Implicit neural representations

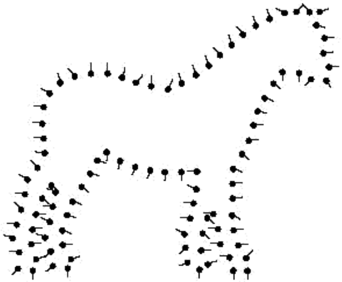
Julie Digne

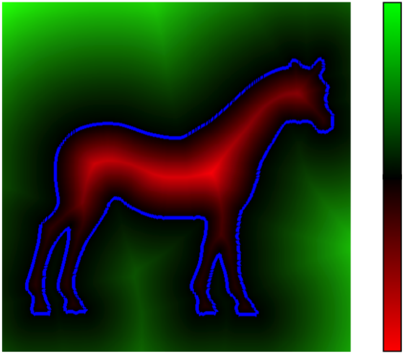
Master ID3D
LIRIS - CNRS
Équipe Origami

28/11/2024

Outline

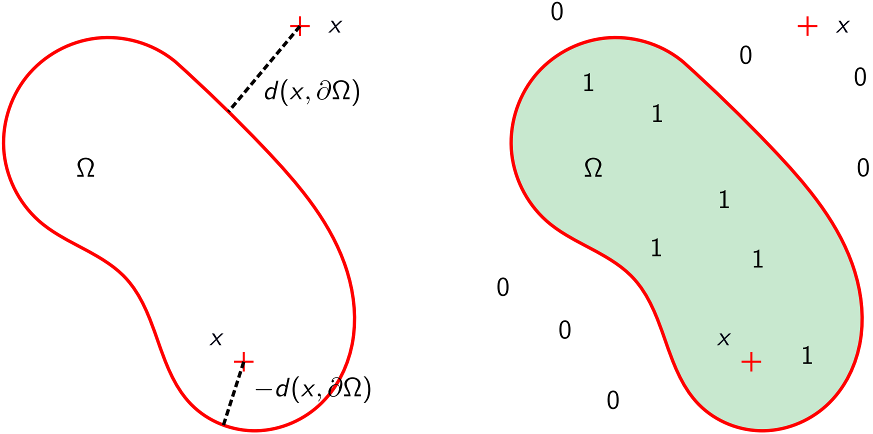
Implicit surface reconstruction - Principle





- 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]

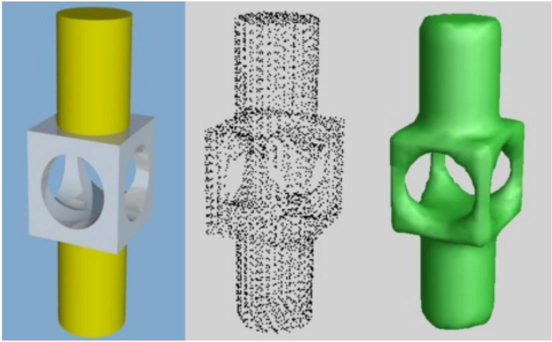
- Input: a set of 3D points
- *Idea:* for points on the surface the signed distance transform has a gradient equal to the normal

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

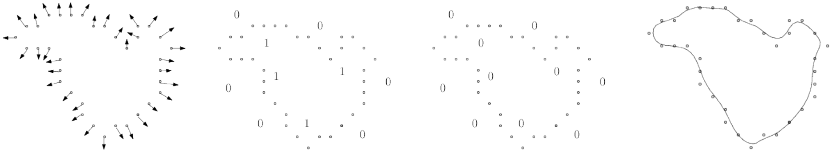
- 0 is a regular value for F and thus the isolevel extraction will give a manifold
- Compute an associated tangent plane (o_i, n_i) for each point p_i of the point set
- Orientation of the tangent planes as explained before.

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: $\hat{f}(p) = n_i \cdot (p - o_i)$



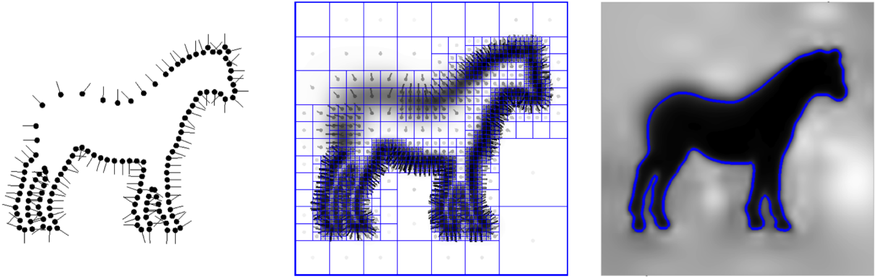
Poisson Surface Reconstruction [Kazhdan et al. 2006]



- 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



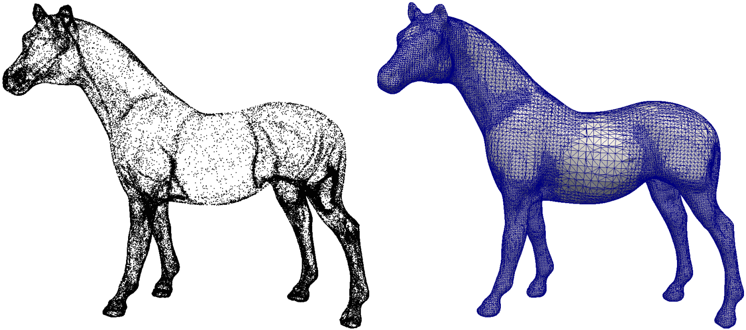
Poisson Surface Reconstruction [Kazhdan et al. 2006]

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

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

- Eq to:

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



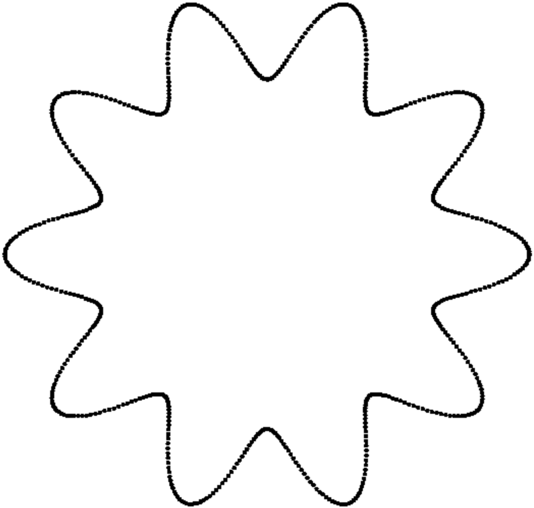
From the signed distance function to the mesh

