Continuous 3D Scene Representations with Implicit Functions

Michaël Ramamonjisoa, Van Nguyen Nguyen

ENPC's Imagine Seminars 16/12/2020

Outline

1. **Implicit functions: an illustration with 3D surface representation**

2. Neural Radiance Field (NeRF)

Explicit vs Implicit Representation (2D)

Explicit:

$$\mathbf{f}(\alpha) = (r\cos(\alpha), r\sin(\alpha))^T$$

Domain: $[0,2\pi]$

Implicit:

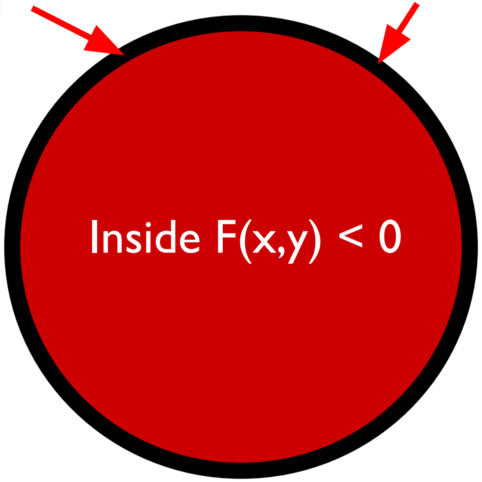
$$F(x,y) = \sqrt{x^2+y^2} - r$$

Domain: (x,y) $\in \mathbb{R}^2$

$\Longrightarrow$ Circle is implicitly defined by $\{(x,y)|F(x,y)=0\}$

$\mathbf{f}([0,2\pi])$

Interface F(x,y) = 0



Outside F(x,y) > 0

$\mathbf{f}(\alpha)$ defines the interface

$F(x,y)$ defines the **Signed Distance Function** of the circle

Explicit vs Implicit Representation (3D)

Explicit:

$$\mathbf{f}(\alpha, \beta) = (r \sin(\alpha) \cos(\beta), -r \cos(\beta), r \sin(\alpha) \sin(\beta))$$

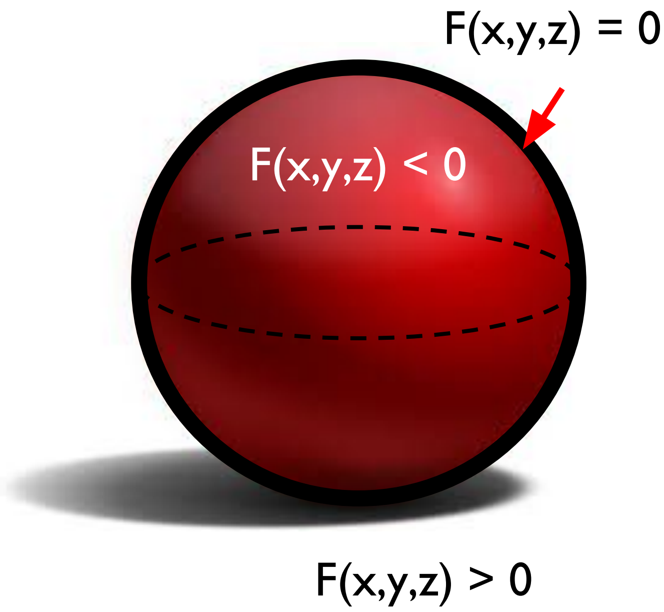
Domain: $\alpha \in [0; 2\pi], \beta \in [0; \pi]$

Implicit:

$$F(x, y, z) = \sqrt{x^2 + y^2 + z^2} - r$$

Domain: $(x,y,z) \in \mathbb{R}^3$

$\Longrightarrow$ Sphere is implicitly defined by $\{(x, y, z) | F(x, y, z) = 0\}$



$\mathbf{f}(\alpha, \beta)$ defines the 3D surface

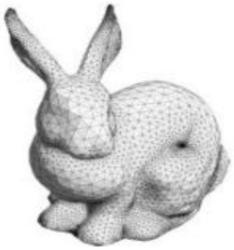
$F(x, y, z)$ defines the **Signed Distance Function** of the sphere

Representing 3D surfaces

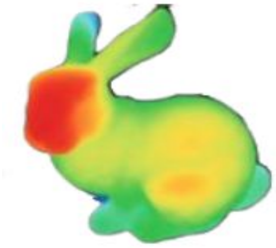
Explicit:

Voxels

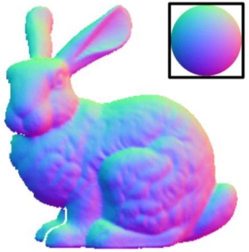
Point clouds



Mesh

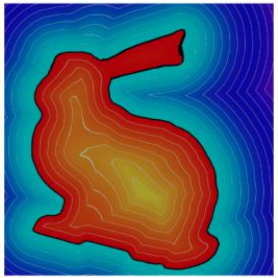


Depth

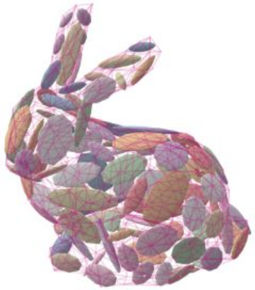


Surface Normals

Implicit:



Signed distance field



Mixture of primitives
(e.g gaussian mixtures)

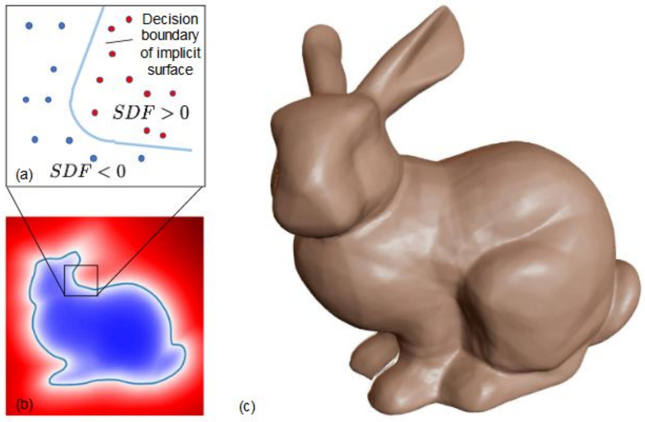
Thomas Funkhouser’s talk at 3DGV seminar

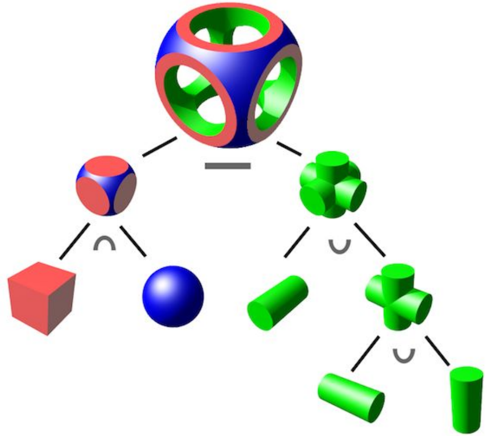
Signed Distance Field (SDF)

- Maps each 3D points ***p*** to it's signed distance from to the object surface *S*

$$SDF(\boldsymbol{p}) = \min_{q \in S} ||\boldsymbol{p} - q||$$

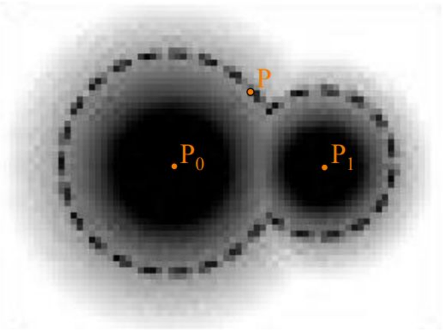
- Sign indicates whether the point ***p*** is inside (-) or outside (+) of the shape
- Shape's boundary as the zero-level-set of SDF
- Allows for Constructive Solid Geometry (CSG) through boolean operations





Mixture of Gaussians

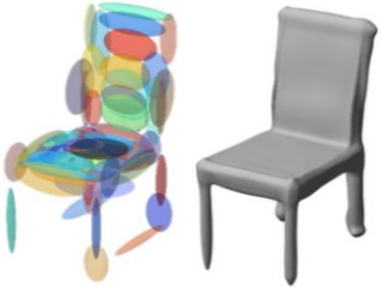
- Represents a shape as a mixture of local implicit functions (3D gaussians)



$$F(\mathbf{x}, \mathbf{\Theta}) = \sum_{i \in [N]} f_i(\mathbf{x}, \theta_i)$$

$$f_i(\mathbf{x}, \theta_i) = c_i \exp\left(\sum_{d \in \{x,y,z\}} \frac{-(\mathbf{p}_{i,d} - \mathbf{x}_d)^2}{2\mathbf{r}_{i,d}^2} \right)$$

- Shape's boundary is defined as an iso-level of the **global** implicit function



[1] Genova19
[2] Genova20

Representing 3D surfaces with Implicit Functions

Pros:

- Compared to **point clouds**: **clearly defines the (iso-)surface**
- Compared to **meshes**: can continuously **adapt to arbitrary topology**
- Compared to **voxels**: can be represented with **few parameters** (e.g. mixture of simple implicit functions)
- They are **continuous** in 3D
- Can give analytic normals, can be applied with boolean operations, etc

Representing 3D surfaces with Implicit Functions

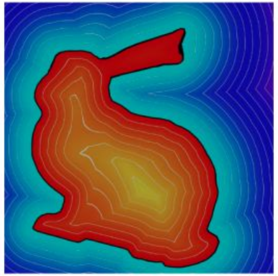
Pros:

- Compared to **point clouds**: **clearly defines the (iso-)surface**
- Compared to **meshes**: can continuously **adapt to arbitrary topology**
- Compared to **voxels**: can be represented with **few parameters** (e.g. mixture of simple implicit functions)
- They are **continuous** in 3D
- Can give analytic normals, can be applied with boolean operations, etc

