NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis

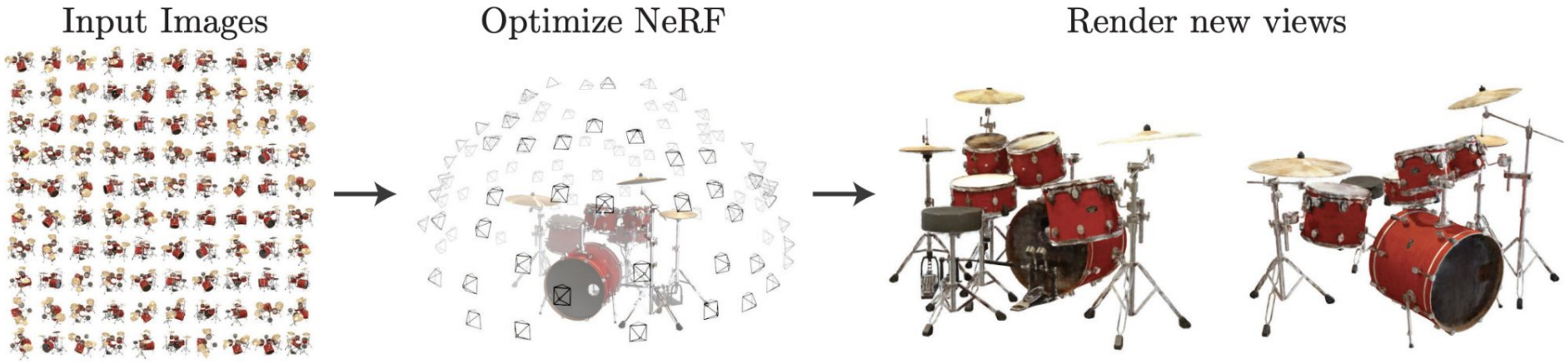
Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, Ren Ng

