Neural Implicit Representations

Dolev Ofri and Eyal Naor

June 2021
A Rapidly Growing Research Field
https://github.com/vsitzmann/awesome-implicit-representations
Outline

- Intro
- NeRF
- Fourier Feat.
- SIREN
- NeX

Explicit vs implicit

3D reconstruction examples
Explicit vs Implicit Representations
Explicit Representations

2D Representations

3D Representations

1D Representations

Continuous Functions

Audio Signals

Voxels
Points
Mesh
Implicit Representations

• Also called “coordinate-based representations”
Implicit Representations

• Also called “coordinate-based representations”
• Parametrize a signal as a continuous function
Implicit Representations

• Also called “coordinate-based representations”
• Parametrize a signal as a continuous function

\[(x, y) \quad \rightarrow \quad f \quad \rightarrow \quad \text{(0.913, 0.909)}\]
Implicit Representations

• Also called “coordinate-based representations”
• Parametrize a signal as a *continuous function*
• Exact mathematical function is unknown

\[ f = ? \]
Implicit Representations

- Also called “coordinate-based representations”
- Parametrize a signal as a continuous function
- **Neural** Implicit Representations: use a neural network!
Implicit Representations

Main advantages:
• Arbitrary resolution
• Memory efficient
Implicit Representations

Main advantages:
• Arbitrary resolution
• Memory efficient

Uses:
• Super resolution
• Geometry representation / 3D reconstruction
• ...
Implicit Representations

Main advantages:
• Arbitrary resolution
• Memory efficient

Uses:
• Super resolution
• Geometry representation / 3D reconstruction
• ...
Occupancy Networks

Learning 3D Reconstruction in Function Space

Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, Andreas Geiger

CVPR 2019

DeepSDF

Learning Continuous Signed Distance Functions for Shape Representation

Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, Steven Lovegrove

CVPR 2019
Occupancy Networks

- Decision boundary

DeepSDF

Learning Continuous Signed Distance Functions for Shape Representation

Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, Steven Lovegrove

CVPR 2019
Occupancy Networks

- Decision boundary

DeepSDF

- Signed Distance Function (SDF)
Occupancy Networks

- Decision boundary

DeepSDF

- Signed Distance Function (SDF)
**Occupancy Networks**

- **Decision boundary**

**DeepSDF**

- **Signed Distance Function (SDF)**

\[ f_\theta \]

- 3D location
- Occupancy Probability \([0,1]\)
Occupancy Networks

- Decision boundary

DeepSDF

- Signed Distance Function (SDF)
Occupancy Networks

Input

3D-R2N2

PSGN

Pix2Mesh

AtlasNet

Ours

Continuous

DeepSDF

(a) Ground-truth

(b) Our Result

(c) [22]-25 patch

(d) [22]-sphere

Continuous

AtlasNet
Scene Representation
Scene Representation

“Classic DL”: The Net == The Task
Single net, Single task
Scene Representation

“Classic DL”: The Net == The Task
Single net, Single task
Scene Representation

“Classic DL”: The Net == The Task
Single net, Single task

The network weights “hold” what’s needed for the task.
Scene Representation

“Classic DL”: The Net == The Task
Single net, Single task

The network weights “hold” what’s needed for the task.
Scene Representation

“Classic DL”: The Net == The Task
Single net, Single task

NeRF: The Net == The Scene
Single net, Single scene
NeRF
Representing Scenes as Neural Radiance Fields for View Synthesis
Ben Mildenhall, Pratul Srinivasan, Matt Tancik, Jon Barron, Ravi Ramamoorth, Ren Ng
ECCV 2020, Best Paper Honorable Mention

Slides on NeRF are based on Yoni Kansten’s slides
Task: Render New Views
Task: Render New Views
Task: Render New Views

Inputs: sparsely sampled images of scene

Output: includes new rendered views

matthewtancik.com/nerf
Inputs

Multiview Images of a single scene
Inputs

Multiview Images of a single scene
Inputs

Multiview Images of a single scene

Camera poses
Scene representation
Scene representation

\[(x, y, z, \theta, \phi)\]

Spatial location  Viewing direction

\[F_\Theta\]
Multi-Layered Perceptron (MLP)
9 layers
256 channels

Slide credit: Jon Barron’s talk
Scene representation

\[(x, y, z, \theta, \phi) \rightarrow F_\theta \rightarrow (r, g, b, \sigma)\]

- **Spatial location**
- **Viewing direction**

\[F_\theta\]

- Multi-Layered Perceptron (MLP)
  - 9 layers
  - 256 channels

- **Output color**
- **Output density**

Slide credit: Jon Barron’s talk
Scene representation

Input is only coordinates
No latent code

$$(x, y, z, \theta, \phi)$$

Spatial location  Viewing direction

$F_{\Theta}$

Multi-Layered Perceptron (MLP)
9 layers
256 channels

$$(r, g, b, \sigma)$$

Output color  Output density
Scene representation

\[(x, y, z)\] \quad \text{Spatial location vector}

\[\sigma \quad \text{Output density}\]

\[(\theta, \phi)\] \quad \text{Viewing Direction}

\[h \quad \text{Output color } c\]

\[\sigma \text{ (spatial location)}\]

\[c \text{ (spatial location, viewing direction)}\]
Scene representation

\[ (x, y, z) \rightarrow \sigma \rightarrow (r, g, b) \]

Spatial location vector

Output color \( c \)

Output density \( \sigma \)

Viewing direction

3D Cartesian unit vector

\( \sigma \) (spatial location)

\( c \) (spatial location, viewing direction)
Viewing Directions as Input

(a) View 1  (b) View 2  (c) Radiance Distributions
The ray hit something

\[ r(t) \] – camera ray \[ r(t) = o + td \]

\[ \sigma \] – volume density
Volume rendering

\[ \mathbf{r}(t) \text{ – camera ray } \mathbf{r}(t) = \mathbf{o} + td \]
\[ \sigma \text{ – volume density} \]
Volume rendering

\[ r(t) \quad \text{– camera ray} \quad r(t) = o + td \]

\[ \sigma \quad \text{– volume density} \]
Volume rendering

\[ C(r) = \sum_{i=1}^{N} T_i \alpha_i c_i \]
Volume rendering

\[ C(r) = \sum_{i=1}^{N} T_i \alpha_i c_i \]

Are you present?

\[ \alpha_i = 1 - e^{-\sigma_i \delta_i} \]

\[ \sigma \quad \text{– volume density} \]
Volume rendering

\[ C(r) = \sum_{i=1}^{N} T_i \alpha_i c_i \]

Are you visible? Are you present?

