# pixelSplat:

## 3D Gaussian Splats from Image Pairs for Scalable Generalizable 3D Reconstruction


Reviewer: Alberto Arkader Kopiler

Archeologist: Davi Guimarães

Hacker: Vitor Pereira Matias

PhD Student: Fernando Pereira de Sá

# Reviewer

Alberto Arkader Kopiler

# Introduction

- 3D reconstruction is one of the most popular research areas in computer vision

1) NeRF introduced the neural network to generate 3D renders.

1) pixelNeRF uses only a few images as input, and a CNN-based Encoder on top of NeRF to generate better 3D renders.

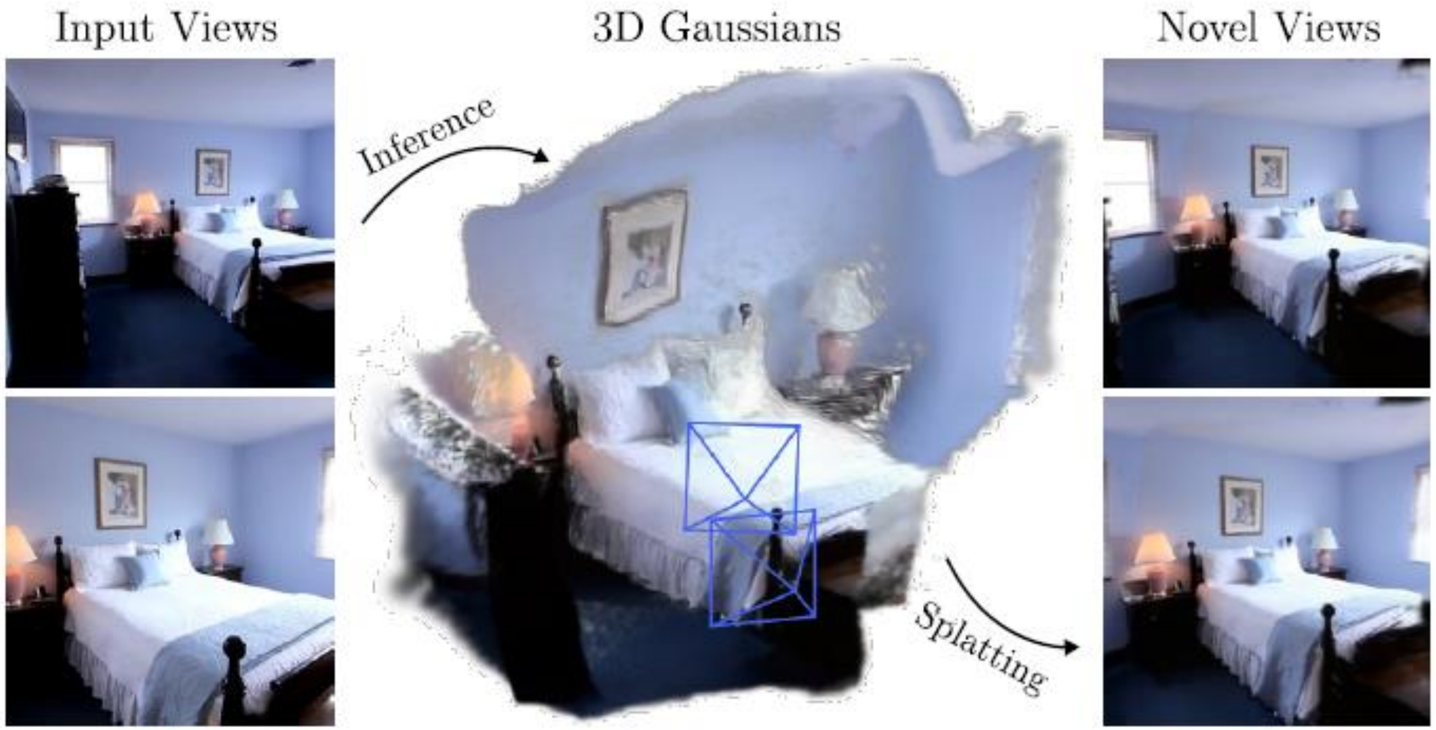
1) 3D Gaussian Splatting uses 3D Gaussian and gradient descent to generate better 3D renders than priors.

1) pixelSplat combines 3D Gaussian splatting with a reparameterization trick and a neural network

- Input two images of an object from two different viewpoints and generates a 3D render within minimal inference time. It is like a combination of 3D Gaussian Splatting and NeRF.

# Summary

- A feed-forward model that learns to reconstruct 3D radiance fields parameterized by 3D Gaussian primitives from pairs of images.




- This results in an explicit 3D representation that is renderable in real time, remains editable, and is cheap to train.

# Model’s Evolution

## Differentiable Rendering

- training, reconstruction, and rendering are memory- and time-intensive
- because requires evaluating dozens or hundreds of points along each camera ray

## Light Field Transformers

- where a ray is rendered by embedding it into a query token and a color is obtained via cross-attention over image tokens.
- do not reconstruct 3D scene representations that can be edited or exported for downstream tasks in vision and graphics
- faster than volume rendering, although far from real-time

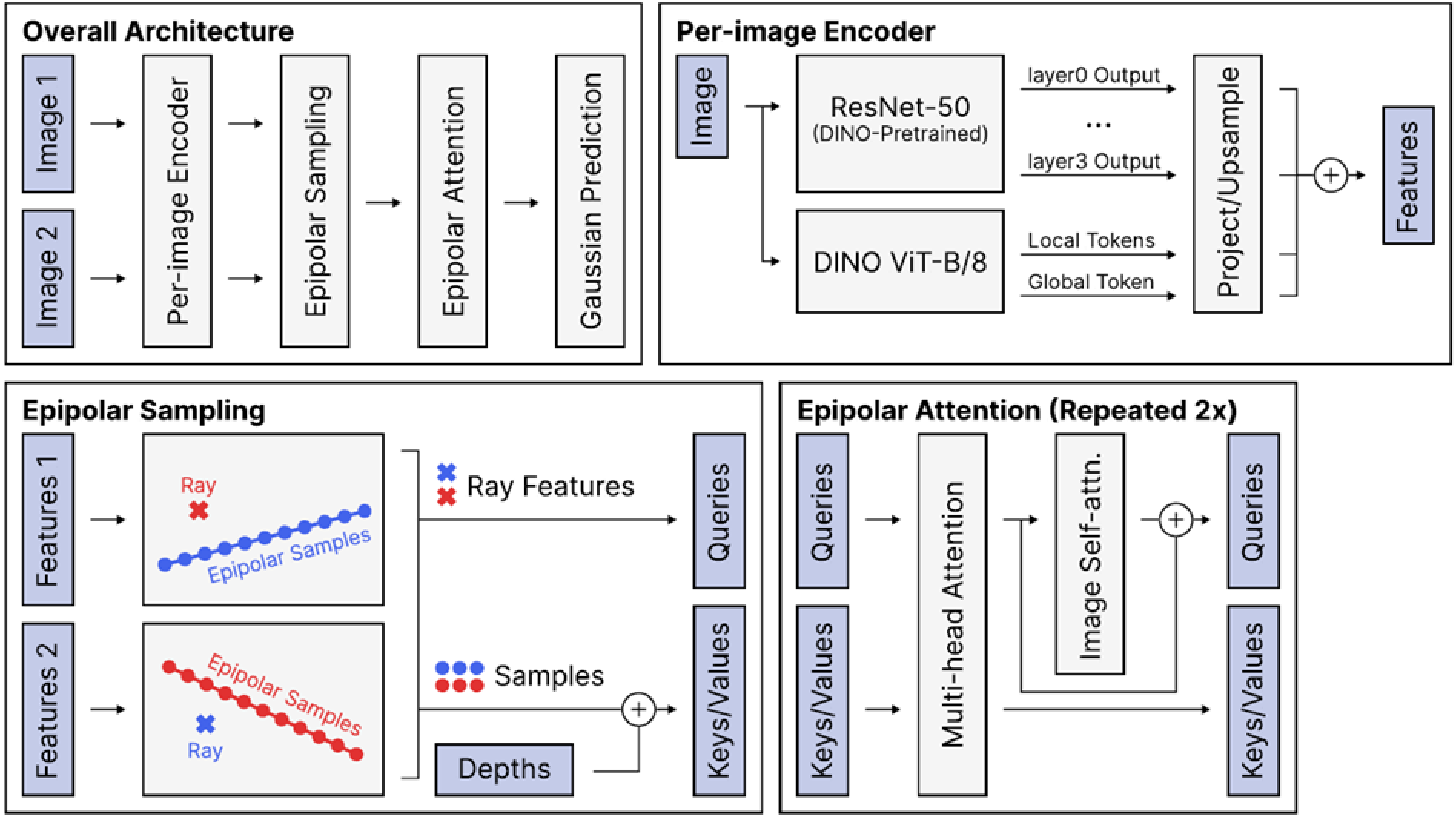
## **Rasterization-based Volume Rendering**

- Fast and memory efficient
- Real-time

## pixelSplat

- benefits of a primitive-based 3D representation: fast and memory efficient
- rendering as well as interpretable 3D structure
- to generalizable view synthesis

# Architecture Diagram

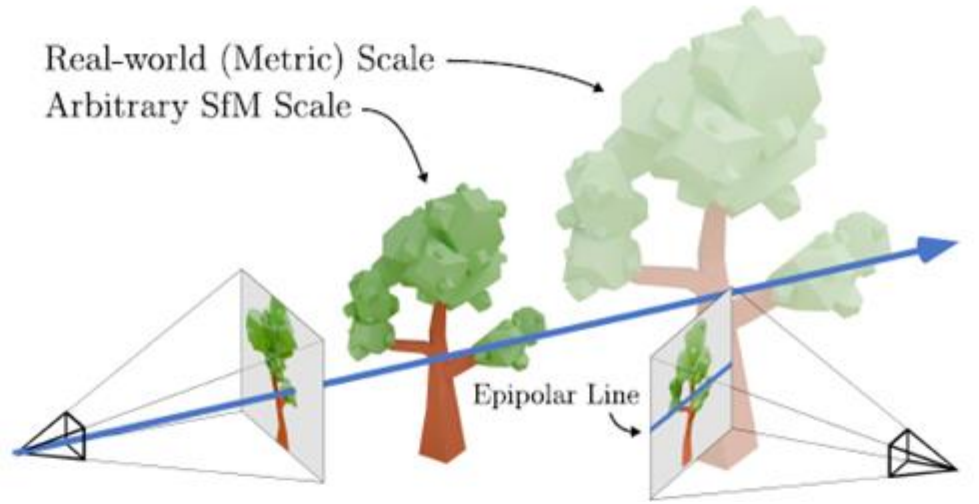


## Two-View Image Encoding

- PixelSplat begins by processing a pair of input images through a feature extraction network, which generates a high-dimensional representation of each image. This neural network, often structured similarly to those used in NeRF architectures, extracts crucial visual and spatial features from the images, setting the stage for understanding the scene's geometry.

## Epipolar Geometry and Scale Ambiguity Resolution

- The extracted features are then processed using an epipolar transformer, a component that leverages the geometric relationship between the two views to resolve scale ambiguity—an inherent challenge in reconstructing 3D scenes from 2D images. This step ensures that the 3D positions inferred from different images are consistent relative to each other, addressing variations in camera positioning and orientation.



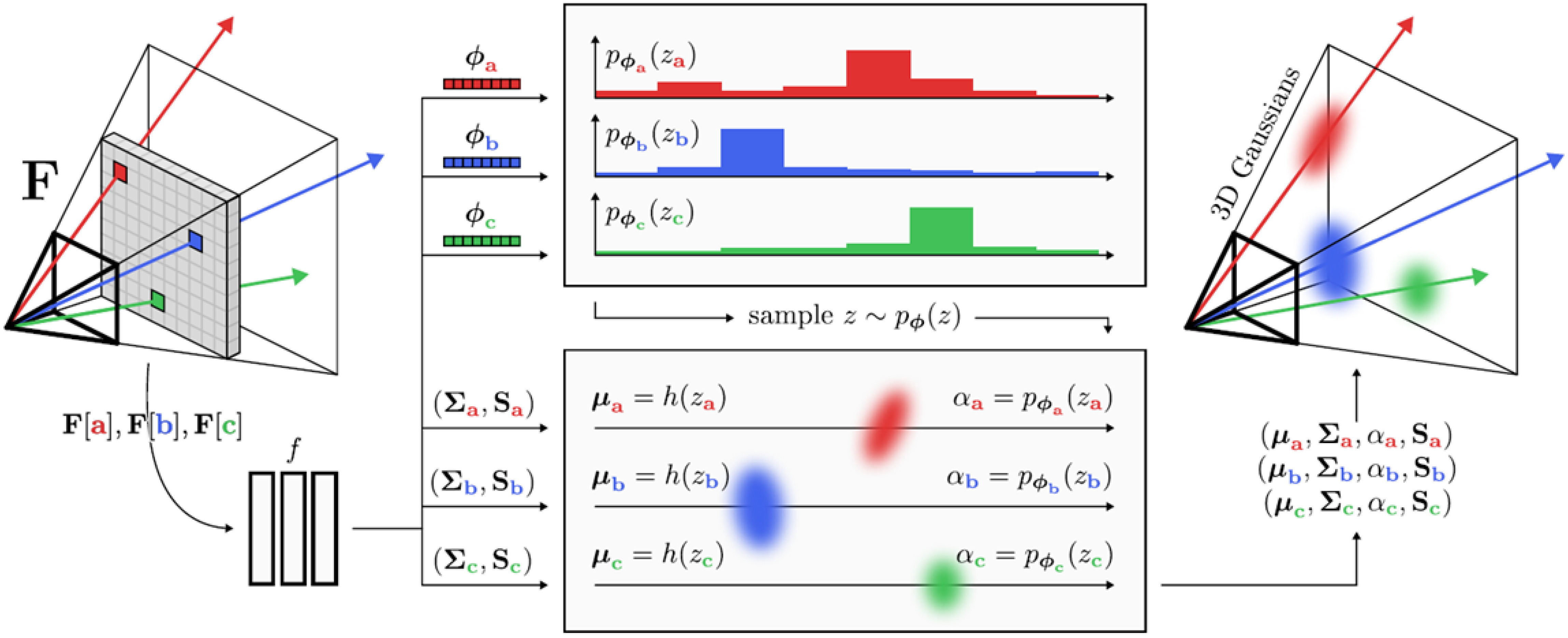
## Probabilistic Sampling of Gaussian Parameters

- With scale and geometry calibrated, the next step involves a novel application of 3D Gaussian splatting, where the model predicts a dense probability distribution for the potential locations of Gaussian primitives. This approach is facilitated by the reparameterization trick, which allows the network to sample these locations differently. Here, each Gaussian's position (mean), shape (covariance), and visibility (opacity) are determined, enabling gradients to be propagated back through the network during training, thus optimizing the Gaussian placement efficiently.

# Two-View Image Encoding

- Rendering and Output Generation: Finally, the parameterized 3D scene, now represented as a collection of Gaussian splats, is rendered to produce novel views. This rendering process is optimized for speed and memory efficiency, making use of the Gaussian splatting technique’s light computational footprint. The output is a set of new images, or novel views, generated from perspectives not originally captured by the input images, showcasing the model’s ability to interpolate and extrapolate 3D space from limited data.

