3 Volume Rendering: Our Piecewise Linear τ Derivation

We now show our full derivation for the volume rendering equation, under the assumption that the opacity $\tau(s)$ is piecewise linear, i.e. it is linear within each interval $[s_i, s_{i+1}]$, and piecewise constant color. We then derive the probability of an interval under this assumption.

S.3.1 Generalized form for P_i.

Recall the generalized form of P_i as derived in the main paper. First from the definition of transmittance, we have

$$
\begin{aligned}
T(s) &= \exp\left(-\int_0^{s} \tau(u)\mathrm{d}u\right) \\
\frac{\mathrm{d}T}{\mathrm{d}s} &= -\exp\left(-\int_0^{s} \tau(u)\mathrm{d}u\right)\tau(s) = -T(s)\tau(s) \\
T'(s) &= -T(s)\tau(s).
\end{aligned}
$$

This results in the exact expression for the probability P_i of an interval given as follows:

$$
P_i = \int_{s_i}^{s_{i+1}} \tau(s)T(s)\,\mathrm{d}s = -\int_{s_i}^{s_{i+1}} T'(s)\,\mathrm{d}s = T(s_i) - T(s_{i+1}). \tag{1}
$$

S.3.2 Evaluating $\tau(s)$ for $s \in [s_i, s_{i+1}]$.

Let $\tau_j = \tau(s_i)$, $\tau_{i+1} = \tau(s_{i+1})$, be sampled points along the ray. For $s \in [s_i, s_{i+1}]$, assuming piecewise linear opacity τ, i.e. $\tau(s)$ is linear within each bin, we have

$$
\begin{aligned}
\tau(s) &= \left(\frac{s_{i+1} - s}{s_{i+1} - s_i}\right)\tau_i + \left(\frac{s - s_i}{s_{i+1} - s_i}\right)\tau_{i+1} \\
&= \frac{1}{s_{i+1} - s_i}\left[(\tau_{i+1} - \tau_i)s + (s_{i+1}\tau_i - s_i\tau_{i+1})\right]
\end{aligned}
$$

S.3.3 Transmittance $T(s_i)$

We first derive expression for transmittance $T(s_i)$ under the piecewise linear τ assumption.

$$
\begin{aligned}
T(s_i) &= \exp\left(-\int_0^{s_i} \tau(u)\mathrm{d}u\right) \\
&= \prod_{j=1}^{i} \exp\left(-\int_{s_{j-1}}^{s_j} \tau(u)\mathrm{d}u\right) \\
&= \prod_{j=1}^{i} \exp\left(\frac{-1}{s_j - s_{j-1}} \int_{s_{j-1}}^{s_j} [(\tau_j - \tau_{j-1})u + (s_j\tau_{j-1} - s_{j-1}\tau_j)]\mathrm{d}u\right) \\
&= \prod_{j=1}^{i} \exp\Big(\frac{-1}{s_j - s_{j-1}} \Big[(\frac{\tau_j - \tau_{j-1}}{2})(s_j^2 - s_{j-1}^2) + (s_j\tau_{j-1} - s_{j-1}\tau_j)(s_j - s_{j-1})\Big]\Big) \\
&= \prod_{j=1}^{i} \exp\Big(-\Big[(\frac{\tau_j - \tau_{j-1}}{2})(s_j + s_{j-1}) + (s_j\tau_{j-1} - s_{j-1}\tau_j)\Big]\Big) \\
&= \prod_{j=1}^{i} \exp\Big(-\frac{1}{2}\Big[\tau_j s_j + \tau_j s_{j-1} - \tau_{j-1}s_j - \tau_{j-1}s_{j-1} + 2s_j\tau_{j-1} - 2s_{j-1}\tau_j\Big]\Big) \\
&= \prod_{j=1}^{i} \exp\Big(-\frac{1}{2}\Big[\tau_j s_j - \tau_j s_{j-1} + \tau_{j-1}s_j - \tau_{j-1}s_{j-1}\Big]\Big)
\end{aligned}
$$

Thus we get

$$
\boxed{T(s_i) = \prod_{j=1}^{i} \exp\Big(-\frac{(\tau_j + \tau_{j-1})(s_j - s_{j-1})}{2}\Big)}. \tag{2}
$$

S.3.4 Probability of interval $[s_i, s_{i+1}]$

From the generalized form for P_i as derived in the main paper, we plug in the expression for transmittance $T(s_i)$ as derived above to obtain:

$$
\begin{aligned}
P_i &= \int_{s_i}^{s_{i+1}} \tau(s)T(s)\mathrm{d}s \\
&= T(s_i) - T(s_{i+1}) \\
&= \prod_{j=1}^{i} \exp\Big(-\frac{(\tau_j + \tau_{j-1})(s_j - s_{j-1})}{2}\Big) - \prod_{j=1}^{i+1} \exp\Big(-\frac{(\tau_j + \tau_{j-1})(s_j - s_{j-1})}{2}\Big)
\end{aligned}
$$

Hence, we obtain

$$
\boxed{P_i = T(s_i) \cdot \Big(1 - \exp\Big[-\frac{(\tau_{i+1} + \tau_i)(s_{i+1} - s_i)}{2}\Big]\Big)}. \tag{3}
$$

S.3.5 Our Precision Importance Sampling

To sample from the ray distribution, inverse transform sampling is needed, that is, one draws $u \sim U(0, 1)$ then passes it to the inverse of a cumulative distribution (CDF), i.e. a sample $x = F^{-1}(u)$,

where F is the CDF of the distribution. Unlike the piecewise constant case, where F is not invertible, needing for a surrogate function G derived from F, we show that under our piecewise linear opacity assumption, we can solve for the solution x for each corresponding u.

