

# NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis

Кузнецов Дмитрий  
БПМИ171

# Мотивация



# Мотивация



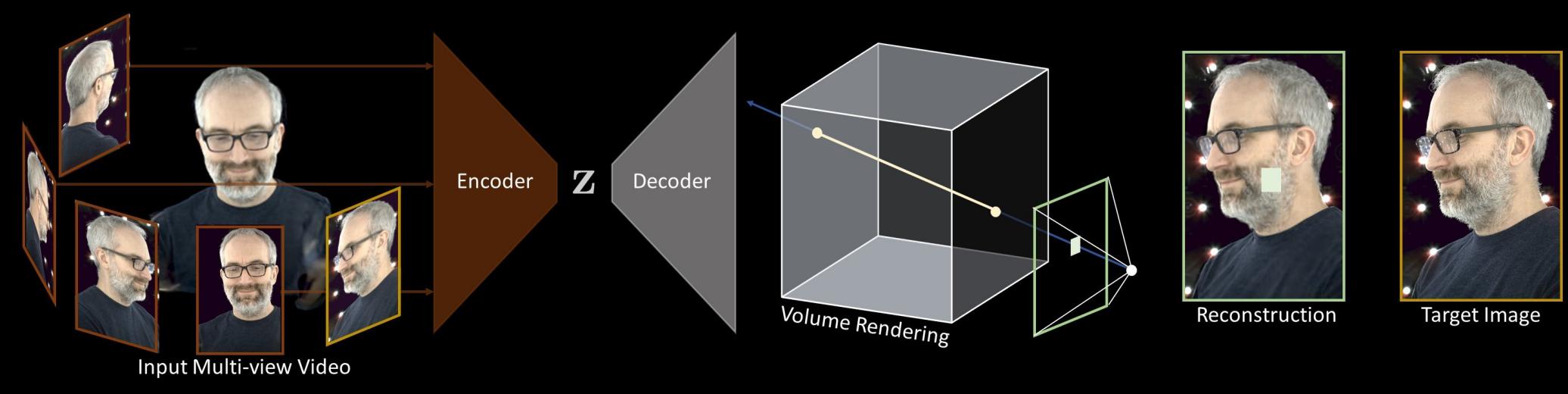
# Мотивация



# Мотивация



# До NeRF: RGBa volume rendering



**Neural Volumes**  
Lombardi et al. 2019

# До NeRF: RGBa volume rendering



**Neural Volumes**  
Lombardi et al. 2019

# До NeRF: Проблемы

- Низкая размерность полученных результатов
- Артефакты на сценах

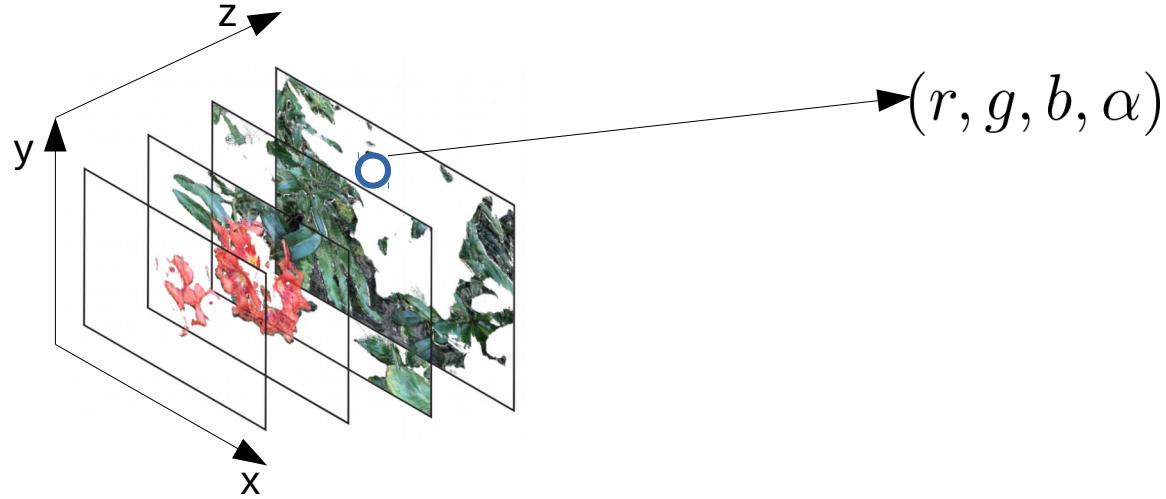
# До NeRF: Проблемы

- Низкая размерность полученных результатов
- Артефакты на сценах

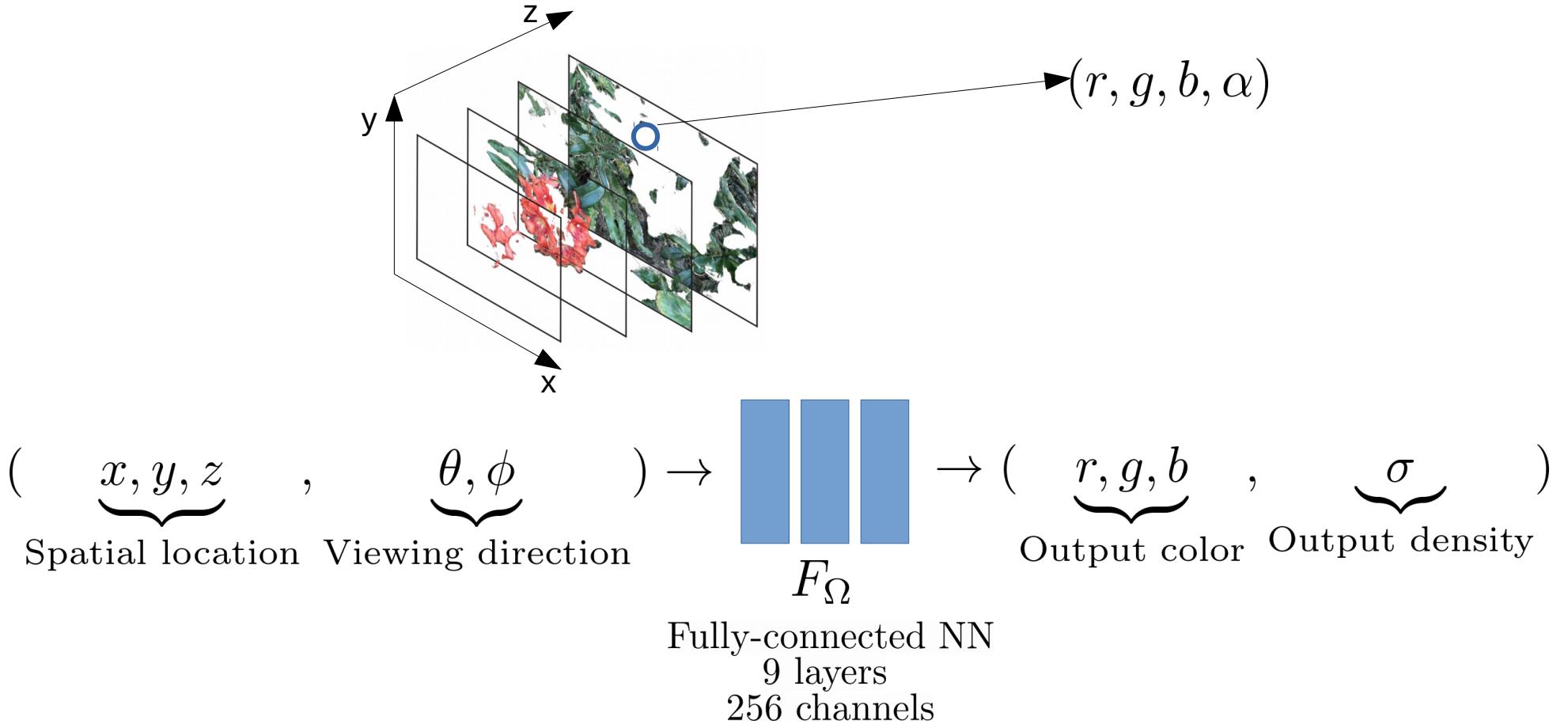
## **Проблемы:**

- При такой постановке требуется высокая сложность моделей и огромные объемы памяти для достижения высокого разрешения
- В других существующих решениях, наоборот, сложно дифференцировать в угоду памяти.

