Novel view synthesis and Geometry Synthesis

Julie Digne

Master MVA

November 13th 2024

Outline

Lipschitz Networks

$$f : \mathcal{X} \to \mathcal{Y};\ \forall (x_1, x_2) \in \mathcal{X}^2,\ d_{\mathcal{Y}}(f(x_1), f(x_2)) \leq K d_{\mathcal{X}}(x_1, x_2)$$

Goal

Neural networks are learned functions f_θ from $\mathbb{R}^n$ to $\mathbb{R}^d$, can we design architectures which yield guaranteed K-Lipschitz functions?

Lipschitz Networks

$$f : \mathcal{X} \to \mathcal{Y}; \forall (x_1, x_2) \in \mathcal{X}^2,\ d_{\mathcal{Y}}(f(x_1), f(x_2)) \leq K d_{\mathcal{X}}(x_1, x_2)$$

Goal

Neural networks are learned functions f_θ from $\mathbb{R}^n$ to $\mathbb{R}^d$, can we design architectures which yield guaranteed K-Lipschitz functions?

With a small K:

- Better generalization

Lipschitz Networks

$$f : \mathcal{X} \to \mathcal{Y};\ \forall (x_1, x_2) \in \mathcal{X}^2,\ d_{\mathcal{Y}}(f(x_1), f(x_2)) \leq K d_{\mathcal{X}}(x_1, x_2)$$

Goal

Neural networks are learned functions f_θ from $\mathbb{R}^n$ to $\mathbb{R}^d$, can we design architectures which yield guaranteed K-Lipschitz functions?

With a small K:

- Better generalization
- Improved adversarial robustness

Lipschitz Networks

$$f : \mathcal{X} \to \mathcal{Y};\ \forall (x_1, x_2) \in \mathcal{X}^2,\ d_{\mathcal{Y}}(f(x_1), f(x_2)) \leq K d_{\mathcal{X}}(x_1, x_2)$$

Goal

Neural networks are learned functions f_θ from $\mathbb{R}^n$ to $\mathbb{R}^d$, can we design architectures which yield guaranteed K-Lipschitz functions?

With a small K:

- Better generalization
- Improved adversarial robustness
- Greater *interpretability*

Lipschitz Networks

$$f : \mathcal{X} \to \mathcal{Y};\ \forall (x_1, x_2) \in \mathcal{X}^2,\ d_{\mathcal{Y}}(f(x_1), f(x_2)) \leq K d_{\mathcal{X}}(x_1, x_2)$$

Goal

Neural networks are learned functions f_θ from $\mathbb{R}^n$ to $\mathbb{R}^d$, can we design architectures which yield guaranteed K-Lipschitz functions?

With a small K:

- Better generalization
- Improved adversarial robustness
- Greater *interpretability*
- Wasserstein distance computation (Peyré & Cuturi 2018).

Lipschitz Networks

$$f : \mathcal{X} \to \mathcal{Y};\ \forall (x_1, x_2) \in \mathcal{X}^2,\ d_{\mathcal{Y}}(f(x_1), f(x_2)) \leq K d_{\mathcal{X}}(x_1, x_2)$$

Goal

Neural networks are learned functions f_θ from $\mathbb{R}^n$ to $\mathbb{R}^d$, can we design architectures which yield guaranteed K-Lipschitz functions?

With a small K:

- Better generalization
- Improved adversarial robustness
- Greater *interpretability*
- Wasserstein distance computation (Peyré & Cuturi 2018).
- Issue: Lipschitz guarantee without sacrificing expressive power.

Notations

- x input, y output
- L layers
- l^{th} layer: dimension n_l, $\mathrm{W}_l \in \mathbb{R}^{n_l \times n_{l-1}}$
- $z_l = \mathrm{W}_l h_{l-1} + b_l,\ h_l = \phi(z_l)$
- $y = z_L$
- $C_L(X, \mathbb{R})$ space of all 1-Lipschitz functions mapping (X, d_X) to $(\mathbb{R}, L_p)$

A first result [Cem 2018]

Composition

Composition of two 1-Lipschitz functions is 1-Lipschitz.

A first result [Cem 2018]

Composition

Composition of two 1-Lipschitz functions is 1-Lipschitz.

Consequence

Compose 1-Lipschitz affine transform ($\|Wx\|_p \leq \|x\|_p, \forall x$) and 1 - Lipschitz activations.

A first result [Cem 2018]

Composition

Composition of two 1-Lipschitz functions is 1-Lipschitz.

Consequence

Compose 1-Lipschitz affine transform ($\|Wx\|_p \leq \|x\|_p, \forall x$) and 1 - Lipschitz activations.

- ReLU, tanh, maxout are 1-Lipschitz (if scaled appropriately)!

So... Are we done?

Theorem

Expressivity [Cem 2018] Consider a neural net $f : \mathbb{R}^n \to \mathbb{R}$, *built with* $\|W\|_2 \leq 1$ *and* 1*-Lipschitz elementwise monotonic activation functions. If* $\|\nabla f\|_2 = 1$ *almost everywhere then* f *is linear*

So... Are we done?

Theorem

Expressivity [Cem 2018] Consider a neural net $f : \mathbb{R}^n \to \mathbb{R}$, built with $\|W\|_2 \leq 1$ and 1-Lipschitz elementwise monotonic activation functions. If $\|\nabla f\|_2 = 1$ almost everywhere then f is linear

- ReLU, sigmoid, tanh?

Semi definite Programming Layer [Araujo et al. 2019]

SDPL

Residual layer with parameters $W \in \mathbb{R}^{k \times k}$, $q \in \mathbb{R}^k$, $b \in \mathbb{R}^k$

$$x \leftarrow x - 2WT^{-1}\sigma(W^T x + b)$$

with:

$$T = \sum_{j=1}^{K} |(W^T W)_{ij} \exp(q_i - q_j)|$$

and σ the ReLU activation function.

- W weight matrices are square (0-padding on the input)

Semi definite Programming Layer [Araujo et al. 2019]

SDPL

Residual layer with parameters $W \in \mathbb{R}^{k \times k}$, $q \in \mathbb{R}^k$, $b \in \mathbb{R}^k$

$$x \leftarrow x - 2WT^{-1}\sigma(W^T x + b)$$

with:

$$T = \sum_{j=1}^{K} |(W^T W)_{ij} \exp(q_i - q_j)|$$

and σ the ReLU activation function.

- W weight matrices are square (0-padding on the input)
- Output layer: affine layer

$$x \leftarrow \frac{w^T x}{\|w\|_2} + b$$

Wasserstein Distance estimation

Kantorovich duality

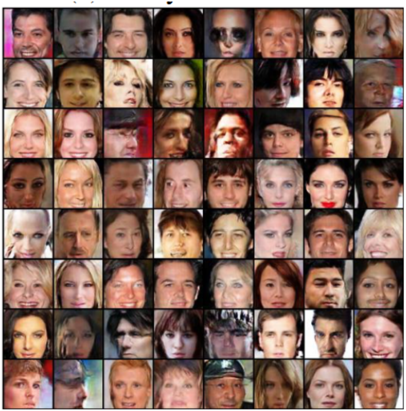
$$W(P_1, P_2) = \sup_{f \in C_L(X,\mathbb{R})} \mathbb{E}_{x\sim P1}[f(x)] - \mathbb{E}_{x\sim P_2}[f(x)]$$

- Wasserstein GAN: Lischitz network for the discriminator by weight clipping [Arjovsky et al. 2017]

$$\mathcal{L}_{WGAN}(G, D) = \mathbb{E}_{x\sim\mu_G}[D(x)] - \mathbb{E}_{x\sim\mu_{ref}}[D(x)]$$

- Kantorovich-Rubinstein dual formulation: for the optimal D, G tries to minimize $\mathcal{L}_{WGAN} \propto W_1(\mu_G, \mu_{ref})$

Wasserstein GAN: Leaky RELU vs MaxMin



[Cem et al. 2018]

Application to signed distance field [Coiffier 2024]

- Set of points x_i with known distances d_i
- Naive approach: combining 1-lipschitz network with a fitting loss:

Application to signed distance field [Coiffier 2024]

- Set of points x_i with known distances d_i
- Naive approach: combining 1-lipschitz network with a fitting loss: underestimates the signed distance field + not robust to imperfect data.

Application to signed distance field [Coiffier 2024]

- Set of points x_i with known distances d_i
- Naive approach: combining 1-lipschitz network with a fitting loss: underestimates the signed distance field + not robust to imperfect data.

