



UNIVERSITY OF
CAMBRIDGE
COMPUTER LABORATORY



Cambridge Summer Course

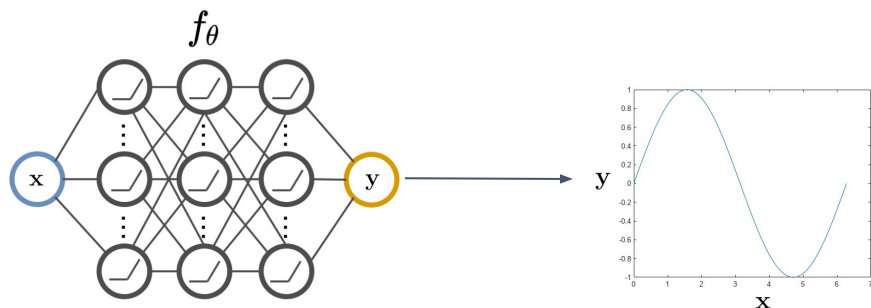
Practical Activity: Neural Radiance Fields

Homerton College. August 14, 2023.

Dr Alejandro Sztrajman

Neural Fields

Neural Fields are simple MLP neural networks that can be used to learn a **signal** or **function**. In other words, it's a neural network that learns the mapping between a **set of coordinates (input)** and a set of **outputs**.



In this simple example, we have an MLP neural network that maps a **coordinate x** to a **value y** . This means that if we give the network different values of x , we will receive the corresponding values of y . This mapping can be expressed like this:

$$y = f_\theta(x)$$

And so the information inside the neural network is encoding a **continuous function f** .

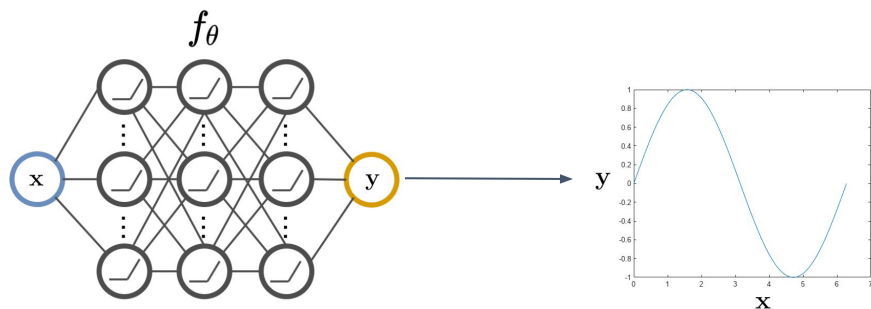
In this simple case, x and y are 1-dimensional and so the network is encoding a specific 1D function (a sine). We can plot this function simply by inputting different values of x to the network and drawing the output y values.



Neural Fields

This neural representation for functions has received many different names recently:

- Neural Fields.
- Implicit Neural Representations (INRs).
- Coordinate-based Networks.
- NeRFs (only a specific type).



There are multiple advantages to representing information/signals using neural fields:

- Continuous Representation (no grid).
- Infinite Resolution.
- Free non-linear interpolation.
- Free derivatives $\frac{\delta y}{\delta x}$ without discretization error.

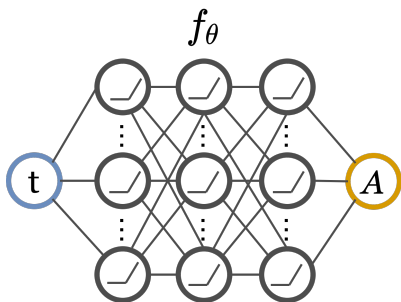
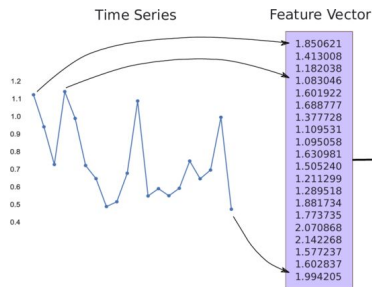
In this simple case, x and y are 1-dimensional and so the network is encoding a specific 1D function (a sine). We can plot this function simply by inputting different values of x to the network and drawing the output y values.