# Proposed probabilistic prediction of pixel-aligned Gaussians



# Probabilistic prediction of pixel-aligned Gaussians

**Algorithm 1** Probabilistic Prediction of a Pixel-Aligned Gaussian.

**Require:** Depth buckets $\mathbf{b} \in \mathbb{R}^Z$, feature $\mathbf{F}[\mathbf{u}]$ at pixel coordinate $\mathbf{u}$, camera origin of reference view $\mathbf{o}$, ray direction $\mathbf{d_u}$.

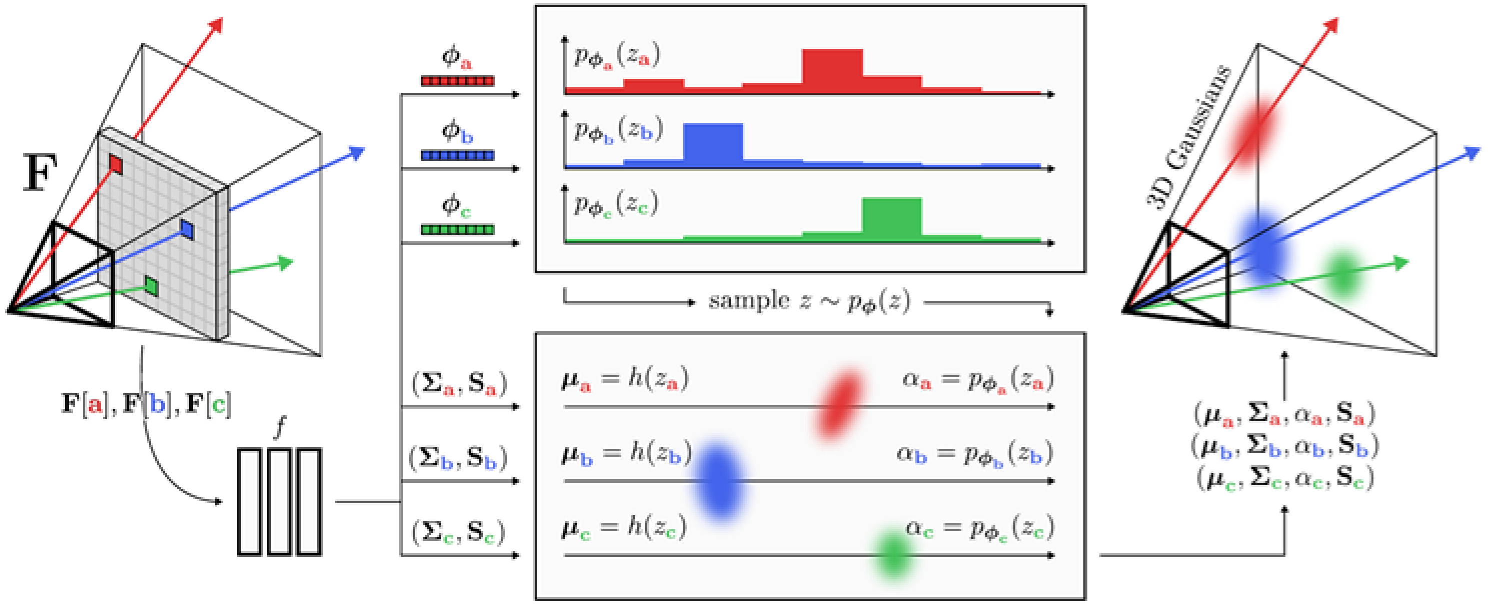
$(\boldsymbol{\phi}, \boldsymbol{\delta}, \boldsymbol{\Sigma}, \mathbf{S}) = f(\mathbf{F}[\mathbf{u}])$ ▷ predict depth probabilities $\boldsymbol{\phi}$ and offsets $\boldsymbol{\delta}$, covariance $\boldsymbol{\Sigma}$, spherical harmonics coefficients $\mathbf{S}$

$z \sim p_{\boldsymbol{\phi}}(z)$ ▷ Sample depth bucket index $z$ from discrete probability distribution parameterized by $\boldsymbol{\phi}$

$\boldsymbol{\mu} = \mathbf{o} + (\mathbf{b}_z + \boldsymbol{\delta}_z)\mathbf{d_u}$ ▷ Compute Gaussian mean $\boldsymbol{\mu}$ by unprojecting with depth $\mathbf{b}_z$ adjusted by bucket offset $\boldsymbol{\delta}_z$

$\boldsymbol{\alpha} = \boldsymbol{\phi}_z$ ▷ Set Gaussian opacity $\boldsymbol{\alpha}$ according to probability of sampled depth (Sec. 4.2).

**return** $(\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\alpha}, \mathbf{S})$



# Quantitative Comparison

| | ACID | | | RealEstate10k | | | Inference Time (s) | | Memory (GB) | |
|---|---|---|---|---|---|---|---|---|---|---|
| | PSNR ↑ | SSIM ↑ | LPIPS ↓ | PSNR ↑ | SSIM ↑ | LPIPS ↓ | Encode ↓ | Render ↓ | Training ↓ | Inference ↓ |
| Ours | **28.27** | **0.843** | **0.146** | **26.09** | **0.863** | **0.136** | 0.102 | **0.002** | **14.4** | **3.002** |
| Du et al. [10] | 26.88 | 0.799 | 0.218 | 24.78 | 0.820 | 0.213 | 0.016 | 1.309 | 314.3 | 19.604 |
| GPNR [46] | 25.28 | 0.764 | 0.332 | 24.11 | 0.793 | 0.255 | **N/A** | 13.340 | 3789.9 | 19.441 |
| pixelNeRF [58] | 20.97 | 0.547 | 0.533 | 20.43 | 0.589 | 0.550 | 0.005 | 5.294 | 436.7 | 3.962 |

# Qualitative Comparison

# Ablation

# Attention Visualization

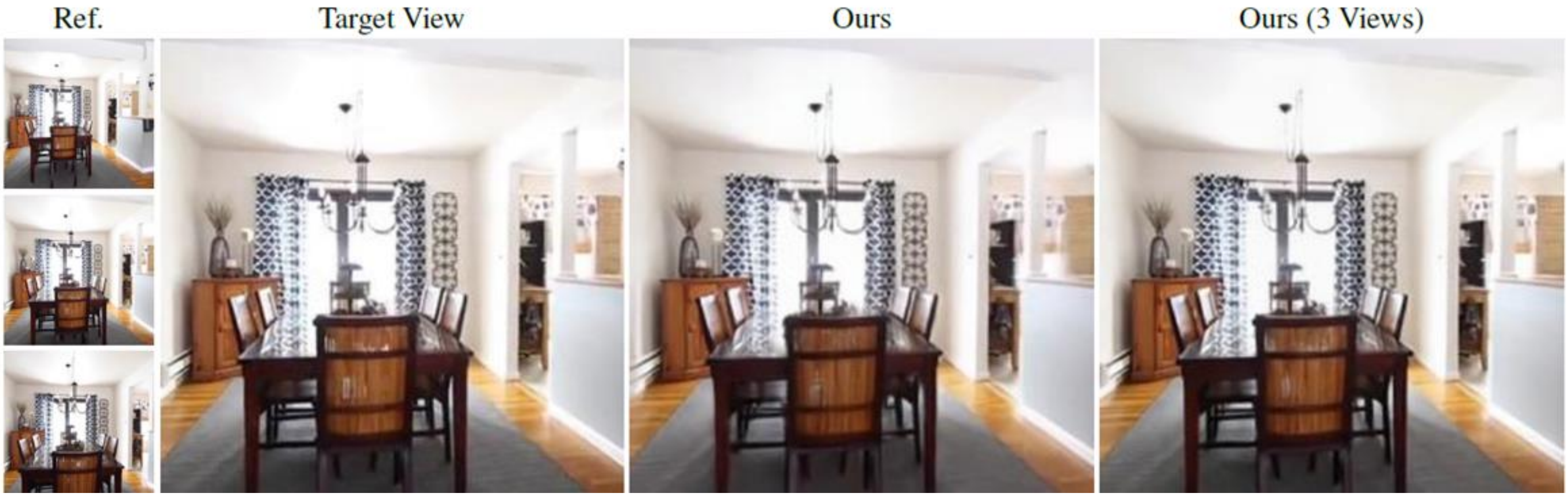
# Limitations

- Rather than fusing or de-duplicating Gaussians observed from both reference views, it simply outputs the union of the Gaussians predicted from each view.

- it does not address generative modelling of unseen parts of the scene.

- When extended to many reference views, their epipolar attention mechanism becomes prohibitively expensive in terms of memory.

# Additional Results



# Conclusions

- It is an original approach to the problem of with only two input images taken from different points of view, synthetize novel views.

- it uses a pipeline of pre-image encoder, followed by epipolar sampling, epipolar attention and gaussian prediction.

- They claim that their work at inference time is significantly faster than prior work on generalizable novel view synthesis while producing an explicit 3D scene representation.

- They claim that to solve the problem of local minima that arises in primitive-based function regression, they introduced a novel parameterization of primitive location via a dense probability distribution and introduced a novel reparameterization trick to backpropagate gradients into the parameters of this distribution.

# Conclusions

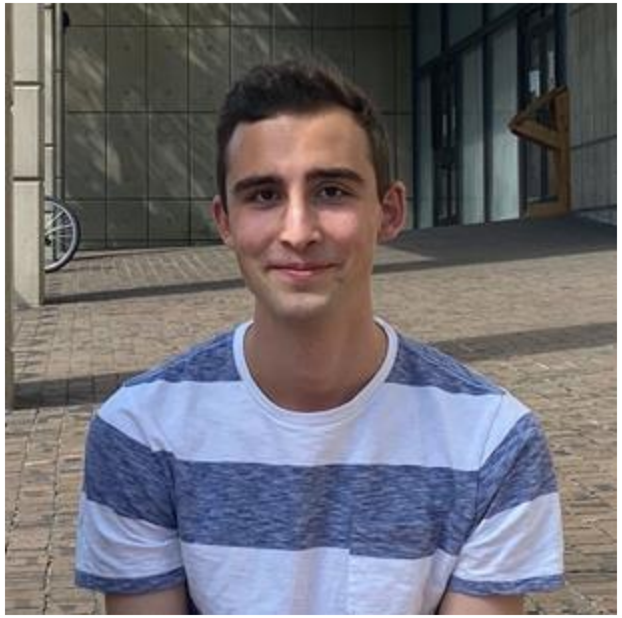
- They claim that their framework is general, and they hope that their work inspires follow-up work on prior-based inference of primitive-based representations across applications.
- They suggest for future to leverage their model for generative modelling by combining it with diffusion models or to remove the need for camera poses to enable large-scale training.
- Their model resolve scale ambiguity.
- Strongly accept.

# Archeologist

Davi Guimarães

# Um pouco sobre os autores

Vincent Sitzmann



David Charatan

Sizhe Li

Andrea Tagliasacchi

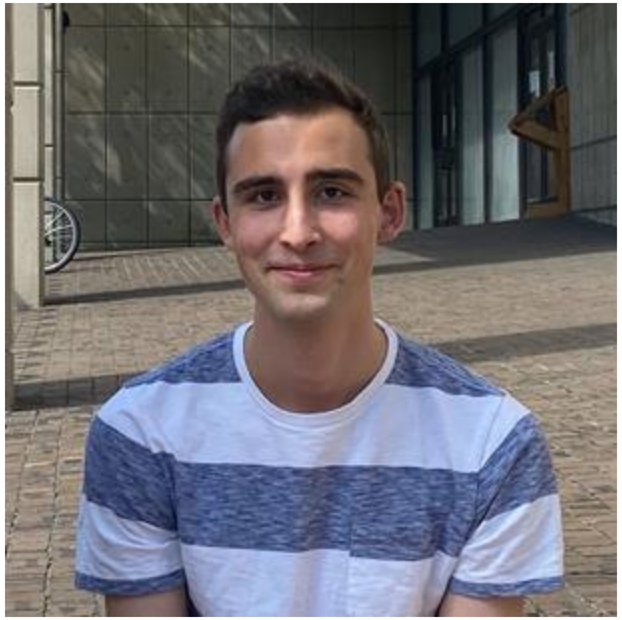
# Vincent Sitzmann

Professor assistente atuando no MIT EECS e liderando o Scene Representation Group.

Fez seu doutorado em Stanford e pós-doutorado em MIT CSAIL.

Possui 3 papers aceitos no CVPR e 10 aceitos no NeurIPS. Também é um dos autores do paper Flowmap.

# David Charatan



Aluno de doutorado no MIT EECS.

Um dos autores do FlowMap. 1 paper aceito no CVPR (PixelSplat), 1 paper aceito no SIGGRAPH e 1 paper aceito no 3DV.

# Sizhe Li

Aluno de doutorado no MIT CSAIL.

1 paper aceito no CVPR (PixelSplat) e 2 aceitos no ICLR.

# Andrea Tagliasacchi

Atua como professor auxiliar na universidade Simon Fraser, cientista pesquisador do Google DeepMind e professor auxiliar no departamento de ciência da computação da universidade de Toronto.

Dentre seus mentores, está Geoffrey Hinton, ganhador do Nobel da física deste ano junto com John Hopfield.

Andrea possui 28 papers aceitos no CVPR, 5 papers aceitos na ECCV e 7 papers aceitos na NeurIPS.

Tendo ganhado o SGP best paper award de 2015, o CVPR best student paper award de 2020, e o CVPR best paper award de 2024.

# Contexto

Que problema exatamente eles estavam tentando resolver?

# Prior-based 3D Reconstruction and View Synthesis

Reconstruções 3D de qualidade de uma cena próxima da câmera usando poucas imagens já estavam sendo feitas.

Reconstruções 3D de boa qualidade não limitadas pela distância da cena e a câmera eram difíceis de serem feitas com poucas imagens.

**Single-View View Synthesis with Multiplane Images**

Richard Tucker Noah Snavely
Google Research

**Pushing the Boundaries of View Extrapolation with Multiplane Images**

Pratul P. Srinivasan[1] Richard Tucker[2] Jonathan T. Barron[2]
Ravi Ramamoorthi[3] Ren Ng[1] Noah Snavely[2]
[1]UC Berkeley, [2]Google Research, [3]UC San Diego

# Prior-based 3D Reconstruction and View Synthesis

Preservar a localidade end-to-end e a equivariância de deslocamento entre o encoder e a representação de cena por meio de pixel-aligned features e transformers, possibilitou a generalização de cenas ilimitadas.

