Implicit neural representations

Julie Digne

Master ID3D
LIRIS - CNRS
Équipe Origami

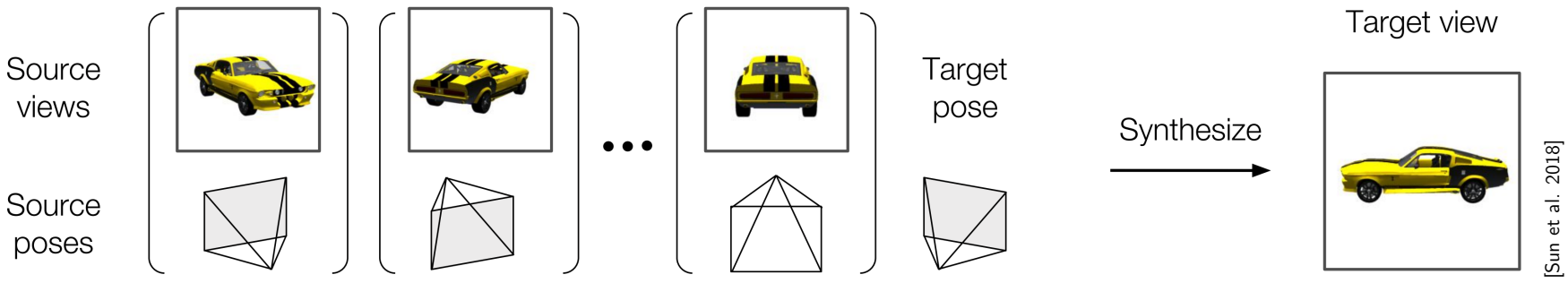
07/10/2025

Neural networks

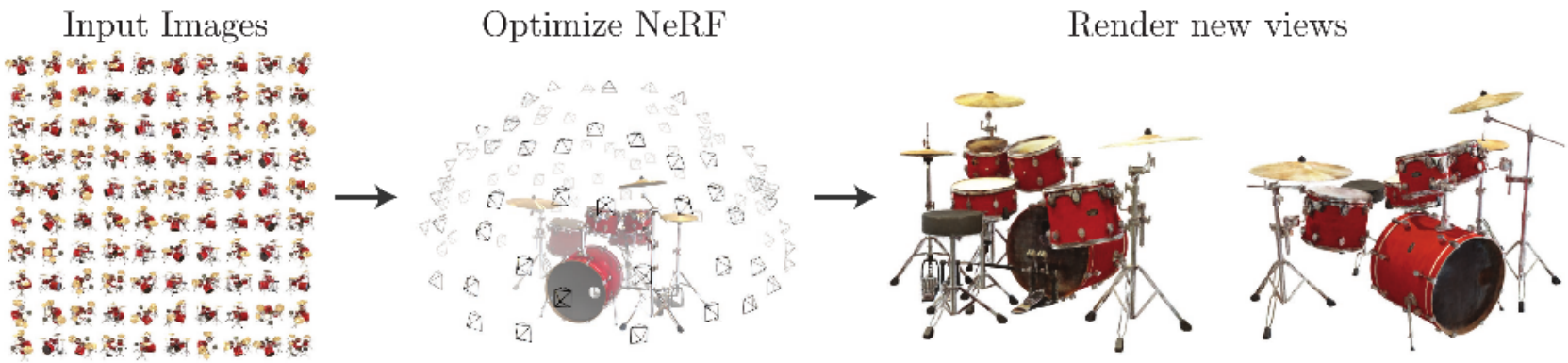
- A way to write a function with (many) parameters to be optimized for a task.
- Neural fields:
  - no need for a dataset (but can be added)
  - no need for high compute power

Outline

Novel View Synthesis



Neural Radiance Field (Nerf [Mildenhall et al. 2020])



- Goal: Generate a new view from a set of views
- Cameras are calibrated (ie we know their positions, orientations and intrinsic parameters)

Principle

Neural network takes as input a 3D coordinate and viewing direction and outputs the volume density and view-dependent emitted radiance at this location and direction.

$$F_\Theta(x, y, z, \theta, \phi) = (R, G, B, \sigma)$$

- Architecture MLP with ReLU activations.

Rendering from the volume

Color of a ray

Ray $\mathrm{r}(t) = \mathrm{o} + t\mathrm{d}$

$$C(\mathrm{r}) = \int_{t_n}^{t_f} T(t)\sigma(\mathrm{r}(t))C(\mathrm{r}(t), \mathrm{d})dt$$

with:

$$T(t) = \exp - \int_{t_n}^{t} \sigma(\mathrm{r}(s))ds$$

- t_n, t_f: near and far bounds

Rendering from the volume

Color of a ray

Ray $\mathrm{r}(t) = \mathrm{o} + t\mathrm{d}$

$$C(\mathrm{r}) = \int_{t_n}^{t_f} T(t)\sigma(\mathrm{r}(t))C(\mathrm{r}(t), \mathrm{d})dt$$

with:

$$T(t) = \exp - \int_{t_n}^{t} \sigma(\mathrm{r}(s))ds$$

- t_n, t_f: near and far bounds
- T: attenuation of the ray so far (Beer's law)

Integral approximation

- Stratified sampling along the ray of positions t_i

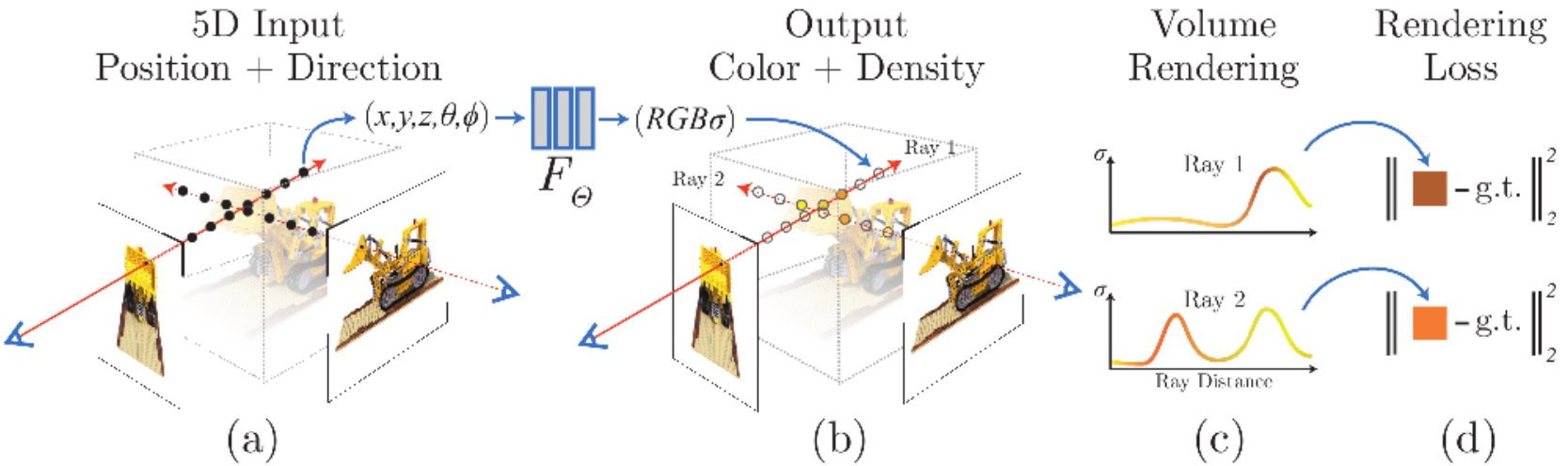
Discrete Version

$$C(\mathrm{r}) = \sum_i T_i (1 - \exp(-\sigma(t_i)\|t_{i+1} - t_i\|)) C(\mathrm{r}_i)$$

with

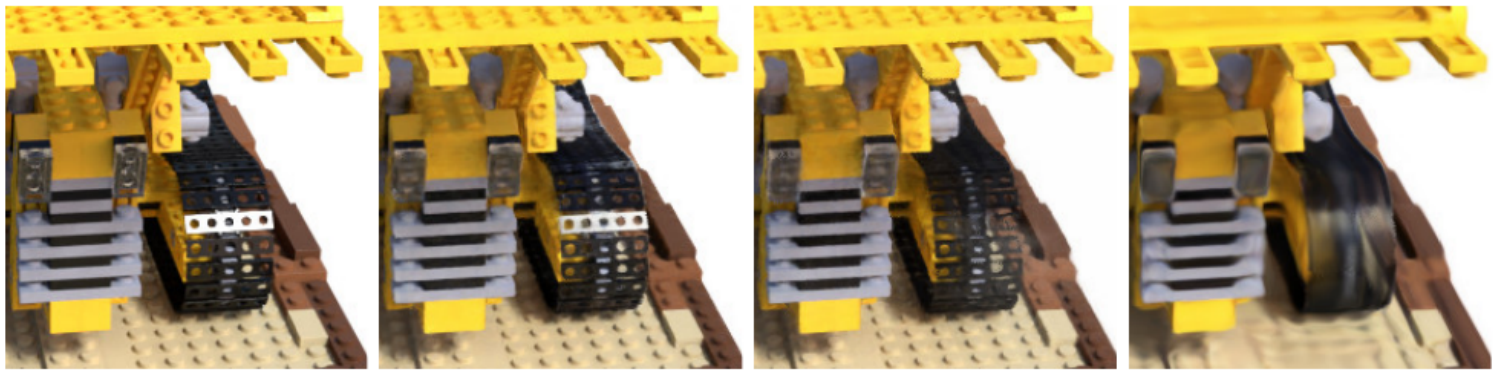
$$T_i = \sum_i \exp(-\sigma(t_i)\|t_{i+1} - t_i\|)$$

Training



[Mildenhall et al. 2020]

Positional Encoding

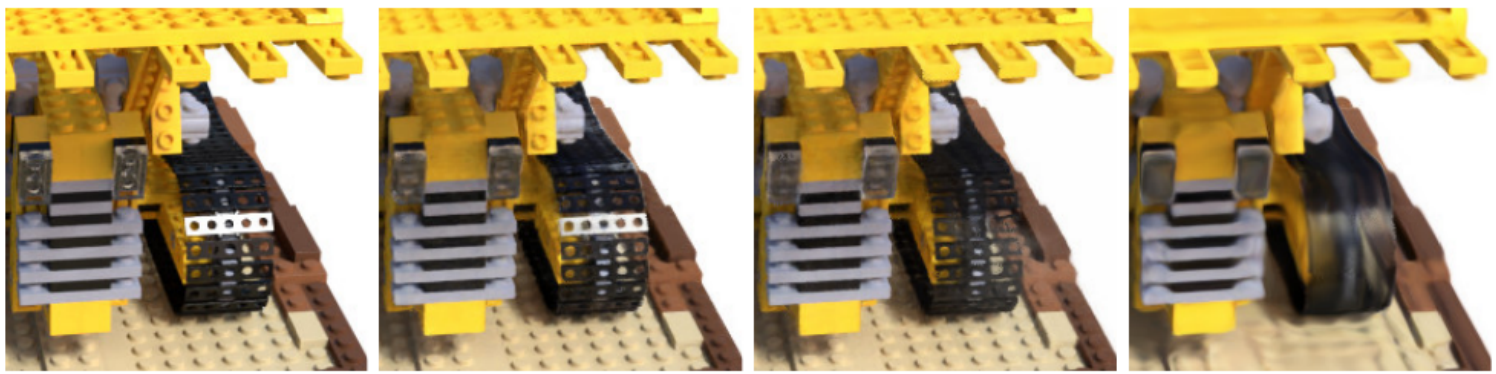


- Add a non-learnable layer to embed the position in a higher dimensional space:

  $$(\cos x, \cos 2x, \cdots, \cos Nx, \cos y, \cos 2y, \cdots, \cos Ny, \cos z, \cos 2z, \cdots, \cos Nz)$$

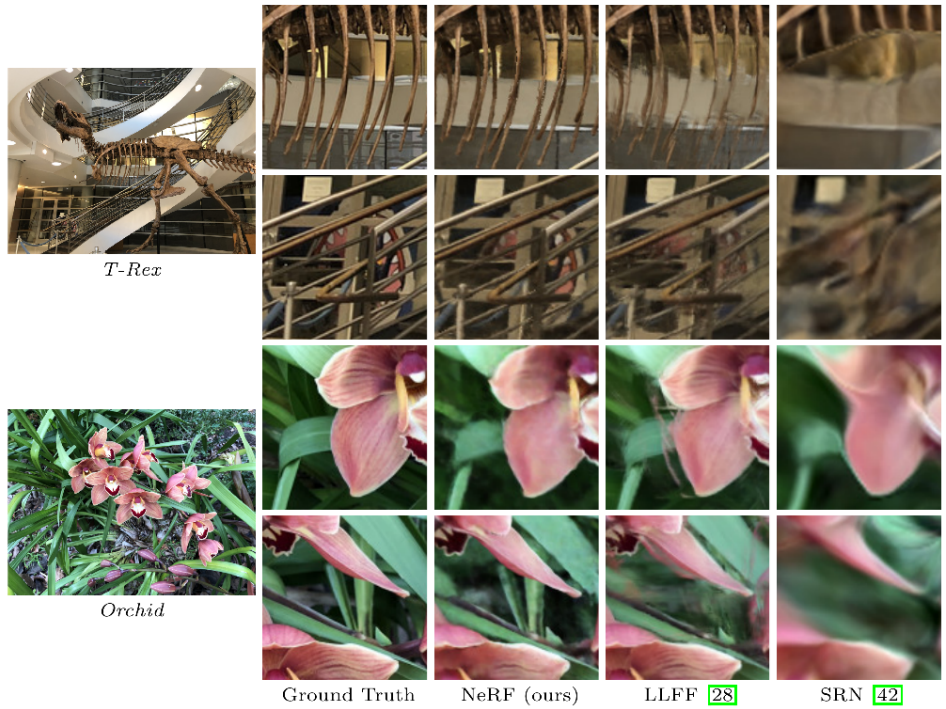
- Intuition: Frequency decomposition, allows to get high frequency information

View-dependency



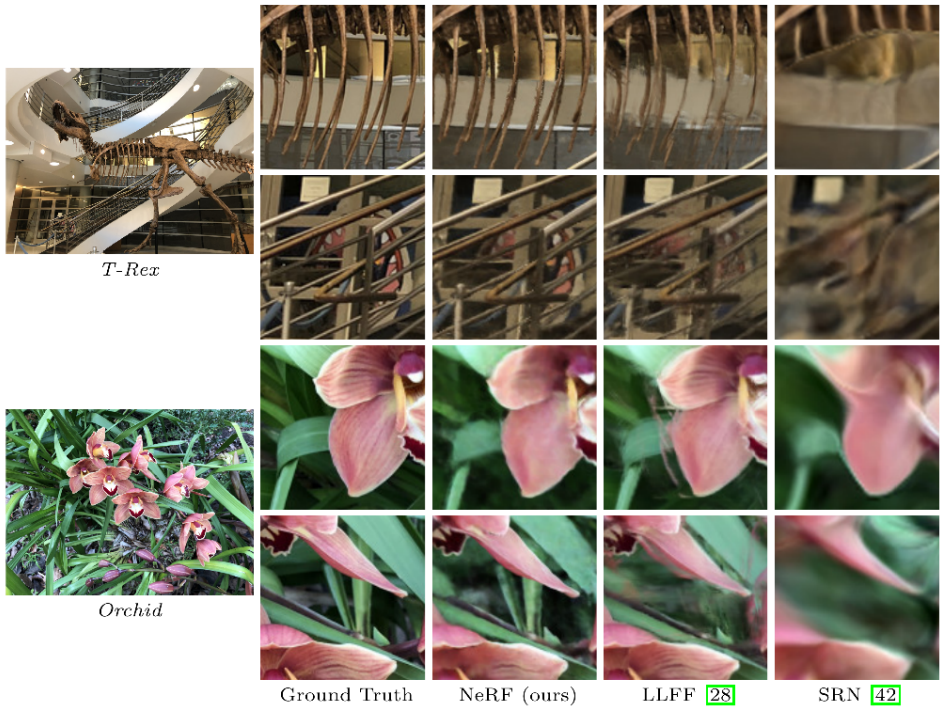
- View-dependent radiance is what allows to capture mirror reflections