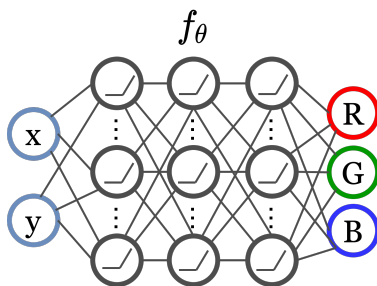
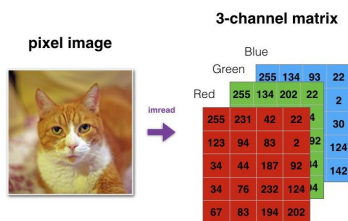
Neural Fields: Representing Complex Signals

So far we showed a simple 1-dimensional example that can be used to represent a plot (or a time-series). We show now examples with higher dimensionality, which can be used to represent more complex signals.

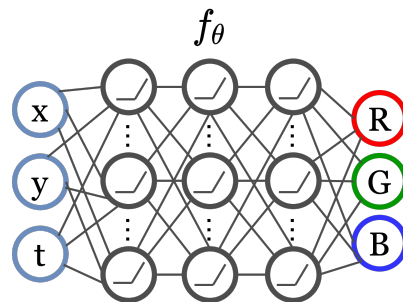
Time Series:



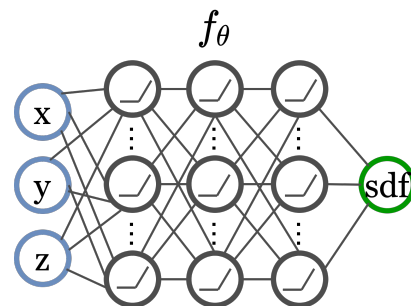
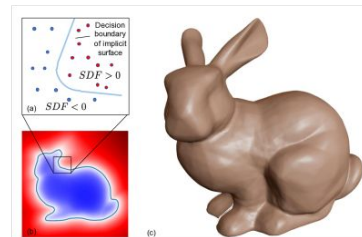
Images:



Video:

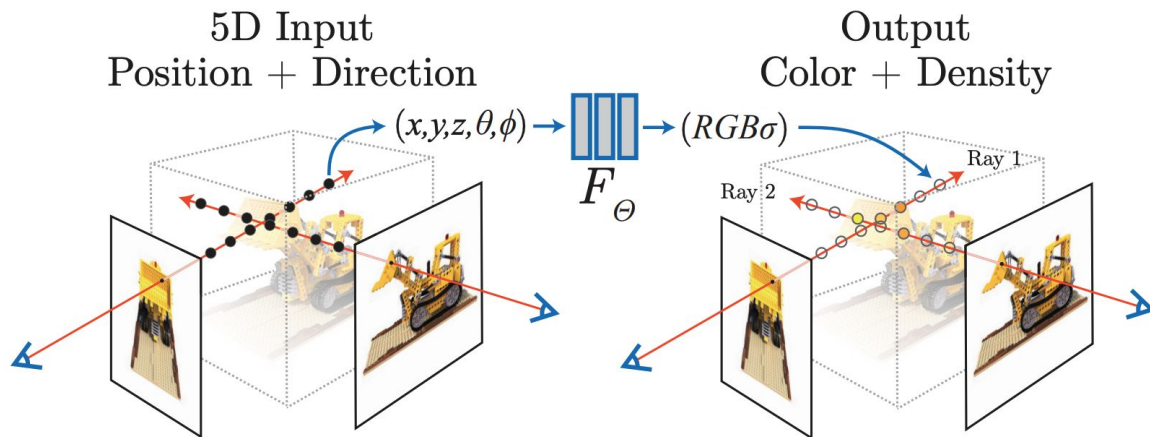
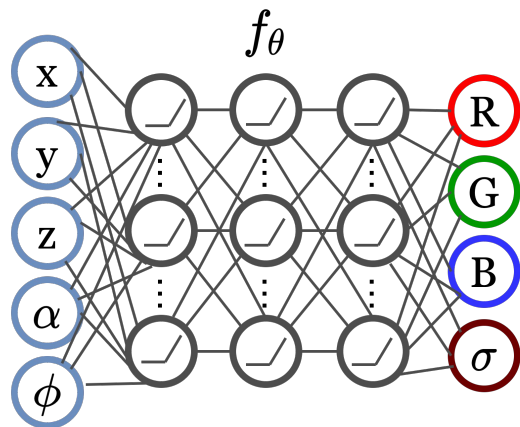