Hinge-Kantorovich-Rubinstein loss [Serrurier 2021]

$$\mathcal{L}_{hKR} = \mathcal{L}_{KR} + \lambda \mathcal{L}^{m}_{hinge}$$

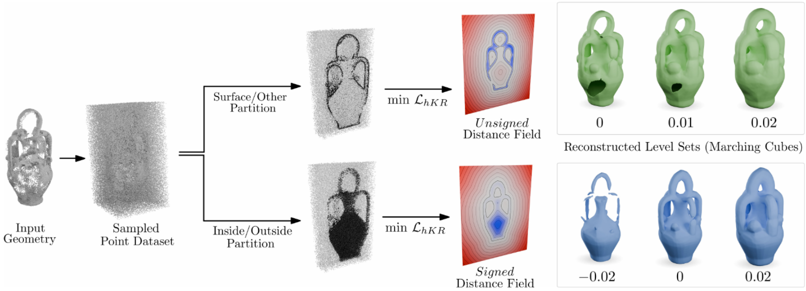
with:

$$\mathcal{L}_{KR} = \sum_i -sign(d_i) u_\theta(x_i)$$

$$\mathcal{L}^{m}_{hinge} = \sum_i \max(0, m - sign(d_i) u(x_i))$$

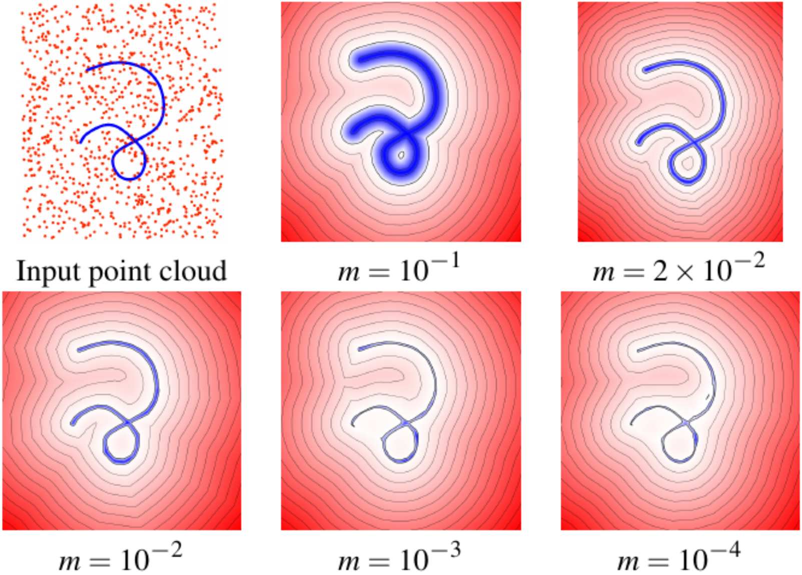
- Under mild assumptions, proof that this converges to an approximation of the signed distance field.

Application to Signed distance field estimation



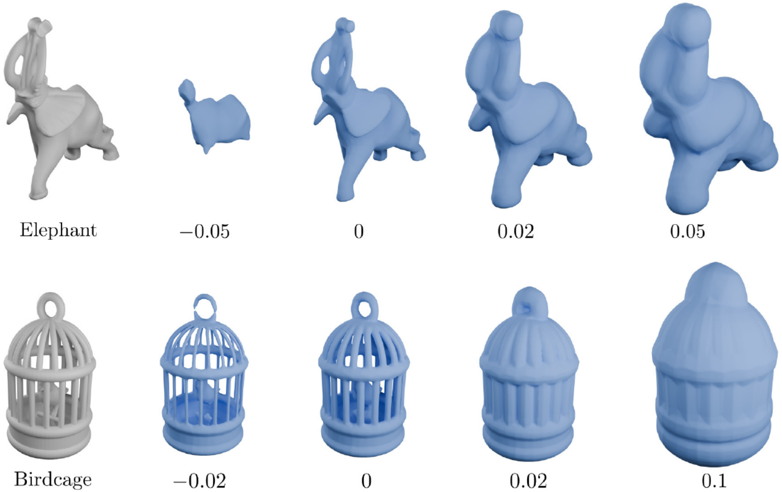
[Coiffier 2024]

Application to Signed distance field estimation



[Coiffier 2024]

Application to Signed distance field estimation



[Coiffier 2024]

Outline

Leverage Neural Implicits with Shape deformation

[Novello 2023]

- When a surface moves its neural representation evolves with it
- Can we link the evolution of the Neural Implicit with the vector field of the deformation?

Leverage Neural Implicits with Shape deformation

[Novello 2023]

- When a surface moves its neural representation evolves with it
- Can we link the evolution of the Neural Implicit with the vector field of the deformation?
- Very old topic (see e.g. [Osher 2000])

The Level Set Equation (LSE)

- $F(x,t)$ *temporal* neural implicit
- $V(x,t)$ Vector Field governing the deformation in ambient space.
- For all t: shape $= 0$ level set.

$$\frac{\partial F(x,t)}{\partial t} + \langle V(x,t), \nabla_x F(x,t) \rangle = 0$$

Mixing the LSE with Neural Networks

- As before model F by a neural network F_θ which takes as input x and t and outputs the signed distance function at x at time t.

Classical Losses

- Shape Data attachment loss

$$\sum_i \|F_\theta(x_i, 0)\|^2 + \|1 - \langle \mathrm{n}_i, \nabla F_\theta(x_i, 0)\rangle\|$$

- Ambient Data attachment loss

$$\sum_j \|F_\theta(y_j, 0) - gtsdf(y_j)\|^2$$

- Eikonal loss

$$\mathbb{E}_x[|1 - \|\nabla F_\theta(x, t)\||]$$

- We add the LSE loss depending on the application case.

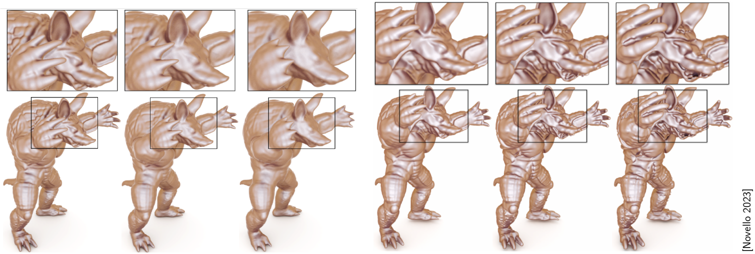
Known Vector Fields

[Novello 2023]

LSE Loss

$$\mathcal{L}_{LSE}(\theta) = \mathbb{E}_{(x,t)}[\| \frac{\partial F_\theta(x,t)}{\partial t} + \langle \nabla F_\theta(x,t), V \rangle \|^2]$$

Mean Curvature Motion



- Points evolve at speed $H(x, t)$ in direction $N(x, t)$ (normal to the level set)
- $H(p, t) = divN$
- $V(p, t) = -H(p, t)N(x, t)$

LSE Loss

$$\mathcal{L}_{LSE}(\theta) = \mathbb{E}_{(x,t)}[\| \frac{\partial F_\theta(x, t)}{\partial t} + \langle \nabla F_\theta(x, t), -H(p, t)\|\nabla_x F_\theta(x, t)\|\rangle \|^2]$$

Interpolation between shapes

- Vector field is not known.
- Two known distance fields f_0 and f_1
- Possible surrogate:

$$V(x,t) = -(f_1(x) - F_\theta(x,t))\frac{\nabla F_\theta(x,t)}{\|\nabla F_\theta(x,t)\|}$$

and $F(x,0) = f_0(x)$.

Interpolation between shapes (2)

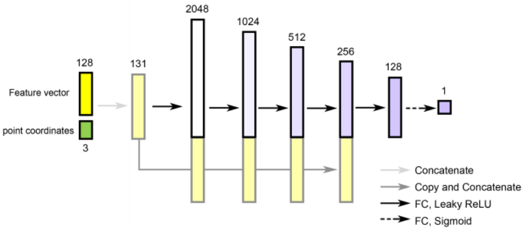
[Novello 2023]

LSE Loss

$$\mathcal{L}_{LSE}(\theta) = \mathbb{E}_{(x,t)}\left[\left\|\frac{\partial F_\theta(x,t)}{\partial t} - (f_1(x) - F_\theta(x,t))\|\nabla_x F_\theta(x,t)\|\right\|^2\right)$$

Outline

Learning Occupancy functions [Chen 2019, Mescheder 2020]



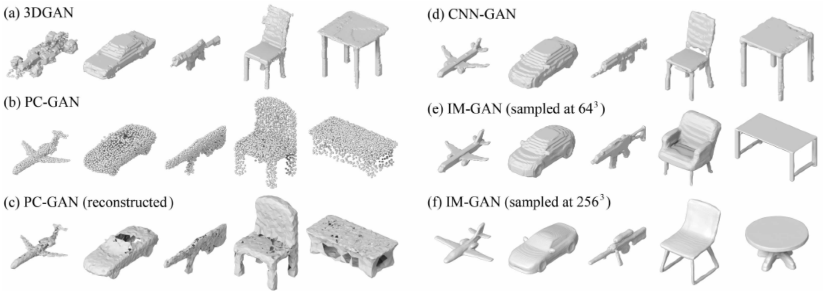
[Chen 2019]