As illustrated in the main paper, since F is continuous and increasing, under our assumption that $\tau > 0$ [1], then F is invertible. Now, without loss of generality, let sample u fall into the CDF interval $[c_k, c_{k+1}]$, where $c_k = \sum_{j<k} P_j$. We know that the probability of the corresponding interval is P_k as given by Eq. 3. Thus we have:

$$\begin{aligned} c_{k+1} - c_k &= P_k \\ &= \int_{s_k}^{s_{k+1}} T(s)\tau(s)\mathrm{d}s \\ &= T(s_k) \cdot \Big(1 - \exp\Big(-\frac{(\tau_{k+1} + \tau_k)(s_{k+1} - s_k)}{2}\Big)\Big) \end{aligned}$$

We want to solve for sample $x \in [s_k, s_{k+1}]$, such that $x = F^{-1}(u)$. Equivalently, since we know that $x \in [s_k, s_{k+1}]$, then we reparameterize and let $x = s_k + t$, where $t \in [0, s_{k+1} - s_k]$. We are solving for x as follows:

$$\begin{aligned} u &= \int_0^x T(s)\tau(s)\mathrm{d}s \\ &= \int_0^{s_k} T(s)\tau(s)\mathrm{d}s + \int_{s_k}^x T(s)\tau(s)\mathrm{d}s \\ &= c_k + \int_{s_k}^x T(s)\tau(s)\mathrm{d}s \\ u - c_k &= \int_{s_k}^x T(s)\tau(s)\mathrm{d}s \end{aligned}$$

Now, from the derivation of the general form for P_i in Eq. 1, we can similarly obtain

$$\begin{aligned} u - c_k &= T(s_k) - T(x). \\ &= T(s_k) \cdot (1 - \exp(-\int_{s_k}^x \tau(u)\mathrm{d}u)). \end{aligned}$$

Thus, simplifying we get

$$\begin{aligned} \frac{u - c_k}{T(s_k)} &= 1 - \exp(-\int_{s_k}^x \tau(u)\mathrm{d}u) \\ \exp(-\int_{s_k}^x \tau(u)\mathrm{d}u) &= 1 - \frac{u - c_k}{T(s_k)} \\ -\int_{s_k}^x \tau(u)\mathrm{d}u) &= \ln\Big(1 - \frac{u - c_k}{T(s_k)}\Big) \end{aligned}$$

[1] In practice, we can simply add a small ϵ, say $\epsilon = 10^{-6}$, to the model output resulting in positive τ, to make τ positive everywhere.

which gives us the expression

$$\int_{s_k}^{x} \tau(u)\mathrm{d}u = \ln(T(s_k)) - \ln(T(s_k) - (u - c_k)). \tag{4}$$

This holds for $T(s_k) \neq 0$, which is true under our assumption that $\tau > 0$, and $T(s_k) - (u - c_k) \geq 0$, which we will show in Sec. S.3.8 below.

S.3.6 Evaluating $-\int_{s_k}^{x} \tau(u)\mathrm{d}u$.

We evaluate the expression $-\int_{s_k}^{x} \tau(u)\mathrm{d}u$ in order to solve for the exact sample x. Recall, for $s \in [s_k, s_{k+1}]$, we have

$$\begin{aligned}
\tau(s) &= (\frac{s_{k+1} - s}{s_{k+1} - k_i})\tau_k + (\frac{s - k_i}{s_{k+1} - s_k})\tau_{k+1} \\
&= \frac{1}{s_{k+1} - s_k}[(\tau_{k+1} - \tau_k)s + (s_{k+1}\tau_k - s_k\tau_{k+1})]
\end{aligned}$$

Let constants $a = \tau_{k+1} - \tau_k$, $b = s_{k+1}\tau_k - s_k\tau_{k+1}$, $d = \frac{1}{s_{k+1} - s_k}$, thus we can write $\tau(s)$ as follows:

$$\tau(s) = d(as + b). \tag{5}$$

Thus, from Eq 5 we have:

$$\begin{aligned}
\int_{s_k}^{x} \tau(u)\mathrm{d}u &= \int_{s_k}^{x} d(au + b)\mathrm{d}u \\
&= d(\int_{s_k}^{x} au\mathrm{d}u + \int_{s_k}^{x} b\mathrm{d}u) \\
&= d[\frac{au^2}{2}\Big|_{s_k}^{x} + b(x - s_i)] \\
&= d[\frac{a(x^2 - s_k^2)}{2} + b(x - s_k)] \\
&= d[\frac{a((s_k + t)^2 - s_k^2)}{2} + b((s_k + t) - s_k)] \\
&= d[\frac{a(t^2 + 2s_k t)}{2} + bt] \\
&= d[\frac{a}{2}t^2 + (as_k + b)t] \\
&= \frac{1}{s_{k+1} - s_k}[\frac{\tau_{k+1} - \tau_k}{2}t^2 + ((\tau_{k+1} - \tau_k)s_k + s_{k+1}\tau_k - s_k\tau_{k+1})t] \\
&= \frac{1}{s_{k+1} - s_k}[\frac{\tau_{k+1} - \tau_k}{2}t^2 + (s_{k+1}\tau_k - s_k\tau_k)t] \\
&= \frac{\tau_{k+1} - \tau_k}{2(s_{k+1} - s_k)}t^2 + \tau_k t
\end{aligned}$$

Hence, plugging this in Eq. 4, we get the quadratic equation

$$\frac{\tau_{k+1} - \tau_k}{2(s_{k+1} - s_k)}t^2 + \tau_k t - (\ln(T(s_k)) - \ln(T(s_k) - (u - c_k))) = 0.$$

We want to solve for $t \in [0, s_{k+1} - s_k]$, and the roots of the quadratic equation is given by

$$t = \frac{(s_{k+1} - s_k)(-\tau_k \pm \sqrt{\tau_k^2 + \frac{2(\tau_{k+1}-\tau_k)(\ln T(s_k) - \ln(T(s_k)-(u-c_k)))}{(s_{k+1}-s_k)}})}{(\tau_{k+1} - \tau_k)} \tag{6}$$

That means to compute for the solution, we need to find the root

$$\frac{(-\tau_k \pm \sqrt{\tau_k^2 + \frac{2(\tau_{k+1}-\tau_k)(\ln T(s_k) - \ln(T(s_k)-(u-c_k)))}{(s_{k+1}-s_k)}})}{(\tau_{k+1} - \tau_k)} \in (0, 1)$$

which we will show always exists and is unique.