Results



Video: `https://www.matthewtancik.com/nerf`

Results

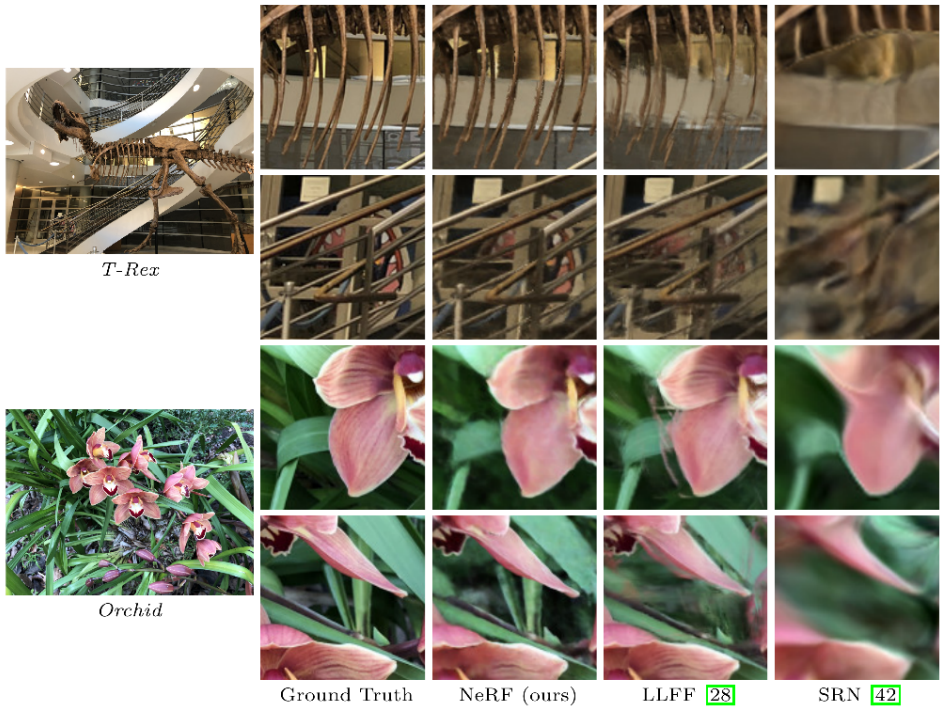


Video: `https://www.matthewtancik.com/nerf`

Training time

The optimization for a single scene typically take around 100– 300k iterations to converge on a single NVIDIA V100 GPU (about 1–2 days).

Results



Video: `https://www.matthewtancik.com/nerf`

Training time

The optimization for a single scene typically take around 100– 300k iterations to converge on a single NVIDIA V100 GPU (about 1–2 days). *(Faster variants released since: Instant NGP [Mueller 2022])*

A detour by variants of Positional encoding

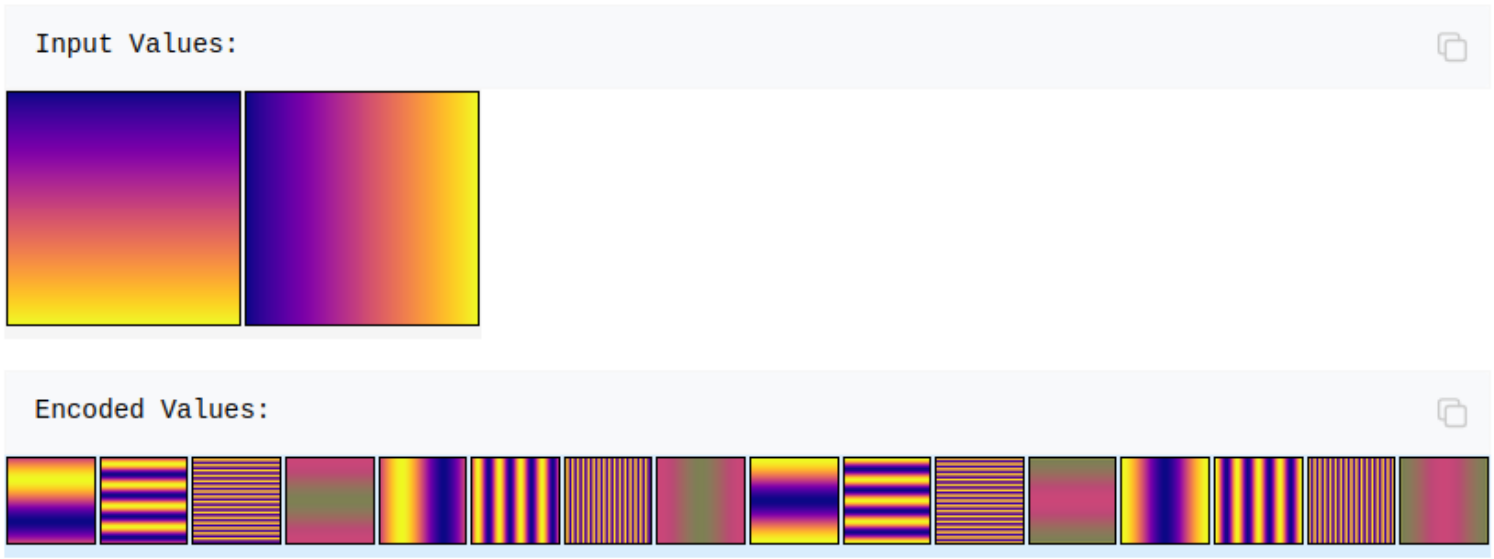
- Crucial ingredient of Nerf
- Allows to make it extremely fast: InstantNGP [Mueller et al. 2022]

Principle

Transform the input $x \in \mathbb{R}^d$ into $f(x) \in \mathbb{R}^N$ (with f fixed or trainable), and feed $f(x)$ as input to an MLP.

Frequency-based positional encoding [Mildenhall et al. 2020]

$$f(x) = (\cos x, \sin x, \cos 2x, \sin 2x, \cdots, \cos Nx, \sin Nx)$$

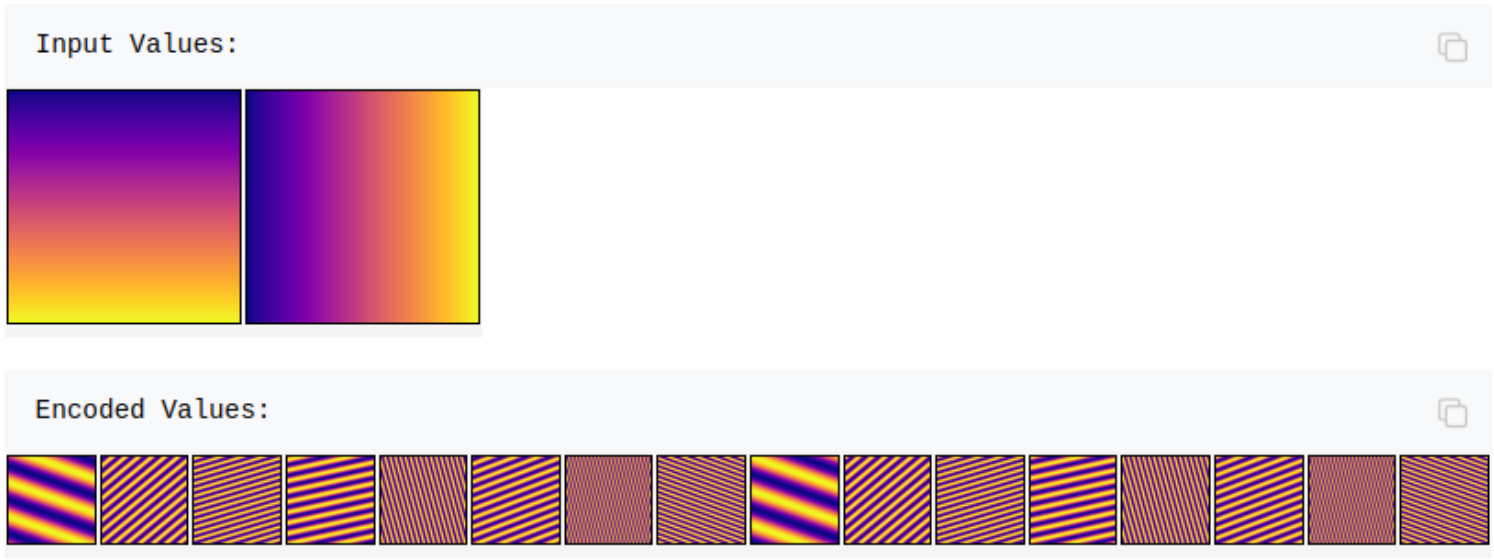


from nerfstudio

Random Fourier Features [Tancik et al. 2020]

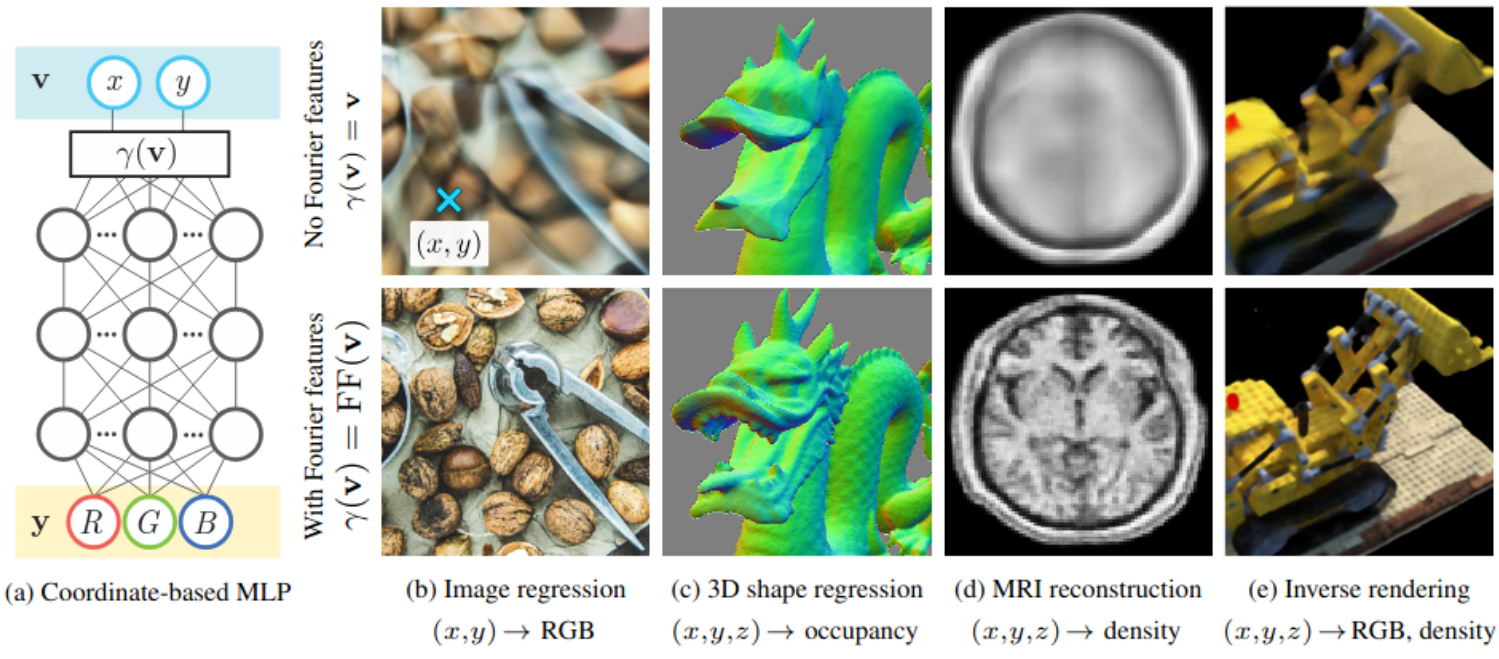
$$f(x) = (\cos 2\pi b_1^T x, \sin 2\pi b_1^T x, \cdots \cos 2\pi b_N^T x, \sin 2\pi b_N^T x, \cdots)$$

With b_i sampled from $\mathcal{N}(0, \sigma^2)$



from nerfstudio

Comparison



(a) Coordinate-based MLP

(b) Image regression $(x,y) \rightarrow$ RGB

(c) 3D shape regression $(x,y,z) \rightarrow$ occupancy

(d) MRI reconstruction $(x,y,z) \rightarrow$ density

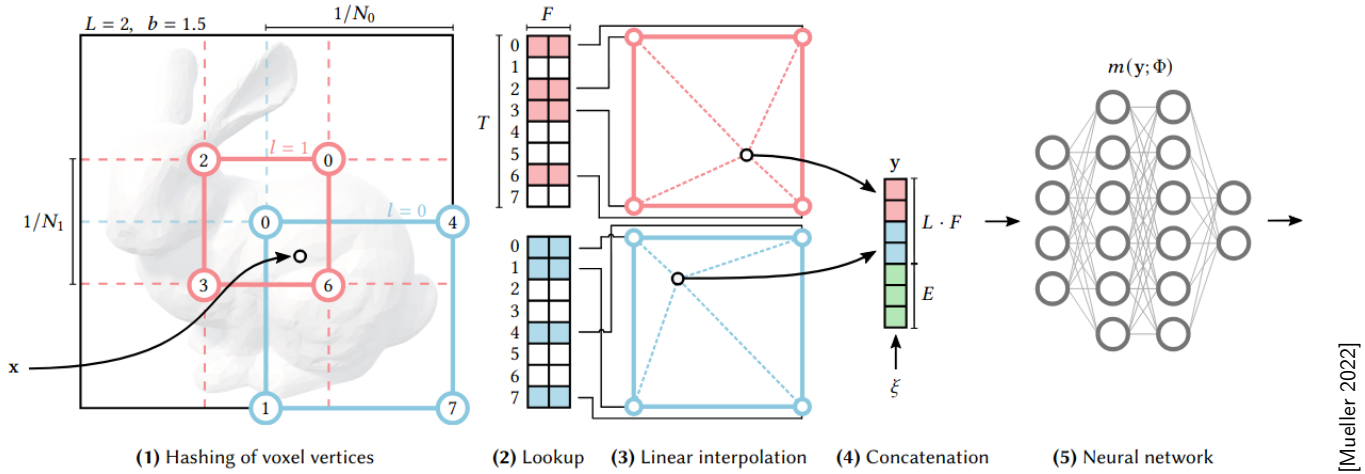
(e) Inverse rendering $(x,y,z) \rightarrow$ RGB, density

[Tancik 2020]

InstantNGP [Mueller 2022] - trainable positional encoding

- Improve quality and speed by learning the positional encoding.
- Multiresolution encoding
- Allows to use a tiny MLP 2 hidden layers - 64 neurons per layer
- Much faster to train!