Подготовили: Гольдман Артур, Пак Ди Ун, Данг Нина

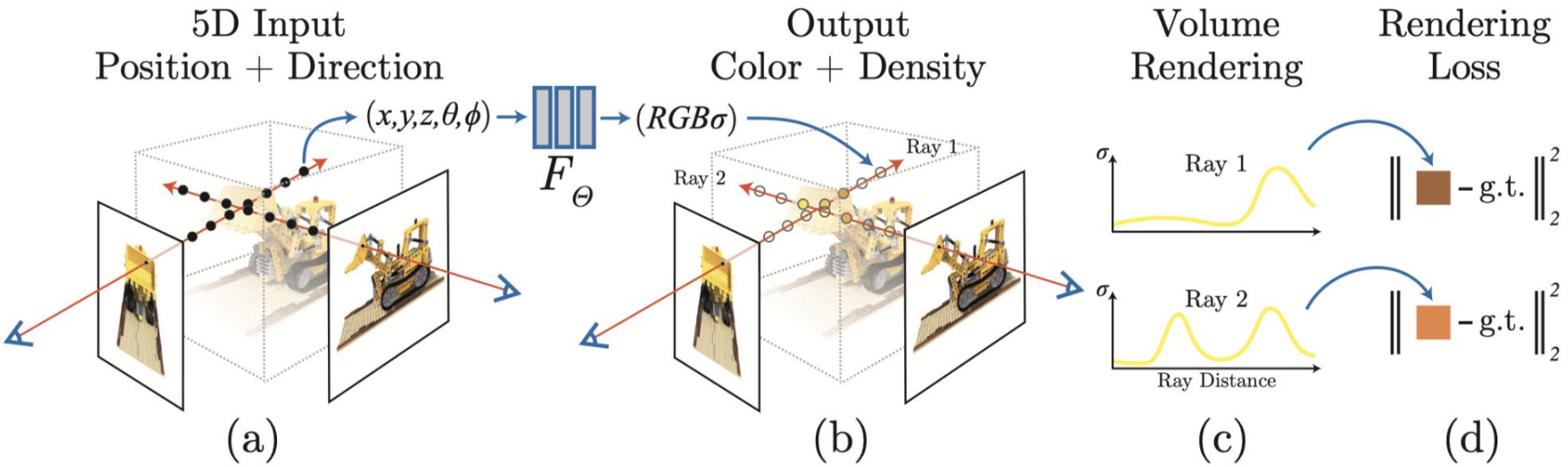
Докладчик

Гольдман Артур

Task formulation

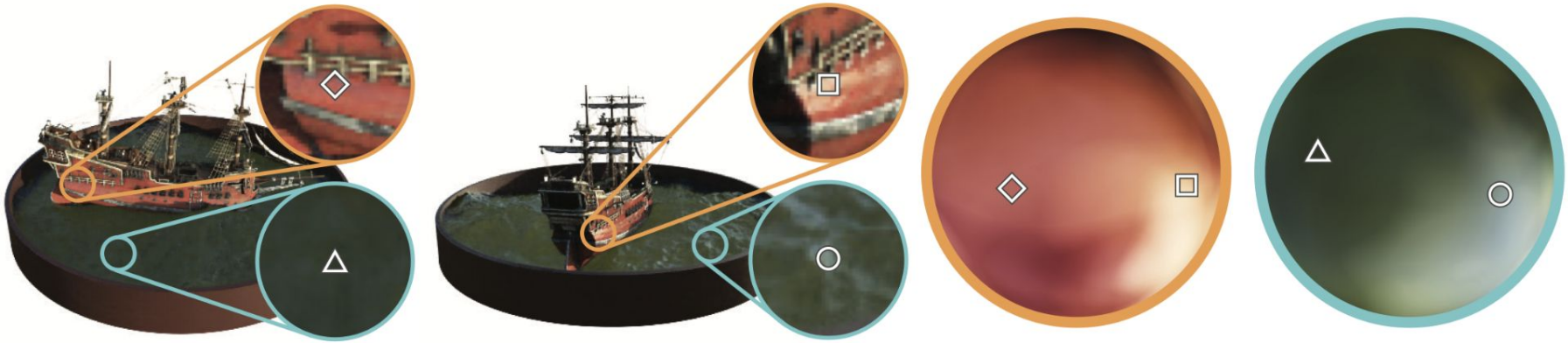


Neural Radiance Field (NeRF) Scene Representation



Заменим вектор (θ, ϕ) на соответствующий ему единичный вектор d.

