AUSTRALIAN CENTRE FOR ROBOTIC VISION SUMMER SCHOOL

Radiance Fields: Some Novel Views

THE UNIVERSITY OF SYDNEY

ACFR AUSTRALIAN CENTRE FOR FIELD ROBOTICS

ROBOTIC IMAGING LAB

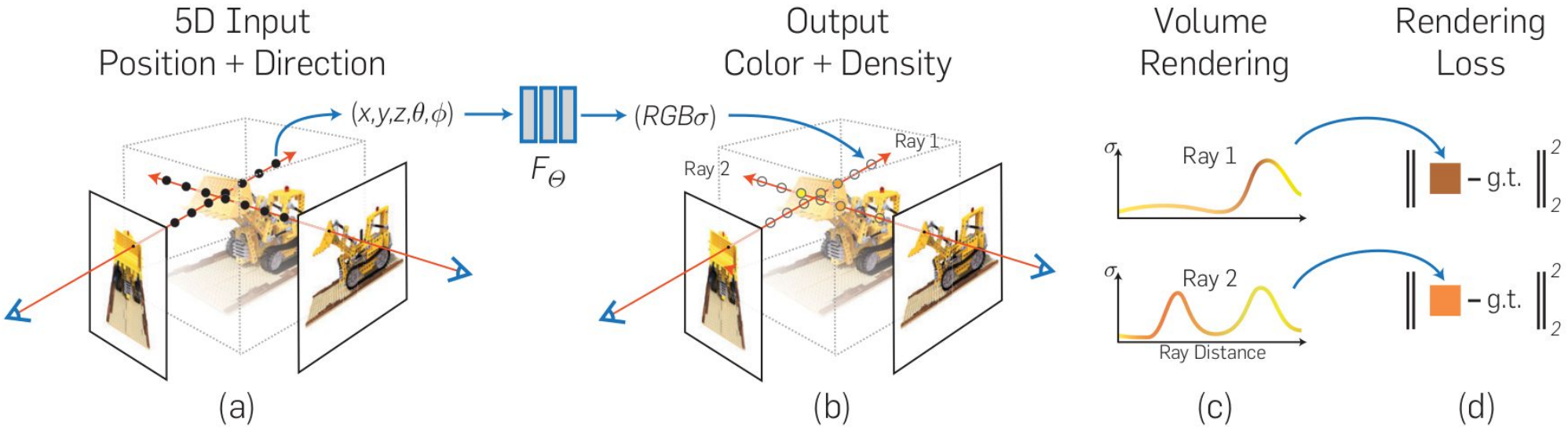
Don Dansereau

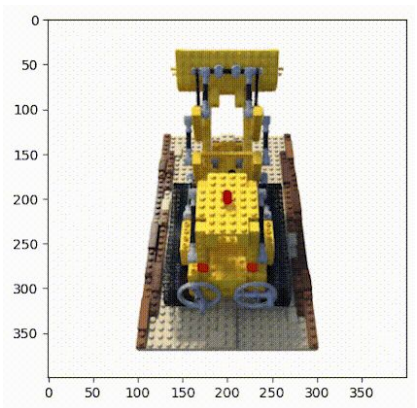
RVSS 2025

2025.Feb.05

Neural Radiance Fields: NeRFs

[Mildenhall2020]

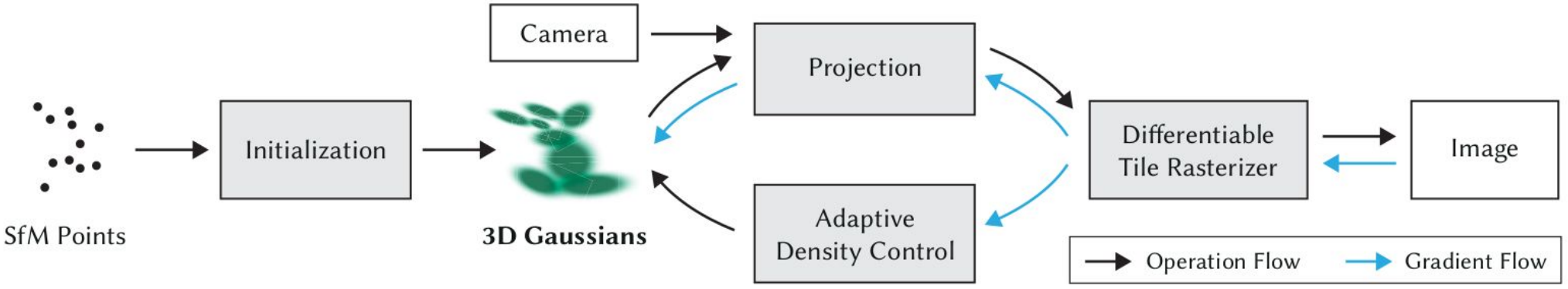




In: collection of images
Out: render novel views
Includes complex appearance
Transparency, reflection, etc.

Gaussian Splatting: 3DGS

[Kerbl2023]



In: collection of images
Out: render novel views
Includes complex appearance
Transparency, reflection, etc

Exciting Possibilities for Robotics

Summarise many images compactly
Natively handles complex visual appearance
Photo-realistic

https://gsplat.tech/

This Lecture

Let's invent NeRFs and 3DGS…

- Intuition for how they work
- Similarities, differences, quirks, limitations

About the code examples*:

- I want to run 100's of fast experiments
- To build my intuition through tweaking and interaction
- On a 5 year old laptop, no GPU, limited internet
- How?

- Low-res, 2D, incomplete, ugly, toy examples
- Not pretty, not state of the art…
- … but that's not the point.

*with credit /blame to ChatGPT

From the Top

What kind of algorithm is this?

Given a set of input images

Adjust a model

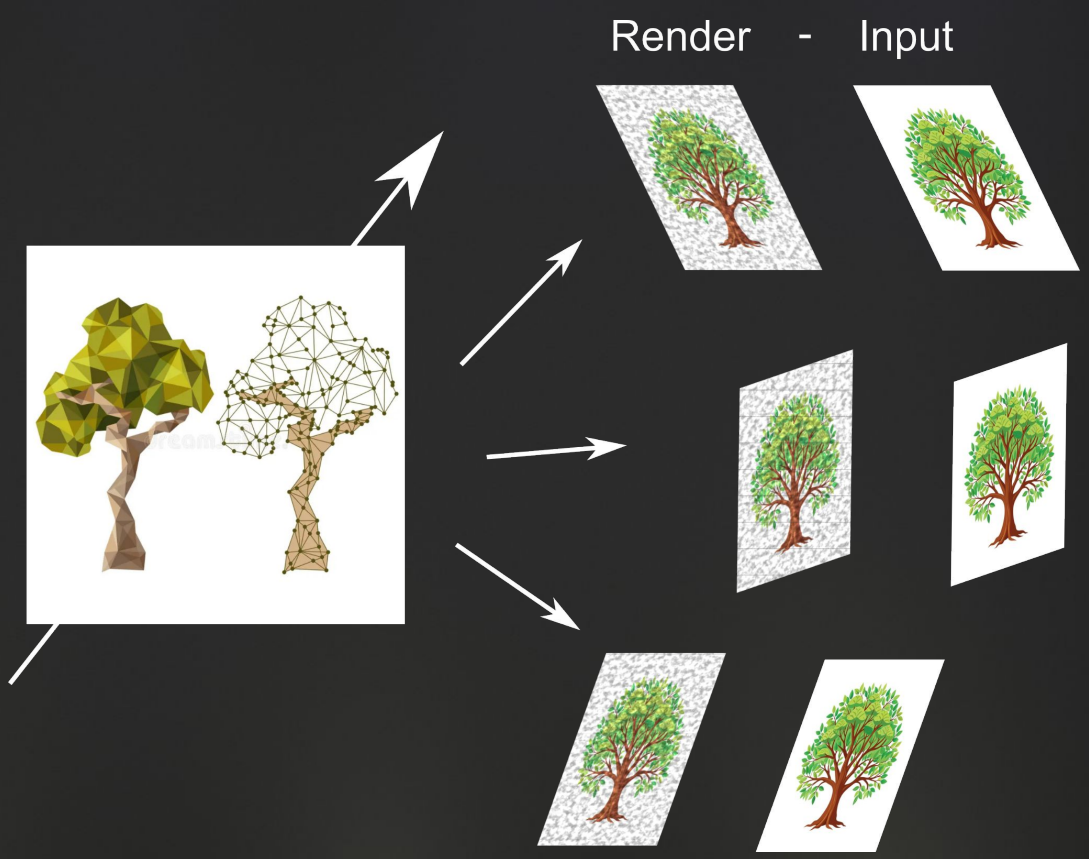
Until renders match inputs

“Inverse rendering”

e.g. Mitsuba

Typically polygonal, “hard” models

Often the model is the point



From the Top

What kind of algorithm is this?

Given a set of input images

Render novel views

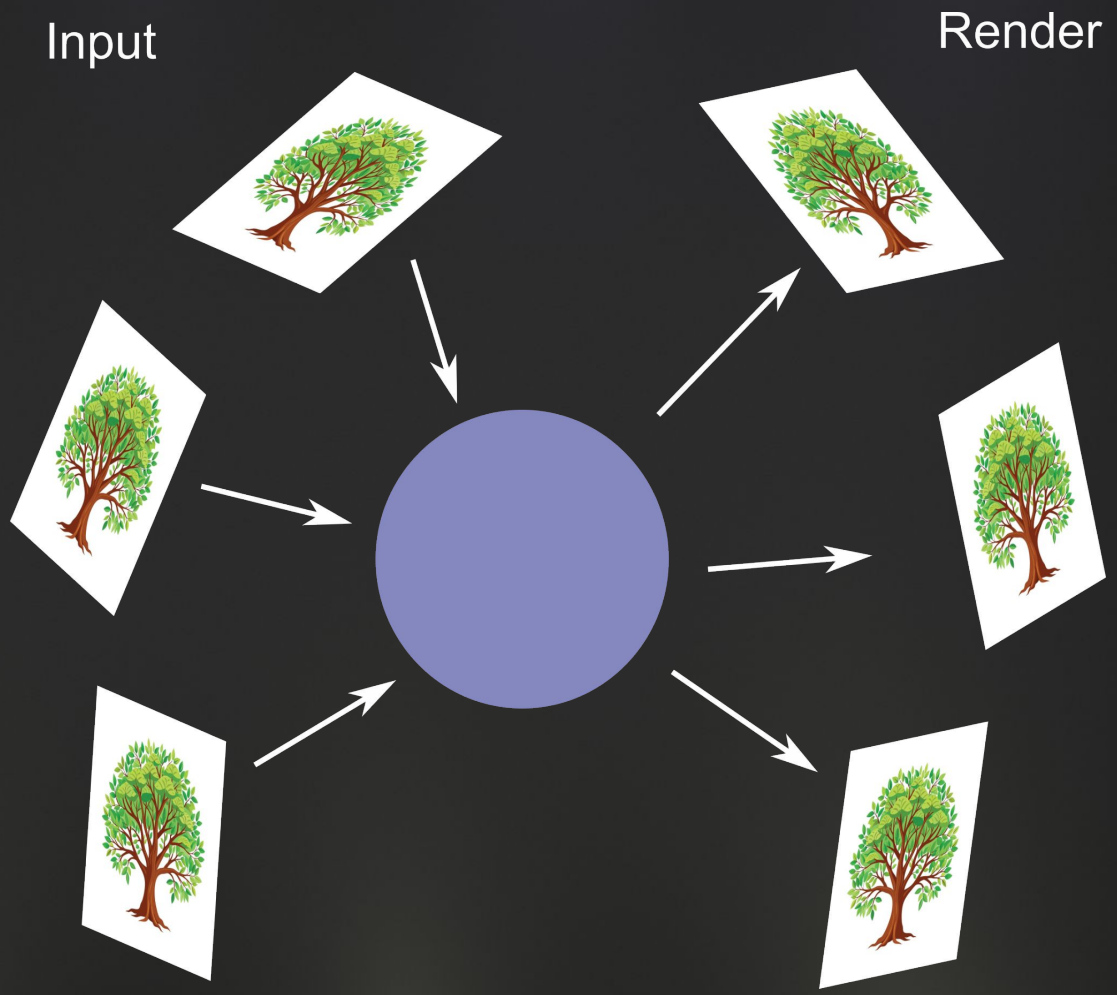
Directly from the images

“Image-based rendering”

e.g. Bullet time effect, The Matrix

Usually novel views are the point

“Novel view synthesis”



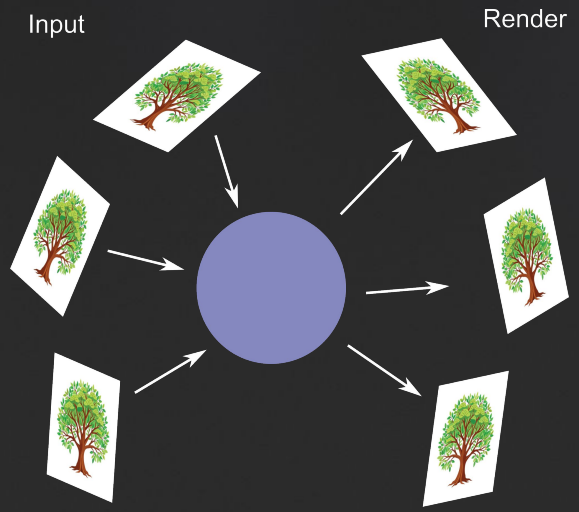
NeRF and 3DGS?