Cons:

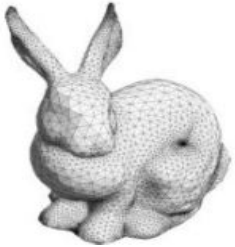
- SDF is well-defined for only watertight meshes (there is an interior and an exterior)
- Need extra steps to visualize

Converting Implicit Surfaces to meshes



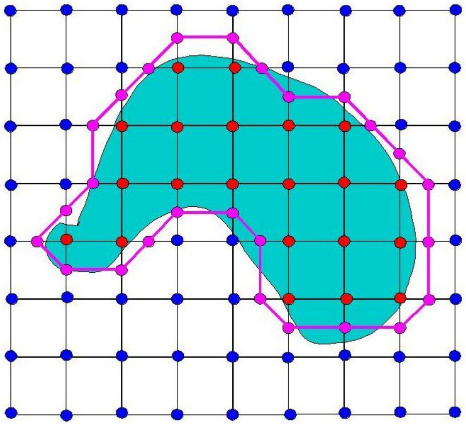
Implicit function

Extract (zero-level) iso-surface

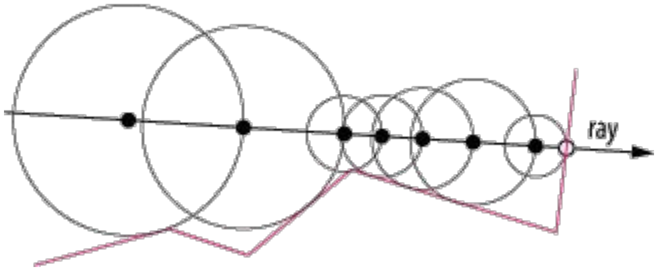


Mesh

Marching Cubes



Ray marching



Representing 3D surfaces with Implicit Functions

Pros:

- Compared to **point clouds**: **clearly defines the (iso-)surface**
- Compared to **meshes**: can continuously **adapt to arbitrary topology**
- Compared to **voxels**: can be represented with few parameters (e.g. mixture of simple implicit functions
- They are **continuous** in 3D
- Can give analytic normals, can be applied with boolean operations, etc

Cons:

- Implicit functions is well-defined for only watertight meshes (there is an interior and an exterior)
- Need extra steps to visualize
- Not all complex shapes can be efficiently / accurately represented with simple primitives

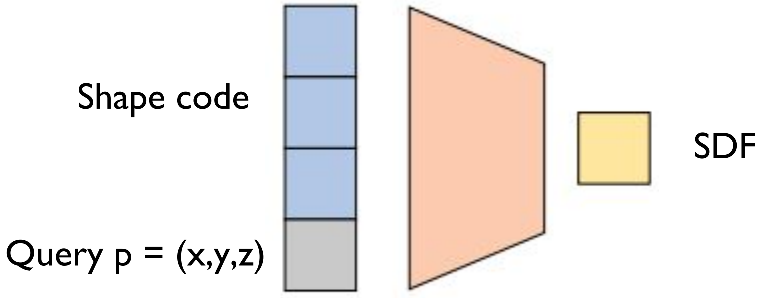
Representing 3D surfaces

DeepSDF: **Efficiently** representing complex shapes by learning their SDF

Idea: Learn a **continuous** representation of 3D implicit surfaces

Query p = (x,y,z), Shape latent code **Z**

$$F(p; \mathbf{Z}) = SDF(p, \mathcal{M})$$

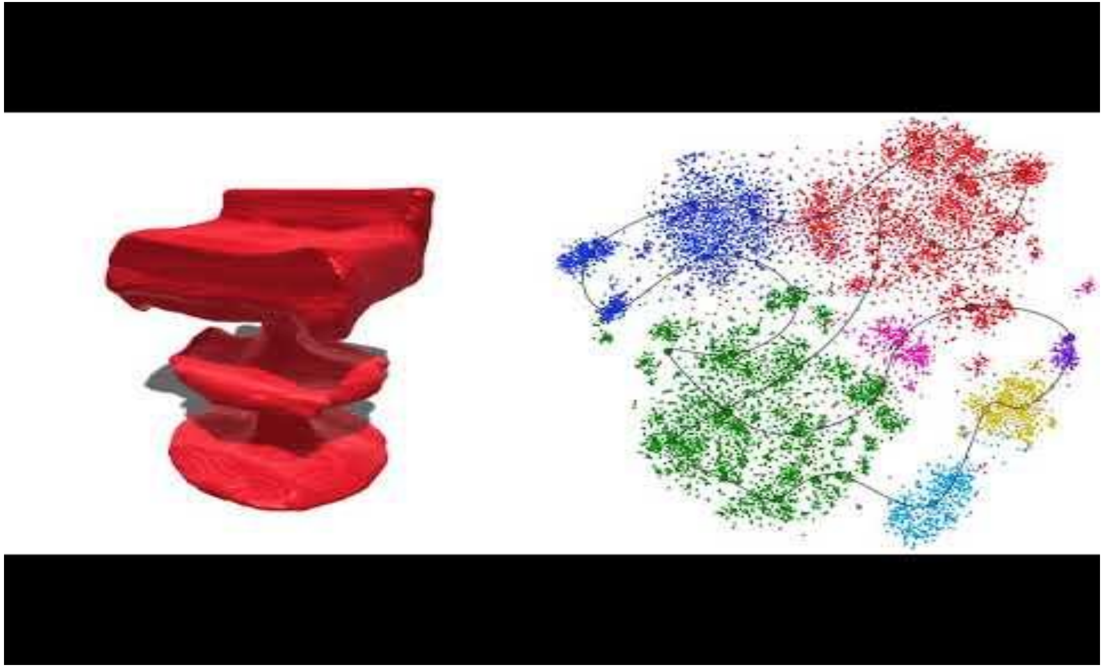


=> **Continuity** in 3D space AND shapes space

[3] Park19

Representing 3D surfaces

DeepSDF: Representing complex shapes by learning their SDF



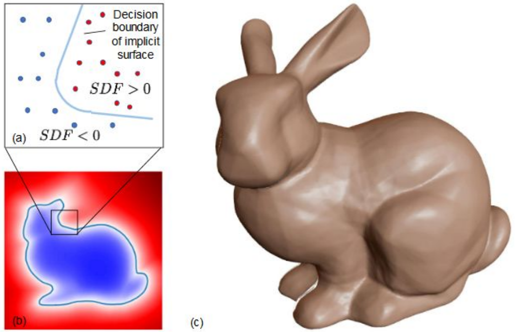
[3] Park19

Take home message on Implicit Functions

Representation of a continuous field

Learned implicit functions:

- Can represent complex shapes
- Are **continuous mappings** because they use **MLPs**
- Are applicable to N-D data: 2D images, 3D shapes, radiance fields

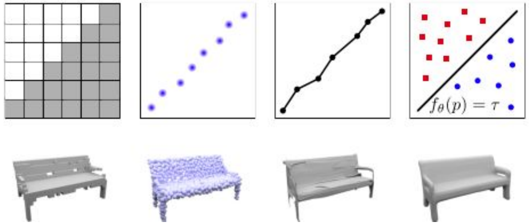


Visualization of implicit functions is done by extracting iso-surfaces:

1. Running inference for multiple queries in input space
2. Rendering the result by combining the queries

More works on Implicit Functions for 3D shape

- Occupancy Networks



[4] Mescheder19

- PiFu and PiFuHD

[5] Saito19
[6] Saito20

References

[1] Genova et al., Learning Shape Templates with Structured Implicit Functions, ICCV 2019
[2] Genova et al., Local Deep Implicit Functions for 3D Shape, CVPR 2020
[3] Park et al., DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation, CVPR 2019
[4] Mescheder et al., Occupancy Networks: Learning 3D Reconstruction in Function Space, CVPR 2019
[5] Saito, Huang, Natsume et al., PIFu: Pixel-Aligned Implicit Function for High-Resolution Clothed Human Digitization, ICCV 2019
[6] Saito et al., PIFuHD: Multi-Level Pixel-Aligned Implicit Function for High-Resolution 3D Human Digitization, CVPR 2020

Courses and Seminars
Lecture on Implicit geometry
Lecture on Implicit surface
Lecture on Explicit & Implicit Surfaces

Thomas Funkhouser's talk at 3DGV seminar

Princeton COS 426, Spring 2014 on Implicit Surfaces & Solid Representations

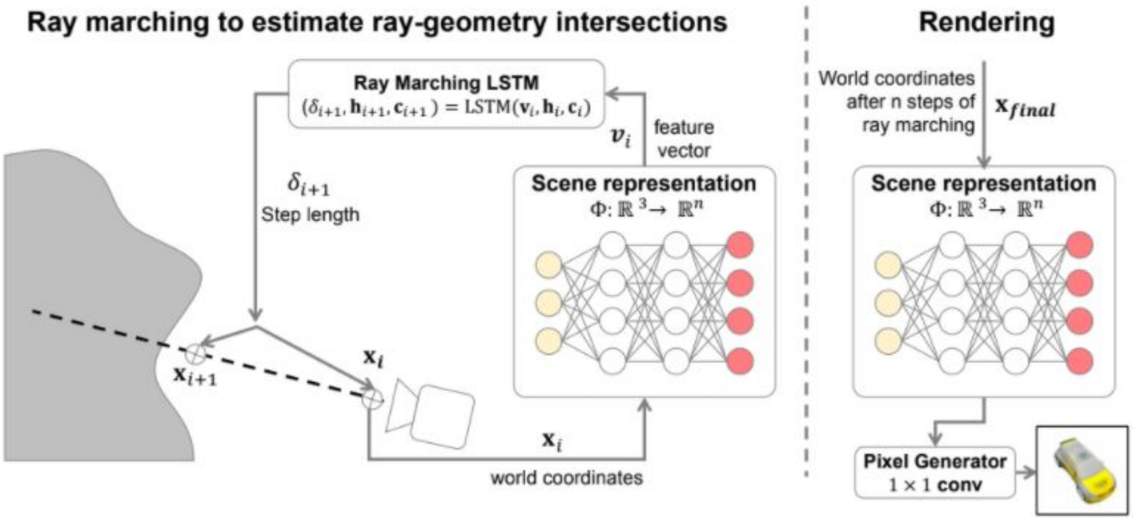
Outline

1. Implicit functions: an illustration with 3D surface representation

2. **Neural Radiance Field (NeRF)**

Scene Representation Networks: Continuous 3D-Structure-Aware Neural Scene Representations