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

\[ \sigma \quad \text{volume density} \]
Volume rendering

\[ C(r) = \sum_{i=1}^{N} T_i \alpha_i c_i \]

Are you visible? Are you present? What is your color?

\[ \rho(t) \]

Camera Ray

\[ \sigma \quad – \text{volume density} \]
The Sampling Method

Uniform sampling with a **small** $N$

→ Low accuracy
The Sampling Method

Uniform sampling with a large $N$ → Inefficient
The Sampling Method

Non-uniform sampling
→ How/where?
Hierarchical Volume Rendering

Uniform samples

$(x, y, z, \theta, \phi) \rightarrow F_{\Theta c} \rightarrow \hat{C}_c, \sigma$
Hierarchical Volume Rendering

Uniform samples

\[(x,y,z,\theta,\phi) \rightarrow F_{\Theta c} \rightarrow \mathcal{C}_c, \sigma\]

Coarse NeRF

Non-uniform samples

\[(x,y,z,\theta,\phi) \rightarrow F_{\Theta f} \rightarrow \mathcal{C}_f, \sigma\]

Fine NeRF
Hierarchical Volume Rendering

Train two networks

\[ (x, y, z, \theta, \phi) \rightarrow F_{\Theta_c} \rightarrow \hat{C}_c, \sigma \]

Coarse NeRF

\[ (x, y, z, \theta, \phi) \rightarrow F_{\Theta_f} \rightarrow \hat{C}_f, \sigma \]

Fine NeRF

Loss = \[ \sum_{r \in \mathcal{R}} \left( \| \hat{C}_c(r) - C(r) \|_2^2 + \| \hat{C}_f(r) - C(r) \|_2^2 \right) \]
What else?

\[(x, y, z, d)\]  \(\rightarrow\)  \(F_\Theta\)  \(\rightarrow\)  \((c, \sigma)\)

- Spatial location
- Viewing direction
- Output color
- Output density
What else?

$\mathbf{(p, d)} \xrightarrow{F_\Theta} \mathbf{(c, \sigma)}$

- Spatial location
- Viewing direction
- Output color
- Output density
Positional encoding

\[ \gamma(p), \gamma(d) \rightarrow F_\Theta \rightarrow (c, \sigma) \]

Spatial location \quad \text{Viewing direction} \quad \text{Output color} \quad \text{Output density}

\[ \gamma(p) = (\sin(2^0 \pi p), \cos(2^0 \pi p), \ldots, \sin(2^{L-1} \pi p), \cos(2^{L-1} \pi p)) \]

* Vaswani et al. NeurIPS, 2017
Positional encoding – 1D

\[ \gamma(x = 0.125) = (0.383, 0.707, 1.0) \]

\[ \gamma(p) = (\sin(2^0 \pi p), \cos(2^0 \pi p), \ldots, \sin(2^{L-1} \pi p), \cos(2^{L-1} \pi p)) \]
Results

Synthetic Scenes
Results
Real Scenes
Results
Representation Benefits
Depth Maps

Rendered Camera Path

Expected Ray Termination Depth
Meshable
Ablation study

Ground Truth

Complete Model
Ablation study

Ground Truth

Complete Model

No View Dependence
Ablation study
NeRF: Summary
NeRF: Summary

\[(x, y, z, \theta, \phi) \rightarrow F_\Theta \rightarrow (r, g, b, \sigma)\]

MLP Architecture
Importance of Positional Encoding

NeRF
No positional encoding

NeRF
With positional encoding
Importance of Positional Encoding

- NeRF
  - No positional encoding
  - With positional encoding

NeX
Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains

Matthew Tancik, Pratul Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, Ren Ng

NeurIPS 2020
Problem Setting

A simpler example: representing a 2D image

\[ \mathbf{v} = (x, y) \]

Coordinate of a pixel in the image

\[ F_{\Theta} \]

MLP

\( (r, g, b) \)
Problem Setting

A simpler example: representing a 2D image

In NeRF: \( \gamma(v) = (\sin(2^0 \pi v), \cos(2^0 \pi v), \ldots, \sin(2^{L-1} \pi v), \cos(2^{L-1} \pi v)) \)
A simpler example: representing a 2D image

In NeRF: $\gamma(v) = (\sin(2^0 \pi v), \cos(2^0 \pi v), ..., \sin(2^{L-1} \pi v), \cos(2^{L-1} \pi v))$
Positional Encoding – With or Without?

Feeding a 2D image to a simple MLP doesn’t work

- Ground truth image
- Standard fully-connected net
- With Positional Encoding
• Theoretical:
  Input mapping using Fourier features works – why?

• + Experimental:
  Dive into different mappings and check what’s important
Theory:
Neural Tangent Kernel (NTK)

Defined architecture + training data

Jacot et al., NeurIPS 2018; Arora et al., ICML 2019; Basri et al. 2020; Du et al., ICML 2019; Lee et al., NeurIPS 2019 and more
Theory: Neural Tangent Kernel (NTK)

*Defined architecture + training data
*under certain conditions

Jacot et al., NeurIPS 2018; Arora et al., ICML 2019; Basri et al. 2020; Du et al., ICML 2019; Lee et al., NeurIPS 2019 and more
Theory: Neural Tangent Kernel (NTK)

*Defined architecture + training data
*under certain conditions

Jacot et al., NeurlPS 2018; Arora et al., ICML 2019; Basri et al. 2020; Du et al., ICML 2019; Lee et al., NeurlPS 2019 and more
Theory:
Neural Tangent Kernel (NTK)