A visualization of view-dependent emitted radiance



(a) View 1 (b) View 2 (c) Radiance Distributions

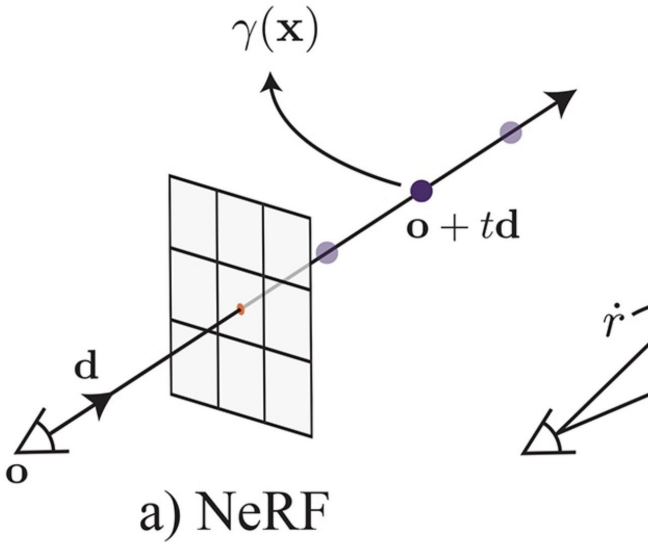
Expected color

o - координата начала луча

d - вектор направления луча

r(t) = o+td

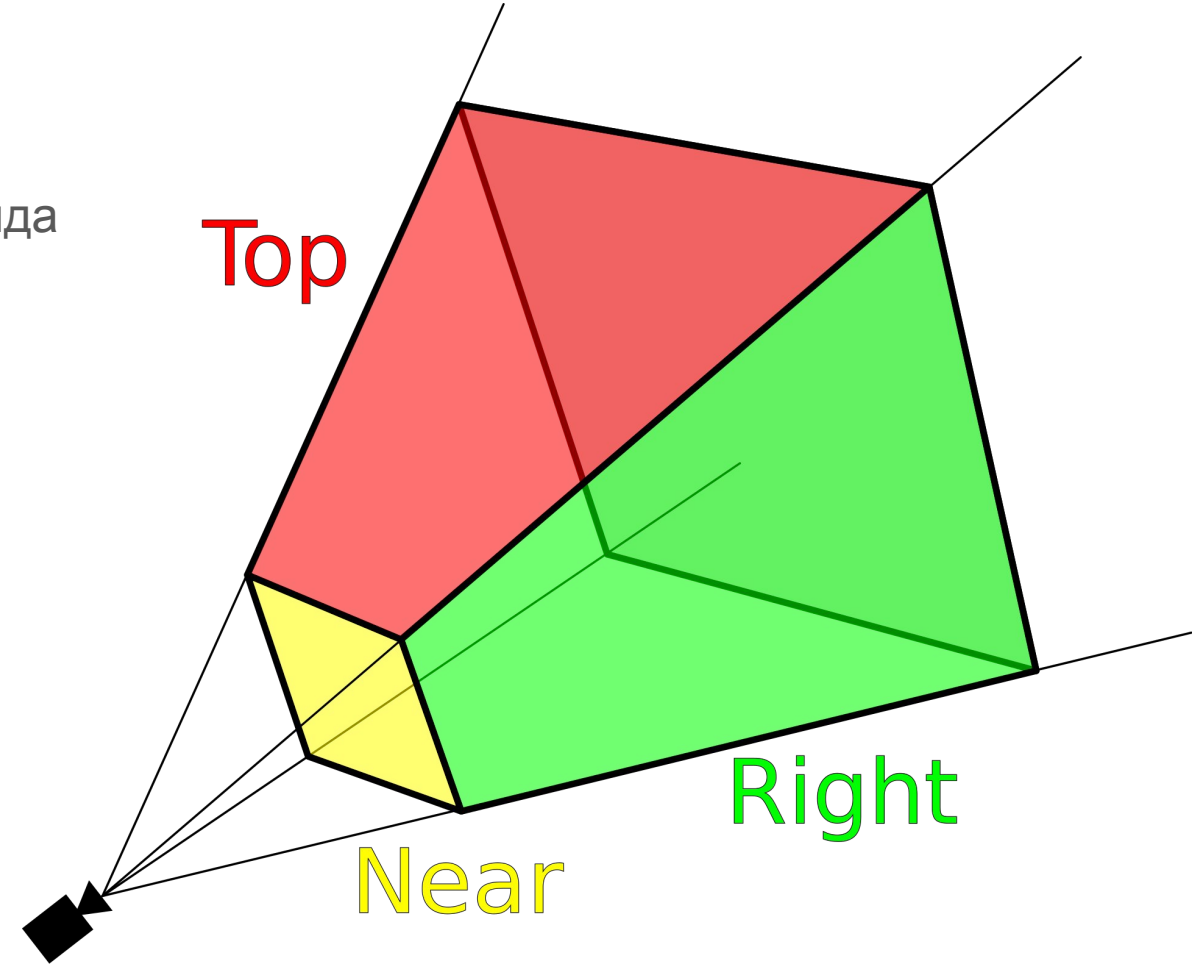
t_n, t_f - ближняя и дальняя граница



$$C(\mathbf{r}) = \int_{t_n}^{t_f} T(t)\sigma(\mathbf{r}(t))\mathbf{c}(\mathbf{r}(t),\mathbf{d})dt\,,\ \text{where}\ T(t) = \exp\left(-\int_{t_n}^{t}\sigma(\mathbf{r}(s))ds\right)$$

View frustum

Frustum - усеченная пирамида



Quadrature approximation

$$\delta_i = t_{i+1} - t_i$$

$$t_i \sim \mathcal{U}\left[t_n + \frac{i-1}{N}(t_f - t_n),\ t_n + \frac{i}{N}(t_f - t_n)\right]$$

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} T_i(1 - \exp(-\sigma_i \delta_i))\mathbf{c}_i\,,\ \text{where } T_i = \exp\left(-\sum_{j=1}^{i-1} \sigma_j \delta_j\right)$$

$$C(\mathbf{r}) = \int_{t_n}^{t_f} T(t)\sigma(\mathbf{r}(t))\mathbf{c}(\mathbf{r}(t), \mathbf{d})dt\,,\ \text{where } T(t) = \exp\left(-\int_{t_n}^{t} \sigma(\mathbf{r}(s))ds\right)$$

Understanding the density

T(t) - вероятность того, что луч пройдет от t_n до t не врезаясь в частицы на своем пути.

1-T(t) - вероятность врезаться в частицу t. Какая плотность?

$$(1 - T(t))'_t = (-\exp\left(-\int_{t_n}^{t} \sigma(r(s))dx\right))'_t =$$

$$= \sigma(r(t)) \exp\left(-\int_{t_n}^{t} \sigma(r(s))dx\right) = \sigma(r(t))T(t)$$

$$C(\mathbf{r}) = \int_{t_n}^{t_f} T(t)\sigma(\mathbf{r}(t))\mathbf{c}(\mathbf{r}(t), \mathbf{d})dt\,, \text{ where } T(t) = \exp\left(-\int_{t_n}^{t} \sigma(\mathbf{r}(s))ds\right)$$

Understanding the probability

T_i - вероятность того, что луч пройдет до t_i не врезаясь в частицы на своем пути.

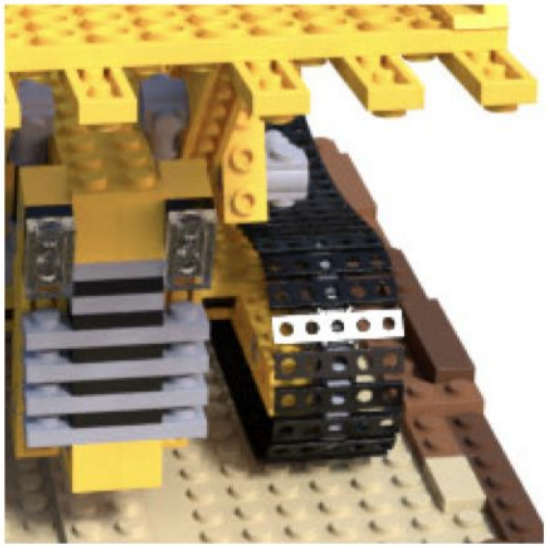
Какова вероятность того, что луч дойдет до точки t_i и в ней остановится? Пусть α_i обозначает вероятность остановиться в точке t_i.

$$1 - \alpha_i = \exp(-\sigma_i \delta_i)$$

$$\prod_{i=1}^{j-1}(1-\alpha_i)\alpha_j = \exp(-\sum_{i=1}^{j-1}\sigma_i\delta_i)(1-\exp(-\sigma_i\delta_i)) = T_i(1-\exp(-\sigma_i\delta_i))$$

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} T_i(1-\exp(-\sigma_i\delta_i))\mathbf{c}_i\,, \text{ where } T_i = \exp\left(-\sum_{j=1}^{i-1}\sigma_j\delta_j\right)$$

Improving quality



Ground Truth

Improving quality: positional encoding

$$\gamma(p) = \left(\sin\left(2^0\pi p\right), \cos\left(2^0\pi p\right), \cdots, \sin\left(2^{L-1}\pi p\right), \cos\left(2^{L-1}\pi p\right)\right)$$

Применяем функцию гамма покоординатно к x, d, которые заранее нормализованы и их координаты лежат в [-1;1]. (Заметим, что координаты d уже лежат в [-1;1]).