S.3.7 Bounding $\Delta = \tau_k^2 + \frac{2(\tau_{k+1}-\tau_k)(\ln T(s_k) - \ln(T(s_k)-(u-c_k)))}{(s_{k+1}-s_k)}$

Let us first bound the discriminant of the quadratic formula. We have

$$\begin{aligned} u - c_k &\leq c_{k+1} - c_k \\ &= \sum_{j=0}^{k} P_j - \sum_{j=0}^{k-1} P_j = P_k \\ &= T(s_k) \cdot (1 - \exp\Big(-\frac{(\tau_{k+1} + \tau_k)(s_{k+1} - s_k)}{2}\Big)) \end{aligned}$$

Thus we have

$$\begin{aligned} \ln(T(s_k) - (u - c_k)) &\geq \ln\Big(T(s_k) - T(s_k) \cdot (1 - \exp\Big(-\frac{(\tau_{k+1} + \tau_k)(s_{k+1} - s_k)}{2}\Big))\Big) \\ &= \ln(T(s_k) \cdot \exp\Big(-\frac{(\tau_{k+1} + \tau_k)(s_{k+1} - s_k)}{2}\Big)) \\ &= \ln(T(s_k)) - \frac{(\tau_{k+1} + \tau_k)(s_{k+1} - s_k)}{2} \\ \ln T(s_k) - \ln(T(s_k) - (u - c_k)) &\leq \frac{(\tau_{k+1} + \tau_k)(s_{k+1} - s_k)}{2} \end{aligned}$$

$$\begin{aligned} \frac{2(\tau_{k+1} - \tau_k)(\ln T(s_k) - \ln(T(s_k) - (u - c_k)))}{(s_{k+1} - s_k)} &\leq \frac{2(\tau_{k+1} - \tau_k)(\frac{(\tau_{k+1}+\tau_k)(s_{k+1}-s_k)}{2})}{(s_{k+1} - s_k)} \\ &= \tau_{k+1}^2 - \tau_k^2 \end{aligned}$$

Hence, computing the discriminant we get:

$$\Delta = \tau_k^2 + \frac{2(\tau_{k+1} - \tau_k)(\ln T(s_k) - \ln(T(s_k) - (u - c_k)))}{(s_{k+1} - s_k)} \quad \leq \tau_k^2 + (\tau_{k+1}^2 - \tau_k^2) = \tau_{k+1}^2.$$

Similarly, $u - c_k \geq 0$, where equality holds when $u = c_k$. This gives us $\Delta \geq \tau_k^2$.

Hence, we know that $\tau_{k+1}^2 \geq \Delta \geq \tau_k^2$. Since we need

$$\frac{(-\tau_k \pm \sqrt{\Delta})}{(\tau_{k+1} - \tau_k)} \in (0, 1), \text{and } \frac{(-\tau_k - \sqrt{\Delta})}{(\tau_{k+1} - \tau_k)} \leq 0$$

Thus to find the solution t, we need to take the positive root. We have

$$\frac{(-\tau_k + \sqrt{\Delta})}{(\tau_{k+1} - \tau_k)} \geq \frac{(-\tau_k + \sqrt{\tau_k^2}}{(\tau_{k+1} - \tau_k)} = 0$$

$$\frac{(-\tau_k + \sqrt{\Delta})}{(\tau_{k+1} - \tau_k)} \leq \frac{(-\tau_k + \sqrt{\tau_{k+1}^2}}{(\tau_{k+1} - \tau_k)} = 1$$

This shows that the solution is within the desired interval. Hence, the solution is

$$t = \frac{(s_{k+1} - s_k)(-\tau_k + \sqrt{\tau_k^2 + \frac{2(\tau_{k+1} - \tau_k)(\ln T(s_k) - \ln(T(s_k) - (u - c_k)))}{(s_{k+1} - s_k)}})}{(\tau_{k+1} - \tau_k)} \tag{7}$$

Hence, for the positive root, we know that $t \in [0, s_{k-1} - s_k]$.

S.3.8 Proof for $T(s_k) \geq (u - c_k)$

$$\begin{aligned} c_k &= \sum_{j=0}^{k-1} P_j \\ &= \sum_{j=0}^{k-1} T(s_j) \cdot (1 - \exp(-\frac{(\tau_{j+1} + \tau_j)(s_{j+1} - s_j)}{2})) \end{aligned}$$

We know that

$$T(s_j) = \prod_{i=1}^{j} \exp(-\frac{(\tau_i + \tau_{i-1})(s_i - s_{i-1})}{2})$$

Let $a_i = \exp(-\frac{(\tau_i + \tau_{i-1})(s_i - s_{i-1})}{2})$, and $T(s_0) = 1$. Hence we have

$$T(s_j) = \prod_{i=1}^{j} a_i,$$

$$\begin{aligned} c_k &= T(s_0)(1 - a_1) + \sum_{j=1}^{k-1} (\prod_{i=1}^{j} a_i) \cdot (1 - a_{j+1})) \\ &= (1 - a_1) + (a_1)(1 - a_2) + (a_1 a_2)(1 - a_3) + ... \\ &= 1 - a_1 a_2 ... a_k \end{aligned}$$

Since $T(s_k) = a_1 a_2 ... a_k$ and $a_i > 0 \forall i$ then

$$\begin{aligned} T(s_k) + c_k &= a_1 a_2 ... a_k + (1 - a_1 a_2 ... a_k) \\ &= 1 \\ &\geq c_{k+1} \\ &\geq u. \square \end{aligned}$$

Note that above proof and solutions hold for $\tau_k \neq \tau_{k+1}$ and $T(s_k) \neq 0$, which is all holds since we have $\tau(s) > 0 \forall s$, which is equivalent to F being an increasing function.

S.3.9 The solution for sample u:

Putting everything together, we have the solution t given as

$$t = \frac{(s_{k+1} - s_k)(-\tau_k + \sqrt{\tau_k^2 + \frac{2(\tau_{k+1}-\tau_k)(\ln T(s_k) - \ln(T(s_k)-(u-c_k)))}{(s_{k+1}-s_k)}})}{(\tau_{k+1} - \tau_k)}. \tag{8}$$

From Sec S.3.8, we have $T(s_k) = a_1 a_2 ... a_k$ and $c_k = 1 - a_1 a_2 ... a_k$, thus $T(s_k) - (u - c_k) = 1 - u$. Hence we can simplify it to

$$t = \frac{(s_{k+1} - s_k)(-\tau_k + \sqrt{\tau_k^2 + \frac{2(\tau_{k+1}-\tau_k)(\ln T(s_k) - \ln(1-u))}{(s_{k+1}-s_k)}})}{(\tau_{k+1} - \tau_k)}$$

$$\boxed{t = \frac{(s_{k+1} - s_k)(-\tau_k + \sqrt{\tau_k^2 + \frac{2(\tau_{k+1}-\tau_k)(-\ln \frac{(1-u)}{T(s_k)})}{(s_{k+1}-s_k)}})}{(\tau_{k+1} - \tau_k)}}. \tag{9}$$

S.4 Piecewise Quadratic and Higher Order Polynomials

Now, we first consider the full derivation for volume rendering equation under the assumption that opacity is *piecewise quadratic* and color is piecewise constant. Consider opacities $\tau_1, \ldots, \tau_n$ queried at n samples $s_1, \ldots, s_n$ along the ray. Here, we set $s_0 = t_n$ and $s_{n+1} = t_f$ to the near and far plane, with $\tau_0 = 0$ and $\tau_{n+1} = 10^{10}$ denoting empty and opaque space.