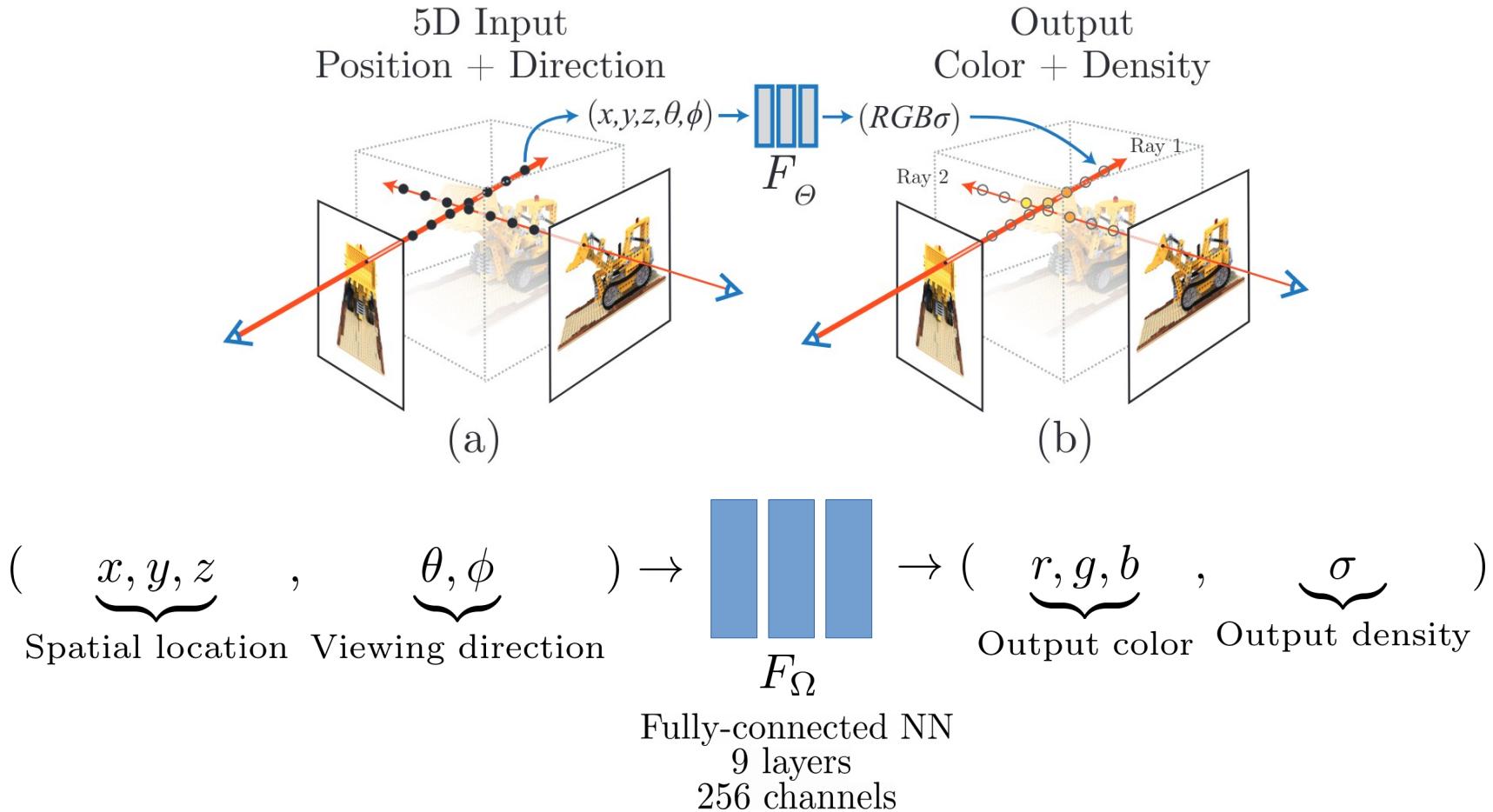
# Решение: NeRF



# Решение: NeRF

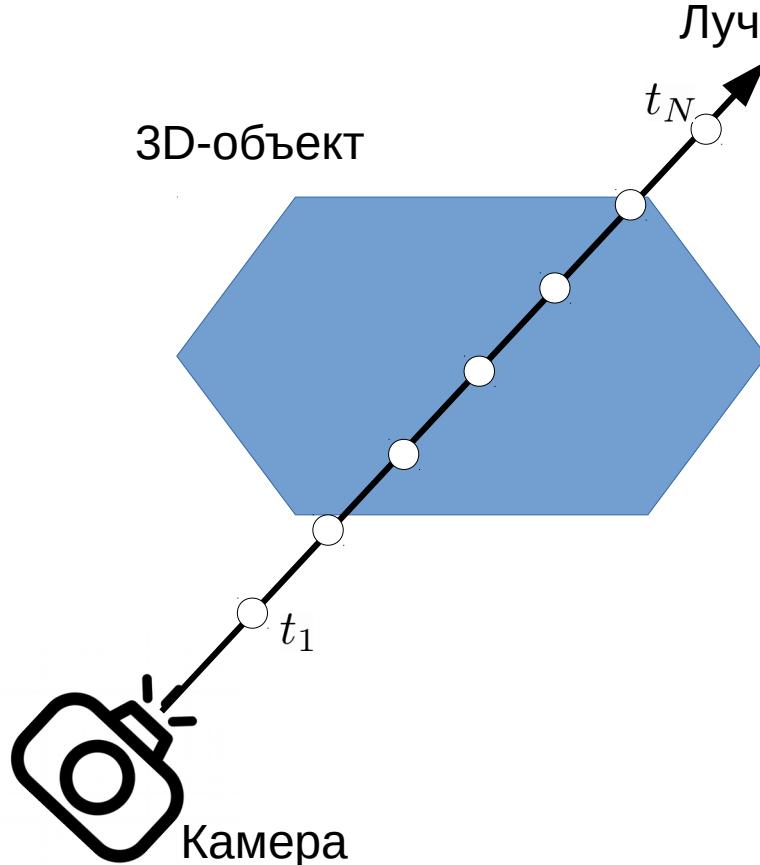


# Решение: NeRF



# NeRF

Луч:  $r(t) = x_0 + t\mathbf{d}$

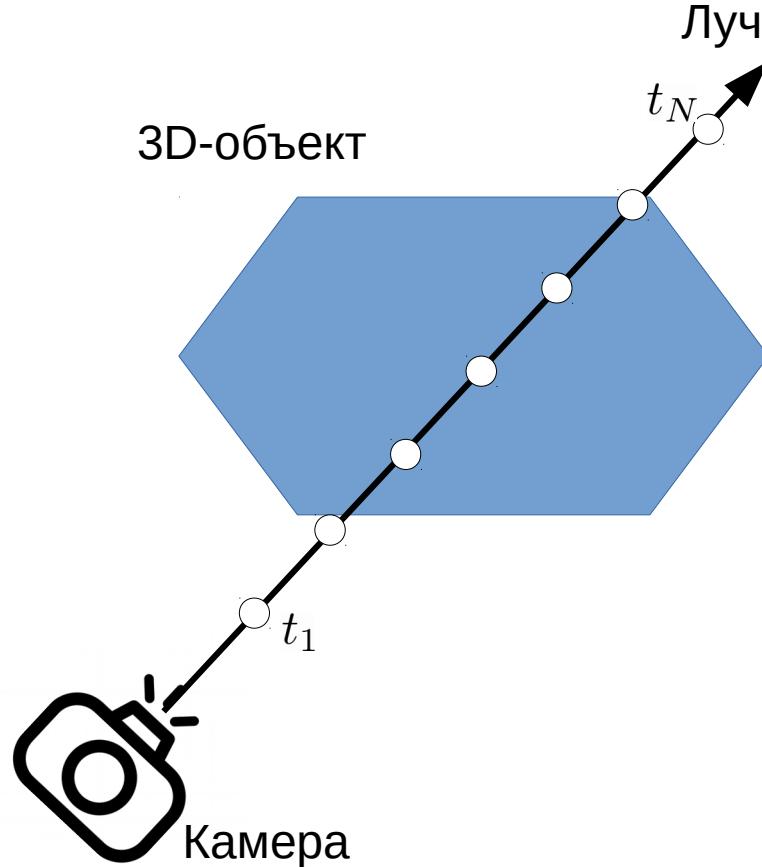


# NeRF

Луч:  $r(t) = x_0 + t\mathbf{d}$

$$Color(r) \simeq \sum_{i=1}^N T_i \alpha_i c_i$$

Вес      Цвет



# NeRF

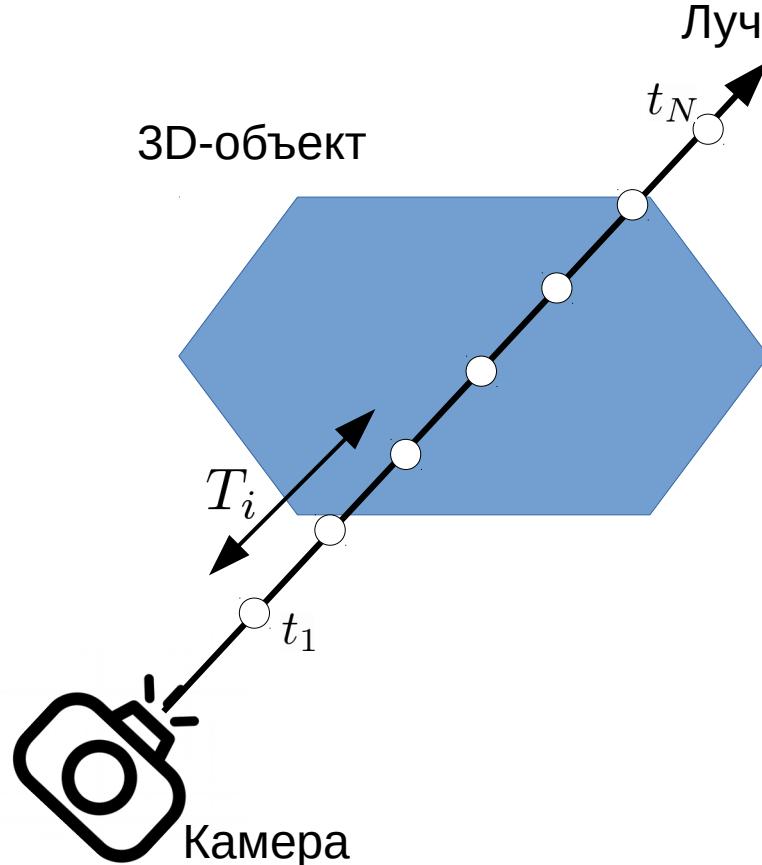
Луч:  $r(t) = x_0 + t\mathbf{d}$

$$Color(r) \simeq \sum_{i=1}^N T_i \alpha_i c_i$$

Вес      Цвет

Сколько света заблокировано ранее:

$$T_i := \prod_{j=1}^{i-1} (1 - \alpha_j)$$



