Reconstructing Solid Models from Oriented Point Sets: FFT vs. Poisson

M. Kazhdan[Δ], M. Bolitho[Δ], R. Burns[Δ],
H. Hoppe[∇], K. Dalal[∇], and A. Klein[∇]

[Δ]Johns Hopkins University [∇]Microsoft (Research)

Reconstruction of Solid Models from Oriented Points Sets. Kazhdan (2005)
Poisson Surface Reconstruction. Kazhdan et al. (2006)
Multilevel Streaming for Out-of-Core Surface Reconstruction. Bolitho et al. (2007)
Unconstrained Isosurface Extraction on Arbitrary Octrees. Kazhdan et al. (2007)
Parallel Poisson Surface Reconstruction. Bolitho et al. (2009)
Screened Poisson Reconstruction. Kazhdan and Hoppe (2013)

Related Work

Three general approaches:

1. Computational Geometry

   Boissonnat, 1984
   Edelsbrunner, 1984
   Amenta *et al*., 1998
   Dey *et al*., 2003

2. Surface Fitting

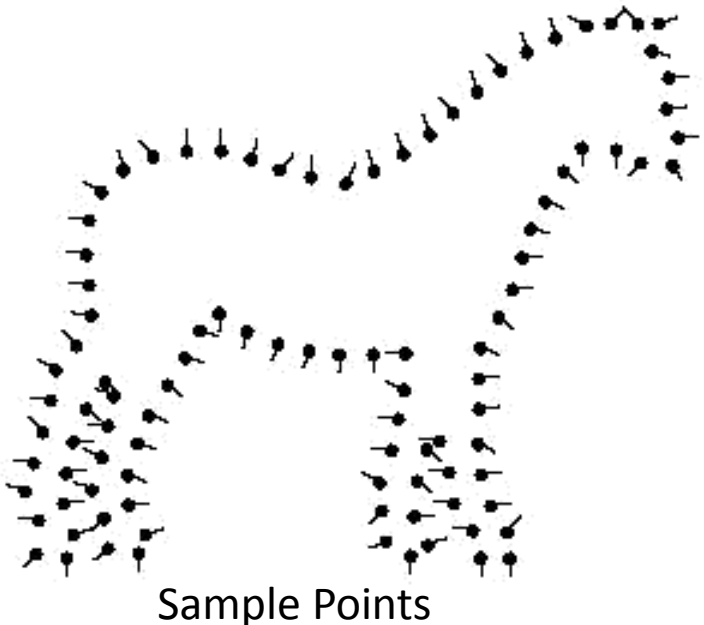
   Terzopoulos *et al.*, 1991
   Chen *et al.*, 1995

3. Implicit Function Fitting

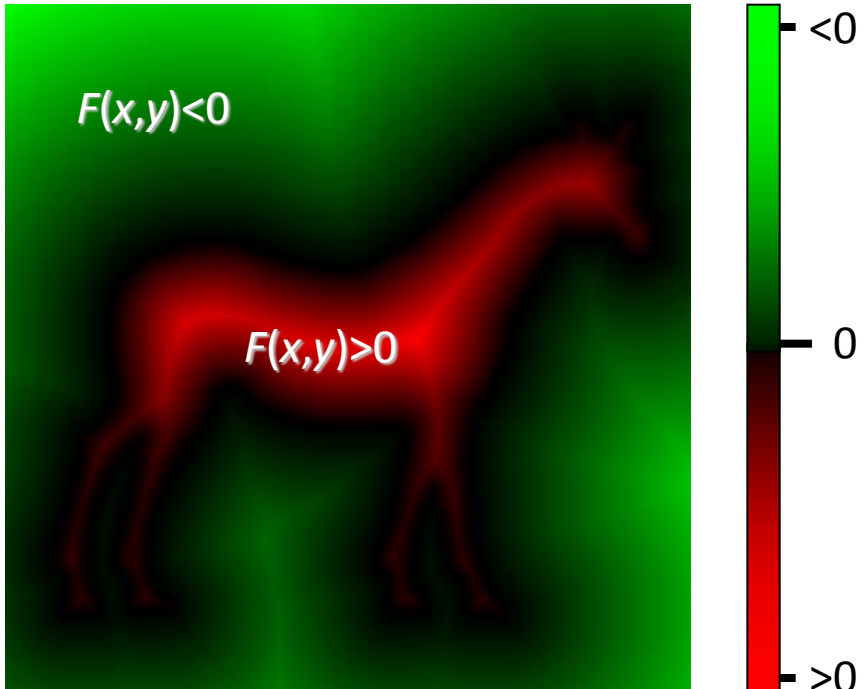
   Hoppe *et al.*, 1992
   Curless *et al.*, 1996
   Whitaker, 1998
   Carr *et al.*, 2001
   Davis *et al.*, 2002
   Ohtake *et al.*, 2004
   Turk *et al.*, 2004
   Shen *et al.*, 2004

Related Work

3. Implicit Function Fitting
   - Use the (oriented) points to define a function that is negative inside the shape and positive outside.

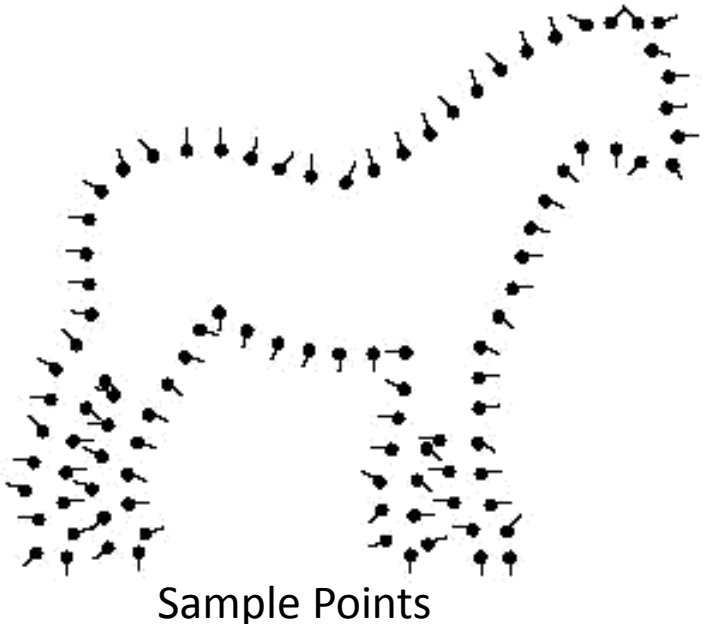


Sample Points

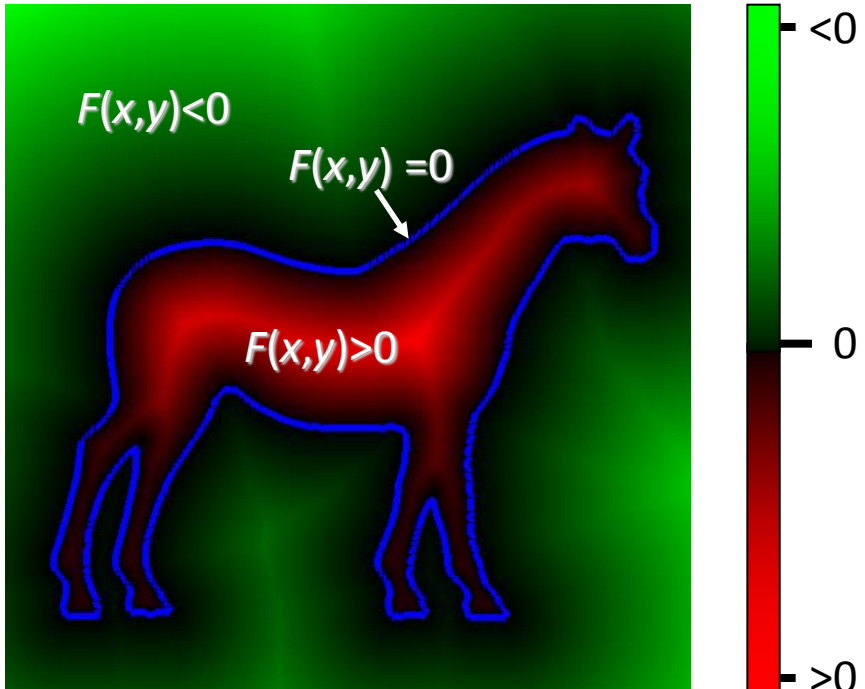


Related Work

3. Implicit Function Fitting
   - Use the (oriented) points to define a function that is negative inside the shape and positive outside.
   - Extract the zero-crossing iso-surface.



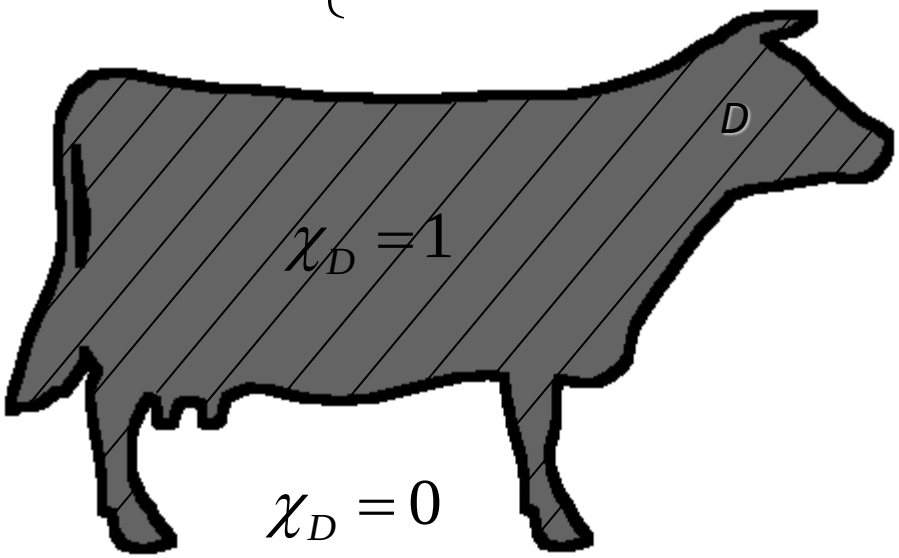
Sample Points



Which Implicit Function?

Our goal is to reconstruct the simplest implicit function characterizing the surface – the *indicator function*:

$$\chi_D(x,y,z) = \begin{cases} 1 & \text{if } (x,y,z) \in D \\ 0 & \text{otherwise} \end{cases}$$



Outline

- Introduction
- Related Work
- Two Approaches
  - FFT: What can we compute?
  - Poisson: What are we representing?
- Results

Motivation

What can we do with a set of oriented points?

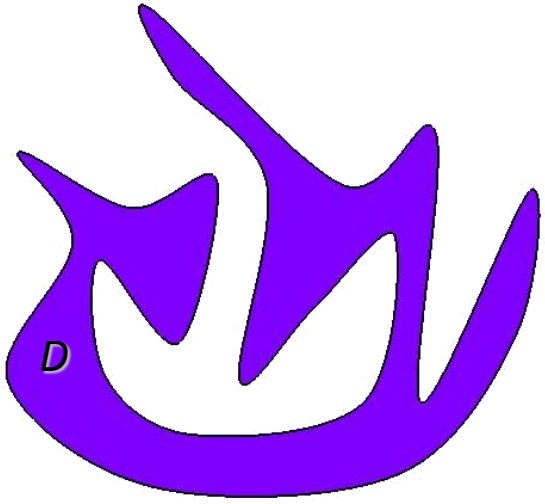
Motivation

What can we do with a set of oriented points?