- 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

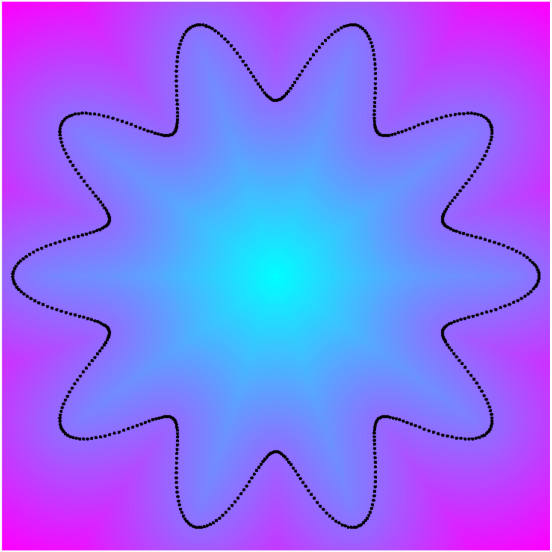
Approximation

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

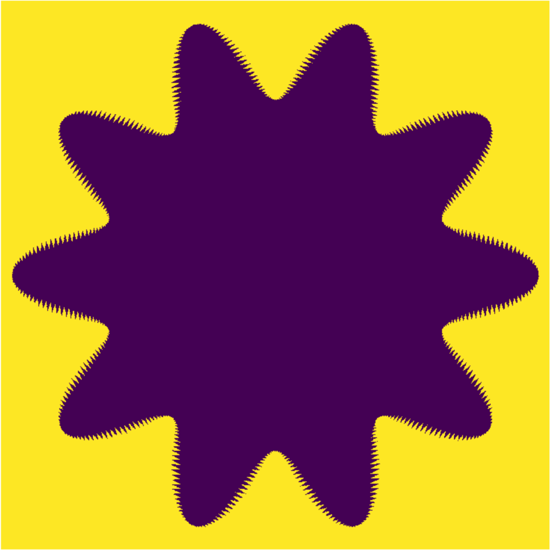
Example in 2D



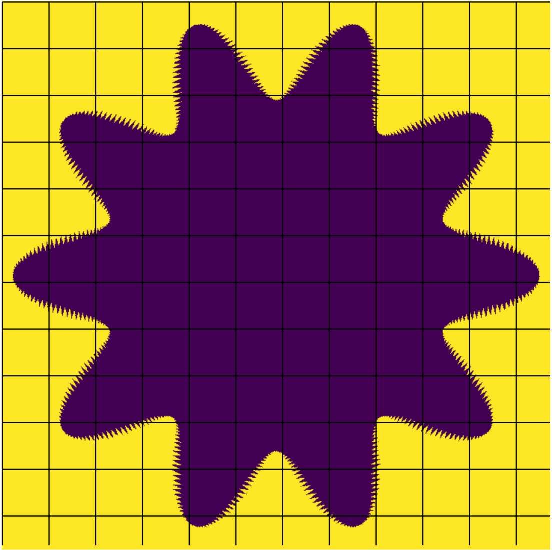
Example in 2D



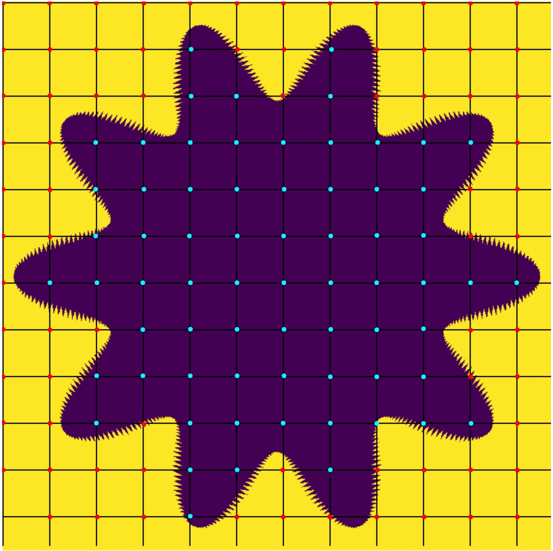
Example in 2D



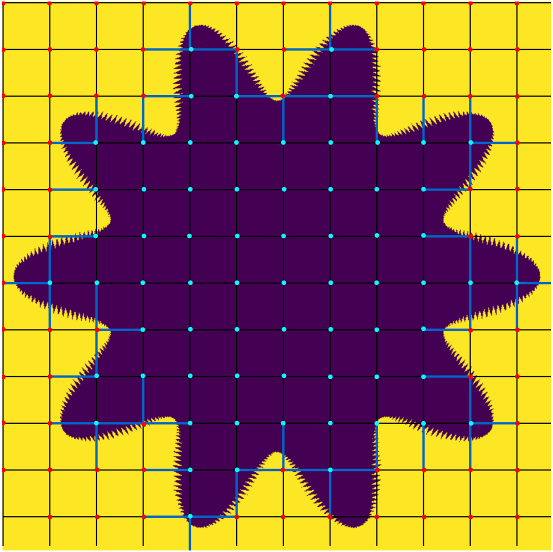
Example in 2D



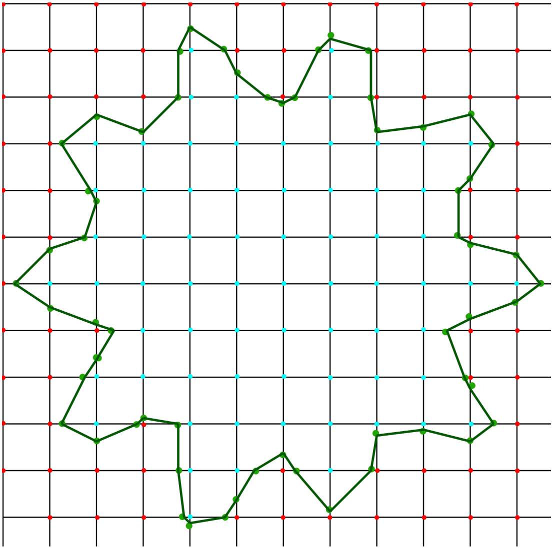
Example in 2D



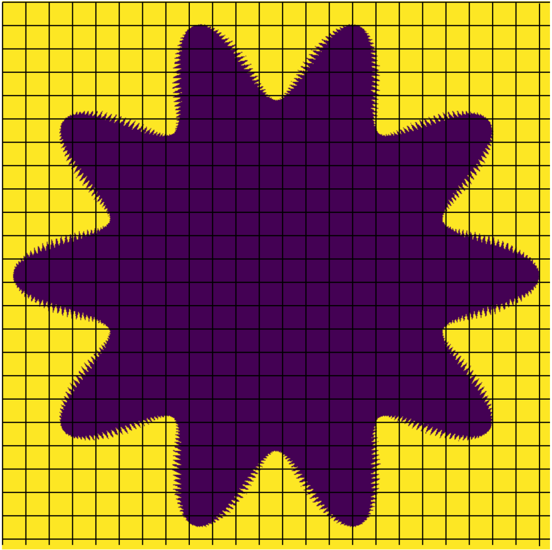
Example in 2D



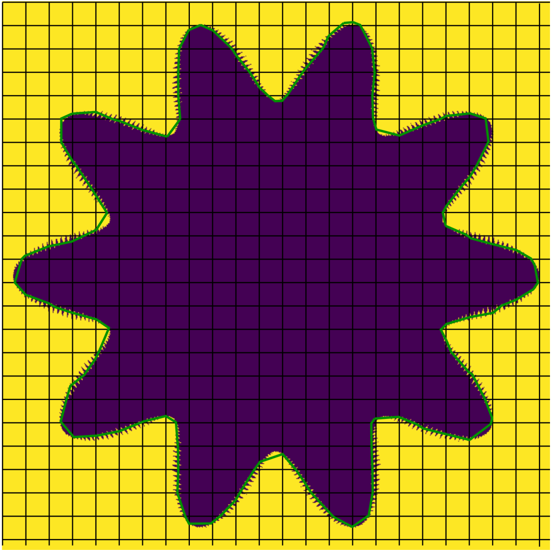
Example in 2D



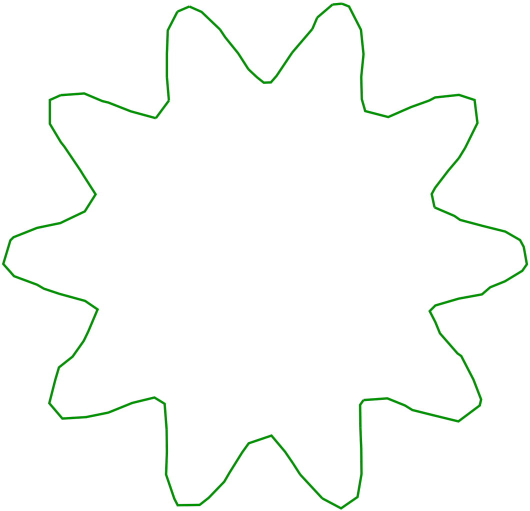
Example in 2D



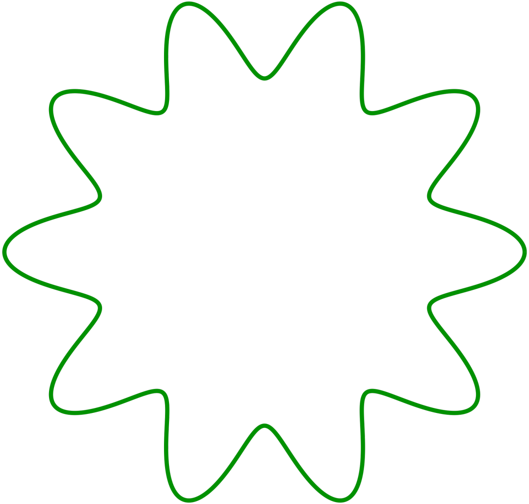
Example in 2D



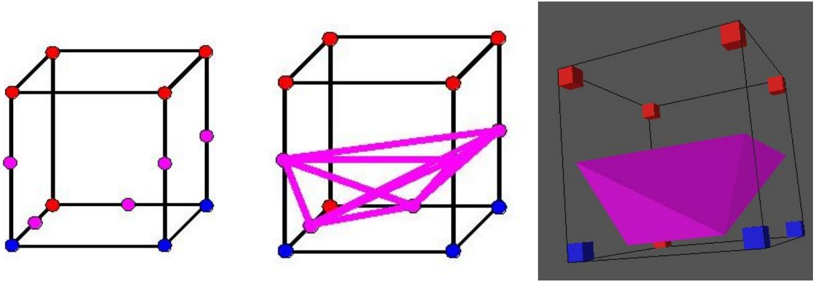
Example in 2D



Example in 2D



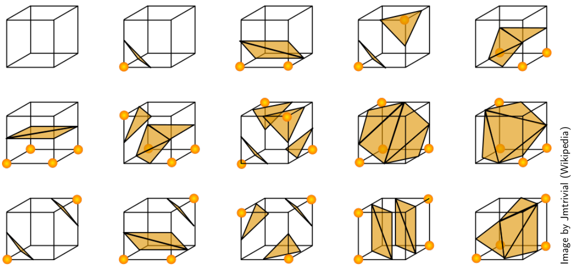
From Marching Squares to Marching Cubes



Images by Ben Anderson

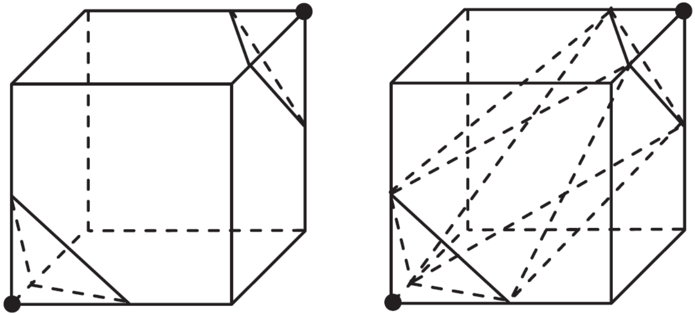
Drawing lines between intersection points is ambiguous and does not give a surface patch.