*Defined architecture + training data
*under certain conditions

\[ K_{NTK} \]

\[ n \times n \]

Jacot et al., NeurIPS 2018; Arora et al., ICML 2019; Basri et al. 2020; Du et al., ICML 2019; Lee et al., NeurIPS 2019 and more
Theory:
Neural Tangent Kernel (NTK)

Used the NTK method to show:
• No input mapping $\rightarrow$ “spectral bias”
• Can overcome this bias using Fourier feature mapping

Jacot et al., NeurIPS 2018; Arora et al., ICML 2019; Basri et al. 2020; Du et al., ICML 2019; Lee et al., NeurIPS 2019 and more
Different Experiment Domains

- No Fourier features: \( \gamma(v) = v \)
  - Image regression \( (x, y) \rightarrow \text{RGB} \)
  - 3D shape regression \( (x, y, z) \rightarrow \text{occupancy} \)
  - MRI reconstruction \( (x, y, z) \rightarrow \text{density} \)

- With Fourier features: \( \gamma(v) = FF(v) \)
  - Inverse rendering \( (x, y, z) \rightarrow \text{RGB, density} \)
Input Mappings

Basic:

\[ \gamma (\mathbf{v}) = [\cos(2\pi \mathbf{v}), \sin(2\pi \mathbf{v})] \]
Input Mappings

Basic:

\[ \gamma(v) = [\cos(2\pi v), \sin(2\pi v)] \]

Positional Encoding:

\[ \gamma(v) = [..., a_j \cos(2\pi \sigma^{j/m} v), a_j \sin(2\pi \sigma^{j/m} v), ...], \quad j = 0, ..., m - 1 \]

\( m \) – number of frequencies
Input Mappings

Basic:

\[ \gamma(v) = [\cos(2\pi v), \sin(2\pi v)] \]

Positional Encoding:

\[ \gamma(v) = [..., a_j \cos(2\pi \sigma^{j/m} v), a_j \sin(2\pi \sigma^{j/m} v), ...], \ j = 0, ..., m - 1 \]

Gaussian Random Fourier Features (RFF)*:

\[ \gamma(v) = [\cos(2\pi Bv), \sin(2\pi Bv)], \quad B \sim N(0, \sigma^2), \quad B \in \mathbb{R}^{m \times d} \]

Input Mappings

Basic:

\[ \gamma (\mathbf{v}) = [\cos(2\pi \mathbf{v}), \sin(2\pi \mathbf{v})] \]

Positional Encoding:

\[ \gamma (\mathbf{v}) = [..., a_j \cos(2\pi \sigma^{j/m} \mathbf{v}), a_j \sin(2\pi \sigma^{j/m} \mathbf{v}), ...], \ j = 0, ..., m - 1 \]

Gaussian Random Fourier Features (RFF)*:

\[ \gamma (\mathbf{v}) = [\cos(2\pi \mathbf{Bv}), \sin(2\pi \mathbf{Bv})], \quad \mathbf{B} \sim \mathcal{N}(0, \sigma^2), \quad \mathbf{B} \in \mathbb{R}^{m \times d} \]

Input Mappings

Basic:

\[ \gamma(v) = [\cos(2\pi v), \sin(2\pi v)] \]

Positional Encoding:

\[ \gamma(v) = [..., a_j \cos(\frac{2\pi \sigma^j}{m} v), a_j \sin(\frac{2\pi \sigma^j}{m} v), ...], \ j = 0, ..., m - 1 \]

Gaussian Random Fourier Features (RFF)*:

\[ \gamma(v) = [\cos(2\pi Bv), \sin(2\pi Bv)], \quad B \sim \mathcal{N}(0, \sigma^2), \quad B \in \mathbb{R}^{m \times d} \]

Gaussian RFF: 1D experiment

\[ \gamma(v) = [\cos(2\pi Bv), \sin(2\pi Bv)], \quad B \sim N(0, \sigma^2), \quad B \in \mathbb{R}^{m \times d} \]
Distribution Types and Mapping Bandwidth

Gaussian RFF: 1D experiment

\[ \gamma(v) = [\cos(2\pi Bv), \sin(2\pi Bv)] , \quad B \sim N(0, \sigma^2), \quad B \in \mathbb{R}^{m \times d} \]

σ is task-specific
Which Mapping is Best Visually?
On/Off-Axis Frequencies

Positional Encoding:

\[(\sin(2\pi\sigma^m x), \sin(2\pi\sigma^m y))\]

Gaussian:

\[B \in \mathbb{R}^{m \times d}\]

\[\sin(2\pi(b_1 x + b_2 y))\]

Images credit: Michal Irani, Intro to Comp. Vision
PE vs Gaussian Comparison

Positional Encoding  Gaussian
PE vs Gaussian Comparison

Positional Encoding vs Gaussian
Try It Yourself!

**Underfitting**
- None
- PE $\sigma = 1$
- PE $\sigma = 70$
- PE $\sigma = 250$

**Overfitting**
- Basic
- Gauss $\sigma = 1$
- Gauss $\sigma = 10$
- Gauss $\sigma = 100$

Add to Your Code!

```python
fc = nn.Linear(input_dim, 256)
x = fc(x)
```
fc = nn.Linear(input_dim, 256)
B = SCALE * torch.randn(input_dim, NUM_FEATURES)
x = torch.cat([torch.sin((2. * math.pi * x) @ B), torch.cos((2. * math.pi * x) @ B)], dim=-1)
x = fc(x)
Summary

Input mapping helps the network learn fine details / high frequencies!

Standard fully-connected net  With Positional Encoding
Input mapping helps the network learn fine details / high frequencies!
Any Questions?
Welcome back
Implicit Neural Representations with Periodic Activation Functions


NeurIPS 2020
Implicit Neural Representations with Periodic Activation Functions, aka SIRENs - Sinusoidal Representation Networks


NeurIPS 2020
SIRENs - Sinusoidal REpresentation Networks

The gist: Neural Internal Representation with sinusoidal activation functions.
SIRENs - Sinusoidal REpresentation Networks

The gist: Neural Internal Representation with sinusoidal activation functions.