Given a set of input images

Render novel views

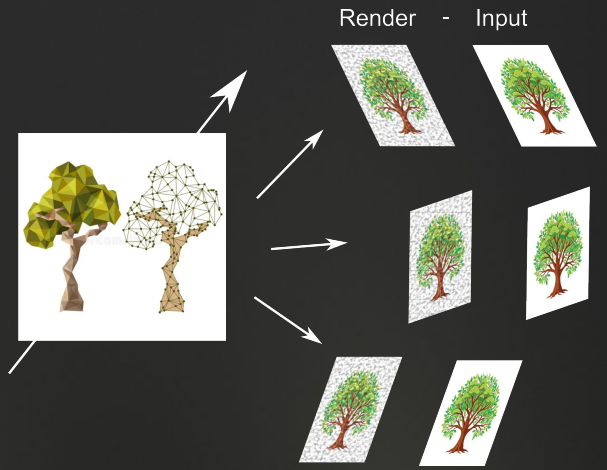
By fitting a sort of a model…

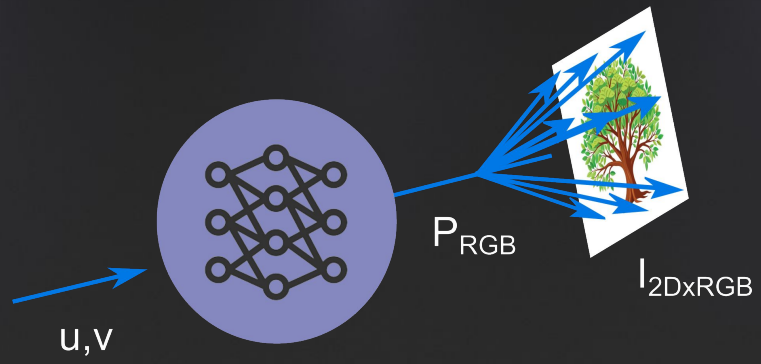
… but not a hard model



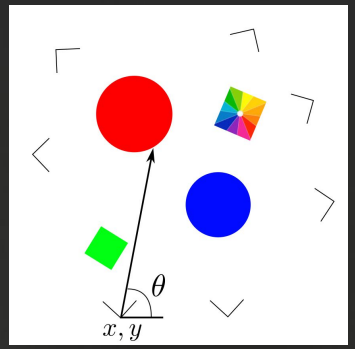
Both image-based and inverse rendering

… but with something special inside.





P1: Inventing Inverse Rendering in 2D

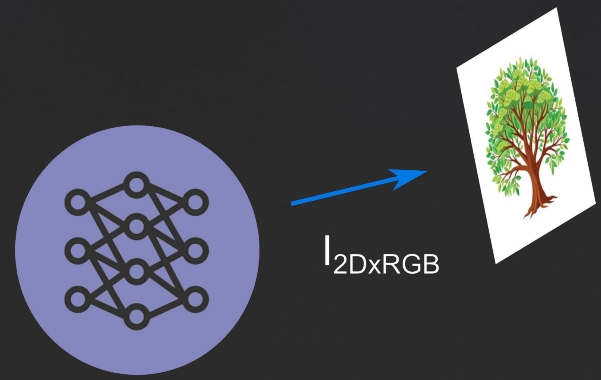


P2: Inventing Novel View Synthesis in Flatland

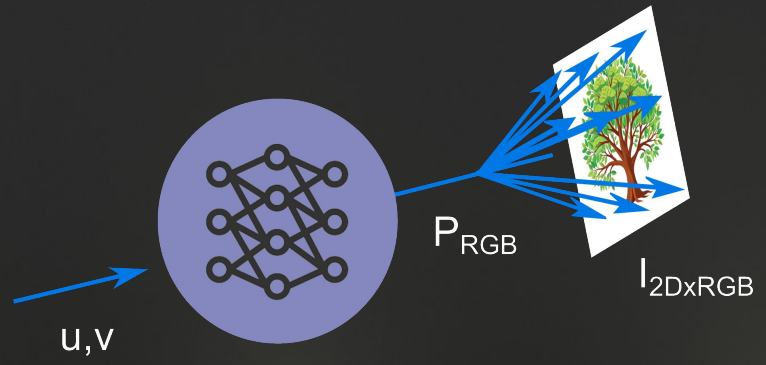
Inventing Neural Inverse Rendering for 2D Images

Approximate an RGB image I(u,v)

NN with big output layer
Directly builds 2D array of RGB pixels
Hard to learn
Not that useful (e.g. interpolate?)



NN with small output layer
Single RGB pixel output
We provide the coordinate
We do more of the work
More useful (e.g. interpolate)

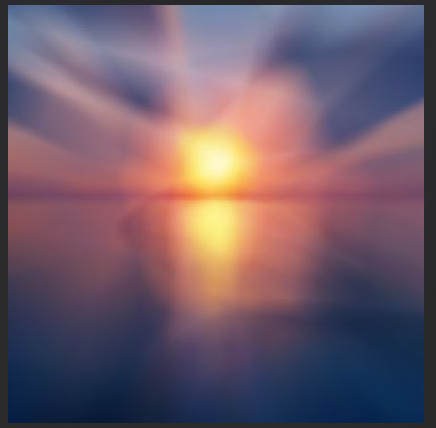


Inventing Neural Inverse Rendering for 2D Images

`invrender_2d_nn_simple.py`

It kind of works!
Struggling with fine detail

Our input is very smooth
v,u:
[0.0, 0.0], [0.0, 0.1], [0.0, 0.2], …
[0.1, 0.0], [0.1, 0.1], [0.1, 0.2], …
[0.2, 0.0], [0.2, 0.1], [0.2, 0.2], …
…
More informative input?



Output

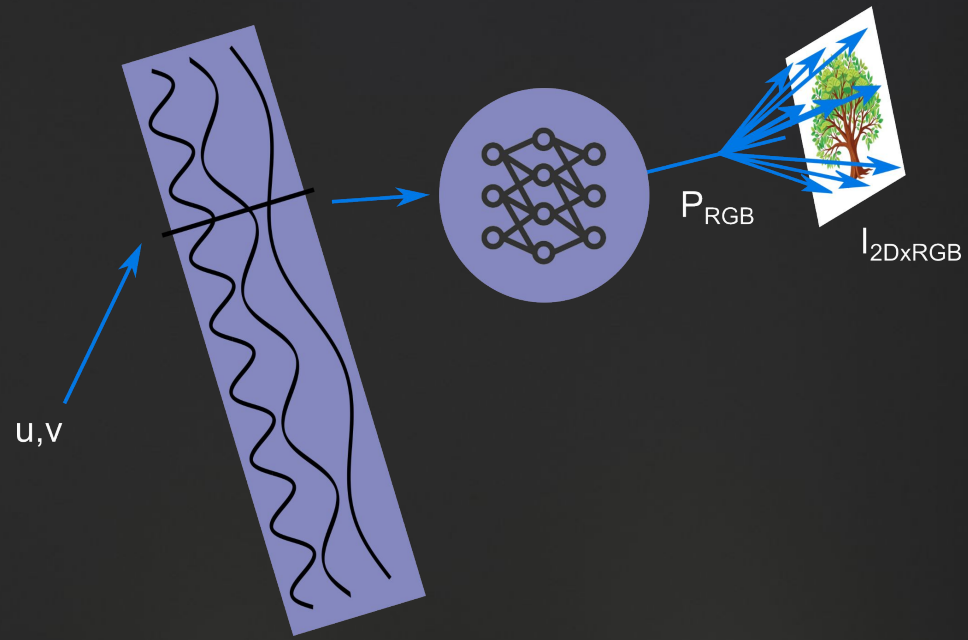
Target

Inventing Positional Encoding

Let's provide more texture in the input
Helps the network distinguish neighbours

u, v
cos(u), cos(2u), cos(4u), …
sin(u), sin(2u), sin(4u), …
cos(v), cos(2v), cos(4v), …
sin(v), sin(2v), sin(4v), …

Neighbours are distinct, unique
(smells a little Fourier?)



Inventing Neural Inverse Rendering for 2D Images

`invrender_2d_nn_positional.py`

Much better with positional encoding!

Try:

- Twiddle every single parameter
- Interpolate / upsample
- Learn multiple images
- Incrementally build a panorama

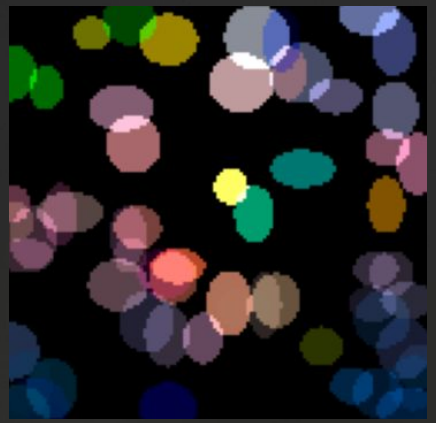
Output

Target

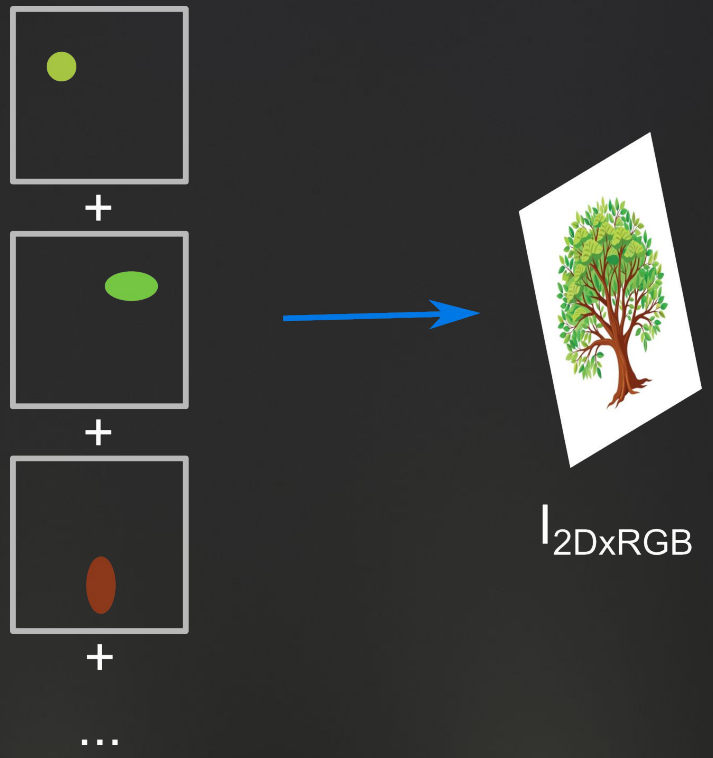
Inverse Rendering 2D Images with Point Primitives

Build image from many simple primitives
Adjust primitives to match target

Let's start with many circles
`invrender_2d_circles.py`



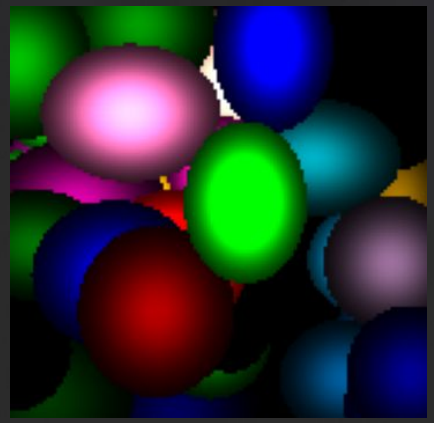
Spatial gradient is not smooth
No slope to follow



Inverse Rendering 2D Images with Gaussian Primitives

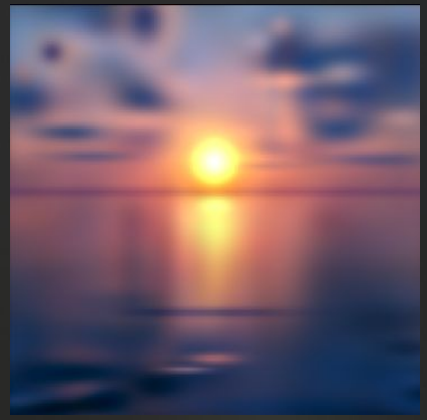
Again with opaque Gaussians
Adjust `invrender_2d_gaussians.py`

Smooth edges allow some gradient
… but most changes are hidden in back



Again with transparent Gaussians
Adjust `invrender_2d_gaussians.py`
Adding up all Gaussians at each pixel

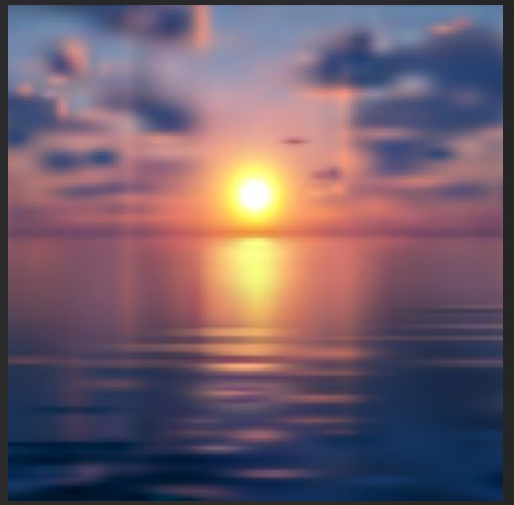
Much more promising!
Again with 128 blobs



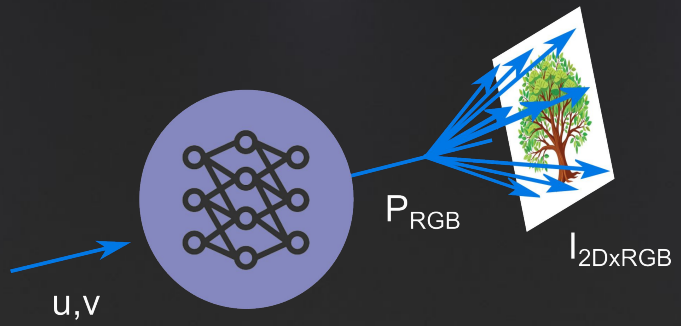
Recap: Inverse Rendering

Many representations can capture the same scene
NNs benefit from input coordinates and positional encoding
Inverse rendering needs soft representations
    Provides a gradient to climb
Watching the optimisation process can build intuition

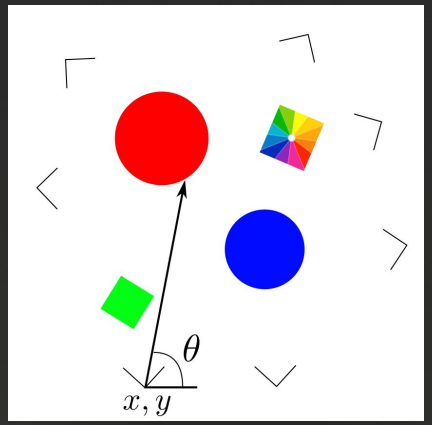
Ideas to try:
    NN hash encoding (sparse and dense examples)
    Gaussian blobs with rotation
    Periodically add / remove blobs / reduce opacity of blobs
    Regularisers!
    Video; video interpolation!