- 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

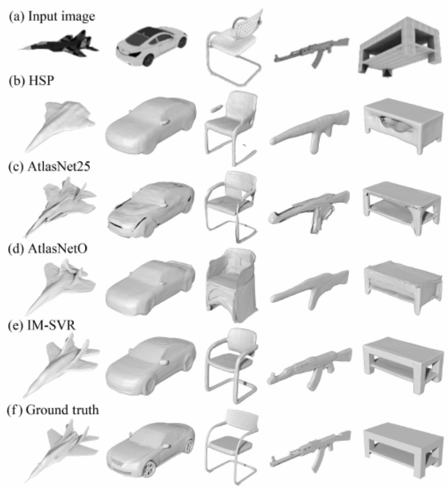
- A set of N shapes S_i with points y_{ik} for which the occupancy is known.
- Training loss:
$$\frac{1}{|\mathcal{B}|}\sum_{i=1}^{N}\sum_{k=1}^{K}\mathcal{L}(u_\theta(y_{ik},\alpha_i),o_{ik})$$
- $\mathcal{L}(u_\theta(y_{ik},\alpha_i),o_{ik}) = |u_\theta(y_{ik},\alpha_i) - o_{ik}|^2$
- Chen et al. [2019] 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

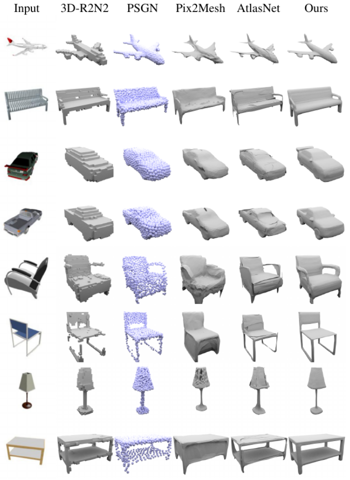


[Chen 2019]

Results - single view reconstruction

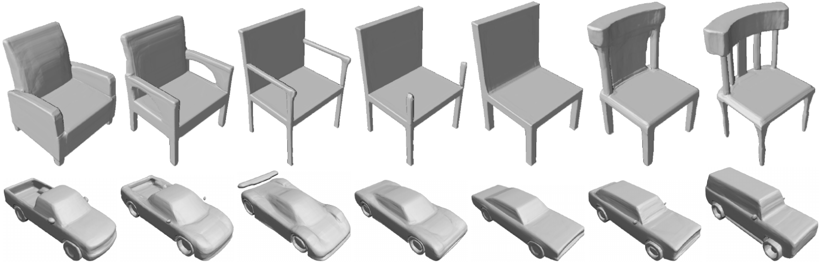


[Chen 2019]



[Mescheder 2020]

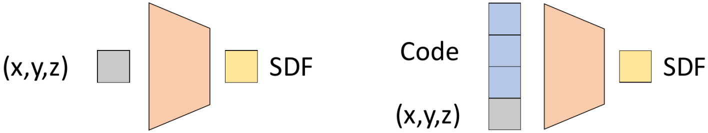
DeepSDF



[Park 2019]

- Represent an entire class of shapes in an implicit way

Training



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

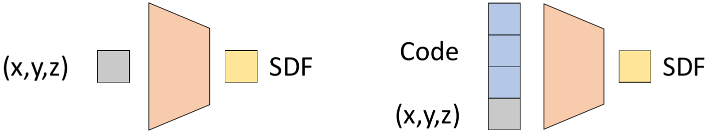
[Park 2019]

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

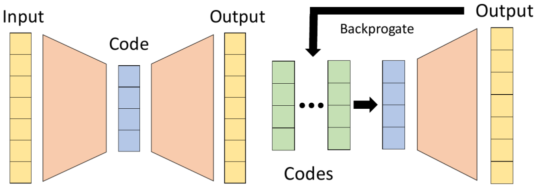
[Park 2019]

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.

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

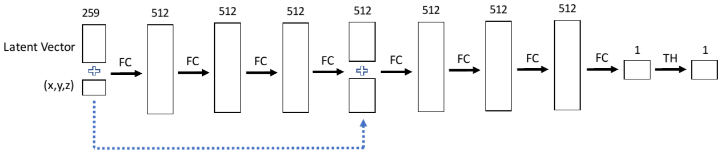
- Reformulation:

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

Training

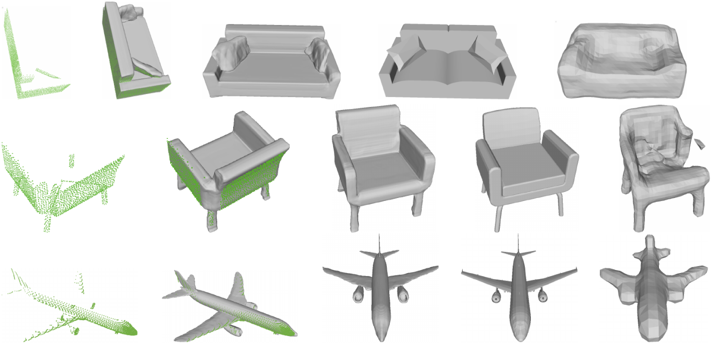
$$\text{argmin}_{\theta, \{z_i\}_{i=1}^N} \sum_{i=1}^N \sum_{j=1}^K \mathcal{L}(f_\theta(z_i, x_j), s_j) + \frac{1}{\sigma^2}\|z_i\|_2^2$$

Network architecture



[park 2019]

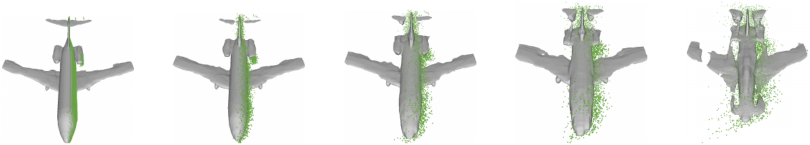
results


(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



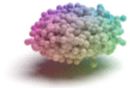
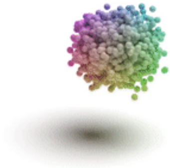
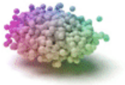
(a) No noise **(b)** $\alpha = 0.01$ **(c)** $\alpha = 0.02$ **(d)** $\alpha = 0.03$ **(e)** $\alpha = 0.05$

[park 2019]

Outline

Normalizing Flows

- "Synthesize a shape resembling a set of shapes"
- More generally synthesize a density of points resembling a density of points.
- Generative Methods: Many are limited in the number of points (PointNet-based) or work in the ambient space (Nerf-like - more recent).

Setting

Idea

- A family of shapes = a distribution of variables in a shape space.
- A shape = a distribution of points

Use the same process to sample a point on the surface or to sample a distribution from the set of distributions.

Parameterization

Instead of parameterizing the distribution of samples, model it as a invertible transformation of samples through *Normalizing Flows*. (samples = shapes OR points).

Normalizing Flow [Rezende 2015]

Normalizing Flow

A series of invertible mapping transforming an initial distribution into another one.

$$y \sim P(y),\ x = f_n \circ f_{n-1} \circ \cdots \circ f_1(y)$$

(x output variable, y latent variable, f_i invertible mappings)

- $y_k = f_k(y_{k-1}); y_0 = y$:

$$P(y_k) = P(y_{k-1}) \Big| \det \frac{\partial f_k}{\partial y_k} \Big|^{-1}$$

Final Formula

$$\log P(x) = \log P(y) - \sum_{k=1}^{n} \log \Big| \det \frac{\partial f_k}{\partial y_{k-1}} \Big|^{-1}$$

In practice f_i modeled by a neural network (Jacobian easy to compute)

Continuous Normalizing Flow [Yang 2020]

CNF

Instead of a series of invertible mapping, use a continuous time dynamic:

$$\frac{\partial y(t)}{\partial t} = f(y(t), t)$$

CNF model for $P(x)$ with $P(y)$ prior

$$x = y(t_0) + \int_{t_0}^{t_1} f(y(t), t)dt \ ; \ y(t_0) \sim P(y)$$

$$\log P(x) = \log P(y(t_0)) - \int_{t_0}^{t_1} Tr(\frac{\partial f}{\partial y(t)})dt$$

f is a neural network, and an ODE solver is used to compute CNF gradients.

Final loss

$$
\begin{aligned}
\mathcal{L}(X,\phi,\psi,\theta) &= \mathcal{E}_{Q_\phi(z|X)}[\log P_\theta(X|z)] - D_{KL}(Q_\phi(z|X)||P_\psi(z)) \\
&= \underbrace{\mathcal{E}_{Q_\phi(z|X)}[\log P_\theta(X|z)]}_{\mathcal{L}_{prior}} + \underbrace{\mathcal{E}_{Q_\phi(z|X)}[\log P_\psi(z)]}_{\mathcal{L}_{reconstruction}} + \underbrace{H[Q_\phi(z|X)]}_{\mathcal{L}_{entropy}}
\end{aligned}
$$

- $\mathcal{L}_{prior}$: the shape code z is generated following F_ψ^{-1} (shape should have a high probability under the prior modeled by a CNF).