В статье брали L=10 для x и L=4 для d.

Improving efficiency: Hierarchical volume sampling

Две сети: coarse and fine.

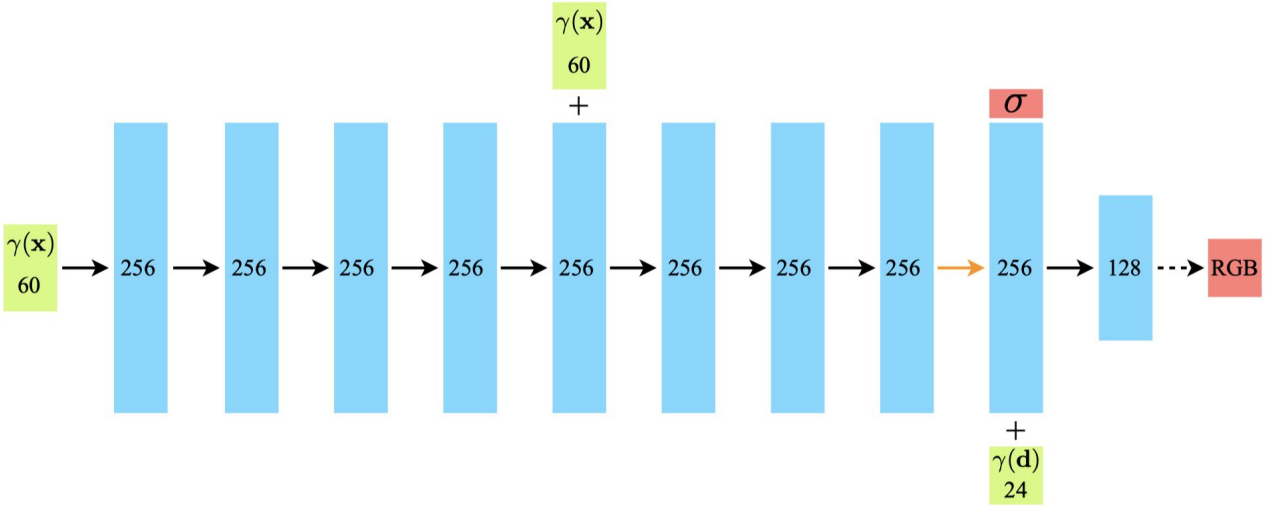
Первая выборка: N_c точек - считаем значения, используя coarse сеть.

$$\hat{C}_c(\mathbf{r}) = \sum_{i=1}^{N_c} w_i c_i \,, \qquad w_i = T_i(1 - \exp(-\sigma_i \delta_i))$$

Вторая выборка: N_f точек из отрезков для N_c, каждый из отрезков выбирается согласно нормированному распределению на w_i.

Итоговую оценку C считаем, используя точки из обеих выборок и fine сеть

Network



All layers are standard fully-connected layers, black arrows indicate layers with ReLU activations, orange arrows indicate layers with no activation, dashed black arrows indicate layers with sigmoid activation, and “+” denotes vector concatenation.

An additional layer outputs the volume density σ (which is rectified using a ReLU to ensure that the output volume density is nonnegative) and a 256-dimensional feature vector. This feature vector is concatenated with the positional encoding of the input viewing direction (γ(d)), and is processed by an additional fully-connected ReLU layer with 128 channels.

Optimization

$$\mathcal{L} = \sum_{\mathbf{r} \in \mathcal{R}} \left[\left\| \hat{C}_c(\mathbf{r}) - C(\mathbf{r}) \right\|_2^2 + \left\| \hat{C}_f(\mathbf{r}) - C(\mathbf{r}) \right\|_2^2 \right]$$

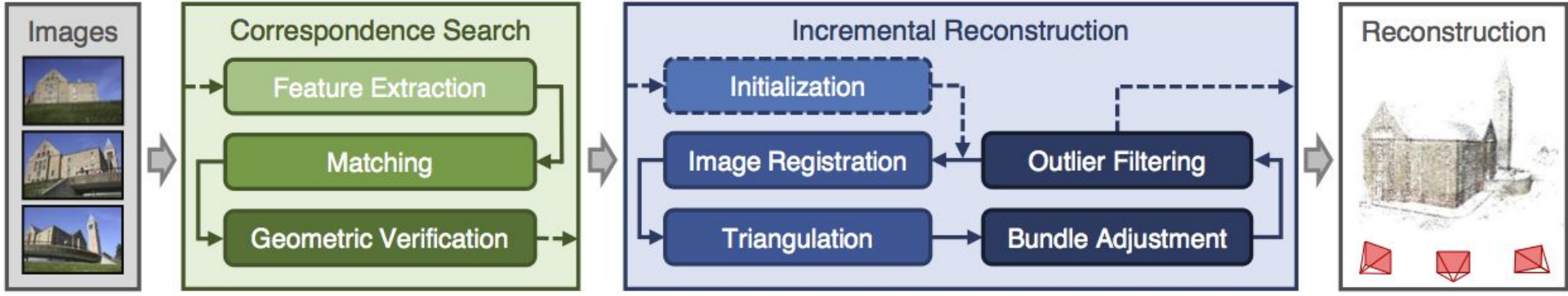
Batch size = 4096 rays

N_c = 64, N_f = 128

The optimization for a single scene typically take around 100-300k iterations to converge on a single NVIDIA V100 GPU (about 1–2 days)