# NeRF

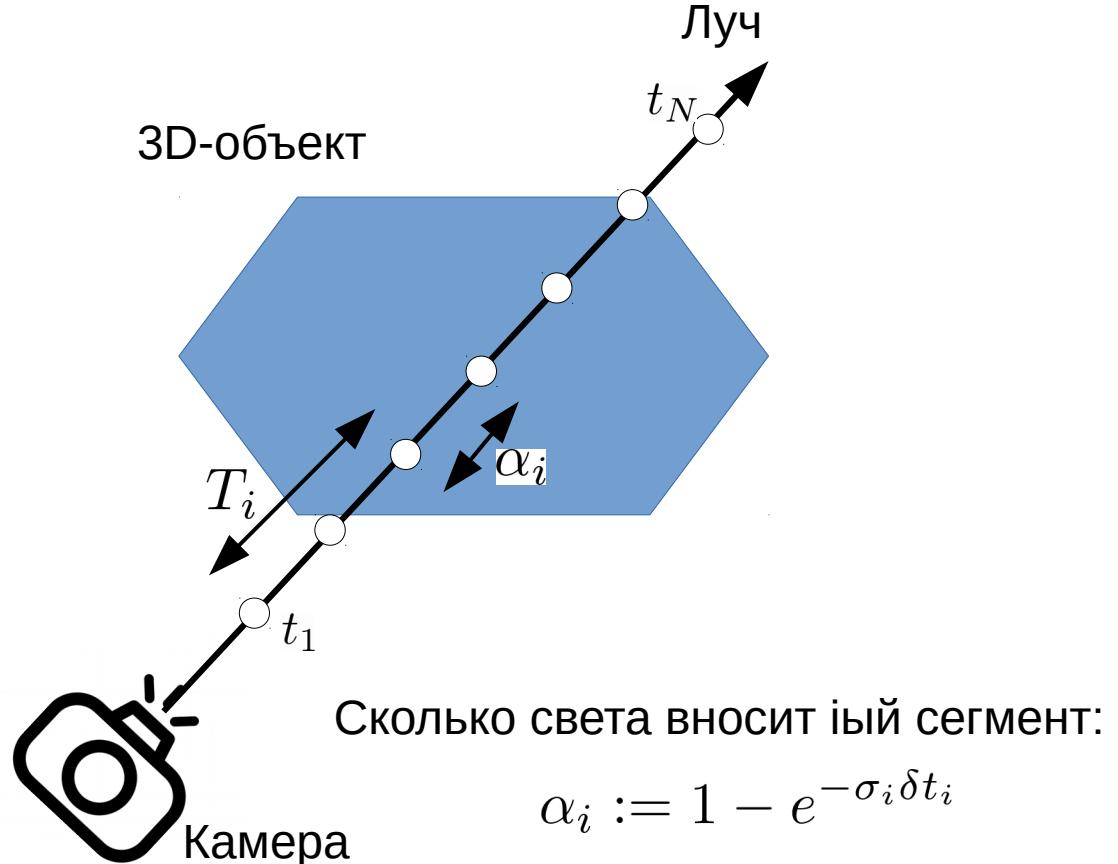
Луч:  $r(t) = x_0 + t\mathbf{d}$

$$Color(r) \simeq \sum_{i=1}^N T_i \alpha_i c_i$$

Вес      Цвет

Сколько света заблокировано ранее:

$$T_i := \prod_{j=1}^{i-1} (1 - \alpha_j)$$

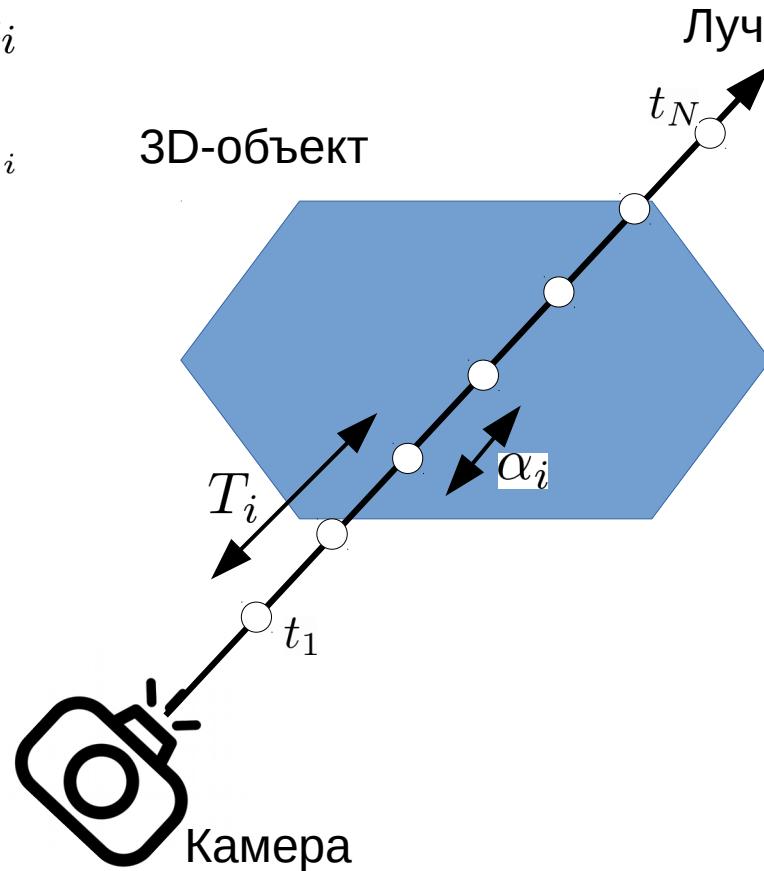


# NeRF

$$r(t) = x_0 + t\mathbf{d} \quad Color(r) \simeq \sum_{i=1}^N T_i \alpha_i c_i$$
$$T_i := \prod_{j=1}^{i-1} (1 - \alpha_j) \quad \alpha_i := 1 - e^{-\sigma_i \delta t_i}$$

---

В отличие от RGBa модели теперь мы можем брать сколько угодно точек, сколько угодно близко