**IBRNet: Learning Multi-View Image-Based Rendering**

Qianqian Wang[1,2] Zhicheng Wang[1] Kyle Genova[1,3] Pratul Srinivasan[1] Howard Zhou[1]
Jonathan T. Barron[1] Ricardo Martin-Brualla[1] Noah Snavely[1,2] Thomas Funkhouser[1,3]
[1]Google Research [2]Cornell Tech, Cornell University [3]Princeton University

**pixelNeRF: Neural Radiance Fields from One or Few Images**

Alex Yu Vickie Ye Matthew Tancik Angjoo Kanazawa
UC Berkeley

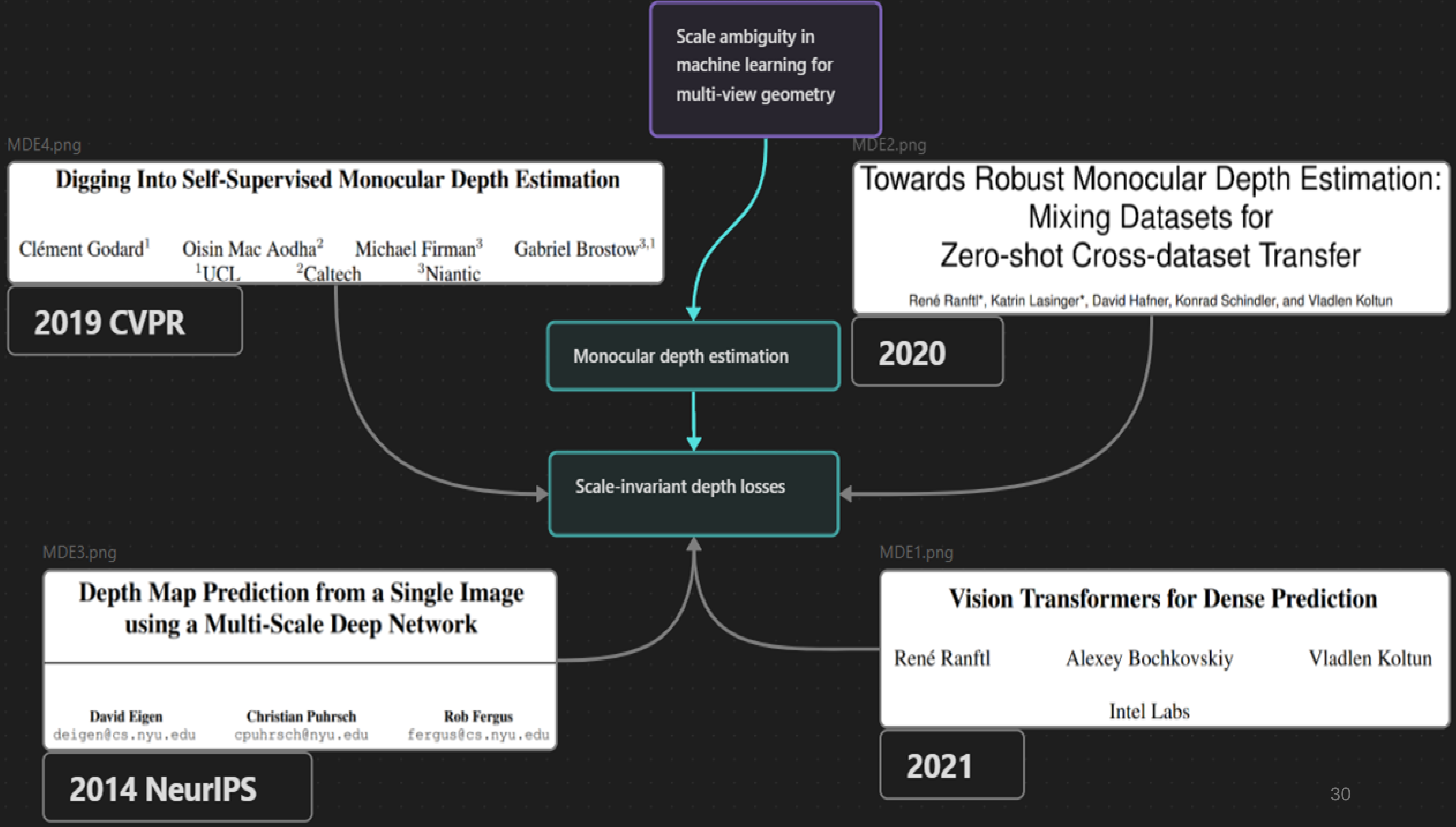
# Prior-based 3D Reconstruction and View Synthesis

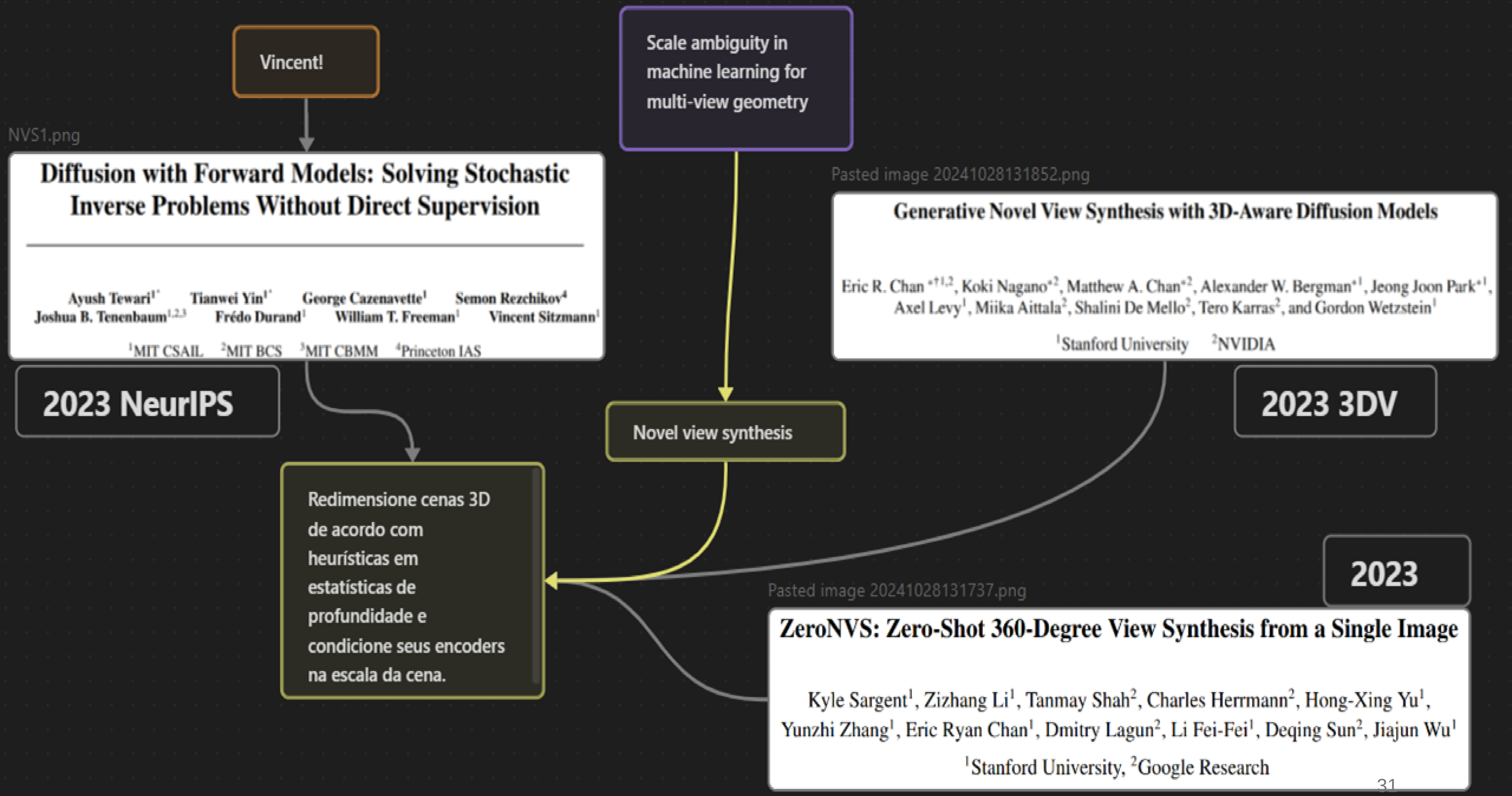
**Cost volume**

MVSNeRF
Stereo Radiance Fields
GeoNeRF

**Light field scene representation**

Scene Representation Transformer
Light Field Networks
Light Field Neural Rendering


Scale ambiguity in machine learning for multi-view geometry
MDE4.png
Digging Into Self-Supervised Monocular Depth Estimation
Clément Godard[1] Oisin Mac Aodha[2] Michael Firman[3] Gabriel Brostow[3,1]
[1]UCL [2]Caltech [3]Niantic
2019 CVPR
MDE2.png
Towards Robust Monocular Depth Estimation: Mixing Datasets for Zero-shot Cross-dataset Transfer
René Ranftl*, Katrin Lasinger*, David Hafner, Konrad Schindler, and Vladlen Koltun
2020
Monocular depth estimation
Scale-invariant depth losses
MDE3.png
Depth Map Prediction from a Single Image using a Multi-Scale Deep Network
David Eigen deigen@cs.nyu.edu
Christian Puhrsch cpuhrsch@nyu.edu
Rob Fergus fergus@cs.nyu.edu
2014 NeurIPS
MDE1.png
Vision Transformers for Dense Prediction
René Ranftl Alexey Bochkovskiy Vladlen Koltun
Intel Labs
2021


Vincent!
Scale ambiguity in machine learning for multi-view geometry
NVS1.png
Diffusion with Forward Models: Solving Stochastic Inverse Problems Without Direct Supervision
Ayush Tewari[1*] Tianwei Yin[1*] George Cazenavette[1] Semon Rezchikov[4]
Joshua B. Tenenbaum[1,2,3] Frédo Durand[1] William T. Freeman[1] Vincent Sitzmann[1]
[1]MIT CSAIL [2]MIT BCS [3]MIT CBMM [4]Princeton IAS
2023 NeurIPS
Pasted image 20241028131852.png
Generative Novel View Synthesis with 3D-Aware Diffusion Models
Eric R. Chan [*†1,2], Koki Nagano[*2], Matthew A. Chan[*2], Alexander W. Bergman[*1], Jeong Joon Park[*1],
Axel Levy[1], Miika Aittala[2], Shalini De Mello[2], Tero Karras[2], and Gordon Wetzstein[1]
[1]Stanford University [2]NVIDIA
2023 3DV
Novel view synthesis
Redimensione cenas 3D de acordo com heurísticas em estatísticas de profundidade e condicione seus encoders na escala da cena.
2023
Pasted image 20241028131737.png
ZeroNVS: Zero-Shot 360-Degree View Synthesis from a Single Image
Kyle Sargent[1], Zizhang Li[1], Tanmay Shah[2], Charles Herrmann[2], Hong-Xing Yu[1],
Yunzhi Zhang[1], Eric Ryan Chan[1], Dmitry Lagun[2], Li Fei-Fei[1], Deqing Sun[2], Jiajun Wu[1]
[1]Stanford University, [2]Google Research

Scale ambiguity in machine learning for multi-view geometry

ET.png

# Epipolar Transformers

Yihui He* Rui Yan* Katerina Fragkiadaki
Carnegie Mellon University
Pittsburgh, PA 15213

Shoou-I Yu
Facebook Reality Labs
Pittsburgh, PA 15213

2020 CVPR

PixelSplat

Construímos um multi-view encoder que pode inferir a escala da cena usando um transformador epipolar.

# Artigos subsequentes

## LGM: Large Multi-View Gaussian Model for High-Resolution 3D Content Creation

ECCV 2024 (Oral)

Jiaxiang Tang[1], Zhaoxi Chen[2], Xiaokang Chen[1], Tengfei Wang[3], Gang Zeng[1], Ziwei Liu[2]

[1] Peking University [2] S-Lab, Nanyang Technological University [3] Shanghai AI Lab

## GS-LRM: Large Reconstruction Model for 3D Gaussian Splatting

Kai Zhang[*1], Sai Bi[*1], Hao Tan[*1], Yuanbo Xiangli[2], Nanxuan Zhao[1], Kalyan Sunkavalli[1], Zexiang Xu[1]

[*](Equal contribution)

[1]Adobe Research [2]Cornell University

Outros papers que tratam do mesmo problema, reconstrução de cenas 3D com poucas imagens.

Esses dois papers ainda possuem o ponto forte de resolverem esse problema sem a necessidade da posição das câmeras.

**No Pose, No Problem: Surprisingly Simple 3D Gaussian Splats from Sparse Unposed Images**

*ICLR 2025 Conference Submission3116 Authors*

**DUSt3R: Geometric 3D Vision Made Easy**

Shuzhe Wang, Vincent Leroy, Yohann Cabon, Boris Chidlovskii, Jérome Revaud

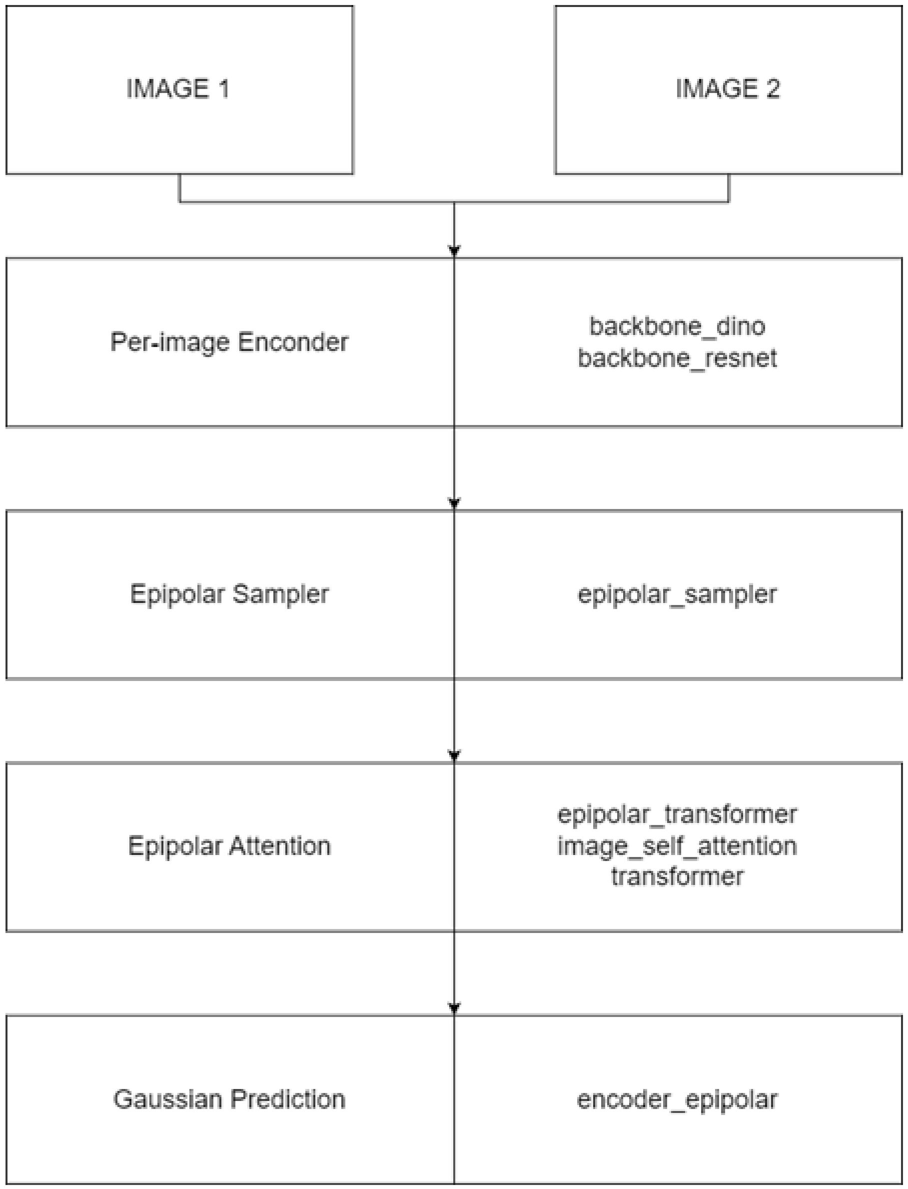
The IEEE / CVF Computer Vision and Pattern Recognition Conference (CVPR), Seattle, USA, 17-21 June, 2024

