



# N-Dimensional Gaussians for Fitting of High Dimensional Functions

Diolatzis et al., SIGGRAPH 2024

GdL 11 décembre 2024  
Gaspard Thévenon

# High-dimensional field fitting

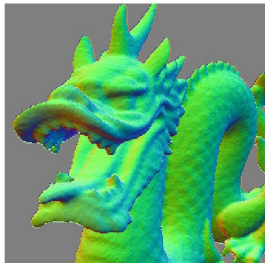
**Goal:** learn a continuous representation for a field from training images, that allows to render it from arbitrary point of views and/or store it efficiently.



*Image regression*  
(x,y) -> RGB



*MRI reconstruction*  
(x,y,z) -> density



*Shape regression*  
(x,y,z) -> occupancy

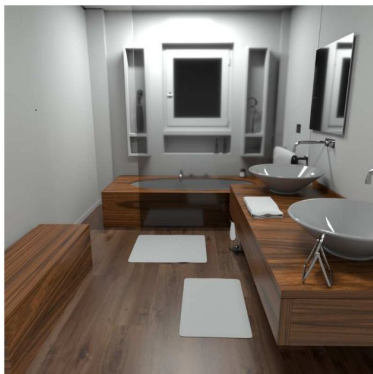


*Radiance field*  
(x,y,z) -> RGB, density

# High-dimensional field fitting

**Goal:** learn a continuous representation for a field from training images, that allows to render it from arbitrary point of views and/or store it efficiently.

**Particularity here:** high-dimensional input, and strong inter-dependencies (ie. anisotropy)



*Global Illumination with Variability*  
( $x, y, z, v, \text{albedo}, r, \dots$ )  $\rightarrow$  RGB



*Dynamic Scenes*  
( $x, y, z, t$ )  $\rightarrow$  RGB

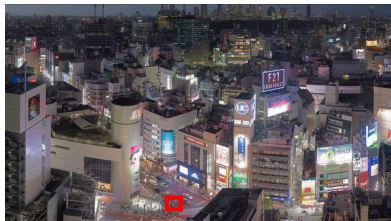


*Radiance field with strong specular effects*  
( $x, y, z, v$ )  $\rightarrow$  RGB

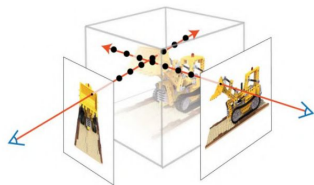
# High-dimensional field fitting

## Implicit Methods

Gigapixels



Radiance  
Fields



*InstantNGP*

*NeRF*

Shape  
regression



*NeuS*

## Hybrid/Explicit Methods

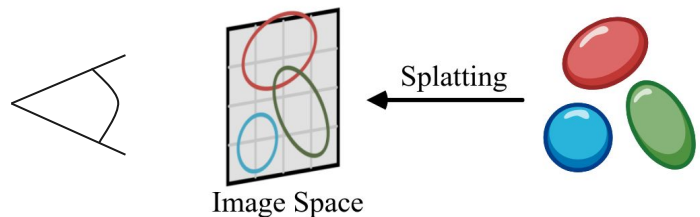


*3D Gaussian Splatting*



*Neural Geometric Levels of  
Details*

# 3D Gaussian Splatting



Specialized Gaussian Mixture Model (**GMM**):

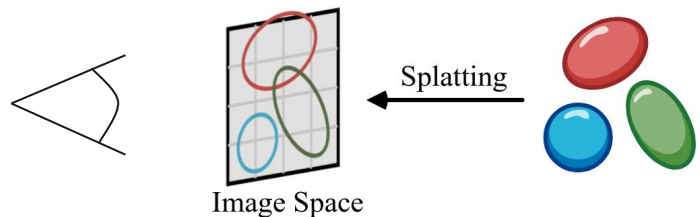
The radiance field is represented as a sum of Gaussians:

$$G(x) = \exp\left(-\frac{1}{2}(x - p)^T V^{-1}(x - p)\right)$$

where  $\mathbf{p}$  is the mean of the Gaussian,  $\mathbf{V}$  its (3x3) covariance matrix.

Each Gaussian carries a view-dependent color (represented by spherical harmonic coefficients) and a transparency coefficient.

# 3D Gaussian Splatting



Two main contributions from 3DGS:

- Fast rasterization, because the splatting process skips empty spaces, and with an efficient GPU implementation
- Specialized heuristics during the optimization, to split/clone gaussians automatically

## Issue:

Here, the field has 3 input dimensions (xyz), and the method doesn't generalize naturally to higher dimensions.

# From 3D to ND



## *Covariance representation*

$V$  has to be positive-definite, which is a difficult constraint to enforce during optimization.