To interpolate opacity, because a quadratic function can only be uniquely defined with 3 points, we choose $\tau(s)$ to be quadratic within each interval $[s_j, s_{j+2}]$ for even values of j. To encapsulate all points, this forces n to be *odd*, e.g. using 127 coarse samples and 64 fine samples.

S.4.1 Derivation: Computing $T(s)$

In the same way as Sec. S.3.3, we can derive transmittance, which is in closed-form. The only modification is that the formulae for integrals of opacity are different over left and right subintervals $[s_j, s_{j+1}]$ and $[s_{j+1}, s_{j+2}]$, with j even (see the next section for a derivation of these integrals):

$$T(s_{2i}) = \exp\left(-\int_{s_0}^{s_{2i}} \tau(u)\mathrm{d}u\right) = \prod_{j=1}^{i} \exp\left(-\int_{s_{2j-1}}^{s_{2j}} \tau(u)\mathrm{d}u\right) \exp\left(-\int_{s_{2j}}^{s_{2j+1}} \tau(u)\mathrm{d}u\right). \tag{10}$$

Substituting the expressions in Eqs. 14 and 15 gives a closed form expression in terms of the s_j's and τ_j's. We can similarly compute

$$T(s_{2i+1}) = \exp(-\int_{s_{2i}}^{s_{2i+1}} \tau(u)\mathrm{d}u) \prod_{j=1}^{i} \exp\left(-\int_{s_{2j-1}}^{s_{2j}} \tau(u)\mathrm{d}u\right) \exp\left(-\int_{s_{2j}}^{s_{2j+1}} \tau(u)\mathrm{d}u\right). \tag{11}$$

Then the probability of the ith interval for each $0 \leq i \leq n$ is, as before,

$$P_i = T(s_i) - T(s_{i+1}). \tag{12}$$

This leads to a closed-form expression for P_i. This means the behavior of P_i depends entirely on that of the opacity integral. However, due to the form of $\int \tau(s)ds$ detailed in the next section, this means P_i involves a *piecewise exponential of a rational function in τ_j's and s_j's*, which leads to poor numerical conditioning and thus optimization instability.

S.4.2 Derivation: Computing and Integrating $\tau(s)$ on Intervals

We now compute $\tau(s)$ and its integral on each interval. Fix the interval $[s_j, s_{j+2}]$, with j odd, and $\tau_j = \tau(s_j)$, $\tau_{j+1} = \tau(s_{j+1})$, $\tau_{j+2} = \tau(s_{j+2})$. By *Lagrange interpolation*, the quadratic $\tau(s)$ passing through $(s_j, \tau_j), (s_{j+1}, \tau_{j+1}), (s_{j+2}, \tau_{j+2})$ is given by:

$$\tau(s) = \frac{\tau_j}{\alpha_j\gamma_j}(s - s_{j+1})(s - s_{j+2}) - \frac{\tau_{j+1}}{\alpha_j\beta_j}(s - s_j)(s - s_{j+2}) + \frac{\tau_{j+2}}{\beta_j\gamma_j}(s - s_j)(s - s_{j+1}),$$
$$\alpha_j = s_{j+1} - s_j, \quad \beta_j = s_{j+2} - s_{j+1}, \quad \gamma_j = s_{j+2} - s_j. \tag{13}$$

Note the integrals of the three monic quadratics over $[s_j, s_{j+1}]$ can be expressed in terms of $\alpha_j, \beta_j, \gamma_j$:

$$\int_{s_j}^{s_{j+1}} (s - s_{j+1})(s - s_{j+2})\mathrm{d}s = \int_{-\alpha_j}^{0} s(s - \beta_j)\mathrm{d}s = \frac{\alpha_j^3}{3} + \frac{\alpha_j^2\beta_j}{2},$$
$$\int_{s_j}^{s_{j+1}} (s - s_j)(s - s_{j+2})\mathrm{d}s = \int_{0}^{\alpha_j} s(s - \gamma_j)\mathrm{d}s = \frac{\alpha_j^3}{3} - \frac{\alpha_j^2\gamma_j}{2},$$
$$\int_{s_j}^{s_{j+1}} (s - s_j)(s - s_{j+1})\mathrm{d}s = \int_{0}^{\alpha_j} s(s - \alpha_j)\mathrm{d}s = -\frac{\alpha_j^3}{6}.$$

Thus, using Eq. 13 gives the integral of opacity over $[s_j, s_{j+1}]$:

$$\int_{s_j}^{s_{j+1}} \tau(s)\mathrm{d}s = \frac{\tau_j}{\gamma_j} \cdot \left[\frac{\alpha_j^2}{3} + \frac{\alpha_j\beta_j}{2}\right] - \frac{\tau_{j+1}}{\beta_j}\left[\frac{\alpha_j^2}{3} - \frac{\alpha_j\gamma_j}{2}\right] + \frac{\tau_{j+2}}{\beta_j\gamma_j} \cdot \left[-\frac{\alpha_j^3}{6}\right]. \tag{14}$$

Similarly, the integrals of those same three quadratics over $[s_{j+1}, s_{j+2}]$ factor out β_j:

$$\int_{s_{j+1}}^{s_{j+2}} (s - s_{j+1})(s - s_{j+2})\mathrm{d}s = \int_{0}^{\beta_j} s(s - \beta_j)\mathrm{d}s = -\frac{\beta_j^3}{6},$$
$$\int_{s_{j+1}}^{s_{j+2}} (s - s_j)(s - s_{j+2})\mathrm{d}s = \int_{-\beta_j}^{0} (s + \gamma_j)s\mathrm{d}s = \frac{\beta_j^3}{3} - \frac{\beta_j^2\gamma_j}{2},$$
$$\int_{s_{j+1}}^{s_{j+2}} (s - s_j)(s - s_{j+1})\mathrm{d}s = \int_{0}^{\beta_j} (s + \alpha_j)s\mathrm{d}s = \frac{\beta_j^3}{3} + \frac{\beta_j^2\alpha_j}{2}.$$