Final loss

$$\begin{aligned}\mathcal{L}(X,\phi,\psi,\theta) &= \mathcal{E}_{Q_\phi(z|X)}[\log P_\theta(X|z)] - D_{KL}(Q_\phi(z|X)||P_\psi(z)) \\ &= \underbrace{\mathcal{E}_{Q_\phi(z|X)}[\log P_\theta(X|z)]}_{\mathcal{L}_{prior}} + \underbrace{\mathcal{E}_{Q_\phi(z|X)}[\log P_\psi(z)]}_{\mathcal{L}_{reconstruction}} + \underbrace{H[Q_\phi(z|X)]}_{\mathcal{L}_{entropy}}\end{aligned}$$

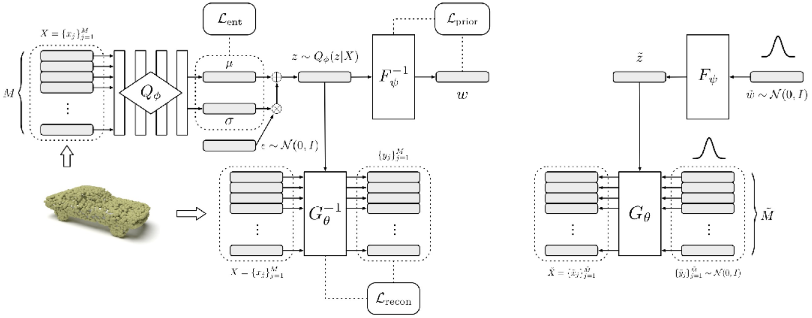
- $\mathcal{L}_{prior}$: the shape code z is generated following F_ψ^{-1} (shape should have a high probability under the prior modeled by a CNF).
- $\mathcal{L}_{reconstruction}$: X is likely to be reconstructed from z following G_θ^{-1}.

Final loss

$$\begin{aligned}\mathcal{L}(X,\phi,\psi,\theta) &= \mathcal{E}_{Q_\phi(z|X)}[\log P_\theta(X|z)] - D_{KL}(Q_\phi(z|X)||P_\psi(z)) \\ &= \underbrace{\mathcal{E}_{Q_\phi(z|X)}[\log P_\theta(X|z)]}_{\mathcal{L}_{prior}} + \underbrace{\mathcal{E}_{Q_\phi(z|X)}[\log P_\psi(z)]}_{\mathcal{L}_{reconstruction}} + \underbrace{H[Q_\phi(z|X)]}_{\mathcal{L}_{entropy}}\end{aligned}$$

- $\mathcal{L}_{prior}$: the shape code z is generated following F_ψ^{-1} (shape should have a high probability under the prior modeled by a CNF).
- $\mathcal{L}_{reconstruction}$: X is likely to be reconstructed from z following G_θ^{-1}.
- $\mathcal{L}_{ent}$ checks that z refers to X.

Full Network

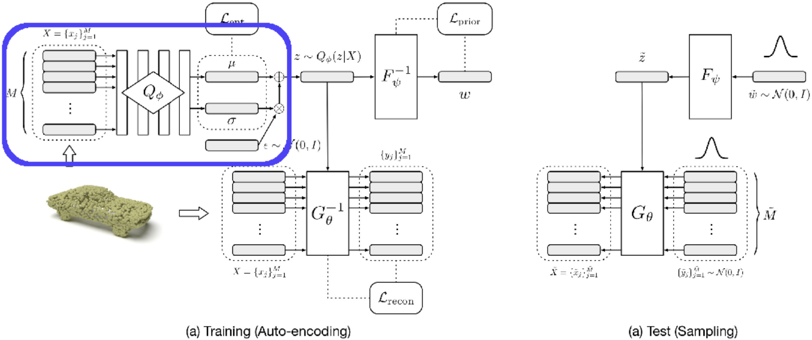


(a) Training (Auto-encoding)

(a) Test (Sampling)

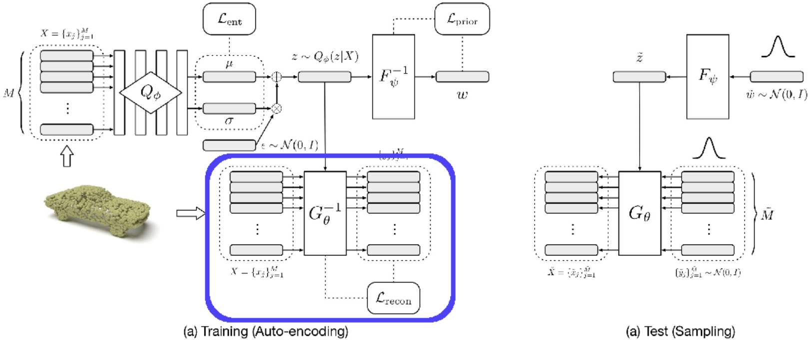
$$\mathcal{L}(X, \phi, \psi, \theta)$$

Breaking it into pieces



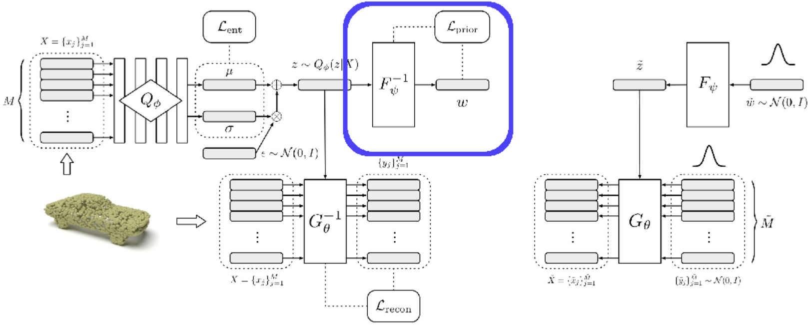
$$\mathcal{L}_{ent}(X, \phi, \psi)$$

Breaking it into pieces



$$\mathcal{L}_{reconstruction}(X, \theta, \phi)$$

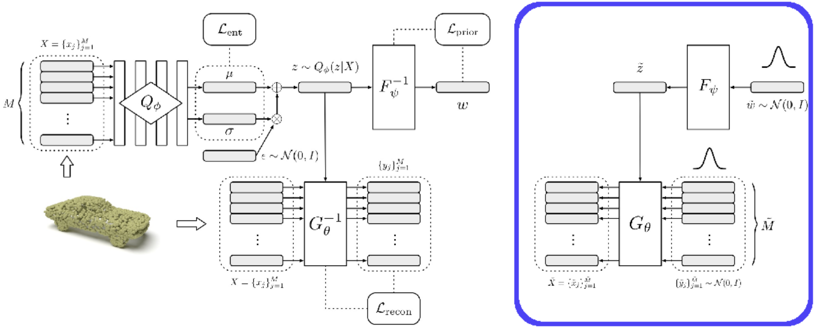
Breaking it into pieces



(a) Training (Auto-encoding) (a) Test (Sampling)

$$\mathcal{L}_{prior}(X, \phi)$$

Sampling



(a) Training (Auto-encoding) (a) Test (Sampling)

Generate w (Gaussian), use CNF F_ψ to get z. Use $G_\theta(.; z)$ to sample points

Optimization

- **Encoder** $Q_\phi(z|x)$: Pointnet 1D convolutions + 2layer-mlp converting into a D_Z-dimensional representation.

Optimization

- **Encoder** $Q_\phi(z|x)$: Pointnet 1D convolutions + 2layer-mlp converting into a D_Z-dimensional representation.
- **CNF Prior** follows Ffjord [Grathwhohl 2018]. Models f_ψ governing the PDE $\frac{\partial y}{\partial t} = f_\psi(y(t), t)$ with a network (using *Concatsquash layer*).