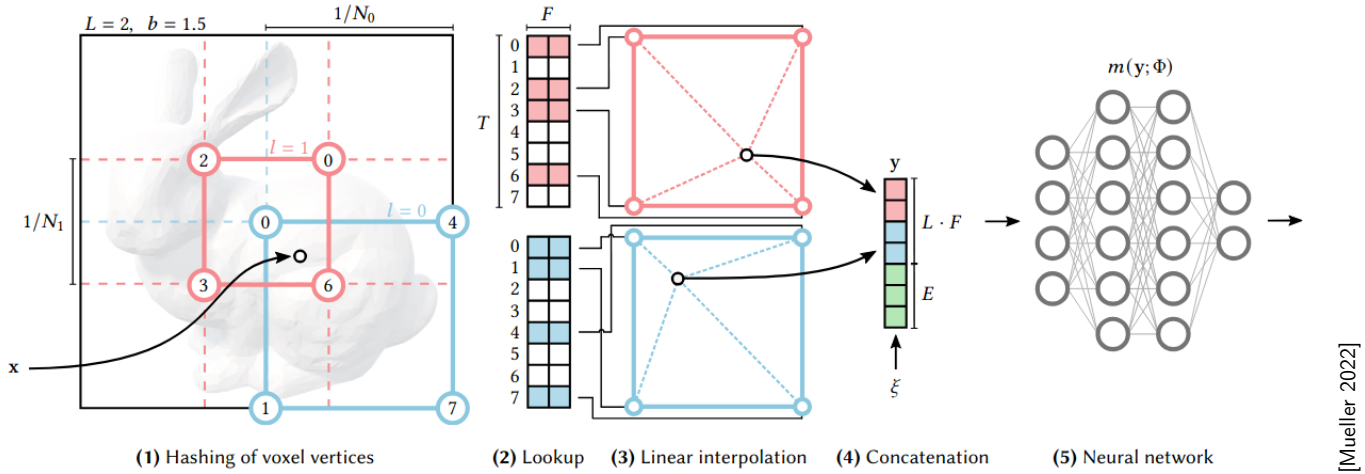
Hash-Encoding (1)



[Mueller 2022]

- L levels with resolution $N_0, \cdots N_{L-1}$
- Coarsest N_{min}; finest N_{max}. $N_l = \left\lfloor N_{min}(\frac{N_{max}}{N_{min}})^{l/(L-1)} \right\rfloor$
- $x \in \mathbb{R}^d$, xN_l falls inside a voxel, each of the 2^d corners mapped to an entry in a feature array
- Each feature level has fixed size T

Hash-Encoding (2)

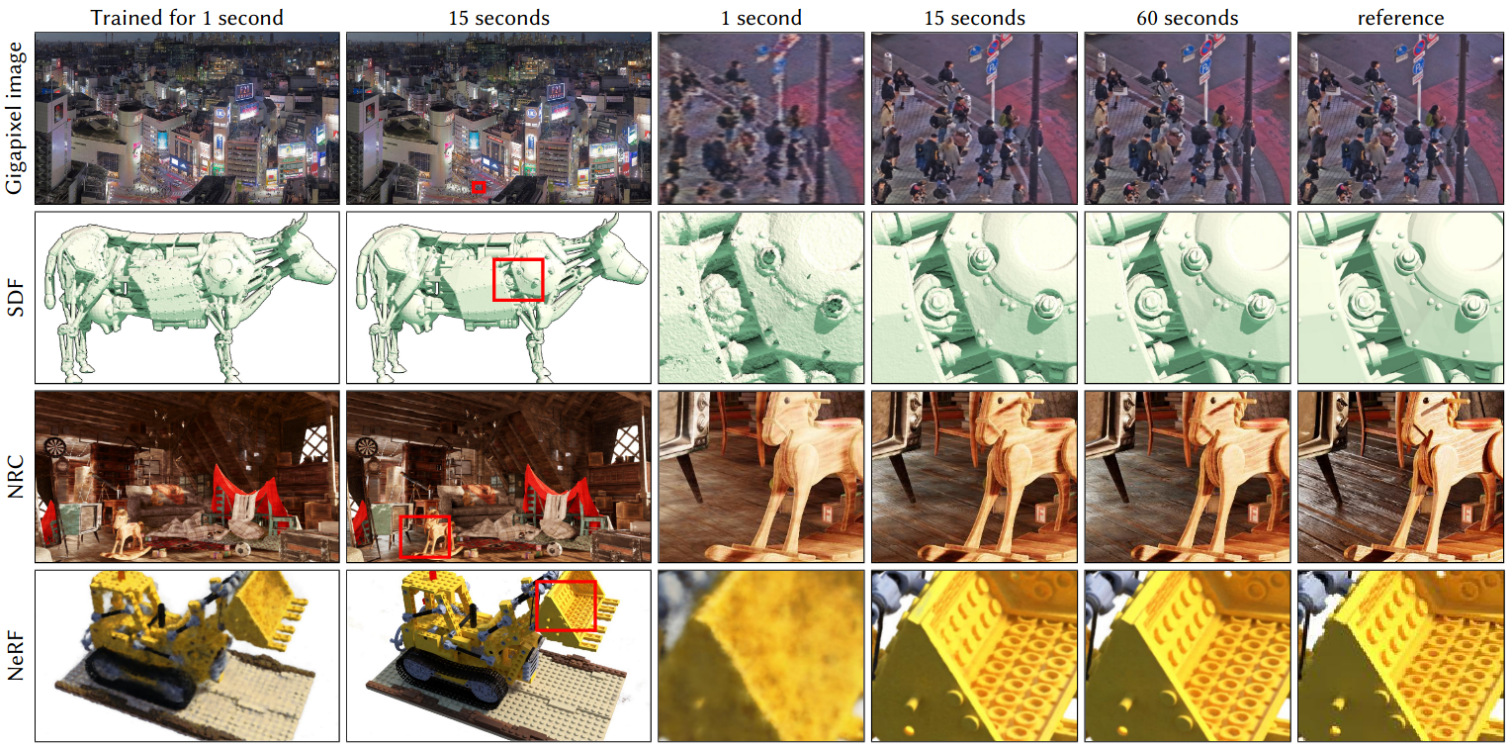


- hash function $h : \mathbb{Z}^d \to \mathbb{Z}/T\mathbb{Z}$

$$h(x) = (\bigoplus_{i=1}^{d} x_i \pi_i)[T]$$

- π_i large unique prime numbers
- No collision detection!
- Interpolation of the feature at x by trilinear interpolation
- L features of size F concatenated (also with non pos. information) fed into a small MLP.

Speed



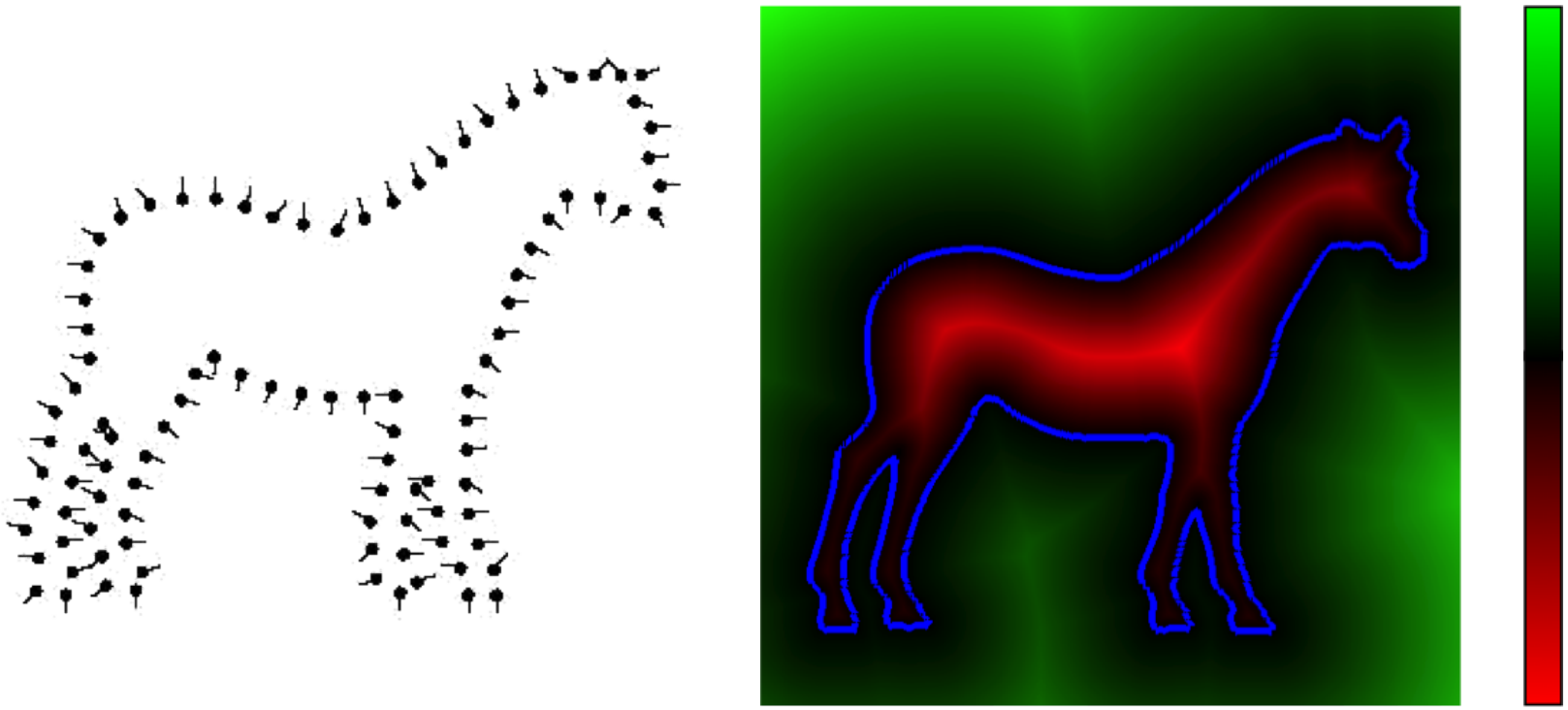
[Mueller 2022]

Quality comparison

[Mueller 2022]

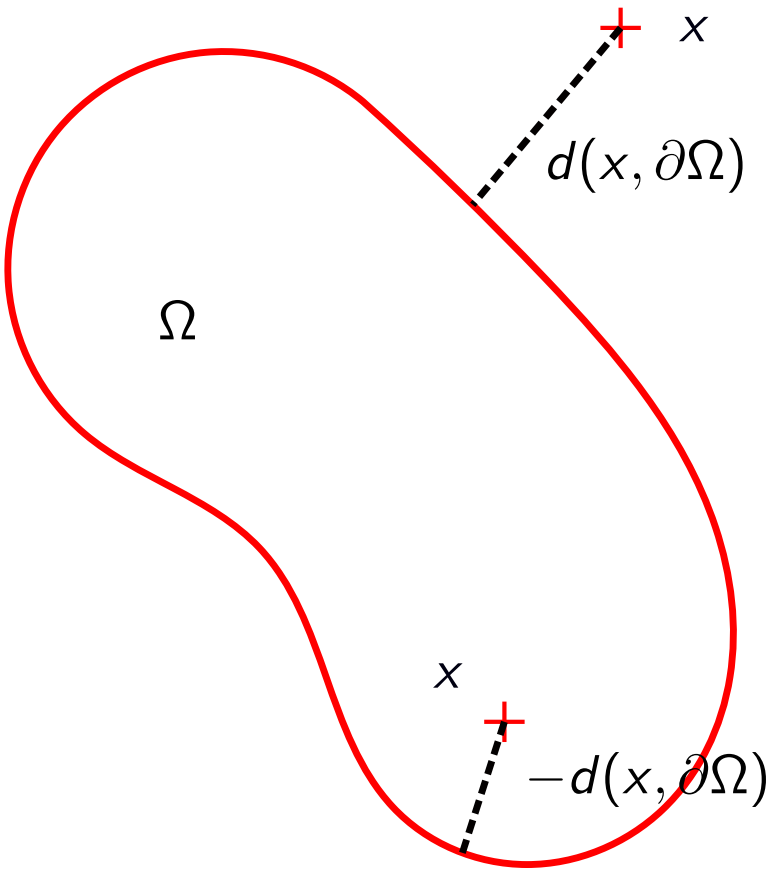
Outline

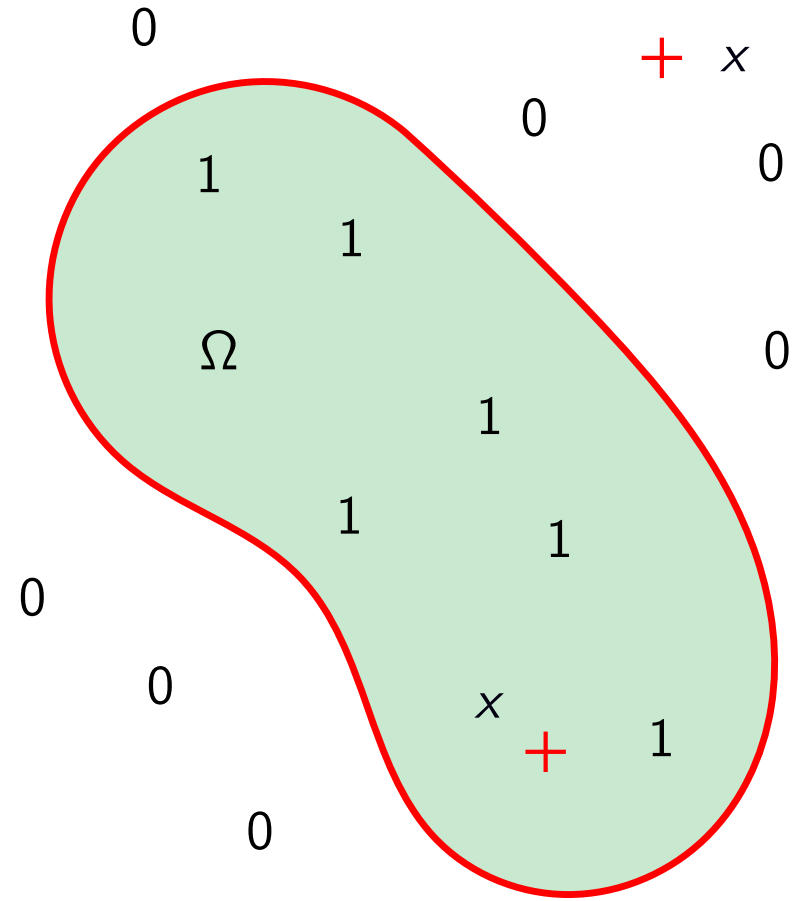
Implicit surface reconstruction - Principle



- See the surface as an isolevel of a given function
- Extract the surface by some contouring algorithm: Marching cubes [Lorensen Cline 87], Particle Systems [Levet et al. 06]

Implicit functions are not necessarily distance fields





Surface reconstruction from unorganized points [Hoppe et al. 92]