**3DGS:**  $V = RSS^T R^T$ , where  $R$  is a 3-dim rotation matrix and  $S$  a scaling matrix.

-> for optimization,  $R$  is represented as a normalized quaternion, and  $S$  as a log-space vector

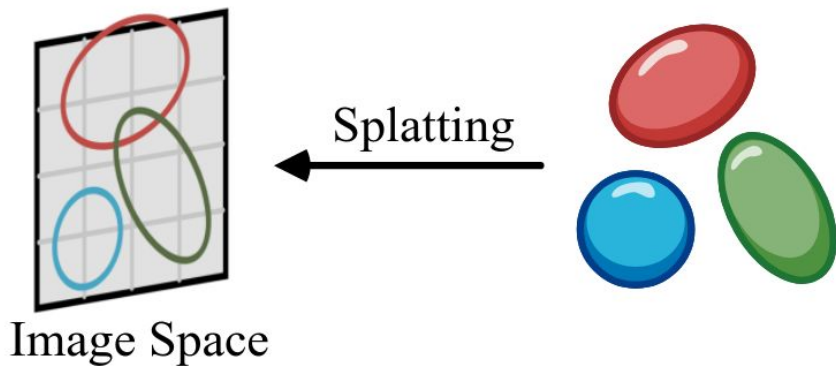
**NDGS:** Rotations are hard to describe in arbitrary dimensions... So they stick to a Cholesky decomposition:

$V = LL^T$ , with  $L$  lower triangular, with diagonal elements in log-space and others in logit-space.

# From 3D to ND

## *Primitive culling*

When we render a pixel, we only blend Gaussians that are “close” to the rendered ray.



3DGS offers a natural culling method:

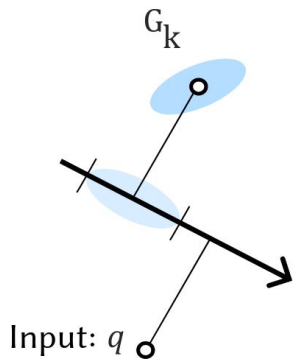
Each Gaussian projected on the render plane only influences the pixels in ellipse corresponding to the standard deviation of the Gaussian.

-> More complicated for higher dimensions.

# From 3D to ND

## Primitive culling

When we render a pixel, we only blend Gaussians that are “close” to the rendered ray.



$$q_{\mathbf{r}} = \mathbf{q}^T \mathbf{r}, \quad m_{\mathbf{r}} = \mathbf{m}^T \mathbf{r}$$

$$\sigma_{\mathbf{r}}^2 = \mathbf{r}^T \mathbf{V} \mathbf{r}$$

*Locality-Sensitive Hashing:*

Given a query point  $\mathbf{q}$ , and a Gaussian  $(\mathbf{m}, \mathbf{V})$ , we project  $\mathbf{q}$ ,  $\mathbf{m}$ , and  $\mathbf{V}$  on  $k$  different random vectors.

The Gaussian is kept if this equality holds for every projection vector:

$$|q_{\mathbf{r}} - m_{\mathbf{r}}| < 3\sigma_{\mathbf{r}}$$

*Conservative culling:* it doesn't discard important Gaussians.

# From 3D to ND

## Densification strategies

3DGS presents heuristics to add and remove Gaussians during the optimization.

### Adding Gaussians

If there are not enough Gaussians to render details, new ones are added. Every  $T$  iterations, Gaussians with positional gradients above a threshold are split or cloned.

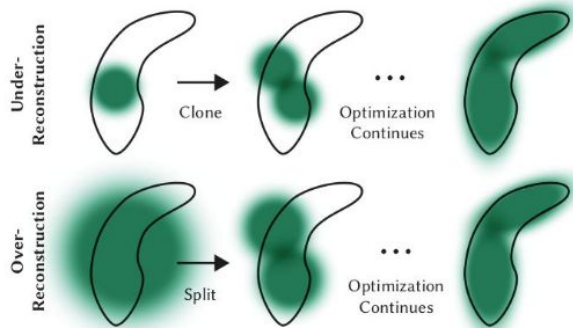


figure from Kerbl et al [1]

### Removing Gaussians

> Every  $T$  iterations, Gaussians that are too large are removed ; Gaussians with opacity less than a threshold are removed

> Every  $K$  iterations, opacities are reset to a low value, in order to remove flying outliers

Typical values in original implementation:  
 $T = 100$ ,  $K = 3000$ ,  
after warm-up phase

# From 3D to ND

## Densification strategies

NDGS aims at a “purely optimizer-controlled” refinement strategy.

Every 300 iterations, Gaussians with low opacity are given children Gaussians.

Parameters of the children Gaussians are defined in the frame of their parents:

$$\mathbf{V}_c = \mathbf{L}\mathbf{U}(\mathbf{L}\mathbf{U})^T, \quad \mathbf{V}_p = \mathbf{L}\mathbf{L}^T, \\ \mathbf{m}_c = \mathbf{L}\mathbf{m}_u + \mathbf{m}_p.$$