The interesting part: opens a door for new applications/implementations.
SIRENs - Motivation

Until now - the network is trained directly by the wanted function.

\[ \phi(x,y) ? \quad \phi(x,y) ! \]
SIRENs - Motivation

Until now - the network is trained directly by the wanted function.

\[ \phi(x,y) ? \xrightarrow{\phi} \phi(x,y)! \]
SIRENs - Motivation

Until now - the network is trained directly by the wanted function.

But for some tasks - the input’s derivatives are essential.
SIRENs - Motivation

Until now - the network is trained directly by the wanted function.

But for some tasks - the input’s derivatives are essential.

\[
\mathcal{L}_{(\phi,\nabla\phi)} = \|\phi(x) - f(x)\|^2 + \|\nabla\phi(x) - \nabla f(x)\|^2
\]
Until now - the network is trained directly by the wanted function.

But for some tasks - the input’s **derivatives** are essential.

\[ \mathcal{L}_{(\phi, \nabla \phi)} = \| \phi(x) - f(x) \|^2 + \| \nabla \phi(x) - \nabla f(x) \|^2 \]
SIRENs - Motivation

Until now - the network is trained directly by the wanted function.

But for some tasks - the input’s derivatives are essential.

\[
\mathcal{L}_{(\phi, \nabla \phi)} = \| \phi(x) - f(x) \|^2 + \| \nabla \phi(x) - \nabla f(x) \|^2
\]
SIRENs - Motivation

Until now - the network is trained directly by the wanted function.

But for some tasks - the input’s derivatives are essential.

\[
\frac{\partial^2 \Phi}{\partial t^2} - c^2 \Delta \Phi = 0
\]
SIRENs - Motivation

Until now - the network is trained directly by the wanted function.

But for some tasks - the input’s **derivatives** are essential.
SIRENs - Motivation

Until now - the network is trained directly by the wanted function.

But for some tasks - the input’s derivatives are essential.

\[
\frac{\partial^2 \Phi}{\partial t^2} - c^2 \Delta \Phi = 0
\]
Until now - the network is trained directly by the wanted function.

But for some tasks - the input’s derivatives are essential.

Are they also represented well?
SIRENs - Motivation

Until now - the network is trained directly by the wanted function.

But for some tasks - the input’s derivatives are essential.

Are they also represented well?

Obviously not.. So SIRENs will help!
SIRENs - Why do they work?
SIRENs - Why do they work?

The derivative of a SIREN is also a SIREN!

\[
\frac{\partial}{\partial x} \sin(x) = \cos(x) = \sin \left( x + \frac{\pi}{2} \right)
\]
SIRENs - Why do they work?

The derivative of a SIREN is also a SIREN!

\[ \frac{d}{dx} \sin(x) = \cos(x) = \sin\left(x + \frac{\pi}{2}\right) \]

Enables supervising complicated signals.
SIRENs - Why do they work?

The derivative of a SIREN is also a SIREN!

\[ \frac{\partial}{\partial x} \sin(x) = \cos(x) = \sin \left( x + \frac{\pi}{2} \right) \]

Enables supervising complicated signals.
SIRENs - Why do they work?

The derivative of a SIREN is also a SIREN!

\[
\frac{\partial}{\partial x} \sin(x) = \cos(x) = \sin\left(x + \frac{\pi}{2}\right)
\]

Enables supervising complicated signals.

\[
\nabla \phi(x, y) \quad \phi \quad \nabla \phi(x, y)!
\]

“well behaved”
SIRENs - Initialization is crucial
SIRENs - Initialization is crucial

Sinusoidal functions are not intuitively good activation functions
SIRENs - Initialization is crucial

Sinusoidal functions are not intuitively good activation functions

tanh  sigmoid  ReLU
SIRENs - Initialization is crucial

Sinusoidal functions are not intuitively good activation functions

tanh  sigmoid  ReLU  sin(x)
SIRENs - Initialization is crucial

Sinusoidal functions are not intuitively good activation functions

To “behave well” and enable deep MLPs, initialization is crucial:

Note that building SIRENS with not carefully chosen uniformly distributed weights yielded poor performance both in accuracy and in convergence speed.
SIRENs - Initialization is crucial

Initialization scheme + explanation
SIREN - Initialization is crucial

Many lemmas, bottom line:

Initializing all weights (except first layer) by uniform distribution in:

$$[-\sqrt{\frac{6}{\text{fan in}}}, \sqrt{\frac{6}{\text{fan in}}}]$$
SIRENs - Initialization is crucial

Many lemmas, bottom line:

Initializing all weights (except first layer) by uniform distribution in:

\[-\frac{6}{\sqrt{\text{fan in}'}}, \frac{6}{\sqrt{\text{fan in}}}]\]
Many lemmas, bottom line:

Initializing all weights (except first layer) by uniform distribution in: $[-\frac{6}{\sqrt{\text{fan in}'}}, \frac{6}{\sqrt{\text{fan in}'}}]$

They claim ("beyond the scope of this paper") - with this initialization - "the frequency throughout the sine network grows only slowly"
SIRENs - Initialization is crucial
SIRENs - Initialization is crucial
SIRENs - Results
SIRENs - Results

Directly on signal
- Images, Videos, Audio
SIRENs - Results

Directly on signal
- Images, Videos, Audio

Only on derivatives
- Poisson (I)
- Helmholtz (I and II)
SIRENs - Results

Directly on signal
- Images, Videos, Audio

Only on derivatives
- Poisson (I)
- Helmholtz (I and II)

Signal + derivatives
- SDF
SIRENs - Results

Directly on signal
- Images, Videos, Audio

Signal + derivatives
- SDF

Spatial & temporal derivatives
- The Wave eq.

Only on derivatives
- Poisson (I)
- Helmholtz (I and II)
SIREN - Results

Directly on signal
- Images, Videos, Audio

Only on derivatives
- Poisson (I)
- Helmholtz (I and II)

Spatial & temporal derivatives
- The Wave eq.