Look-up tables



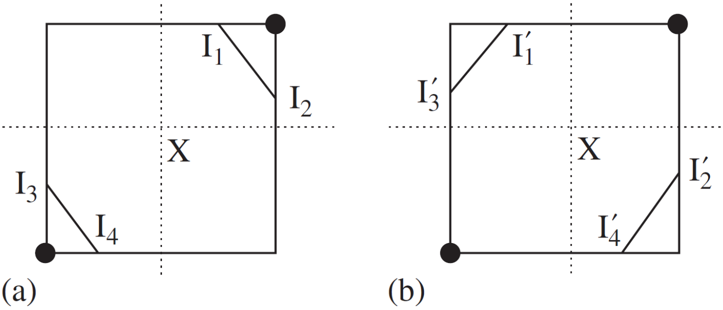
- There are $2^8 = 256$ possible cases for cube corner values.
- By symmetry + rotation arguments it reduces to 15 cases.
- Build a look-up table giving the grid cell triangulation based on the corner values case.

Ambiguous cases



(a) (b)

Ambiguous cases

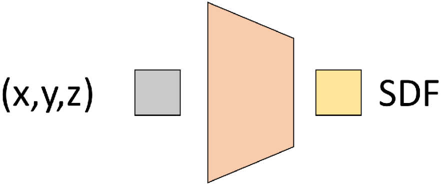


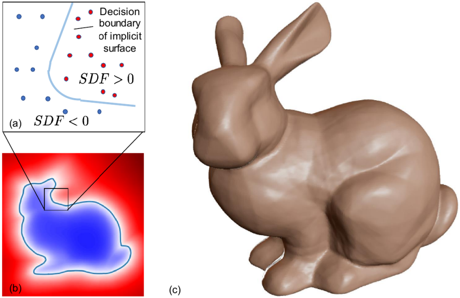
- Refine the grid to remove ambiguation
- Switch to marching tetrahedra algorithm

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 $x_i \in X$.





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

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

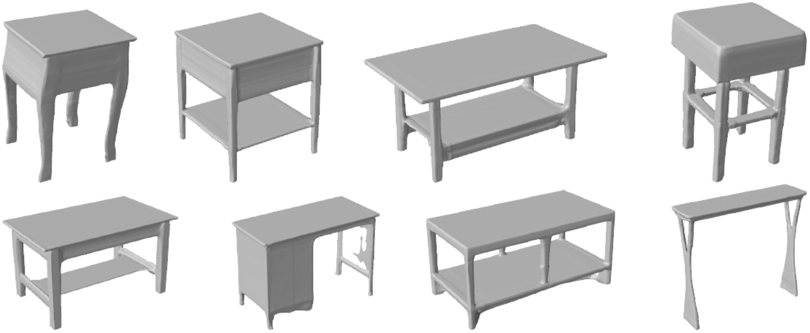
with $clamp(h, \delta) = \min(\delta, \max(-\delta, h))$.

- 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

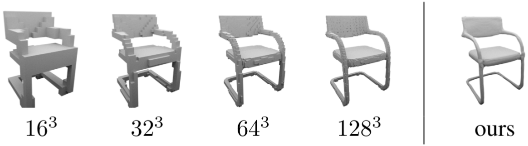
- Input data: a set of points y_i in $\mathbb{R}^3$ and their positions relatively to the surface $o_i = 0$ or 1.

Loss function

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

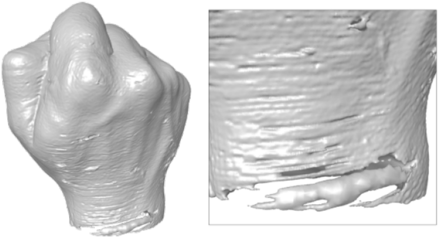
- This is the single shape loss. Occupancy networks are mostly used in the context of latent shape spaces, see next course for more details!

Results



[Mescheder 2019]

Learning an unsigned distance

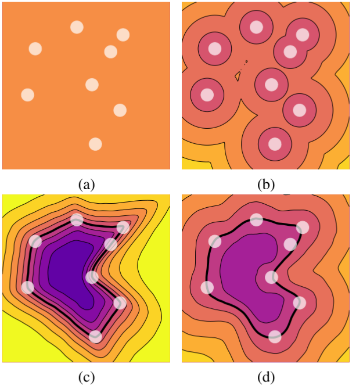


Mullen et al. 2010

- Normal direction is *easy* to compute
- *Consistent* normal orientation is hard to compute
- Bad normal orientations create artifacts for the SDF estimation

Sign agnostic distance function (Aatzmon 2020]

- Unoriented points (not even using normal direction)
- Signed distance function or surface indicator function



Losses

Loss function

$$loss(\theta) = \mathrm{E}_{x \in \mathcal{D}_X}[\tau(|u_\theta(x)|, h_X(x))]$$

- $\mathcal{D}_X$ is a distribution of points
- τ is a similarity function.
- h_X is an unsigned distance to the shape.

Losses

Loss function

$$loss(\theta) = \mathrm{E}_{x \in \mathcal{D}_X}[\tau(|u_\theta(x)|, h_X(x))]$$

- $\mathcal{D}_X$ is a distribution of points
- τ is a similarity function.
- h_X is an unsigned distance to the shape.

Conditions on τ

$\tau : \mathbb{R} \times \mathbb{R}^+ \rightarrow \mathbb{R}$ is such that:

- Sign agnostic: $\tau(-a, b) = \tau(a, b) \forall (a, b) \in \mathbb{R} \times \mathbb{R}^+$
- Monotonicity: $\frac{\partial \tau}{\partial a} = \rho(a - b) \forall (a, b) \in \mathbb{R}^+ \times \mathbb{R}$

Useful for the theorems guaranteeing reconstruction properties.

Choice of h_X and similarity τ

- ℓ^2 distance:

$$h_2(y) = \min_{x \in X} \|y - x\|_2$$

- ℓ^0 distance:

$$h_0(y) = \begin{cases} 1 \text{ if } y \in X \\ 0 \text{ otherwise.} \end{cases}$$

- $\tau(a, b) = ||a| - b|^l$

Choice of h_X and similarity τ

- ℓ^2 distance: Signed distance function

$$h_2(y) = \min_{x \in X} \|y - x\|_2$$

- ℓ^0 distance: indicator of the surface

$$h_0(y) = \begin{cases} 1 \text{ if } y \in X \\ 0 \text{ otherwise.} \end{cases}$$

- $\tau(a, b) = ||a| - b|^l$

Choice of point distribution $\mathcal{D}_X$

- Data points $X = x$, not enough to learn the whole field
- For the ℓ^2 distance:

$$\mathcal{D}_X = \sum_i \mathcal{N}(x_i, \sigma^2 I)$$

$$\mathcal{L}_2(\theta) = \mathrm{E}_{y \sim \mathcal{D}_X}[|u_\theta(y)| - h_2(y)]$$

- For the ℓ^0 distance:

$$\mathcal{D}_X = \sum_i \mathcal{N}(x_i, \sigma^2 I) + \sum_i \delta_{x_i}$$

$$\mathcal{L}_0(\theta) = \mathrm{E}_{y \sim \sum_i \mathcal{N}(x_i, \sigma^2 I)}[|u_\theta(y)| - 1] + \mathrm{E}_{y \sim \sum_i \delta_{x_i}}[|u_\theta(y)|]$$

Neural Architecture

MLP Architecture

$$u_\theta(x) = \varphi(w^T f_l \circ f_{l-1} \circ \cdots \circ f_1(x) + b) + c$$

with:

$$f_i(x) = \nu(W_i x + b_i)$$

$b_i \in \mathbb{R}^{d_i^{out}}$, $W_i \in \mathbb{R}^{d_i^{out} \times d_i^{in}}$, $w \in \mathbb{R}^{d_l^{out}}$ and $c \in \mathbb{R}$. ν are ReLU activation functions and φ a strong nonlinearity activation function.
+ Skip connection to the middle layer.

Neural Architecture

MLP Architecture

$$u_\theta(x) = \varphi(w^T f_l \circ f_{l-1} \circ \cdots \circ f_1(x) + b) + c$$

with:

$$f_i(x) = \nu(W_i x + b_i)$$

$b_i \in \mathbb{R}^{d_i^{out}}$, $W_i \in \mathbb{R}^{d_i^{out} \times d_i^{in}}$, $w \in \mathbb{R}^{d_l^{out}}$ and $c \in \mathbb{R}$. ν are ReLU activation functions and φ a strong nonlinearity activation function.
+ Skip connection to the middle layer.

Strong activation

$\varphi : \mathbb{R} \leftarrow \mathbb{R}$ is called a strong non-linearity if it is differentiable almost everywhere, antisymmetric: $\varphi(a) = -\varphi(-a)$ and there exists $\beta \in \mathbb{R}^+$ so that $\frac{1}{\beta} \geq \varphi'(a) \geq \beta$ for all $a \in \mathbb{R}$ where it is defined.

Neural Architecture

MLP Architecture

$$u_\theta(x) = \varphi(w^T f_l \circ f_{l-1} \circ \cdots \circ f_1(x) + b) + c$$

with:

$$f_i(x) = \nu(W_i x + b_i)$$

$b_i \in \mathbb{R}^{d_i^{out}}$, $W_i \in \mathbb{R}^{d_i^{out} \times d_i^{in}}$, $w \in \mathbb{R}^{d_l^{out}}$ and $c \in \mathbb{R}$. ν are ReLU activation functions and φ a strong nonlinearity activation function.
+ Skip connection to the middle layer.