Then, summing these up along Eq. 13 gives the integral of opacity over $[s_{j+1}, s_{j+2}]$:

$$\int_{s_{j+1}}^{s_{j+2}} \tau(s)\mathrm{d}s = \frac{\tau_j}{\alpha_j\gamma_j} \cdot \left[-\frac{\beta_j^3}{6}\right] - \frac{\tau_{j+1}}{\alpha_j}\left[\frac{\beta_j^2}{3} - \frac{\beta_j\gamma_j}{2}\right] + \frac{\tau_{j+2}}{\gamma_j} \cdot \left[\frac{\beta_j^2}{3} + \frac{\beta_j\alpha_j}{2}\right]. \tag{15}$$

Observe in Eqs. 14 and 15 that the integral of opacity involves some terms with $\alpha_j, \beta_j, \gamma_j$ in the denominator, which *do not cancel*. Thus, the integral of opacity over an interval *is not a polynomial in τ_i's and s_i's*, but is instead a *rational function*. This contrasts with the linear derivation in Sec. S.3.3, where this integral was a degree 2 multivariate polynomial in τ_i's and s_i's. This caveat causes numerical instability, which will be discussed further in Secs. S.4.3 and S.4.4.

Generally, following the steps of the above derivation shows that if we interpolate τ piecewise by *any degree d polynomial, $d \geq 2$,* then the result is a rational function in s_i's and τ_i's, but not a polynomial, which would also lead to training instability as in the quadratic case.

S.4.3 Piecewise Quadratic Problem 1: Negative Interpolated Opacity

As seen above, one problem with the piecewise quadratic model is that the integral of opacity is a *rational function in* $\alpha_i, \beta_i, \gamma_i$. In particular, due to the presence of negatives in front of certain rational terms in Eqs. 14 and 15, it may become *negative* as the denominators of these terms approach zero. One example is shown in Figure S.4.3, for samples s_1, s_2, s_3 with $\tau_1 = \tau_2 < \tau_3$ and $s_3 - s_2 \ll s_2 - s_1$, where the interpolated quadratic dips far below the x-axis. Note this interpolation is physically implausible, as *opacity should be nonnegative everywhere*.

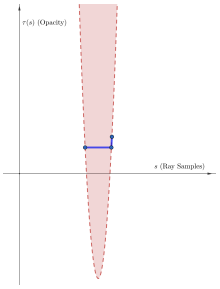

Figure S3: Interpolation gives negative τ-values when s-values are close.

Furthermore, from Eqs. 10 and 11, when the opacity integral is negative on intervals, transmittance can then be a product with exponentials of negative terms, and potentially be greater than 1. This is incompatible with the physical interpretation of transmittance as a *probability* that a light travels a distance along a ray without being absorbed.

S.4.4 Piecewise Quadratic Problem 2: Instability from Sample Proximity

In Eqs. 14 and 15, we observe the presence of terms such as $\alpha_j, \beta_j, \gamma_j$ in the denominator. *These do not appear in the piecewise linear model.* As a result, because these quantities approach zero as samples s_j, s_{j+1}, s_{j+2} become closer, then the integral of opacity can become an arbitrarily large positive or even *negative* (as shown qualitatively in the previous section). Referencing the transmittance formulae in Eqs. 10 and 11, this means transmittance can approach zero or infinity and gradients can explode as samples are clustered together. This is fairly common in stratified sampling, and even more so with importance sampling.

To formally describe this instability, we first look at what happens when s_j and s_{j+1} coincide:

$$\lim_{s_{j+1} \to s_j^+} \int_{s_j}^{s_{j+1}} \tau(s) \mathrm{d}s = 0$$

because $\tau : \mathbb{R}_{\geq 0} \to \mathbb{R}$ is continuous. There is no instability in this integral when $s_{j+1} \to s_j^+$. However, when s_{j+1} and s_{j+2} coincide, because $\beta_j \to 0^+$ and $\gamma_j \to \alpha_j^+$, the integral approaches

$$\begin{aligned}
\lim_{s_{j+2} \to s_{j+1}^+} \int_{s_j}^{s_{j+1}} \tau(s) \mathrm{d}s &= \lim_{s_{j+2} \to s_{j+1}^+} \left[\frac{\tau_j}{\gamma_j} \cdot \left[\frac{\alpha_j^2}{3} + \frac{\alpha_j \beta_j}{2} \right] - \frac{\tau_{j+1}}{\beta_j} \left[\frac{\alpha_j^2}{3} - \frac{\alpha_j \gamma_j}{2} \right] + \frac{\tau_{j+2}}{\beta_j \gamma_j} \cdot \left[-\frac{\alpha_j^3}{6} \right] \right] \\
&= \frac{\tau_j \alpha_j}{3} - \lim_{s_{j+2} \to s_{j+1}^+} \frac{\frac{\tau_{j+1}\alpha_j^2}{3} - \frac{\tau_{j+1}\alpha_j\gamma_j}{2} + \frac{\tau_{j+2}\alpha_j^3}{6\gamma_j}}{\beta_j} \\
&= \frac{\tau_j \alpha_j}{3} - \lim_{s_{j+2} \to s_{j+1}^+} \frac{2\tau_{j+1}\alpha_j^2\gamma_j - 3\tau_{j+1}\alpha_j\gamma_j^2 + \tau_{j+2}\alpha_j^3}{6\gamma_j\beta_j} \\
&= \frac{\tau_j \alpha_j}{3} - \lim_{s_{j+2} \to s_{j+1}^+} \frac{\tau_{j+1}\alpha_j(\alpha_j^2 + 2\alpha_j\gamma_j - 3\gamma_j^2) + \alpha_j^3(\tau_{j+2} - \tau_{j+1})}{6\gamma_j\beta_j} \\
&= \frac{\tau_j \alpha_j}{3} - \lim_{s_{j+2} \to s_{j+1}^+} \frac{\tau_{j+1}\alpha_j(\alpha_j + 3\gamma_j)(\alpha_j - \gamma_j) + \alpha_j^3(\tau_{j+2} - \tau_{j+1})}{6\gamma_j\beta_j} \\
&= \frac{\tau_j \alpha_j}{3} - \lim_{s_{j+2} \to s_{j+1}^+} \frac{-\tau_{j+1}\alpha_j(\alpha_j + 3\gamma_j)\beta_j + \alpha_j^3(\tau_{j+2} - \tau_{j+1})}{6\gamma_j\beta_j}
\end{aligned}$$

$$= \frac{\tau_j \alpha_j}{3} + \frac{2\tau_{j+1}\alpha_j}{3} - \lim_{s_{j+2} \to s_{j+1}^+} \frac{\alpha_j^3(\tau_{j+2} - \tau_{j+1})}{6\gamma_j \beta_j}$$
$$= \frac{\tau_j \alpha_j}{3} + \frac{2\tau_{j+1}\alpha_j}{3} - \frac{\alpha_j^2}{6} \lim_{s_{j+2} \to s_{j+1}^+} \frac{\tau_{j+2} - \tau_{j+1}}{\beta_j}.$$