COLMAP

Sparse model of central Rome using 21K photos produced by COLMAP's SfM pipeline.



Results

DeepVoxels:
Learning Persistent 3D Feature Embeddings









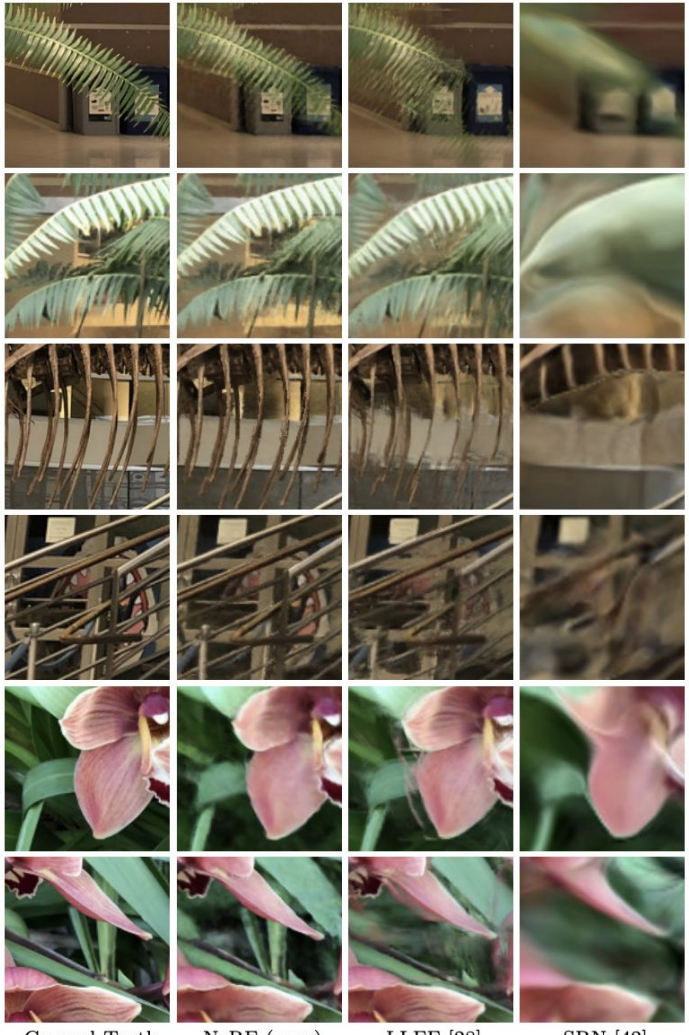

Results

Method	Diffuse Synthetic 360° [41]			Realistic Synthetic 360°			Real Forward-Facing [28]		
	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
SRN [42]	33.20	0.963	0.073	22.26	0.846	0.170	22.84	0.668	0.378
NV [24]	29.62	0.929	0.099	26.05	0.893	0.160	-	-	-
LLFF [28]	34.38	0.985	0.048	24.88	0.911	0.114	24.13	0.798	**0.212**
Ours	**40.15**	**0.991**	**0.023**	**31.01**	**0.947**	**0.081**	**26.50**	**0.811**	0.250

Fern

T-Rex

Orchid



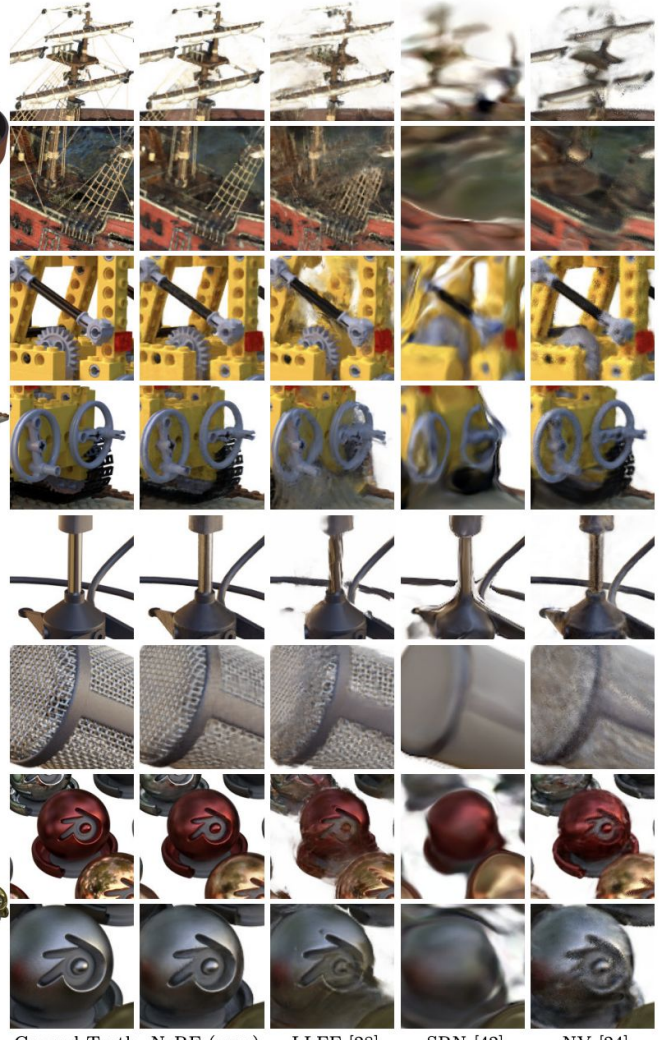
Ground Truth NeRF (ours) LLFF [28] SRN [42]