- One of the first relevant works on scene representation, also benchmark for most of NeRF’s paper

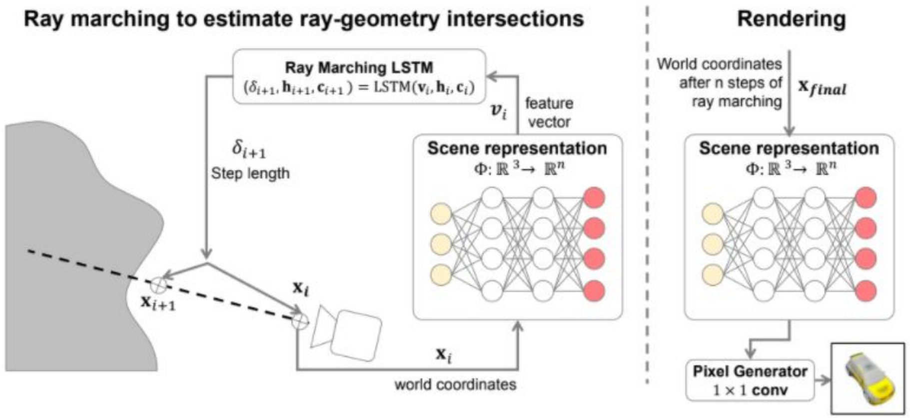


Scene Representation Networks: Continuous 3D-Structure-Aware Neural Scene Representations

- Represent a scene as a function Φ which maps a spatial location x to a feature representation v

$$\Phi : \mathbb{R}^3 \to \mathbb{R}^n, \quad \mathbf{x} \mapsto \Phi(\mathbf{x}) = \mathbf{v}.$$

- v may encode:
  - visual information: **surface color** or reflectance
  - geometry: signed distance of x
- Then learn a differentiable renderer to render v (using LSTM)

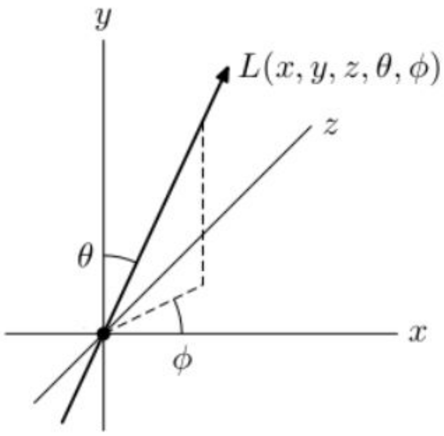


Definition of radiance field

- Radiance field is a 5-dimensional function which maps a 3D location **x** and a direction in 3D sphere **d** to a color **(r,g,b)**:

$$L : R^3 x S^2 \to R^3$$

$$L(\underline{x}, \underline{\mathbf{d}}) = (r, g, b)$$



- Intuitively, "radiance" is the amount of light energy passing through a given point in space, heading in a given direction

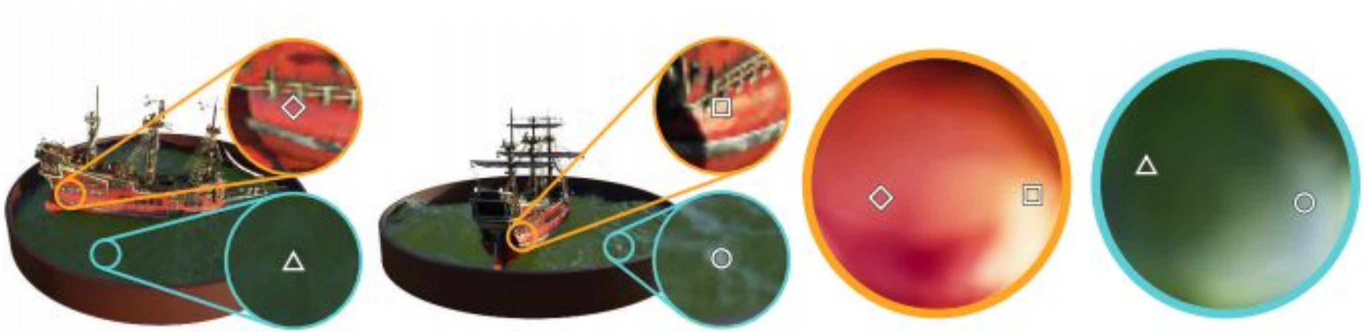
- In NeRF, there is an additional output is volume density $\boldsymbol{\sigma} \in R$

$$L(\underline{x}, \underline{\mathbf{d}}) = (r, g, b, \boldsymbol{\sigma})$$

http://reedbeta.com/blog/the-radiance-field/

Definition of radiance field

- Radiance field is a 5-dimensional function which maps a 3D location **x,y,z** and a direction in 3D sphere **d** to a color **(r,g,b):**



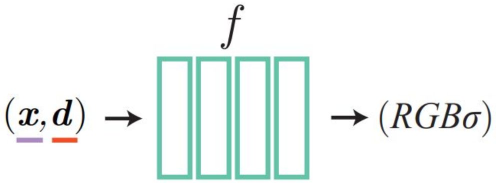
(a) View 1 (b) View 2 (c) Radiance Distributions

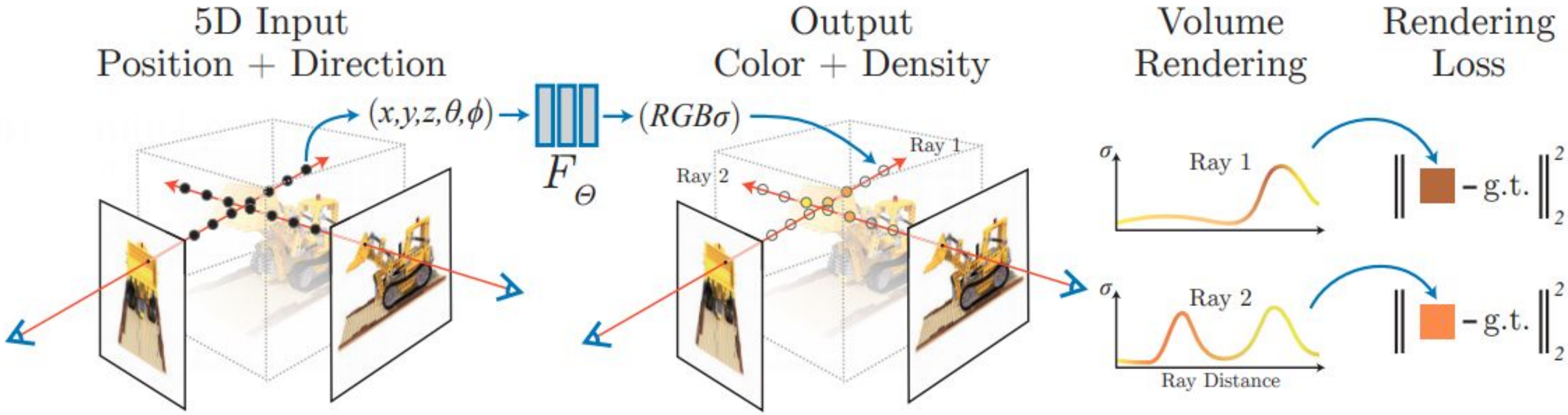
[7] Mildenhall20

Neural Radiance field (NeRF)

Idea:

- Continuous neural networks as a view-dependent volumetric scene representation (xyz + view direction d)
- Using volumetric rendering to synthesize new views





[7] Mildenhall20

Neural Radiance field (NeRF)

Volumetric rendering with ray tracing:

Opacity Predicted colors

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

Volume density

Rendering model for ray r(t) = o + td:

$$C \approx \sum_{i=1}^{N} T_i \alpha_i c_i$$

colors

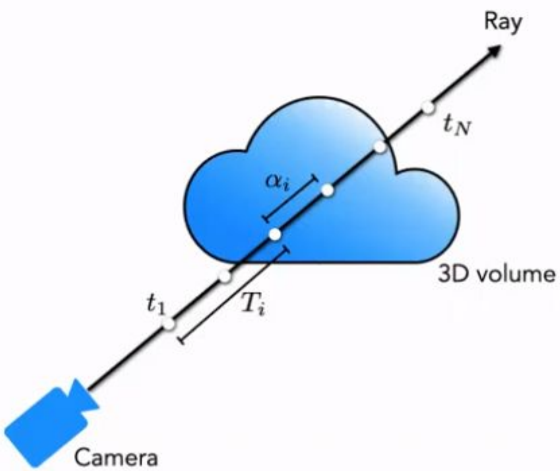
Opacity

How much light is blocked earlier along ray:

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

How much light is contributed by ray segment *i*:

$$\alpha_i = 1 - e^{-\sigma_i \delta t_i}$$



[7] Mildenhall20

Neural Radiance field (NeRF)

Volumetric rendering with ray tracing:

Opacity

Predicted colors

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

Volume density

Rendering model for ray r(t) = o + td:

(approximation with numerical quadrature)

$$\boxed{C \approx \sum_{i=1}^{N} T_i \alpha_i c_i}$$

colors

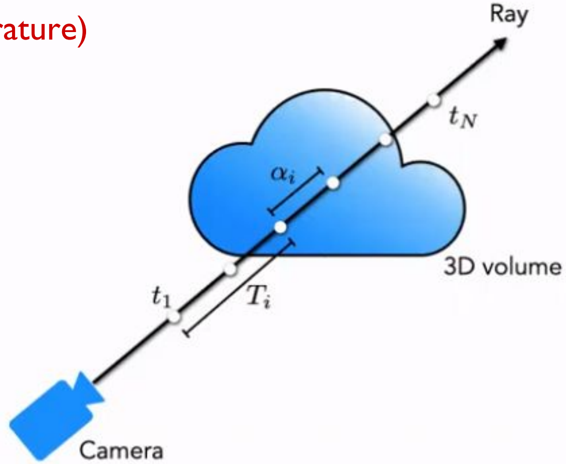
Opacity

How much light is blocked earlier along ray:

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

How much light is contributed by ray segment *i*:

$$\alpha_i = 1 - e^{-\sigma_i \delta t_i}$$



[7] Mildenhall20

Neural Radiance field (NeRF)

Volumetric rendering with ray tracing:

Opacity　　Predicted colors

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

Volume density

Rendering model for ray r(t) = o + td:

$$\boxed{C \approx \sum_{i=1}^{N} T_i \alpha_i c_i}$$

colors

Opacity

(approximation with numerical quadrature)

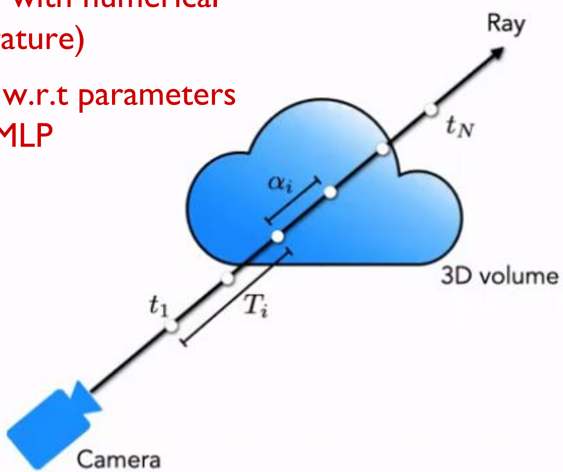
-> differentiable w.r.t parameters of MLP

How much light is blocked earlier along ray:

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

How much light is contributed by ray segment *i*:

$$\alpha_i = 1 - e^{-\sigma_i \delta t_i}$$

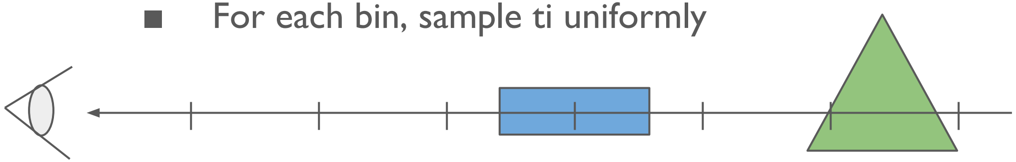


[7] Mildenhall20

Neural Radiance field (NeRF)

Tricks:

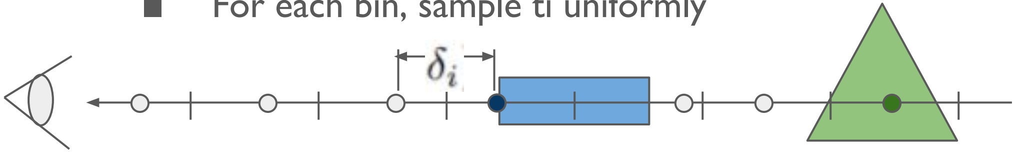
- **<u>Hierarchical Sampling:</u> coarse to fine importance sampling**
    - First sample coarsely along the ray with stratified sampling
        - Create Nc bins between tn and tf
        - For each bin, sample ti uniformly



Neural Radiance field (NeRF)

Tricks:

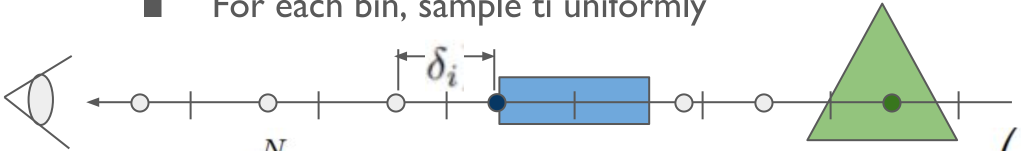
- **<u>Hierarchical Sampling:</u> coarse to fine importance sampling**
  - First sample coarsely along the ray with stratified sampling
    - Create Nc bins between tn and tf
    - For each bin, sample ti uniformly



Neural Radiance field (NeRF)

Tricks:

- <u>**Hierarchical Sampling:**</u> **coarse to fine importance sampling**
    - First sample coarsely along the ray with stratified sampling
        - Create Nc bins between tn and tf
        - For each bin, sample ti uniformly

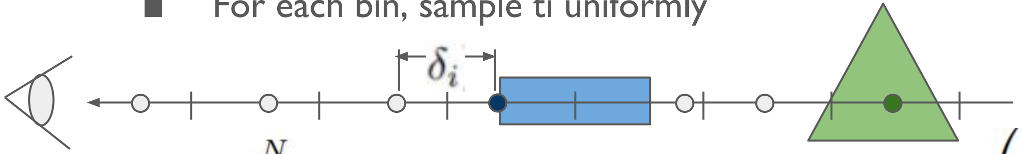


<u>(Ray-) Volume rendering</u> $$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} T_i(1 - \exp(-\sigma_i\delta_i))\mathbf{c}_i\,, \text{ where } T_i = \exp\left(-\sum_{j=1}^{i-1}\sigma_j\delta_j\right)$$

Neural Radiance field (NeRF)

Tricks:

- **<u>Hierarchical Sampling:</u> coarse to fine importance sampling**
  - First sample coarsely along the ray with stratified sampling
    - Create Nc bins between tn and tf
    - For each bin, sample ti uniformly



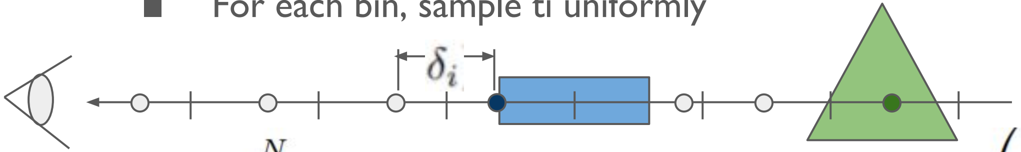
<u>(Ray-) Volume rendering</u> $$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} \underbrace{T_i(1 - \exp(-\sigma_i \delta_i))}_{\mathbf{w}_i} \mathbf{c}_i \,, \text{ where } T_i = \exp\left(- \sum_{j=1}^{i-1} \sigma_j \delta_j \right)$$

$\mathbf{w}_i \approx 0$

Neural Radiance field (NeRF)

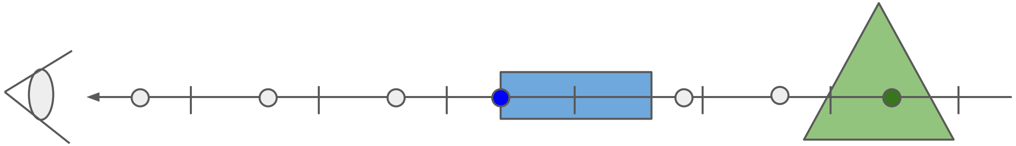
Tricks:

- **Hierarchical Sampling: coarse to fine importance sampling**
  - First sample coarsely along the ray with stratified sampling
    - Create Nc bins between tn and tf
    - For each bin, sample ti uniformly



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

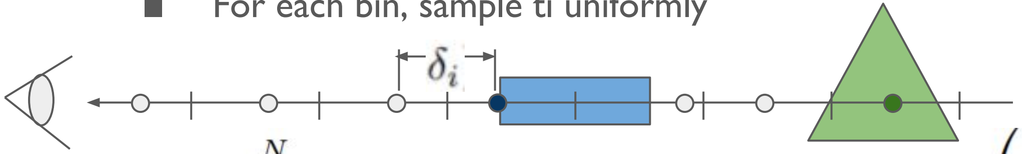
  - Then do importance sampling based on color weight $\mathbf{w}_i$



Neural Radiance field (NeRF)

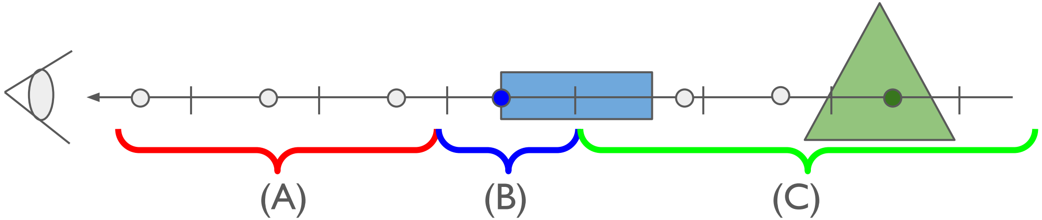
Tricks:

- **<u>Hierarchical Sampling:</u> coarse to fine importance sampling**
  - First sample coarsely along the ray with stratified sampling
    - Create Nc bins between tn and tf
    - For each bin, sample ti uniformly



<u>(Ray-) Volume rendering</u> $$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} \underbrace{T_i(1-\exp(-\sigma_i\delta_i))}_{\mathbf{w}_i}\mathbf{c}_i\,, \quad \text{where } T_i = \exp\left(-\sum_{j=1}^{i-1}\sigma_j\delta_j\right)$$