The behavior of this integral limit depends on the last limit. To analyze this, let $h : \mathbb{R}_{\geq 0} \to \mathbb{R}$ be the network function which takes in samples on the ray and outputs opacity. The last limit is

$$\lim_{s_{j+2} \to s_{j+1}^+} \frac{\tau_{j+2} - \tau_{j+1}}{\beta_j} = \lim_{s_{j+2} \to s_{j+1}^+} \frac{h(s_{j+2}) - h(s_{j+1})}{s_{j+2} - s_{j+1}}. \tag{16}$$

Because h is almost always differentiable at s_{j+1}, this becomes $h'(s_{j+1})$, and so

$$\lim_{s_{j+2} \to s_{j+1}^+} \int_{s_j}^{s_{j+1}} \tau(s) \mathrm{d}s = \frac{\tau_j \alpha_j}{3} + \frac{2\tau_{j+1}\alpha_j}{3} - \frac{\alpha_j^2}{6} h'(s_{j+1}). \tag{17}$$

There are no constraints on the value of $h'(s_{j+1})$ as the network trains, so the above limit can achieve any real value. In particular, we can derive a condition for when the limit is negative:

$$\lim_{s_{j+2} \to s_{j+1}^+} \int_{s_j}^{s_{j+1}} \tau(s) \mathrm{d}s < 0 \iff h'(s_{j+1}) > \frac{2\tau_j + 4\tau_{j+1}}{\alpha_j}. \tag{18}$$

That is, whenever there is a sharp enough increase in opacity (which can happen at, say, a surface crossing) and samples s_{j+1}, s_{j+2} are sufficiently close, the interpolated quadratic can have a negative integral on $[s_j, s_{j+1}]$, which leads to the issues described in Sec. S.4.3. *In other words, the integral of opacity over $[s_j, s_{j+1}]$ is unstable as $s_{j+2} \to s_{j+1}^+$.* A similar analysis holds to show the instability of the opacity integral on $[s_{j+1}, s_{j+2}]$ as $s_{j+1} \to s_j^+$.

S.4.5 Piecewise Quadratic Problem 3: Importance Sampling

Suppose we wish to importance sample in the same manner as piecewise linear, that is, we use inverse transform sampling. In essence, we draw $u \sim U(0, 1)$ and then sample $x = F^{-1}(u)$, with F the CDF of the distribution. Recall F is computed as

$$f(x) = \int_{s_0}^{x} T(s)\tau(s)ds = \int_{s_0}^{s_k} T(s)\tau(s)ds + \int_{s_k}^{x} T(s)\tau(s)ds = c_k + \int_{s_k}^{x} T(s)\tau(s)ds. \tag{19}$$

As before, the latter equation becomes

$$f(x) = c_k + T(s_k) - T(x). \tag{20}$$

Hence, computing x amounts to solving the equation $f(x) = u$, which becomes from above:

$$T(x) = c_k + T(s_k) - u. \tag{21}$$

As T is the exponential of a cubic, this amounts to solving a cubic above. This does have a real solution, as F is increasing and continuous, so the Intermediate Value Theorem implies $f(x) = u$ has a solution; and the solution is unique because F is strictly increasing, as it is an exponential of a polynomial. *However, the complexity of exact importance sampling would be large.*

This analysis reveals another downside of higher order polynomials. In general, suppose we wish to interpolate opacity τ with a piecewise degree n polynomial. Then following the same method as above, we see transmittance T is the piecewise exponential of a degree $n + 1$ polynomial, which is the integral of τ. So inverse transform sampling reduces to solving a degree $n + 1$ polynomial. The *Abel-Ruffini Theorem* asserts for $n + 1 \geq 5$ that this polynomial is in general not solvable by radicals. In other words, there is no simple closed form for exact importance sampling when $n \geq 4$.

Theoretically, this could be exactly solved for $n = 2$ and $n = 3$ (i.e. when opacity is piecewise quadratic or cubic, respectively), but the formulae to derive cubic and quartic solutions can become sufficiently complicated and the resulting complicated expressions may result in numerical instability during optimization, especially when taking the gradient w.r.t. the samples.

S.5 Experiment details, Reproducibility and Compute

S.5.1 Implementation Details

We include the core code snippets of our implementation of **PL-NeRF**. Figure S4 shows the volume rendering equation that includes the implementation of Eq. 10 and 11 (main paper). This is a direct replacement of the original constant approximation, where we also show the code snippet in the figure as reference. Figure S5 shows the implementation of our precise importance sampling from Eq. 13 (main paper), which is also a direct replacement of the constant importance sampling implementation (Figure S6) for reference. We highlight that our formulation is a direct replacement of the functions from the original implementation, and hence for the depth experiments, we are also able to directly adapt the codebase from (5). We use Nvidia v100 and A5000 GPU's for our experiments. Each scene is trained on a single GPU and takes $15 - 20$ hours. We used an internal academic cluster and cloud compute resources to train and evaluate our models.

For the MipNeRF-based experiments, our experiments are also run on the standard train and test split of the Blender dataset with the official released hyperparameters of Mip-NeRF using the NerfStudio (4) codebase. For PL-MipNeRF, we use the two-MLP training scheme with a coarse loss weight of 1.0.

For DIVeR-based experiments, we use their official implementation and configuration for DIVeR64 at 128 voxels. For PL-DIVeR, we utilize their voxel-based representation and feature integration and dropping in our piecewise linear opacity rendering formulation. We similarly run the DIVeR models on a single Nvidia v100 GPU trained using their default configurations and hyperparameters for DIVeR64 at 128 voxels.