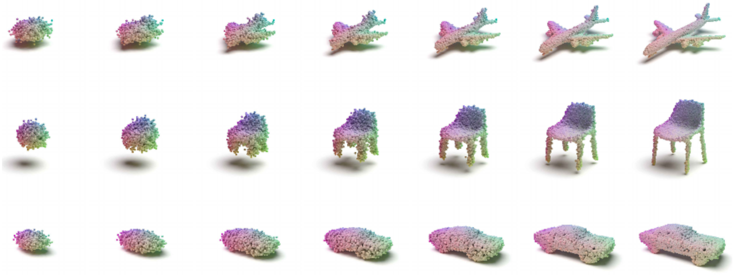
Optimization

- **Encoder** $Q_\phi(z|x)$: Pointnet 1D convolutions + 2layer-mlp converting into a D_Z-dimensional representation.
- **CNF Prior** follows Ffjord [Grathwhohl 2018]. Models f_ψ governing the PDE $\frac{\partial y}{\partial t} = f_\psi(y(t), t)$ with a network (using *Concatsquash layer*).
- **CNF Decoder** uses *Conditional concatsquash layers*

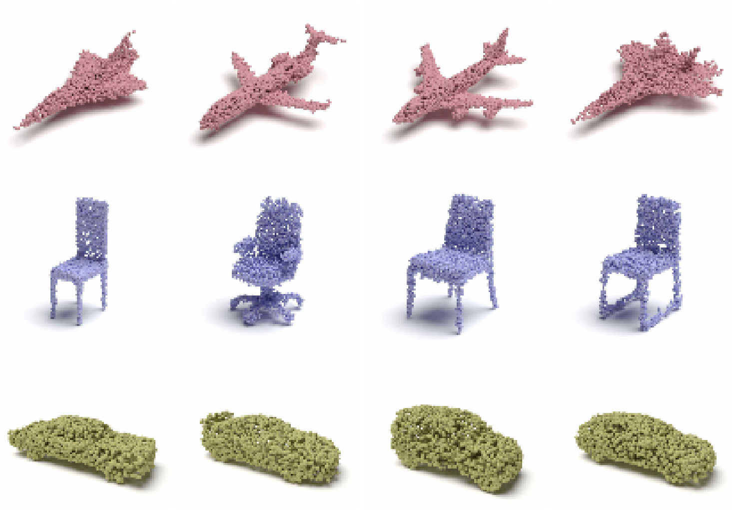
Optimization

- **Encoder** $Q_\phi(z|x)$: Pointnet 1D convolutions + 2layer-mlp converting into a D_Z-dimensional representation.
- **CNF Prior** follows Ffjord [Grathwhohl 2018]. Models f_ψ governing the PDE $\frac{\partial y}{\partial t} = f_\psi(y(t), t)$ with a network (using *Concatsquash layer*).
- **CNF Decoder** uses *Conditional concatsquash layers*
- ODE-compatible backprop: Backpropagating through ODE solutions with the adjoint Method [Chen 2018] (in practice: DOPRI method Dormand & Prince 1980, RKDP).

Results



Results



Latent space

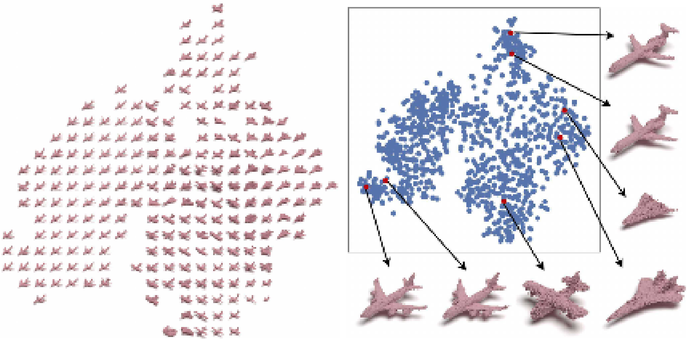


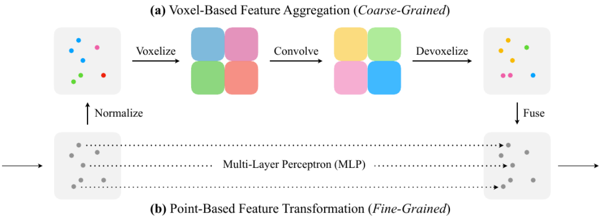
Figure 6: Visualization of latent space.

Diffusion-based shape synthesis [LION, Zeng 2022]

[Zeng 2022]

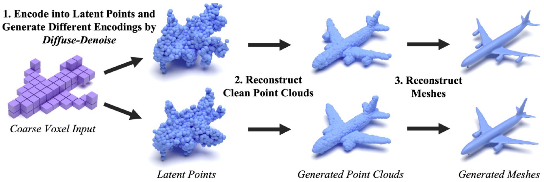
- No grid, non-euclidean data $\rightarrow$ extremely hard.
- Based on denoising diffusion in latent space and in ambient space.
- Point set structured through a voxel grid Point-Voxel CNN [Liu 2019]

Convolution on point clouds via voxel proxy [Liu 19]



- Features per points but aggregated per voxel (coarse grained level)
- Per point feature (fine grained level)

Results

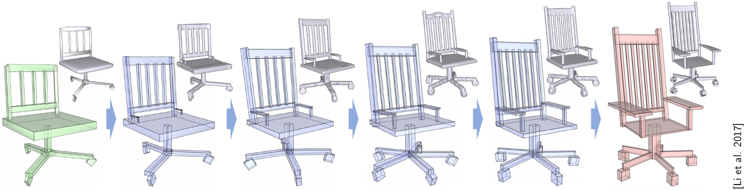


[Zeng 2022]

Detail variation from a coarse shape embedding.

Outline

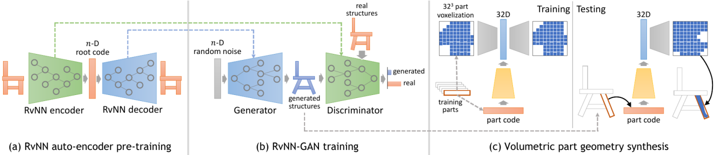
An example for generating shapes [GRASS, Li et al. 2017]



- Input data: set of shapes with a semantic segmentation into parts.

Algorithm

- Step 1: Learn a code representing an arrangement of boxes.
- Step 2: Train a GAN for generating a new structure
- Step 3: Use voxelization in each box to synthesize the local geometry.



[Li et al. 2017]

Step 1: Learn a code

Key idea

Shape components are commonly arranged or perceived to be arranged hierarchically. Goal of the code: encode this hierarchy of parts

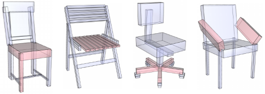
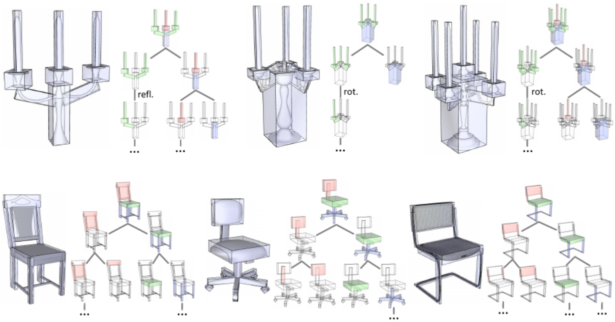


Fig. 3. Merging criteria used by our model demonstrated with 3D shapes represented by part bounding boxes (relevant parts highlighted in red). From left: (a) two adjacent parts, (b) translational symmetry, (c) rotational symmetry, and (d) reflective symmetry.

[Li et al. 2017]

- Recursive auto-encoder for binary trees: encode the structure into a code; decode and compare the recovered structure.
- Recursively merge parts that are either adjacent or symmetric (rotational, translational, reflectional)
- Training: generate plausible hierarchies for each shape (sample the space of plausible part groupings)
- Adjacency and Symmetry encoder/decoder (transform a code into another encodes the symmetry and the generator)
- Additionally: Box encoder/Node classifier

Learned hierarchies

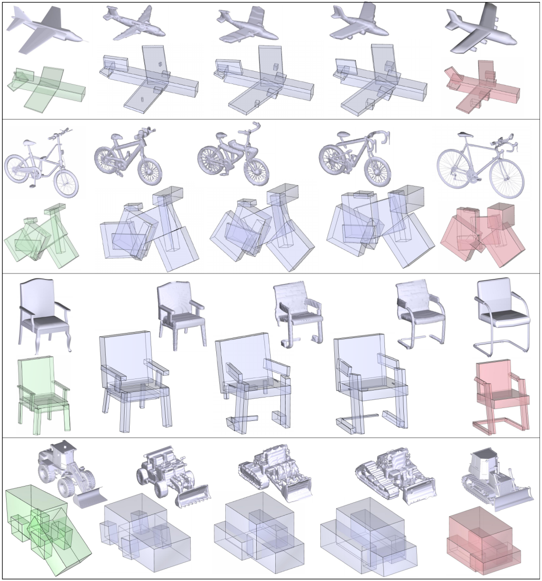


[Li et al. 2017]

In a nutshell

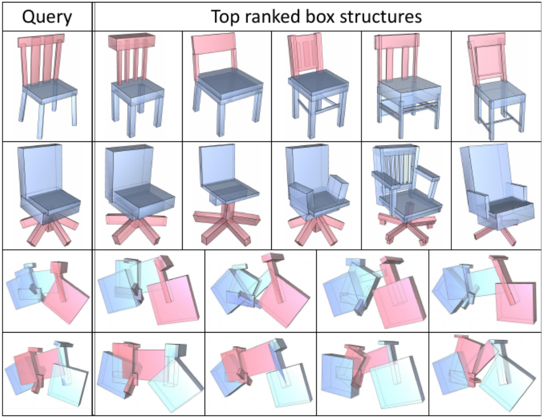
Transform a binary tree into a meaningful hierarchy while minimizing the loss (sum of bounding boxes distances)