  - Then do importance sampling based on color weight $\mathbf{w}_i$



(A) $\boxed{T_i \approx 1,\ \ \sigma_i \approx 0 \quad \mathbf{w}_i \approx 0}$

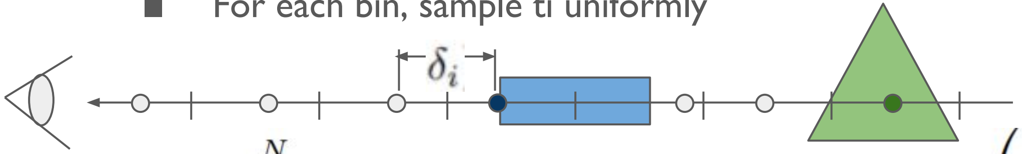
(B) $\boxed{T_i > 0,\ \ \sigma_i > 0 \quad \mathbf{w}_i > 0}$

(C) $\boxed{T_i \approx 0 \quad \mathbf{w}_i \approx 0}$

Neural Radiance field (NeRF)

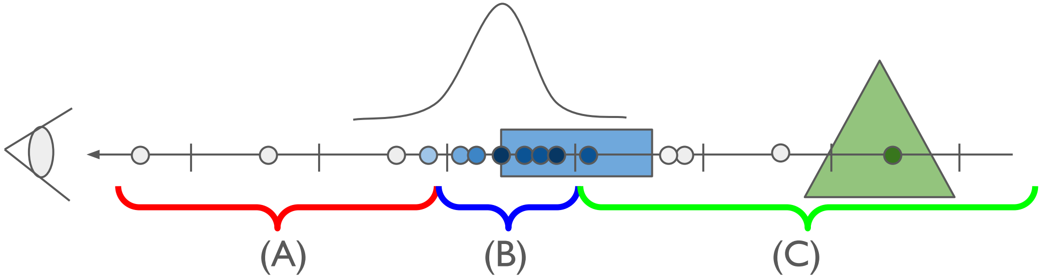
Tricks:

- **<u>Hierarchical Sampling:</u> coarse to fine importance sampling**
  - First sample coarsely along the ray with stratified sampling
    - Create Nc bins between tn and tf
    - For each bin, sample ti uniformly



<u>(Ray-) Volume rendering</u> $$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} \underbrace{T_i(1-\exp(-\sigma_i\delta_i))}_{\mathbf{w}_i}\mathbf{c}_i\,, \text{ where } T_i = \exp\left(-\sum_{j=1}^{i-1}\sigma_j\delta_j\right)$$

  - Then do importance sampling based on color weight $\mathbf{w}_i$



(A) $\boxed{T_i \approx 1,\ \ \sigma_i \approx 0 \quad \mathbf{w}_i \approx 0}$

(B) $\boxed{T_i > 0,\ \ \sigma_i > 0 \quad \mathbf{w}_i > 0}$

(C) $\boxed{T_i \approx 0 \quad \mathbf{w}_i \approx 0}$

Neural Radiance field (NeRF)

Tricks:

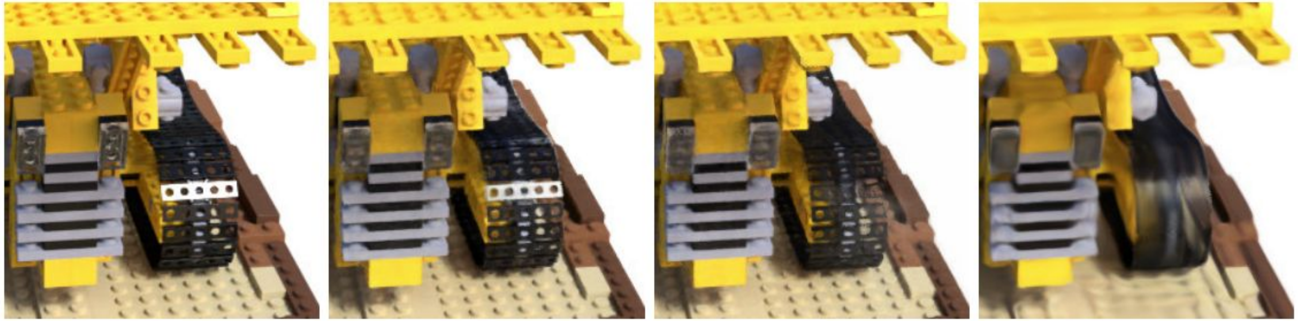
- **Positional encoding** to map each input 5D coordinate into a higher dimensional space
  - Learning in high-frequency mappings is difficult to learn
    $\gamma(p) = \left(\sin(2^0 \pi p), \cos(2^0 \pi p), \cdots, \sin(2^{L-1} \pi p), \cos(2^{L-1} \pi p)\right)$
  - Fourier Basis feature mapping allocates neurons to different spatial frequency bands (frequency disentangling)

(a) Coordinate-based MLP (b) Image regression $(x,y) \rightarrow$ RGB (c) 3D shape regression $(x,y,z) \rightarrow$ occupancy (d) MRI reconstruction $(x,y,z) \rightarrow$ density (e) Inverse rendering $(x,y,z) \rightarrow$ RGB, density

[8] Tancik20

Neural Radiance field (NeRF)

	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**



Ground Truth Complete Model No View Dependence No Positional Encoding

	Diffuse Synthetic 360° [41]			Realistic Synthetic 360°			Real Forward-Facing [28]		
Method	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

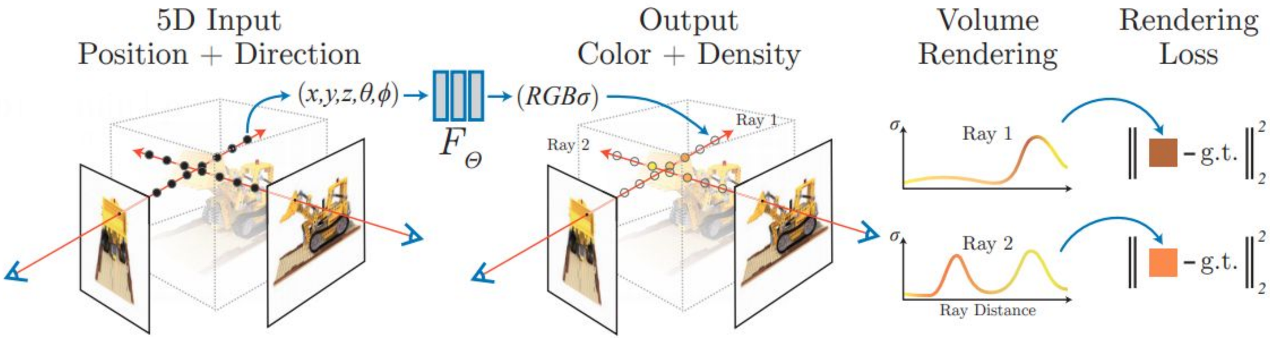
[7] Mildenhall20

Neural Radiance field (NeRF)

[7] Mildenhall20

Neural Radiance field (NeRF)

NeRF in a nutshell:



- Learn the radiance field of a scene based on a collection of calibrated images
  - Use an MLP to learn continuous geometry and view-dependent appearance
- Use fully differentiable volume rendering with reconstruction loss
- Combines <u>importance sampling</u> and <u>Fourier-basis encoding</u> of 5D query to produce **high-fidelity novel view synthesis results**
- Allows efficient storage of scenes (x3000 gain over voxelized representations)

Neural Radiance field (NeRF)

Remaining challenges

- Handling dynamic scenes when acquiring calibrated views
- One network trained per scene - no generalization

Neural Radiance field (NeRF)

Remaining challenges

- Handling dynamic scenes when acquiring calibrated views
  - D-NeRF: Neural Radiance Fields for Dynamic Scenes
  - Deformable Neural Radiance Fields
- One network trained per scene - no generalization

D-NeRF: Neural Radiance Fields for Dynamic Scenes

NeRF

- Only applicable to rigid scenes
- 5D continuous function
- Requiring multiple views of a rigid scene

D-NeRF

+ Applicable for rigid and non-rigid scenes
+ 6D continuous function by considering time-component as an additional input
+ Requiring a single view per time instant for non-rigid scenes.

[11] Pumarola20

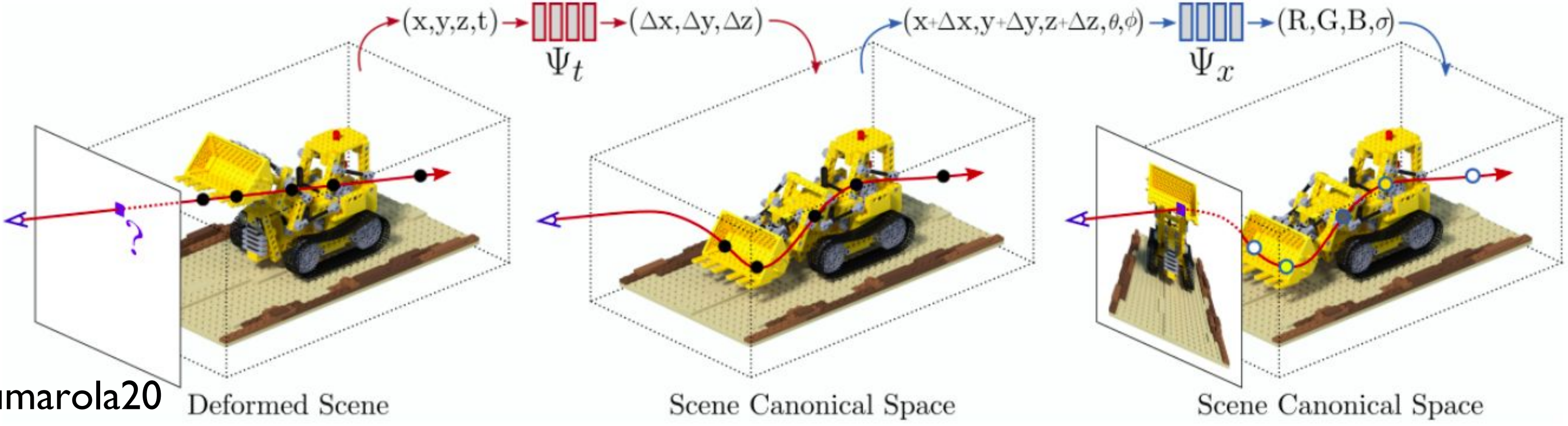
D-NeRF: Neural Radiance Fields for Dynamic Scenes