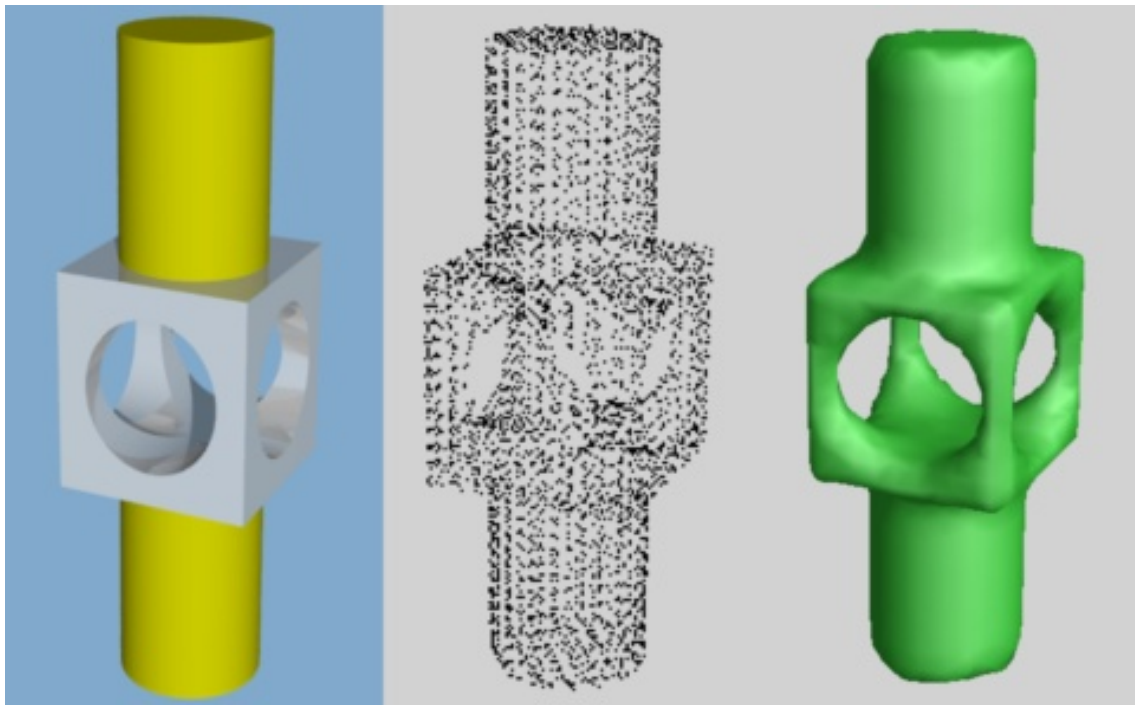
- Input: a set of 3D points
- *Idea:* for points on the surface the signed distance transform has a gradient equal to the normal

$$F(p) = \pm \min_{q \in \mathcal{S}} \|p - q\|$$

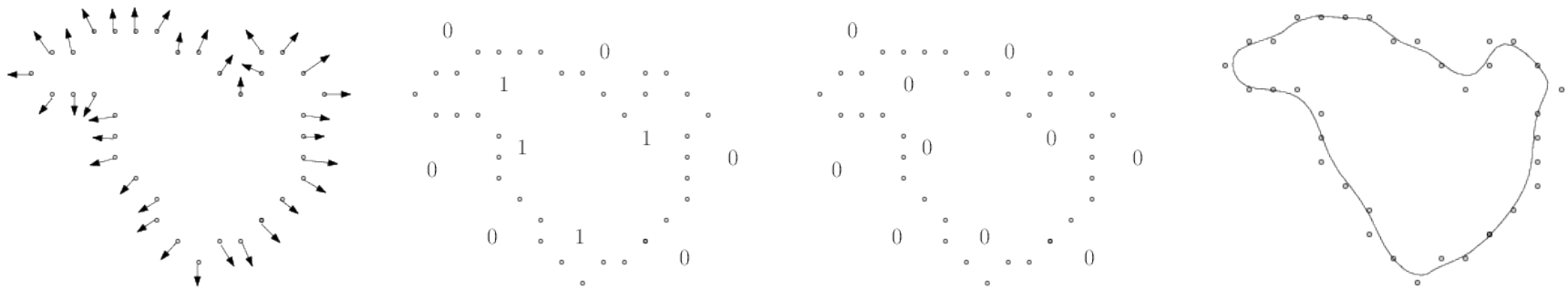
- 0 is a regular value for F and thus the isolevel extraction will give a manifold
- Compute an associated tangent plane (o_i, n_i) for each point p_i of the point set
- Orientation of the tangent planes as explained before.

Surface reconstruction from unorganized points [Hoppe et al. 92]

- Once the points are oriented
- For each point p, find the closest centroid o_i
- Estimated signed distance function: $\hat{f}(p) = n_i \cdot (p - o_i)$



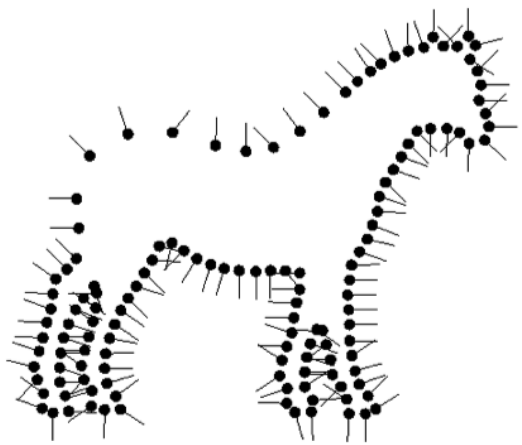
Poisson Surface Reconstruction [Kazhdan et al. 2006]

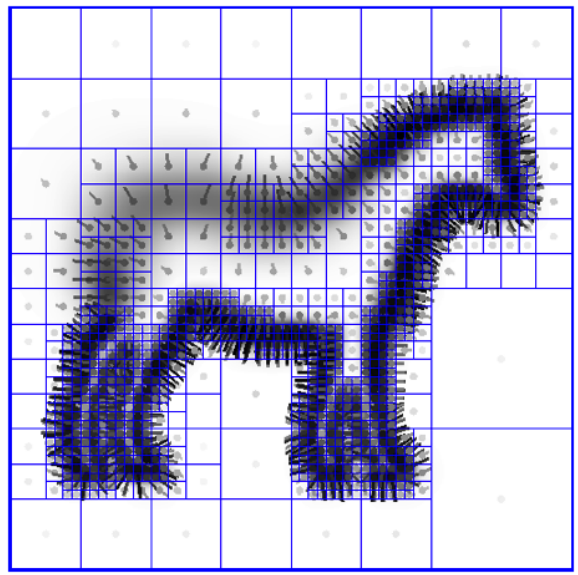


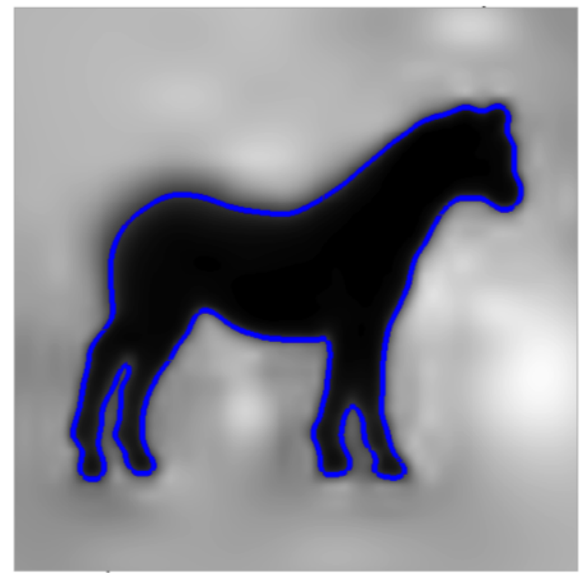
- Input: a set of oriented samples
- Reconstructs the indicator function of the surface and then extracts the boundary.
- Trick: Normals sample the function's gradients

Poisson Surface Reconstruction [Kazhdan et al. 2006]

1. Transform samples into a vector field
2. Fit a scalar-field to the gradients
3. Extract the isosurface







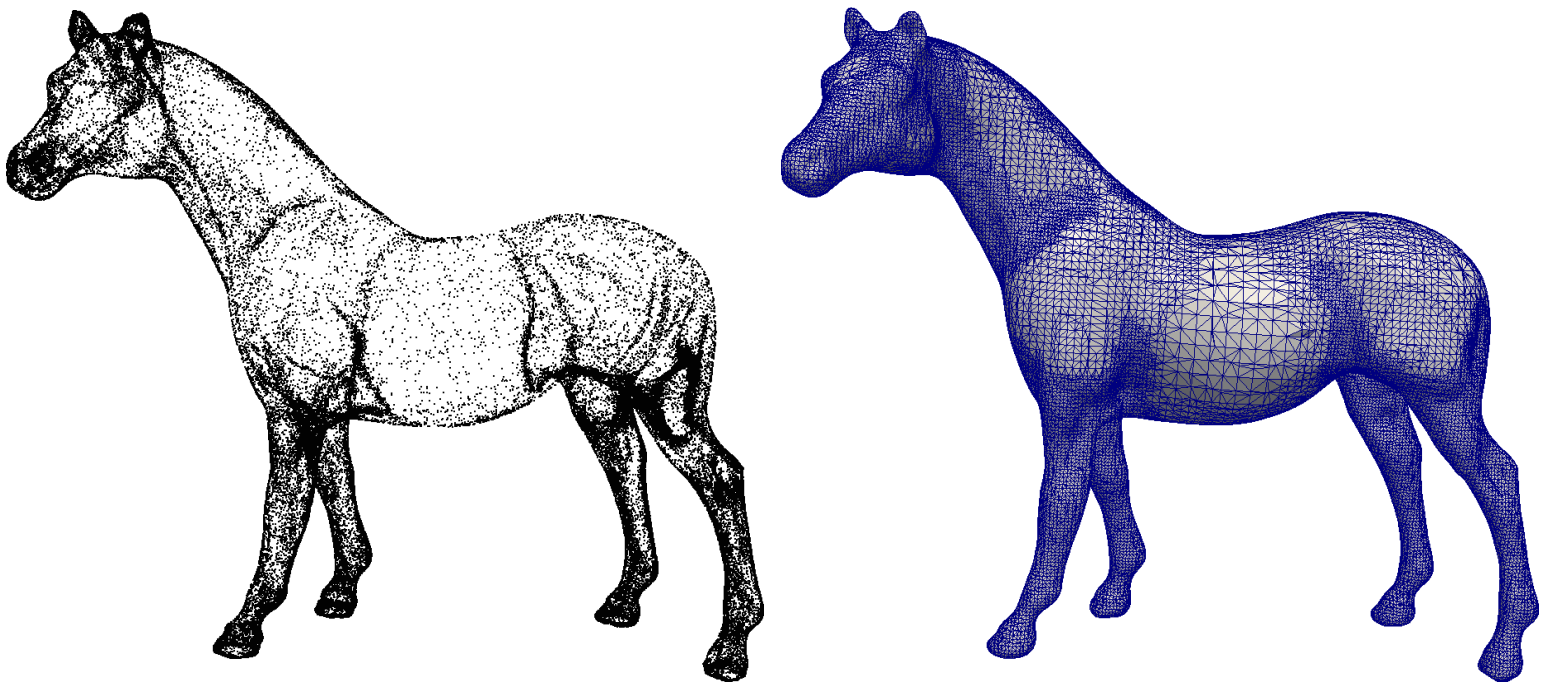
Poisson Surface Reconstruction [Kazhdan et al. 2006]

- To fit a scalar field χ to gradients $\vec{V}$, solve:

$$\min_{\chi} \|\nabla\chi - \vec{V}\|$$

- Eq to:

$$\nabla \cdot (\nabla\chi) - \nabla \cdot \vec{V} = 0 \Leftrightarrow \Delta\chi = \nabla \cdot \vec{V}$$



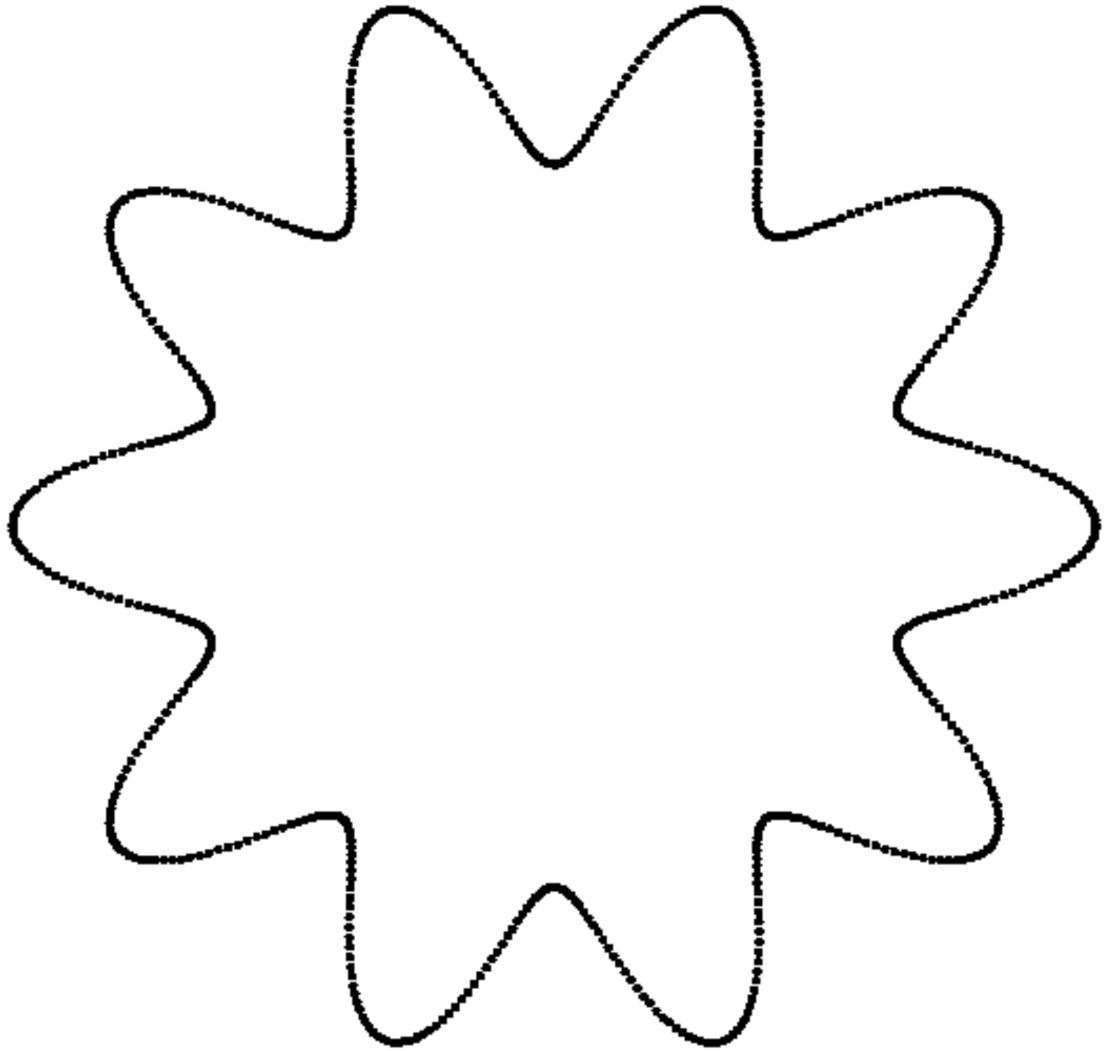
From the signed distance function to the mesh

- At each point in $\mathbb{R}^3$, the signed distance function to the surface can be estimated
- Extract the 0 levelset of this function: points where this function is 0