Application: interpolation



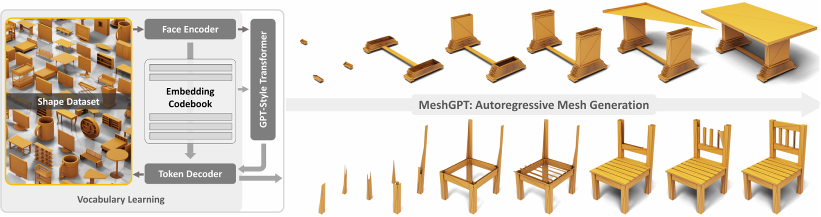
[Li et al. 2017]

Application: shape query



[Li et al. 2017]

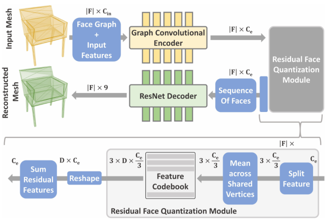
MeshGPT [Siddiqi et al. 2023]



[Siddiqi et al. 2023]

- Following text generation idea: generate a mesh as a sequence of triangles

MeshGPT - Principle



[Siddiqi et al. 2023]

- Learns a vocabulary of latent representations of faces
- Uses these latent representations as tokens
- GPT-like transformer: predicts next token from previous tokens auto-regressively.
- 1D Resnet decodes the latent representation sequences into triangles

Result

Resuls is a triangle soup: needs post-processing to turn it into a watertight mesh

MeshGPT - Results

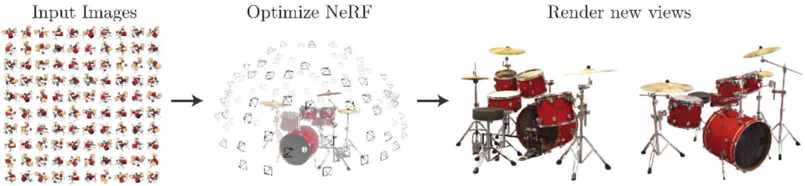
[Siddiqi et al. 2023]

MeshGPT - Results

[Siddiqi et al. 2023]

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

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

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

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

with:

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

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

Integral approximation

- Stratified sampling along the ray of positions t_i

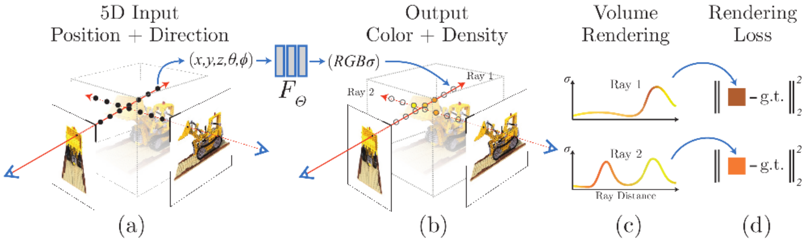
Discrete Version

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

with

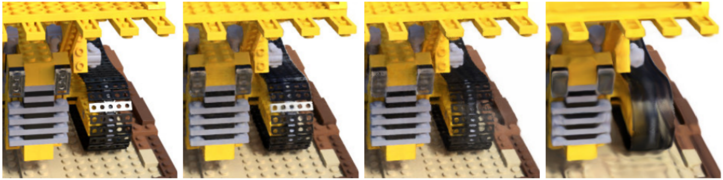
$$T_i = \sum_i \exp(-\sigma(t_i)\|t_{i+1} - t_i\|)$$

Training



[Mildenhall et al. 2020]

Positional Encoding



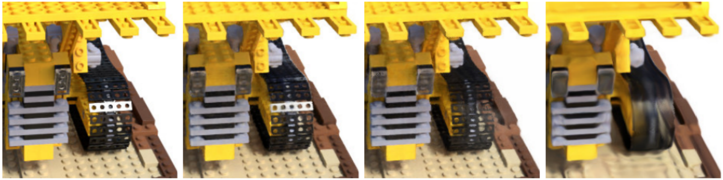
Ground Truth | Complete Model | No View Dependence | No Positional Encoding

- 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

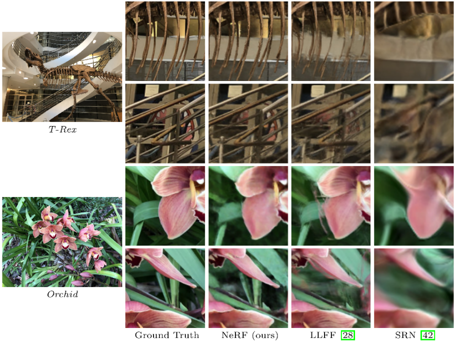
View-dependency



Ground Truth | Complete Model | No View Dependence | No Positional Encoding

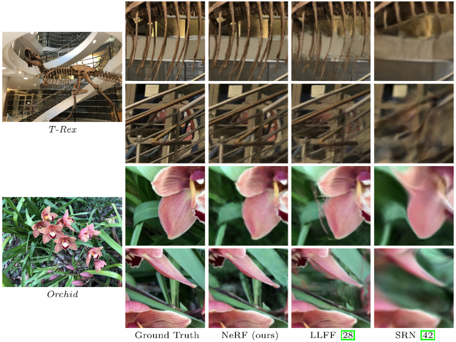
- View-dependent radiance is what allows to capture mirror reflections

Results



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

Results

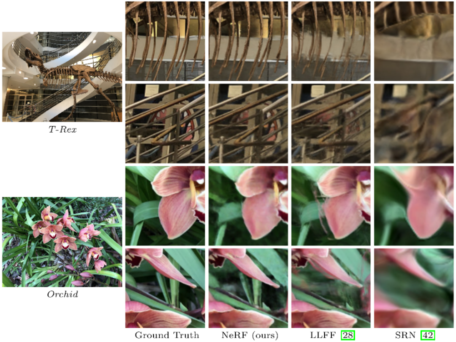


Video: `https://www.matthewtancik.com/nerf`

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

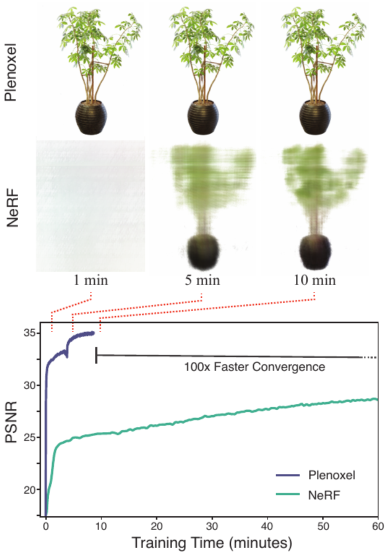


Video: `https://www.matthewtancik.com/nerf`

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])*

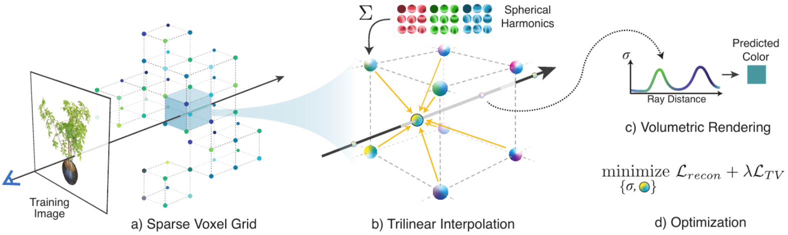
After Nerf... Plenoxels [Yu et al. 2021]



[Yu et al. 2021]

- No neural net
- (way) faster than nerf

Method



[Yu et al. 2021]

Spherical harmonics

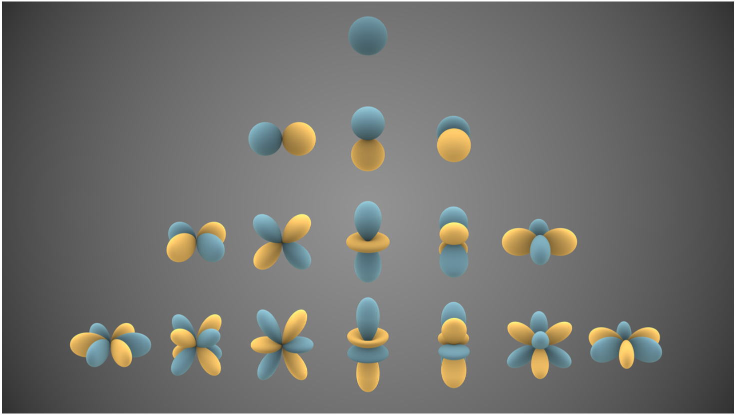


Image by Inigo Quilez (Wikipedia)

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

- 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:

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

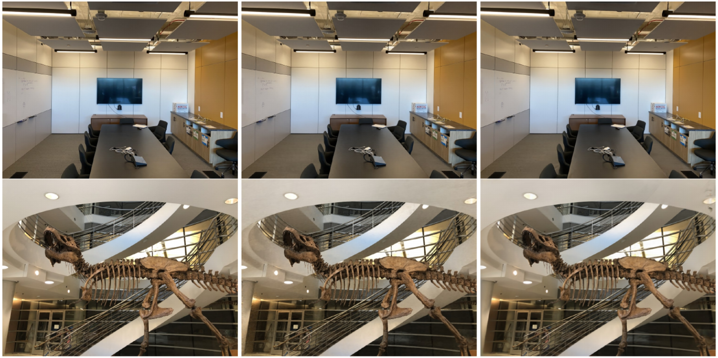
($\mathcal{V}$ and $\mathcal{R}$ stochastic samplings of the grid vertices and rays)

Results

Ground Truth NeRF++ [57] Plenoxels

[Yu et al. 2021]

Results



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

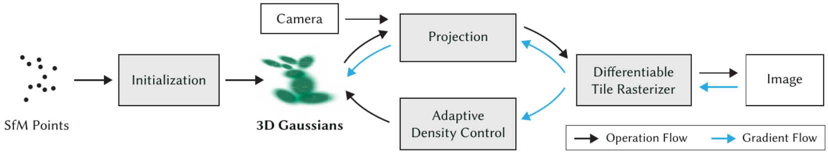
[Yu et al. 2021]

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

Gaussian Splatting

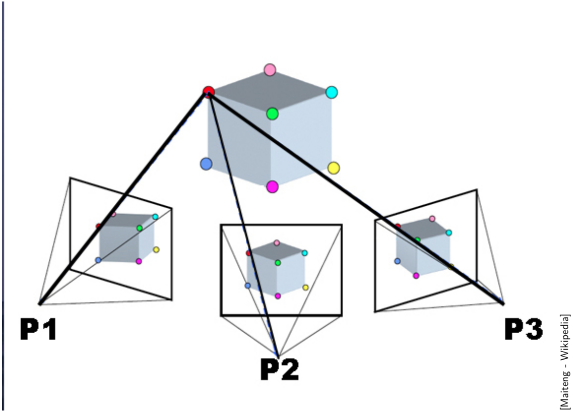
- Build on point set Splatting [Zwicker 2001]
- Each point is the center of a small 3D Gaussian on it,
- Each 3D Gaussian is represented by a quaternion and 3 scaling factors.
- Gaussian splat = gaussian parameters + opacity + Spherical harmonics

Overview



[Kerble 2023]

Structure from Motion (SfM)



- Cameras calibrated by Structure from Motion [Snavely 2006]

Rendering a Gaussian splat scene

- Projective space Gaussian giving the color.

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

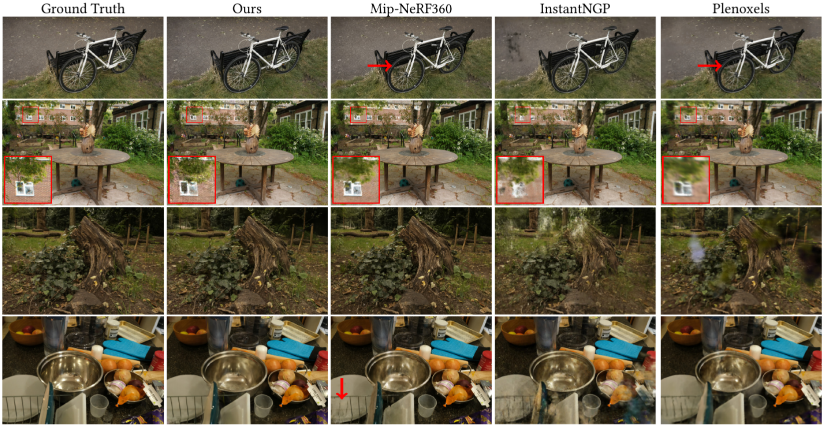
- Viewing direction W $\Sigma' = JW\Sigma W^T$
- J jacobian of the affine approx of the projective transformation:

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

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



imcredits[Kerble 2023]

Number of iterations

imcredits[Kerble 2023]

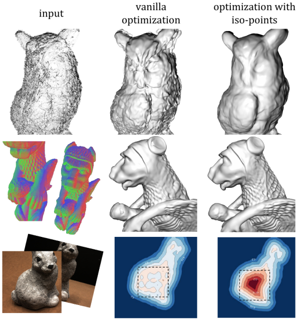
Conclusion

- Geometric data synthesis is hard
- Nerf/Gaussian Splat: do we need to compute the geometry or only render?
- Multi-resolution, levels of details for neural implicits.

Outline

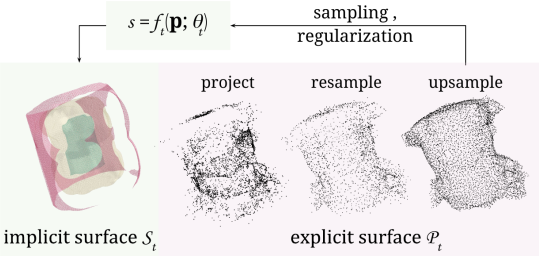
Projecting points on the surface [Yifan 2021]

- Sample points on a neural implicit
- Use them to improve robustness and accuracy



[Yifan 2021]

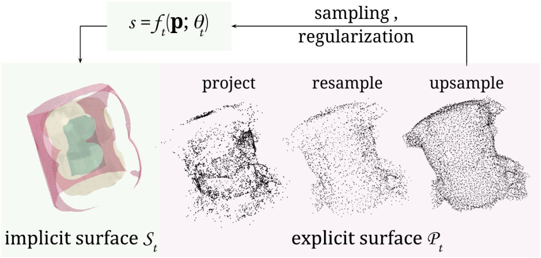
Projection on the surface



[Yifan 2021]

- Starting from a point q_0 in $\mathbb{R}^3$ project it on the surface
- Newton Iterations: $q_{k+1} = q_k - J_f^+(q_k) f_\theta(q_k)$ with $J_f^+(q_k) = \frac{1}{\|J_f(q_k)\|^2} J_f(q_k)$
- For nonsmooth fields, set an upper threshold for the displacement magnitude

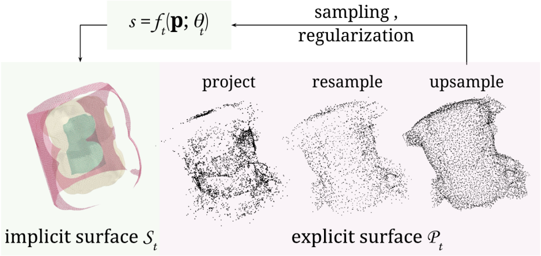
Uniform resampling



[Yifan 2021]

- Move the points away from dense areas $\tilde{q} \leftarrow \tilde{q} - \alpha r$
- α step size
- $r = \sum_{\tilde{q}_i \in \mathcal{N}(\tilde{q})} w(\tilde{q}_i, \tilde{q}) \frac{\tilde{q}_i - \tilde{q}}{||\tilde{q}_i - \tilde{q}||}$

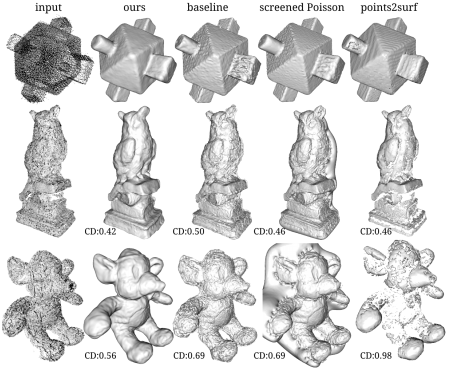
Upsampling



[Yifan 2021]

- Move the points away from the edges (Edge-away resampling [Huang 2011])
- Each point is :
  - attracted to points that have a similar normal
  - repulsed from dense areas.
- Upsampled points are reprojected on the surface

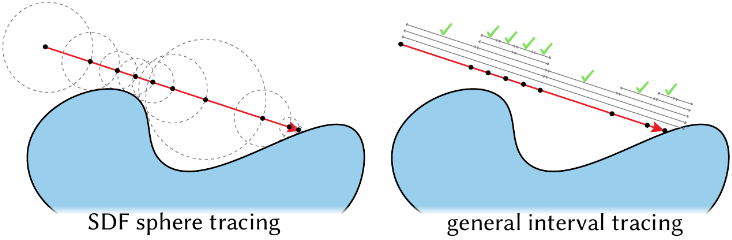
Application to INR fitting regularization



[Yifan 2021]

- Warmup training (300 iterations)
- Extract isopoints + add isopoints to data points
- Update the isopoints every 1000 iterations

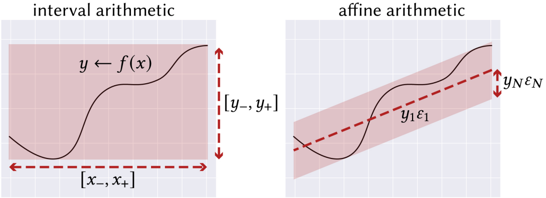
Arithmetic Queries [Sharp 2022]



[Sharp 2022]

- f_θ a neural implicit *Not necessarily a signed distance field*.
- Sphere tracing for SDF, interval arithmetic for general implicit field.
- Goal: adapt interval arithmetic for neural implicits.