Can learn priors
- Inpainting: encoder → SIREN’s params
SIRENs - Directly on signal

\[ \bar{\mathcal{L}} = \sum_i \| \Phi(x_i) - f(x_i) \|^2 \]
SIRENs - Directly on signal

$$\tilde{L} = \sum_i \| \Phi(x_i) - f(x_i) \|^2$$

Images
SIREN\textsuperscript{a}s - Directly on signal

\[ \tilde{L} = \sum_{i} \| \Phi(x_i) - f(x_i) \|^2 \]
SIRENs - Directly on signal

\[ \tilde{\mathcal{L}} = \sum_{i} \| \Phi(x_i) - f(x_i) \|^2 \]
SIRENs - Directly on signal

\[ \tilde{L} = \sum_i \| \Phi(x_i) - f(x_i) \|^2 \]

The data

Audio

Results

Ground Truth

ReLU MLP

ReLU w/ positional encoding

SIREN
SIRENs - Directly on signal

\[ \tilde{\mathcal{L}} = \sum_i \| \Phi(x_i) - f(x_i) \|^2 \]

The data

Audio

Results

Representing Audio – Music

Ground Truth

ReLU MLP
ReLU w/ positional encoding
SIREN
SIRENs - Signal + derivatives
SIRENs - Signal + derivatives

Signed Distance Function (SDF):
SIRENs - Signal + derivatives

Signed Distance Function (SDF):

$$\mathcal{L}_{sdf} = \int_{\Omega} \| \nabla_x \Phi(x) \| dx + \int_{\Omega_0} \| \Phi(x) \| + (1 - \langle \nabla_x \Phi(x), n(x) \rangle) dx + \int_{\Omega \setminus \Omega_0} \psi(\Phi(x)) dx$$
Signed Distance Function (SDF):

\[
\mathcal{L}_{\text{sdf}} = \int_{\Omega} \| \nabla_x \Phi(x) \| - 1 \, dx + \int_{\Omega_0} \| \Phi(x) \| + \left( 1 - \langle \nabla_x \Phi(x), n(x) \rangle \right) \, dx + \int_{\Omega \setminus \Omega_0} \psi(\Phi(x)) \, dx
\]
SIRENs - Signal + derivatives

Signed Distance Function (SDF):

$\mathcal{L}_{sdf} = \int_{\Omega} \left\| \nabla_x \Phi(x) \right\| - 1 \, dx + \int_{\Omega_0} \left\| \Phi(x) \right\| + (1 - \left\langle \nabla_x \Phi(x), n(x) \right\rangle) \, dx + \int_{\Omega \setminus \Omega_0} \psi(\Phi(x)) \, dx$

- Everywhere
- On border
- Not on border

SDF $\rightarrow 0$

Penalty on small SDF

$\psi(x) = \exp(-\alpha \cdot \left\| \Phi(x) \right\|)$

$\alpha \gg 1$
SIRENs - Signal + derivatives

Signed Distance Function (SDF):

Everywhere
\[ \mathcal{L}_{sdf} = \int_{\Omega} \| \nabla_x \Phi(x) \| - 1 \, dx + \int_{\Omega_0} \| \Phi(x) \| + \left( 1 - \langle \nabla_x \Phi(x), n(x) \rangle \right) \, dx + \int_{\Omega \setminus \Omega_0} \psi(\Phi(x)) \, dx \]

On border
|\nabla x \Phi(x)| \rightarrow 1

Not on border
SDF \rightarrow 0

Penalty on small SDF
\[ \psi(x) = \exp(-\alpha \cdot |\Phi(x)|) \]
\[ \alpha \gg 1 \]
SIRENs - Signal + derivatives

Signed Distance Function (SDF):

\[ \mathcal{L}_{\text{sdf}} = \int_{\Omega} \| |\nabla \Phi(x)| - 1 \| \, dx + \int_{\Omega_0} \| \Phi(x) \| + \left(1 - \langle \nabla \Phi(x), n(x) \rangle \right) \, dx + \int_{\Omega \setminus \Omega_0} \psi(\Phi(x)) \, dx \]

- **Everywhere**
  - \(|\nabla| \rightarrow 1|

- **On border**
  - \(\text{SDF} \rightarrow 0\)

- **Not on border**
  - \(\text{grad} \parallel \text{normal}\)
  - \(\psi(x) = \exp(-\alpha \cdot |\Phi(x)|)\)
  - \(\alpha \gg 1\)

Penalty on small SDF
\[ \mathcal{L}_{sdf} = \int_\Omega \| |\nabla_x \Phi(x)| - 1\| \, dx + \int_{\Omega_0} \|\Phi(x)\| + (1 - \langle \nabla_x \Phi(x), n(x) \rangle) \, dx + \int_{\Omega \setminus \Omega_0} \psi(\Phi(x)) \, dx \]

SIRENs - Signal + derivatives

3D Shapes - solving the Eikonal equation

ReLU

SIREN

5 layers, 256 hidden units
\[ L_{sdf} = \int_{\Omega} \| \nabla_x \Phi(x) \| - 1 \| dx + \int_{\Omega_0} \| \Phi(x) \| + (1 - \langle \nabla_x \Phi(x), n(x) \rangle) dx + \int_{\Omega \setminus \Omega_0} \psi(\Phi(x)) dx \]

SIRENs - Signal + derivatives

ReLU PE (baseline)

SIREN (ours)
\[ L_{sdf} = \int_{\Omega} \left\| \nabla_x \Phi(x) \right\| - 1 \, dx + \int_{\Omega_0} \| \Phi(x) \| + (1 - \langle \nabla_x \Phi(x), n(x) \rangle) \, dx + \int_{\Omega \setminus \Omega_0} \psi(\Phi(x)) \, dx \]

SIRENs - Signal + derivatives
SIRENs - The wave equation
SIRENs - The wave equation

The system:

\[ \frac{\partial^2 \Phi}{\partial t^2} - c^2 \Delta \Phi = 0 \]

Input: \((t, x, y)\)
The system:

\[ \frac{\partial^2 \Phi}{\partial t^2} - c^2 \Delta \Phi = 0 \]

Input: \((t, x, y)\)