Gaussians w/dynamic addition / removal



P1: Inventing Inverse Rendering in 2D



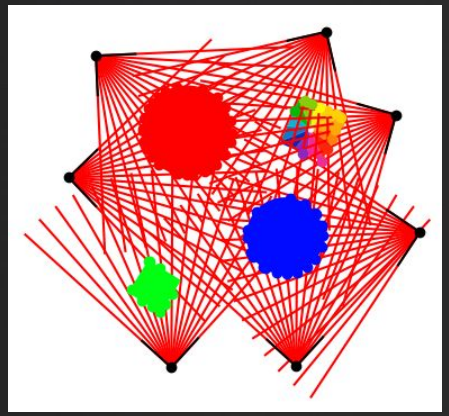
P2: Inventing Novel View Synthesis in Flatland

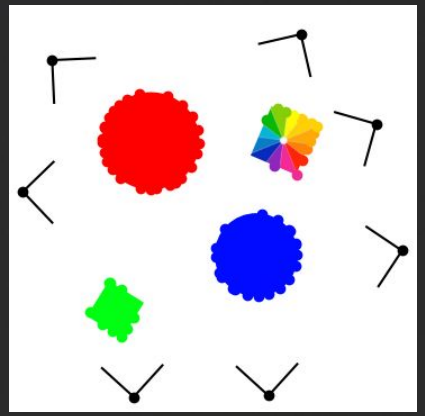
Welcome to Flatland

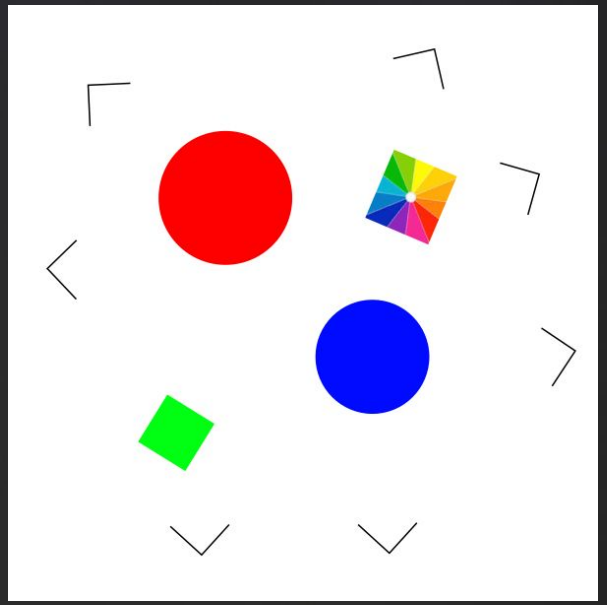
What does a camera see in flatland?

`renderer2d.py, flatland_intro.svg`

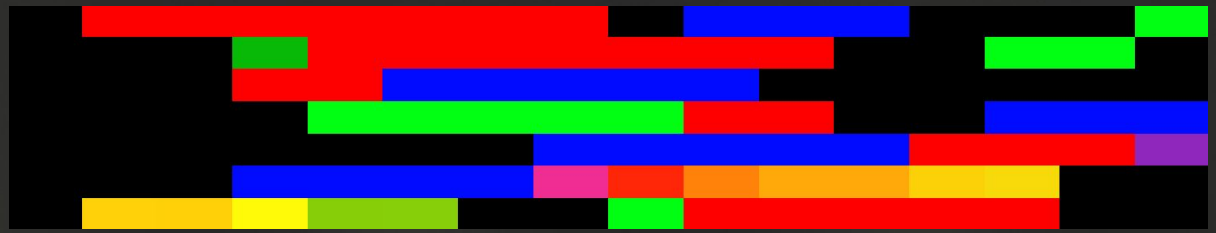
Loads and renders from SVG files







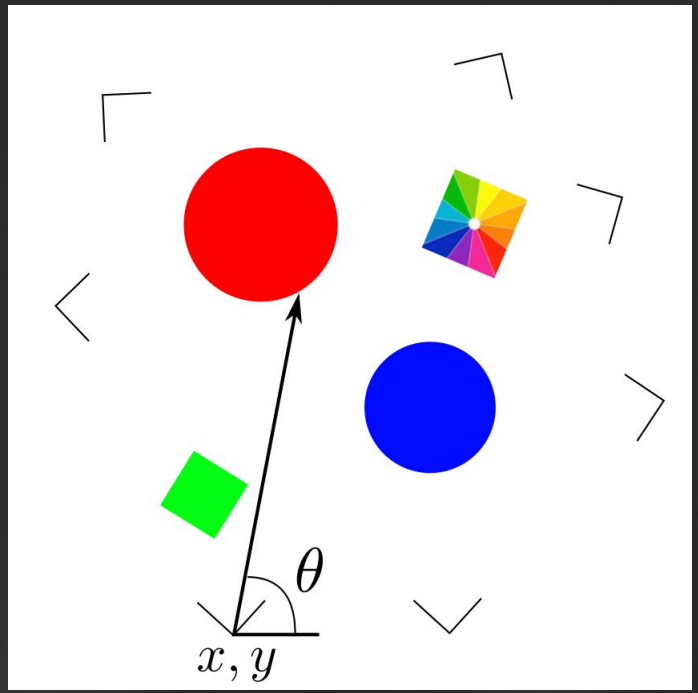
Cameras

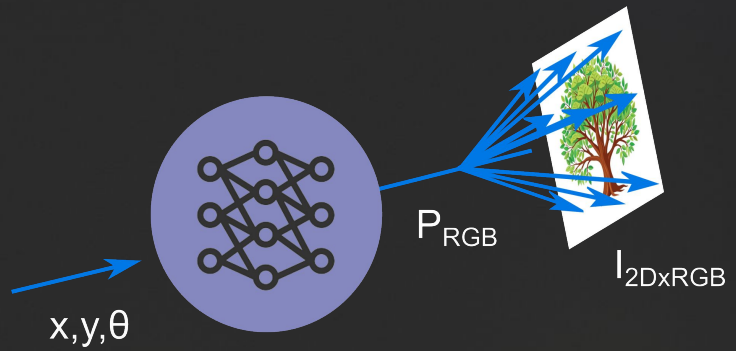


Pixels

Inverse Rendering from Rays

Let’s do inverse rendering on the space of light rays
Like our 2D renderer on the space of pixels
How to describe each light ray?



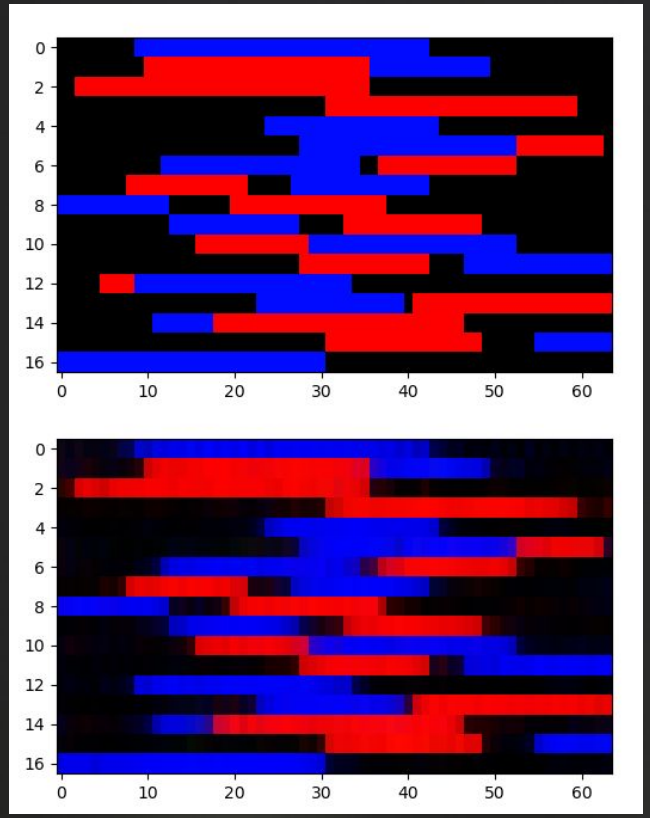


+Positional encoding

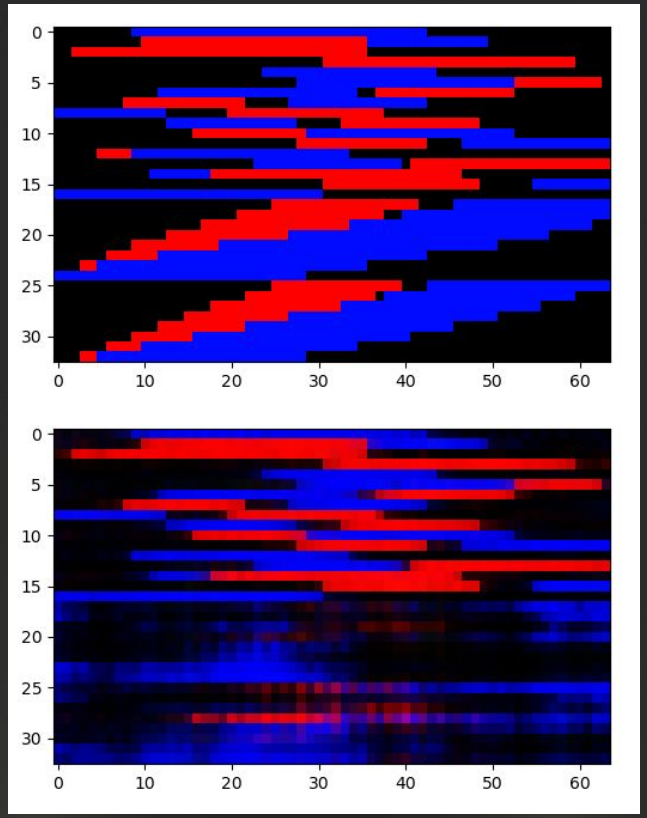
Inverse Rendering from Rays

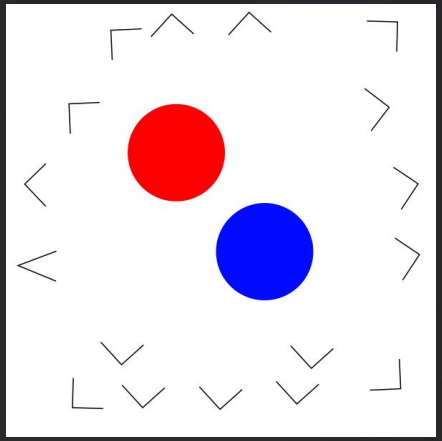
`nvs_nn_full_ray.py, scene_simple.svg`

Trains fine

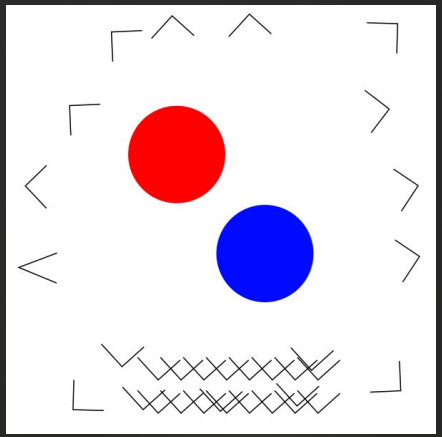


Test not so good





Train



Test

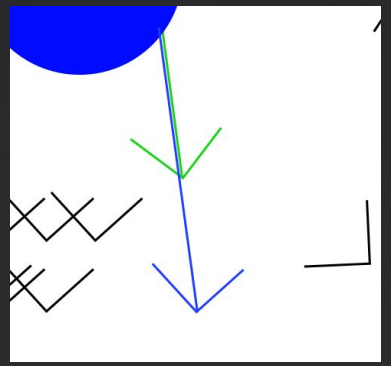
Let's Invent a Simpler Ray

Some rays are identical

But have different parameterizations in x,y,θ

New ray representation:

Same identity along direction of propagation



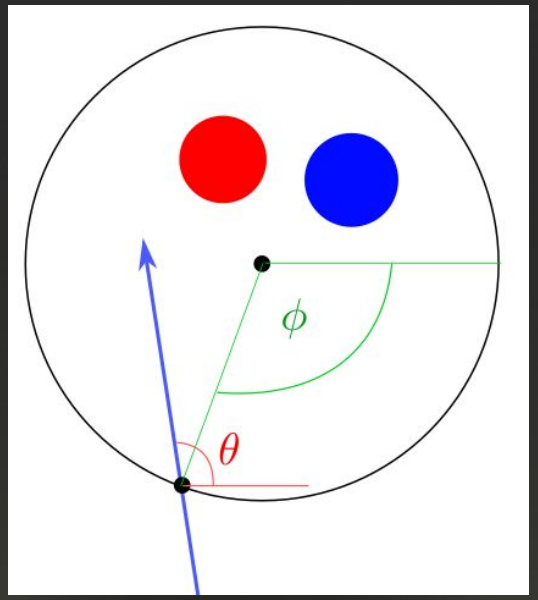
Reference surface around scene

Point of intersection on the surface ϕ

Direction θ

Overlapping rays get same parameterisation

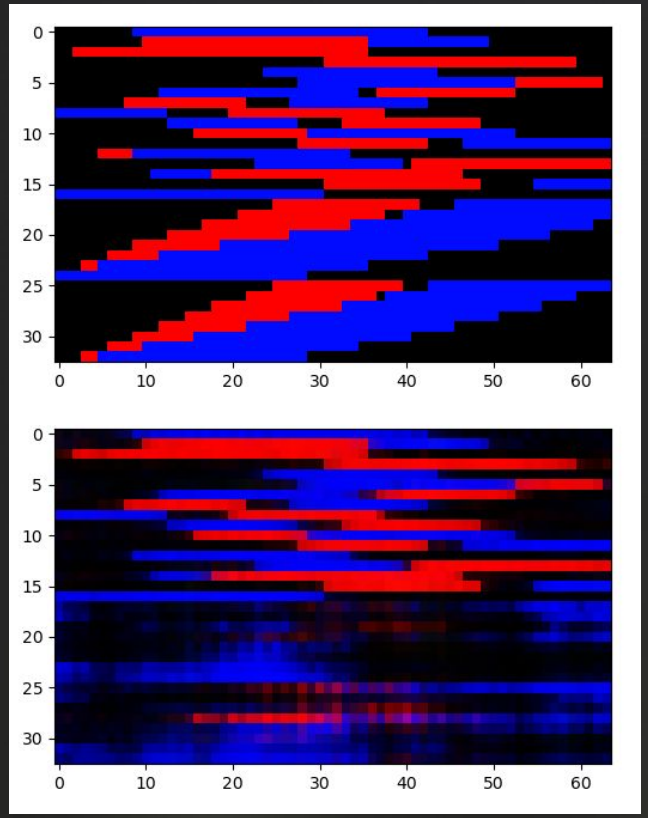
Fewer dims: learning in 2D should be easier…



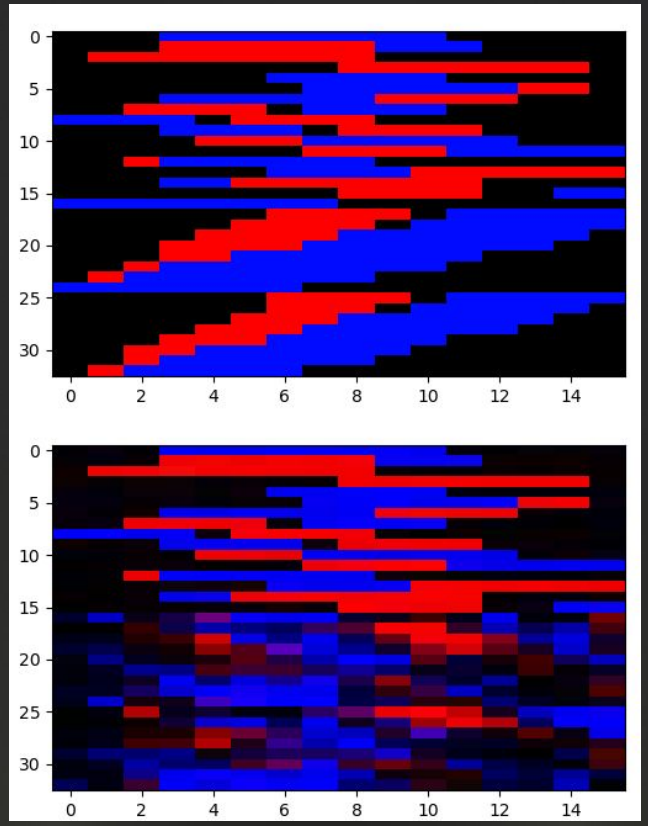
Rendering Simpler Rays

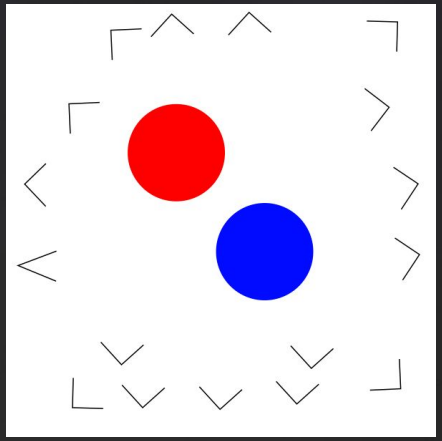
`nvs_nn_lf.py`

Full ray

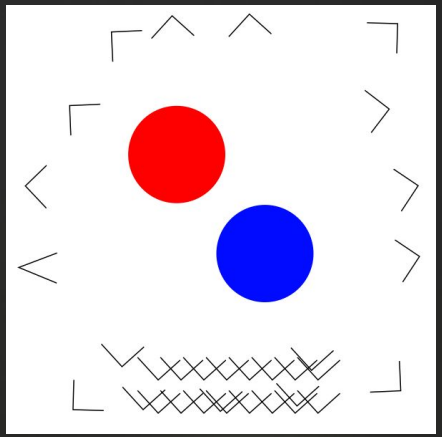


Simplified ray, not great…





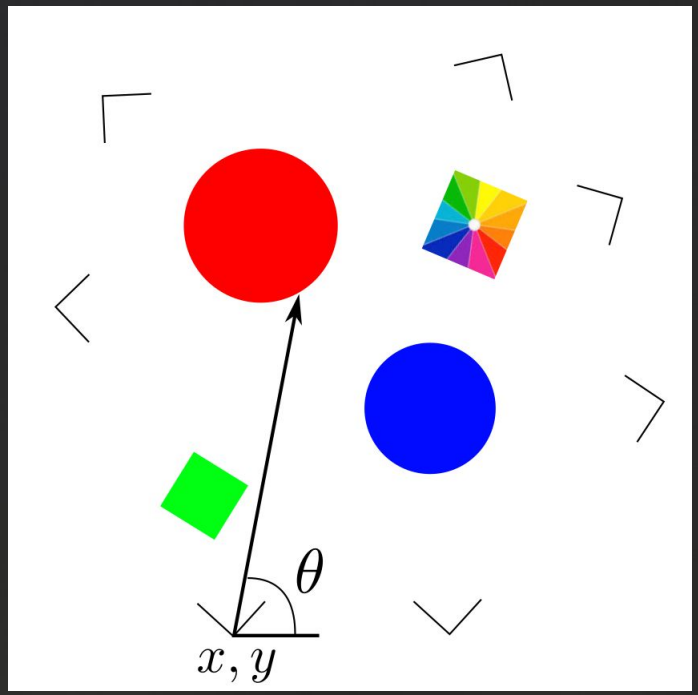
Train



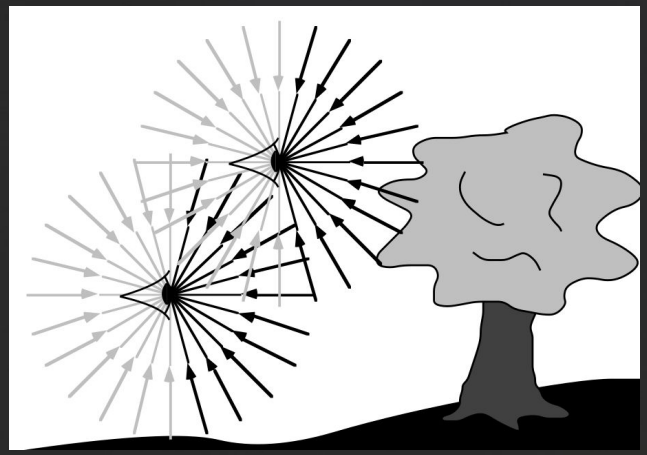
Test

Recap and Terminology so Far

In 2D: L(x, y, θ)



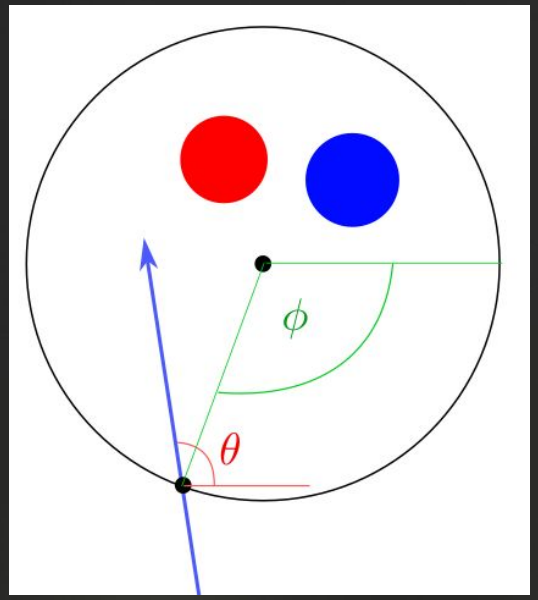
In 3D: L(x, y, z, θ, ϕ)



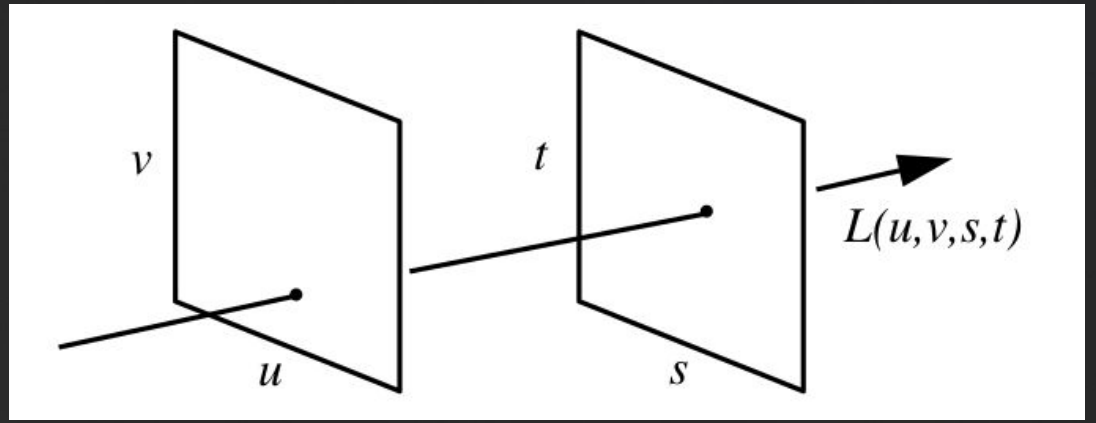
Adelson and Bergen, 1991
"The Plenoptic Function and the Elements of Early Vision"

Recap and Terminology

In 2D: L(θ, ϕ)
Reference surface
1D position
1D direction



In 3D: L(s, t, u, v)



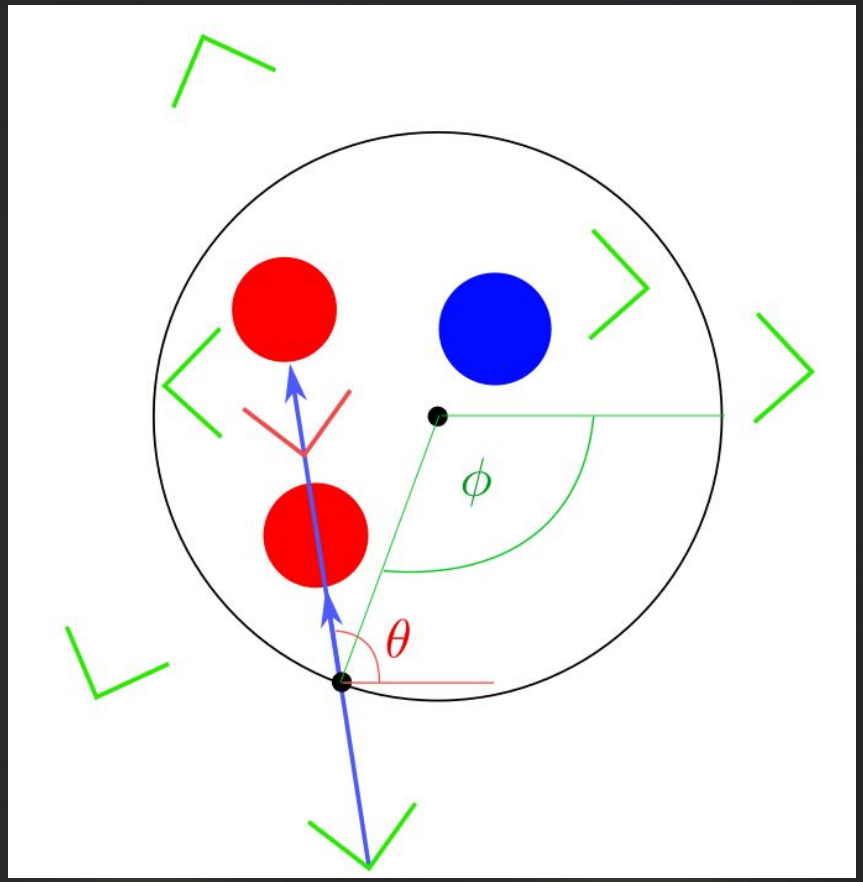
Levoy and Hanrahan, 1996
“Light Field Rendering”
2D position
2D direction