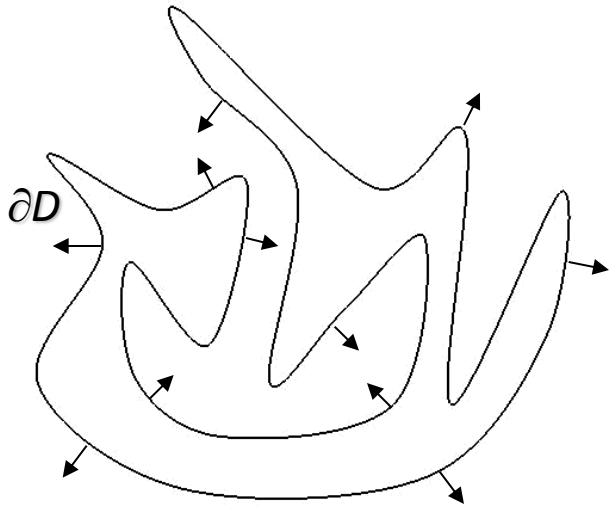
Divergence (Gauss's) Theorem:

Given a vector field $\vec{F}$ and a domain *D*:

$$\int_D \operatorname{div}\left(\vec{F}\right)(p)\, dp = \int_{\partial D} \left\langle \vec{F}(p), \vec{n}(p) \right\rangle dp$$



=



$$\int_D \operatorname{div}\left(\vec{F}\right)(p)\, dp \qquad \int_{\partial D} \left\langle \vec{F}(p), \vec{n}(p) \right\rangle dp$$

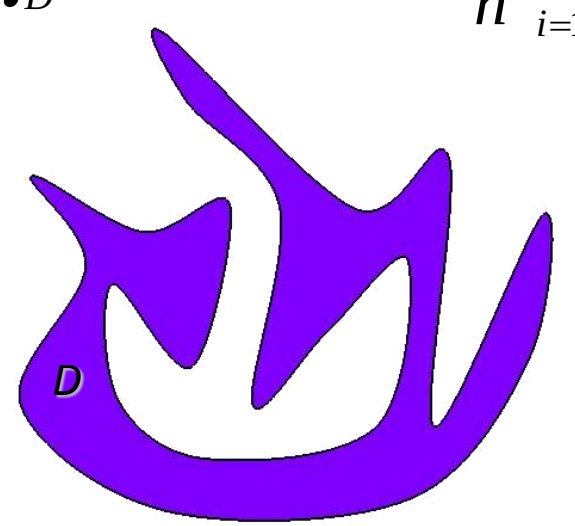
Motivation

What can we do with a set of oriented points?

Divergence (Gauss's) Theorem:

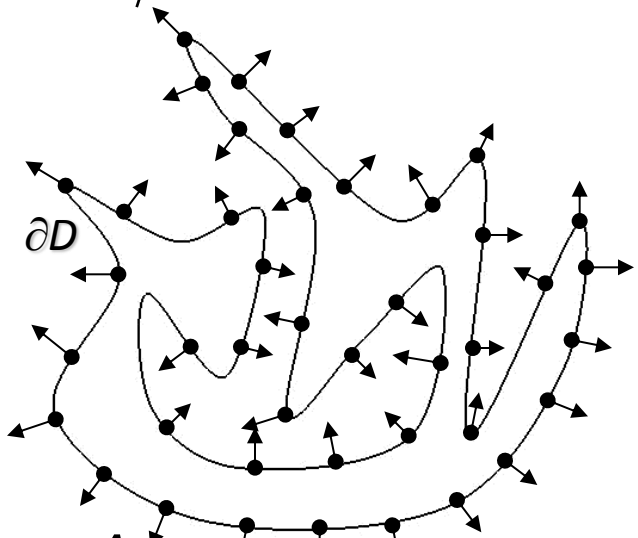
Given a vector field $\vec{F}$ and a domain *D*:

$$\int_D \operatorname{div}\left(\vec{F}\right)(p)\,dp \approx \frac{A}{n}\sum_{i=1}^{n}\left\langle \vec{F}(p_i), n_i \right\rangle$$



$$\int_D \operatorname{div}\left(\vec{F}\right)(p)\,dp$$

$\approx$



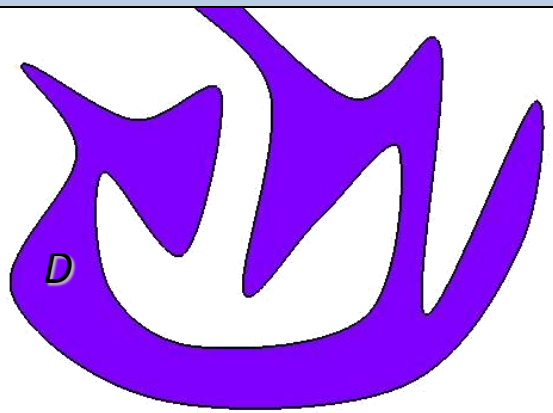
$$\frac{A}{n}\sum\left\langle \vec{F}(p_i), n_i \right\rangle$$

Motivation

What can we do with a set of oriented points?

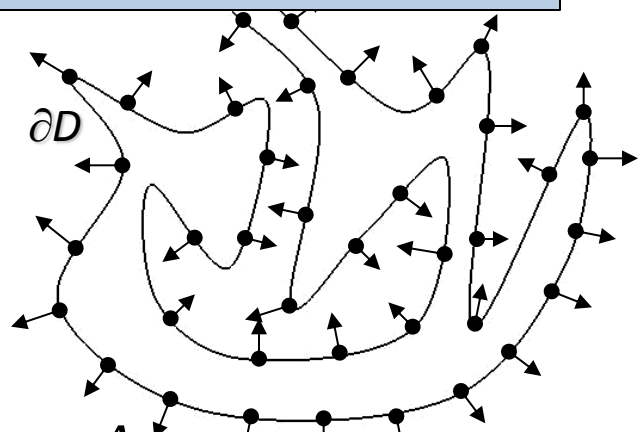
Divergence (Gauss's) Theorem:

Without explicitly knowing the volume, with a set of oriented points we can approximate volume integrals.



$$\int_D \operatorname{div}\left(\vec{F}\right)(p)\,dp$$

$\approx$



$$\frac{A}{n}\sum\left\langle \vec{F}(p_i), n_i \right\rangle$$

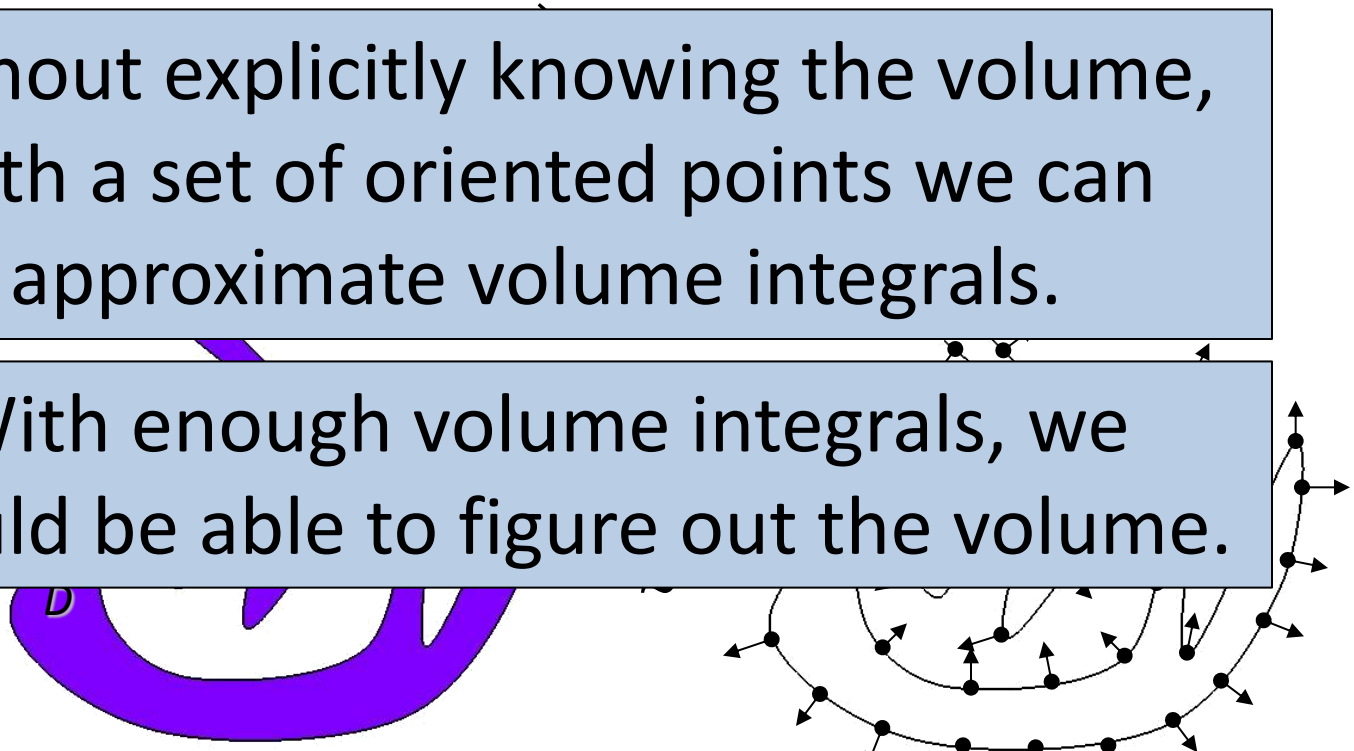
Motivation

What can we do with a set of oriented points?

Divergence (Gauss's) Theorem:

Without explicitly knowing the volume, with a set of oriented points we can approximate volume integrals.

With enough volume integrals, we should be able to figure out the volume.



$$\int_D \operatorname{div}\left(\vec{F}\right)(p)\,dp$$

$$\frac{A}{n}\sum\left\langle \vec{F}(p_i), n_i \right\rangle$$

Computing χ_D (Step 1)

Computing the indicator function is equivalent to computing its Fourier decomposition:

$$\chi_D(x,y,z) = \sum_{l,m,n} \hat{\chi}_D(l,m,n) e^{2\pi i(lx+my+nz)}$$

Computing χ_D (Step 2)

Computing the Fourier decomposition of the indicator function is equivalent to computing a set of volume integrals:

$$\hat{\chi}_D(l,m,n) = \int_{[0,1]^3} \chi_D(x,y,z) e^{-2\pi i(lx+my+nz)} dxdydz$$

Computing χ_D (Step 2)

Computing the Fourier decomposition of the indicator function is equivalent to computing a set of volume integrals:

$$\hat{\chi}_D(l,m,n) = \int_{[0,1]^3} \chi_D(x,y,z) e^{-2\pi i(lx+my+nz)} dxdydz$$

$$= \int_D e^{-2\pi i(lx+my+nz)} dxdydz$$

since the indicator function is one inside of *D* and zero outside.

Computing χ_D (Step 3)

Surface Integration

If $\vec{F}_{lmn}(x,y,z)$ is any vector field whose divergence is equal to the (l,m,n)-th complex exponential:

$$\operatorname{div}\left(\vec{F}_{lmn}\right)(x,y,z) = e^{-2\pi i(lx+my+nz)}$$

Then the volume integral can be expressed as a surface integral:

$$\int_D e^{-2\pi i(lx+my+nz)}\,dxdydz = \int_{\partial D}\left\langle \vec{F}_{lmn}(p), n(p)\right\rangle dp$$

Reconstruction Algorithm

Given a set of oriented point samples $\{(p_j, n_j)\}$

- We approximate the Fourier coefficients of χ_D:

$$\hat{\chi}_D(l,m,n) \approx \sum_{j=1}^{k} \left\langle \vec{F}_{lmn}(p_k), n_k \right\rangle$$

Reconstruction Algorithm

Given a set of oriented point samples $\{(p_j, n_j)\}$

- We approximate the Fourier coefficients of χ_D:

$$\hat{\chi}_D(l,m,n) \approx \sum_{j=1}^{k} \left\langle \vec{F}_{lmn}(p_k), n_k \right\rangle$$

- To obtain the Fourier Decomposition:

$$\chi_D(x,y,z) = \sum_{l,m,n}^{n} \hat{\chi}_D(l,m,n) e^{2\pi i(lx+my+nz)}$$

Reconstruction Algorithm

Given a set of oriented point samples $\{(p_j, n_j)\}$

- We approximate the Fourier coefficients of χ_D:

$$\hat{\chi}_D(l,m,n) \approx \sum_{j=1}^{k} \left\langle \vec{F}_{lmn}(p_k), n_k \right\rangle$$

- To obtain the Fourier Decomposition:

$$\chi_D(x,y,z) = \sum_{l,m,n}^{n} \hat{\chi}_D(l,m,n) e^{2\pi i(lx+my+nz)}$$

- From which we extract the iso-surface of χ_D:

$$\partial D = \left\{ p \middle| \chi_D(p) = 0.5 \right\}$$

Implementation

To implement, we must find vector fields whose divergences are the complex exponentials:

$$\mathrm{div}\left(\vec{F}_{lmn}\right)(x,y,z) = e^{-2\pi i(lx+my+nz)}$$

Implementation

To implement, we must find vector fields whose divergences are the complex exponentials:

$$\operatorname{div}\left(\vec{F}_{lmn}\right)(x,y,z) = e^{-2\pi i(lx+my+nz)}$$

There are many of these:

$$\vec{F}_{lmn}(x,y,z) = \frac{1}{-2\pi i}\begin{pmatrix}\frac{1}{l}\\0\\0\end{pmatrix}e^{-2\pi i(lx+my+nz)} \qquad \vec{F}_{lmn}(x,y,z) = \frac{1}{-2\pi i}\begin{pmatrix}0\\\frac{1}{m}\\0\end{pmatrix}e^{-2\pi i(lx+my+nz)} \qquad \vec{F}_{lmn}(x,y,z) = \frac{1}{-2\pi i}\begin{pmatrix}0\\0\\\frac{1}{n}\end{pmatrix}e^{-2\pi i(lx+my+nz)}$$