Strong activation

$\varphi : \mathbb{R} \leftarrow \mathbb{R}$ is called a strong non-linearity if it is differentiable almost everywhere, antisymmetric: $\varphi(a) = -\varphi(-a)$ and there exists $\beta \in \mathbb{R}^+$ so that $\frac{1}{\beta} \geq \varphi'(a) \geq \beta$ for all $a \in \mathbb{R}$ where it is defined.

- In practice we take $\varphi(a) = a$ or $\varphi(a) = \tanh(a) + \gamma a$ with $\gamma \geq 0$.

Initialization

- Why? Avoid some local minima.

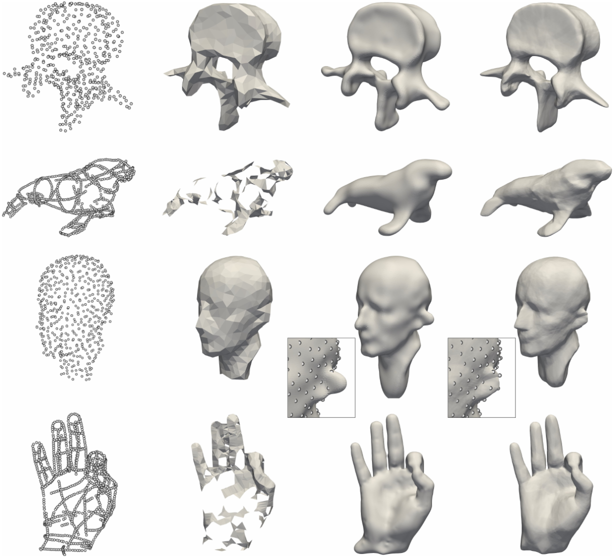
Theorem

Let u_θ be an MLP, Let $b_i = 0$, and W_i iid for a normal distribution $\mathcal{N}(0, \frac{\sqrt{2}}{\sqrt{d_i^{out}}})$ $1 \leq i \leq l$, $w = \frac{\sqrt{\pi}}{\sqrt{d_l^{out}}} 1$, $c = -r$ then: $u_\theta(x) = \phi(\|x\| - r)$.

Properties

- *Plane reproduction:* If data points lie on a hyperplane, this plane is a critical point of the loss.
- *Local plane reconstruction*: Can be applied locally for surfaces.

Results



Input point cloud, Ball Pivoting, Variational implicit reconstruction, SAL

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

$$\left\{\begin{array}{rl} \|\nabla u\| & = 1 \\ u_{|\partial\Omega} & = 0 \\ \nabla u_{|\partial\Omega]} & = \mathrm{n} \end{array}\right. \tag{1}$$

- Loss [Gropp 2020]

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

Periodic Activation Functions [Sitzmann 2021]

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

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

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

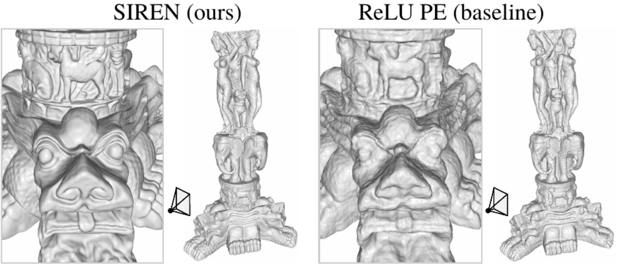


Figure 4: A comparison of SIREN used to fit a SDF from an oriented point clouse against the same fitting performed by an MLP using a ReLU PE (proposed in [35]).

From [Sitzmann 2020]

Periodic Activation Functions [Sitzmann 2021]

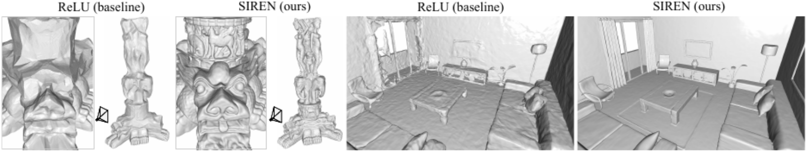


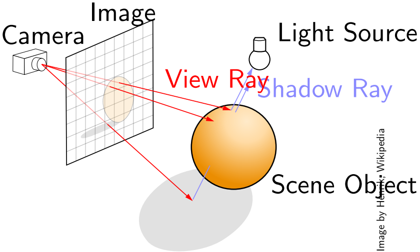
Figure 4: Shape representation. We fit signed distance functions parameterized by implicit neural representations directly on point clouds. Compared to ReLU implicit representations, our periodic activations significantly improve detail of objects (left) and complexity of entire scenes (right).

From [Sitzmann 2020]

Outline

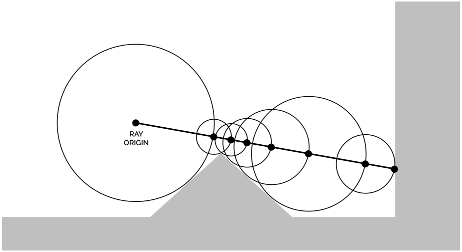
1. Implicit surface reconstruction - a short history
2. Neural single shape reconstruction
3. Geometric prior - Eikonal equation
4. Rendering Implicit surfaces
5. INR for Shape Analysis
6. Novel View Synthesis

Sphere tracing



- Requires to compute ray/surface intersection.
- Direct intersection with explicit representations (Meshes/Geometric primitives)

Sphere tracing [Hart 1996]



1. Input: a point x and direction v, a signed distance field u.
2. Initialize $t = 0$
3. While $t < D$
    1. $x_t = x + t\mathrm{v}$
    2. $d = u(x_t)$
    3. If $d < \varepsilon$ Return x_t
    4. Else Increment $t = t + d$

After intersection...

- Similar to ray tracing, rebounds can be computed
- Direct light only: color = scalar product of normal at intersection point and light direction.

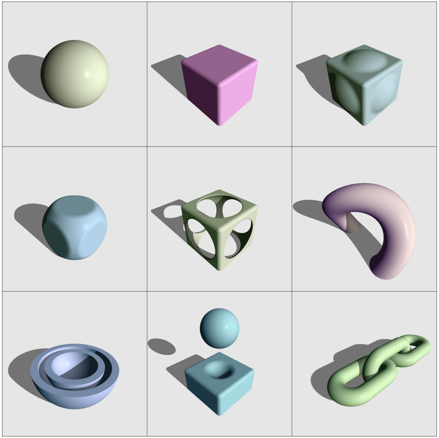
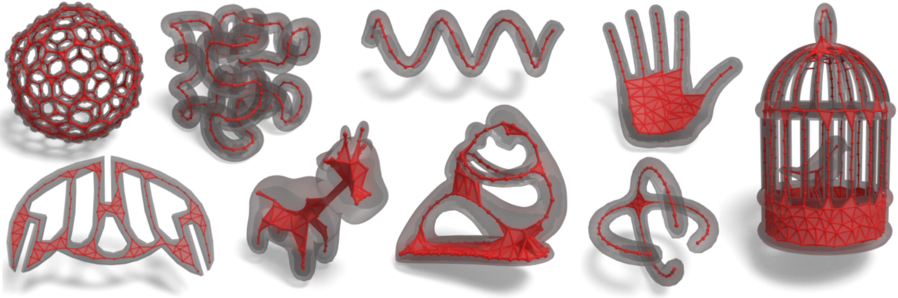


Image by Hiroki Sakuma

Outline

Regularizing INR away from the surface

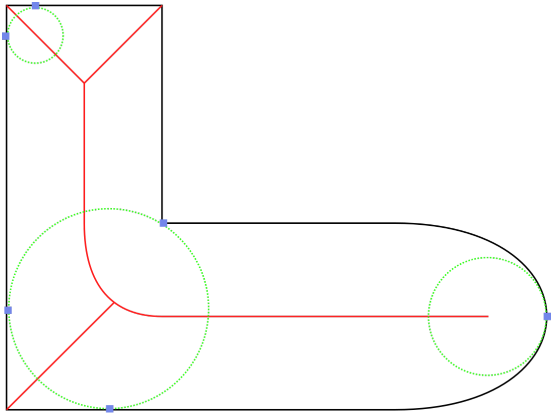


[Clémot, Digne 2023]

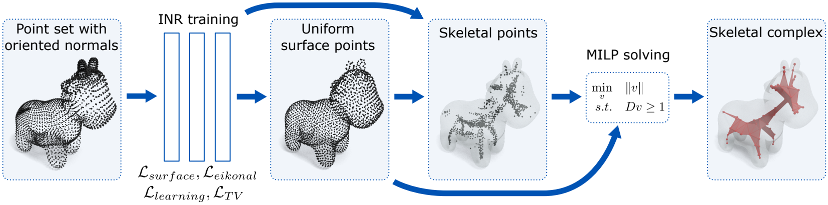
Medial Axis

Definition

A point p belongs to the medial axis of a compact shape if it has at least two distinct nearest neighbors on the shape surface.



Overview

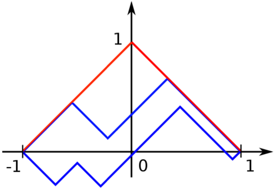

Point set with oriented normals
INR training
$\mathcal{L}_{surface}, \mathcal{L}_{eikonal}$
$\mathcal{L}_{learning}, \mathcal{L}_{TV}$
Uniform surface points
Skeletal points
MILP solving
$\min_v \|v\|$
$s.t.\ Dv \geq 1$
Skeletal complex

Eikonal Equation

- Infinite number of solutions
- Viscosity solution theory: allows to select the right solution
- Use smooth eikonal equation (not practical [Lipman 2019])

$$\|\nabla u\| - \varepsilon \Delta u = 1$$

- Consequence: blobs appear



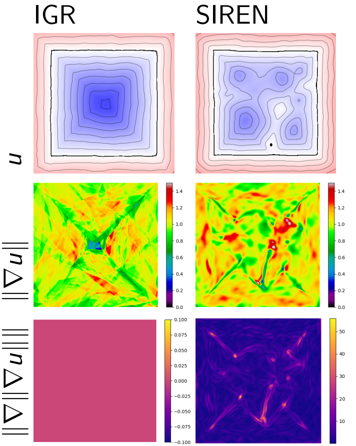
[Camilli 2014]

Infinite nber of solutions

Not an issue close to the surface – but far away?