Problems with the Light Field

Occlusions are mostly fine…

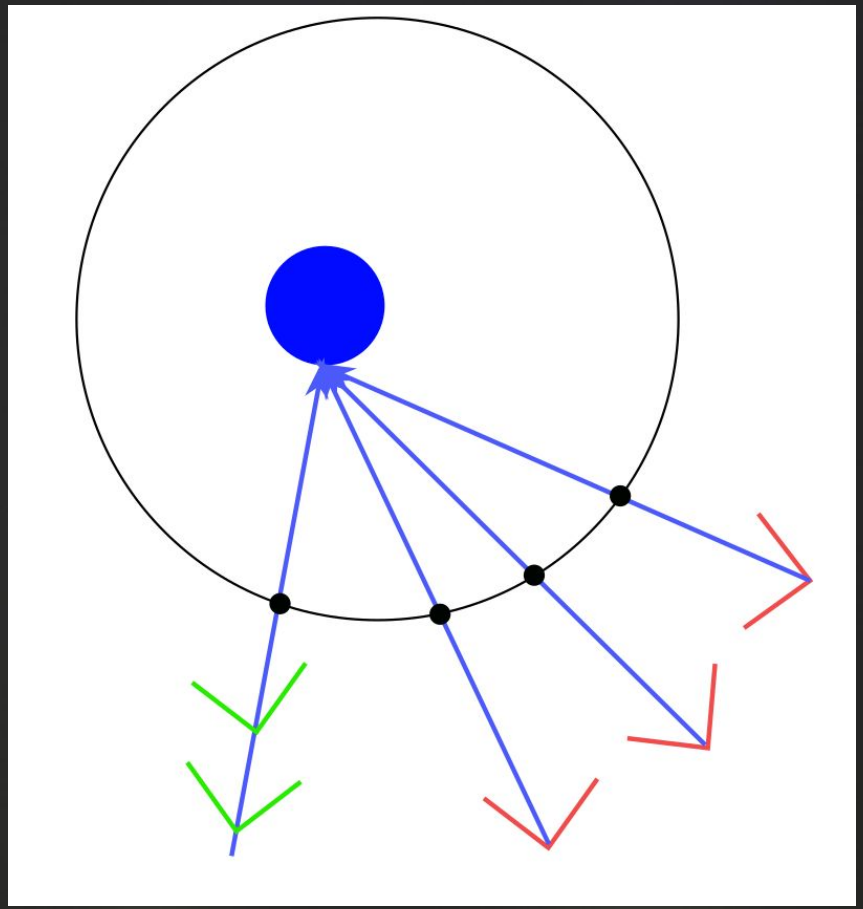
With some exceptions (e.g. in red):

One ray, two values

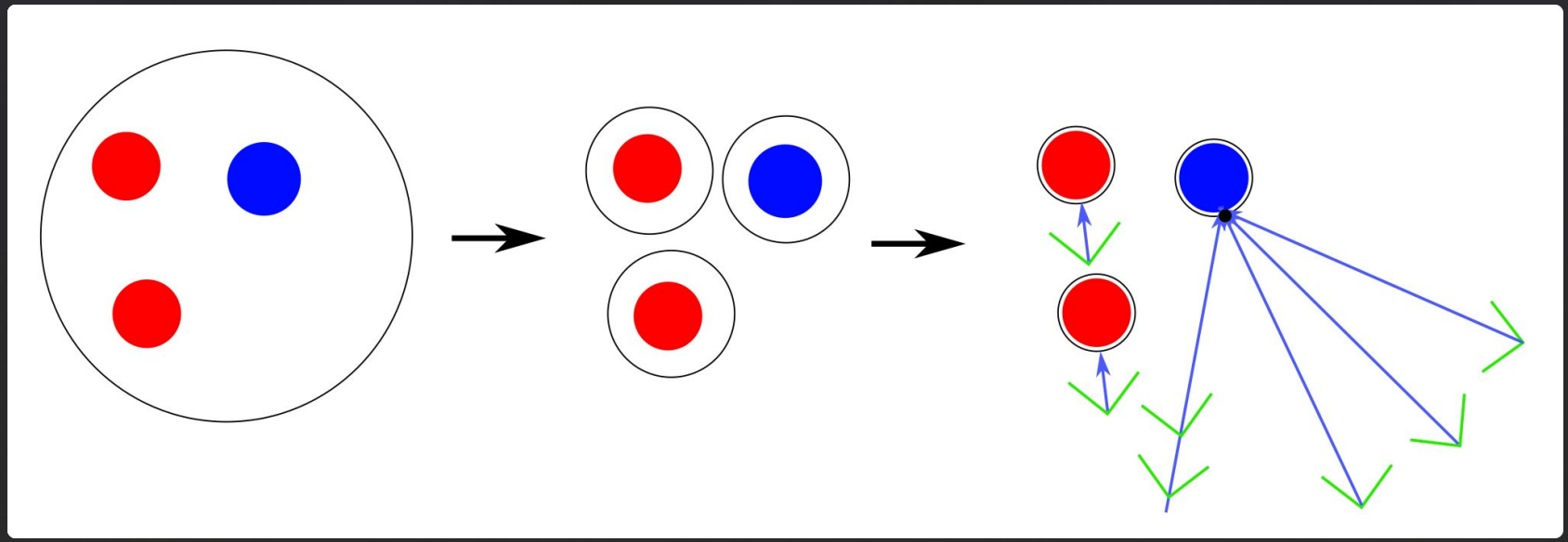


Problems with the Light Field

We fixed redundancy along ray…
But many related rays are far apart in θ,ϕ
(for Lambertian scenes)
→ Hard to learn a regression



Let's Invent…



Single light field Multiple light fields Surface light field

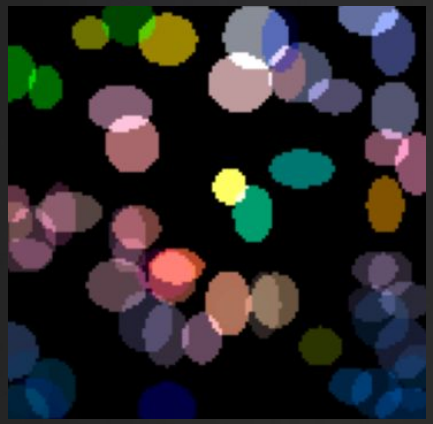
Occlusion problem fixed

Similar rays have similar parameterisations

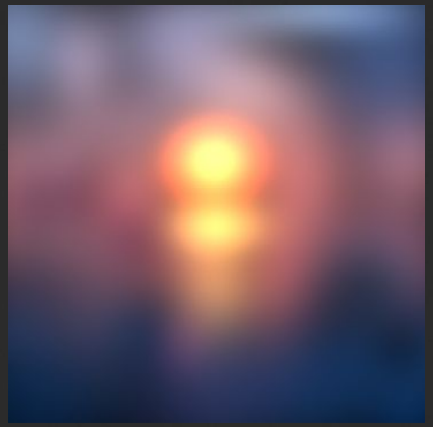
… but how do we pick the surface? i.e. learn the ray parameterisation?

Learning a Light Field Ray Parameterisation

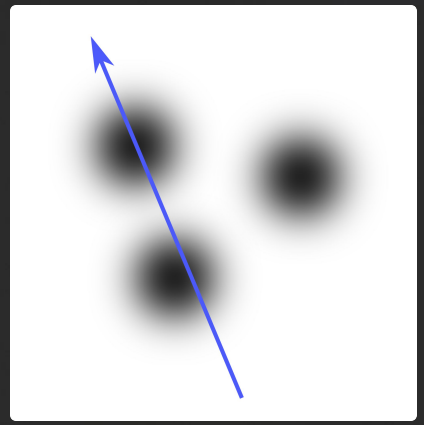
Hard edges?



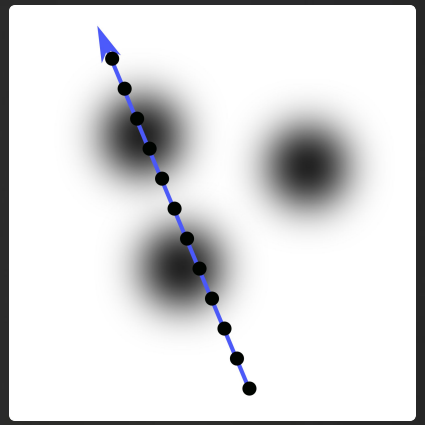
Soft blobs!



Where's the surface?



Along the whole ray!



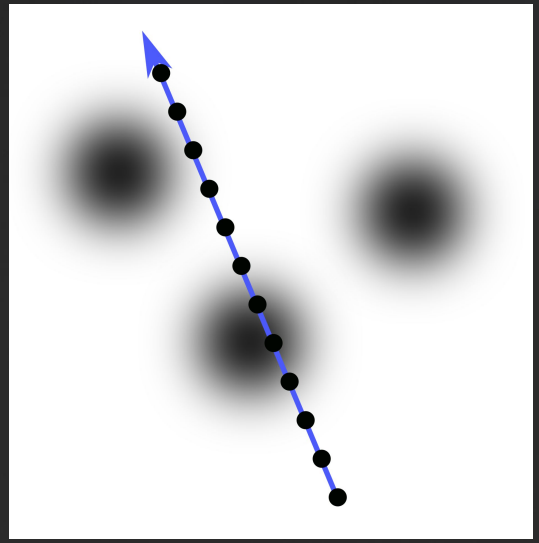
Using our Distributed Light Field Parameterisation

At each point along a ray
Evaluate the local LF
Density d (x, y)
Color c (x, y, θ)

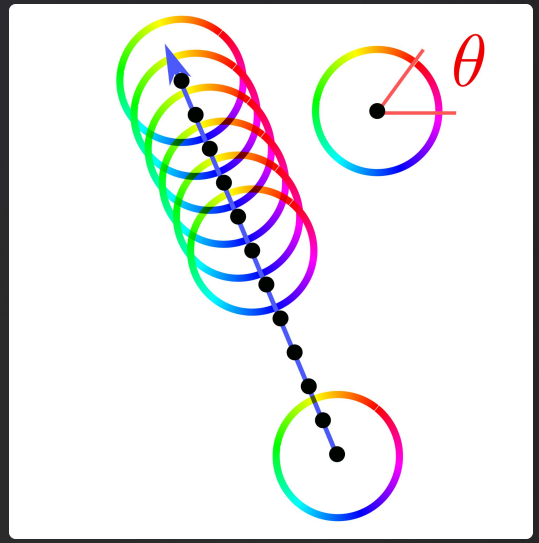
“Density” means how much does this point contribute visually

A soft surface light field!

We need to account for occlusion…
Easiest to understand approach:



Density



Colour

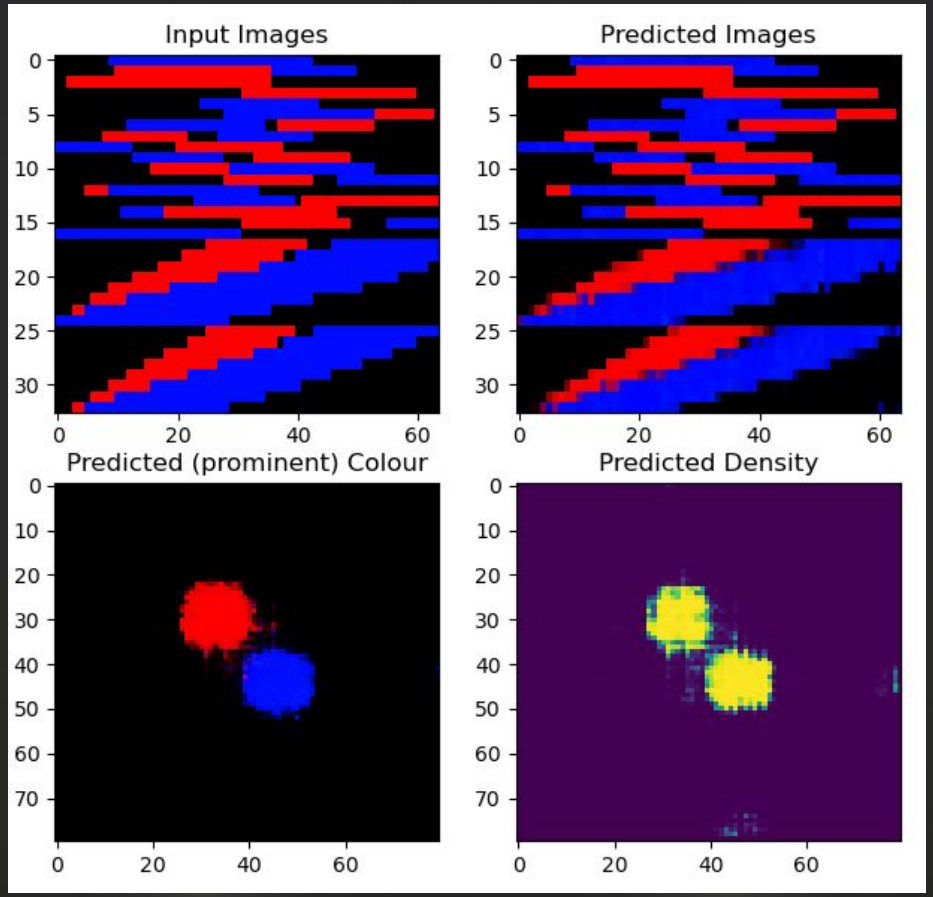
Along the ray from far to near
Take new colour proportional to density

$$C_i = c \cdot d + C_{i-1} \cdot (1 - d)$$

Let’s Neural Regress the Soft Surface Light Field

Neurally regress density d (x, y) and color c (x, y, θ): `nvs_nn_nerf.py`

Test: Success!