Initial conditions:

\[ \frac{\partial \Phi(0, x)}{\partial t} = 0 \]
\[ \Phi(0, x) = f(x) \]
The system:

\[ \frac{\partial^2 \Phi}{\partial t^2} - c^2 \Delta \Phi = 0 \]

Input: \((t, x, y)\)

Initial conditions:

\[ \frac{\partial \Phi(0, x)}{\partial t} = 0 \]
\[ \Phi(0, x) = f(x) \]

How to enforce?
The system:

\[ \frac{\partial^2 \Phi}{\partial t^2} - c^2 \Delta \Phi = 0 \]

Input: \((t, x, y)\)

Initial conditions:

\[ \frac{\partial \Phi(0, x)}{\partial t} = 0 \]

\[ \Phi(0, x) = f(x) \]

How to enforce? Inside the loss!

\[ L_{wave} = \int_{\Omega} \left\| \frac{\partial^2 \Phi}{\partial t^2} - c^2 \Delta \Phi \right\|_1 \]
SIREN - The wave equation

The system:

\[ \frac{\partial^2 \Phi}{\partial t^2} - c^2 \Delta \Phi = 0 \]

Input: \((t, x, y)\)

Initial conditions:

\[ \frac{\partial \Phi(0, x)}{\partial t} = 0 \]
\[ \Phi(0, x) = f(x) \]

How to enforce? Inside the loss!

\[ L_{\text{wave}} = \int_{\Omega} \left\| \frac{\partial^2 \Phi}{\partial t^2} - c^2 \Delta \Phi \right\|_1 + \lambda_1(x) \left\| \frac{\partial \Phi}{\partial t} \right\|_1 + \lambda_2(x) \| \Phi - f(x) \| dxdt \]

\(\lambda \neq 0\) only when \(t=0\)
SIRENs - The wave equation
SIRENs - Summary

Simple gist
SIRENs - Summary

Simple gist

Impressive application potential
SIRENs - Questions?
A Rapidly Growing Research Field

**iNeRF: Inverting Neural Radiance Fields**

**NeRF++: Analyzing and Improving Neural Radiance Fields**

**NeX: Real-time View Synthesis with Neural Basis Expansion**

Suttisak Wizadwongsa*, Pakkapon Phongthawee*, Jiraphon Yenphraphai*
Supasorn Suwajanakorn
VISTEC, Thailand
{suttisak.w.s19, pakkapon.p.s19, jiraphony.pro, supasorn.s}@vistec.ac.th

**pixelNeRF: Neural Radiance Fields**

Alex Yu


https://github.com/yenchenlin/awesome-NeRF
NeX: Real-time View Synthesis with Neural Basis Expansion

Suttisak Wizadwongsa, Pakkapon Phongthawee, Jiraphon Yenphraphai, Supasorn Suwajanakorn

CVPR 2021
NeX - Real-time View Synthesis with Neural Basis Expansion
NeX - Contributions
NeX - Contributions

1. Real time rendering (new view synthesis)
NeX - Contributions

1. Real time rendering

On same NVIDIA RTX 2080Ti:
300 fps VS NeRF: 0.018 (55 spf)
NeX - Contributions

1. Real time rendering

On same NVIDIA RTX 2080Ti:
300 fps VS NeRF: 0.018 (55 spf)

PC with Nvidia GeForce GTX 1650
NeX - Contributions

1. Real time rendering

2. Better results on reflections/refractions (+ “Shiny” dataset)
NeX - Contributions

1. Real time rendering

2. Better results on reflections/refractions (+ “Shiny” dataset)
NeX - Contributions

1. Real time rendering

2. Better results on reflections/refractions (+ “Shiny” dataset)

3. Representation method: Implicit/Explicit & Learned Basis
NeX - Implementation
NeX - Implementation

Use Multi-Plane Image (MPI)

NeX - Implementation

Use Multi-Plane Image (MPI)

For new angle: Homography
NeX - Implementation

Use Multi-Plane Image (MPI)

For new angle: Homography

Downside:

Only front facing scenes
NeX - Implementation

Use Multi-Plane Image (MPI)

For new angle: Homography

Downside:

Only front facing scenes

When too far:
NeX - Color representation
NeX - Color representation

Each pixel’s RGB is “broken down”:
NeX - Color representation

Each pixel’s RGB is “broken down”:

\[ k_0 \]
NeX - Color representation

Each pixel’s RGB is “broken down”:

\[ k_0 \]

- View independent
- Explicitly
NeX - Color representation

Each pixel’s RGB is “broken down”:

\[ k_0 + k_1 H_1 + k_2 H_2 + \cdots + k_N H_N \]

View independent
Explicitly
NeX - Color representation

Each pixel’s RGB is “broken down”:

$$k_0 + k_1 H_1 + k_2 H_2 + \cdots + k_N H_N$$

View independent
Explicitly
NeX - Color representation

Each pixel’s RGB is “broken down”:

\[ \begin{align*}
    k_0 + k_1 H_1 + k_2 H_2 + \cdots + k_N H_N
\end{align*} \]

View independent
Explicitly
Each pixel’s RGB is “broken down”:

\[ \begin{align*}
    & \= k_0 + k_1 \cdot H_1 + k_2 \cdot H_2 + \ldots + k_N \cdot H_N \\
    \text{View independent} & \quad \text{View dependent} \\
    \text{Explicitly} & \quad \text{Implicitly}
\end{align*} \]
NeX - Color representation

Each pixel’s RGB is “broken down”:

\[
C = K_0 + \overrightarrow{K} \cdot \overrightarrow{H}_\phi(V_i)
\]
NeX - Color representation - Questions?

\[ C = K_0 + \overrightarrow{K} \cdot \overrightarrow{H}_\phi(V_i) \]

Explicitly Learned

Implicitly Represented
NeX - Implicit/Explicit
NeX - Implicit/Explicit

In NeRF: entire scene represented implicitly in the MLP.
NeX - Implicit/Explicit

In NeRF: entire scene represented implicitly in the MLP.