# Hacker

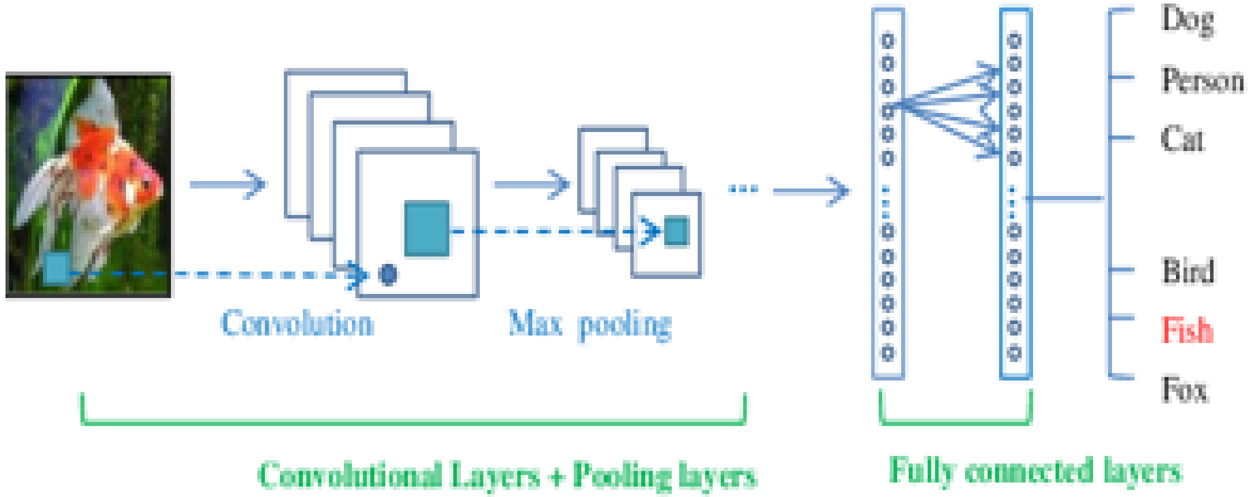
Vitor Pereira Matias

# General code

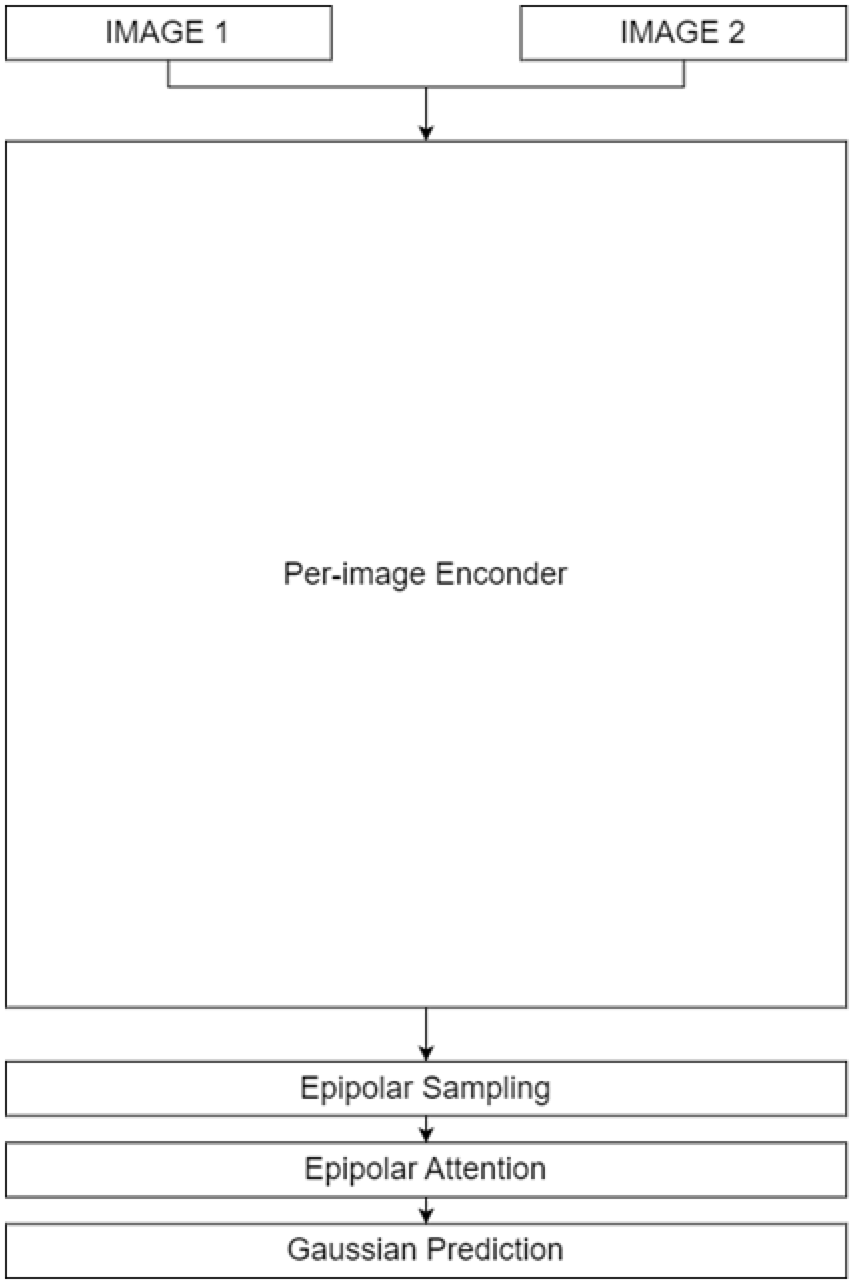
- on the image:
  - left: method architecture
  - right: .py files of that method



# Image Features

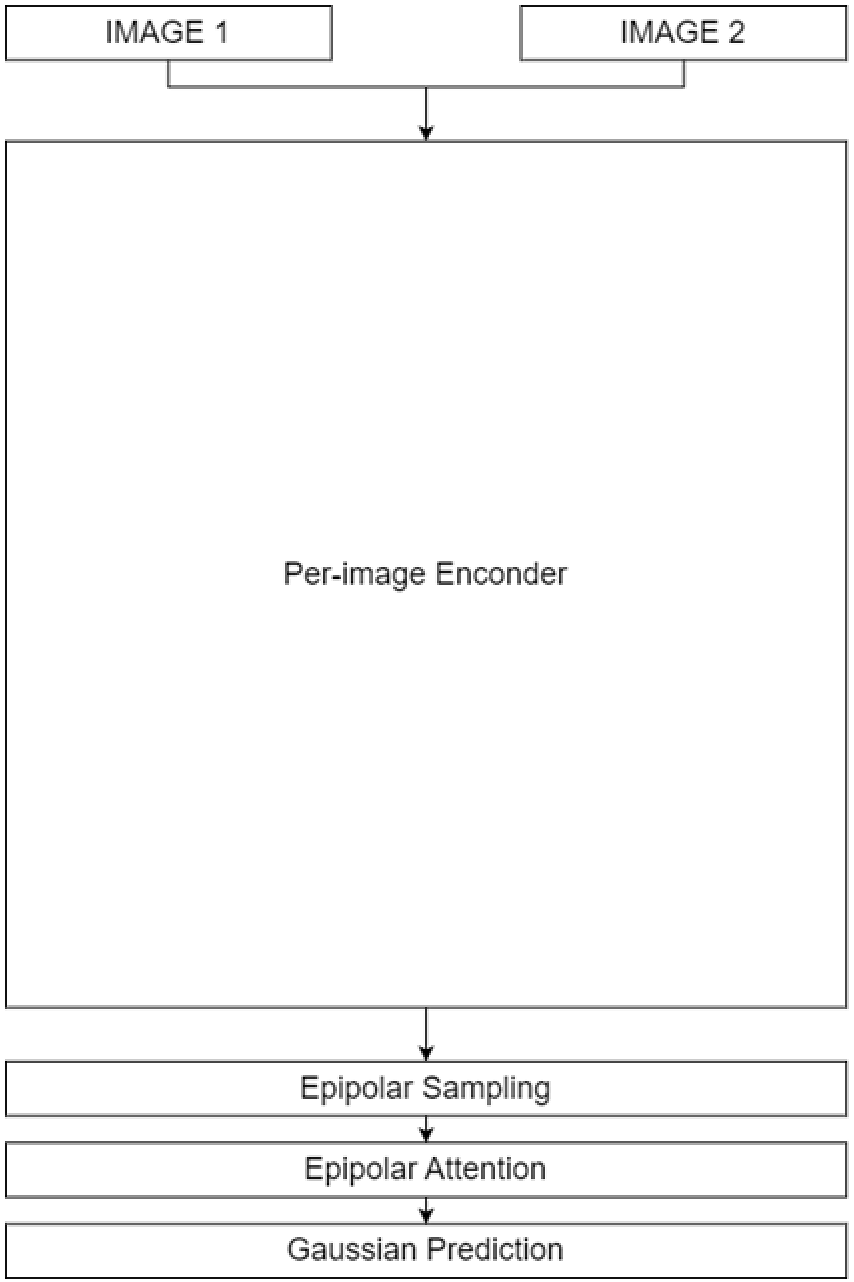



Design of Convolution Neural Network

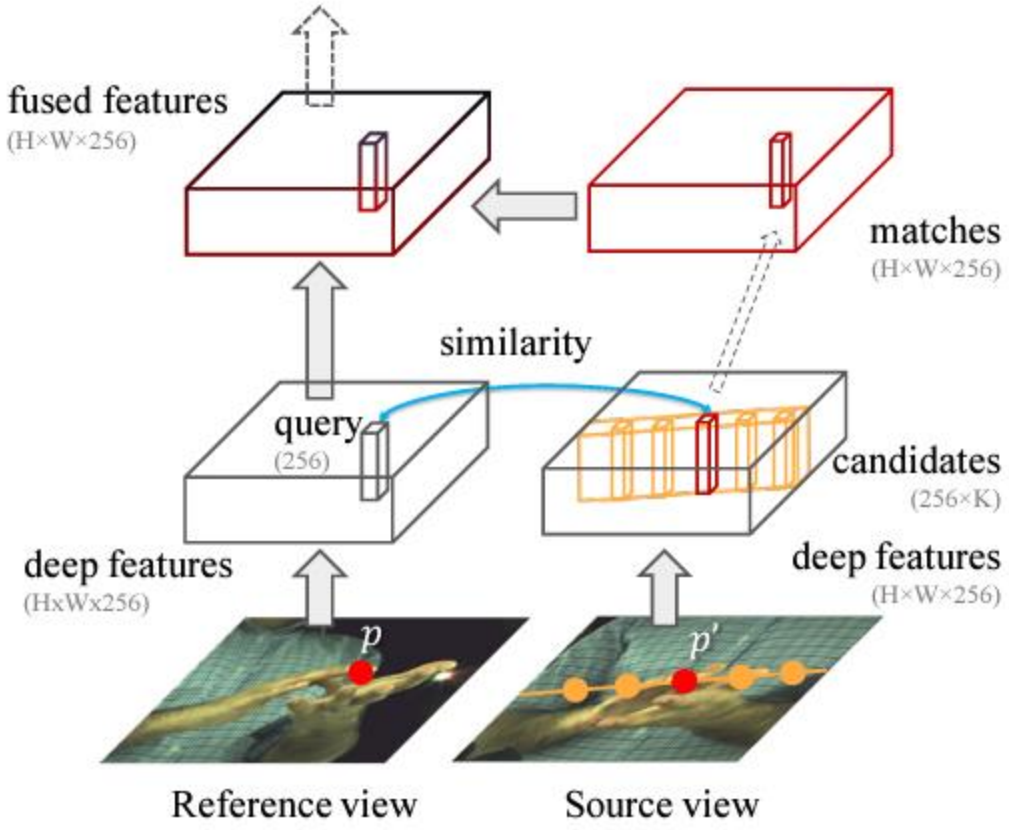


# Image Features

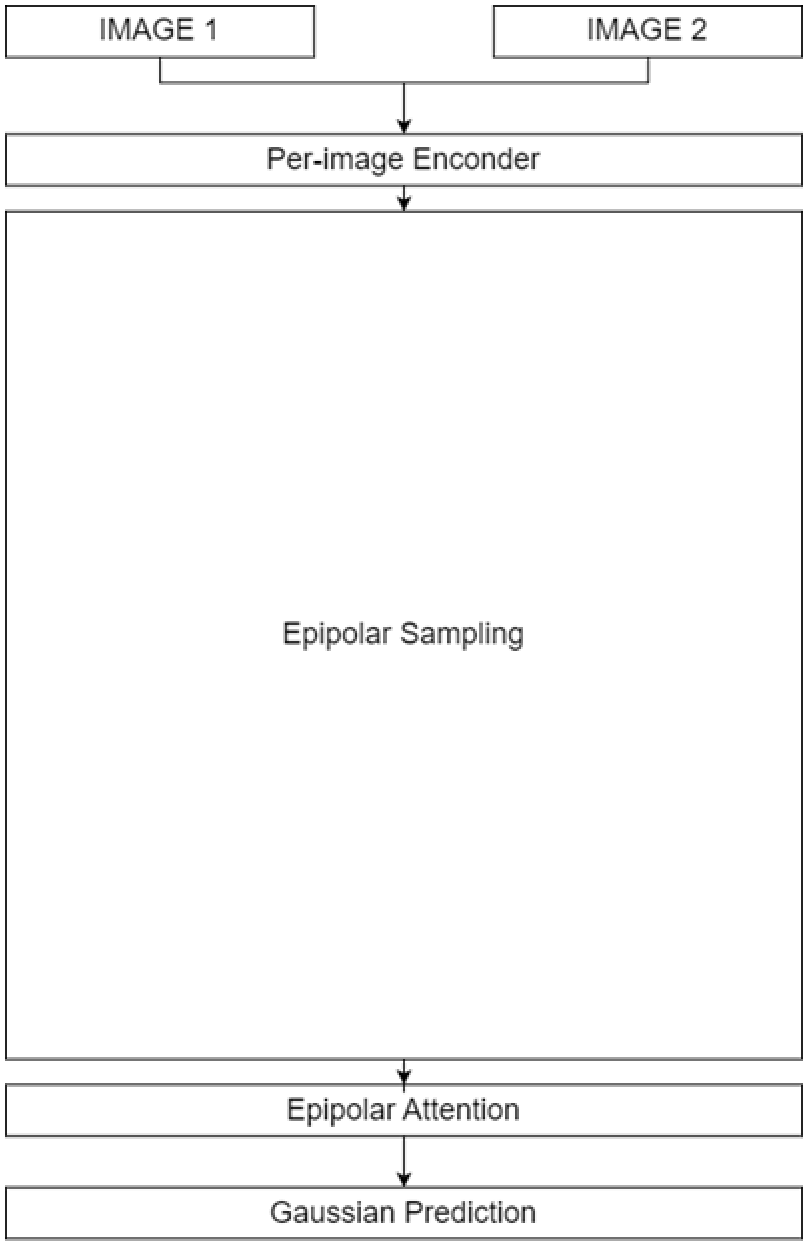
```python
class BackboneDino(Backbone[BackboneDinoCfg]):
    def __init__(self, cfg: BackboneDinoCfg, d_in: int) -> None:···

    def forward(        David Charatan, 10 months ago • Release code.
        self,
        context: BatchedViews,
    ) -> Float[Tensor, "batch view d_out height width"]:
        # Compute features from the DINO-pretrained resnet50.
        resnet_features = self.resnet_backbone(context)

        # Compute features from the DINO-pretrained ViT.
        b, v, _, h, w = context["image"].shape
        assert h % self.patch_size == 0 and w % self.patch_size == 0
        tokens = rearrange(context["image"], "b v c h w -> (b v) c h w")
        tokens = self.dino.get_intermediate_layers(tokens)[0]
        global_token = self.global_token_mlp(tokens[:, 0])
        local_tokens = self.local_token_mlp(tokens[:, 1:])

        # Repeat the global token to match the image shape.
        global_token = repeat(global_token, "(b v) c -> b v c h w", b=b,
        h=h, w=w)

        # Repeat the local tokens to match the image shape.
        local_tokens = repeat(···

        return resnet_features + local_tokens + global_token
```