Which neural network?

- MLP (6 layers, 128-256 neurons/layer) with ReLU activation functions
- ReLU yields a function in $W^{1,p}$ [Lipman 2019]
- But: not always easy to train
- Sitzman (2021) replaces ReLU with sine activation function: smooth function



TV regularization - some theory

- Look for a smooth surrogate for the signed distance function
- Medial axis: zeros of the gradient
- The TV term favors that u has no second order differential content along the gradient lines

Since $\nabla u = (u_x, u_y, u_z)$, it follows:

$$
\begin{aligned}
\nabla \|\nabla u\| &= \nabla \sqrt{u_x^2 + u_y^2 + u_z^2} \\
&= \frac{1}{2\|\nabla u\|} \begin{pmatrix} 2u_x u_{xx} + 2u_y u_{xy} + 2u_z u_{xz} \\ 2u_x u_{xy} + 2u_y u_{yy} + 2u_z u_{yz} \\ 2u_x u_{zx} + 2u_y u_{zy} + 2u_z u_{zz} \end{pmatrix} \\
&= H_u \frac{\nabla u}{\|\nabla u\|}
\end{aligned}
$$

Total loss

- Eikonal loss:

$$\mathcal{L}_{eikonal} = \int_{\mathbb{R}^3} (1 - \|\nabla u(p)\|)^2 \, dp \tag{2}$$

- Surface loss:

$$\mathcal{L}_{\text{surface}} = \int_{\partial\Omega} u(p)^2 dp + \int_{\partial\Omega} 1 - \frac{\mathrm{n}(p) \cdot \nabla u(p)}{\|\mathrm{n}(p)\| \, \|\nabla u(p)\|} dp \tag{3}$$

- Learning point loss

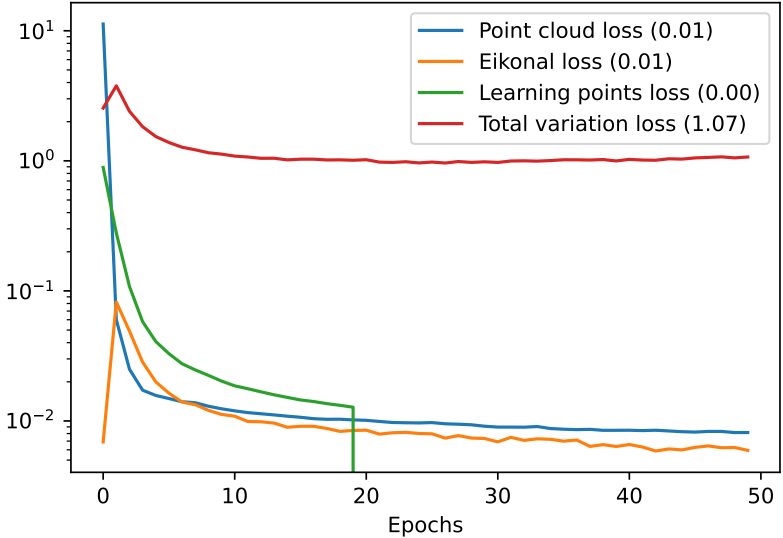
$$\mathcal{L}_{\text{learning}} = \sum_{p \in \mathcal{P}} (u(p) - d(p))^2 + \sum_{p \in \mathcal{P}} 1 - \frac{\nabla u(p) \cdot \nabla d(p)}{\|\nabla u(p)\| \, \|\nabla d(p)\|} \tag{4}$$

- \+ TV loss

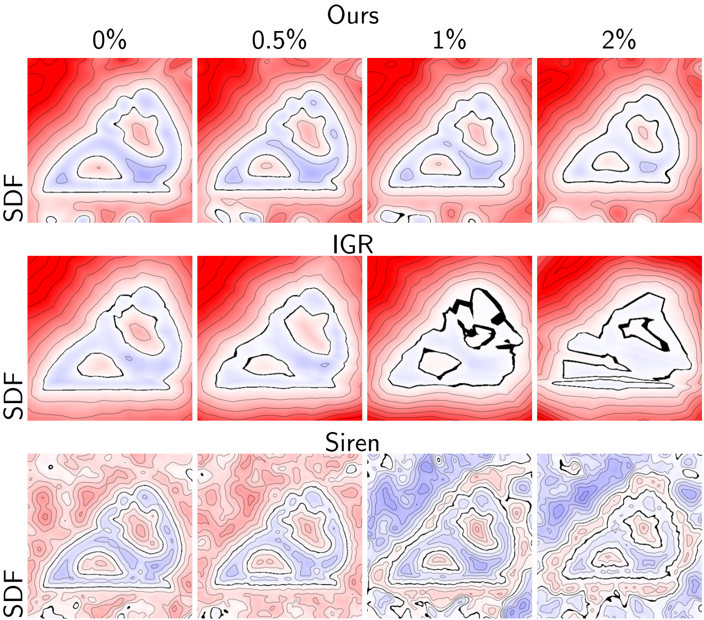
Loss

$$\mathcal{L} = \lambda_e \mathcal{L}_{eikonal} + \lambda_s \mathcal{L}_{surface} + \lambda_l \mathcal{L}_{learning} + \lambda_{TV} \mathcal{L}_{TV} \tag{5}$$

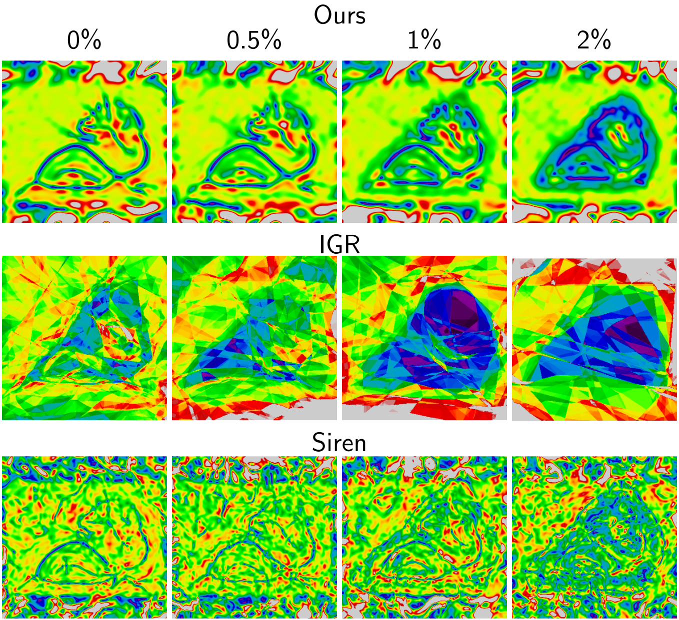
Convergence



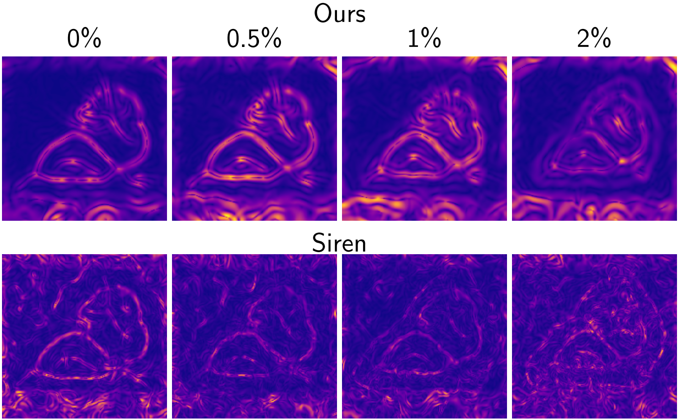
Resulting Fields



$\|\nabla u\|$



$\nabla \|\nabla u\|$



then...