In NeX: First order found \textit{explicitly} by minimizing TV.

\[ C = K_0 + \vec{K} \cdot \vec{H}_\phi(V_i) \]

Explicitly Learned

Implicitly Represented
NeX - Implicit/Explicit

In NeRF: entire scene represented implicitly in the MLP.

In NeX: First order found explicitly by minimizing TV.

\[ C = K_0 + \mathbf{K} \cdot \mathcal{H}_\phi(\mathcal{V}_i) \]

“... helps ease the network’s burden ... and leads to sharper results”
NeX - Implicit/Explicit

In NeRF: entire scene represented implicitly in the MLP.

In NeX: First order found explicitly by minimizing TV.

\[ C = K_0 + \vec{K} \cdot \vec{H}_\phi(V_i) \]

"... helps ease the network's burden ... and leads to sharper results"

(Reminds me of external+internal learning)
NeX - Learning the basis Functions

\[ C = K_0 + \vec{K} \cdot \overline{H}_\phi(V_i) \]
Why learn the basis functions?

\[ C = K_0 + \vec{K} \cdot \vec{H}_\phi(V_i) \]
Why learn the basis functions?

$$C = K_0 + \vec{K} \cdot \overrightarrow{H}_\phi(V_i)$$

- Spherical harmonics
- Hemispherical harmonics
- Fourier
NeX - Learning the basis Functions

Why learn the basis functions?

NeX - Learning the basis Functions

Why learn the basis functions?

1. Better results. **Higher frequencies** with same rank order.

\[ C = K_0 + \vec{K} \cdot \vec{H}_\phi(V_i) \]
NeX - Learning the basis Functions

Why learn the basis functions?

1. Better results. **Higher frequencies** with same rank order.

2. Since global - incorporates Image Prior.

\[ C = K_0 + \vec{K} \cdot \vec{H}_\phi(V_i) \]

Spherical harmonics  
Hemispherical harmonics  
Fourier
NeX - Learning the basis Functions

Why learn the basis functions?

1. Better results. **Higher frequencies** with same rank order.
2. Since global - incorporates Image Prior.

\[ C = K_0 + \vec{K} \cdot \overrightarrow{H}_\phi(V_i) \]
NeX - Learning the basis Functions

Why learn the basis functions?

\[ C = K_0 + \vec{K} \cdot \vec{H}_\phi(V_i) \]

1. Better results. **Higher frequencies** with same rank order.
2. Since global - incorporates Image Prior.

Less is more. Too many basis vectors \( \rightarrow \) overfit
NeX - Real Time Rendering
NeX - Real Time Rendering

Why is NeRF rendering so slow?
Why is NeRF rendering so slow?

For each new view synthesis:
NeX - Real Time Rendering

Why is NeRF rendering so slow?

For each new view synthesis:

   For each pixel:
NeX - Real Time Rendering

Why is NeRF rendering so slow?

For each new view synthesis:

For each pixel:

Multiple forward passes on coarse → Where to look
Why is NeRF rendering so slow?

For each new view synthesis:

For each pixel:

- **Multiple** forward passes on coarse → Where to look
- **Multiple** forward passes on fine → color & density
NeX - Real Time Rendering

Why is NeX faster?
NeX - Real Time Rendering

Why is NeX faster?

They split \((x, y, d)\) from viewing angle
NeX - Real Time Rendering

Why is NeX faster?

They split $\langle x, y, d \rangle$ from viewing angle

$$C = K_0 + \vec{K} \cdot \vec{H}_\phi(V_i)$$
NeX - Real Time Rendering

Why is NeX faster?

They split \((x, y, d)\) from viewing angle

\[ C = K_0 + \vec{K} \cdot \vec{H}_\phi(V_i) \]

1. One-time run for each pixel \(\rightarrow\) magnitudes in an unknown basis
NeX - Real Time Rendering

Why is NeX faster?

They split \((x, y, d)\) from viewing angle

\[
C = K_0 + \sum_{i} K \cdot H_{\phi}(V_i)
\]

1. One-time run for each pixel \(\rightarrow\) magnitudes in an unknown basis
NeX - Real Time Rendering

Why is NeX faster?

They split \((x,y,d)\) from viewing angle

\[
C = K_0 + \mathbf{K} \cdot \vec{H}_\phi(V_i)
\]

1. One-time run for each pixel \(\rightarrow\) magnitudes in an unknown basis

\[
\begin{array}{c}
k_0 \\
k_1 \\
k_2 \\
\vdots \\
k_N
\end{array} = \begin{array}{c}
\text{?} \\
\text{?} \\
\text{?} \\
\vdots \\
\text{?}
\end{array}
\]
NeX - Real Time Rendering

Why is NeX faster?

They split \((x, y, d)\) from viewing angle

\[
C = K_0 + \vec{K} \cdot \vec{H}_\phi(V_i)
\]

1. One-time run for each pixel \(\rightarrow\) magnitudes in an unknown basis

2. In test time - single forward pass: viewing angle \(\rightarrow\) basis vectors.
NeX - Real Time Rendering

Why is NeX faster?

They split \((x, y, d)\) from viewing angle

\[
C = K_0 + \mathbf{K} \cdot \mathbf{H}_\phi(V_i)
\]

1. One-time run for each pixel \(\rightarrow\) magnitudes in an unknown basis

2. In test time - single forward pass: viewing angle \(\rightarrow\) basis vectors.

\[
\begin{align*}
\text{Intro} & & \text{NeRF} & & \text{Fourier Feat.} & & \text{SIREN} & & \text{NeX}
\end{align*}
\]
NeX - Real Time Rendering

Why is NeX faster?

They split \((x,y,d)\) from viewing angle

\[
C = K_0 + \sum \vec{K} \cdot \vec{H}_\phi(V_i)
\]

1. One-time run for each pixel \(\rightarrow\) magnitudes in an unknown basis

2. In test time - single forward pass: viewing angle \(\rightarrow\) basis vectors.
NeX - Real Time Rendering

Why is NeX faster?

They split \((x,y,d)\) from viewing angle

1. One-time run for each pixel $\rightarrow$ magnitudes in an unknown basis

2. In test time - single forward pass: viewing angle $\rightarrow$ basis vectors.
NeX - Real Time Rendering

Why is NeX faster?