# NeRF

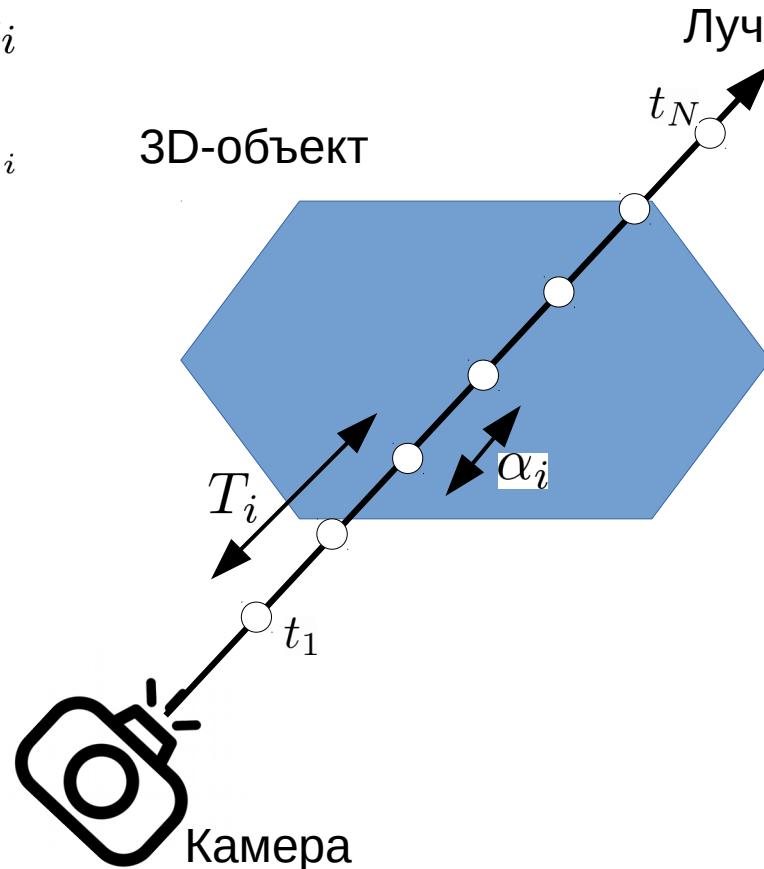
$$r(t) = x_0 + t\mathbf{d} \quad Color(r) \simeq \sum_{i=1}^N T_i \alpha_i c_i$$

$$T_i := \prod_{j=1}^{i-1} (1 - \alpha_j) \quad \alpha_i := 1 - e^{-\sigma_i \delta t_i}$$

---

Заметим, что разрешение изображения зависит от того, сколько и какие точки на луче мы берем

Можем взять очень много, очень близко точки... Но это плохо



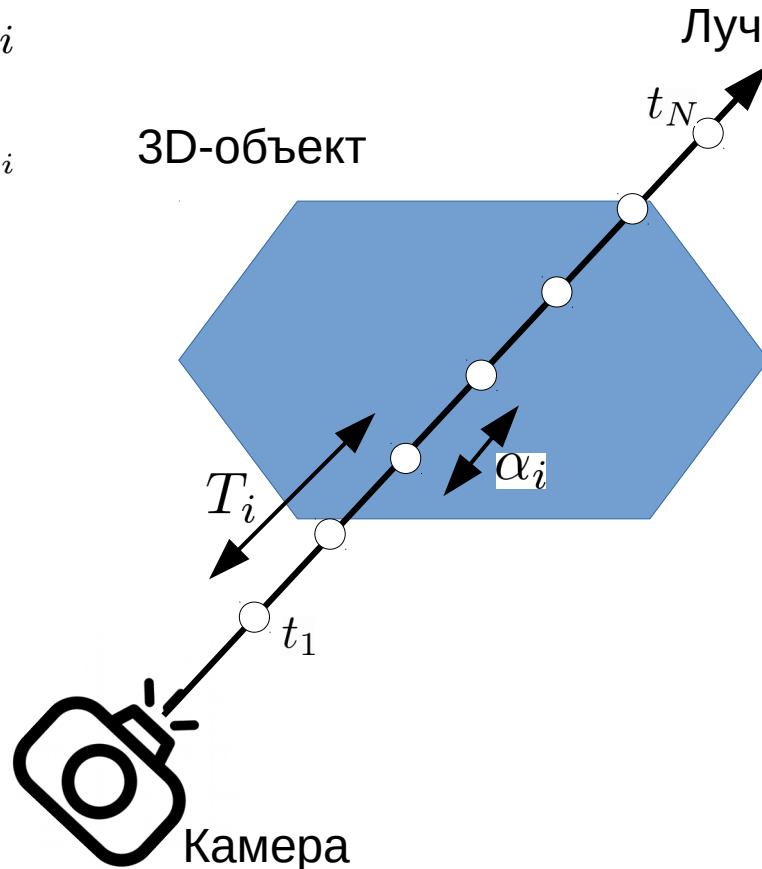
# NeRF

$$r(t) = x_0 + t\mathbf{d} \quad Color(r) \simeq \sum_{i=1}^N T_i \alpha_i c_i$$
$$T_i := \prod_{j=1}^{i-1} (1 - \alpha_j) \quad \alpha_i := 1 - e^{-\sigma_i \delta t_i}$$

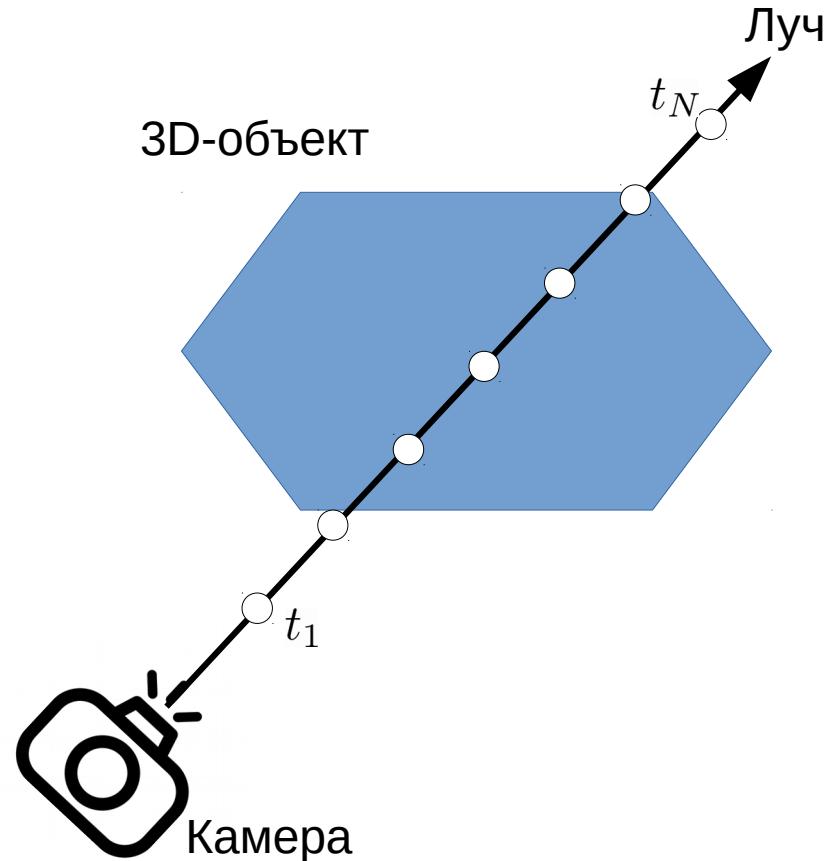
---

Заметим, что разрешение изображения зависит от того, сколько и какие точки на луче мы берем

Можем взять очень много, очень близко точки... Но это плохо (Каждая точка — прогон через сеть)