- GPU skeleton tracing to extract points on the skeleton
- Select a subset based on the Coverage Axis method [Dou 2022]
  - N points x_i, M skeletal points s_i with distance r_i to the surface.
  - Coverage matrix: D $(N \times M)$

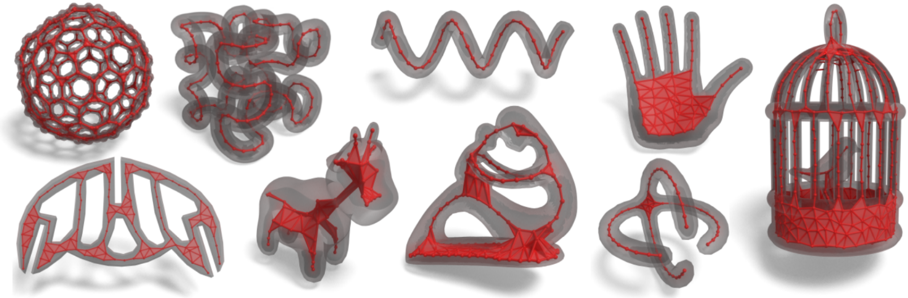
$$D_{ij} = 1 \text{ if } \|p_i - s_j\| - r_j \leq \delta \text{ and } 0 \text{ otherwise}$$

  - Mixed Integer Linear Problem:

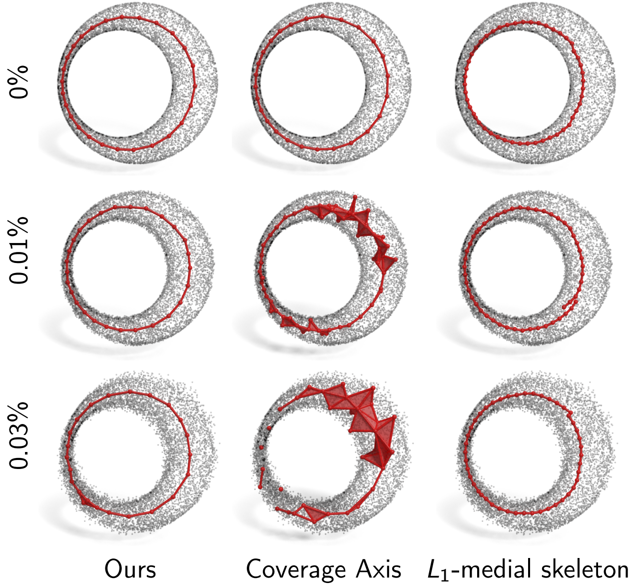
$$\begin{array}{rl} \min & \|v\|_2 \\ \text{s.t.} & Dv \succeq 1 \end{array} \tag{6}$$

- Link the selected points by computing the regular triangulation of weighted skeletal points and surface points + keep simplices between skeletal points

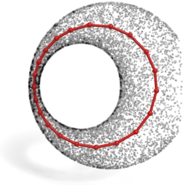
Results



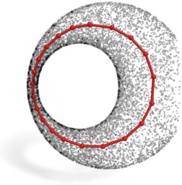
Results



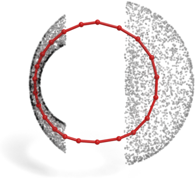
Results



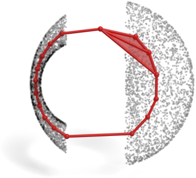
Ours



Coverage Axis

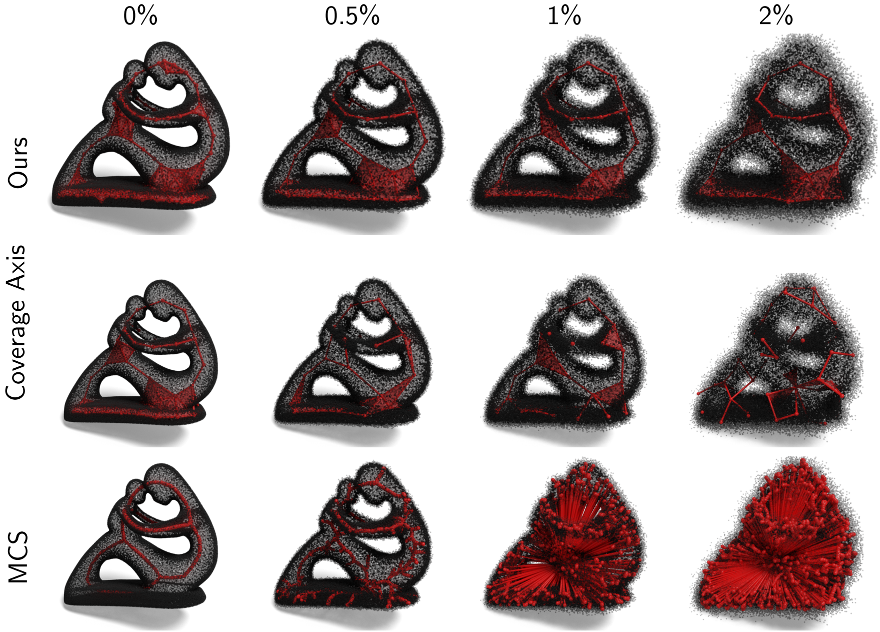


Ours

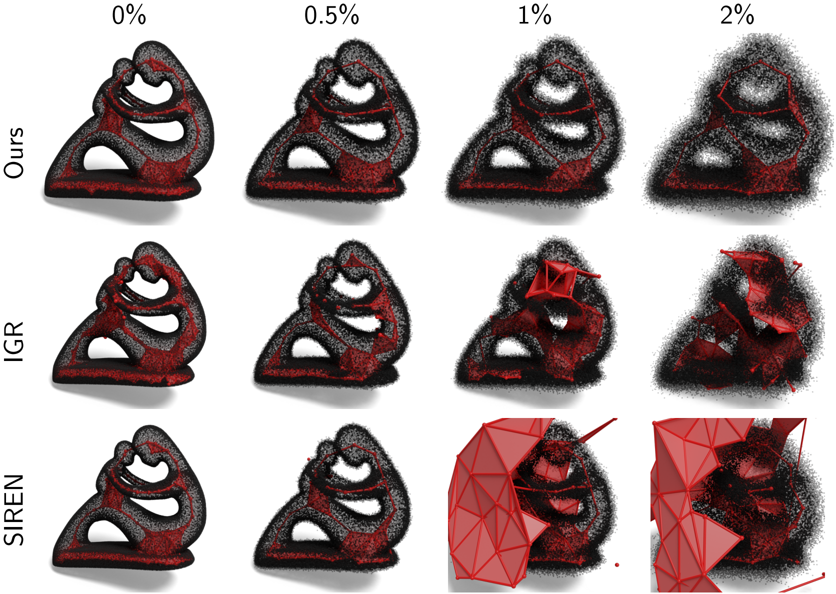


Coverage Axis

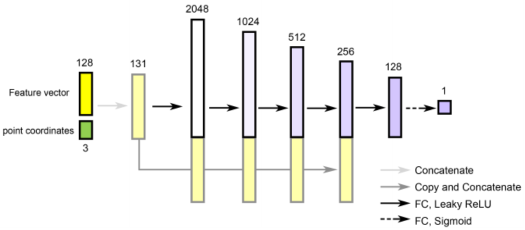
With noise



With noise



Learning Occupancy functions [Chen 2019, Mescheder 2020]



[Chen 2019]

- Use an encoder (e.g. PointNet [Qi 2017]) to get the shape latent description α.
- Train a neural network to compute the occupancy network of a shape given (x, y, z, α).

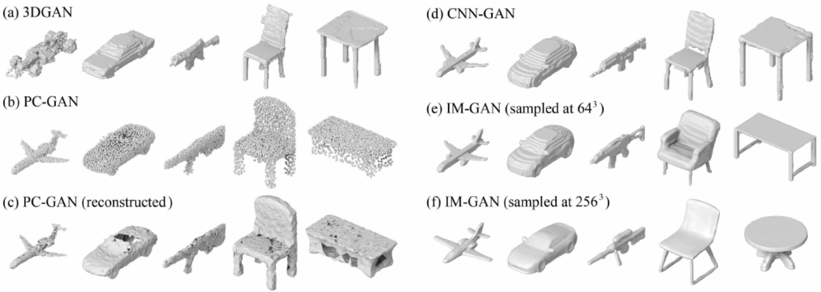
Data and Losses

- A set of N shapes S_i with points y_{ik} for which the occupancy is known.
- Training loss:

$$\frac{1}{|\mathcal{B}|} \sum_{i=1}^{N} \sum_{k=1}^{K} \mathcal{L}(u_\theta(y_{ik}, \alpha_i), o_{ik})$$

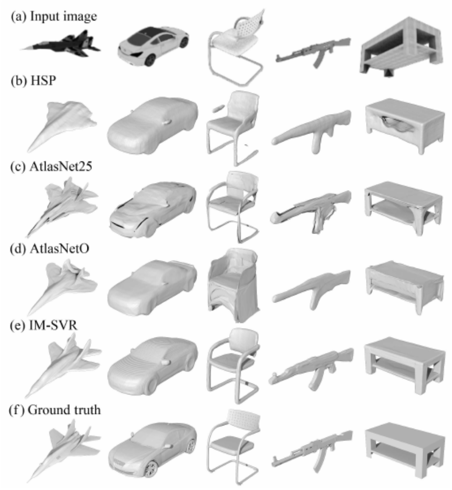
- $\mathcal{L}(u_\theta(y_{ik}, \alpha_i), o_{ik}) = |u_\theta(y_{ik}, \alpha_i) - o_{ik}|^2$
- Chen et al. [2019] adds a sampling density weight
- Mescheder et al. [2020] adds a KL divergence between a latent description prior and the encoder distribution.