- **Deformation network** Ψ_t : to predict deformation field between the scene at time instant t and the scene in canonical space (t=0)

$$\Psi_t(\mathbf{x}, t) = \begin{cases} \Delta\mathbf{x}, & \text{if } t \neq 0 \\ 0, & \text{if } t = 0 \end{cases}$$

- **Canonical network** Ψ_x: to predict color and density in canonical configuration

$$\Psi_x(\mathbf{x}, \mathbf{d}) \mapsto (\mathbf{c}, \sigma)$$



[11] Pumarola20

D-NeRF: Neural Radiance Fields for Dynamic Scenes

Volumetric rendering is the same as NeRF in **canonical space**:

$$\boxed{\text{6D radiance field}} \xrightarrow{\textbf{Deformation network } \Psi_t} \boxed{\text{5D radiance field } \textit{(canonical space)}} \xrightarrow{\textbf{NeRF's rendering}} \boxed{\text{Rendering image}}$$

Opacity

Predicted colors

$$C(p,t) = \int_{h_n}^{h_f} \mathcal{T}(h,t)\sigma(\mathbf{p}(h,t))\mathbf{c}(\mathbf{p}(h,t),\mathbf{d})dh$$

Volume density

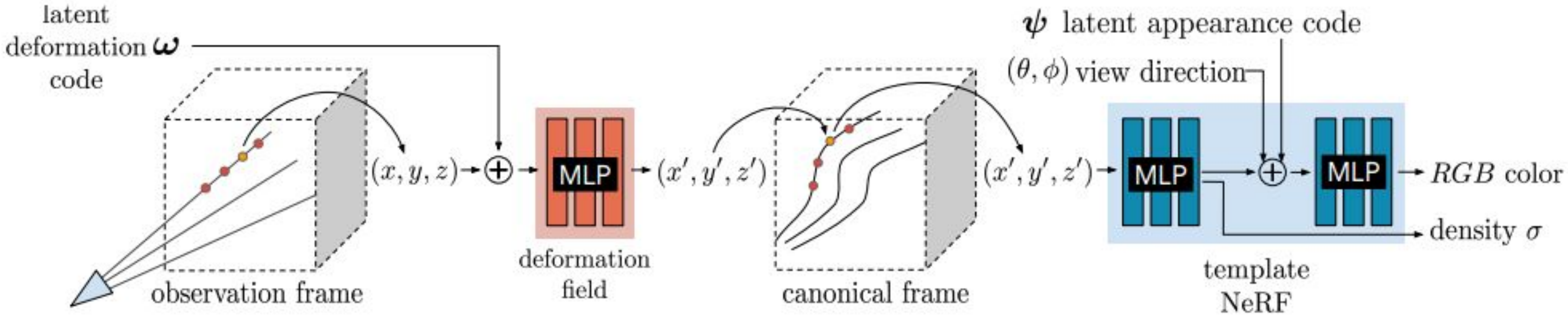
$$\text{where} \quad \mathbf{p}(h,t) = \mathbf{x}(h) + \Psi_t(\mathbf{x}(h),t),$$
$$[\mathbf{c}(\mathbf{p}(h,t),\mathbf{d}),\sigma(\mathbf{p}(h,t))] = \Psi_x(\mathbf{p}(h,t),\mathbf{d}),$$
$$\text{and} \quad \mathcal{T}(h,t) = \exp\left(-\int_{h_n}^{h}\sigma(\mathbf{p}(s,t))ds\right).$$

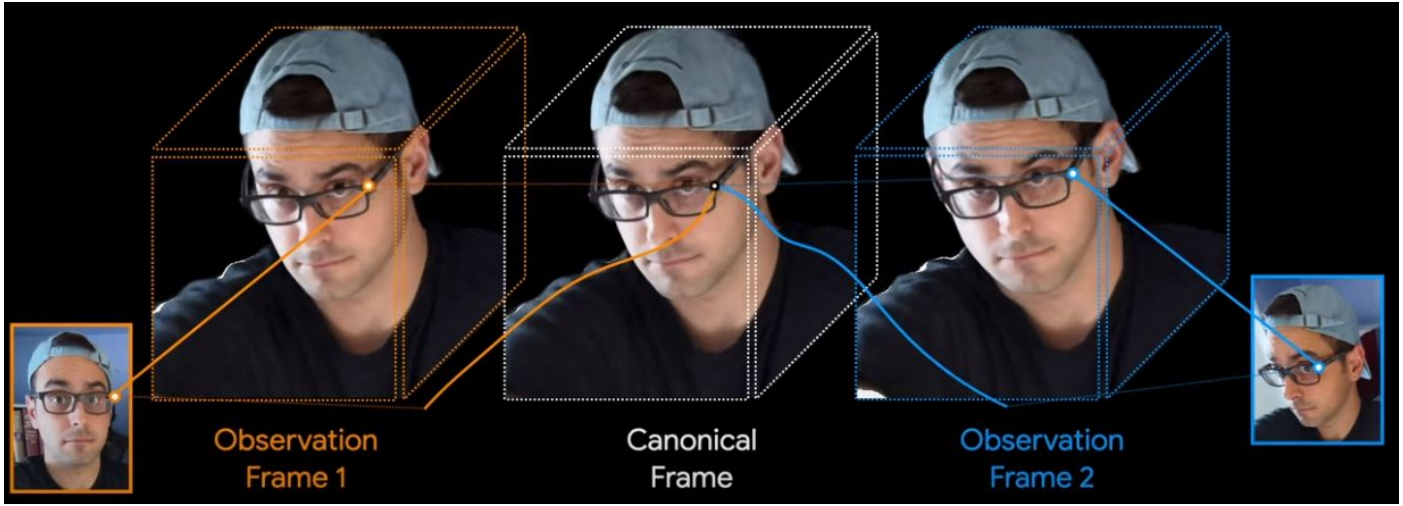
[11] Pumarola20

D-NeRF: Neural Radiance Fields for Dynamic Scenes

[11] Pumarola20

Deformable Neural Radiance Fields





[12] Park20

Deformable Neural Radiance Fields

[12] Park20

Deformable Neural Radiance Fields vs D-NeRF

Deformable Neural Radiance Fields

Submission history

From: Keunhong Park [view email]

[v1] Wed, 25 Nov 2020 18:55:04 UTC (47,887 KB)

[v2] Thu, 26 Nov 2020 01:52:45 UTC (47,887 KB)

We present the first method capable of photorealistically reconstructing a non-rigidly deforming scene using photos/videos captured casually from mobile phones. Our approach – D-NERF *– augments neural radiance fields (NeRF)*

+ Works on real data
- Relies on pretrained foreground dynamic object segmentation
+ Formulation of elastic deformation regularization
- Does not explore time dependency

D-NeRF

Submission history

From: Albert Pumarola [view email]

[v1] Fri, 27 Nov 2020 19:06:50 UTC (16,352 KB)

ages. In this paper we introduce D-NeRF, a method that extends neural radiance fields to a dynamic domain, allowing to reconstruct and render novel images of objects under rigid and non-rigid motions from a single camera moving around the scene. For this purpose we consider time as an

- Works on synthetic data
- Works on scenes with isolated object

+ Time as input

Neural Radiance field (NeRF)

Remaining challenges

- Handling dynamic scenes when acquiring calibrated views
- One network trained per scene - no generalization
  - PixelNeRF (CVPR'21 submission)
  - General radiance field (ICLR'21 submission)

PixelNeRF: Neural Radiance Fields from One or Few Images

NeRF

[7] Mildenhall20

- Optimizing NeRF of each scene independently
- Requiring many calibrated views (TODO put number estimate)
- Using canonical coordinate frame