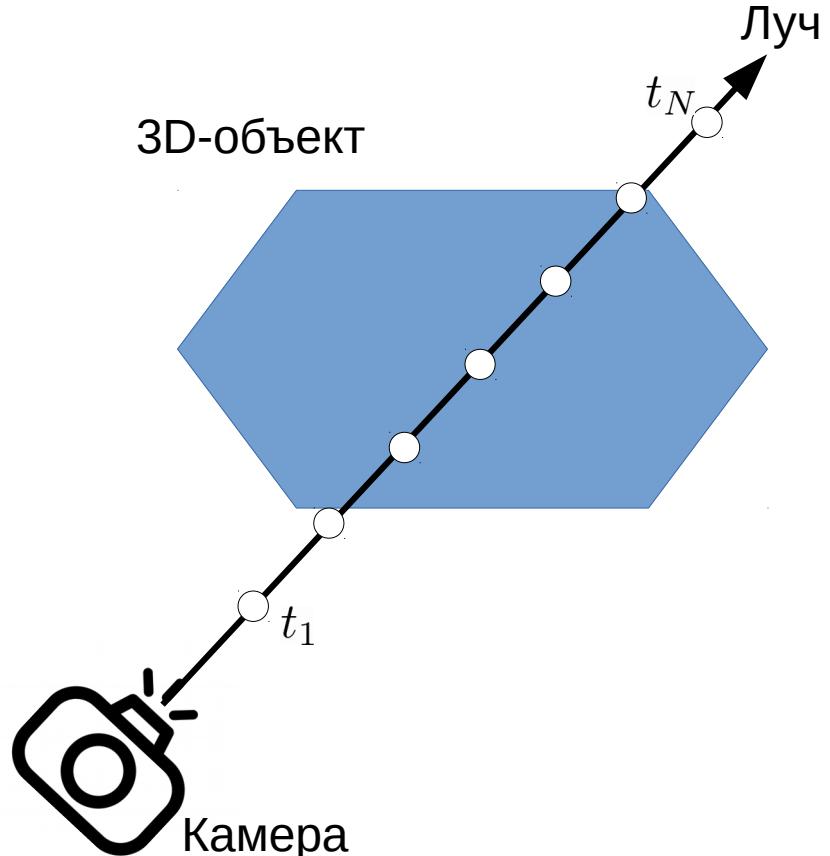


# NeRF: Как подобрать точки луча



# NeRF: Как подобрать точки луча

1. Запускаем первый проход на N равноудаленных точках

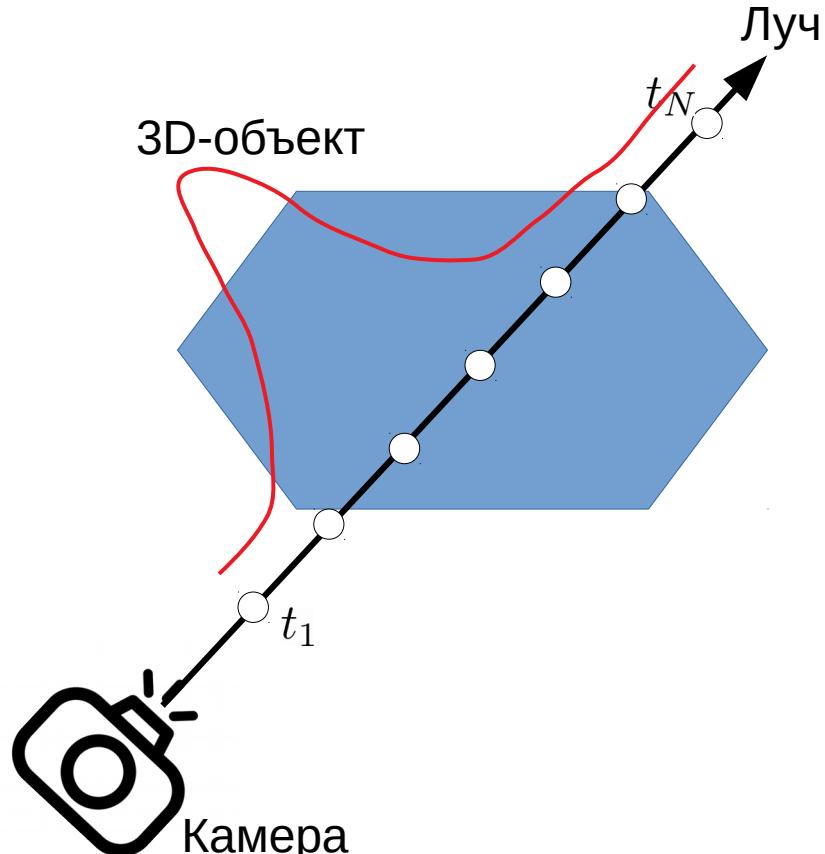


# NeRF: Как подобрать точки луча

1. Запускаем первый проход на N равноудаленных точках
2. Собираем новый набор точек

$$Color(r) \simeq \sum_{i=1}^N T_i \alpha_i c_i$$

Вероятность  
получить точку при  
генерации (после  
нормализации)

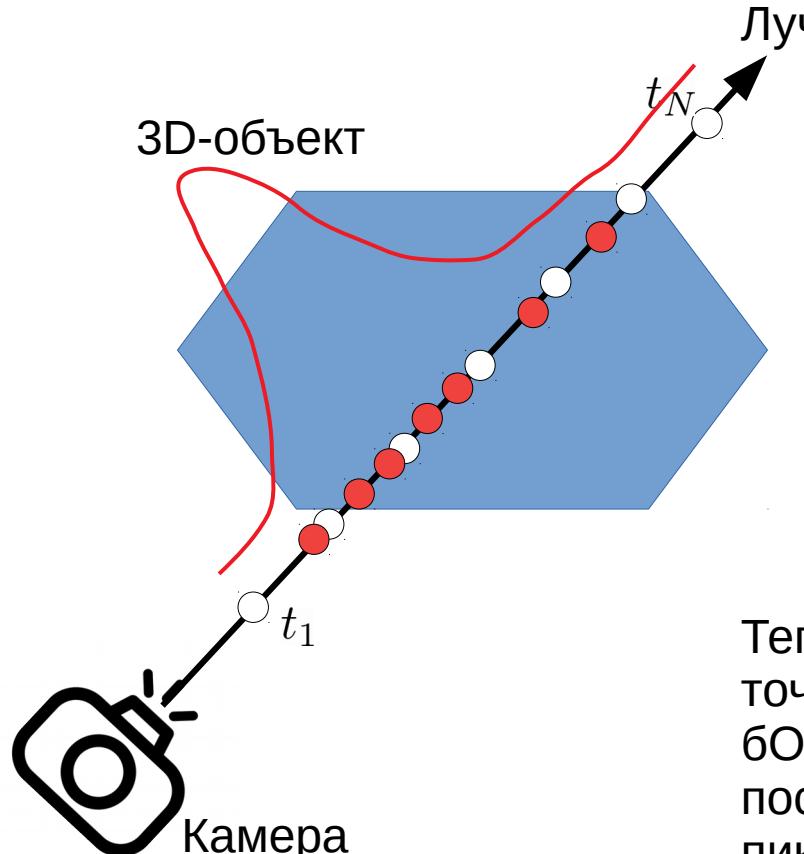


# NeRF: Как подобрать точки луча

1. Запускаем первый проход на N равноудаленных точках
2. Собираем новый набор точек

$$Color(r) \simeq \sum_{i=1}^N T_i \alpha_i c_i$$

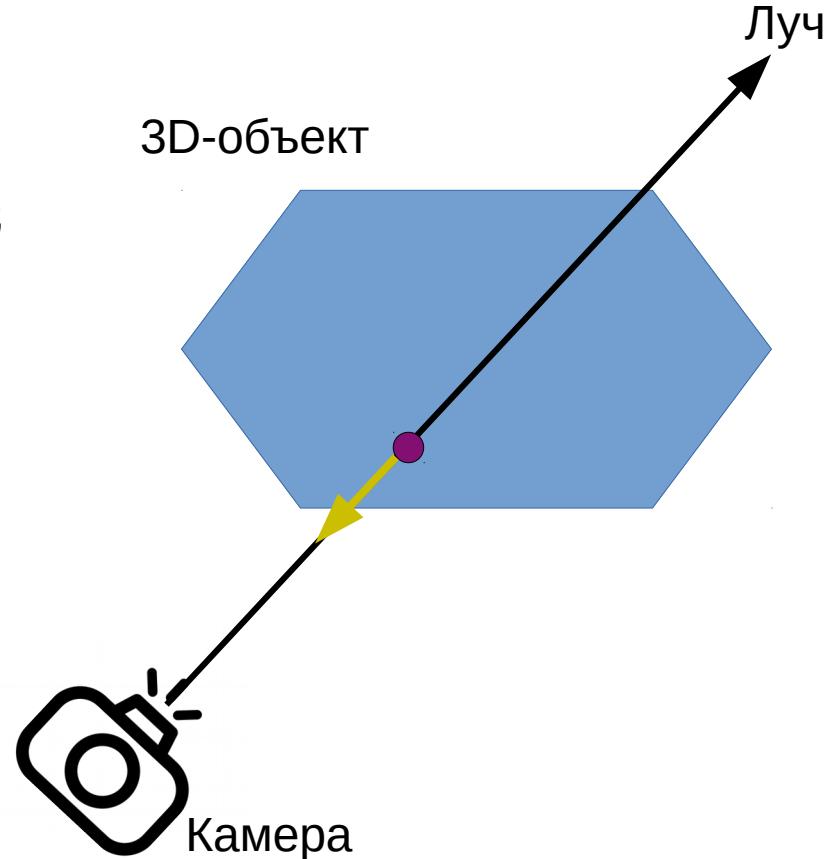
Вероятность  
получить точку при  
генерации (после  
нормализации)