Affine arithmetic [Comba and Stolfi 1993]



[Sharp 2022]

- Interval arithmetic gives loose bounds
- Affine arithmetic: tracks affine coefficients through computation
- Similar to forward auto-diff: linear operations, nonlinear operations by linearization (adds affine coefficients!)

MLP

Affine operations followed by ReLU nonlinearity

Nonlinearities

- $\hat{x} = x_0 + \sum_{i=1}^{N} x_i \varepsilon_i \;\; \varepsilon_i \in [-1, 1]$
- Replace f by a linear approximation $\hat{f}(x) \approx \alpha x + \beta$
- $\gamma = \max_{x \in range(\hat{x})} |f(x) - \hat{f}(x)|$

Nonlinearities

- $\hat{x} = x_0 + \sum_{i=1}^{N} x_i \varepsilon_i \;\; \varepsilon_i \in [-1, 1]$
- Replace f by a linear approximation $\hat{f}(x) \approx \alpha x + \beta$
- $\gamma = \max_{x \in range(\hat{x})} |f(x) - \hat{f}(x)|$
- $\hat{y} = f(\hat{x}) = \alpha x_0 + \beta + \sum_{i=1}^{N} \alpha x_i \varepsilon_i + \gamma \varepsilon_{N+1}$

Nonlinearities

- $\hat{x} = x_0 + \sum_{i=1}^{N} x_i \varepsilon_i \;\; \varepsilon_i \in [-1, 1]$
- Replace f by a linear approximation $\hat{f}(x) \approx \alpha x + \beta$
- $\gamma = \max_{x \in range(\hat{x})} |f(x) - \hat{f}(x)|$
- $\hat{y} = f(\hat{x}) = \alpha x_0 + \beta + \sum_{i=1}^{N} \alpha x_i \varepsilon_i + \gamma \varepsilon_{N+1}$
- Each layer with width W adds W new coefficients.

Nonlinearities

- $\hat{x} = x_0 + \sum_{i=1}^{N} x_i \varepsilon_i \;\; \varepsilon_i \in [-1, 1]$
- Replace f by a linear approximation $\hat{f}(x) \approx \alpha x + \beta$
- $\gamma = \max_{x \in range(\hat{x})} |f(x) - \hat{f}(x)|$
- $\hat{y} = f(\hat{x}) = \alpha x_0 + \beta + \sum_{i=1}^{N} \alpha x_i \varepsilon_i + \gamma \varepsilon_{N+1}$
- Each layer with width W adds W new coefficients.

Solution

Periodically replace a set of coefficients with a single new coefficients

$$condense(\hat{x}, \mathcal{D}) = x_0 + \sum_{i \notin \mathcal{D}} x_i \varepsilon_i + (\sum_{i \in \mathcal{D}} |x_i|)\varepsilon_{N+1}$$

Range bounds

Procedure 1 RANGEBOUND($f_\theta, c, \{v_i\}$)

Input: A function $f_\theta : \mathbb{R}^d \to \mathbb{R}$ and a query box B of dimension $s \leq d$ defined by its center $c \in \mathbb{R}^d$, and s orthogonal box axis vectors $\{v_i \in \mathbb{R}^d\}$, not necessarily coordinate axis-aligned.

Output: A bound on the sign of $f_\theta(x)\ \ \forall x \in B$ as one of `POSITIVE`, `NEGATIVE`, or `UNKNOWN`.

$\hat{\mathbf{x}} \leftarrow c + \sum_{i=1}^{s} v_i \varepsilon_i$ ▷*Construct affine bounds defining the box*

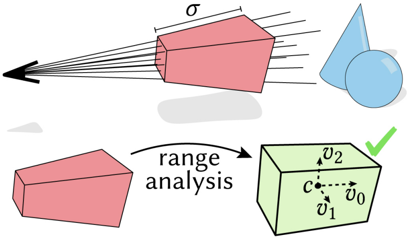
$\hat{\mathbf{y}} \leftarrow f_\theta(\hat{\mathbf{x}})$ ▷*Propagate affine bounds (Section 3.2)*

$[y_-, y_+] \leftarrow \text{range}(\hat{\mathbf{y}})$ ▷*Bound the output (Equation 3)*

if $y_- > 0$ **then return** `POSITIVE`

if $y_+ < 0$ **then return** `NEGATIVE`

else return `UNKNOWN`



[Sharp 2022]

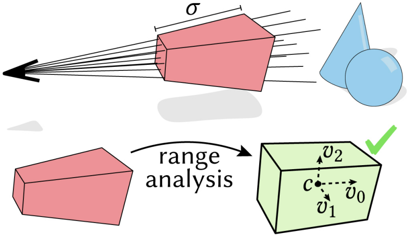
Range bounds

Procedure 1 RangeBound($f_\theta, c, \{v_i\}$)

Input: A function $f_\theta : \mathbb{R}^d \to \mathbb{R}$ and a query box B of dimension $s \le d$ defined by its center $c \in \mathbb{R}^d$, and s orthogonal box axis vectors $\{v_i \in \mathbb{R}^d\}$, not necessarily coordinate axis-aligned.

Output: A bound on the sign of $f_\theta(x)$ $\forall x \in B$ as one of `POSITIVE`, `NEGATIVE`, or `UNKNOWN`.

$\hat{\mathbf{x}} \leftarrow c + \sum_{i=1}^{s} v_i \varepsilon_i$ ▷ *Construct affine bounds defining the box*
$\hat{\mathbf{y}} \leftarrow f_\theta(\hat{\mathbf{x}})$ ▷ *Propagate affine bounds (Section 3.2)*
$[y_-, y_+] \leftarrow \mathrm{range}(\hat{\mathbf{y}})$ ▷ *Bound the output (Equation 3)*
if $y_- > 0$ **then return** `POSITIVE`
if $y_+ < 0$ **then return** `NEGATIVE`
else return `UNKNOWN`

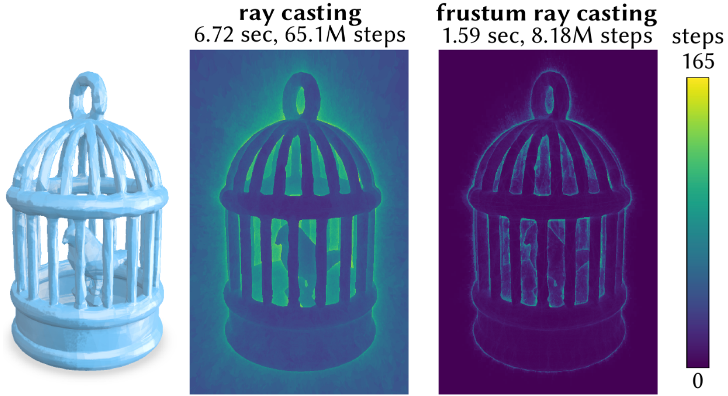


[Sharp 2022]

Unknown?

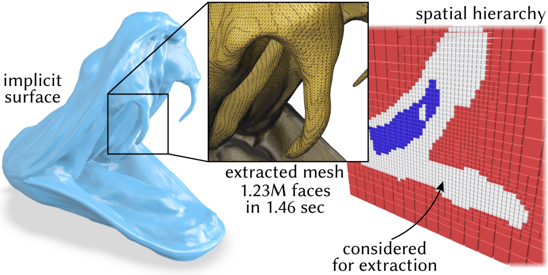
Subdivide the box.

Ray casting vs frustum ray casting

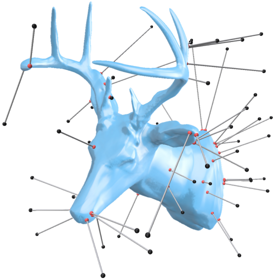


[Sharp 2022]

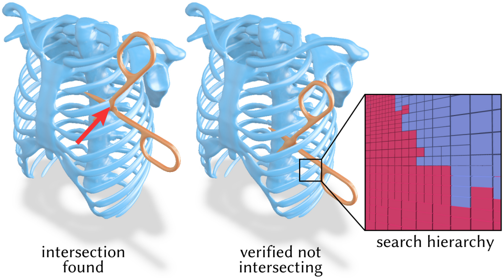
Applications



Mesh extraction



Closest point



Mesh Intersection