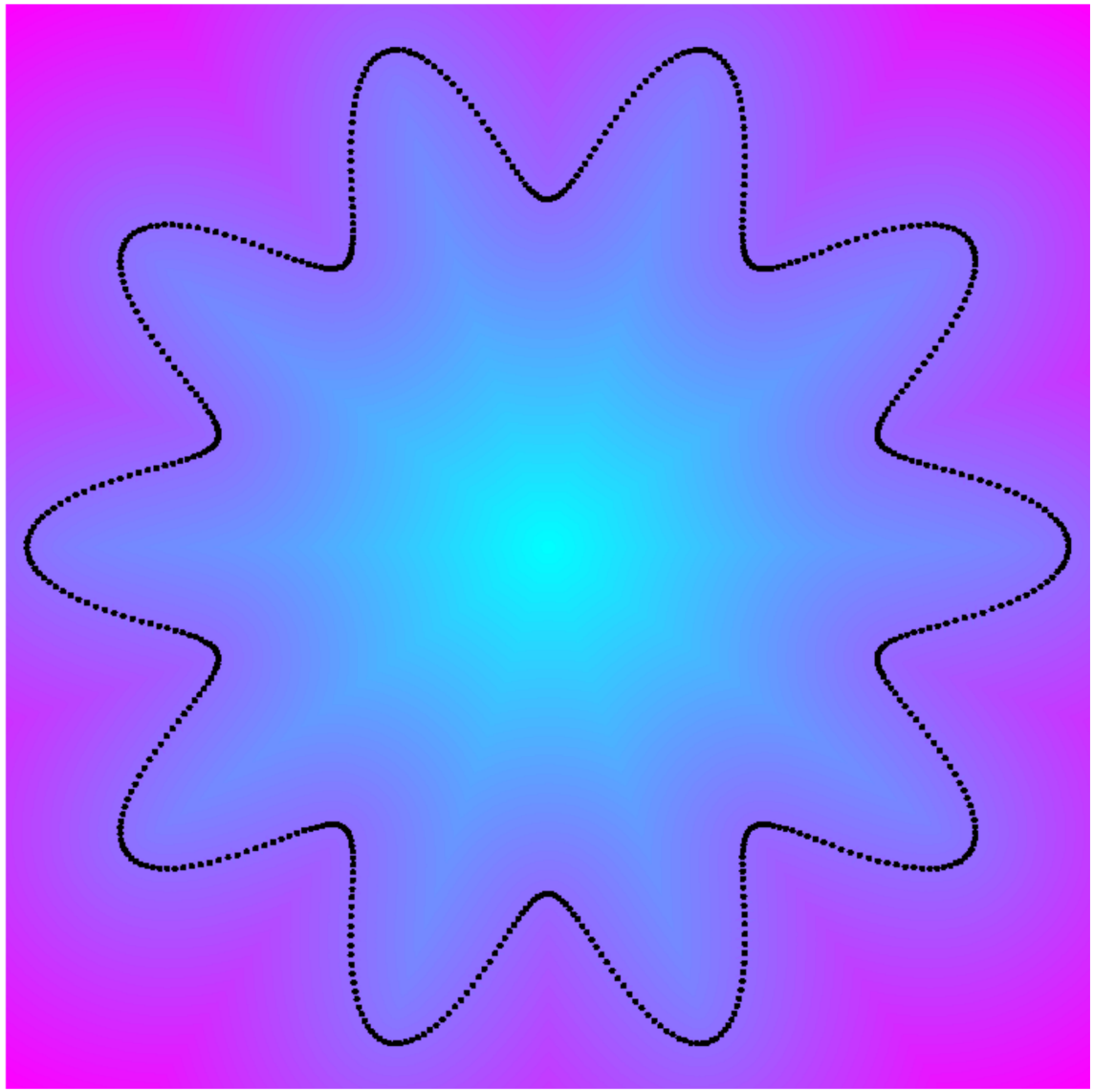
Approximation

Evaluate the function at the vertices of a grid and deduce the local geometry of the surface in each grid cube.