Теперь у нас больше  
точек, вносящих  
больший вклад в  
построении итогового  
пикселя

# NeRF: Зависимость от обзора

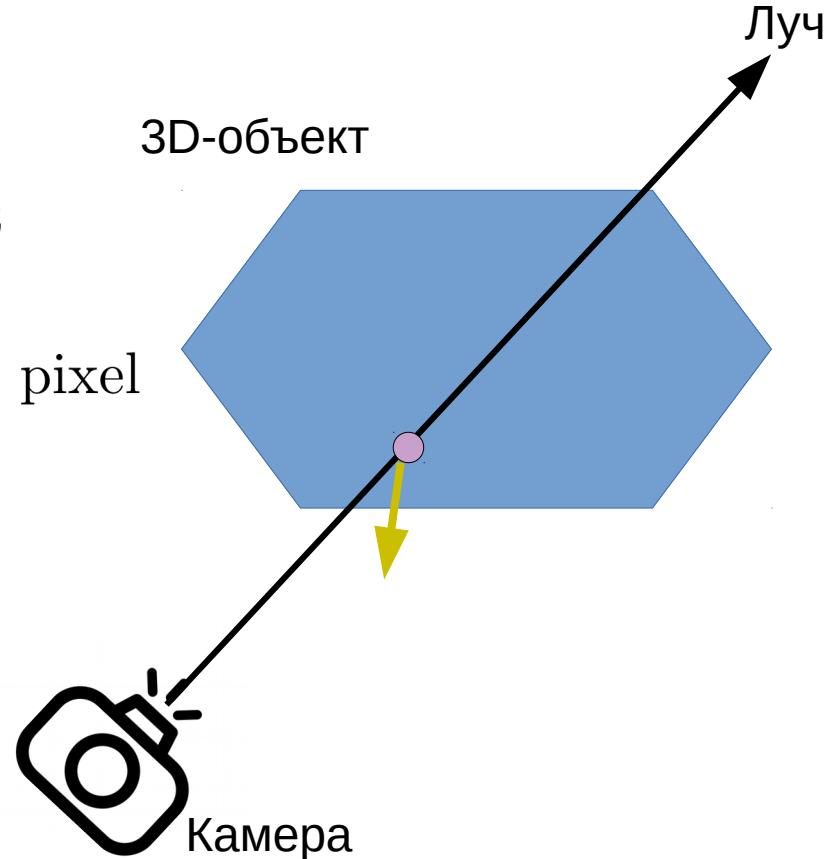
$(x, y, z, \theta, \phi)$  – input



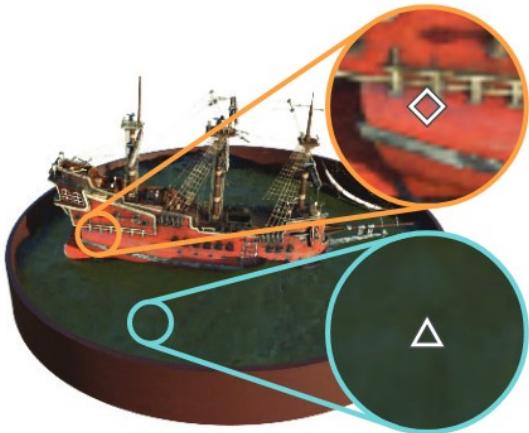
# NeRF: Зависимость от обзора

$(x, y, z, \theta, \phi)$  – input

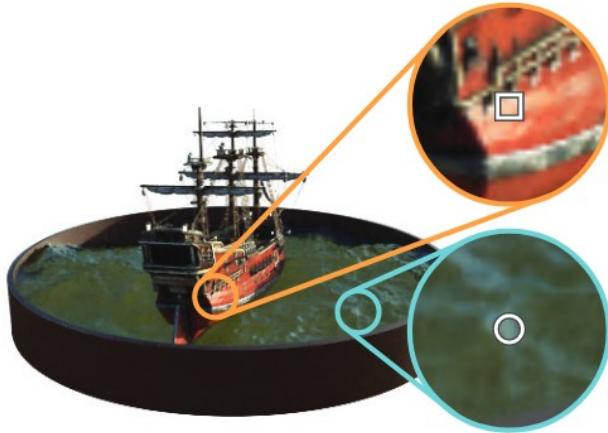
$color(\theta, \phi)$  – Voxel grid pixel



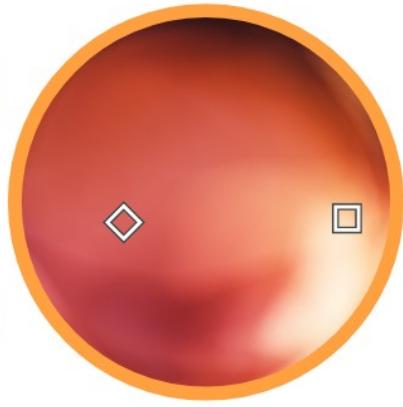
# NeRF: Зависимость от обзора



(a) View 1



(b) View 2

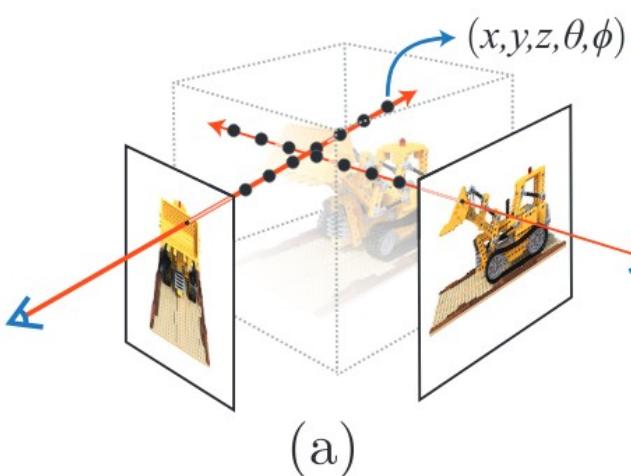


(c) Radiance Distributions

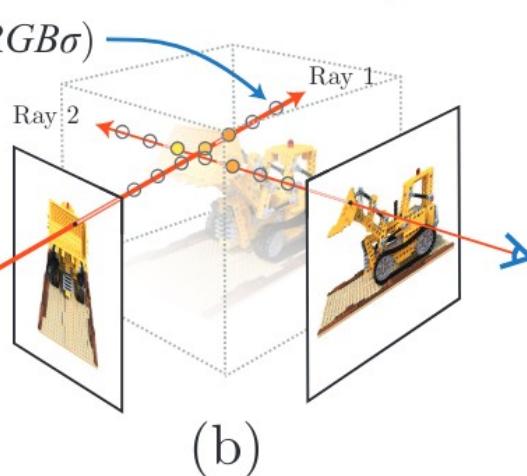


# NeRF: Summary

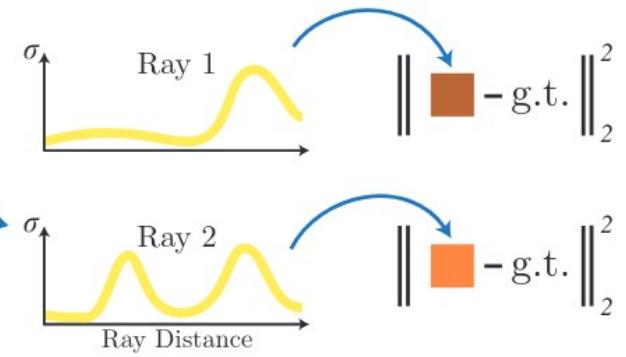
5D Input  
Position + Direction



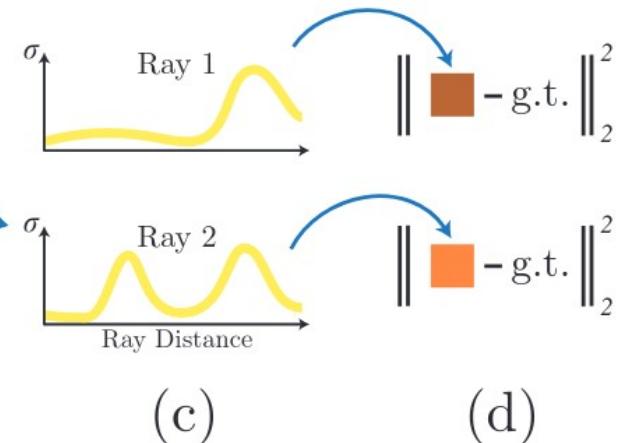
Output  
Color + Density



Volume  
Rendering



Rendering  
Loss



# NeRF: Summary

SRN [Sitzmann 2019]



NeRF



Nearest Input

# Почему NeRF лучше?



Naive

# Почему NeRF лучше?



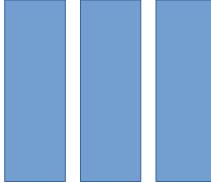
With positional encoding

# Почему NeRF лучше?



With positional encoding

# Почему NeRF лучше?

$(x, y) \rightarrow$    $\rightarrow (r, g, b)$

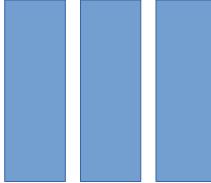


ground-truth



MLP

# Почему NeRF лучше?

$(x, y) \rightarrow$    $\rightarrow (r, g, b)$



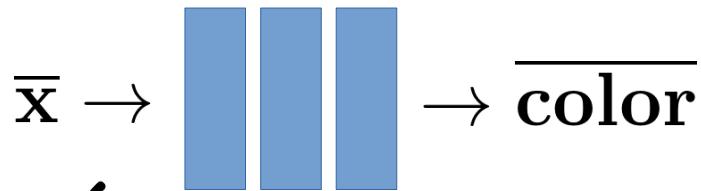
ground-truth

Хотя у НС в 8 раз больше параметров, чем пикселов у изображения!