Ship

Lego

Microphone

Materials



Ground Truth NeRF (ours) LLFF [28] SRN [42] NV [24]

Ablation study

	Input	#Im.	L	(N_c, N_f)	PSNR↑	SSIM↑	LPIPS↓
1) No PE, VD, H	xyz	100	-	(256, -)	26.67	0.906	0.136
2) No Pos. Encoding	$xyz\theta\phi$	100	-	(64, 128)	28.77	0.924	0.108
3) No View Dependence	xyz	100	10	(64, 128)	27.66	0.925	0.117
4) No Hierarchical	$xyz\theta\phi$	100	10	(256, -)	30.06	0.938	0.109
5) Far Fewer Images	$xyz\theta\phi$	25	10	(64, 128)	27.78	0.925	0.107
6) Fewer Images	$xyz\theta\phi$	50	10	(64, 128)	29.79	0.940	0.096
7) Fewer Frequencies	$xyz\theta\phi$	100	5	(64, 128)	30.59	0.944	0.088
8) More Frequencies	$xyz\theta\phi$	100	15	(64, 128)	30.81	0.946	0.096
9) Complete Model	$xyz\theta\phi$	100	10	(64, 128)	**31.01**	**0.947**	**0.081**

Volume Bounds

Our dataset of real images contains content that can exist anywhere between the closest point and infinity, so we use **normalized device coordinates** to map the depth range of these points into [−1, 1]. This shifts all the ray origins to the near plane of the scene, maps the perspective rays of the camera to parallel rays in the transformed volume, and uses disparity (inverse depth) instead of metric depth, so all coordinates are now bounded.

Once we convert to the NDC ray, this allows us to simply sample t' linearly from 0 to 1 in order to get a linear sampling in disparity from n to ∞ in the original space.

Rendering Details

To render new views at test time, we sample 64 points per ray through the coarse network and 64 + 128 = 192 points per ray through the fine network, for a total of 256 network queries per ray. Our realistic synthetic dataset requires 640k rays per image, and our real scenes require 762k rays per image, resulting in between 150 and 200 million network queries per rendered image. On an NVIDIA V100, this takes approximately 30 seconds per frame.

Рецензент

Пак Ди Ун

Содержание и вклад

Статья содержит в себе описание для задачи восстановления плотностных трехмерных сцен на основе изображений с разных ракурсов. Основными концепциями в данной работе являются позиционное кодирование, view-dependant представление данных, а также иерархическое семплирование. С помощью этих приемов предложенная модель достигает state-of-the-art результатов на своей задаче.

Сильные стороны

1. Все теоретические детали объяснены. В аппендиксе дополнительно показан вывод формулы проекции лучей
2. В аппендиксе указаны все реализации бейзлайнов с точностью до ссылки на репозиторий
3. Работа показывает высокие результаты в своей задаче, обходя аналоги по качеству
4. Вслед за публикацией статьи NeRF последовала волна различных модификаций, что показывает высокий вклад работы в сообщество, а также применимость идеи на другие подзадачи

Сильные стороны: модификации на ICCV2021

1. Mip-NeRF, адаптация MIP-текстурирование для семплирования в конусе
2. MVSNeRF, адаптация, работающая сразу на нескольких сценах
3. DietNeRF, семантическая сегментация
4. UNISURF, замена плотностного метода сеткой занятости
5. NerfingMVS
6. FastNeRF, 3000x скорость с помощью факторизации рендера объема
7. KiloNeRF, ускорение на 3 порядка, благодаря подмене MLP тысячью маленьких