# Epipolar Samples from rays

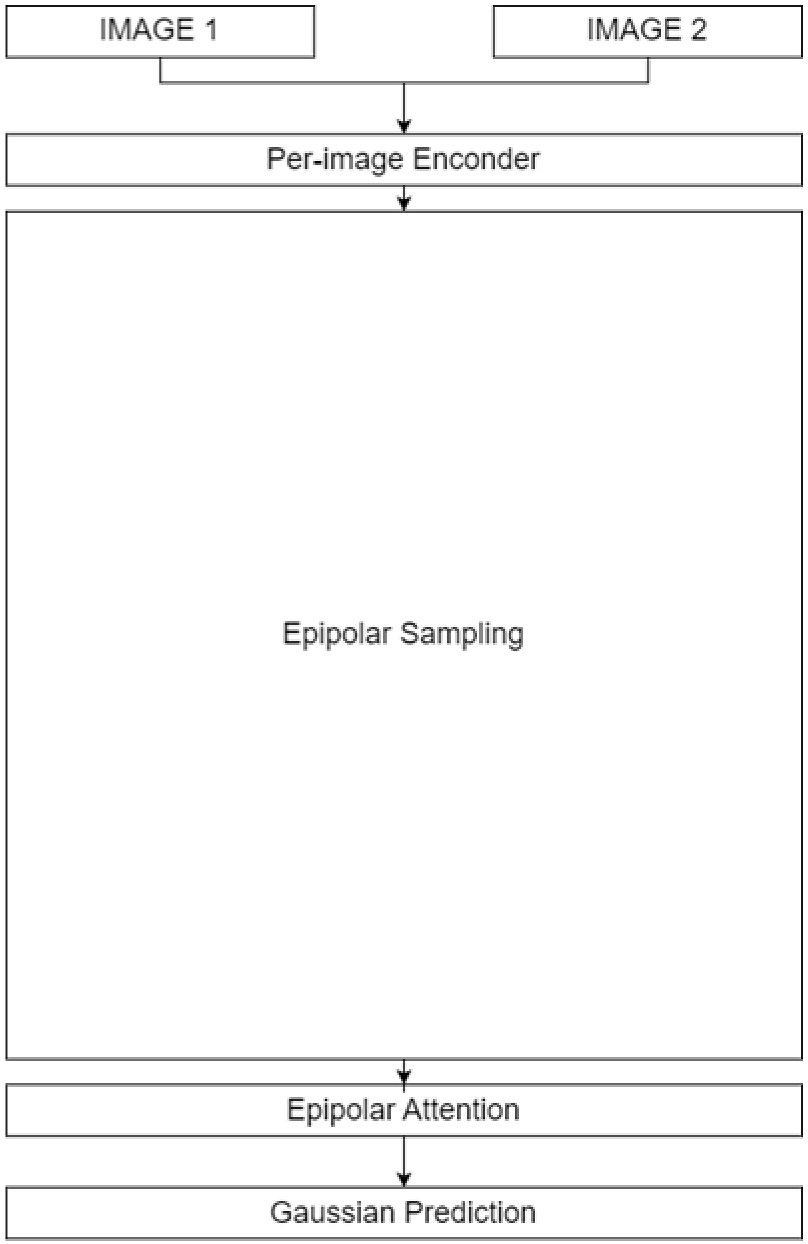



- source paper: Epipolar Transformer



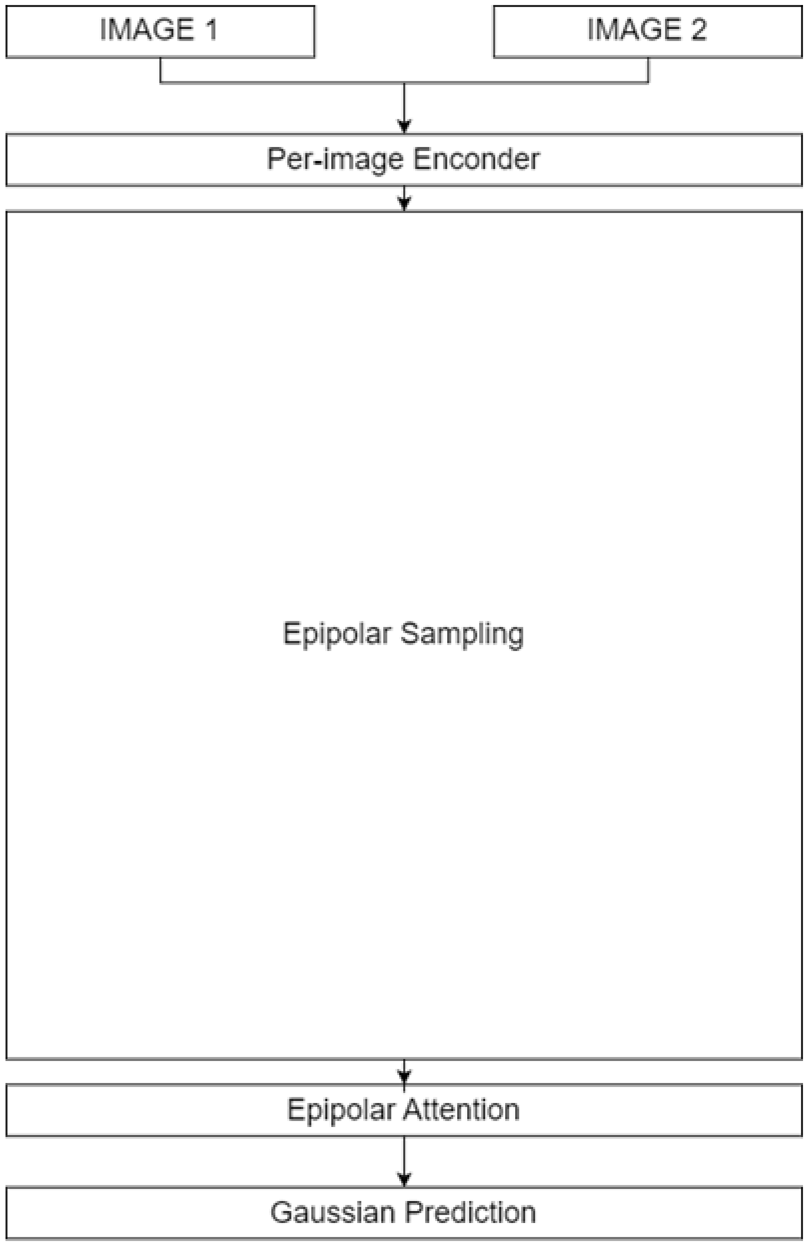
# Epipolar Samples from rays

```python
class EpipolarSampler(nn.Module):
    num_samples: int
    index_v: Index
    transpose_v: Index
    transpose_ov: Index

    def __init__(...

    def forward(
        self,
        images: Float[Tensor, "batch view channel height width"],
        extrinsics: Float[Tensor, "batch view 4 4"],
        intrinsics: Float[Tensor, "batch view 3 3"],
        near: Float[Tensor, "batch view"],
        far: Float[Tensor, "batch view"],
    ) -> EpipolarSampling:
        device = images.device
        b, v, _, _, _ = images.shape
      David Charatan, 10 months ago • Release code.
        # Generate the rays that are projected onto other views.
        xy_ray, origins, directions = self.generate_image_rays(
            images, extrinsics, intrinsics
        )

        # Select the camera extrinsics and intrinsics to project onto. For each
        context
        # view, this means all other context views in the batch.
        projection = project_rays(
            rearrange(origins, "b v r xyz -> b v () r xyz"),
            rearrange(directions, "b v r xyz -> b v () r xyz"),
            rearrange(self.collect(extrinsics), "b v ov i j -> b v ov () i j"),
            rearrange(self.collect(intrinsics), "b v ov i j -> b v ov () i j"),
            rearrange(near, "b v -> b v () ()"),
            rearrange(far, "b v -> b v () ()"),
        )
```



# Epipolar Samples …

```python
# Generate sample points.
s = self.num_samples
sample_depth = (torch.arange(s, device=device) + 0.5) / s
sample_depth = rearrange(sample_depth, "s -> s ()")
xy_min = projection["xy_min"].nan_to_num(posinf=0, neginf=0)
xy_min = xy_min * projection["overlaps_image"][..., None]
xy_min = rearrange(xy_min, "b v ov r xy -> b v ov r () xy")
xy_max = projection["xy_max"].nan_to_num(posinf=0, neginf=0)
xy_max = xy_max * projection["overlaps_image"][..., None]
xy_max = rearrange(xy_max, "b v ov r xy -> b v ov r () xy")
xy_sample = xy_min + sample_depth * (xy_max - xy_min)

samples = self.transpose(xy_sample)
samples = F.grid_sample(
    rearrange(images, "b v c h w -> (b v) c h w"),
    rearrange(2 * samples - 1, "b v ov r s xy -> (b v) (ov r s) () xy"),
    mode="bilinear",
    padding_mode="zeros",
    align_corners=False,
)
samples = rearrange(
    samples, "(b v) c (ov r s) () -> b v ov r s c", b=b, v=v, ov=v - 1,
    s=s
)
samples = self.transpose(samples)

# Zero out invalid samples.
samples = samples * projection["overlaps_image"][..., None, None]

half_span = 0.5 / s
return EpipolarSampling(
```



# The transformer

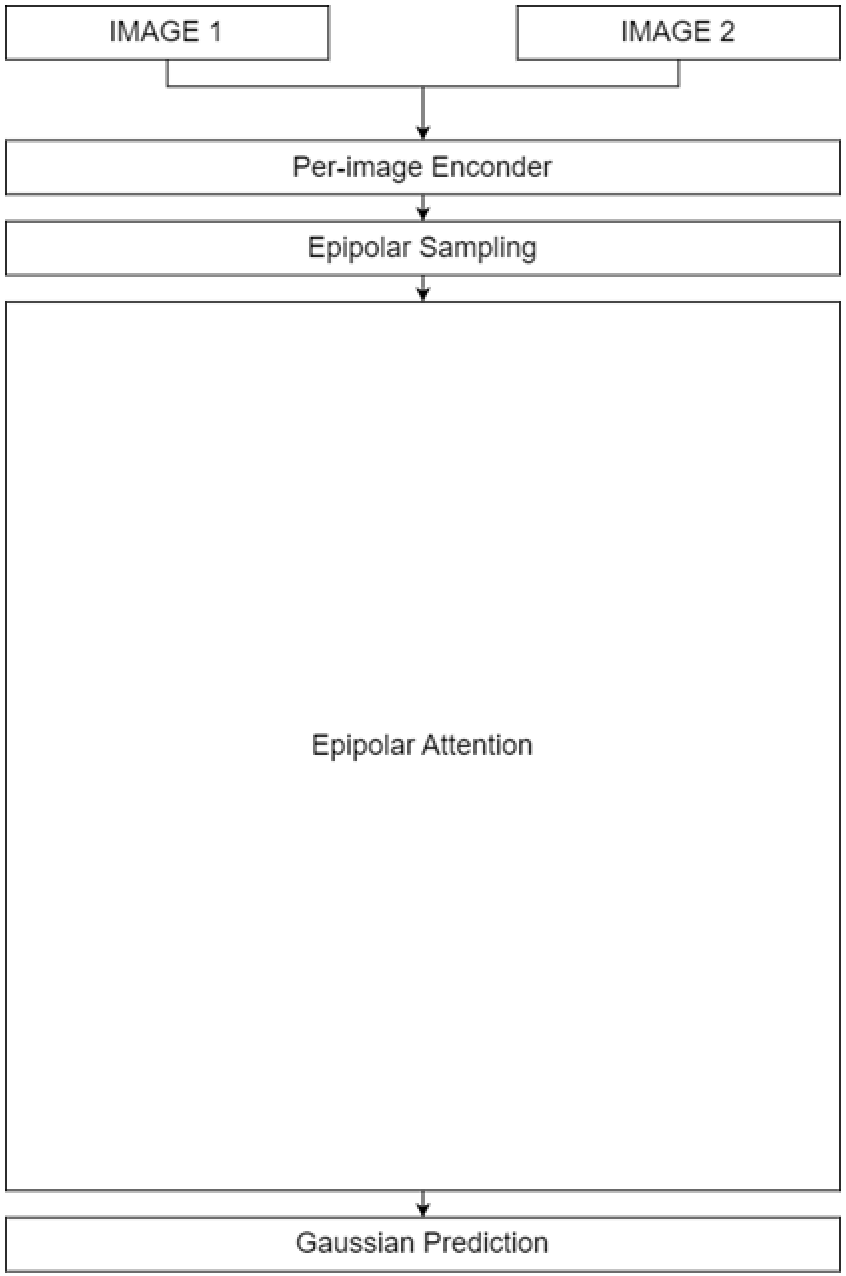
- forward method inputs:
    - self, features, extrinsics, intrinsics, near, far

```python
class EpipolarTransformer(nn.Module):
        sampling = self.epipolar_sampler.forward(⋯
        q = rearrange(features, "b v c h w -> (b v h w) () c")
        features = self.transformer.forward(
            q,
            rearrange(kv, "b v ov r s c -> (b v r) (s ov) c"),
            b=b,
            v=v,
            h=h // self.cfg.downscale,
            w=w // self.cfg.downscale,
        )
        features = rearrange(
            features,
            "(b v h w) () c -> b v c h w",
            b=b,
            v=v,
            h=h // self.cfg.downscale,
            w=w // self.cfg.downscale,
        )

        # If needed, apply upscaling.
        if self.upscaler is not None:
            features = rearrange(features, "b v c h w -> (b v) c h w")
            features = self.upscaler(features)
            features = self.upscale_refinement(features) + features
            features = rearrange(features, "(b v) c h w -> b v c h w", b=b, v=v)
```