MLP

# Positional encoding



$$\left( \begin{array}{l} \sin(\bar{x}), \cos(\bar{x}) \\ \sin(2\bar{x}), \cos(2\bar{x}) \\ \sin(4\bar{x}), \cos(4\bar{x}) \\ \vdots \\ \sin(2^N\bar{x}), \cos(2^N\bar{x}) \end{array} \right) \rightarrow \begin{array}{|c|c|c|}\hline & & \\ \hline\end{array} \rightarrow \overline{\text{color}}$$

# Positional encoding



ground-truth



Naive MLP

# Positional encoding



ground-truth



With encoding

# Positional encoding

**Проблема заключалась:**

НС склонны учить низко-частотные функции, что не соотносится с нашей функцией ( $r$ ,  $g$ ,  $b$ ,  $s$ ).

Утверждается, что в новом пространстве эта проблема решается.[4]

# Небольшие детали

**Используем две сети: «грубую» и «полноценную»**

**«Грубая»:** Генерируем  $N$  точек равно-удаленных и с помощью грубой сети считаем цвет и плотность в точках луча

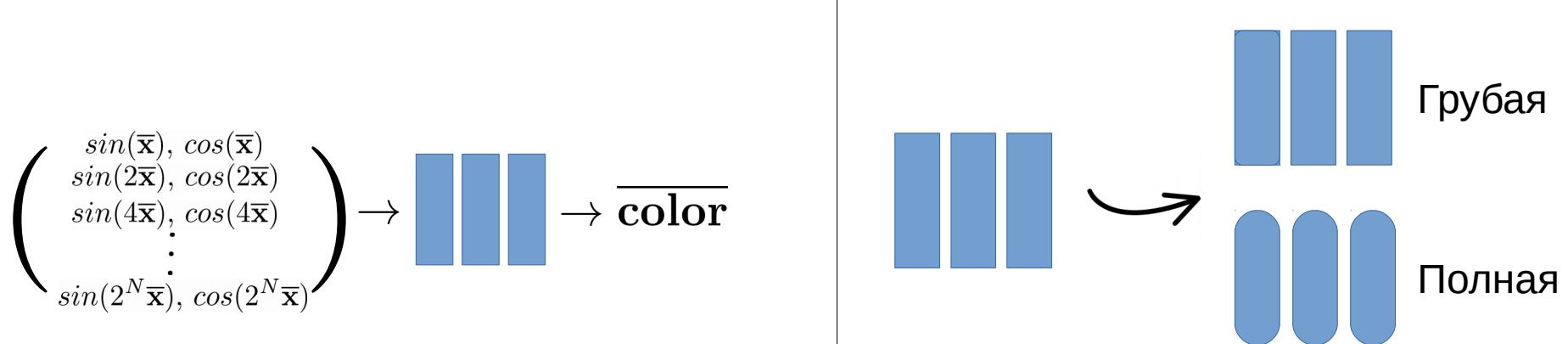
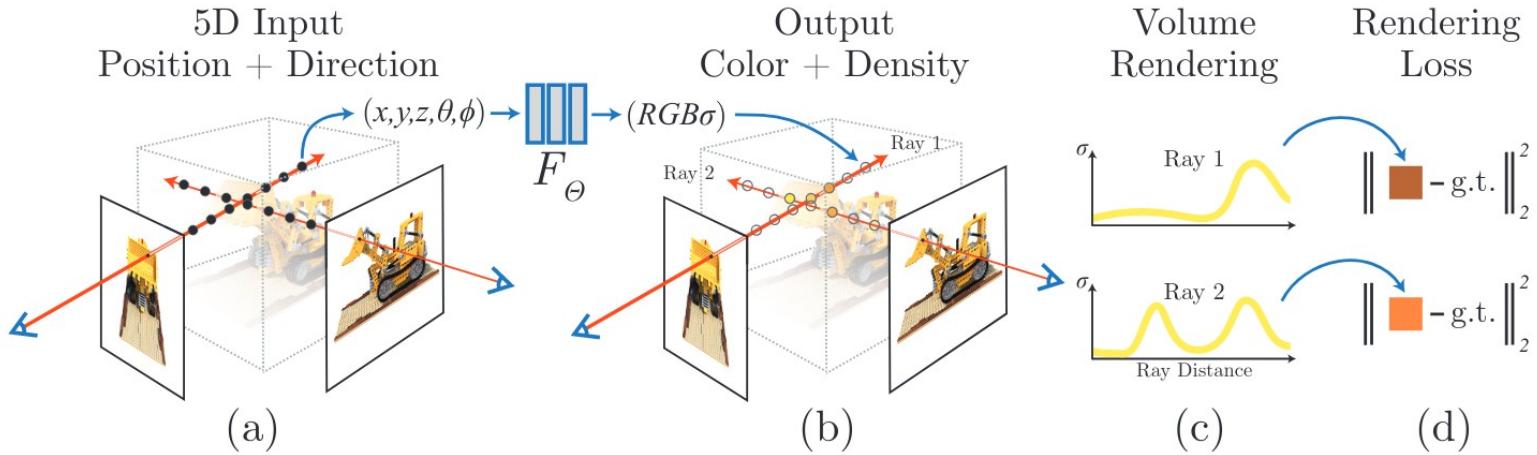
**«Полноценная»:** После получения новых точек, считаем на новом объединенном множестве цвет и плотность в точках луча.

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

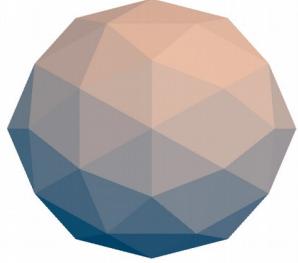
# Небольшие детали

	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)	<b>31.01</b>	<b>0.947</b>	<b>0.081</b>