Results and Comparisons

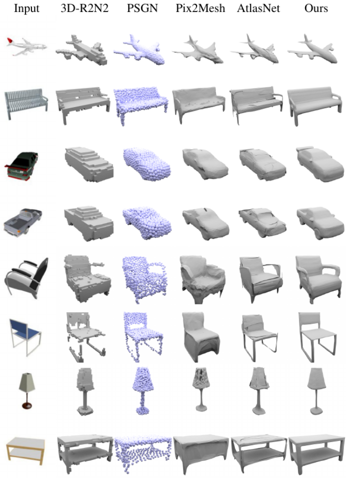


[Chen 2019]

Results - single view reconstruction

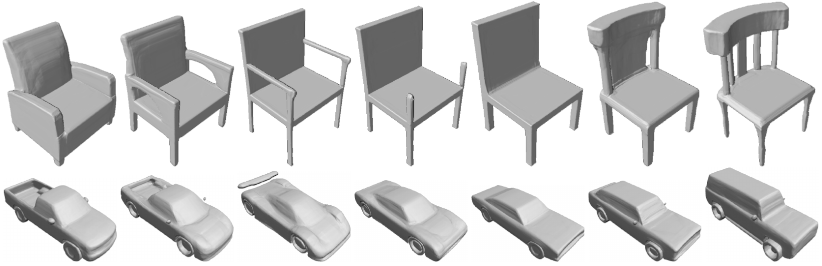


[Chen 2019]



[Mescheder 2020]

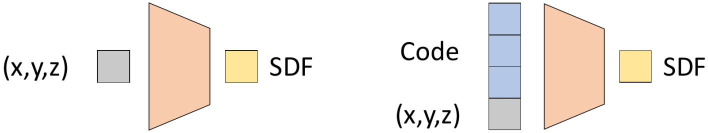
DeepSDF



[Park 2019]

- Represent an entire class of shapes in an implicit way

Training



(a) Single Shape DeepSDF **(b)** Coded Shape DeepSDF

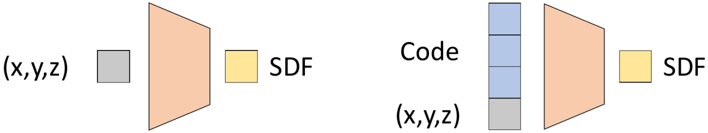
[Park 2019]

Single shape version

$$\mathcal{L}(f_\theta(x), s) = |clamp(f_\theta, \delta) - clamp(x, \delta)|$$

with $clamp(x, \delta) = \min(\delta, \max(-\delta, x))$, s isovalue.

Training



(a) Single Shape DeepSDF **(b)** Coded Shape DeepSDF

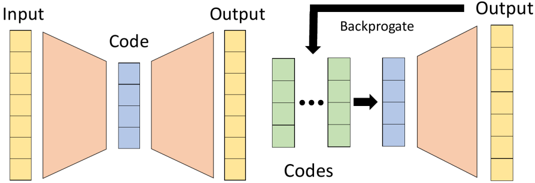
[Park 2019]

Latent shape version

$$f_\theta(z_i, x) = SDF^i(x)$$

Model several distance fields with a single network (factor in shape space)

Auto-decoder



(a) Auto-encoder **(b)** Auto-decoder

[Park 2019]

- Usually: train an auto-encoder + throw away the encoder.
- Here: avoid spending computational resources on encoder.
- Handle shapes of different number of samples.

Model for the auto-decoder

- Data: N shapes $X_i = \{(x_j, s_j), s_j = SDF^i(x_j)\}$.
- Latent code z_i, prior $p(z_i)$ = centered Gaussian with spherical covariance.

$$p_\theta(z_i|X_i) = p(z_i) \prod_j p_\theta(s_j|z_i, x_j)$$

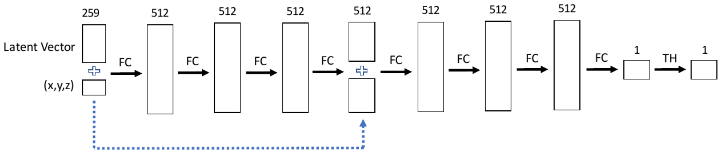
- Reformulation:

$$p(s_j|z_i, x_j) = \exp(-\mathcal{L}(f_\theta(z_i, x_j), s_j)) \text{ with } f_\theta \text{ an MLP.}$$

Training

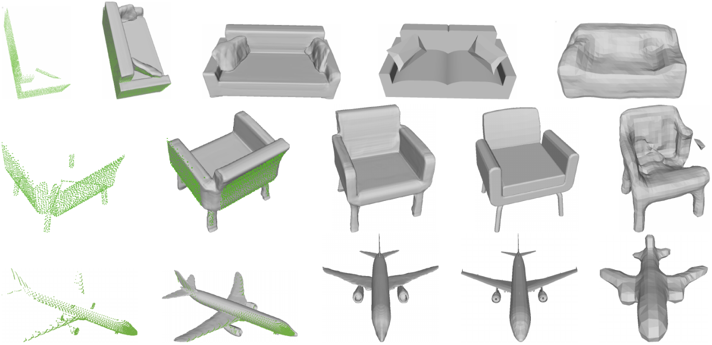
$$\mathrm{argmin}_{\theta, \{z_i\}_{i=\mathbf{1}}^N} \sum_{i=1}^N \sum_{j=1}^K \mathcal{L}(f_\theta(z_i, x_j), s_j) + \frac{1}{\sigma^2}\|z_i\|_2^2$$

Network architecture



[park 2019]

results

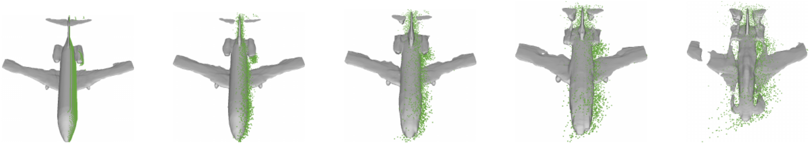


(a) Input Depth (b) Completion (ours) (c) Second View (ours) (d) Ground truth (e) 3D-EPN

[park 2019]

- solve for the shape code from partial shapes and reconstruct

results

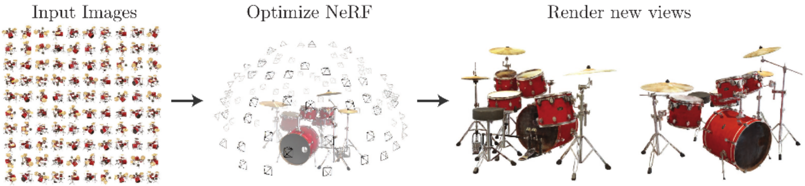


(a) No noise (b) $\alpha = 0.01$ (c) $\alpha = 0.02$ (d) $\alpha = 0.03$ (e) $\alpha = 0.05$

[park 2019]

Outline

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

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

with:

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

- 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

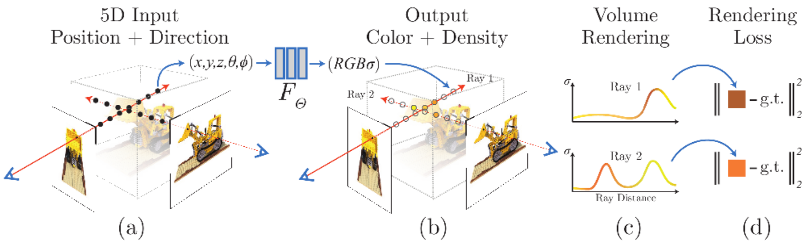
Discrete Version

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

with

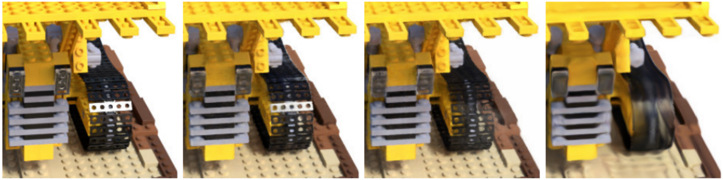
$$T_i = \sum_i \exp(-\sigma(t_i)\|t_{i+1} - t_i\|)$$

Training



[Mildenhall et al. 2020]

Positional Encoding



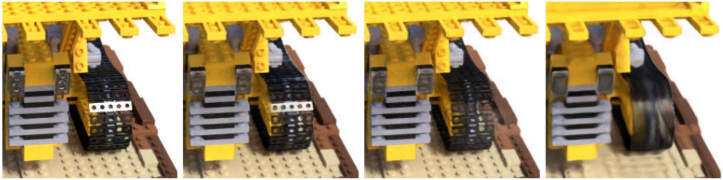
Ground Truth Complete Model No View Dependence No Positional Encoding

- 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

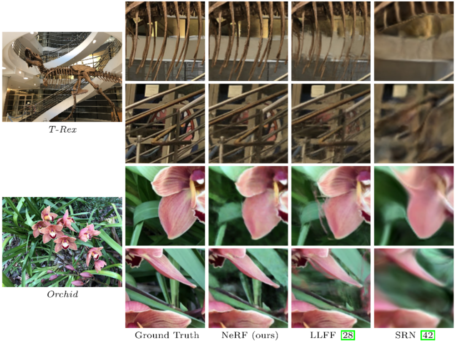
View-dependency



Ground Truth Complete Model No View Dependence No Positional Encoding

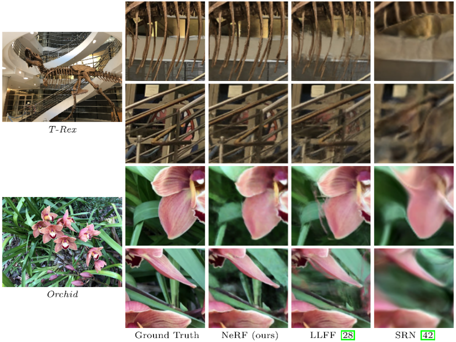
- View-dependent radiance is what allows to capture mirror reflections