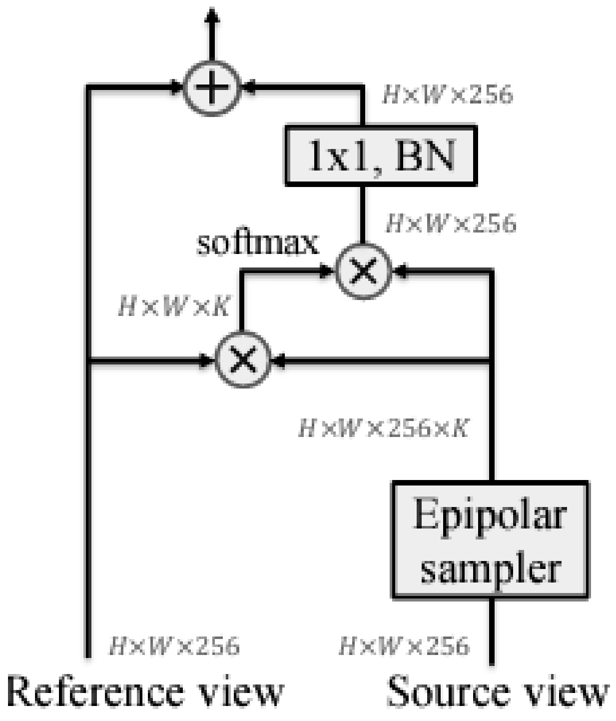
- for each input image F and the other is called ~F

$$\mathbf{s} = \tilde{\mathbf{F}}[\tilde{\mathbf{u}}_l] \oplus \gamma(\tilde{d}_{\tilde{\mathbf{u}}_l})$$

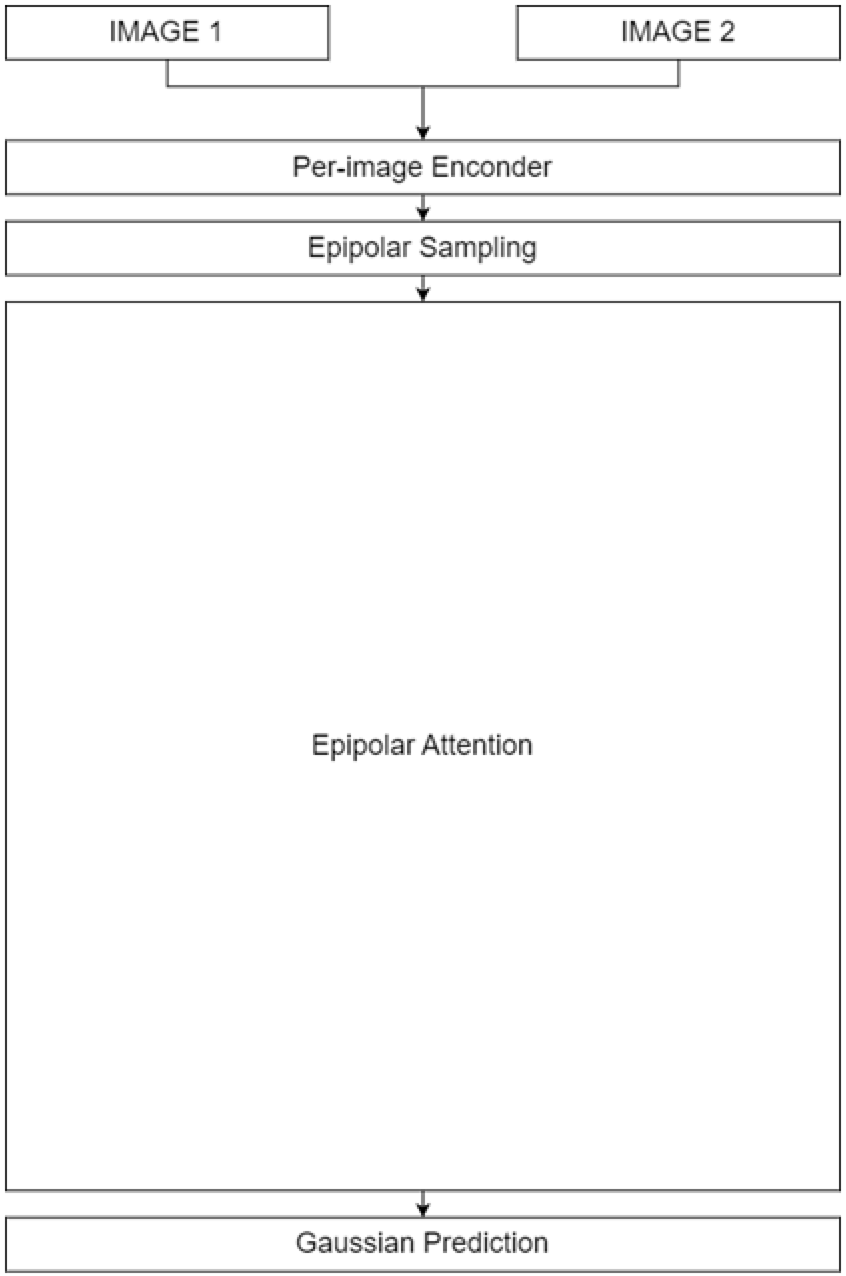
$$\mathbf{q} = \mathbf{Q} \cdot \mathbf{F}[\mathbf{u}], \quad \mathbf{k}_l = \mathbf{K} \cdot \mathbf{s}, \quad \mathbf{v}_l = \mathbf{V} \cdot \mathbf{s},$$



# The transformer

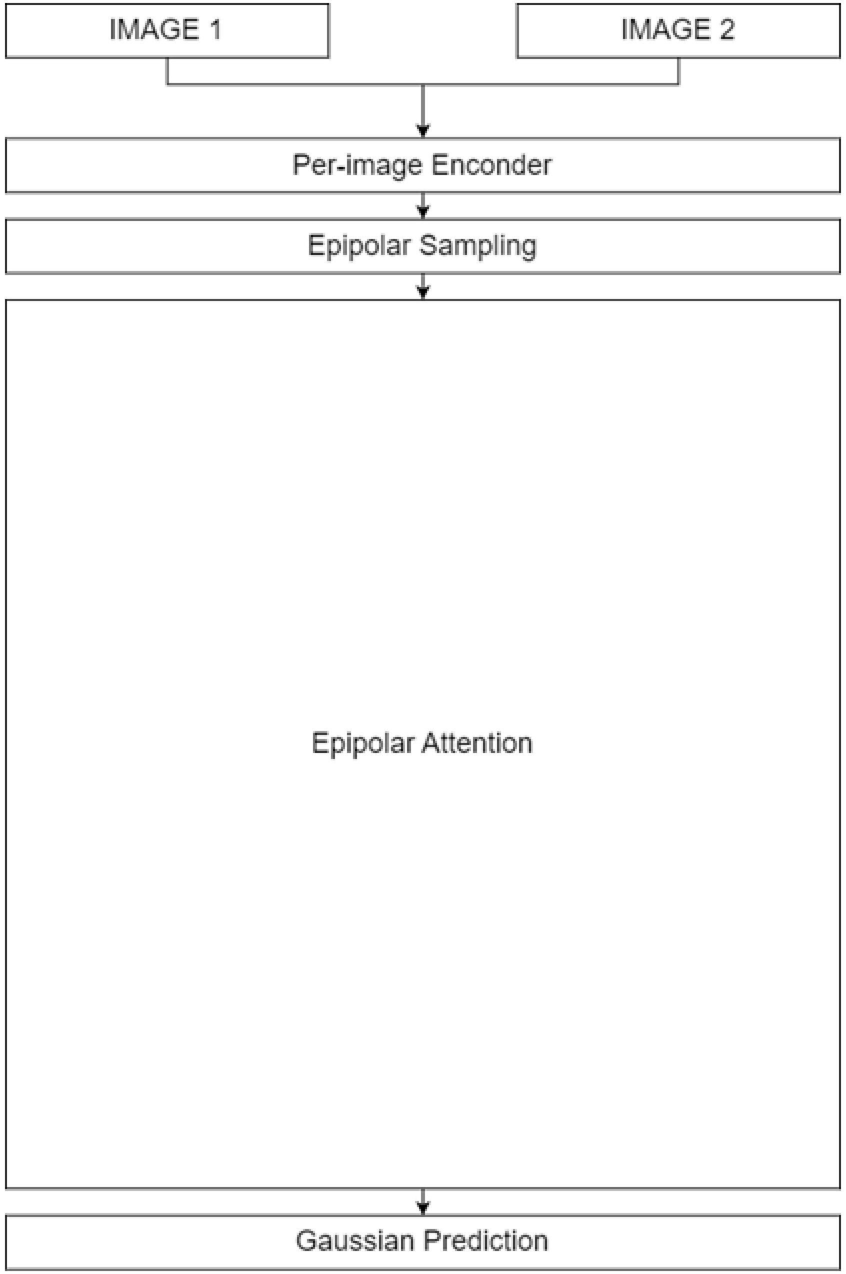



- source paper: Epipolar Transformer

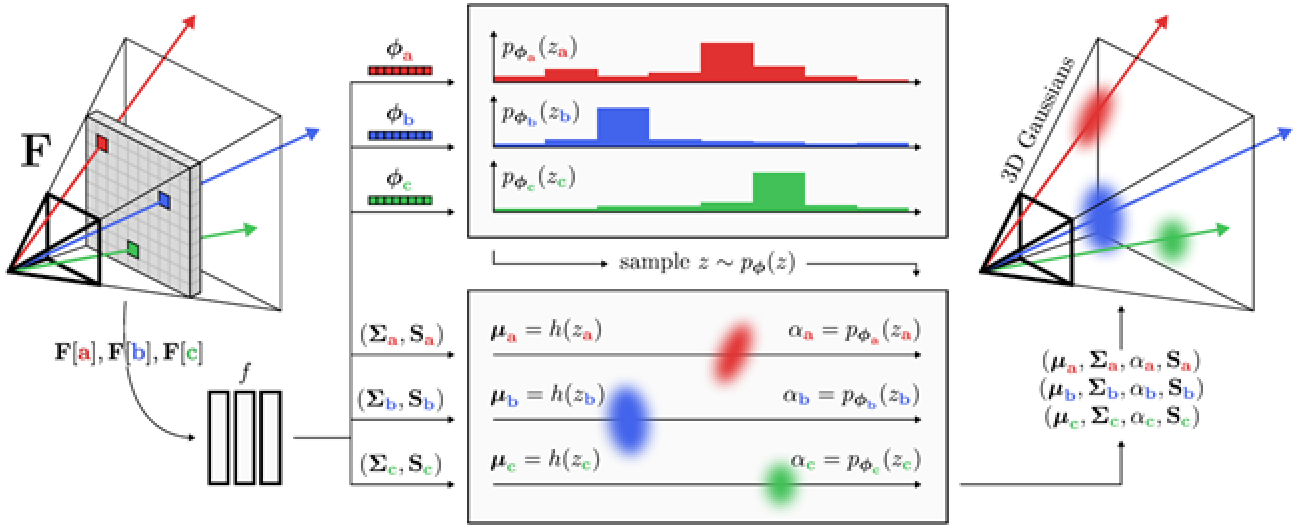


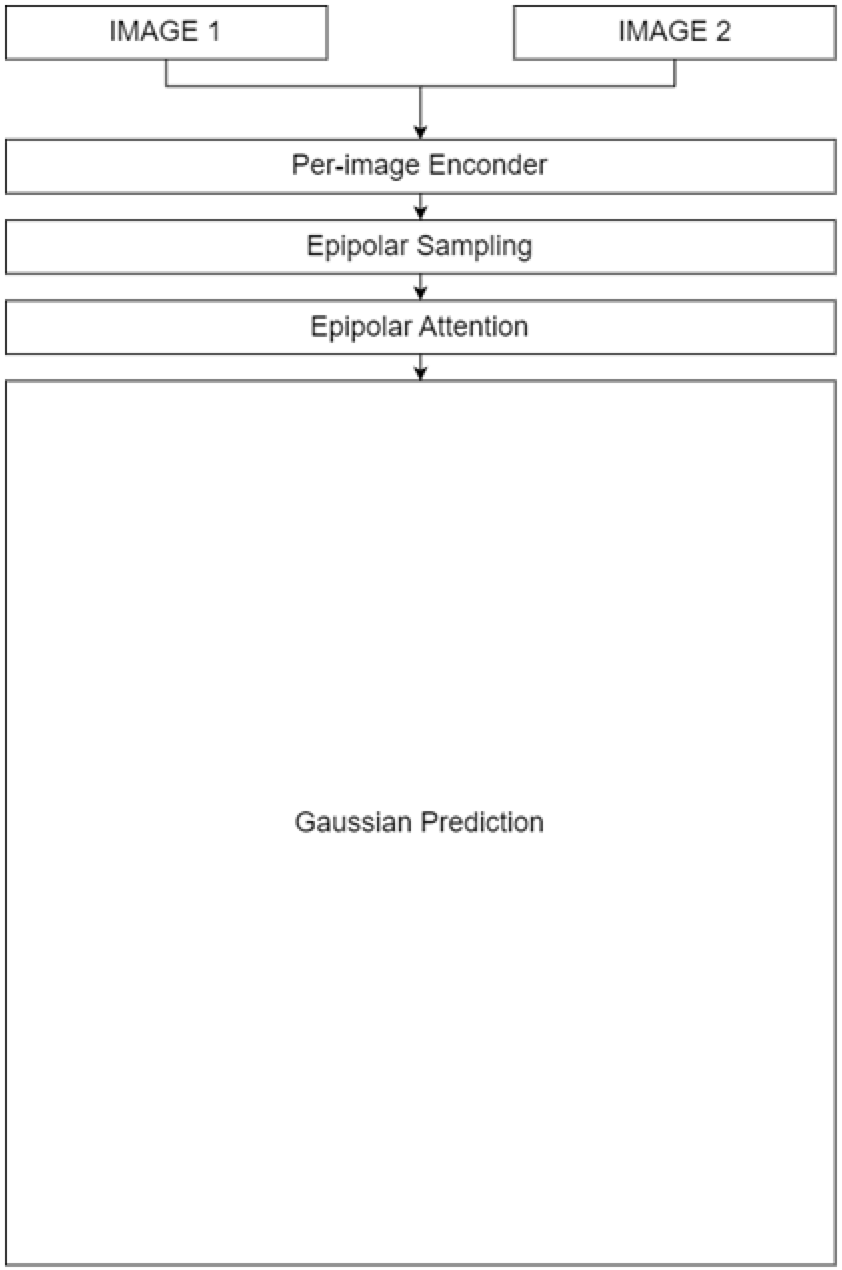
# Attention

```python
class Attention(nn.Module):
    def __init__(···

    def forward(self, x, z=None):
        if z is None:
            qkv = self.to_qkv(x).chunk(3, dim=-1)
        else:
            q = self.to_q(x)
            k, v = self.to_kv(z).chunk(2, dim=-1)
            qkv = (q, k, v)

        q, k, v = map(lambda t: rearrange(t, "b n (h d) -> b h n d", h=self.
        heads), qkv)

        dots = torch.matmul(q, k.transpose(-1, -2)) * self.scale

        attn = self.attend(dots)

        out = torch.matmul(attn, v)
        out = rearrange(out, "b h n d -> b n (h d)")
        return self.to_out(out)
```