The total training time for 500k iterations on a single Nvidia v100 GPU is 17.78 and 21.43 hours for Vanilla NeRF and PL-NeRF, respectively. Figure 2-c shows the head-to-head comparison of training PSNR (y-axis) with respect to time (x-axis) of Vanilla NeRF vs PL-NeRF on the Lego scene. Rendering a single 800x800 image takes 25.59 and 32.35 seconds for Vanilla NeRF and PL-NeRF, respectively.

S.5.2 Computational complexity under different number of samples

We measured the total rendering time for a single 800x800 image under different numbers of samples for our PL-NeRF. The total rendering time for (64+64), (64+128) and (128+64) are 19.20, 25.85 and 32.35 seconds, respectively.

S.5.3 Convergence plots under different number of samples

Figure **??** shows the convergence plots under different numbers of samples of our PL-NeRF vs Vanilla NeRF. We see that under different number of samples, our linear approach converges to a higher training PSNR.

S.6 Limitations

Our piecewise linear opacity approximation is able to handle arbitrarily small opacity, e.g. $1e^{-6}$, however, we cannot handle coordinates with exactly zero opacity. This special case is not an issue in practice. Also theoretically, any atom absorbs light and thus will not have exactly zero opacity, except in a vacuum. Another limitation is our method is slightly slower than the original piecewise constant approximation, and this is due to requiring more FLOPS for our importance sampling computation (Eq. 13 main paper). Another additional limitation is we still assume piecewise color, i.e. within a bin we do not handle color integration. Modeling this can potentially handle difficult scenarios such as double-walled colored glass or atmospheric effects such as fog or smoke. Lastly, we also inherit the limitations of NeRFs in general, such as requiring known camera poses.

S.7 Broader Impact

Our models require the usage of GPUs both in training time and rendering time, and GPUs use up energy to run and power them. We acknowledge that this contributes to climate change that is an important societal issue. Despite this, we observe the improvement in the results that is theoretically

```python
### Our Piecewise Linear Opacity Reformulation
def compute_weights_piecewise_linear(raw, z_vals, near, far, rays_d, noise=0., return_tau=False):
    raw2expr = lambda raw, dists: torch.exp(-raw*dists)

    dists = z_vals[...,1:] - z_vals[...,:-1]
    dists = dists * torch.norm(rays_d[...,None,:], dim=-1)

    ### tau(near) = 0, tau(far) = very big (will hit an opaque surface)
    tau = torch.cat([torch.ones((raw.shape[0], 1), device=device)*1e-10, raw[...,3] + noise, torch.ones((raw.shape[0], 1), device=device)*1e10], -1)

    tau = F.relu(tau) + 1e-6 ## Make positive

    interval_ave_tau = 0.5 * (tau[...,1:] + tau[...,:-1])

    '''
    Evaluating exp(-0.5 (tau_{i+1}+tau_i) (s_{i+1}-s_i) )
    '''
    expr = raw2expr(interval_ave_tau, dists)  # [N_rays, N_samples+1]

    ### Transmittance until s_n
    T = torch.cumprod(torch.cat([torch.ones((expr.shape[0], 1), device=device), expr], -1), -1) # [N_rays, N_samples+2], T(near)=1, starts off at 1

    ### Factor to multiply transmittance with
    factor = (1 - expr)
    weights = factor * T[:, :-1] # [N_rays, N_samples+1]

    else:
        return weights

### Original Piecewise Constant Opacity
def compute_weights(raw, z_vals, rays_d, noise=0.):
    raw2alpha = lambda raw, dists, act_fn=F.relu: 1.-torch.exp(-act_fn(raw)*dists)

    dists = z_vals[...,1:] - z_vals[...,:-1]
    dists = torch.cat([dists, torch.full_like(dists[...,:1], 1e10, device=device)], -1)  # [N_rays, N_samples]
    dists = dists * torch.norm(rays_d[...,None,:], dim=-1)

    alpha = raw2alpha(raw[...,3] + noise, dists)  # [N_rays, N_samples]
    weights = alpha * torch.cumprod(torch.cat([torch.ones((alpha.shape[0], 1), device=device), 1.-alpha + 1e-10], -1), -1)[:, :-1]
    return weights
```

Figure S4: **Code snippet for volume rendering.** The implementation for our piecewise linear opacity approximation is a drop-in replacement from the original piecewise constant.

grounded and believe that it is beneficial for the pursuit of science. We responsibly ran our models by first prototyping on selected scenes before scaling up to different scenes across datasets to minimize its impact to climate change.

References

[1] Henrik Aanæs, Rasmus Ramsbøl Jensen, George Vogiatzis, Engin Tola, and Anders Bjorholm Dahl. Large-scale data for multiple-view stereopsis. *International Journal of Computer Vision*, pages 1–16, 2016.

[2] Nelson Max and Min Chen. Local and global illumination in the volume rendering integral. In Hans Hagen, editor, *Scientific Visualization: Advanced Concepts*, volume 1 of *Dagstuhl Follow-Ups*, pages 259–274, Dagstuhl, Germany, 2010. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[3] 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.

[4] Matthew Tancik, Ethan Weber, Evonne Ng, Ruilong Li, Brent Yi, Justin Kerr, Terrance Wang, Alexander Kristoffersen, Jake Austin, Kamyar Salahi, Abhik Ahuja, David McAllister, and Angjoo Kanazawa. Nerfstudio: A modular framework for neural radiance field development. In *ACM SIGGRAPH 2023 Conference Proceedings*, 2023.

[5] Mikaela Angelina Uy, Ricardo Martin-Brualla, Leonidas Guibas, and Ke Li. Scade: Nerfs from space carving with ambiguity-aware depth estimates. In *CVPR*, 2023.