# NeRF: complete summary



# Результаты



512x512



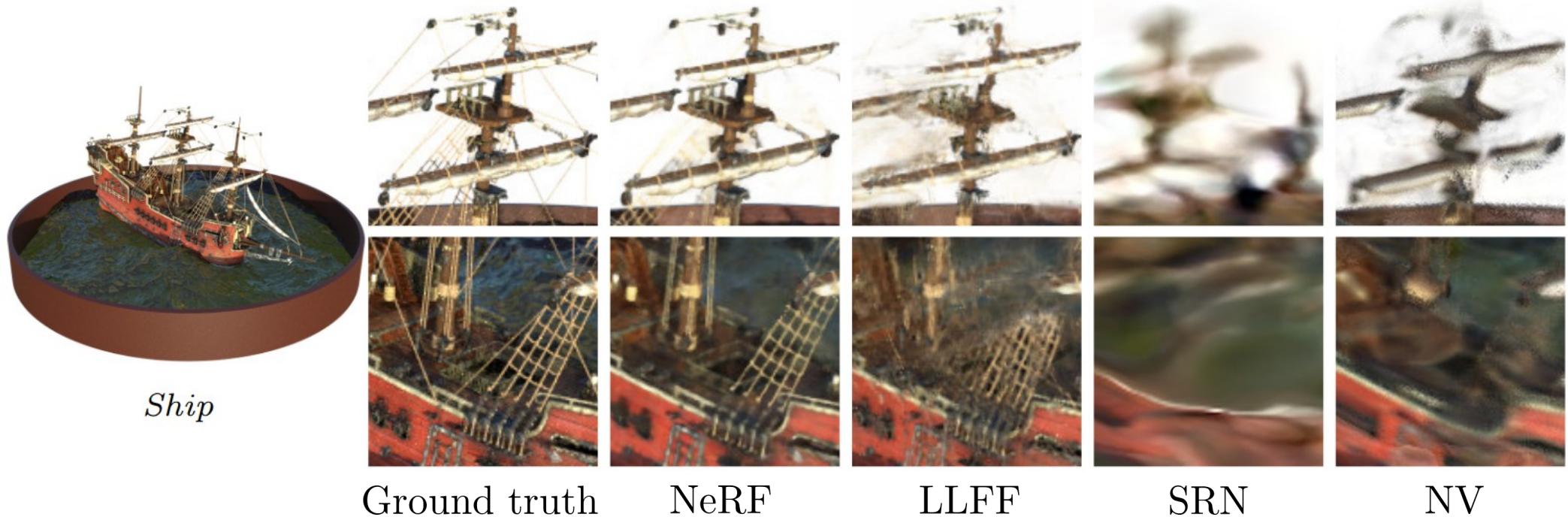
800x800



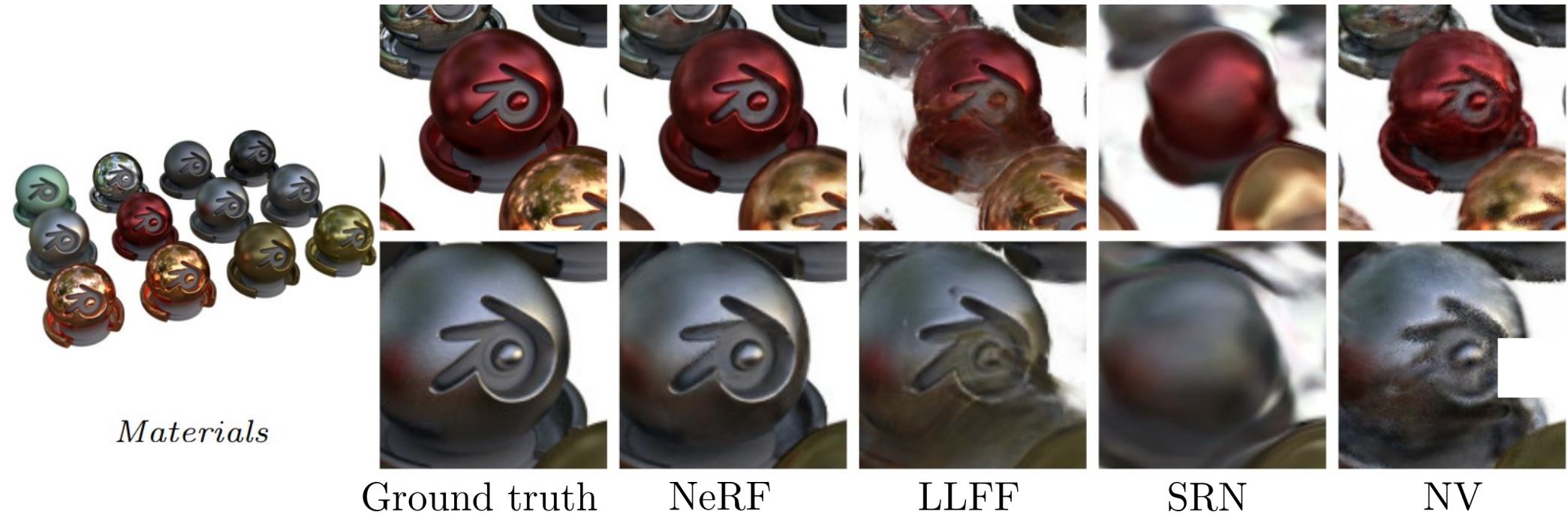
1008x756

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

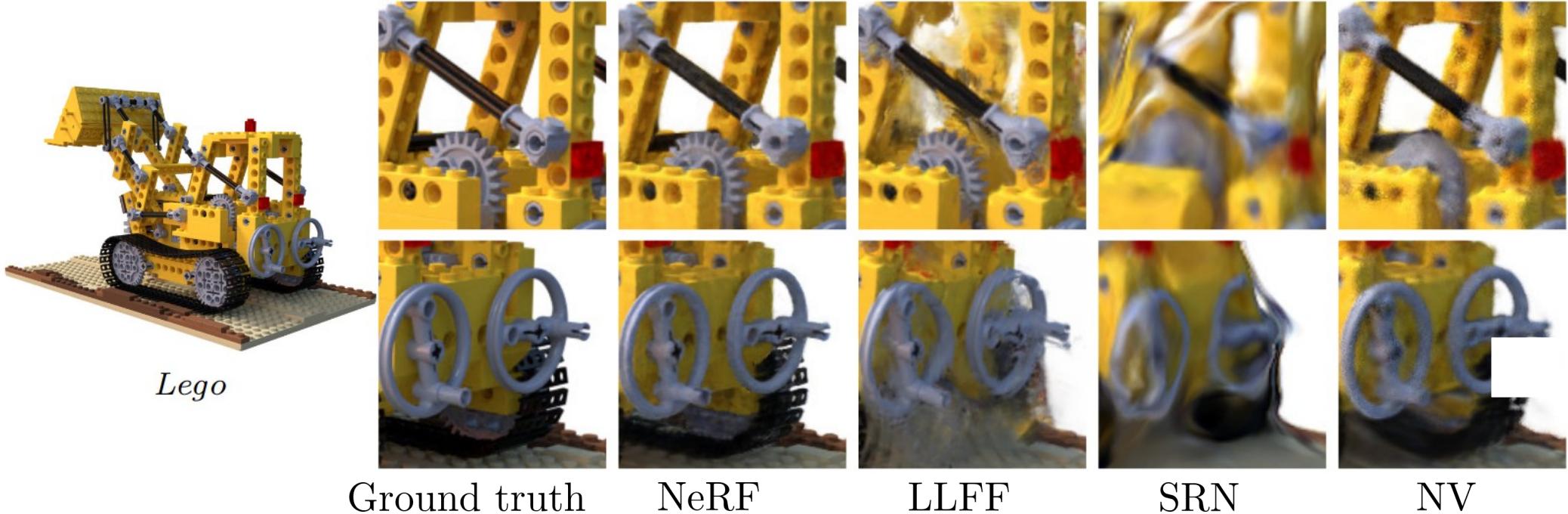
# Результаты



# Результаты



# Результаты



# Результаты



*Fern*



Ground truth



NeRF



LLFF



SRN

# Результаты



*Orchid*



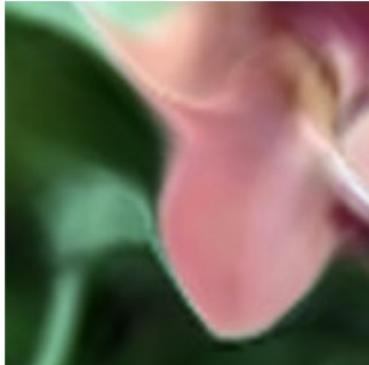
Ground truth



NeRF



LLFF



SRN

# Вопросы

## 1. Предоставьте общую схему работы NeRF:

- Что подается на вход?
- Что ожидается у нейронной сети на выходе?
- Как обучать такую сеть?
- Как строится итоговый видео-снимок?

*(Подробное описание каждого пункта не требуется, но хотелось бы каких-либо точных формулировок)*

## 2. В чем заключается **positional encoding** в контексте NeRF?

## 3. Подробно расскажите процесс получения 360°-модели объекта?

*При условии, что уже есть обученная модель и подготовлены снимки объекта со всех необходимых ракурсов.*

# Ссылки

## 1. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis

Mildenhall et al. 2020

<https://arxiv.org/pdf/2003.08934.pdf>

## 2. Neural Volumes: Learning Dynamic Renderable Volumes from Images

Lombardi et al. 2019

<https://arxiv.org/pdf/1906.07751.pdf>

## 3. Local Light Field Fusion: Practical View Synthesis with Prescriptive Sampling

Guidelines

Mildenhall et al. 2019

<https://arxiv.org/pdf/1905.00889.pdf>

## 4. Fourier Features Let Networks Learn High Frequency Functions in Low

Dimensional Domains

Tancik et al. 2020

<https://arxiv.org/pdf/2006.10739.pdf>

# Приложение: Метрики

$$PSNR := 20 \log_{10} \left( \frac{MAX_I}{\sqrt{MSE}} \right)$$

$$MSE := \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i, j) - K(i, j)]^2$$

Max pixel  
value

$$SSIM(x, y) := \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$