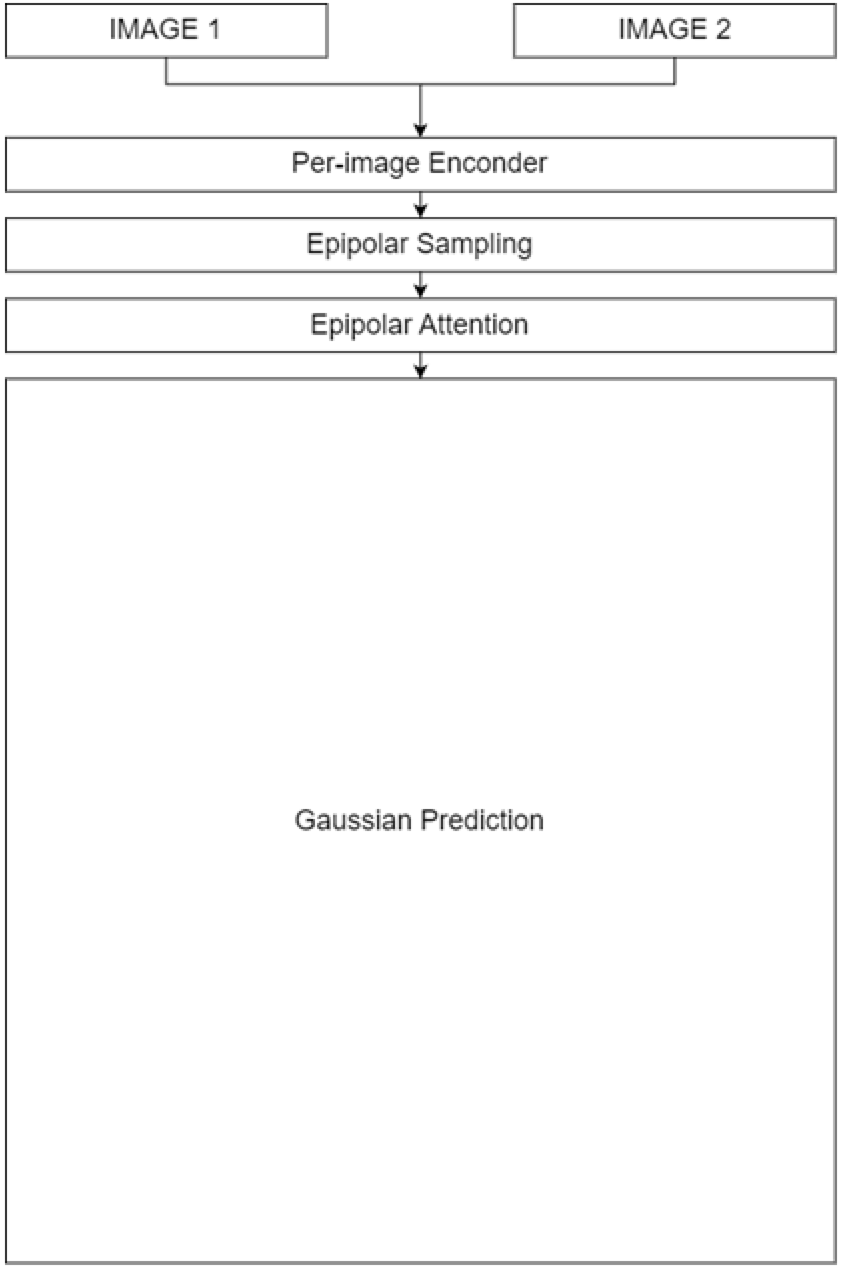
# Generating Gaussians



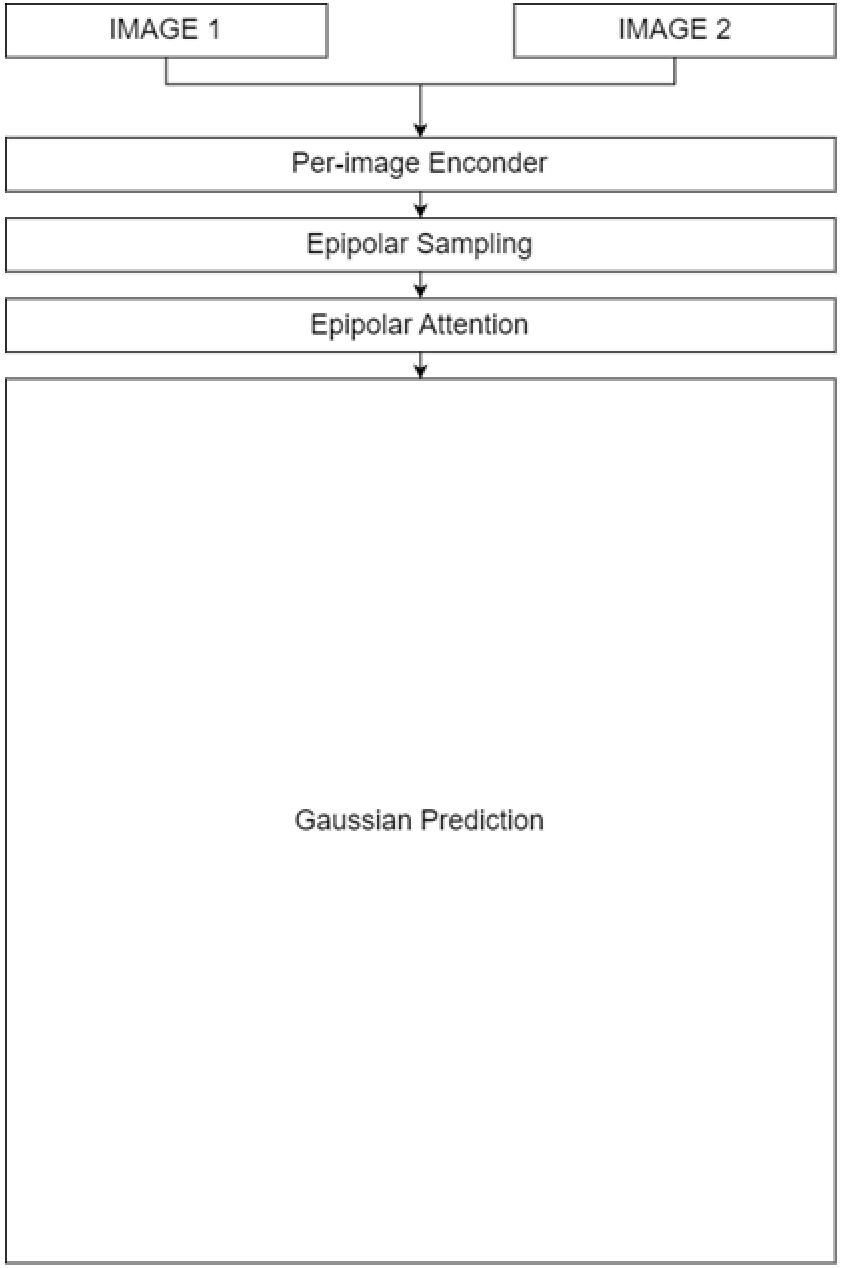
# Generating Gaussians

```python
class EncoderEpipolar(Encoder[EncoderEpipolarCfg]):
    backbone: Backbone
    backbone_projection: nn.Sequential
    epipolar_transformer: EpipolarTransformer | None
    depth_predictor: DepthPredictorMonocular
    to_gaussians: nn.Sequential
    gaussian_adapter: GaussianAdapter
    high_resolution_skip: nn.Sequential

    def __init__(self, cfg: EncoderEpipolarCfg) -> None:...

    def map_pdf_to_opacity(...

    def forward(
        self,
        context: dict,
        global_step: int,
        deterministic: bool = False,
        visualization_dump: Optional[dict] = None,
    ) -> Gaussians:
        device = context["image"].device
        b, v, _, h, w = context["image"].shape

        # Encode the context images.
        features = self.backbone(context)
        features = rearrange(features, "b v c h w -> b v h w c")
        features = self.backbone_projection(features)
        features = rearrange(features, "b v h w c -> b v c h w")

        # Run the epipolar transformer.
        if self.cfg.use_epipolar_transformer:        David Charatan, 10 months ago • R

        # Add the high-resolution skip connection.
        skip = rearrange(context["image"], "b v c h w -> (b v) c h w")
        skip = self.high_resolution_skip(skip)
        features = features + rearrange(skip, "(b v) c h w -> b v c h w", b=b,
        v=v)
```



# Generating Gaussians

```python
# Sample depths from the resulting features.
features = rearrange(features, "b v c h w -> b v (h w) c")
depths, densities = self.depth_predictor.forward(
    features,
    context["near"],
    context["far"],
    deterministic,
    1 if deterministic else self.cfg.gaussians_per_pixel,
)

# Convert the features and depths into Gaussians.
xy_ray, _ = sample_image_grid((h, w), device)
xy_ray = rearrange(xy_ray, "h w xy -> (h w) () xy")
gaussians = rearrange(
    self.to_gaussians(features),
    "... (srf c) -> ... srf c",
    srf=self.cfg.num_surfaces,
)
offset_xy = gaussians[..., :2].sigmoid()
pixel_size = 1 / torch.tensor((w, h), dtype=torch.float32,
device=device)
xy_ray = xy_ray + (offset_xy - 0.5) * pixel_size
gpp = self.cfg.gaussians_per_pixel
gaussians = self.gaussian_adapter.forward(
    rearrange(context["extrinsics"], "b v i j -> b v () () () i j"),
    rearrange(context["intrinsics"], "b v i j -> b v () () () i j"),
    rearrange(xy_ray, "b v r srf xy -> b v r srf () xy"),
    depths,
    self.map_pdf_to_opacity(densities, global_step) / gpp,
    rearrange(gaussians[..., 2:], "b v r srf c -> b v r srf () c"),
    (h, w),
)
```



# Some sayings

- extrinsics, intrinsics, far, near, and other factors are user parameters
- github only explains training, evaluation and some tests
  - it does not state how to run on 2 images
  - it also does not show how to export .ply files

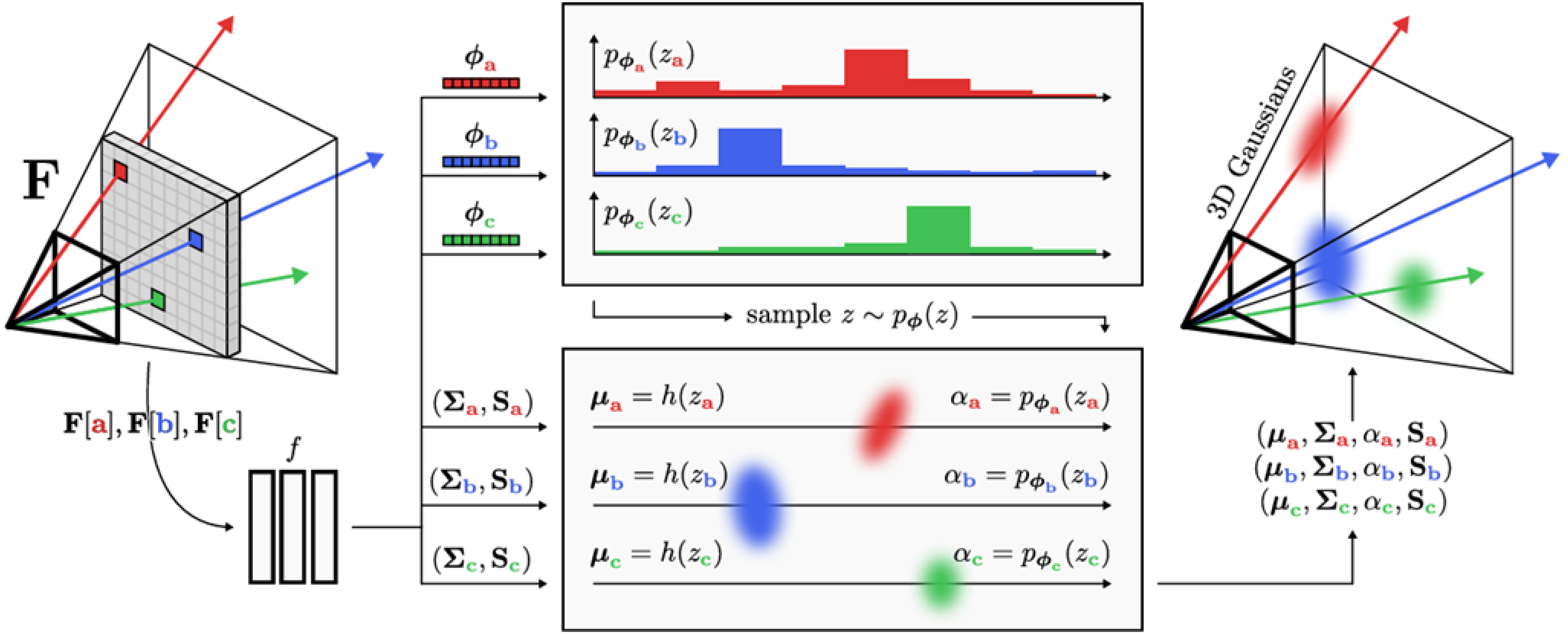
# Running the code once

# MVSplat vs PixelSplat

# PhD Student

# Fernando Pereira de Sá

# 1 - Research Proposition




**Algorithm 1** Probabilistic Prediction of a Pixel-Aligned Gaussian.

**Require:** Depth buckets $\mathbf{b} \in \mathbb{R}^Z$, feature $\mathbf{F}[\mathbf{u}]$ at pixel coordinate $\mathbf{u}$, camera origin of reference view $\mathbf{o}$, ray direction $\mathbf{d_u}$.

$(\boldsymbol{\phi}, \boldsymbol{\delta}, \boldsymbol{\Sigma}, \mathbf{S}) = f(\mathbf{F}[\mathbf{u}])$ ▷ predict depth probabilities $\boldsymbol{\phi}$ and offsets $\boldsymbol{\delta}$, covariance $\boldsymbol{\Sigma}$, spherical harmonics coefficients $\mathbf{S}$

$z \sim p_{\boldsymbol{\phi}}(z)$ ▷ Sample depth bucket index $z$ from discrete probability distribution parameterized by $\boldsymbol{\phi}$

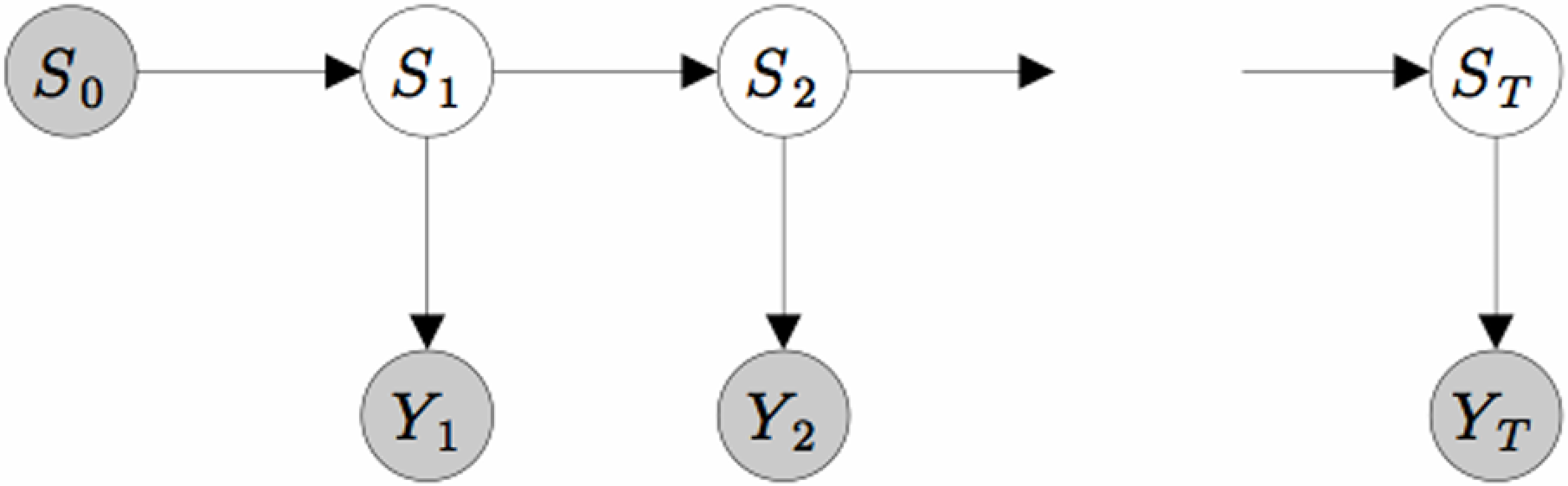
$\boldsymbol{\mu} = \mathbf{o} + (\mathbf{b}_z + \boldsymbol{\delta}_z)\mathbf{d_u}$ ▷ Compute Gaussian mean $\boldsymbol{\mu}$ by unprojecting with depth $\mathbf{b}_z$ adjusted by bucket offset $\boldsymbol{\delta}_z$

$\boldsymbol{\alpha} = \boldsymbol{\phi}_z$ ▷ Set Gaussian opacity $\boldsymbol{\alpha}$ according to probability of sampled depth (Sec. 4.2).