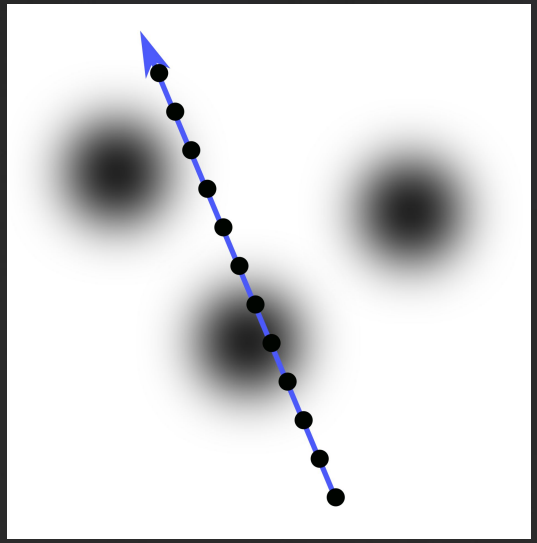
Let's Get Explicit: Gaussian Primitives

Density = ∑ Gaussians(x,y)
Just like our 2D sunset renders!
Blob has position, size, density

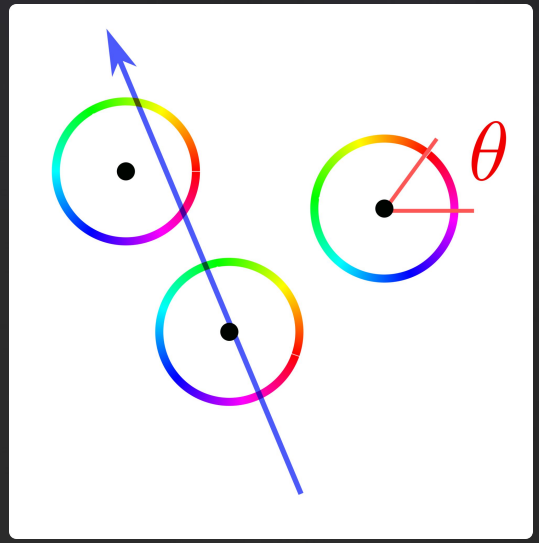
What about color c (x, y, θ)?

We could store a sampled function c (x, y, θ).

Why not store a single c(θ) with each Gaussian?



Density



Colour

And compose c(θ) with circular harmonics?

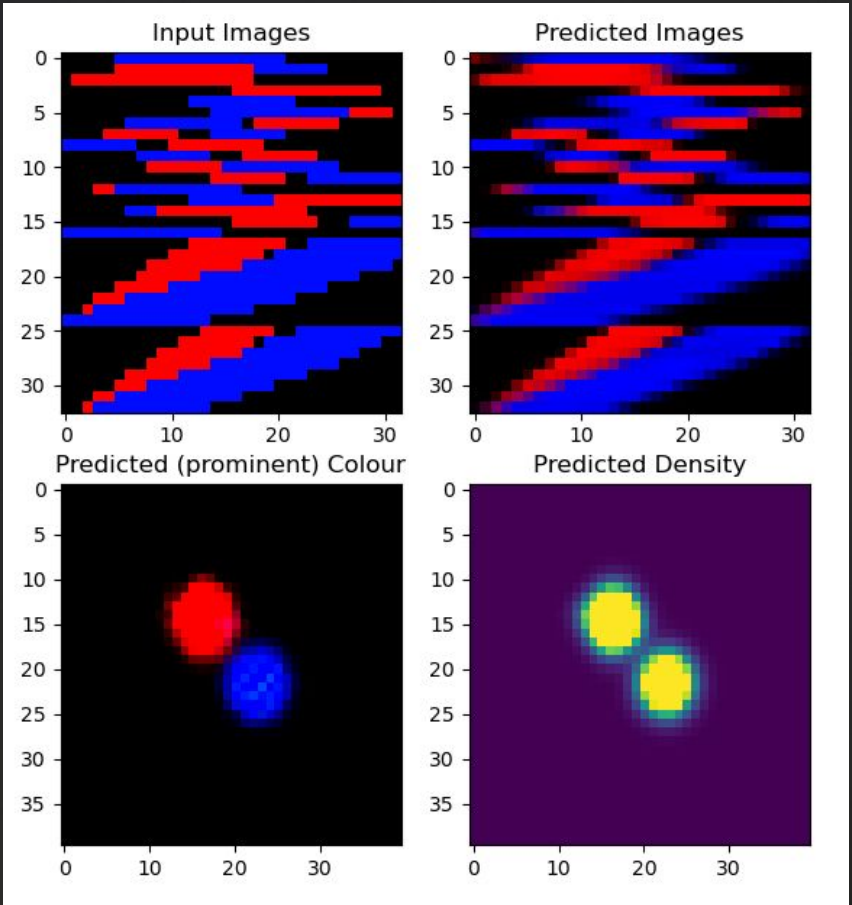
$c(\theta) = \sum A_n \cos(n \cdot \theta) + B_n \sin(n \cdot \theta)$

n = …, -2, -1, 0, 1, 2, …

Ray Marching on Gaussian Primitives

Raymarch Gaussians d (x, y) with harmonics c (θ): `nvs_gaussian_raymarch.py`

Test: Success! *(Sometimes)*



To try:

Anisotropy

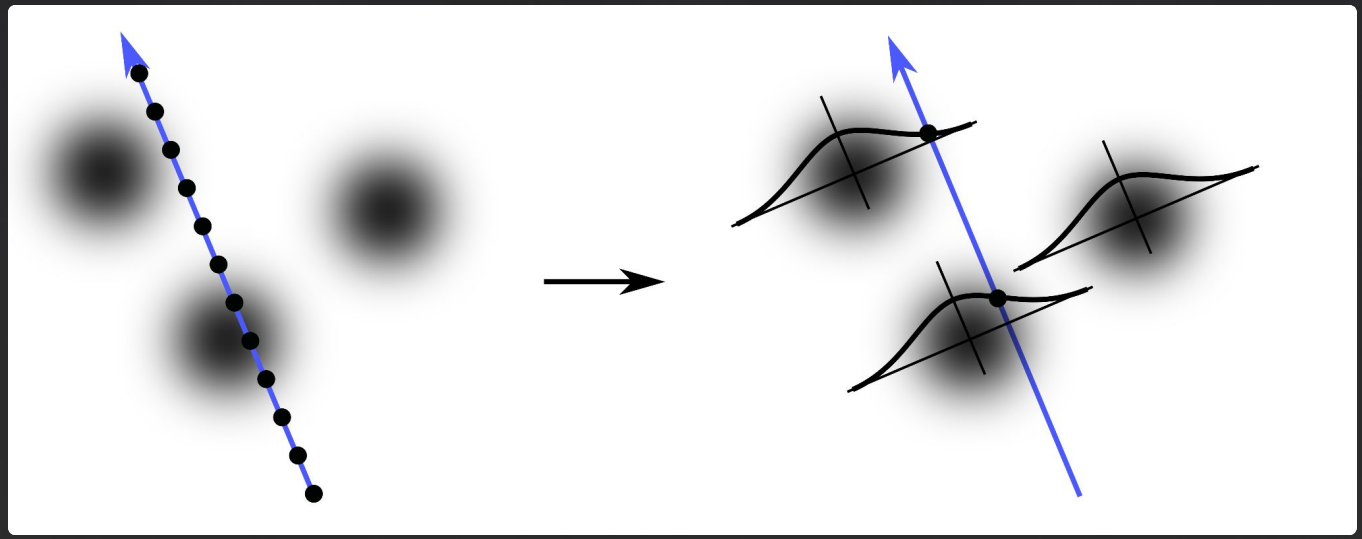
Start without harmonics

Different init params

Periodically reduce alpha

Regularisers

Taking a Shortcut



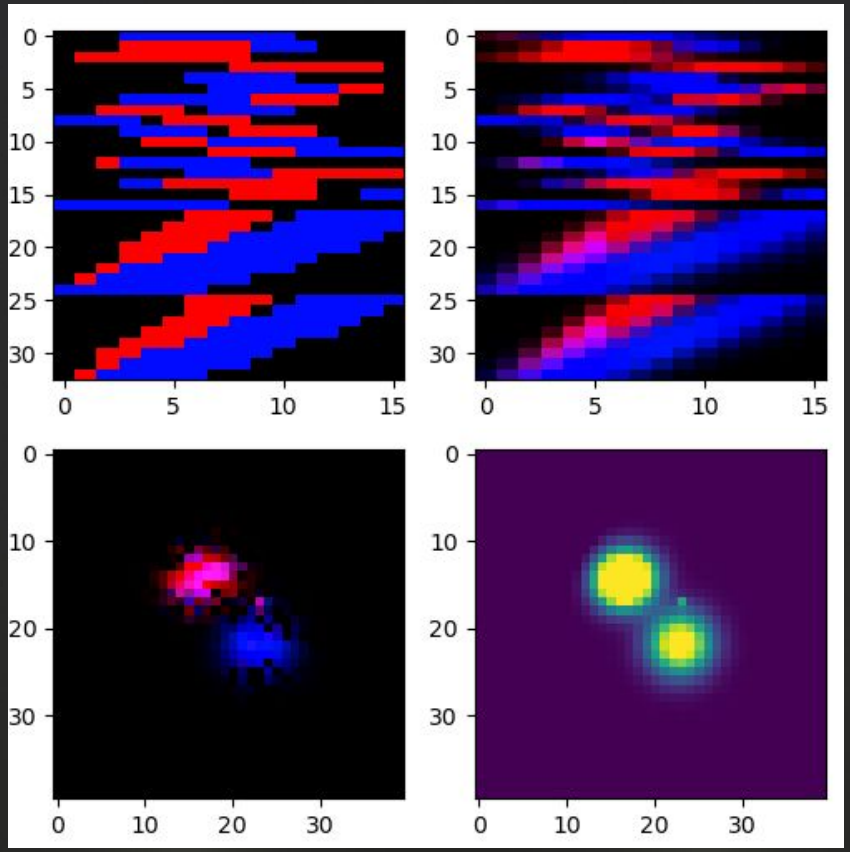
Many evaluations
for every ray
for every blob in 2D

Evaluate each blob once
At densest point
Fewer evaluations
Same blending
Much faster
Approximate

Splatting Gaussian Primitives

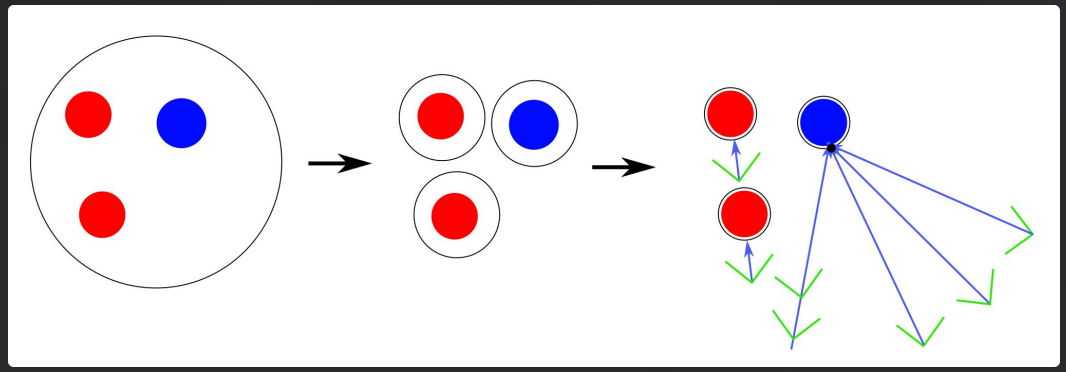
Splat Gaussians for d (x, y) with harmonics c (θ): `nvs_gaussian_splat.py`