Geometry / Shape:



Neural Radiance Fields (NeRF)

Our last example of **Neural Fields** is the **Neural Radiance Field (NeRF)**. These are particular cases of neural fields that are used to represent entire realistic 3D scenes, including geometries, illumination and materials.



The input coordinates of the **NeRF** are the **position of a point in 3D space** (x,y,z) and the **view angles** α and ϕ . The output that we obtain are the **RGB values of outgoing light/radiance**, and the **density** σ of the material at the point (x,y,z) . **In short: we can ask the network the color and intensity of light that comes from every 3D point in the scene, in every direction.**



How to Train your NeRF?

Training a NeRF requires two things:

- Many pictures of a scene (e.g. 200).
- The **camera pose** of each picture (translation, rotation, etc. of the camera). This information is usually encoded in the **extrinsic matrix**.

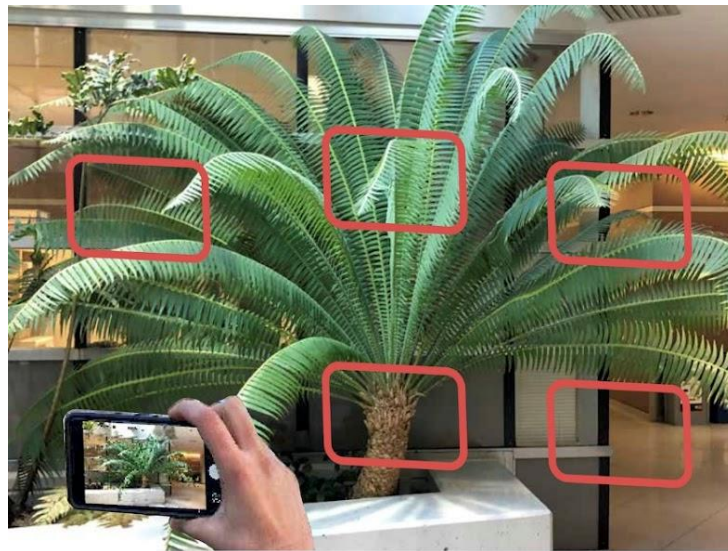


Image taken from: Mildenhall, Ben et al. "Local Light Field Fusion: Practical View Synthesis with Prescriptive Sampling Guidelines." *ACM Trans. on Graphics* 38, no. 4 (July 12, 2019): 1–14. <https://doi.org/10.1145/3306346.3322980>.



How to Train your NeRF?

Training a NeRF requires two things:

- Many pictures of a scene (e.g. 200).
- The **camera pose** of each picture (translation, rotation, etc. of the camera). This information is usually encoded in the **extrinsic matrix**.

After training is complete, the **NeRF** encodes all the information of the scene: **geometries, lights, materials** (even view-dependent effects). This enables **rendering** the scene from new points of view (known as *Novel View Synthesis*).



NeRF Rendering of fern from:
<https://www.matthewtancik.com/nerf>



How to Train your NeRF?



Training a NeRF requires two things:

- Many pictures of a scene (e.g. 200).
- The **camera pose** of each picture (translation, rotation, etc. of the camera). This information is usually encoded in the **extrinsic matrix**.

But how can we obtain the poses of the camera in the pictures?

Instead of taking individual pictures, we will record a video of the scene while moving the camera. We will then use a software that extracts images from the video and performs structure-from-motion (SfM) to estimate the camera poses of the individual images.