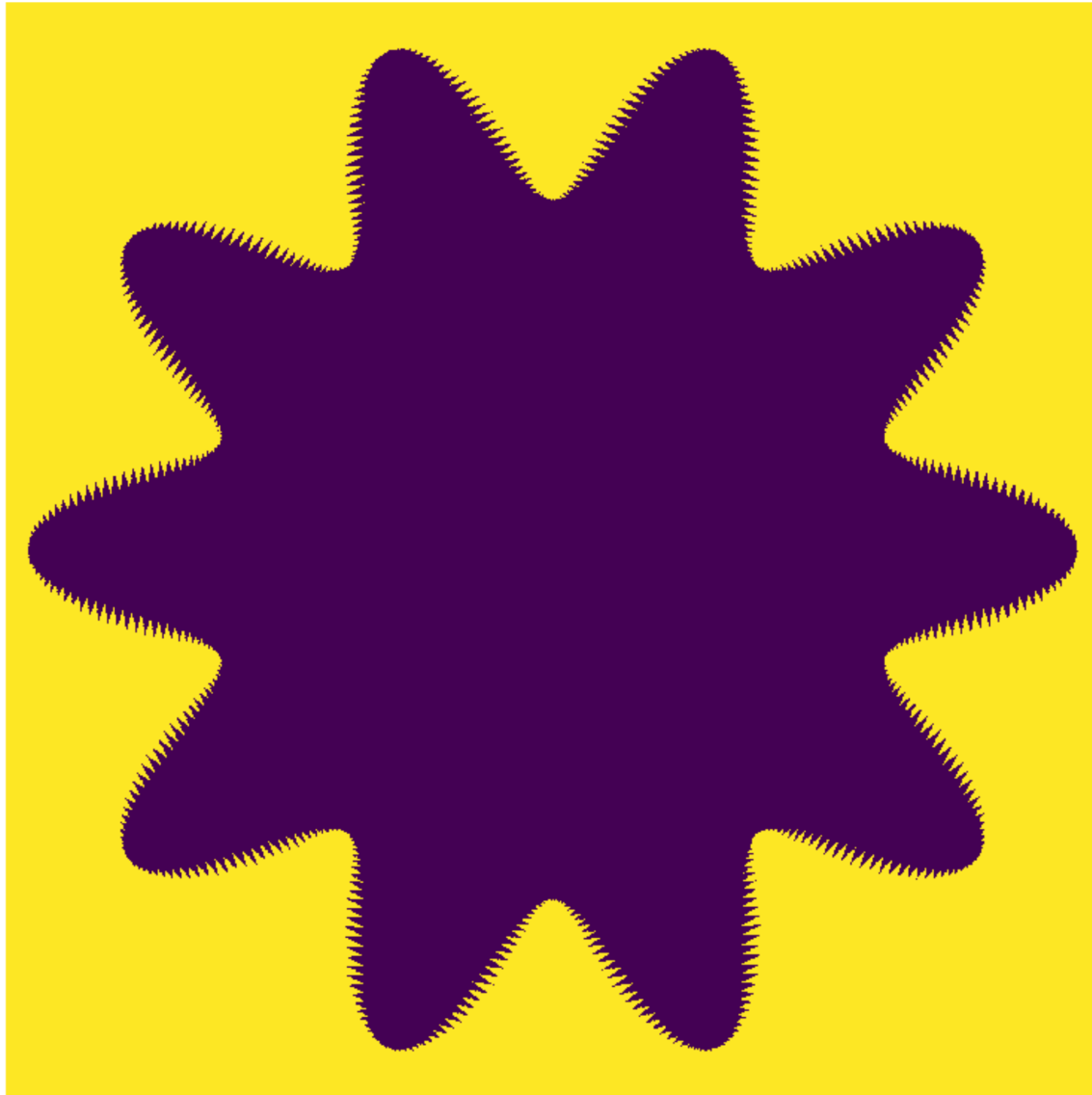
Example in 2D



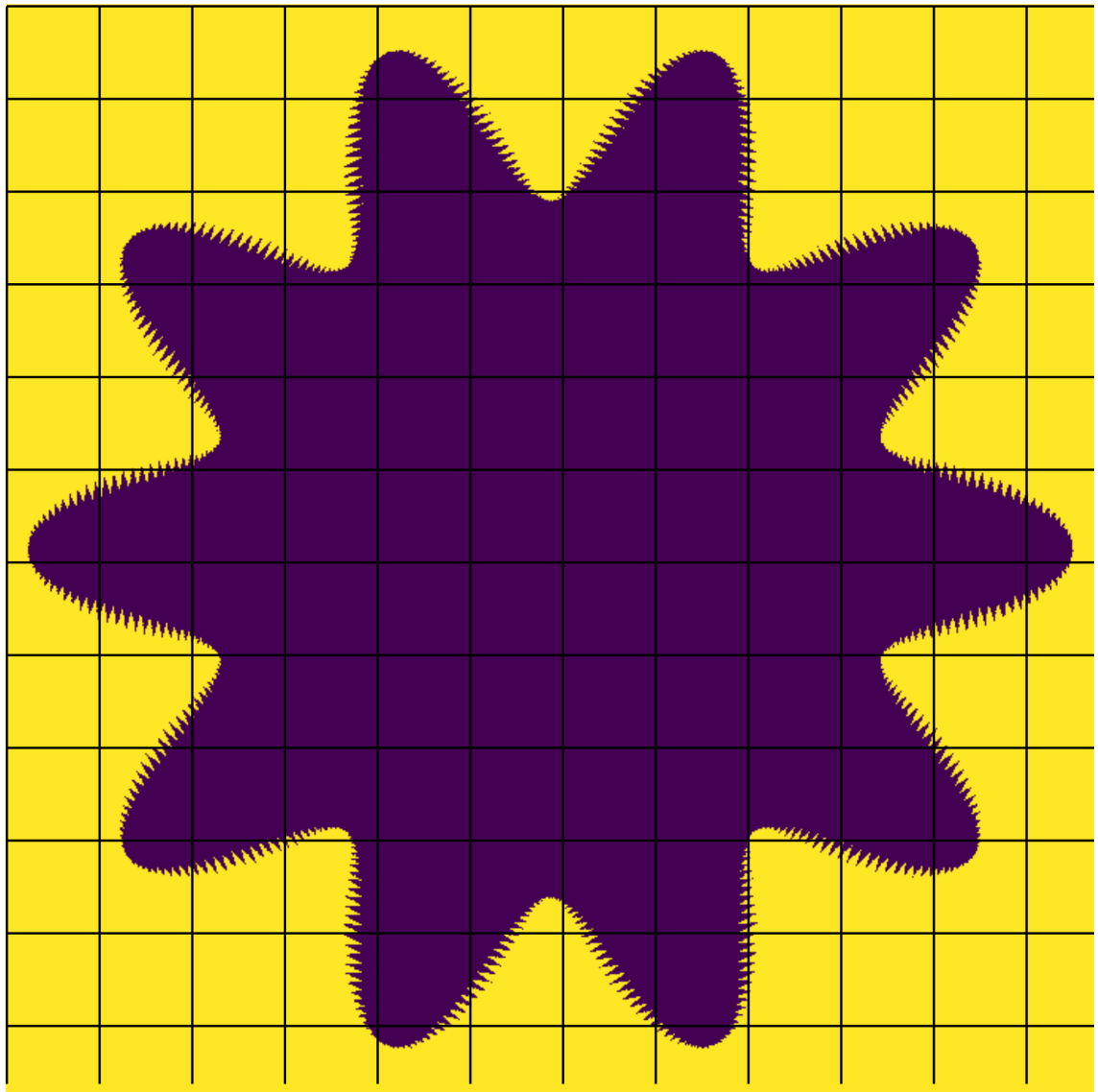
Example in 2D



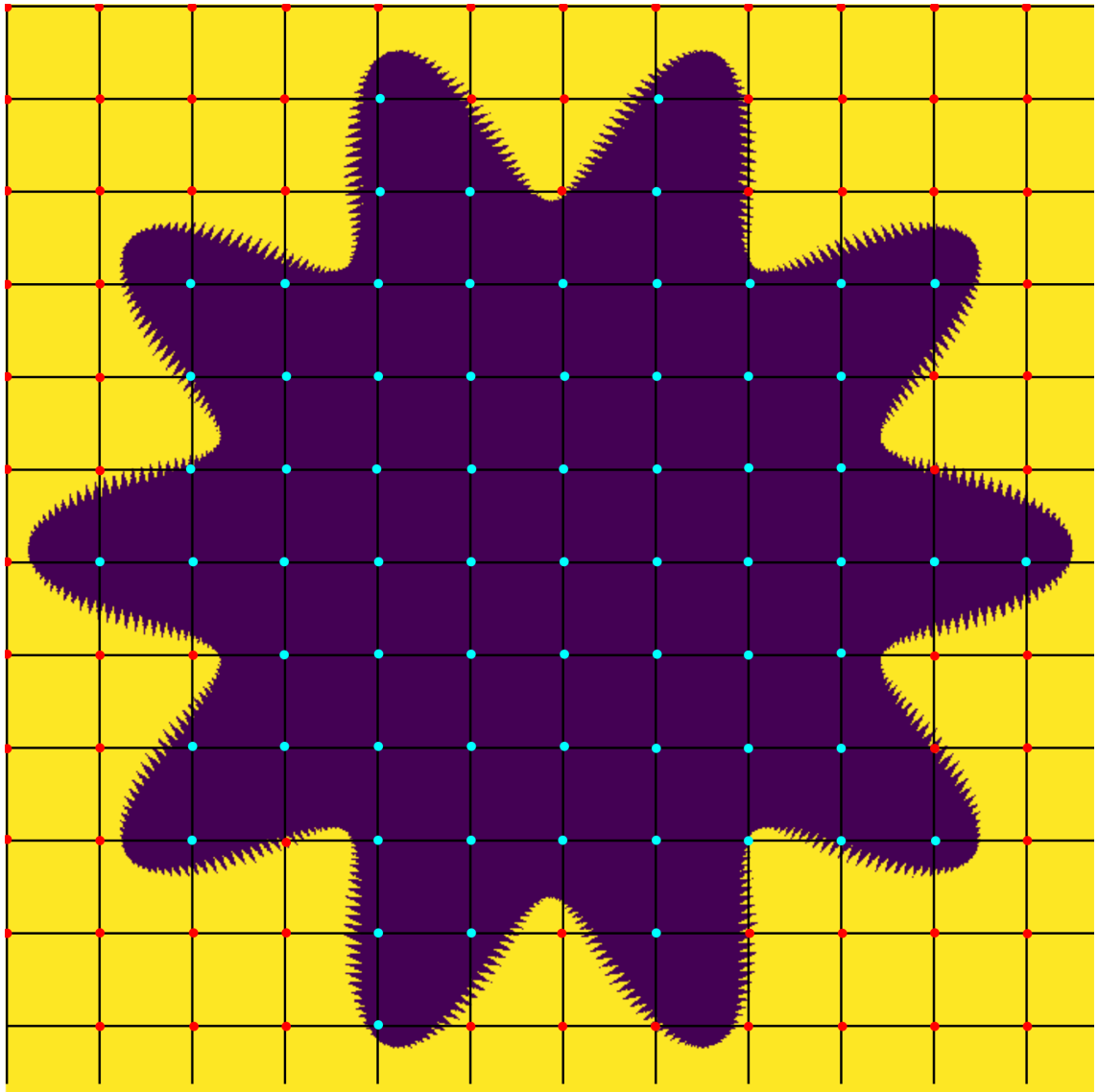
Example in 2D



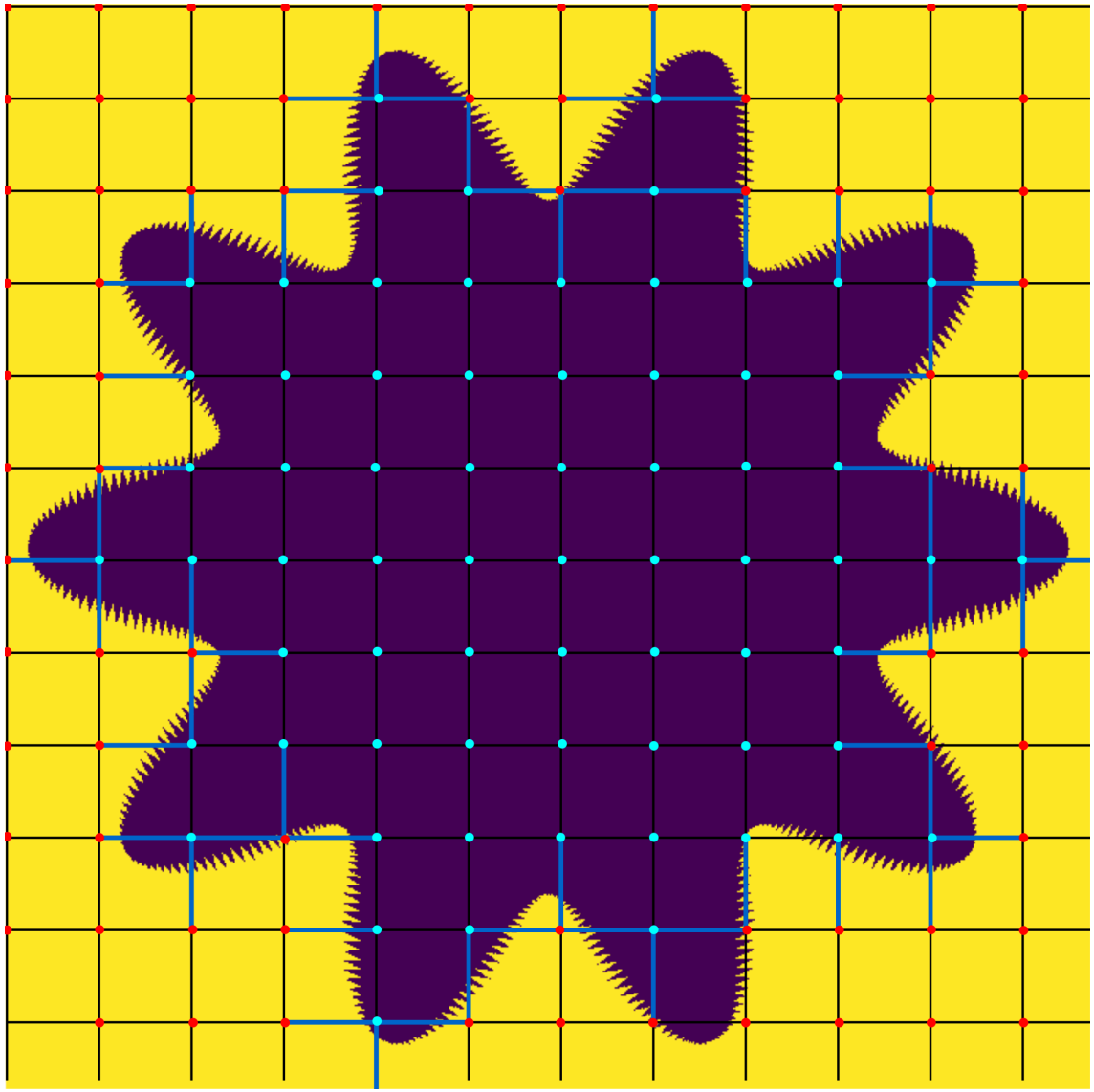
Example in 2D



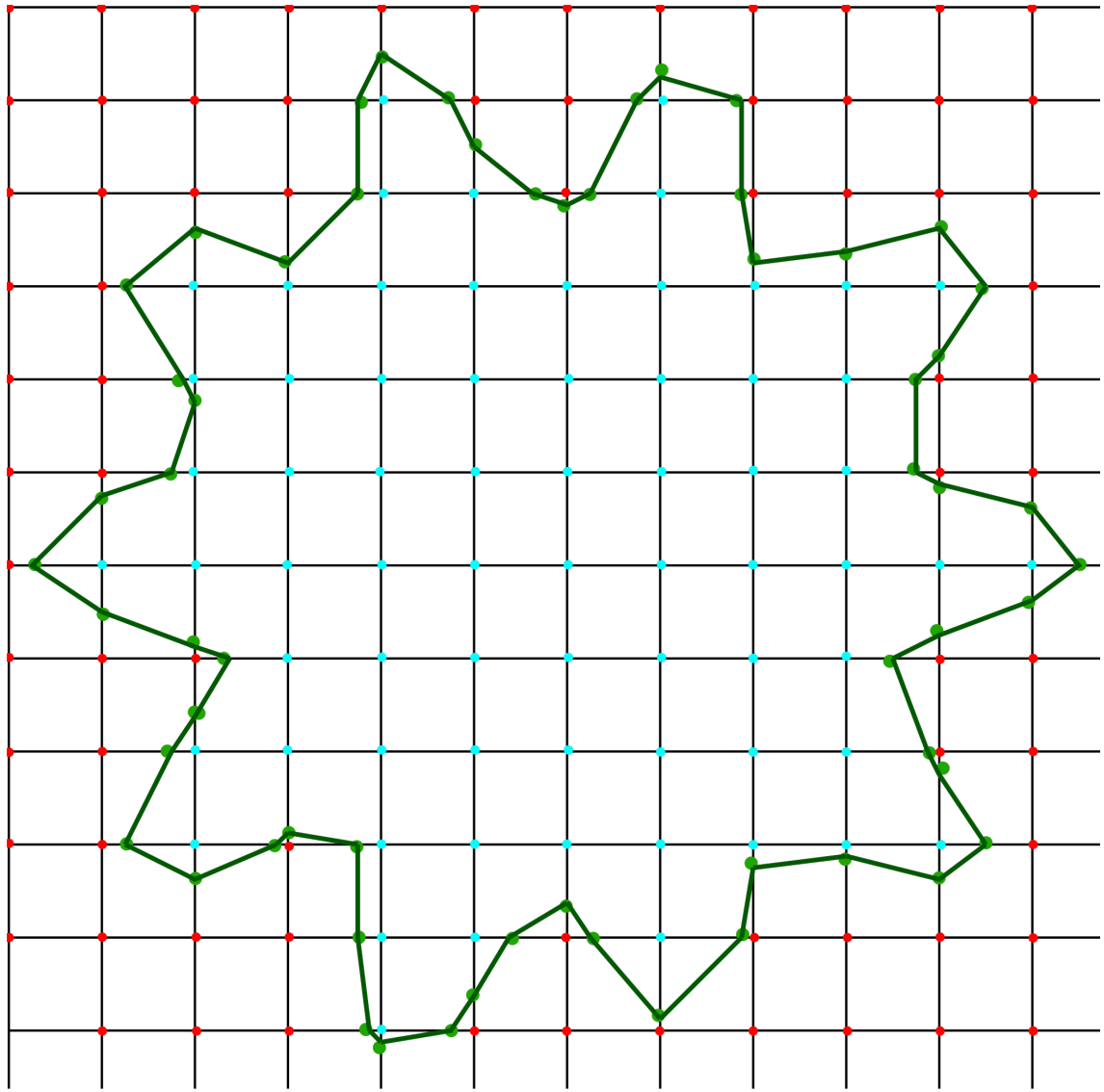
Example in 2D



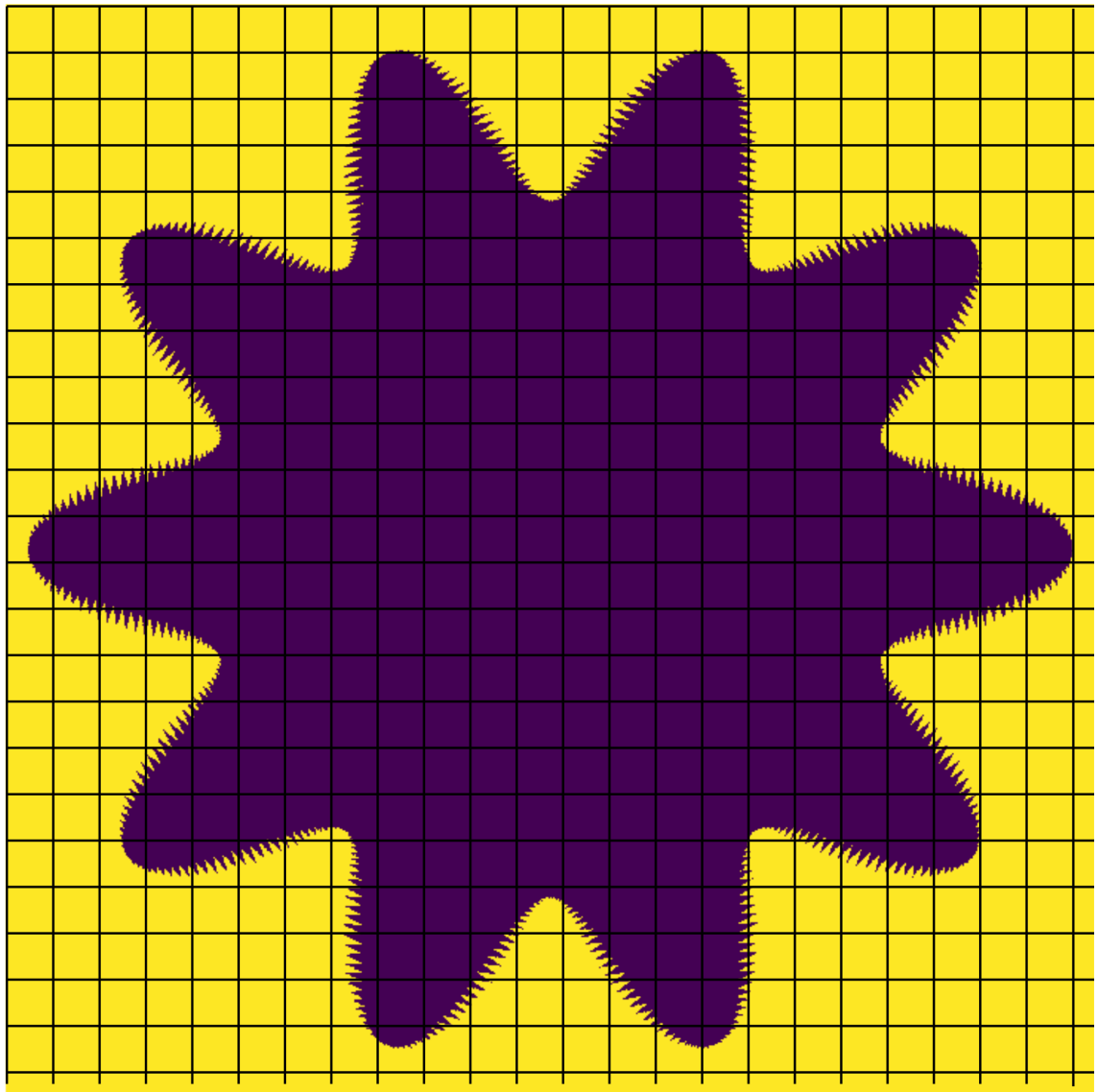
Example in 2D



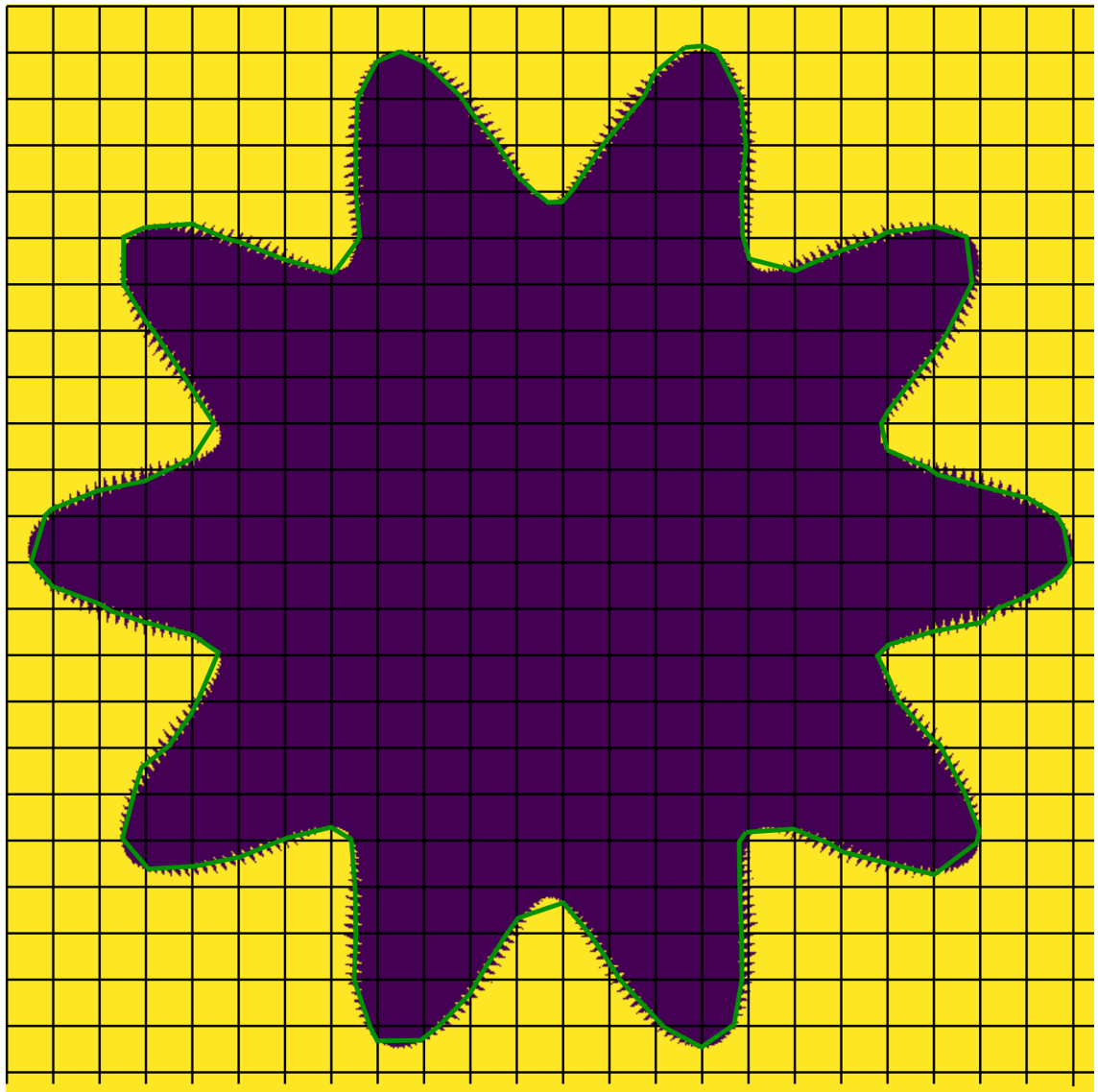
Example in 2D



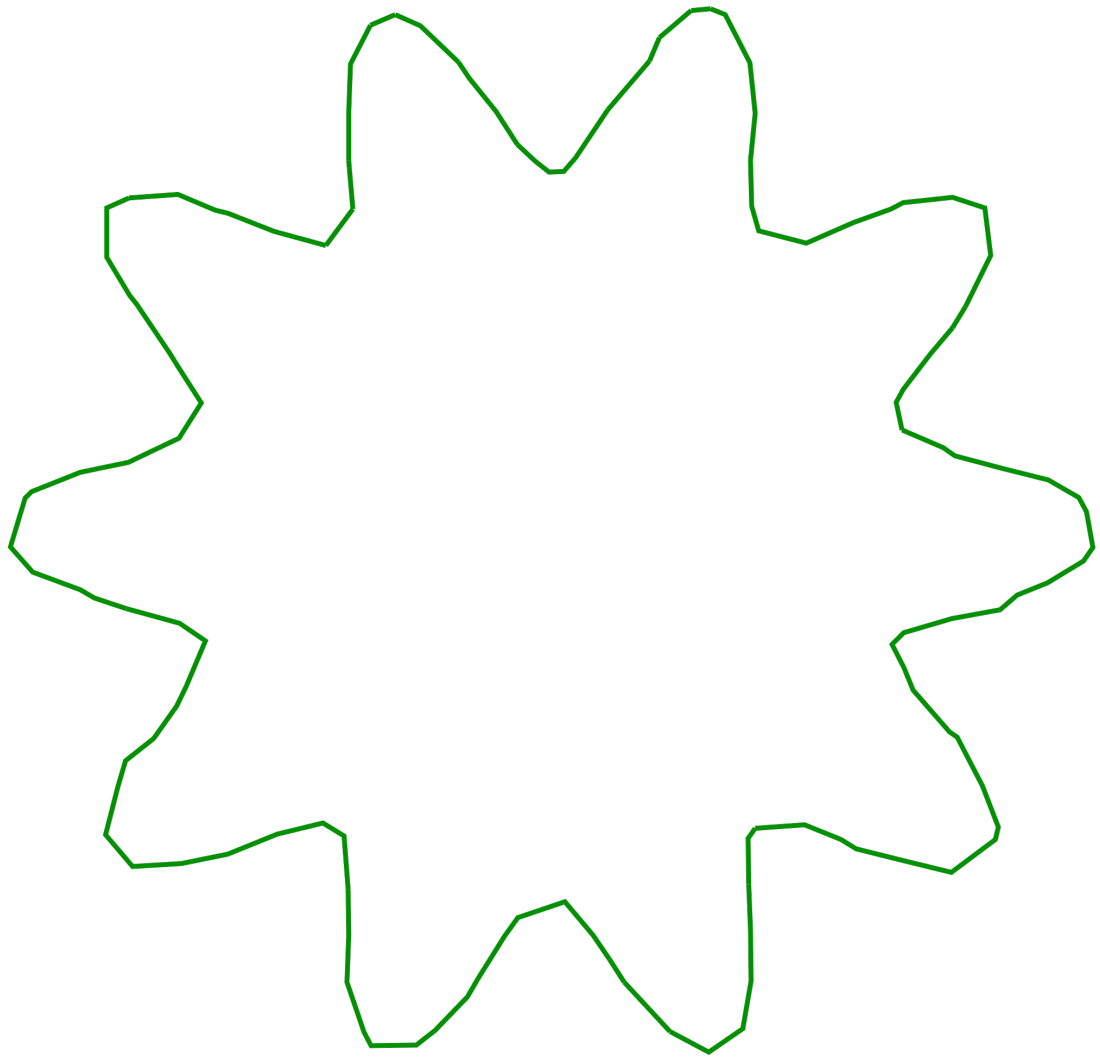
Example in 2D



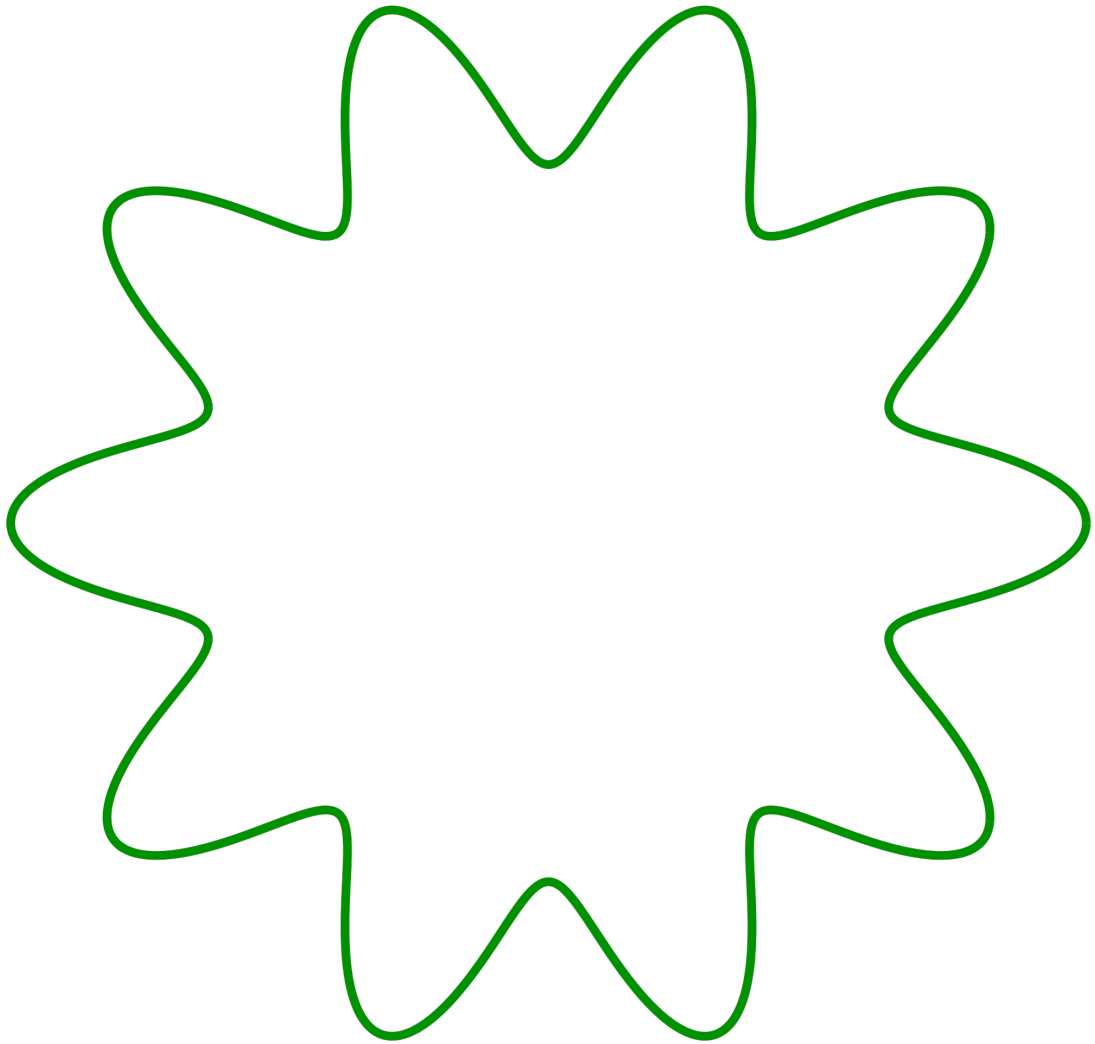
Example in 2D



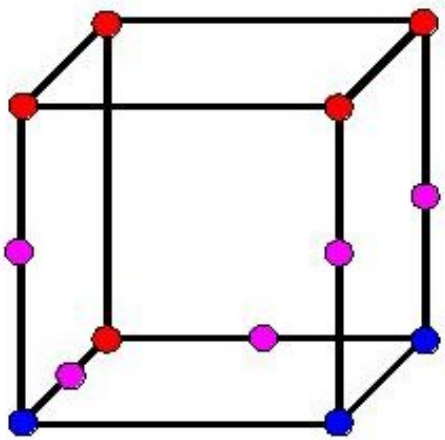
Example in 2D

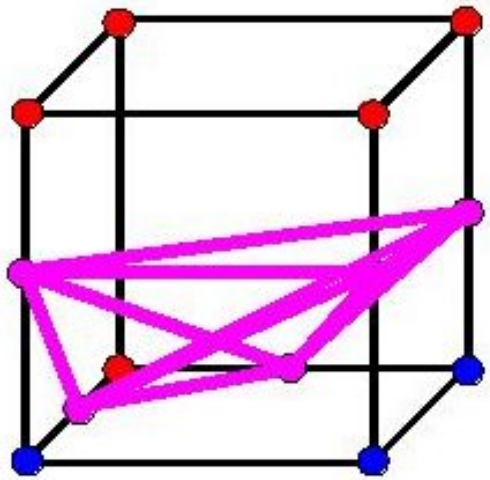