Test: Success! *(Sometimes)*



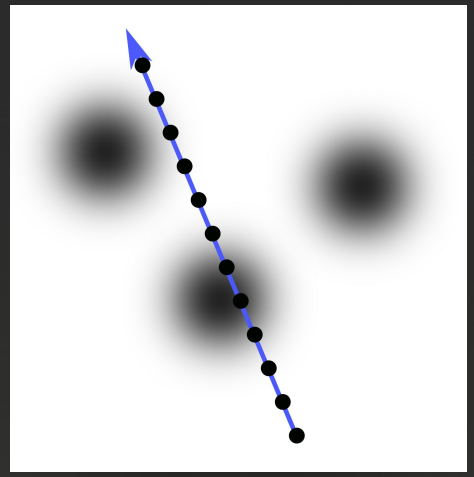
Recap and Terminology

Surface light field
Distributed density + colour

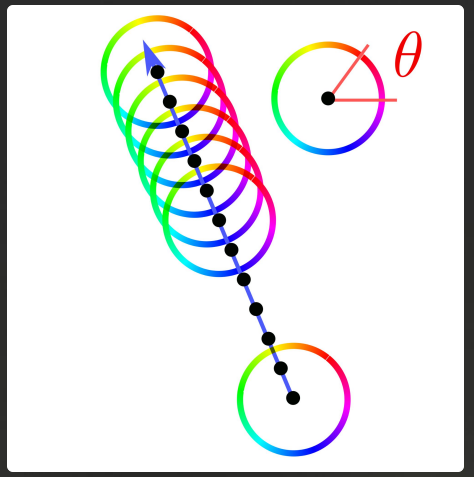


In 3D: "**Radiance Field**"

Occlusion handling:
$C_i = c \cdot d + C_{i-1} \cdot (1 - d)$
"Alpha blending"



2D: Density $d(x, y)$
3D: Density $d(x, y, z)$

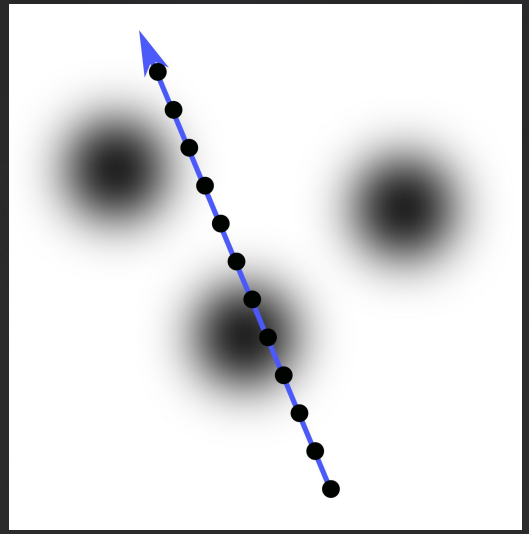


Colour $c(x, y, \theta)$
Colour $c(x, y, z, \theta, \phi)$

Recap and Terminology

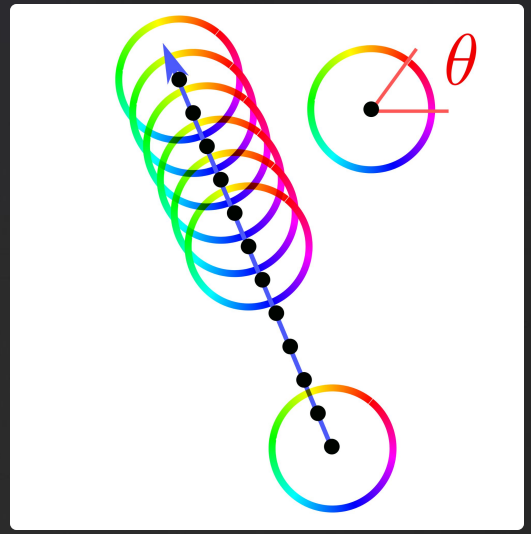
Neurally regress density, colour

"**Neural Radiance Field**", NeRF



2D: Density $d(x, y)$

3D: Density $d(x, y, z)$



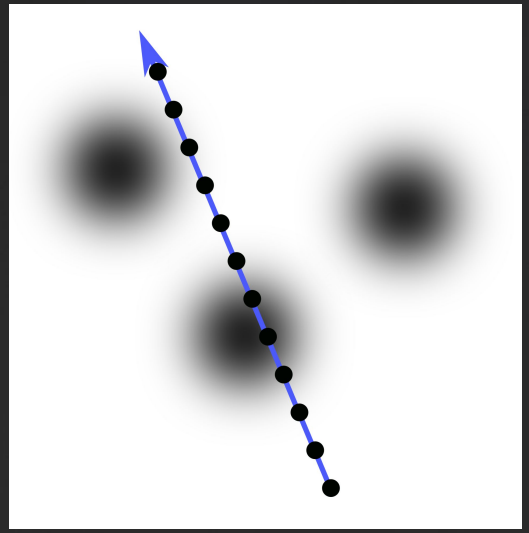
Colour $c(x, y, \theta)$

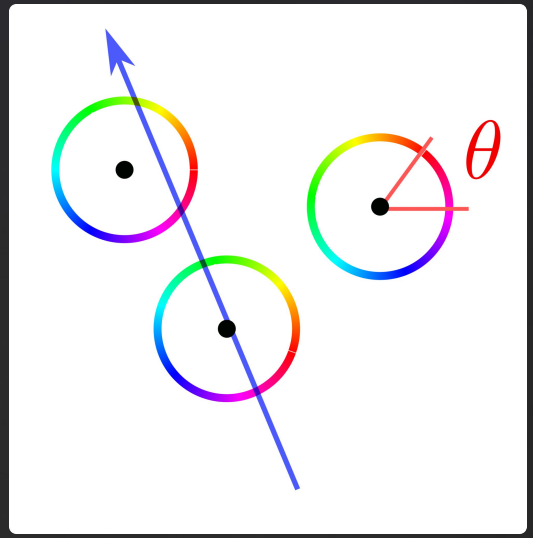
Colour $c(x, y, z, \theta, \phi)$

Recap and Terminology

Build density with Gaussians
Per-blob colour with harmonics

“Gaussian Splatting”, GS3D
*(**almost**)*





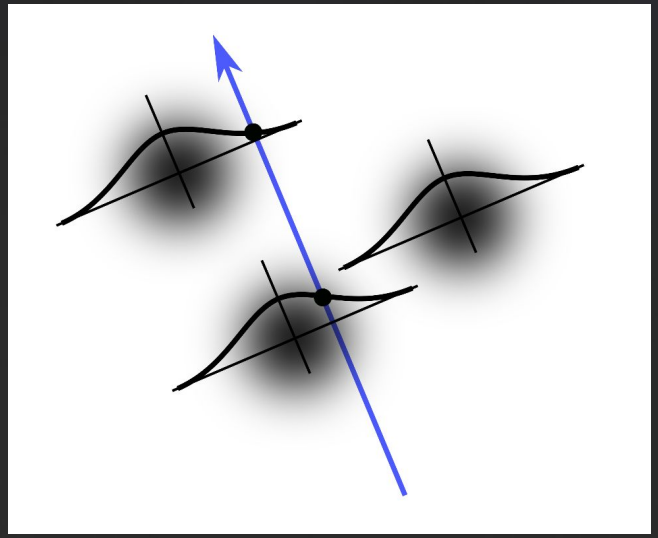
2D: $d = \sum \text{Gauss}(x, y)$
3D: $d = \sum \text{Gauss}(x, y, z)$

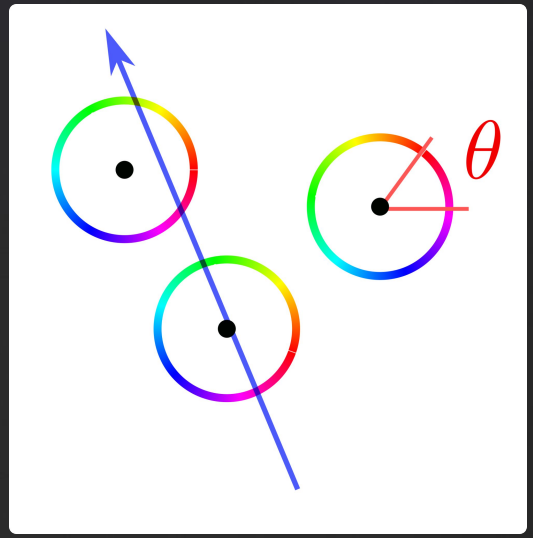
$c(\theta) = \sum$ circular harmx
$c(\theta, \phi) = \sum$ spherical h.

Recap and Terminology

Build density with Gaussians
Per-blob colour with harmonics

Approximate via splatting
"**Gaussian Splatting**", GS3D





2D: Evaluate each blob once
3D: Evaluate each blob once

“Splatting”? “Alpha blending”? Whence these terms?

“(Incomplete) History of Points”, a SIGGRAPH tutorial

Particle systems [Reeves]

Points as a display primitive [Whitted, Levoy]

Oriented particles [Szeliski, Tonnesen]

Particles and **implicit surfaces** [Witkin, Heckbert]

Surfels [Pfister et al.]

QSplat [Rusinkiewicz, Levoy]

Point Clouds [Linsen, Prautzsch]

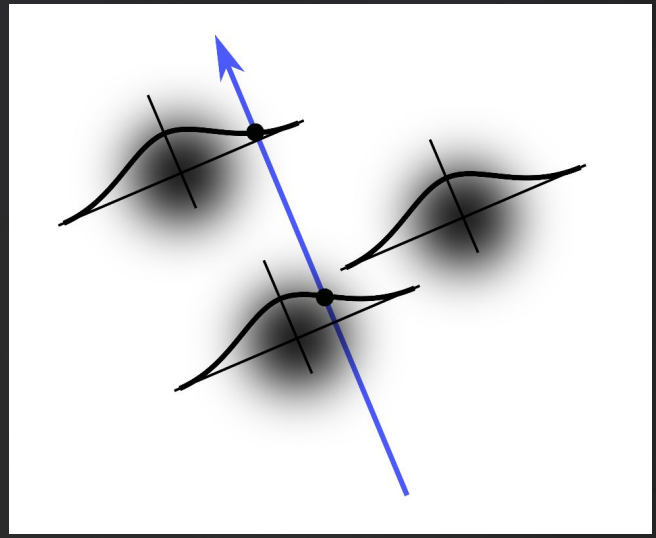
Point set surfaces [Alexa et al.]

Radial basis functions [Carr et al.]

Surface splatting [Zwicker et al.]

Opacity hulls [Matusik et al.]

Sequential Point Trees [Dachsbacher, Stamminger]



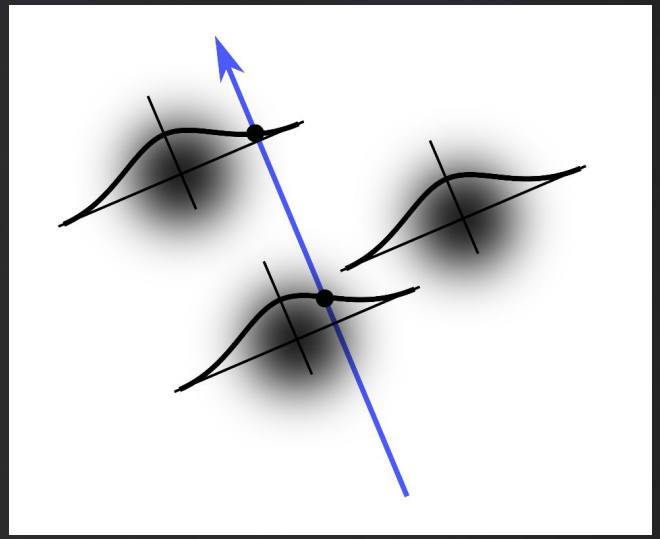
$$C_i = c \cdot d + C_{i-1} \cdot (1 - d)$$

GUESS THE YEAR

The Future Lies in the Past!

(Incomplete) History of Points, from a SIGGRAPH tutorial **2004**

Particle systems [Reeves **1983**]
Points as a display primitive [Whitted, Levoy 1985]
Oriented particles [Szeliski, Tonnesen 1992]
Particles and **implicit surfaces** [Witkin, Heckbert 1994]
Surfels [Pfister et al. 2000]
QSplat [Rusinkiewicz, Levoy 2000]
Point Clouds [Linsen, Prautzsch 2001]
Point set surfaces [Alexa et al. 2001]
Radial basis functions [Carr et al. 2001]
Surface splatting [Zwicker et al. 2001]
Opacity hulls [Matusik et al. 2002]
Sequential Point Trees [Dachsbacher, Stamminger 2003]



$$C_i = c \cdot d + C_{i-1} \cdot (1 - d)$$

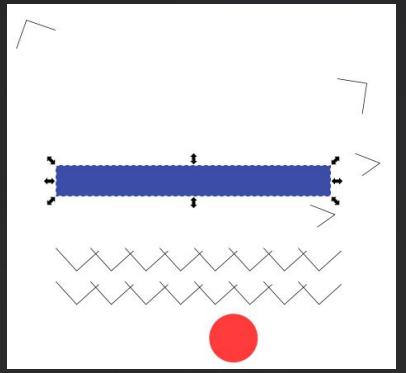
https://graphics.stanford.edu/~mapauly/Pdfs/SigCourse04.pdf
https://graphics.stanford.edu/papers/points/levoy-pointbook-ch2.pdf