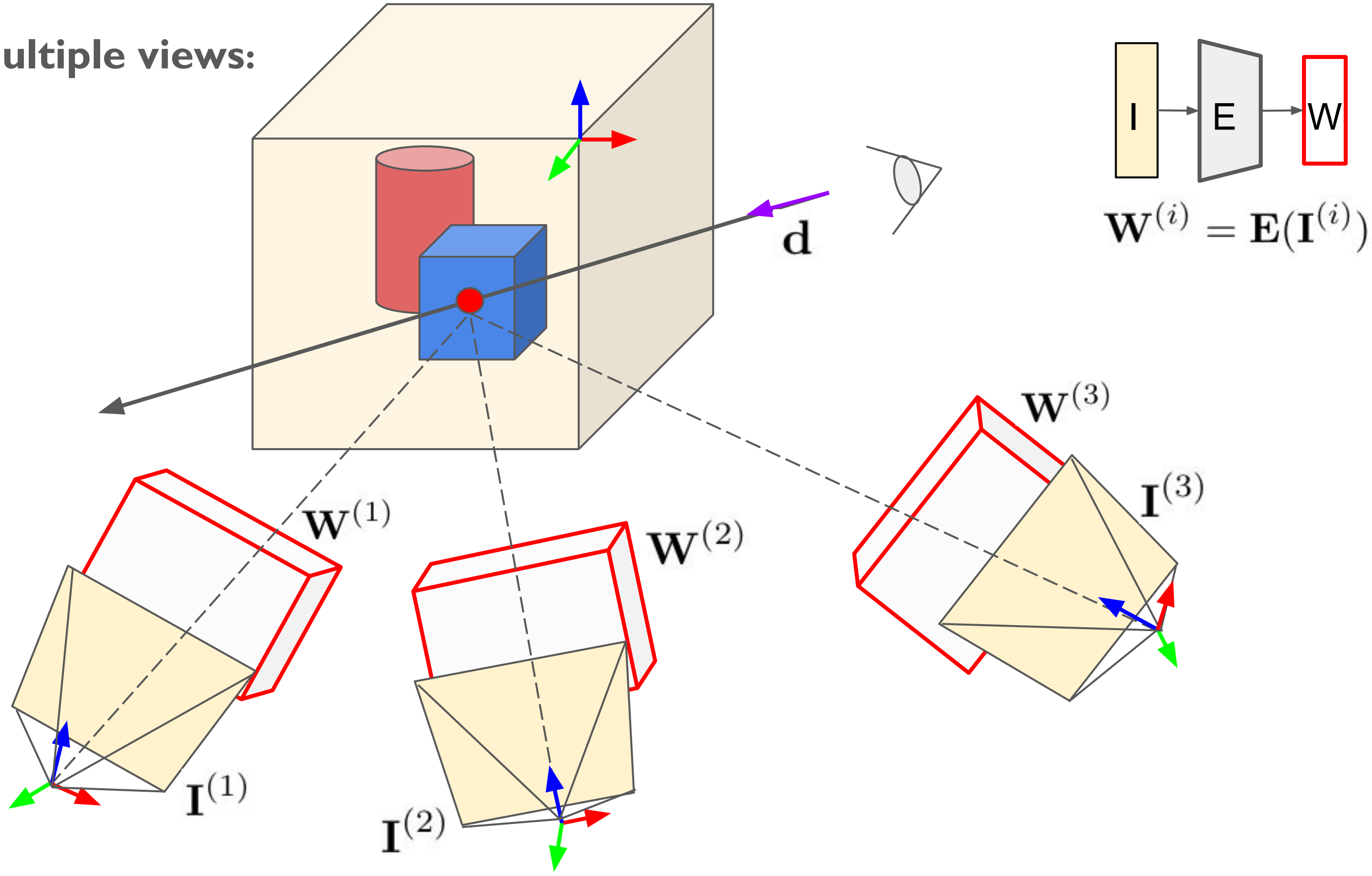
PixelNeRF

[9] Yu20

+ Training across multiple scenes to learn a (image(s) conditioned?) scene prior
+ Address few-shot view synthesis task with sparse set of views
+ Predicting a NeRF representation in the camera coordinate system
+ **Incorporate a variable number of posed input views**

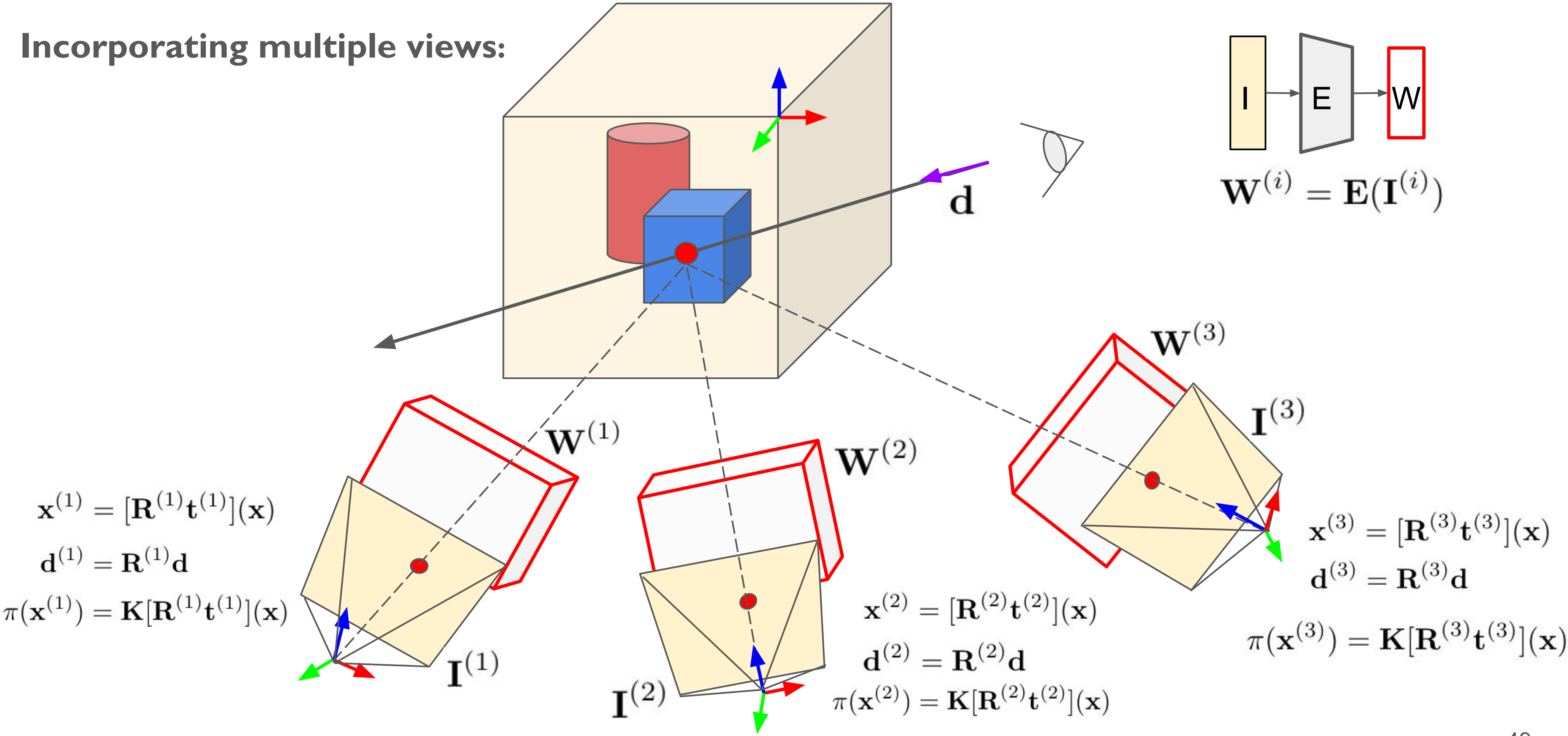
PixelNeRF: Neural Radiance Fields from One or Few Images

Incorporating multiple views:



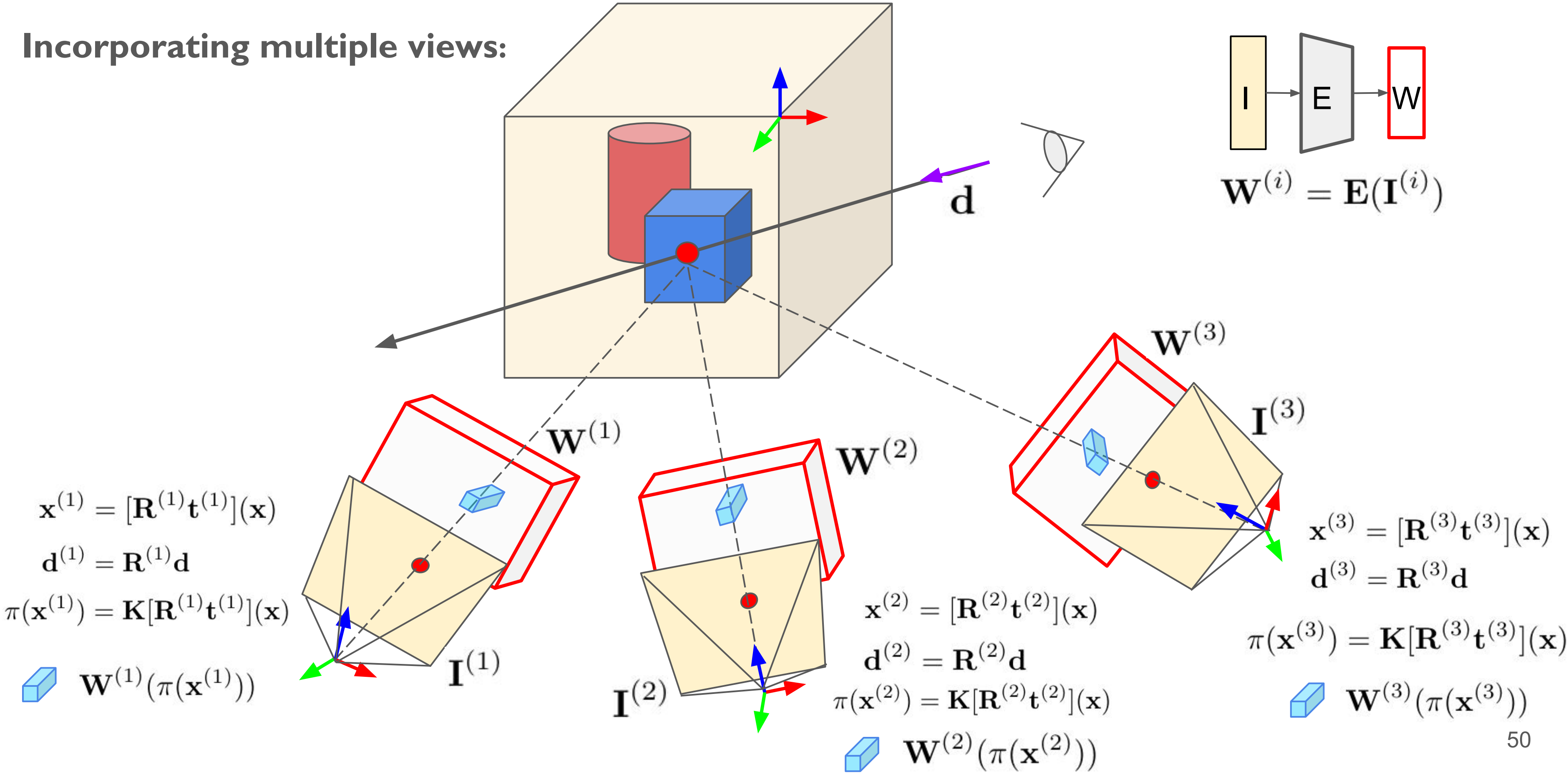
PixelNeRF: Neural Radiance Fields from One or Few Images