They split \((x,y,d)\) from viewing angle

1. One-time run for each pixel $\rightarrow$ magnitudes in an unknown basis

2. In test time - single forward pass: viewing angle $\rightarrow$ basis vectors.
NeX - Real Time Rendering

Why is NeX faster?

They split \((x,y,d)\) from viewing angle

1. **One-time run for each pixel** → magnitudes in an unknown basis
2. **In test time - single forward pass**: viewing angle → basis vectors.
NeX - Real Time Rendering

Why is NeX faster?

They split \((x, y, d)\) from viewing angle

1. One-time run for each pixel $\rightarrow$ magnitudes in an unknown basis

2. **In test time - single forward pass:** viewing angle $\rightarrow$ basis vectors.
NeX - Real Time Rendering

Why is NeX faster?

They split \((x, y, d)\) from viewing angle

1. One-time run for each pixel \(\rightarrow\) magnitudes in an unknown basis
2. **In test time - single forward pass:** viewing angle \(\rightarrow\) basis vectors.
NeX - Short-term Nostalgia

Throwback:
NeX - Short-term Nostalgia

Throwback:

1. They use positional encoding (for both spatial coordinates and angles)
NeX - Short-term Nostalgia

Throwback:

1. They use positional encoding (for both spatial coordinates and angles)

2. They use gradients in their loss.

\[
L_{\text{rec}}(\hat{I}_i, I_i) = \| \hat{I}_i - I_i \|^2 + \omega \| \nabla \hat{I}_i - \nabla I_i \|_1
\]
NeX - Short-term Nostalgia

Throwback:

1. They use positional encoding (for both spatial coordinates and angles)

2. They use gradients in their loss. Perhaps SIRENs would help?

\[ L_{\text{rec}}(\hat{I}_i, I_i) = \|\hat{I}_i - I_i\|^2 + \omega \| \nabla \hat{I}_i - \nabla I_i \|_1 \]
NeX - (Our) disclaimers
NeX - (Our) disclaimers

A lot of hypertuning took place:

- $\alpha$ uses a sigmoid activation, and the others use tanh activations.
- Positional Encoding: $(x,y) \rightarrow 20$ dims, $d \rightarrow 16$, angle $\rightarrow 12$
- Scan for optimal number of basis functions
- To be lighter: Multiple planes (4) share color, differ in density
A lot of hypertuning took place:

- $\alpha$ uses a sigmoid activation, and the others use tanh activations.
- Positional Encoding: $(x,y) \rightarrow 20$ dims, $d \rightarrow 16$, angle $\rightarrow 12$
- Scan for optimal number of basis functions
- To be lighter: Multiple planes (4) share color, differ in density

Improvement from there? Or “deeper”?
NeX - (Our) disclaimers

Fishy comparisons:

1. NeRF is 360°, they are front-facing
NeX - (Our) disclaimers

Fishy comparisons:

1. NeRF is 360°, they are front-facing
2. One of comparisons w.o. NeRF:

<table>
<thead>
<tr>
<th>Method</th>
<th>PSNR↑</th>
<th>SSIM↑</th>
<th>LPIPS↓</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRN [34]</td>
<td>21.82</td>
<td>0.744</td>
<td>0.464</td>
</tr>
<tr>
<td>LLFF [21]</td>
<td>24.41</td>
<td>0.863</td>
<td>0.211</td>
</tr>
<tr>
<td>NeRF [22]</td>
<td>26.76</td>
<td>0.883</td>
<td>0.246</td>
</tr>
<tr>
<td>NeX (Ours)</td>
<td>27.26</td>
<td>0.904</td>
<td>0.178</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>PSNR↑</th>
<th>SSIM↑</th>
<th>LPIPS↓</th>
</tr>
</thead>
<tbody>
<tr>
<td>Soft3D [24]</td>
<td>31.57</td>
<td>0.964</td>
<td>0.126</td>
</tr>
<tr>
<td>Deepview [6]</td>
<td>31.60</td>
<td>0.978</td>
<td>0.085</td>
</tr>
<tr>
<td>NeX (Ours)</td>
<td>35.84</td>
<td>0.985</td>
<td>0.083</td>
</tr>
</tbody>
</table>

Table 1: Average scores across 8 scenes in Real Forward-Facing dataset.

Table 2: Average scores across 8 scenes in Shiny dataset.
NeX - Summary
NeX - Summary

Realtime new view synthesis.
NeX - Summary

Realtime new view synthesis.

Do so with “a step back” after NeRF
NeX - Summary

Realtime new view synthesis.

Do so with “a step back” after NeRF:

1. Some return to global
2. Some return to explicit representation

\[
C = \begin{bmatrix} K_0 \end{bmatrix} + \vec{K} \cdot \vec{H}_\phi(V_i)
\]
NeX - Questions?
What did we see today?

Neural Implicit Representation – Representing data implicitly inside a NN
What did we see today?

**3D reconstruction**: Implicit representation of functions

### Occupancy Networks
- Decision boundary

### DeepSDF
- Signed Distance Function (SDF)
What did we see today?

**3D reconstruction**: Implicit representation of a function

**NeRF**: Implicit representation of a scene
What did we see today?

**3D reconstruction**: Implicit representation of a function

**NeRF**: Implicit representation of a scene

Positional Encoding $\rightarrow$ Fourier Features

- General PE
- Gauss $\sigma = 1$
- Gauss $\sigma = 10$
- Gauss $\sigma = 100$
What did we see today?

**3D reconstruction**: Implicit representation of a function

**NeRF**: Implicit representation of a scene

  Positional Encoding $\rightarrow$ Fourier Features

**SIRENs**: NIR with sine activations $\rightarrow$ new applications
What did we see today?

**3D reconstruction**: Implicit representation of a function

**NeRF**: Implicit representation of a scene

Positional Encoding $\rightarrow$ Fourier Features

**SIRENs**: NIR with sine activations $\rightarrow$ new applications

**NeX**: (one) Followup of NeRF
Questions?