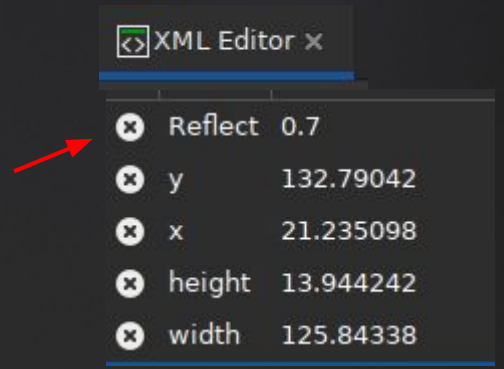
BONUS: A Study in Complex Appearance

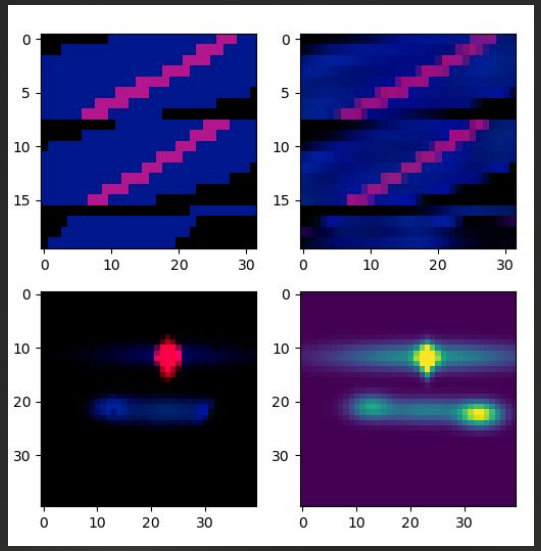
Let's add reflections to our renderer
And fit some radiance fields

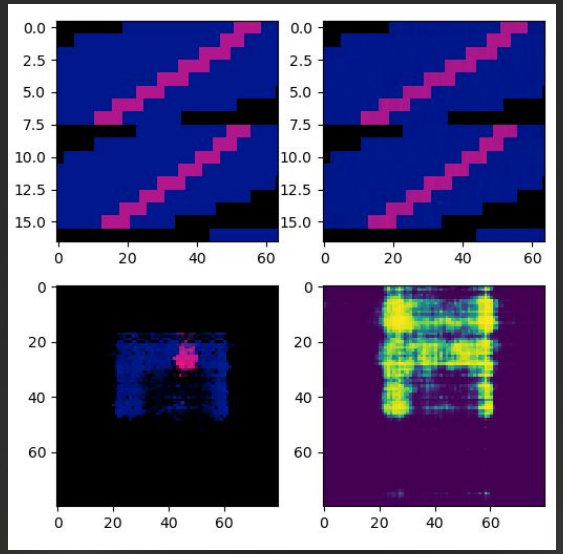
`scene_refl.svg`

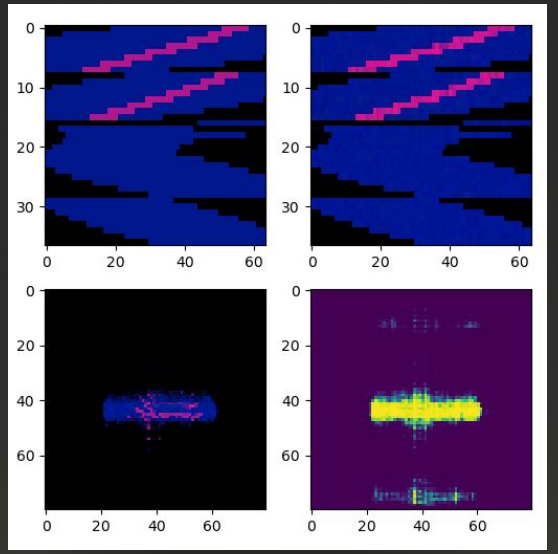
Broad range of behaviours:











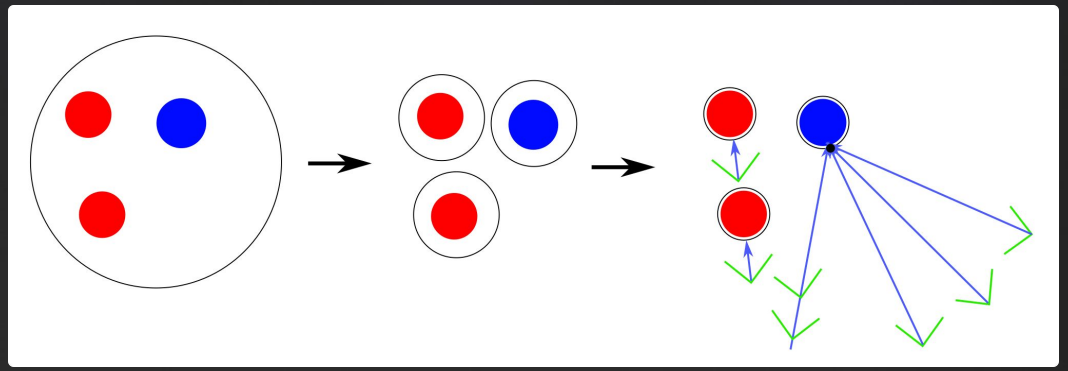
Radiance Fields are Under-Constrained

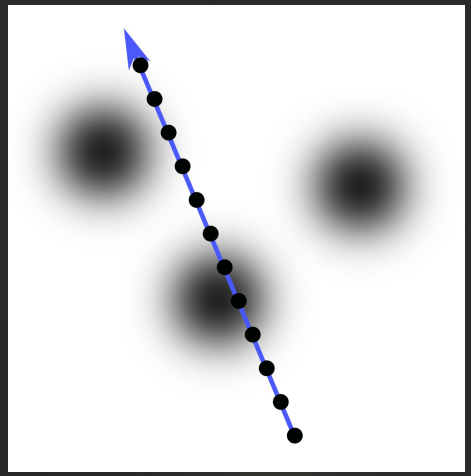
Radiance field is a distributed light field

Parameterisation is a free choice

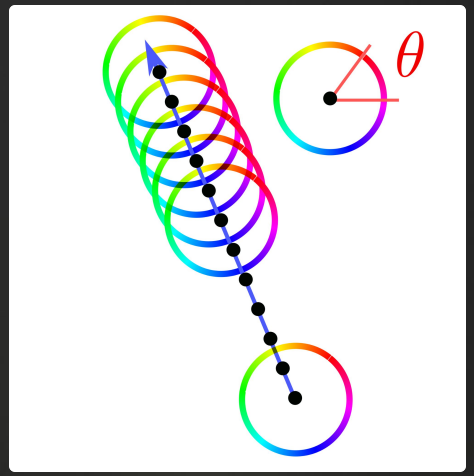
Some easier to train
Learning introduces inductive bias

Many radiance fields have identical appearance





Density



Colour

Checking in 3D

GS3D

NeRF

TODO: explore up-close in-browser GS3D: https://gsplat.tech/

Summary: What is a Radiance Field?

Representation of light

Decomposes the 5D plenoptic function into two pieces:

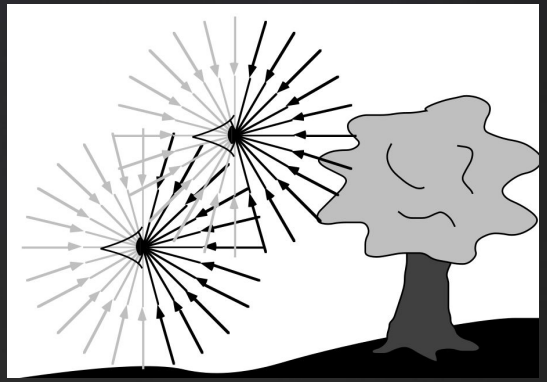
- Density in 3D space
- Colour in 2D direction

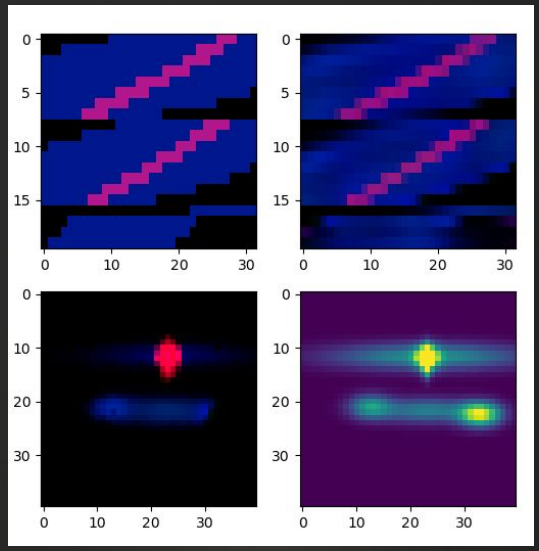
As a soft, distributed light field suitable for inverse rendering

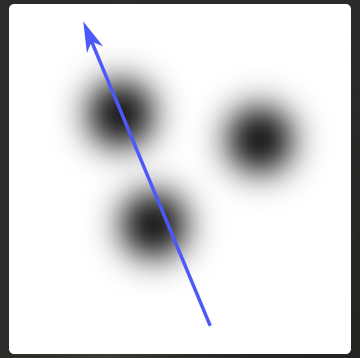
Natively handles complex appearance in a photo-realistic way!

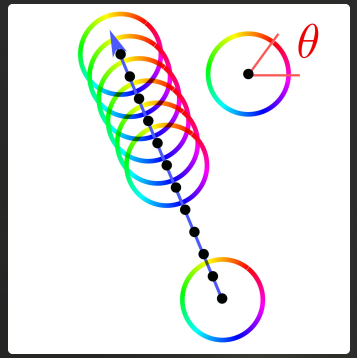
Density is free, visual parameterisation, not physical density

Many radiance fields have *identical* appearance









Summary: What are NeRF and 3DGS?

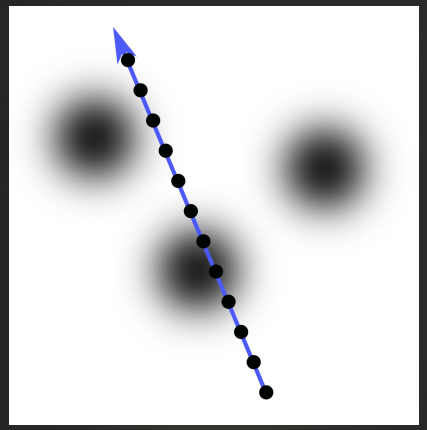
Inverse image-based renderers

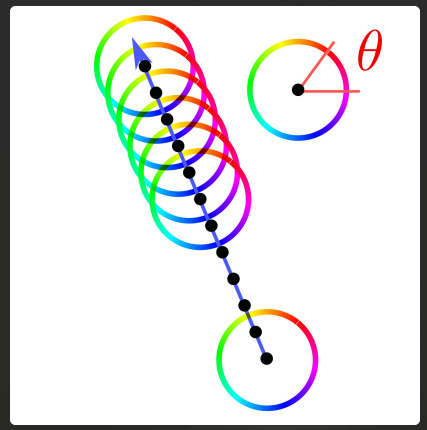
Approximate a radiance field to model the world

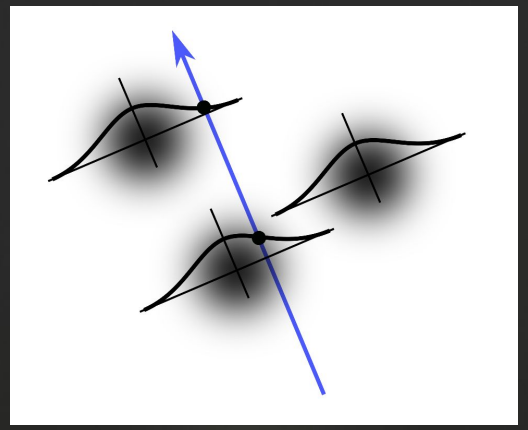
Via neural regression or by splatting Gaussian primitives

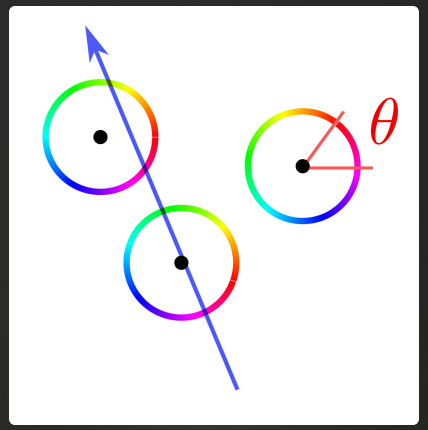
Natively handle complex appearance in a photo-realistic way!

Many NNs and splats yield *similar or identical* radiance fields









Questions and Further Reading

Exciting to summarise many images in a photo-realistic way… but:
How can I analyse what's in the model to understand the scene?
How do I know the camera poses?
Can I build incrementally?
How do I add new images?
How / when do I remove old images?
Should I keep the model and throw away (some?) images?
…

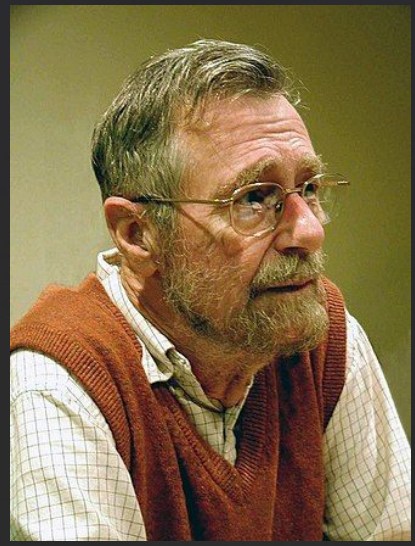
Freitas, Davi R., et al. "A Comparative Assessment Of Implicit and Explicit Plenoptic Scene Representations." 2024 IEEE 26th International Workshop on Multimedia Signal Processing (MMSP). IEEE, 2024.

A Parting Thought

"Computer science is no more about computers than astronomy is about telescopes."

-- Edsger Dijkstra

(maybe)



Elevates computer science to a legitimate science

Also elevates the role of computers and programs

- Your code is a scientific instrument
- Small, fast examples build understanding and intuition