There are multiple software that can perform pose estimation:

Data	Capture Device	Requirements
 Images	Any	COLMAP
 Video	Any	COLMAP
 360 Data	Any	COLMAP
 Polycam	IOS with LiDAR	Polycam App
 KIRI Engine	IOS or Android	KIRI Engine App
 Record3D	IOS with LiDAR	Record3D app
 Metashape	Any	Metashape
 RealityCapture	Any	RealityCapture

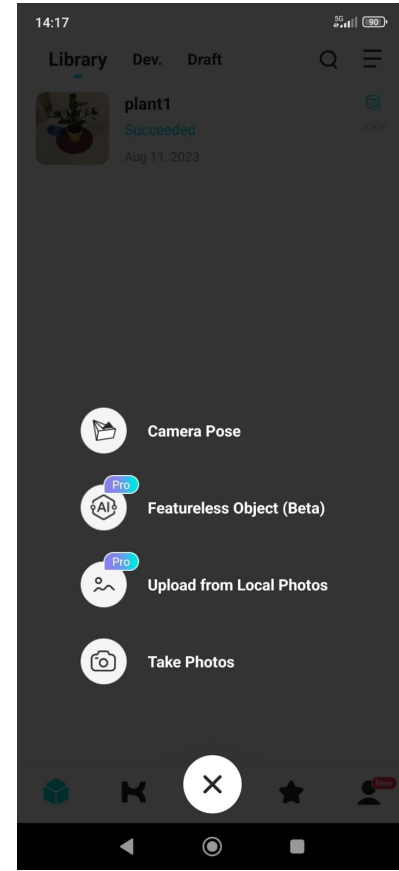


Practical Activity: What do we need to do?

We want to render a short video of a scene, where we move the camera around the scene a bit (similar to the rendering that we saw of the fern). In order to do that, we will need to follow these steps:

CAPTURE

- 1) Install the app "Kiri Engine" in your phone.
- 2) Go to the following [link](#) and follow the instructions to enable developer mode in the app.
- 3) In the same page, follow the instructions to do "pose estimation" and export the capture (download or email).
 - The captured scene must be static.
 - Move the camera slowly around the objects that you want to capture.
 - Try to capture the objects from multiple angles (from the top, from the sides, from below).
 - Try to keep the video short (15 sec. approx).



Practical Activity: What do we need to do?

We want to render a short video of a scene, where we move the camera around the scene a bit (similar to the rendering that we saw of the fern). In order to do that, we will need to follow these steps:

NERFSTUDIO COLAB TRAINING AND RENDERING

- 1) Go to <https://docs.nerf.studio/> and click in "colab" on the upper left corner of the page.
- 2) Run the cell "Install Nerfstudio and Dependencies" and check that everything went well.
- 3) Skip the cell "Downloading and Processing Data" (this is useful for other scan methods, not Kiri).
- 4) Create a new cell and type: `!npm install -g npm`. Run the cell.
- 5) Upload the scene exported from Kiri into colab (in a known location, e.g. /content/).
- 6) Run the "Start Training" Cell. You might need to make minor modifications (e.g. paths) in lines 4 and 5.
- 7) During the execution of the training loop, the output in colab will show an url to connect to the nerfstudio viewer. Click on this link. A new tab should open, with a 3D viewer of the scene. The quality of the scene should improve as the training progresses. You should also be able to see the set of individual pictures of the scene that were extracted from the original video.



Practical Activity: What do we need to do?

We want to render a short video of a scene, where we move the camera around the scene a bit (similar to the rendering that we saw of the fern). In order to do that, we will need to follow these steps:

NERFSTUDIO COLAB TRAINING AND RENDERING

8) Investigate the use of the viewer. You should be able to position multiple "cameras" in different positions in the scene. After positioning the cameras, it should be possible to export a .json file that contains the information about the "camera movement" (the viewer will generate a smooth movement of the camera between the locations that we selected previously).

9) Download the generated .json file.

10) Wait until the training has finished (without errors). Upload the .json file from step 9 to Google Colab, to the folder created by the training process.

11) Run the last cell in the Google Colab. You might need to modify the definition of "base_dir" to point to the folder created by the training (and where the .json file is). This will generate a rendered video of the scene that you should be able to download.