Incorporating multiple views:



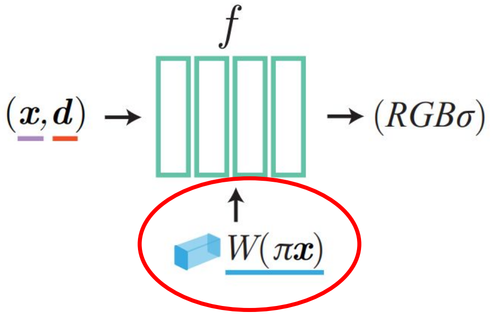
PixelNeRF: Neural Radiance Fields from One or Few Images

Incorporating multiple views:



PixelNeRF: Neural Radiance Fields from One or Few Images

Incorporating multiple views:



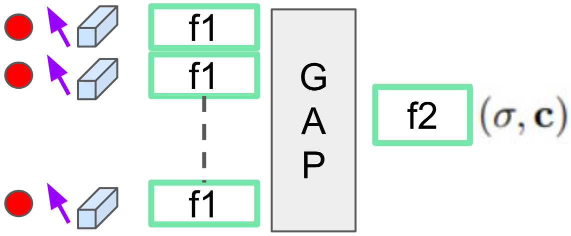
- First, transform 5D input into coordinate system of each view given camera transform

- Then, calculate intermediate feature vector for each view:

$$\mathbf{V}^{(i)} = f_1\left(\boxed{\gamma(\mathbf{x}^{(i)})}, \mathbf{d}^{(i)}; \mathbf{W}^{(i)}\left(\pi(\mathbf{x}^{(i)})\right)\right)$$

- Finally, aggregate with the average pooling operator ψ and passed into a the final layer

$$(\sigma, \mathbf{c}) = f_2\left(\psi\left(\mathbf{V}^{(1)}, \ldots, \mathbf{V}^{(n)}\right)\right)$$



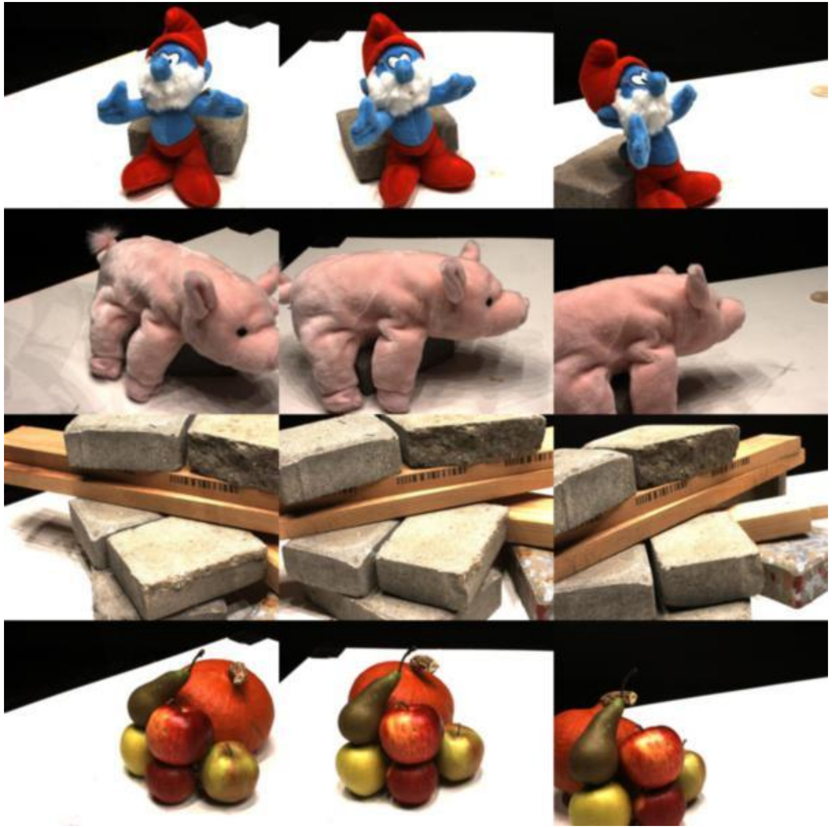
PixelNeRF: Neural Radiance Fields from One or Few Images

	1-view			2-view		
	↑ PSNR	↑ SSIM	↓ LPIPS	↑ PSNR	↑ SSIM	↓ LPIPS
– Local	20.39	0.848	0.196	21.17	0.865	0.175
– Dirs	21.93	0.885	0.139	23.50	0.909	0.121
Full	**23.43**	**0.911**	**0.104**	**25.95**	**0.939**	**0.071**

Table 3: **Ablation studies for ShapeNet chair reconstruction.** We show the benefit of using local features over a global code to condition the NeRF network (−Local vs Full), and of providing view directions to the network (−Dirs vs Full).

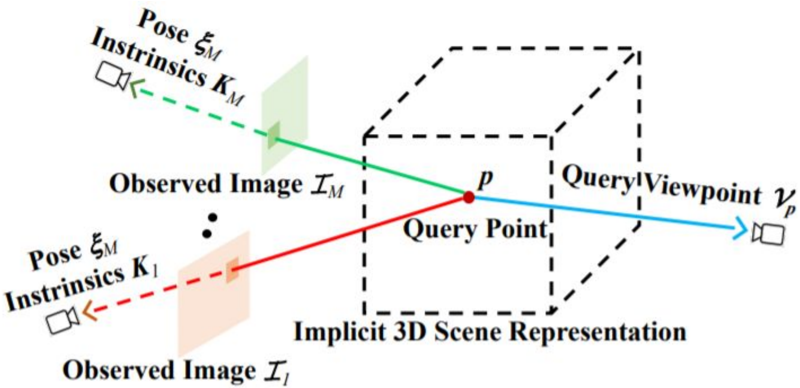
[9] Yu20

PixelNeRF: Neural Radiance Fields from One or Few Images



[9] Yu20

GRF: Learning a General Radiance Field for 3D Scene Representation and Rendering



GRF

[v1] Fri, 9 Oct 2020 14:21:43 UTC (7,696 KB)
[v2] Sun, 29 Nov 2020 06:33:25 UTC (25,183 KB)

ICLR21 submission

OpenReview grades: 7, 6, 5, 4

[10] Trevithick20

PixelNeRF

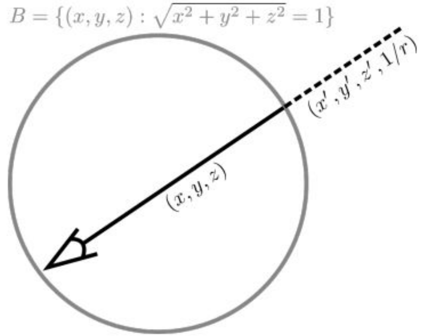
[v1] Thu, 3 Dec 2020 18:59:54 UTC (9,768 KB)

IEEE International Conference on Neural Radiance Fields (ICNeRF)

Related Work Lastly, note that concurrent work [42] adds image features to NeRF. A key difference is that we operate in view rather than canonical space, which makes our approach applicable in more general settings.

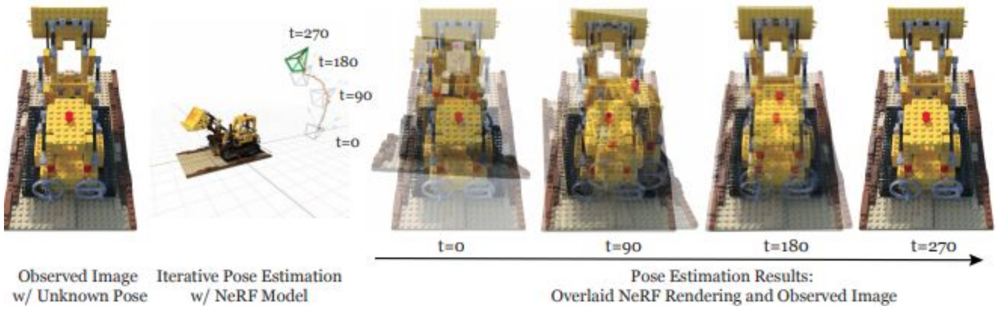
Moreover, we extensively demonstrate our method's performance in few-shot view synthesis, while GRF shows very limited quantitative results for this task.

More works on NeRF



- NeRF++: Analyzing and Improving Neural Radiance Fields [Zhang20]

- iNeRF: Inverting Neural Radiance Fields for Pose Estimation [Yen-Chen20]



- NeRF in the Wild [Ricardo20]...

References

[7] Mildenhall, Srinivasan, Tancik et al., NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis, ECCV 2020
[8] Tancik, Srinivasan, Mildenhall et al., Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains, NeurIPS 2020
[9] Yu et al., PixelNeRF: Neural Radiance Fields from One or Few Images, *Arxiv preprint* 2020
[10] Trevithick and Yang, GRF: Learning a General Radiance Field for 3D Scene Representation and Rendering, *Arxiv preprint 2020*
[11] Pumarola et al., D-NeRF: Neural Radiance Fields for Dynamic Scenes, *Arxiv preprint* 2020
[12] Park et al., Deformable Neural Radiance Fields, *Arxiv preprint* 2020

[13] Ricardo et al., NeRF in the Wild: Neural Radiance Fields for Unconstrained Photo Collections, *Arxiv preprint*
[14] Zhang et al., NeRF++: Analyzing and improving neural radiance fields, *Arxiv preprint*

[15] Yen-Chen et al., iNeRF: Inverting Neural Radiance Fields for Pose Estimation, *Arxiv preprint*

Matthew Tancik's 1h talk at Tübingen seminar of the Autonomous Vision Group

Awesome Neural Radiance Fields: https://github.com/yenchenlin/awesome-NeRF

NeRF papers with code: https://paperswithcode.com/method/nerf