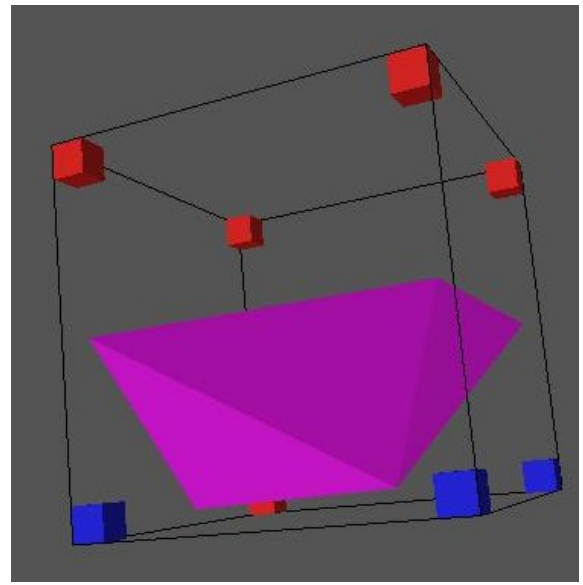
Example in 2D



From Marching Squares to Marching Cubes







Images by Ben Anderson

Drawing lines between intersection points is ambiguous and does not give a surface patch.

Look-up tables

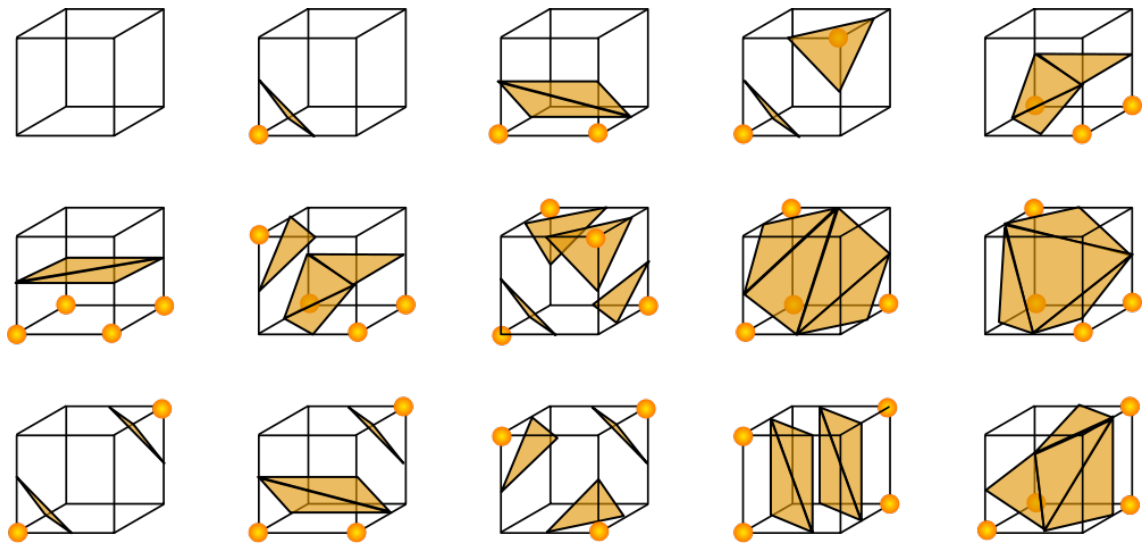
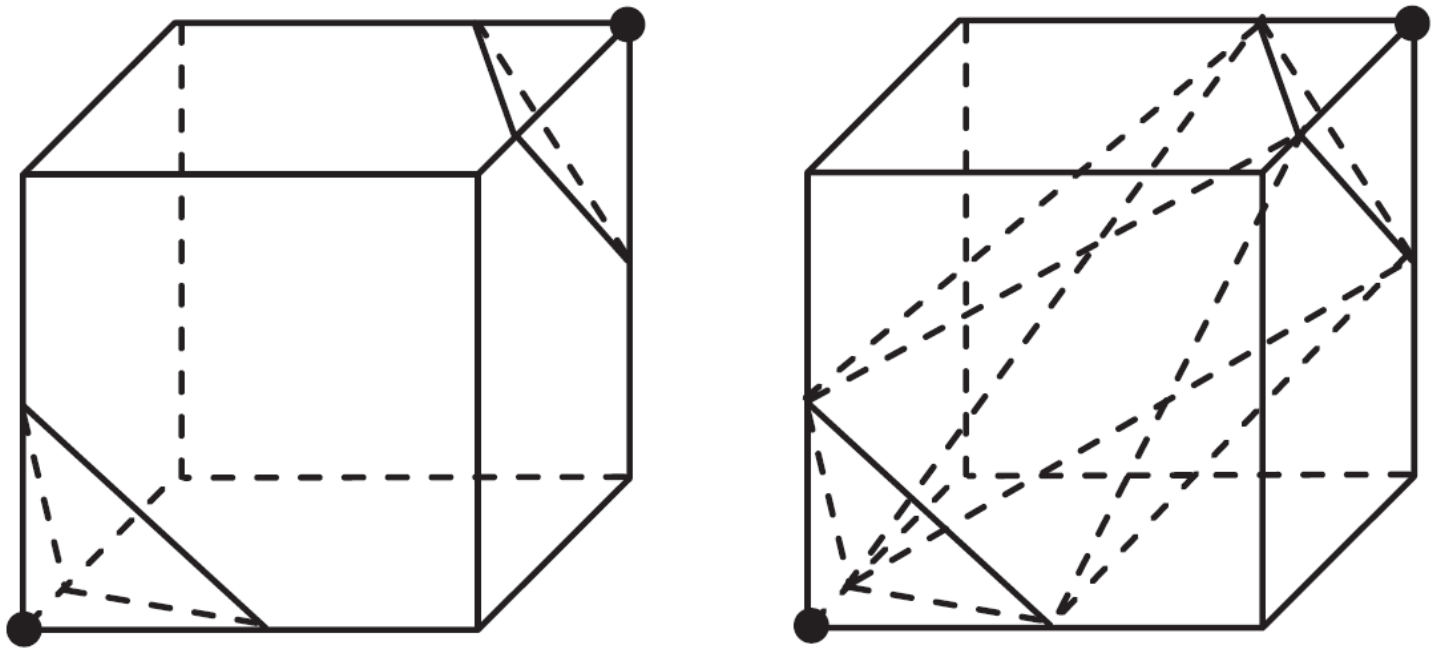


Image by Jmtrivial (Wikipedia)

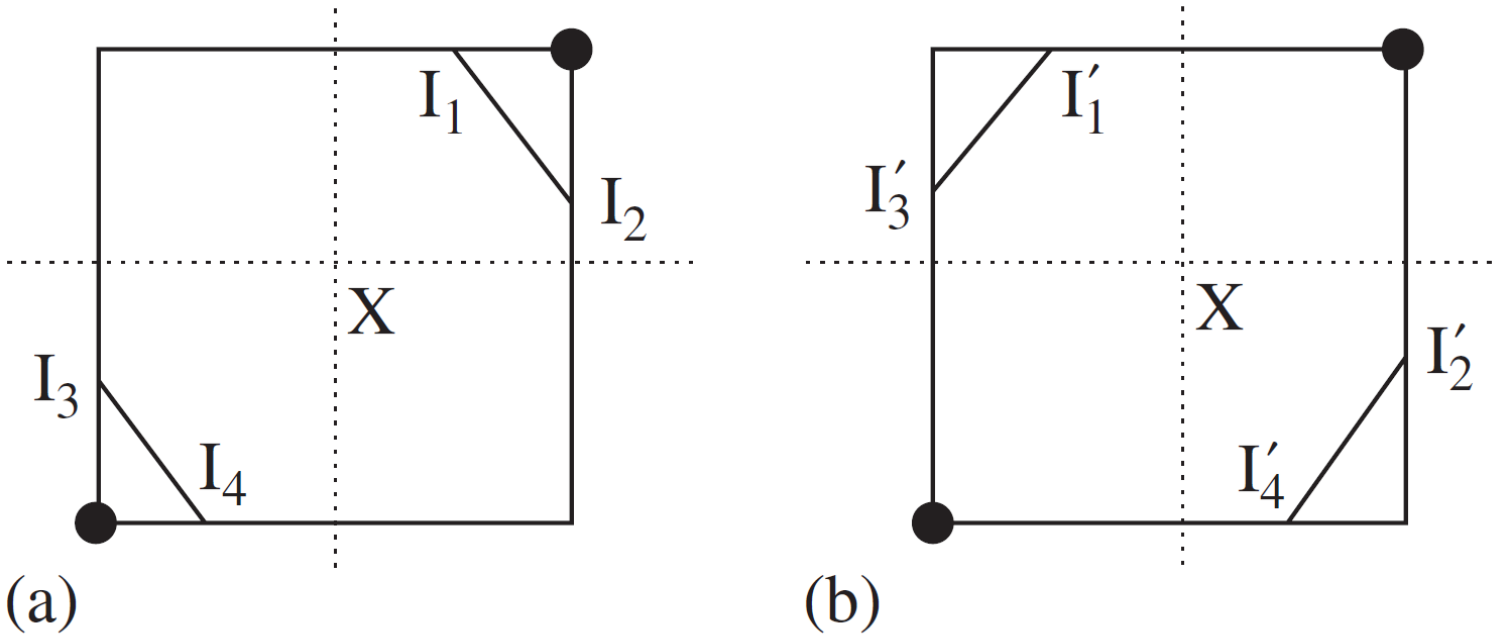
- There are $2^8 = 256$ possible cases for cube corner values.
- By symmetry + rotation arguments it reduces to 15 cases.
- Build a look-up table giving the grid cell triangulation based on the corner values case.