Implementation

To implement, we must find vector fields whose divergences are the complex exponentials:

$$\operatorname{div}\left(\vec{F}_{lmn}\right)(x,y,z) = e^{-2\pi i(lx+my+nz)}$$

There are many of these:

$$\vec{F}_{lmn}(x,y,z) = \frac{1}{-2\pi i}\begin{pmatrix}\frac{1}{l}\\0\\0\end{pmatrix}e^{-2\pi i(lx+my+nz)} \qquad \vec{F}_{lmn}(x,y,z) = \frac{1}{-2\pi i}\begin{pmatrix}0\\\frac{1}{m}\\0\end{pmatrix}e^{-2\pi i(lx+my+nz)} \qquad \vec{F}_{lmn}(x,y,z) = \frac{1}{-2\pi i}\begin{pmatrix}0\\0\\\frac{1}{n}\end{pmatrix}e^{-2\pi i(lx+my+nz)}$$

$$\vec{F}_{lmn}(x,y,z) = \frac{1}{-6\pi i}\begin{pmatrix}\frac{1}{l}\\\frac{1}{m}\\\frac{1}{n}\end{pmatrix}e^{-2\pi i(lx+my+nz)} \qquad \vec{F}_{lmn}(x,y,z) = \frac{1}{-2\pi i(l+m+n)}\begin{pmatrix}1\\1\\1\end{pmatrix}e^{-2\pi i(lx+my+nz)}$$

Implementation

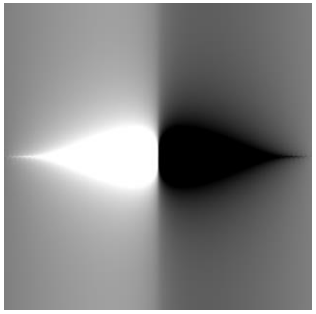
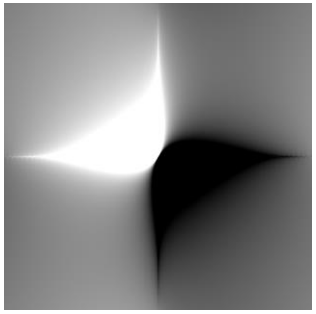
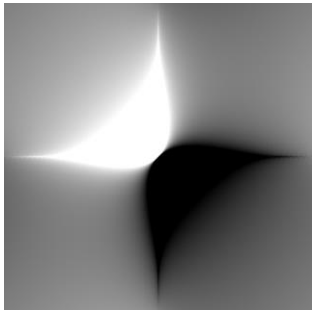
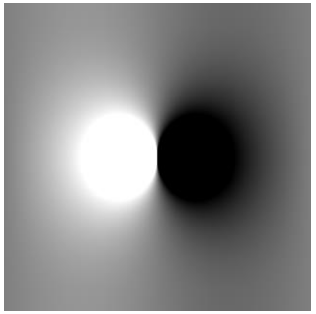
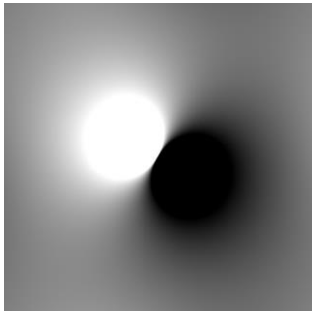
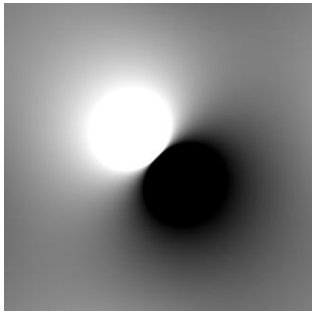
To implement, we must find vector fields whose divergences are the complex exponentials:

$$\operatorname{div}\left(\vec{F}_{lmn}\right)(x,y,z) = e^{-2\pi i(lx+my+nz)}$$

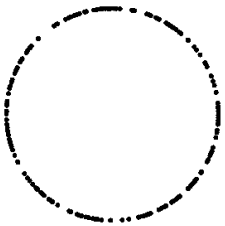
But there is only one vector field that commutes with both translation and rotation:

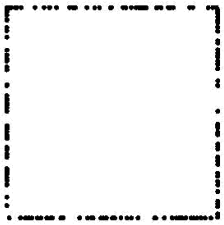
$$\vec{F}_{lmn}(x,y,z) = \frac{1}{-2\pi i\left(l^2+m^2+n^2\right)}\begin{pmatrix} l \\ m \\ n \end{pmatrix} e^{-2\pi i(lx+my+nz)}$$

Implementation

	 0°	 30°	 45°
Does not Commute: $\vec{F}_{lmn}(x,y,z) = \frac{1}{-2\pi i(l+m+n)} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} e^{-2\pi i(lx+my+nz)}$			
Commutes: $\vec{F}_{lmn}(x,y,z) = \frac{1}{-2\pi i(l^2+m^2+n^2)} \begin{pmatrix} l \\ m \\ n \end{pmatrix} e^{-2\pi i(lx+my+nz)}$			

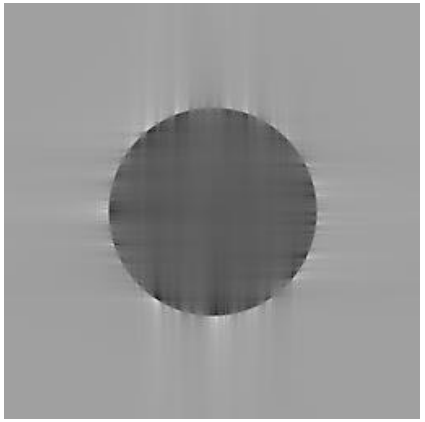
Implementation

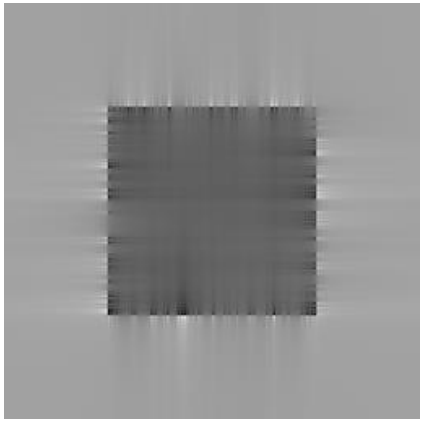




Does not Commute:

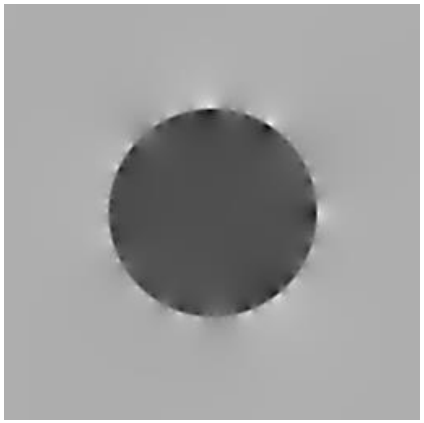
$$\vec{F}_{lmn}(x,y,z) = \frac{1}{-2\pi i(l+m+n)} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} e^{-2\pi i(lx+my+nz)}$$

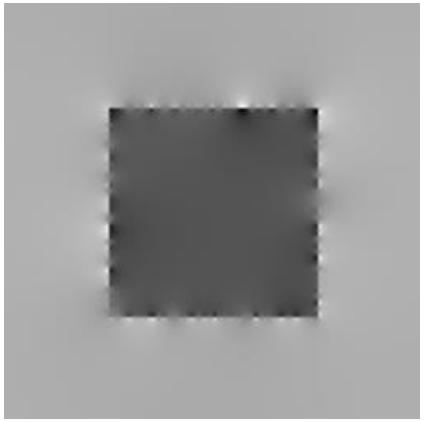




Commutes:

$$\vec{F}_{lmn}(x,y,z) = \frac{1}{-2\pi i(l^2+m^2+n^2)} \begin{pmatrix} l \\ m \\ n \end{pmatrix} e^{-2\pi i(lx+my+nz)}$$





Efficient Implementation

Explicitly summing over all the points to find each Fourier coefficient would be too slow:

$$\hat{\chi}_V(l,m,n) = \sum_{j=1}^{k} \left\langle \vec{F}_{lmn}(p_j), n_j \right\rangle$$

Efficient Implementation

Explicitly summing over all the points to find each Fourier coefficient would be too slow:

$$\hat{\chi}_V(l,m,n) = \sum_{j=1}^{k} \left\langle \vec{F}_{lmn}(p_j), n_j \right\rangle$$

However, because the filter commutes with translation this can be reduced to a convolution:

$O(N^5) \rightarrow O(N^3 \log N)$.

Properties

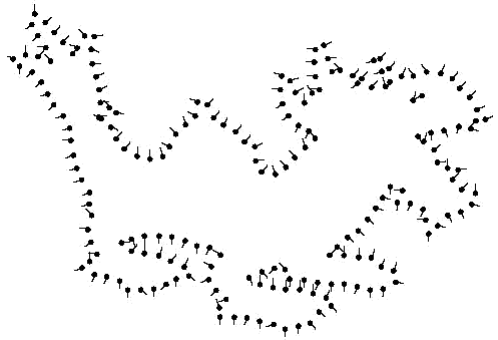
Advantages:

- Mathematical Correctness:
  - For a sufficiently dense/uniform sampling, the indicator function is guaranteed to be accurate

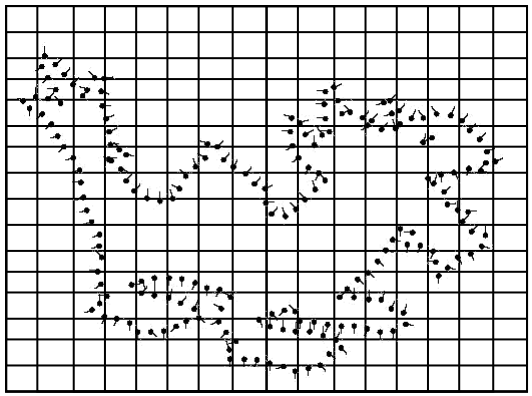
Properties

<u>Advantages</u>:

- Computational Simplicity:
  - Splat
  - Convolve
  - Extract



$\{p_i, n_i\}$



$\vec{N}(p)$

$\chi(p) = \vec{N}(p) * \vec{F}(p)$

$S = \{p \mid \chi(p) = 0.5\}$

Properties

Disadvantages

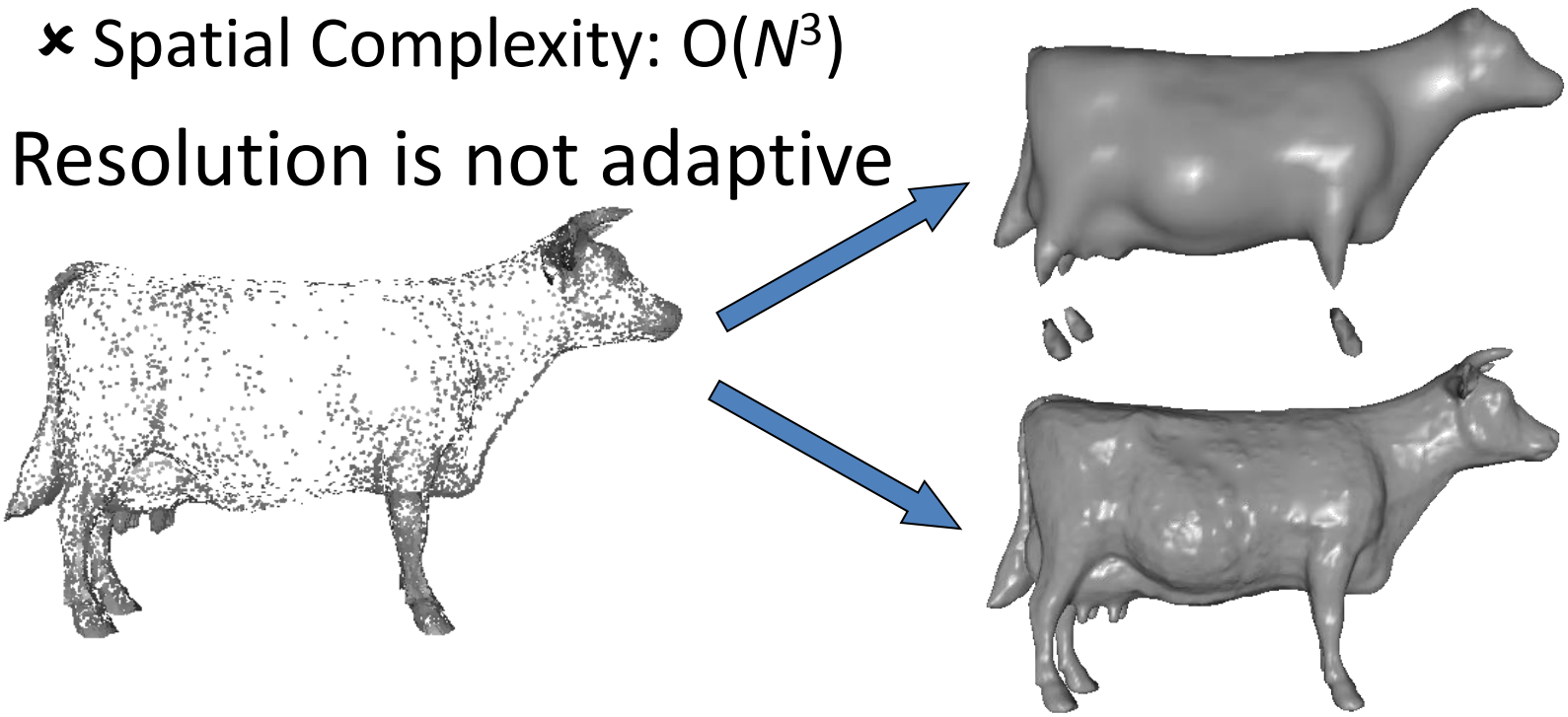
- For an $O(N^2)$ reconstruction:
  - ✘ Temporal Complexity: $O(N^3 \log N)$
  - ✘ Spatial Complexity: $O(N^3)$

  On a machine with 4GB of RAM, reconstruction resolutions will be limited to 512^3 voxel grids.

Properties

<u>Disadvantages</u>

- For an O(N^2) reconstruction:
  - ✗ Temporal Complexity: O($N^3 \log N$)
  - ✗ Spatial Complexity: O(N^3)
- ✗ Resolution is not adaptive



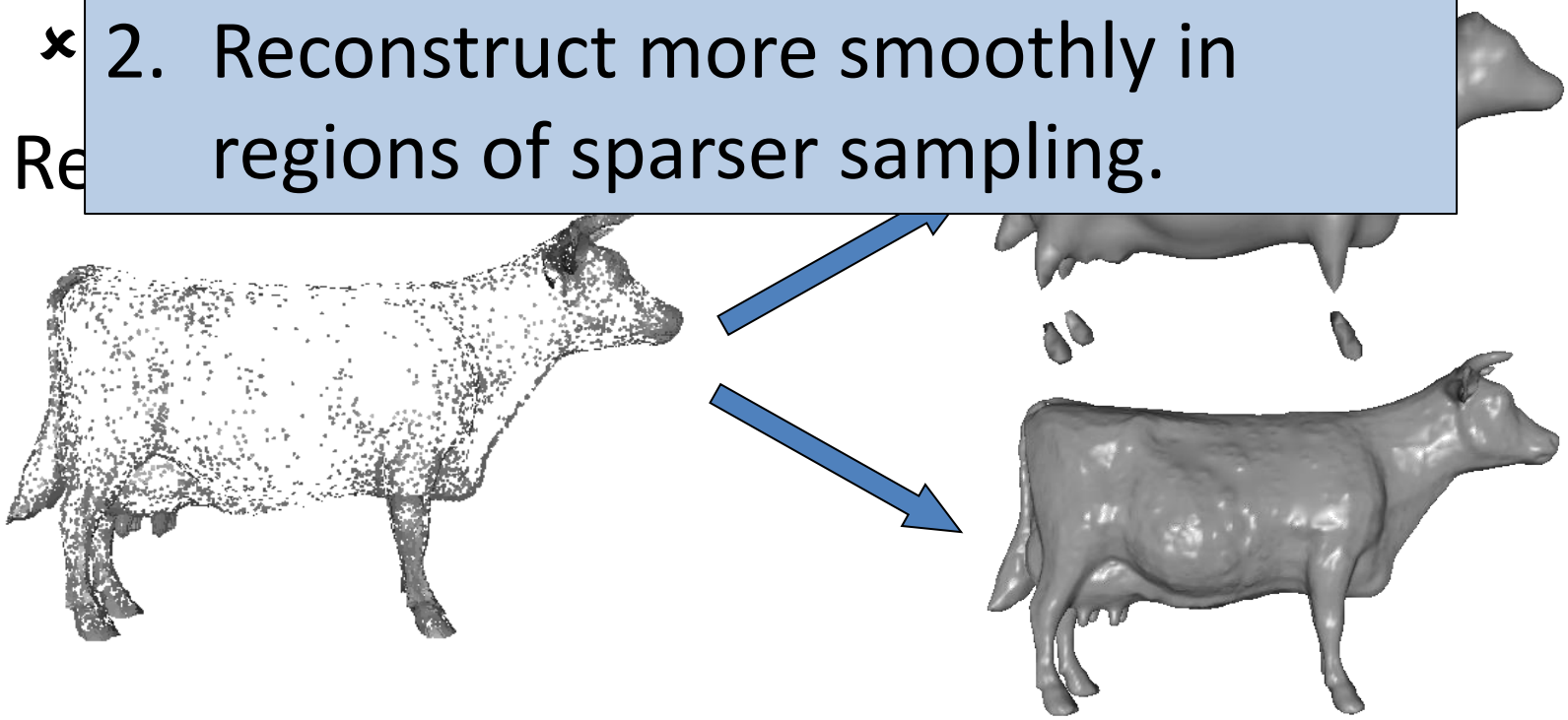
Properties

Disadvantages

- Fo
  - ×
  - ×
- × Re

To adapt the to the sampling density:

1. Weight the samples' contribution.
2. Reconstruct more smoothly in regions of sparser sampling.



Properties

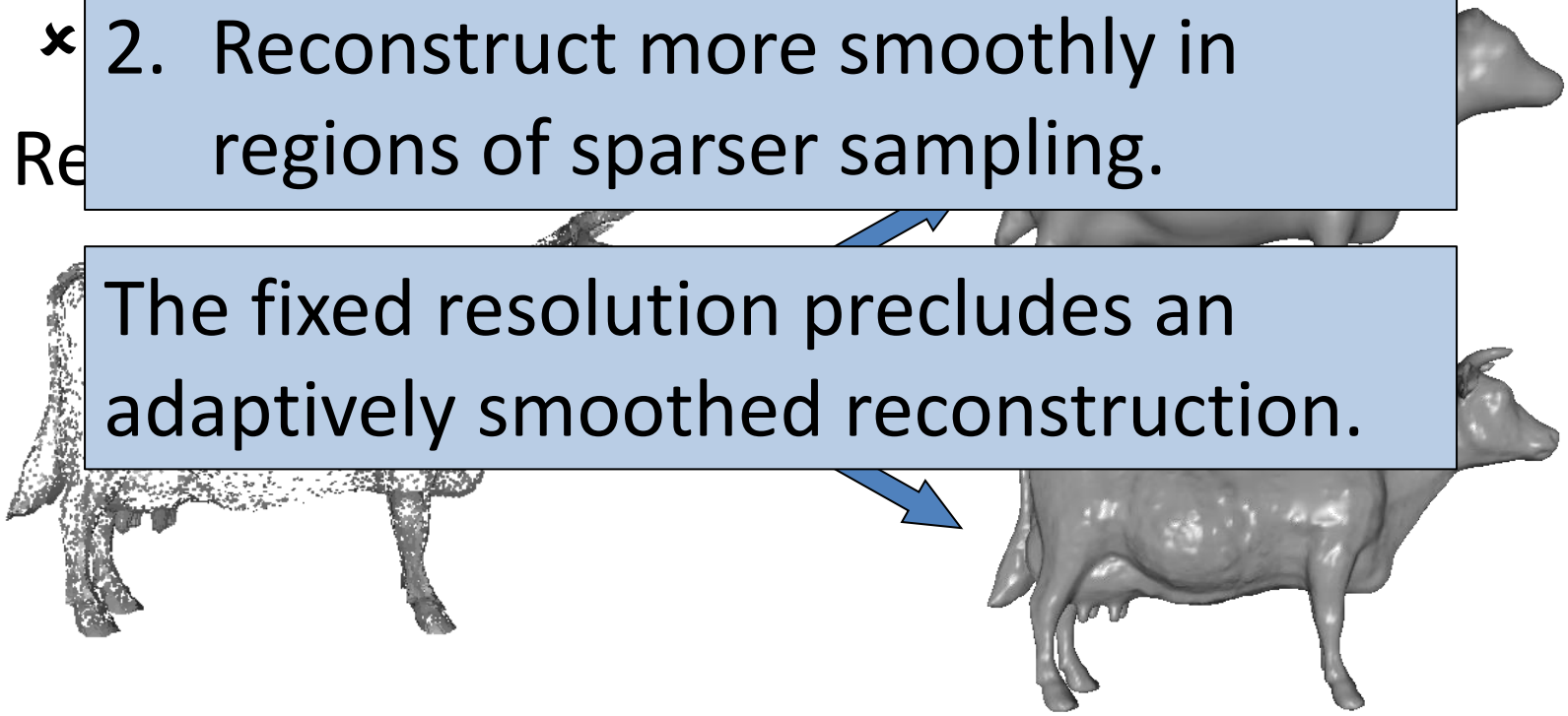
Disadvantages

- Fo
  - ×
  - ×
- × Re

To adapt the to the sampling density:

1. Weight the samples' contribution.
2. Reconstruct more smoothly in regions of sparser sampling.

The fixed resolution precludes an adaptively smoothed reconstruction.



Outline

- Introduction
- Related Work
- Two Approaches
  - FFT: What can we compute?
  - Poisson: What are we representing?
- Results

Motivation

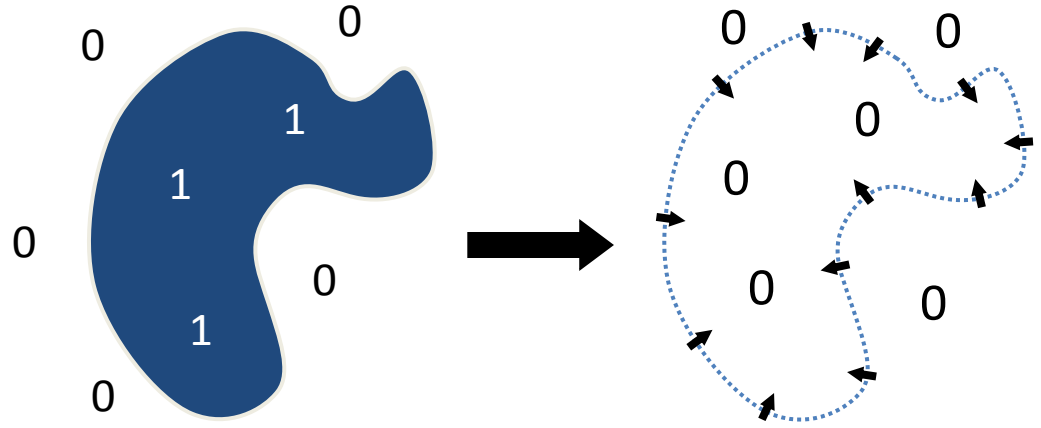
What information about the indicator function do a set of oriented points represent?

Motivation

What information about the indicator function do a set of oriented points represent?