[6] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. pixelnerf: Neural radiance fields from one or few images. In *CVPR*, pages 4578–4587, 2021.

```python
### Our Piecewise Linear Opacity Reformulation
def sample_pdf_reformulation(bins, weights, tau, T, near, far, N_samples, det=False, pytest=False, \
    zero_threshold = 1e-4, epsilon_=1e-3):
    ### bins = z_vals, ie bin boundaries, input does not include near and far plane yet
    ### weights is the PMF of each bin ## N_samples + 1

    bins = torch.cat([near, bins, far], -1)

    pdf = weights

    cdf = torch.cumsum(pdf, -1)
    cdf = torch.cat([torch.zeros_like(cdf[...,:1]), cdf], -1)  # (batch, len(bins))

    ### CDF sums up to 1.0
    cdf[:,-1] = 1.0

    # Take uniform samples
    if det:
        u = torch.linspace(0., 1., steps=N_samples, device=bins.device)
        u = u.expand(list(cdf.shape[:-1]) + [N_samples])
    else:
        u = torch.rand(list(cdf.shape[:-1]) + [N_samples], device=bins.device)

    # Pytest, overwrite u with numpy's fixed random numbers
    if pytest:
        np.random.seed(0)
        new_shape = list(cdf.shape[:-1]) + [N_samples]
        if det:
            u = np.linspace(0., 1., N_samples)
            u = np.broadcast_to(u, new_shape)
        else:
            u = np.random.rand(*new_shape)
        u = torch.Tensor(u)

    # Invert CDF
    u = u.contiguous()
    inds = torch.searchsorted(cdf, u, right=True)

    below = torch.max(torch.zeros_like(inds-1), inds-1)
    above = torch.min((cdf.shape[-1]-1) * torch.ones_like(inds), inds)
    inds_g = torch.stack([below, above], -1)  # (batch, N_samples, 2)


    matched_shape = [inds_g.shape[0], inds_g.shape[1], cdf.shape[-1]]
    cdf_g = torch.gather(cdf.unsqueeze(1).expand(matched_shape), 2, inds_g)
    bins_g = torch.gather(bins.unsqueeze(1).expand(matched_shape), 2, inds_g)
    T_g = torch.gather(T.unsqueeze(1).expand(matched_shape), 2, inds_g)
    tau_g = torch.gather(tau.unsqueeze(1).expand(matched_shape), 2, inds_g)

    ### Get tau diffs
    tau_diff = tau[...,1:] - tau[...,:-1]
    matched_shape_tau = [inds_g.shape[0], inds_g.shape[1], tau_diff.shape[-1]]

    tau_diff_g = torch.gather(tau_diff.unsqueeze(1).expand(matched_shape_tau), 2, below.unsqueeze(-1)).squeeze()

    s_left = bins_g[...,0]
    s_right = bins_g[...,1]
    T_left = T_g[...,0]
    tau_left = tau_g[...,0]
    tau_right = tau_g[...,1]

    dummy = torch.ones(s_left.shape, device=s_left.device)*-1.0
    samples1 = torch.where(torch.logical_and(tau_diff_g < zero_threshold, tau_diff_g > -zero_threshold), s_left, dummy)

    ### Our Precision Importance Sampling
    samples2 = torch.where(tau_diff_g >= zero_threshold, \
        pw_linear_sample(s_left, s_right, T_left, tau_left, tau_right, u, epsilon=epsilon_), samples1)

    samples = torch.where(torch.isnan(samples3), s_left, samples3)

    tau_g = torch.gather(tau.unsqueeze(1).expand(matched_shape), 2, inds_g)
    T_g = torch.gather(T.unsqueeze(1).expand(matched_shape), 2, inds_g)

    T_below = T_g[...,0]
    tau_below = tau_g[...,0]
    bin_below = bins_g[...,0]
    ################################

    return samples, T_below, tau_below, bin_below


def pw_linear_sample(s_left, s_right, T_left, tau_left, tau_right, u, epsilon=1e-3):
    ln_term = -torch.log(torch.max(torch.ones_like(T_left)*epsilon, \
        torch.div(1-u, torch.max(torch.ones_like(T_left)*epsilon,T_left) ) ))
    discriminant = tau_left**2 + torch.div( 2 * (tau_right - tau_left) * ln_term , \
        torch.max(torch.ones_like(s_right)*epsilon, s_right - s_left) )

    t = torch.div( (s_right - s_left) * (-tau_left + torch.sqrt(\
        torch.max(torch.ones_like(discriminant)*epsilon, discriminant))) , \
    torch.max(torch.ones_like(tau_left)*epsilon, tau_right - tau_left))

    ### clamp t to [0, s_right - s_left]
    t = torch.clamp(t, torch.ones_like(t, device=t.device)*epsilon, s_right - s_left)

    sample = s_left + t

    return sample
```

Figure S5: **Code snippet for our Precise Importance Sampling.** The implementation of our precision importance sampling is also a direct replacement from the original function from the constant implementation called sample_pdf (See next figure for reference).

```python
### Original Piecewise Constant Opacity
def sample_pdf(bins, weights, N_samples, det=False, pytest=False):
    # Get pdf
    weights = weights + 1e-5 # prevent nans
    pdf = weights / torch.sum(weights, -1, keepdim=True)

    cdf = torch.cumsum(pdf, -1)
    cdf = torch.cat([torch.zeros_like(cdf[...,:1]), cdf], -1)  # (batch, len(bins))

    # Take uniform samples
    if det:
        u = torch.linspace(0., 1., steps=N_samples, device=bins.device)
        u = u.expand(list(cdf.shape[:-1]) + [N_samples])
    else:
        u = torch.rand(list(cdf.shape[:-1]) + [N_samples], device=bins.device)

    # Pytest, overwrite u with numpy's fixed random numbers
    if pytest:
        np.random.seed(0)
        new_shape = list(cdf.shape[:-1]) + [N_samples]
        if det:
            u = np.linspace(0., 1., N_samples)
            u = np.broadcast_to(u, new_shape)
        else:
            u = np.random.rand(*new_shape)
        u = torch.Tensor(u)

    # Invert CDF
    u = u.contiguous()

    inds = torch.searchsorted(cdf, u, right=True)

    below = torch.max(torch.zeros_like(inds-1), inds-1)
    above = torch.min((cdf.shape[-1]-1) * torch.ones_like(inds), inds)
    inds_g = torch.stack([below, above], -1)  # (batch, N_samples, 2)

    # cdf_g = tf.gather(cdf, inds_g, axis=-1, batch_dims=len(inds_g.shape)-2)
    # bins_g = tf.gather(bins, inds_g, axis=-1, batch_dims=len(inds_g.shape)-2)
    matched_shape = [inds_g.shape[0], inds_g.shape[1], cdf.shape[-1]]
    cdf_g = torch.gather(cdf.unsqueeze(1).expand(matched_shape), 2, inds_g)
    bins_g = torch.gather(bins.unsqueeze(1).expand(matched_shape), 2, inds_g)

    denom = (cdf_g[...,1]-cdf_g[...,0])
    denom = torch.where(denom<1e-5, torch.ones_like(denom), denom)
    t = (u-cdf_g[...,0])/denom
    samples = bins_g[...,0] + t * (bins_g[...,1]-bins_g[...,0])

    return samples
```

Figure S6: This is the original importance sampling for the constant approximation for reference.