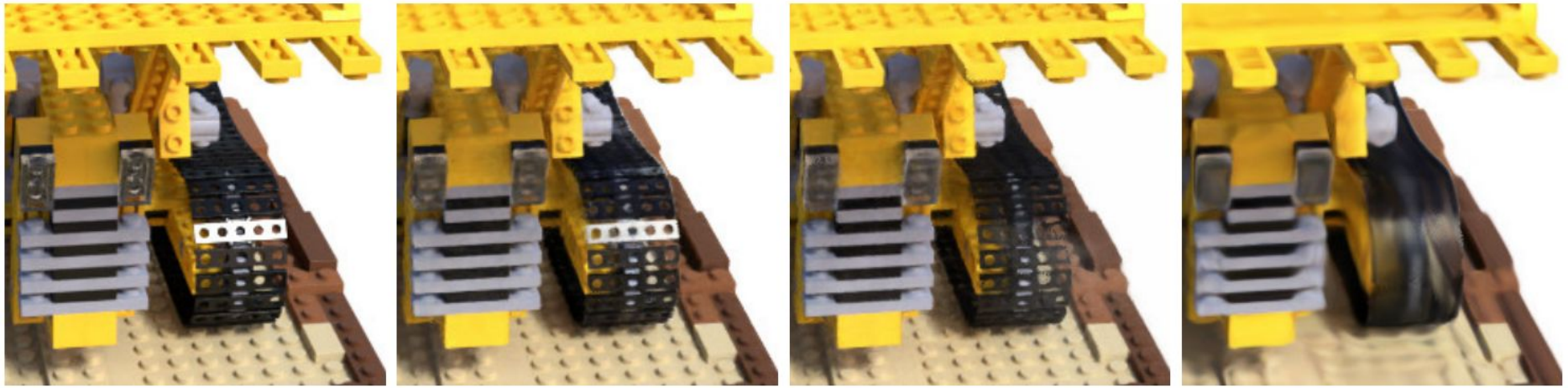
И еще 25 работ, включая ГАНы, динамику, изменение сцены и рендер без позиции

Оценка

1. К качеству статьи нет замечаний. Ablation study, описание экспериментов, выбор методов и их математическое обоснование подробно и доступно описано
2. Воспроизводимость: присутствует репозиторий с кодом на TensorFlow с конфигами из статьи.
3. Оценка NeurIPS: 9/10
4. Уверенность: 4.5/5

Хакер

Данг Нина



Ground Truth | Complete Model | No View Dependence | No Positional Encoding

Tiny NeRF

- Без view dependence (направления луча)
- Без hierarchical volume sampling
- Простой 3х-слойный MLP

Positional Encoding

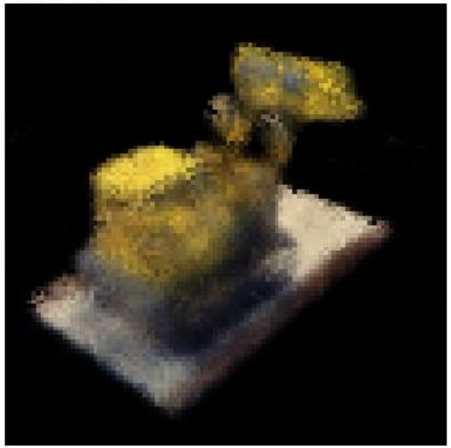
ground truth

tiny-nerf-6-pe

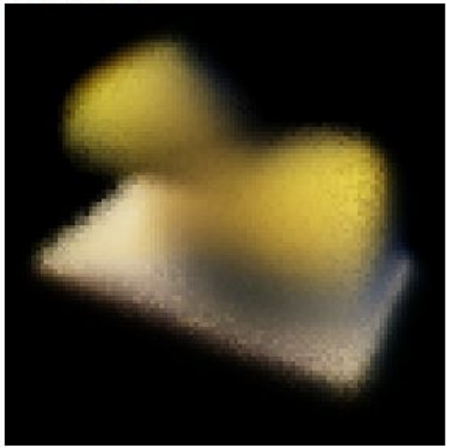
tiny-nerf-no-pe

predicted

tiny-nerf-6-pe



tiny-nerf-no-pe



Positional Encoding

$$\gamma(p) = \left(\sin\left(2^0\pi p\right), \cos\left(2^0\pi p\right), \cdots, \sin\left(2^{L-1}\pi p\right), \cos\left(2^{L-1}\pi p\right)\right)$$

	Input	#Im.	L	(N_c, N_f)	PSNR↑	SSIM↑	LPIPS↓
Fewer Frequencies	$xyz\theta\phi$	100	5	(64, 128)	30.59	0.944	0.088
More Frequencies	$xyz\theta\phi$	100	15	(64, 128)	30.81	0.946	0.096
Complete Model	$xyz\theta\phi$	100	10	(64, 128)	**31.01**	**0.947**	**0.081**