Ambiguous cases



(a) (b)

Ambiguous cases

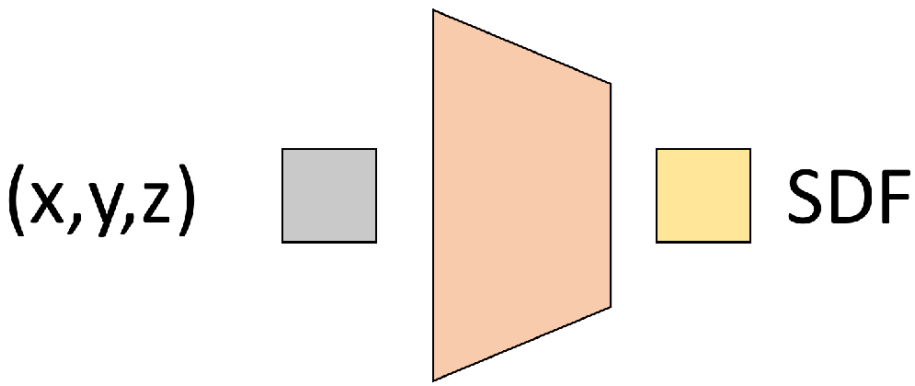


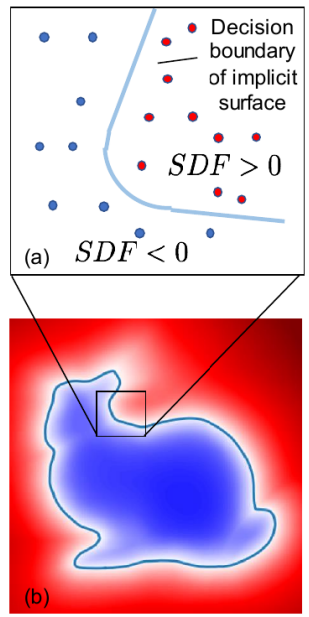
- Refine the grid to remove ambiguation
- Switch to marching tetrahedra algorithm

Outline

Learning a signed distance [DeepSDF - Park et al. 2019]

- Learn a SDF u_θ to a shape $\mathcal{X}$, knowing a set of points $x_i \in X$.





(c)

Learning

- Input data: a set of points y_i in $\mathbb{R}^3$ and their distance to the surface $s_i = SDF(y_i)$

Loss function

$$\mathcal{L}(\theta) = \sum_i |clamp(u_\theta(y_i), \delta) - clamp(s_i, \delta)|$$

with $clamp(h, \delta) = \min(\delta, \max(-\delta, h))$.

- δ controls the width of the region of interest around the surface. In practice $\delta = 0.1$.

Learning

- Input data: a set of points y_i in $\mathbb{R}^3$ and their distance to the surface $s_i = SDF(y_i)$

Loss function

$$\mathcal{L}(\theta) = \sum_i |clamp(u_\theta(y_i), \delta) - clamp(s_i, \delta)|$$

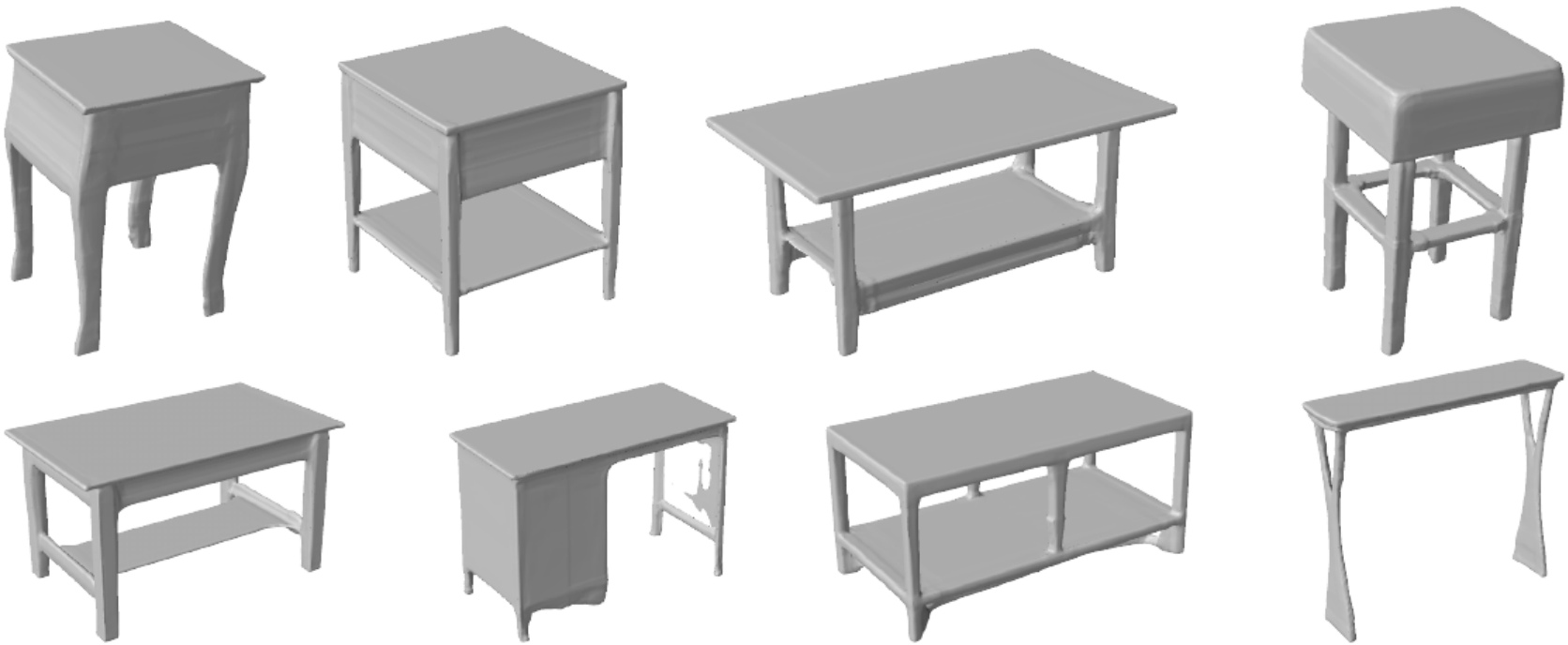
with $clamp(h, \delta) = \min(\delta, \max(-\delta, h))$.

- δ controls the width of the region of interest around the surface. In practice $\delta = 0.1$.

Architecture

8 layers MLP, (width 512), dropout, ReLU activation function (tanh for the last layer) + weight normalization.

Results



[Park 2019]

Learning an occupancy function [Mescheder 2019]

Occupancy function

Given an object as a compact subset $\Omega \subset \mathbb{R}^3$, the occupancy function $u : \mathbb{R}^3 \to 0, 1$ is such that:

$$u_\theta(x) = \begin{cases} 1 & \text{if } x \in \Omega \\ 0 & \text{otherwise} \end{cases}$$

- Neural network will learn a function $u_\theta(x)$ predicting whether u is inside Ω or outside Ω

Losses

- Input data: a set of points y_i in $\mathbb{R}^3$ and their positions relatively to the surface $o_i = 0$ or 1.

Loss function

$$\mathcal{L}(\theta) = \sum_i BCE(u_\theta(y_i), o_i)$$

- This is the single shape loss. Occupancy networks are mostly used in the context of latent shape spaces, see next course for more details!

Results

[Mescheder 2019]

Outline

Implicit neural field

- Signed distance field u to a surface S satisfies the Eikonal equation:

$$\|\nabla u\| = 1 \text{ with } u(x) = 0 \ \forall x \in \partial S$$

Implicit neural field

- Signed distance field u to a surface S satisfies the Eikonal equation:

$$\|\nabla u\| = 1 \text{ with } u(x) = 0 \ \forall x \in \partial S$$

- Since a MLP is differentiable use the Eikonal equation as a loss function [Gropp 2020]

Optimization Process

- Input data a set of points $(x_i, \mathsf{n}_i), i \in I$
- Look for u continuous and a.e. $\mathcal{C}^1$ such that:

$$\begin{cases} \|\nabla u\| & = 1 \\ u_{|\partial\Omega} & = 0 \\ \nabla u_{|\partial\Omega} & = \mathsf{n} \end{cases} \tag{1}$$

- Loss [Gropp 2020]

$$l(\theta) = \frac{1}{|I|} \sum_{i \in I} (|u_\theta(x_i)| + \tau \|\nabla u_\theta(x_i) - \mathsf{n}_i\|) + \lambda \mathbb{E}_x[(\|\nabla u_\theta(x)\| - 1)^2]$$

Periodic Activation Functions [Sitzmann 2021]

- Replace ReLU by periodic activation function $x \to \sin(\omega x)$. Better differentiability
- Loss:

$$\mathcal{L}_{sdf} = \frac{1}{|I|}\sum_{i\in I}(|u_\theta(x_i)| + \tau\|\nabla u_\theta(x_i) - \mathsf{n}_i\|)$$
$$+ \lambda\mathbb{E}_x[(\|\nabla u_\theta(x)\| - 1)^2] + \lambda_2\mathbb{E}_{x\notin\Omega}[(\|\psi(u_\theta(x)\|]$$

with $\psi(u_\theta(x)) = \exp -\alpha|u_\theta(x)|$; $\alpha >> 1$

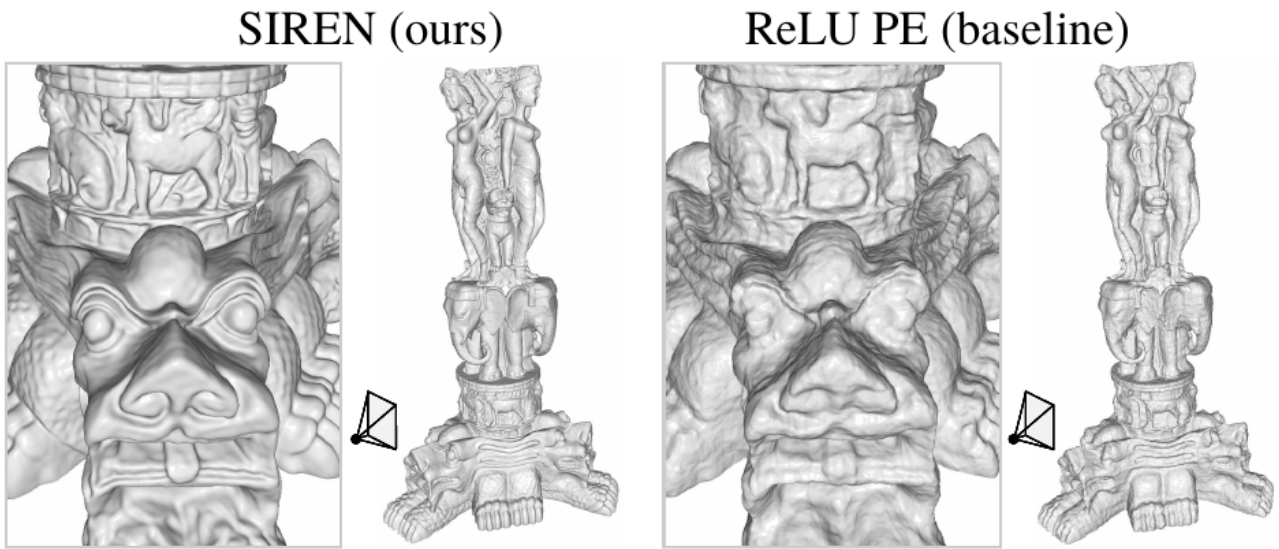


Figure 4: A comparison of SIREN used to fit a SDF from an oriented point clouse against the same fitting performed by an MLP using a ReLU PE (proposed in [35]).

From [Sitzmann 2020]

Periodic Activation Functions [Sitzmann 2021]

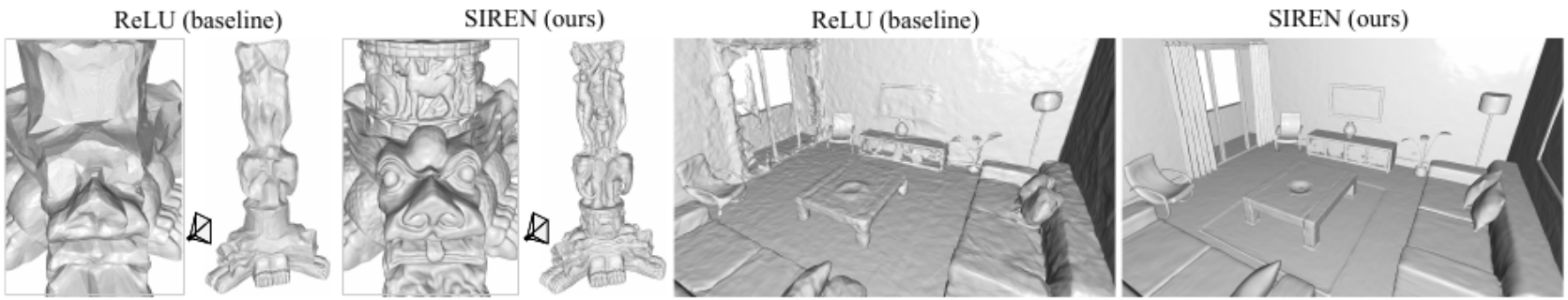


Figure 4: Shape representation. We fit signed distance functions parameterized by implicit neural representations directly on point clouds. Compared to ReLU implicit representations, our periodic activations significantly improve detail of objects (left) and complexity of entire scenes (right).

From [Sitzmann 2020]

Conclusion

- Back to the basics: MLP are (weird) functions that can overfit on a single data.
- Link to Physically Informed Neural Networks (PINNs).
- Positional encoding is very important?
- Can be used for other tasks: see next course.
- For novel view synthesis, the successor of Nerf is Gaussian Splatting: see next course.