**return** $(\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\alpha}, \mathbf{S})$

# 1 - Research Proposition

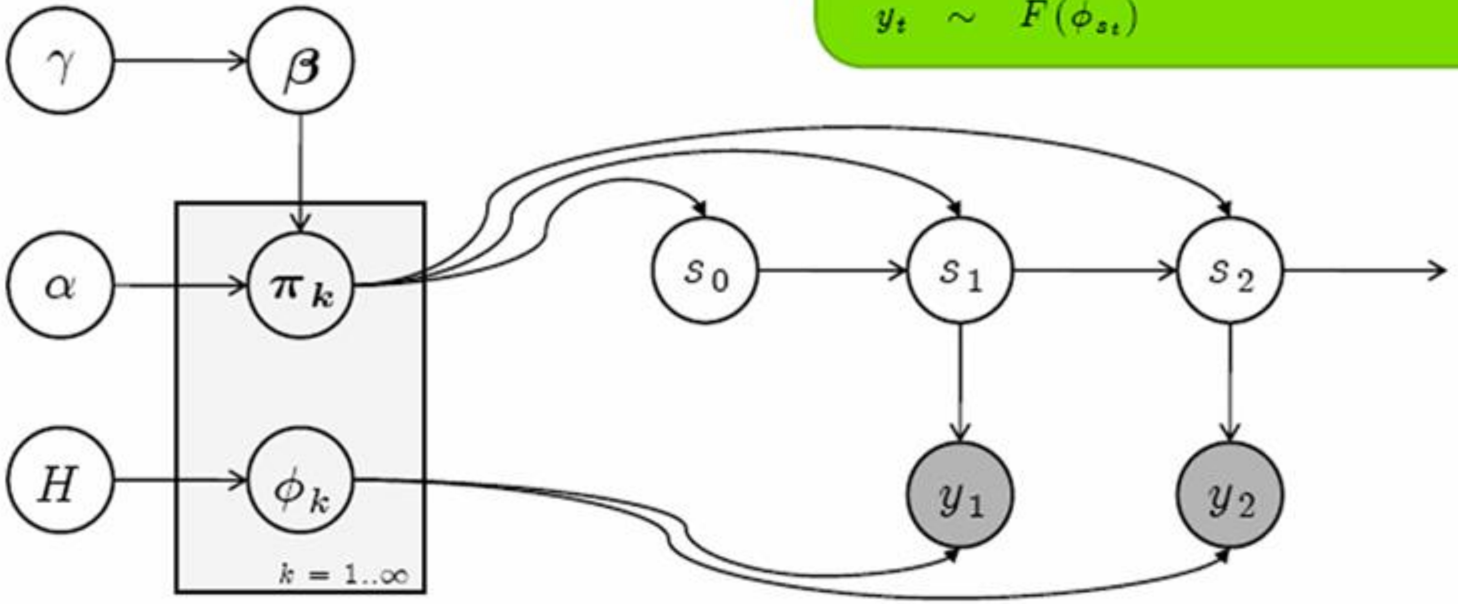
Hidden Markov Models



# 1 - Research Proposition

- Generative Model for iHMM

$$
\begin{aligned}
\boldsymbol{\beta} &\sim \mathrm{Stick}(\gamma), \\
\boldsymbol{\phi}_k &\sim H, \\
\boldsymbol{\pi}_k &\sim \mathrm{Dirichlet}(\alpha\boldsymbol{\beta}), \\
s_t &\sim \mathrm{Multinomial}(\boldsymbol{\pi}_{s_{t-1}}), \qquad (s_0 = 1) \\
y_t &\sim F(\boldsymbol{\phi}_{s_t})
\end{aligned}
$$




Teh, Jordan, Beal and Blei (2005) derived iHMMs in terms of Hierarchical Dirichlet Processes.

# 1 - Research Proposition

## Inference and Learning: Gibbs Sampling

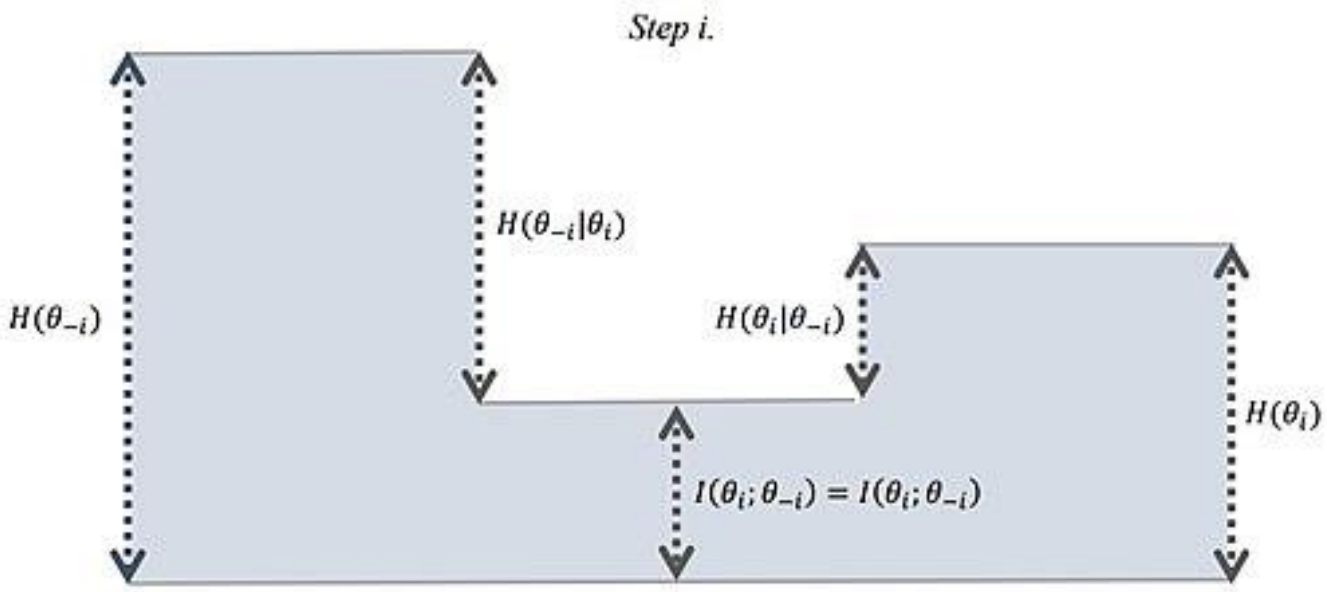
initialize $Y^0, X^0$

for j = 1, 2, 3,… do

  sample $X^j \sim p(X|Y^{j-1})$

  sample $Y^j \sim p(Y|X^j)$

end for



# 1 - Research Proposition

## Inference and Learning: Gibbs Sampling

### Gibbs sampling the posterior of neural networks

Giovanni Piccioli[2,1], Emanuele Troiani[1] and Lenka Zdeborová[1]

Published 11 March 2024 • © 2024 The Author(s). Published by IOP Publishing Ltd

Journal of Physics A: Mathematical and Theoretical, Volume 57, Number 12

**Bayesian Statistics for Complex Systems**

**Citation** Giovanni Piccioli *et al* 2024 *J. Phys. A: Math. Theor.* **57** 125002

**DOI** 10.1088/1751-8121/ad2c26

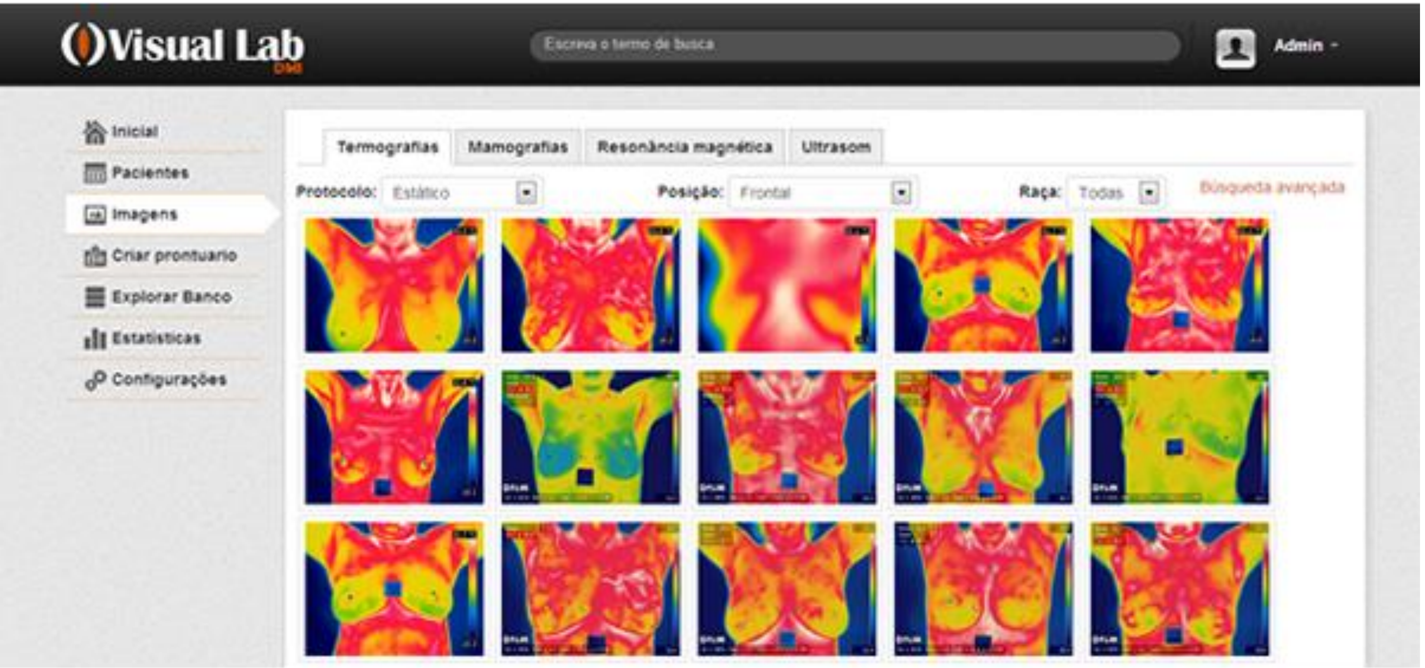
**Abstract**

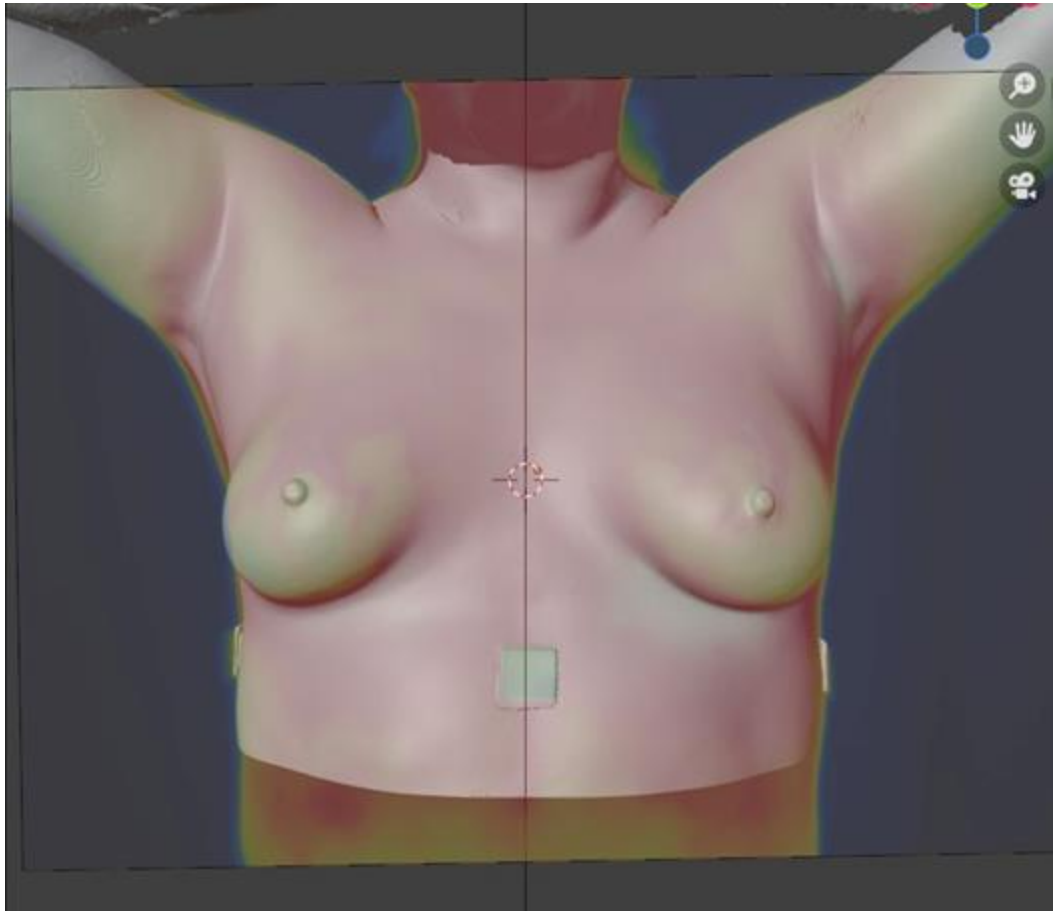
In this paper, we study sampling from a posterior derived from a neural network. We propose a new probabilistic model consisting of adding noise at every pre- and post-activation in the network, arguing that the resulting posterior can be sampled using an efficient Gibbs sampler. For small models, the Gibbs sampler attains similar performances as the state-of-the-art Markov chain Monte Carlo methods, such as the Hamiltonian Monte Carlo or the Metropolis adjusted Langevin algorithm, both on real and synthetic data. By framing our analysis in the teacher-student setting, we introduce a thermalization criterion that allows us to detect when an algorithm, when run on data with synthetic labels, fails to sample from the posterior. The criterion is based on the fact that in the teacher-student setting we can initialize an algorithm directly at equilibrium.

Keywords: MCMC, Bayesian learning, neural networks,
sampling algorithms, MCMC thermalization, statistical physics

# 2 - Use Case


Visual Lab
33.5 °C
20.0


Visual Lab
Escreva o termo de busca
Admin
Inicial
Pacientes
Imagens
Criar prontuario
Explorar Banco
Estatísticas
Configurações
Termografias
Mamografias
Resonância magnética
Ultrasom
Protocolo:
Estático
Posição:
Frontal
Raça:
Todas
Busqueda avançada



Thank you!