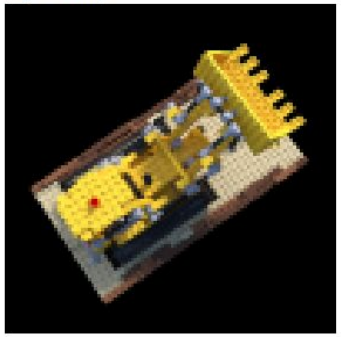
Positional Encoding

ground truth

tiny-nerf-3-pe

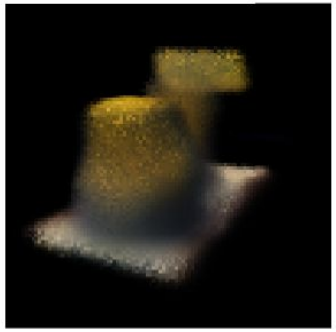
tiny-nerf-6-pe

tiny-nerf-10-pe

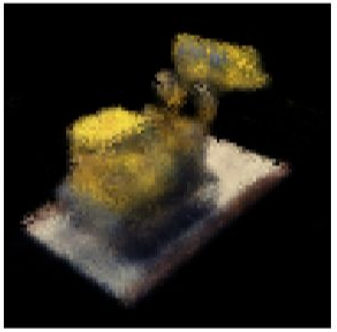


predicted

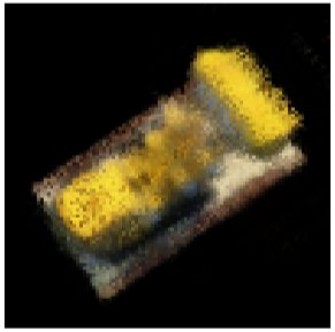
tiny-nerf-3-pe

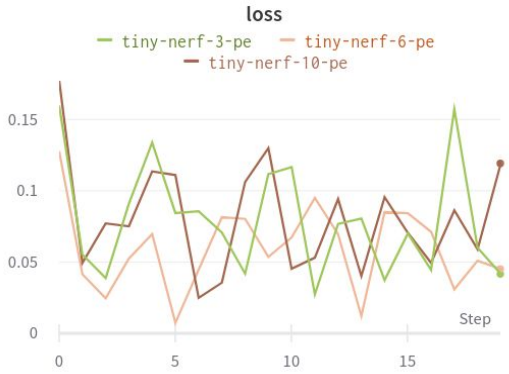


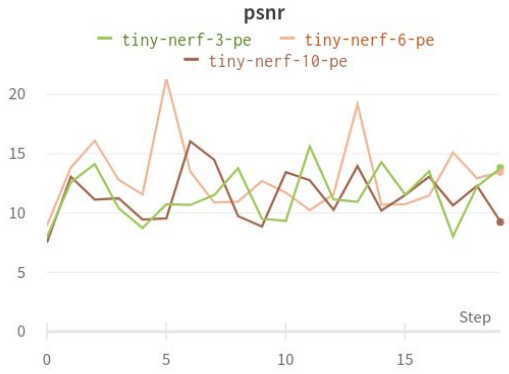
tiny-nerf-6-pe



tiny-nerf-10-pe







Fewer Images

ground truth

tiny-nerf-25img-6-pe

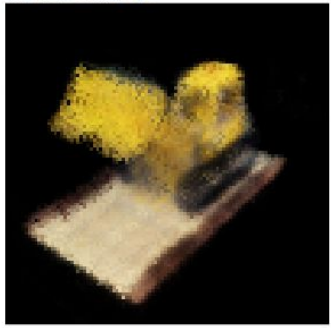
tiny-nerf-50img-6-pe

tiny-nerf-6-pe

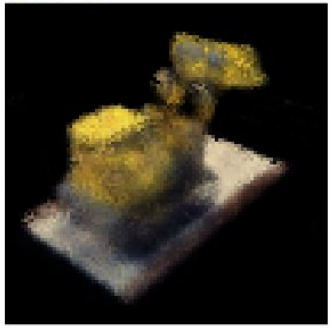
predicted

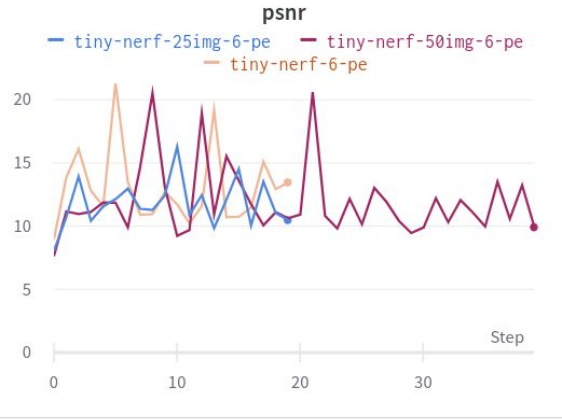
tiny-nerf-25img-6-pe

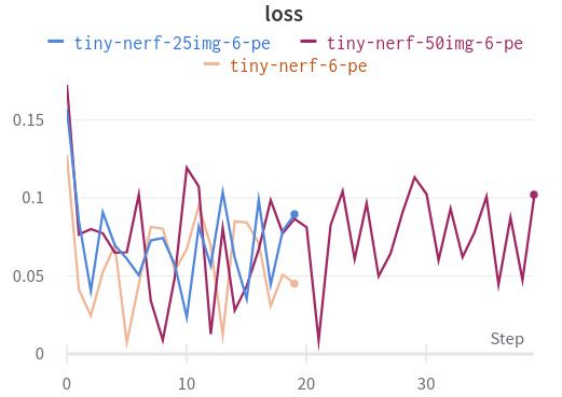
tiny-nerf-50img-6-pe



tiny-nerf-6-pe







Full NeRF

- С view dependence (вектором направления луча)
- С hierarchical volume sampling (две модели: fine и coarse)
- Архитектура модели остаётся такой же

Full NeRF

Источники

- https://github.com/bmild/nerf
- https://github.com/krrish94/nerf-pytorch