Results



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

Results

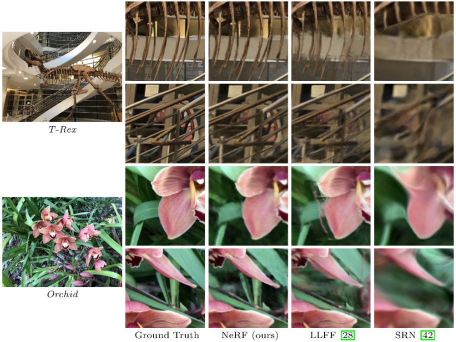


Video: `https://www.matthewtancik.com/nerf`

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

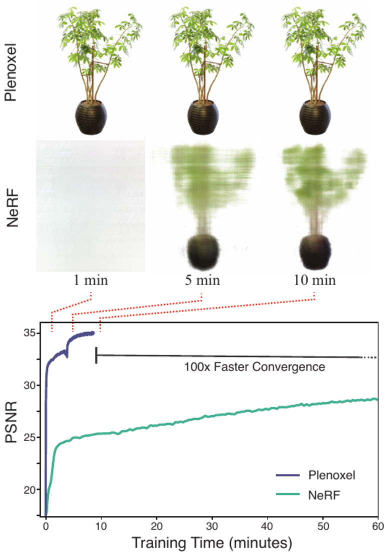


Video: `https://www.matthewtancik.com/nerf`

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])*

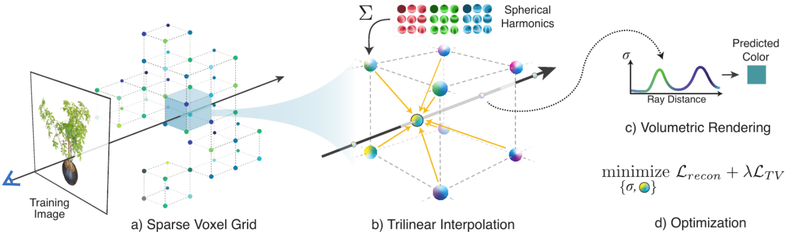
After Nerf... Plenoxels [Yu et al. 2021]



[Yu et al. 2021]

- No neural net
- (way) faster than nerf

Method



[Yu et al. 2021]

Spherical harmonics

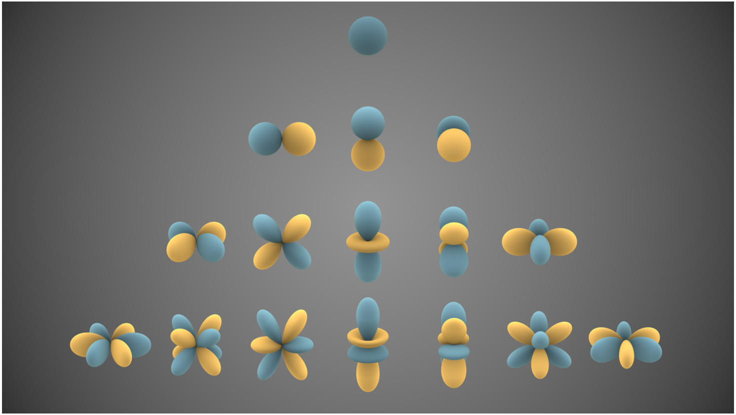


Image by Inigo Quilez (Wikipedia)

$$Y_l^m(\theta, \varphi) = e^{im\varphi} P_l^m(\cos(\theta))$$

- P_l^m Associated Legendre polynomial

$$P_l^m(x) = (-1)^m (1 - x^2)^{m/2} \sum_{k=m}^{l} \frac{k!}{(k-m)!} x^{k-m} \binom{l}{k} \binom{(l+k-1)/2}{l}$$

Color and spherical harmonics

- Spherical harmonics of degree 2 $\rightarrow$ 9 coefficients per color channel
- Color $C(r)$ = sum of the spherical harmonics evaluated in the ray direction
- Estimation on the vertices of a sparse grid and linear interpolation per grid cell.

Losses

- Optimization on SH coefficients and density minimizing the Loss:

$$\mathcal{L}_{recon} + \lambda \mathcal{L}_{TV}$$

- Reconstruction Loss:

$$\mathcal{L}_{recon} = \sum_{r \in \mathcal{R}} \|C(r) - \hat{C}(r)\|_2^2$$

- TV Loss:

$$\mathcal{L}_{TV} = \frac{1}{|\mathcal{V}|} \sum_{v \in \mathcal{V}, d \in \mathcal{D}} \sum_i \|\nabla_x SH_i\|_2 + \|\nabla_x \sigma\|_2$$

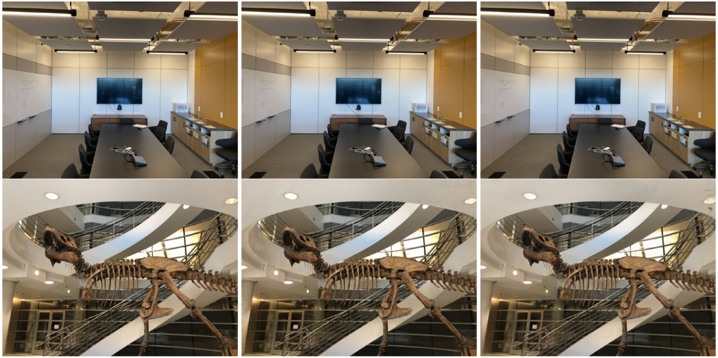
($\mathcal{V}$ and $\mathcal{R}$ stochastic samplings of the grid vertices and rays)

Results

Ground Truth NeRF++ [57] Plenoxels

[Yu et al. 2021]

Results



(d) Ground Truth (e) JAXNeRF (f) Plenoxels

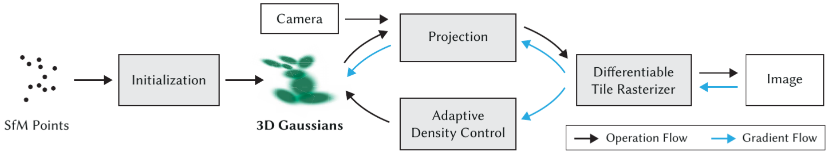
[Yu et al. 2021]

- Insight: What makes nerf work is not the neural net but *Differentiable* rendering.

Gaussian Splatting

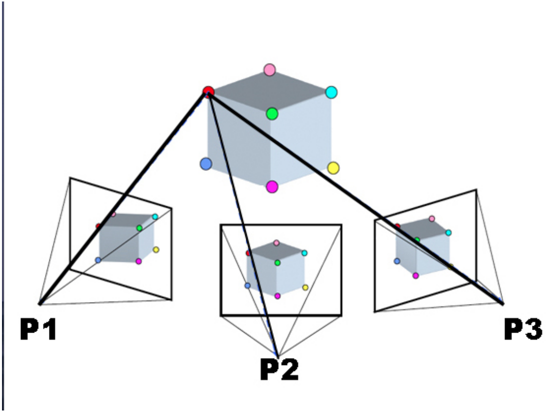
- Build on point set Splatting [Zwicker 2001]
- Each point is the center of a small 3D Gaussian on it,
- Each 3D Gaussian is represented by a quaternion and 3 scaling factors.
- Gaussian splat = gaussian parameters + opacity + Spherical harmonics

Overview



[Kerble 2023]

Structure from Motion (SfM)



[Maiteng - Wikipedia]

- Cameras calibrated by Structure from Motion [Snavely 2006]

Rendering a Gaussian splat scene

- Projective space Gaussian giving the color.

$$G(x) = \exp -x^T \Sigma^{-1} x \rightarrow G'(x) = \exp -x^T \Sigma'^{-1} x$$

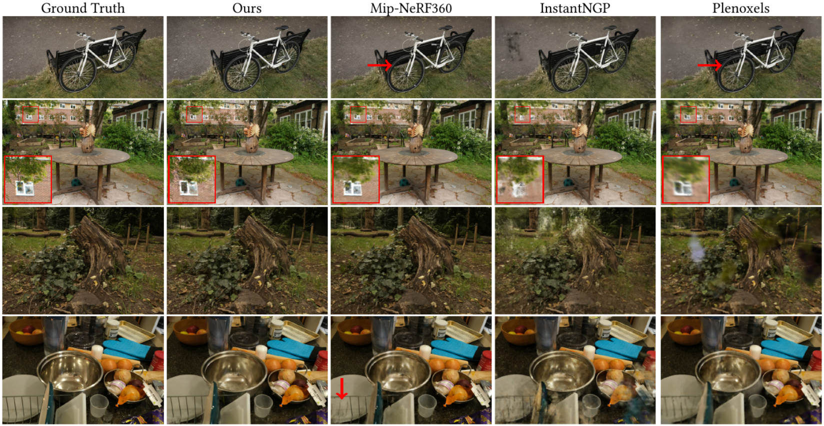
- Viewing direction W $\Sigma' = JW\Sigma W^T$
- J jacobian of the affine approx of the projective transformation:

$$J = \begin{pmatrix} f_x/z & 0 & -f_x t_x/z^2 \\ 0 & f_y/z & -f_y t_y/z^2 \\ 0 & 0 & 0 \end{pmatrix}$$

Rasterizer

- Split screen in tiles
- Cull 3d Gaussians against view frustrum
- Each tile = depth sorted Gaussians
- When saturation level is reached: stop

Creating or Destroying Geometry



[Kerble 2023]

Number of iterations

[Kerble 2023]

Conclusion

- Geometric data synthesis is hard
- Overview of *Single shape* implicit representation techniques
- Signed distance field or occupancy function or ??
- Nerf/Gaussian Splat: do we need to compute the geometry or only render?
- Multi-resolution, levels of details for neural implicits.

Temporary page!

LaTeX was unable to guess the total number of pages correctly. As the
unprocessed data that should have been added to the final page this e
has been added to receive it.
If you rerun the document (without altering it) this surplus page will g
because LaTeX now knows how many pages to expect for this documen