Indicator Function Gradient:

Because χ_D is piecewise constant, its gradient will be zero almost everywhere…



Indicator function χ_D

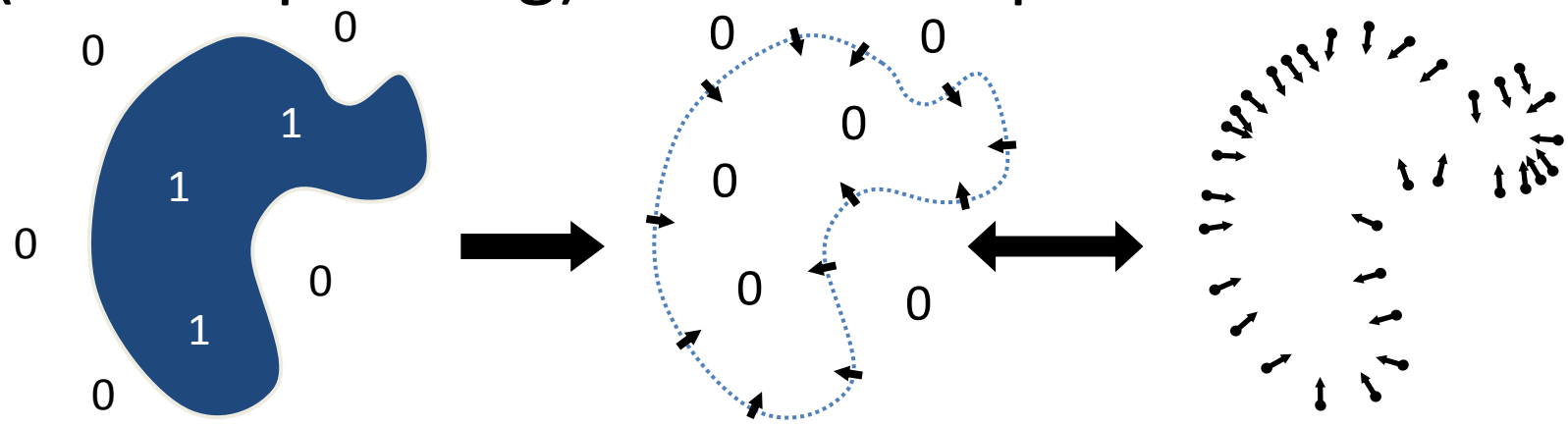
Indicator gradient $\nabla\chi_D$

Motivation

What information about the indicator function do a set of oriented points represent?

Indicator Function Gradient:

… Which looks distinctively like the oriented (inward-pointing) surface samples.



Indicator function χ_D

Indicator gradient $\nabla\chi_D$

Oriented points $\{p_i, n_i\}$

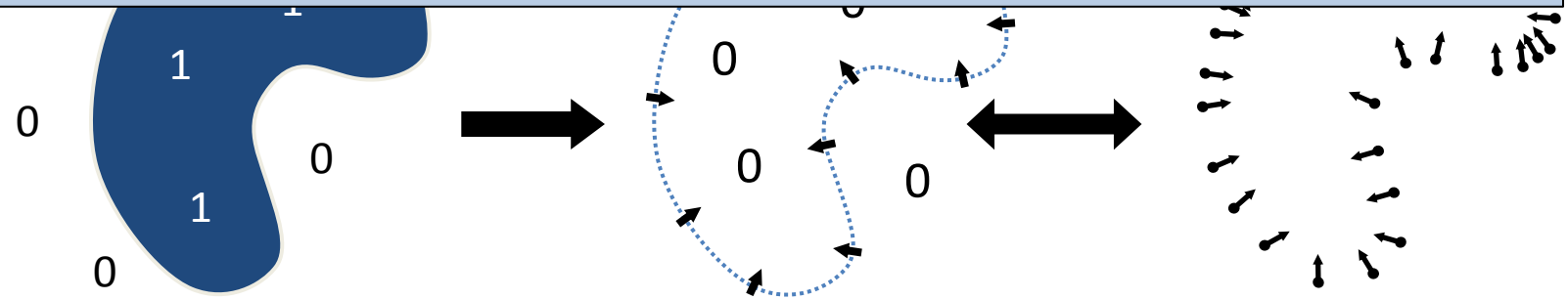
Motivation

W
d
In

The gradient of the smoothed indicator function is equal to the vector field obtained by smoothing the surface normal field:

$$\nabla(\chi_D * F)(q) = \int_{\partial D} F(q-p)\vec{N}_{\partial D}(p)dp$$

$$\approx \frac{A}{n}\sum_{i=1}^{n} F(q-p_i)n_i$$



Indicator function χ_D

Indicator gradient $\nabla\chi_D$

Oriented points $\{p_i, n_i\}$

Approach

We reconstruct by solving for the indicator function whose gradient is most similar to the vector field $\vec{N}$ represented by the samples:

$$\chi_D = \arg\min_f \left\| \nabla f - \vec{N} \right\|$$

Approach

We reconstruct by solving for the indicator function whose gradient is most similar to the vector field $\vec{N}$ represented by the samples:

$$\chi_D = \arg\min_f \left\| \nabla f - \vec{N} \right\|$$

This can be done by:

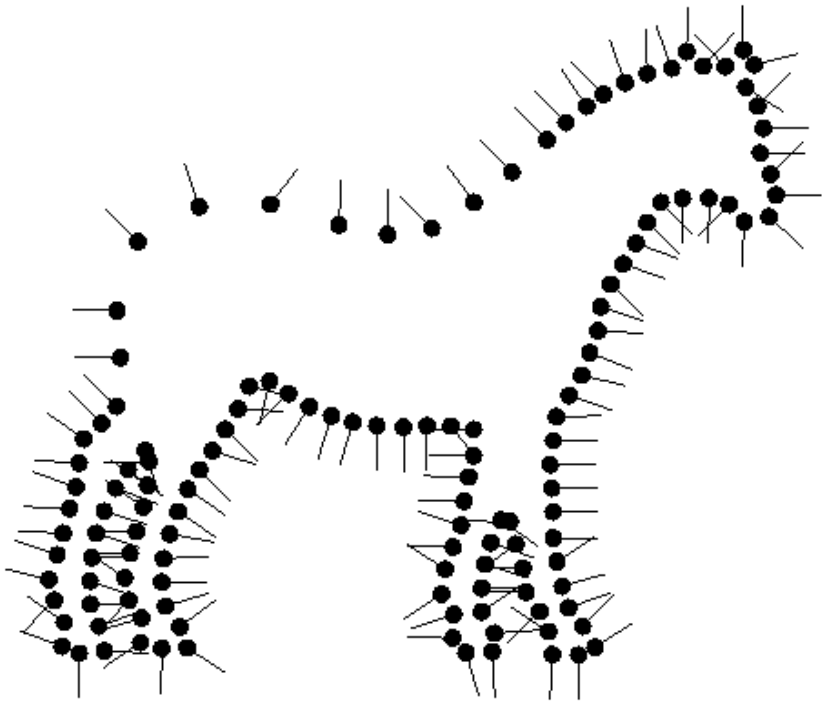
1. Computing the divergence of $\vec{N}$
2. Solving the Poisson equation:

$$\chi_D = \Delta^{-1}\left(\nabla \cdot \vec{N}\right)$$

Implementation

Given the Points:

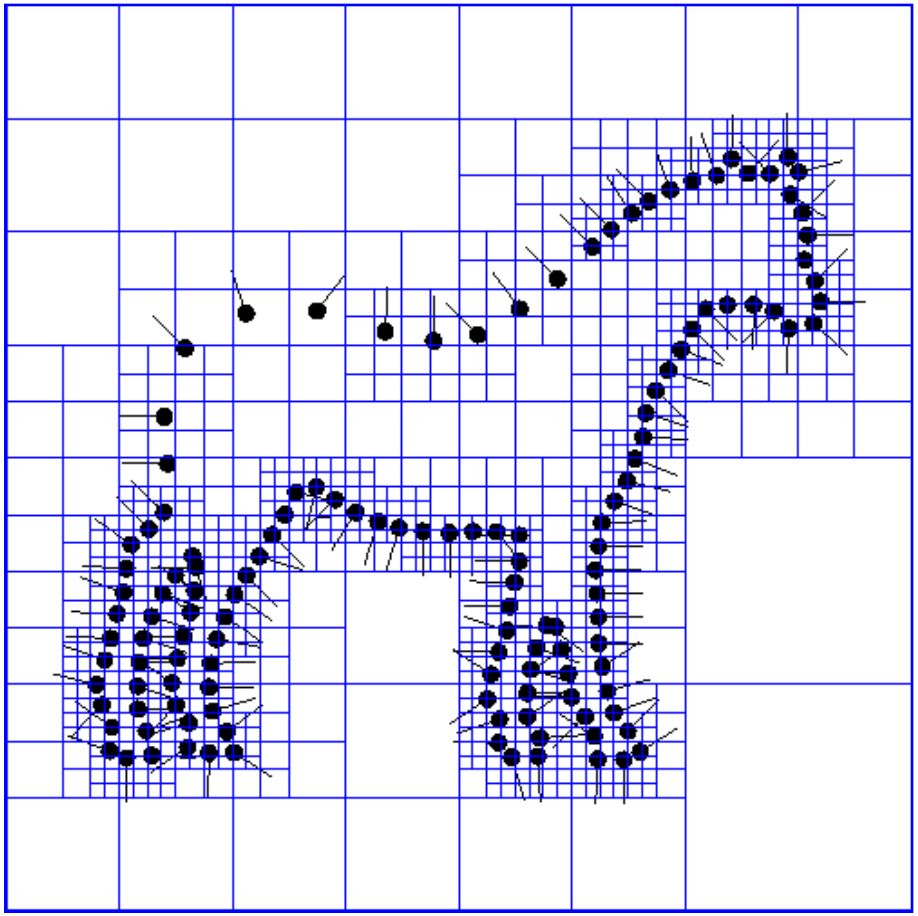
- Set up an octree
- Compute vector field
- Compute indicator function
- Extract iso-surface



Implementation: Adapted Octree

Given the Points:

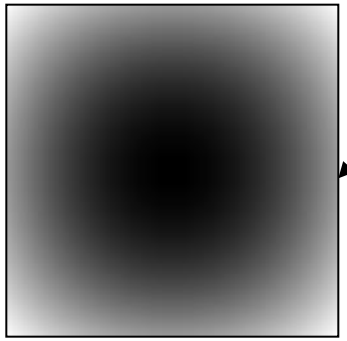
- Set up an octree
- Compute vector field
- Compute indicator function
- Extract iso-surface

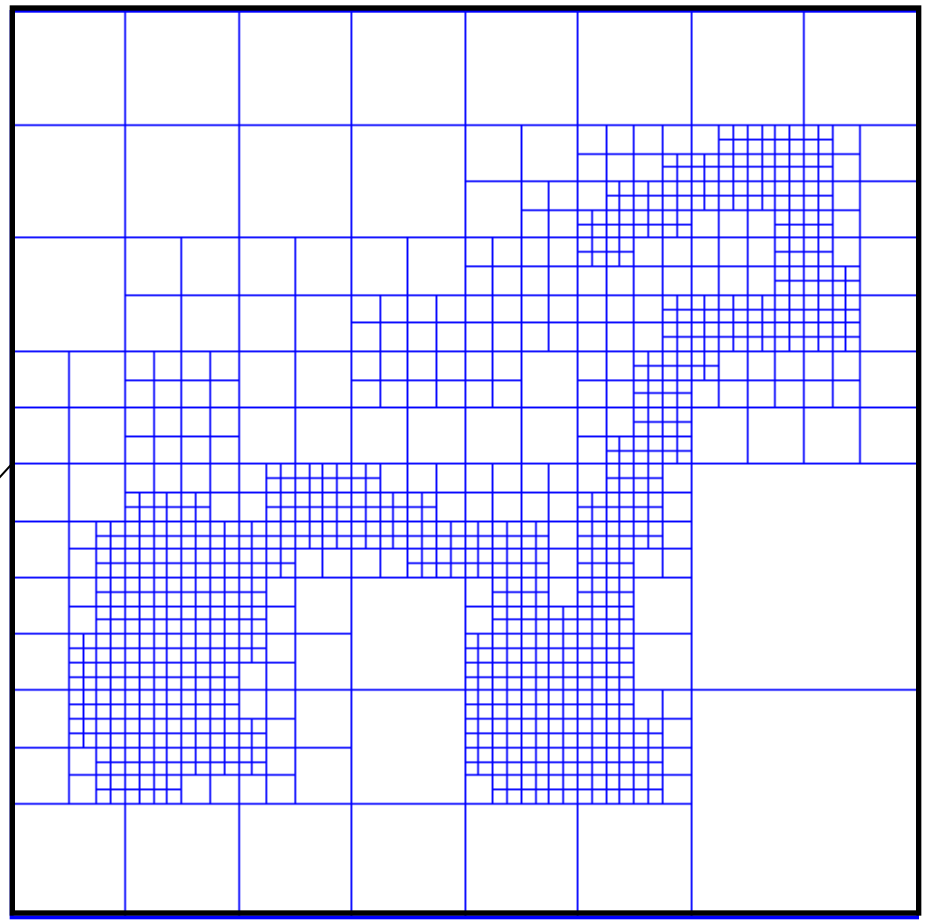


Implementation: Vector Field

Given the Points:

- Set up an octree
- Compute vector field
  - Define a function space
  - Splat the samples
- Compute indicator function
- Extract iso-surface

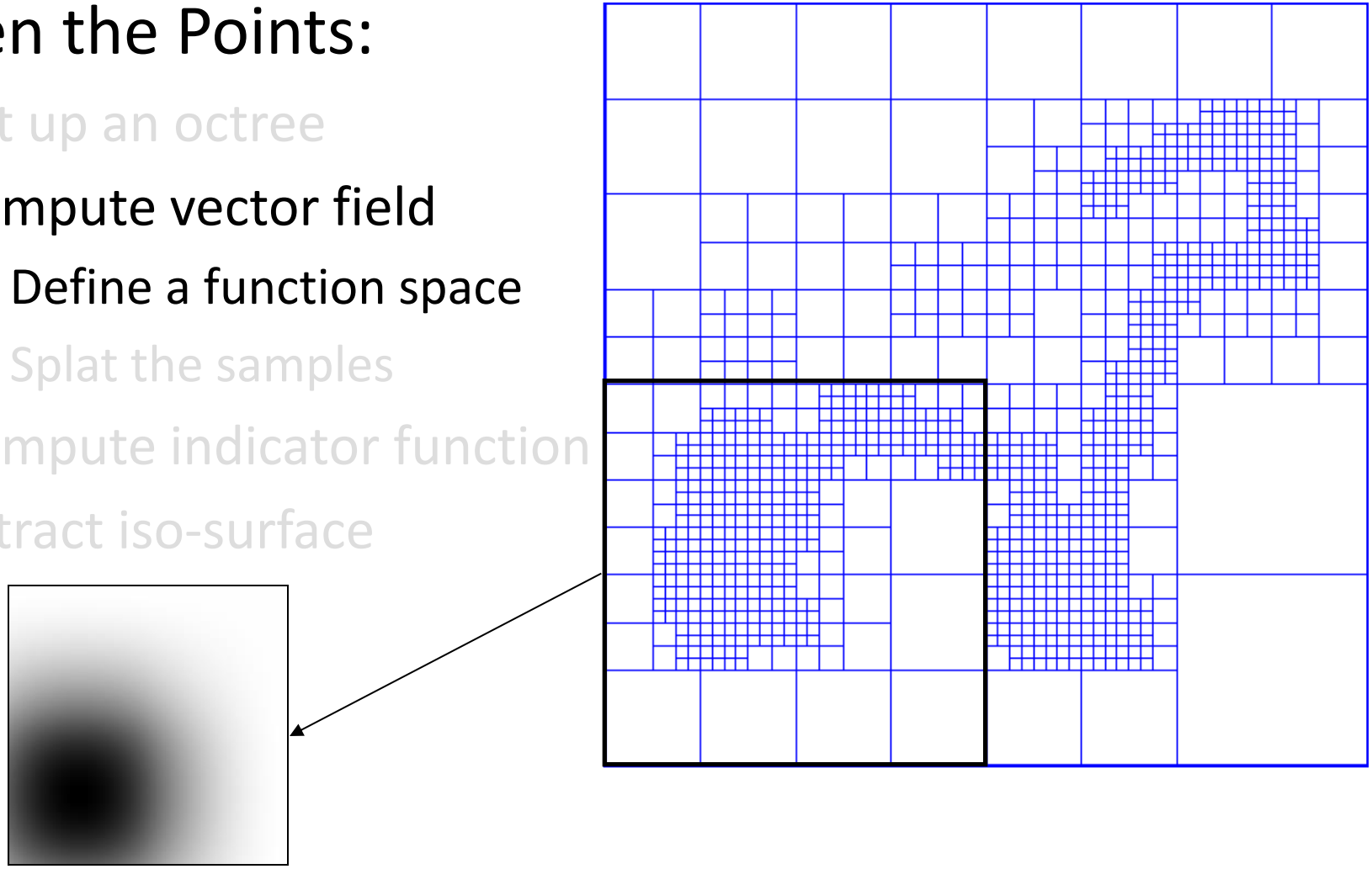




Implementation: Vector Field

Given the Points:

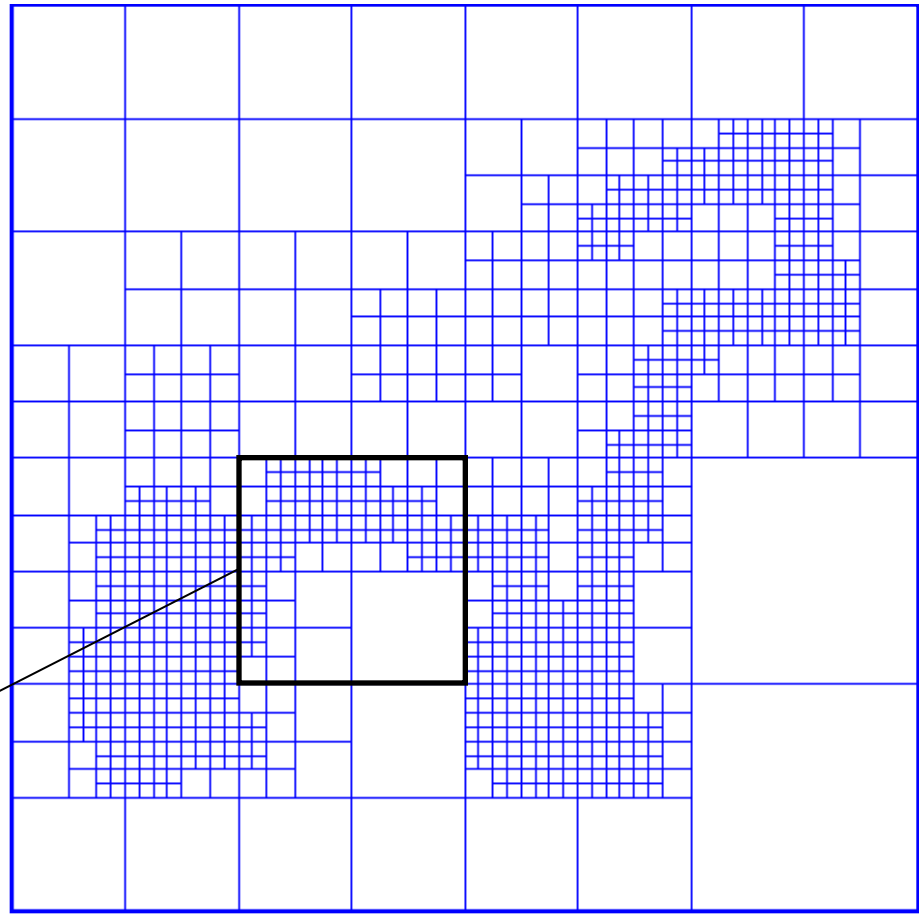
- Set up an octree
- Compute vector field
  - Define a function space
  - Splat the samples
- Compute indicator function
- Extract iso-surface

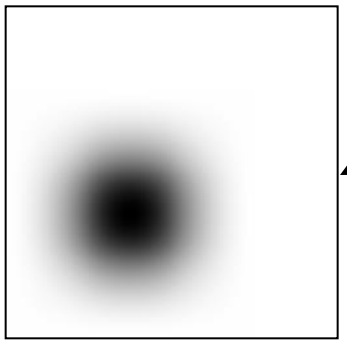


Implementation: Vector Field

Given the Points:

- Set up an octree
- Compute vector field
  - Define a function space
  - Splat the samples
- Compute indicator function
- Extract iso-surface

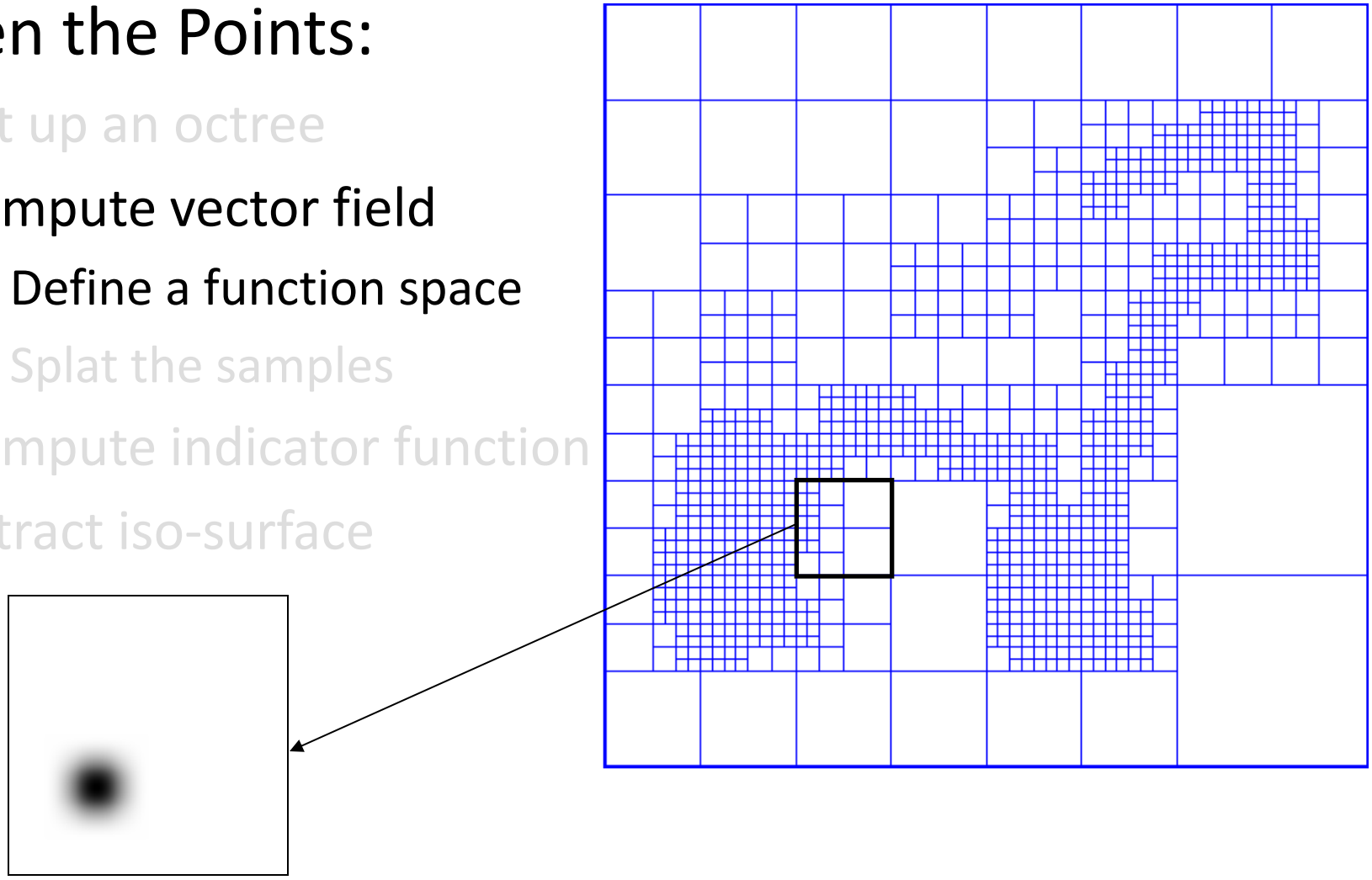




Implementation: Vector Field

Given the Points:

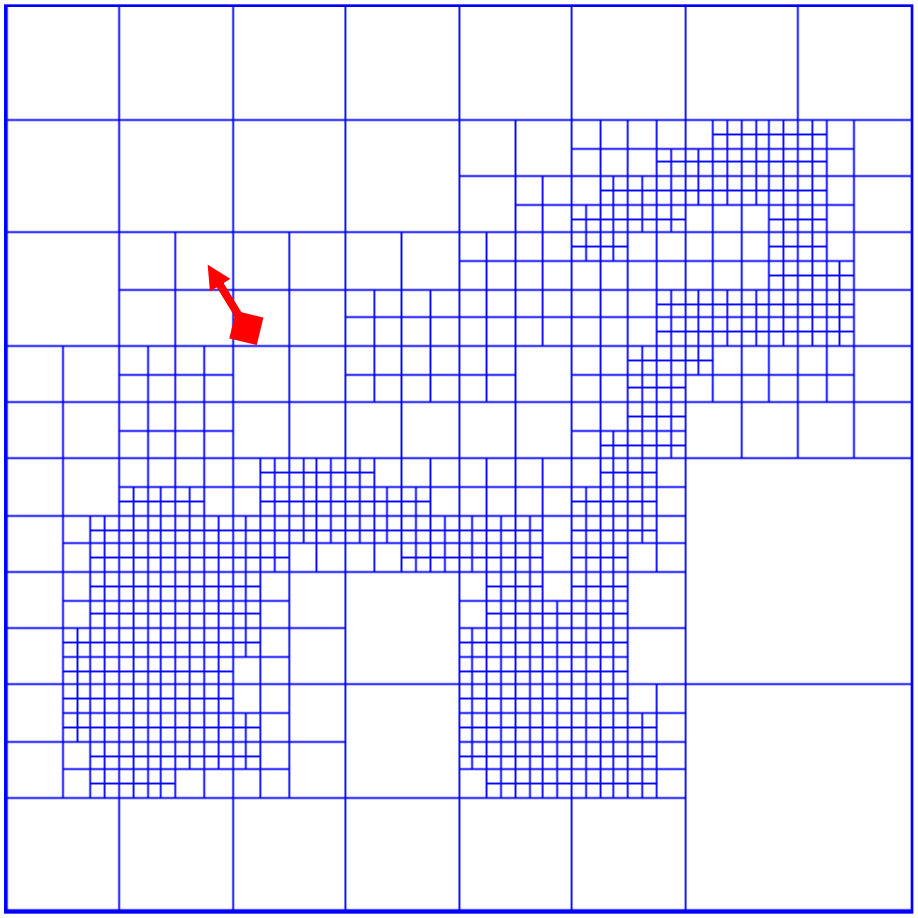
- Set up an octree
- Compute vector field
  - Define a function space
  - Splat the samples
- Compute indicator function
- Extract iso-surface



Implementation: Vector Field

Given the Points:

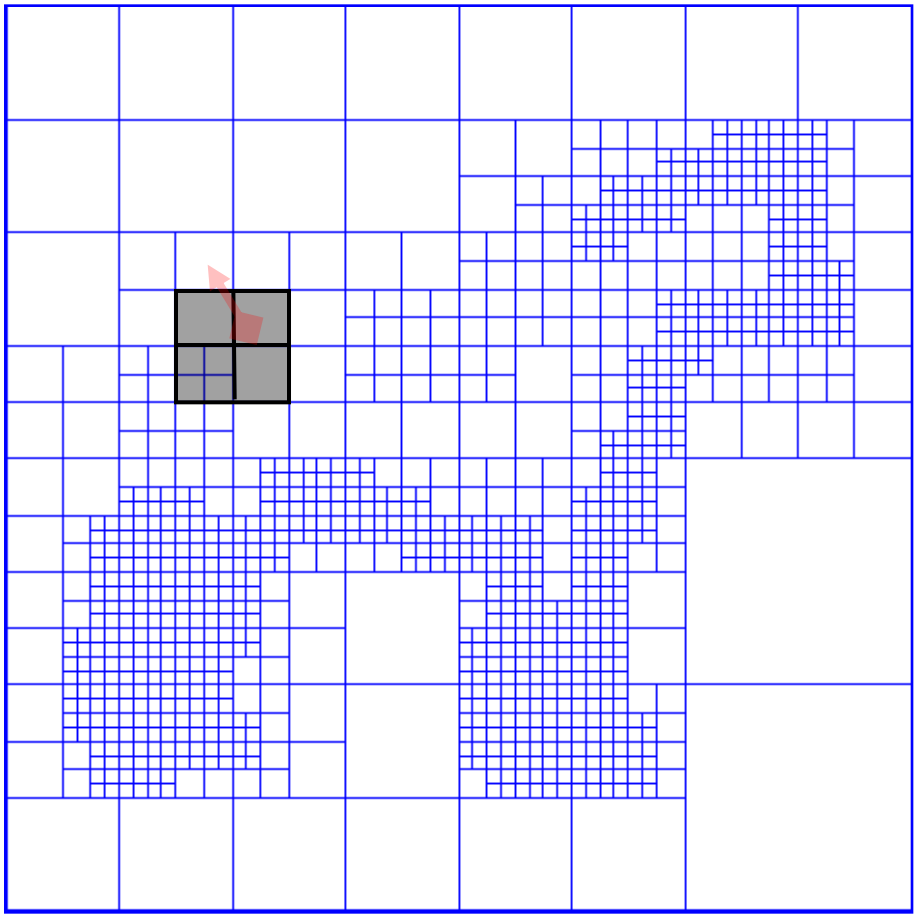
- Set up an octree
- Compute vector field
  - Define a function basis
  - Splat the samples
- Compute indicator function
- Extract iso-surface



Implementation: Vector Field

Given the Points:

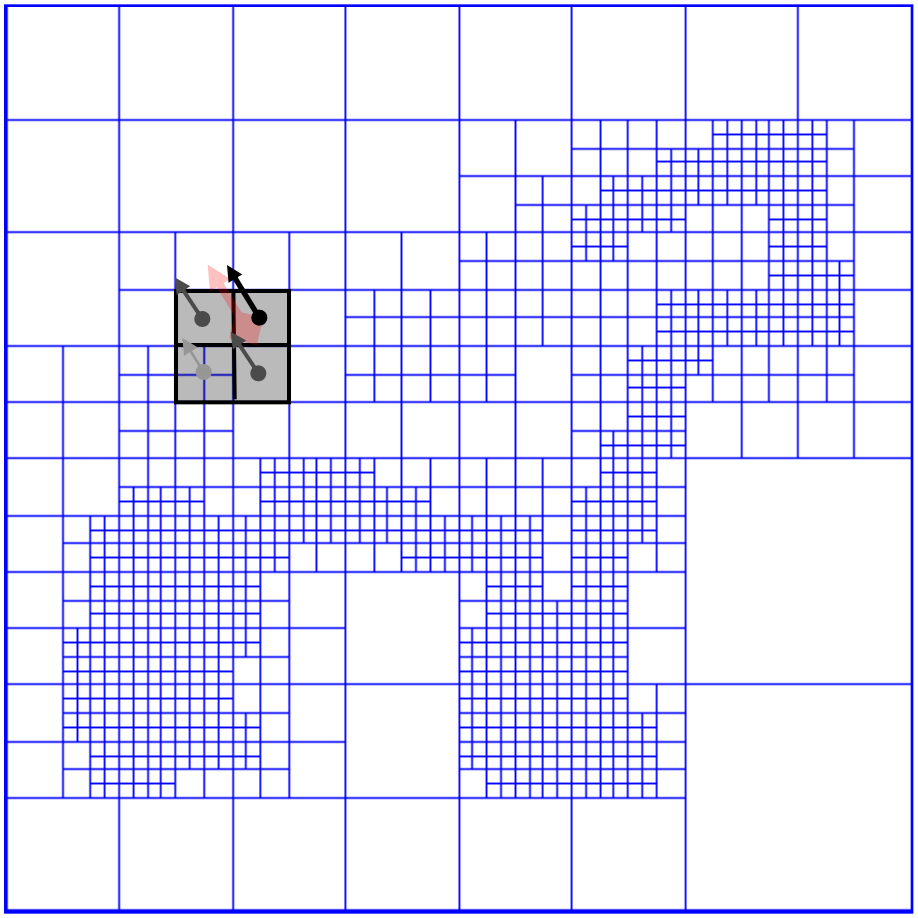
- Set up an octree
- Compute vector field
  - Define a function basis
  - Splat the samples
- Compute indicator function
- Extract iso-surface



Implementation: Vector Field

Given the Points:

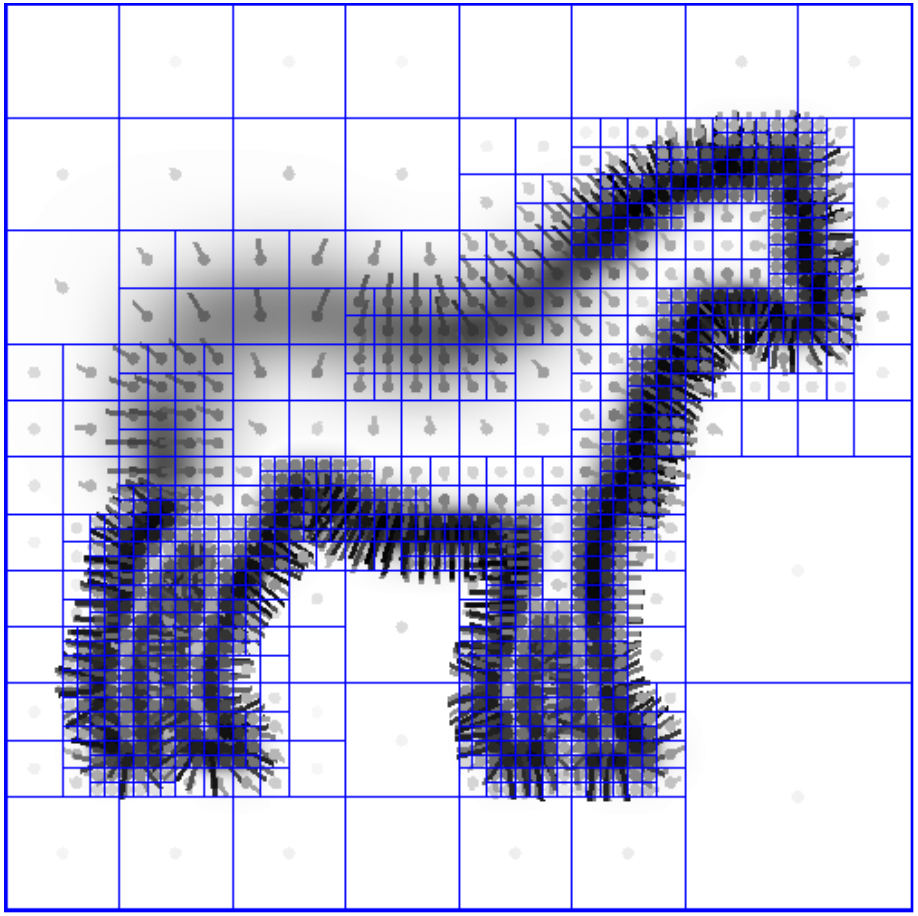
- Set up an octree
- Compute vector field
  - Define a function basis
  - Splat the samples
- Compute indicator function
- Extract iso-surface



Implementation: Vector Field

Given the Points:

- Set up an octree
- Compute vector field
  - Define a function basis
  - Splat the samples
- Compute indicator function
- Extract iso-surface



Implementation: Indicator Function

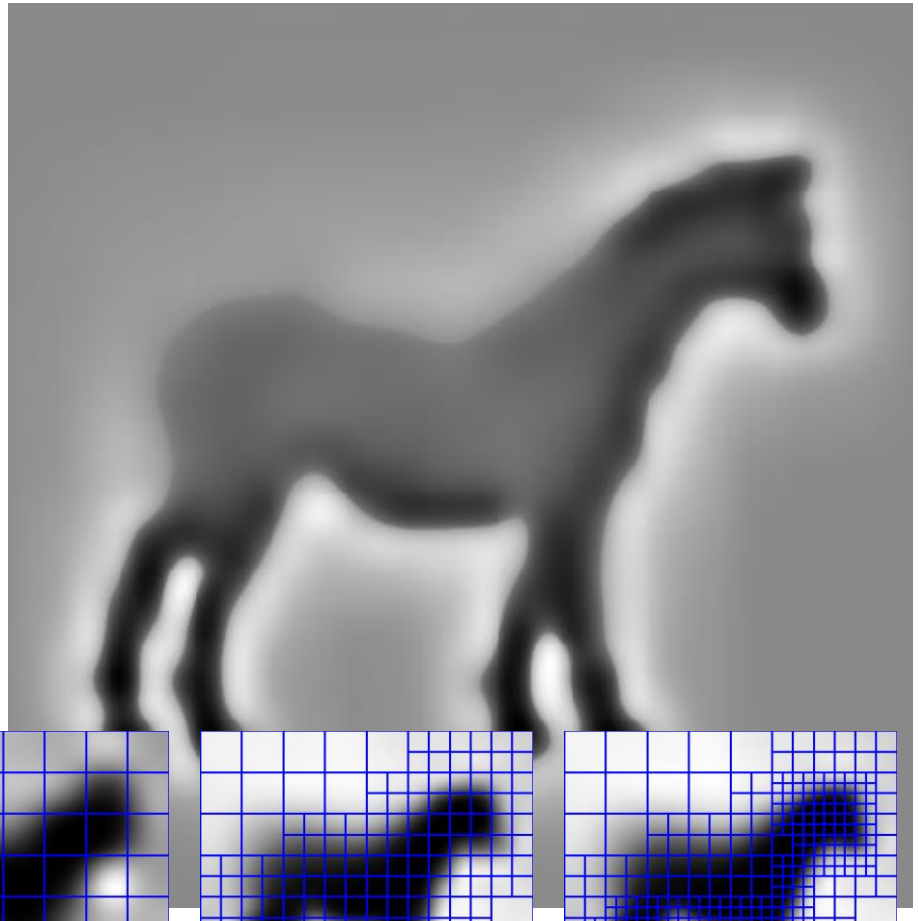
Given the Points:

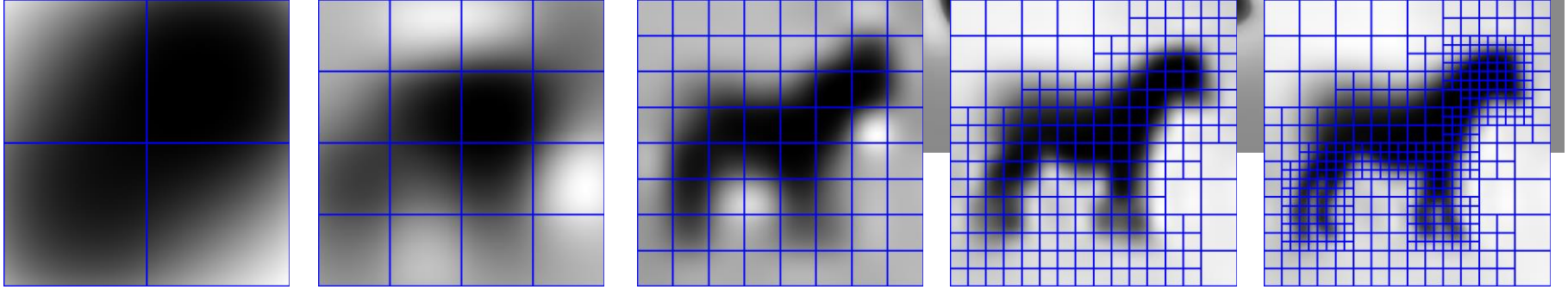
- Set up an octree
- Compute vector field
- Compute indicator function
  - Compute divergence
  - Solve Poisson equation
- Extract iso-surface

Implementation: Indicator Function

Given the Points:

- Set up an octree
- Compute vector field
- Compute indicator function
  - Compute divergence
  - Solve Poisson equation
- Extract iso-surface

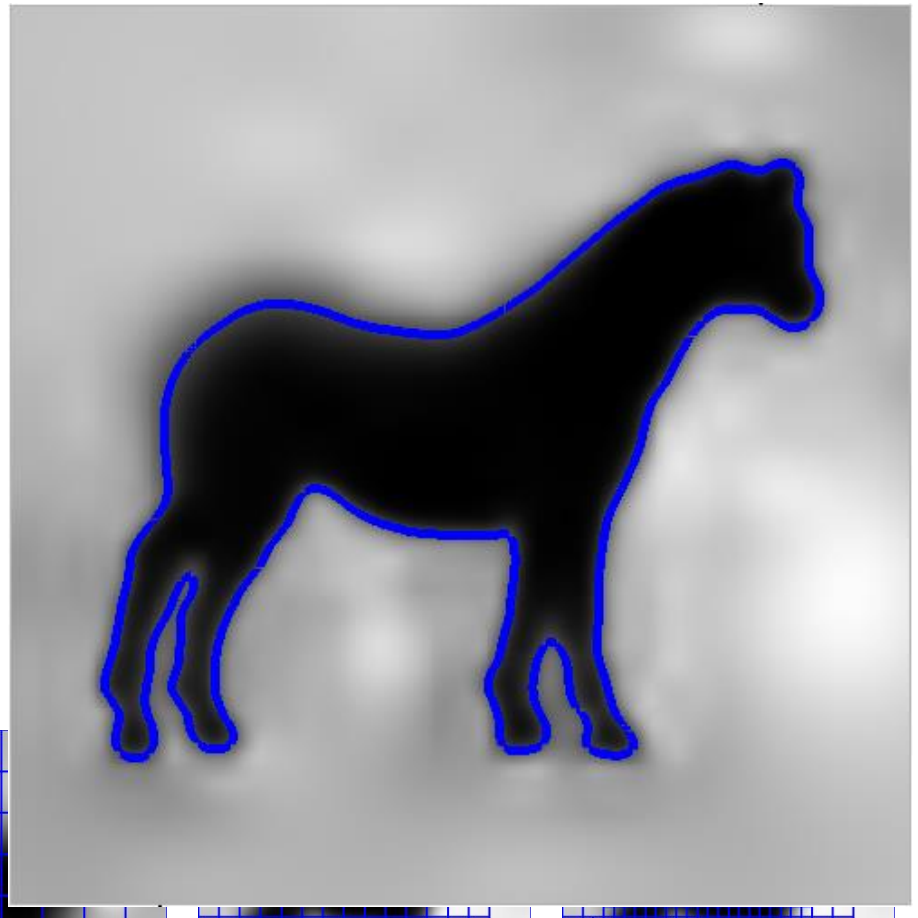


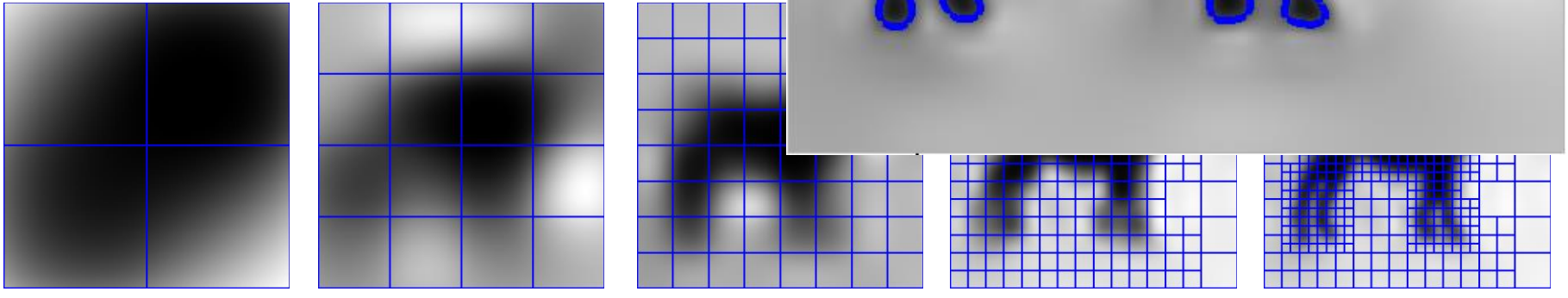


Implementation: Surface Extraction

Given the Points:

- Set up an octree
- Compute vector field
- Compute indicator function
- Extract iso-surface

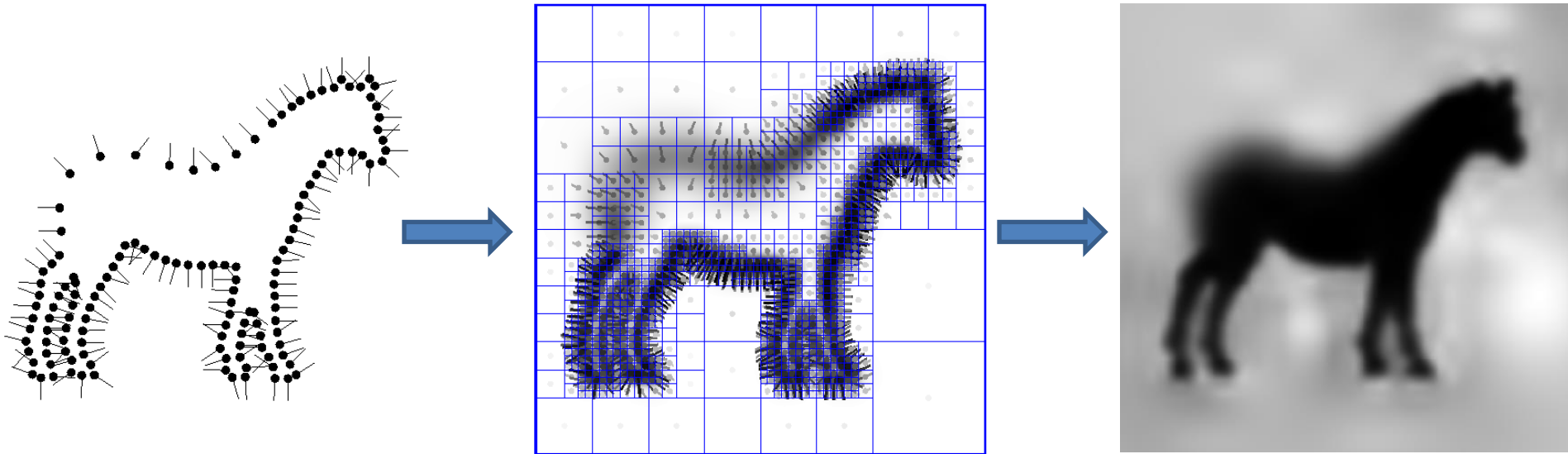




Properties

<u>Advantages</u>:

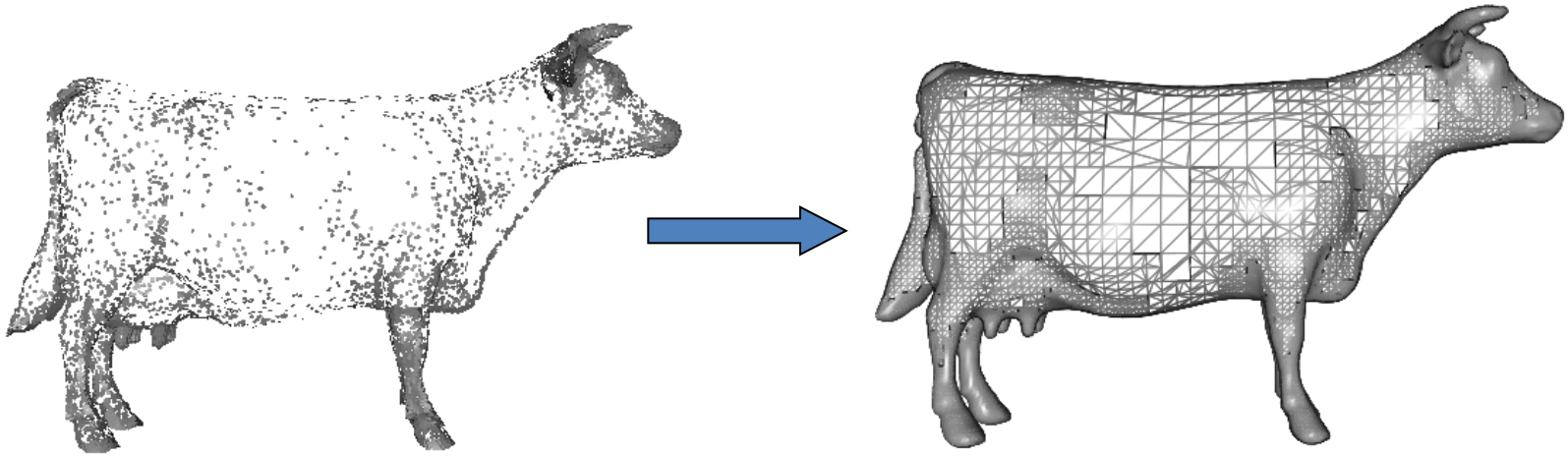
- Solving over an octree, reduces both the space and time complexity of the reconstruction algorithm:
  - Space: $O(N^3) \rightarrow O(N^2)$
  - Time: $O(N^3 \log N) \rightarrow O(N^2)$



Properties

Advantages:

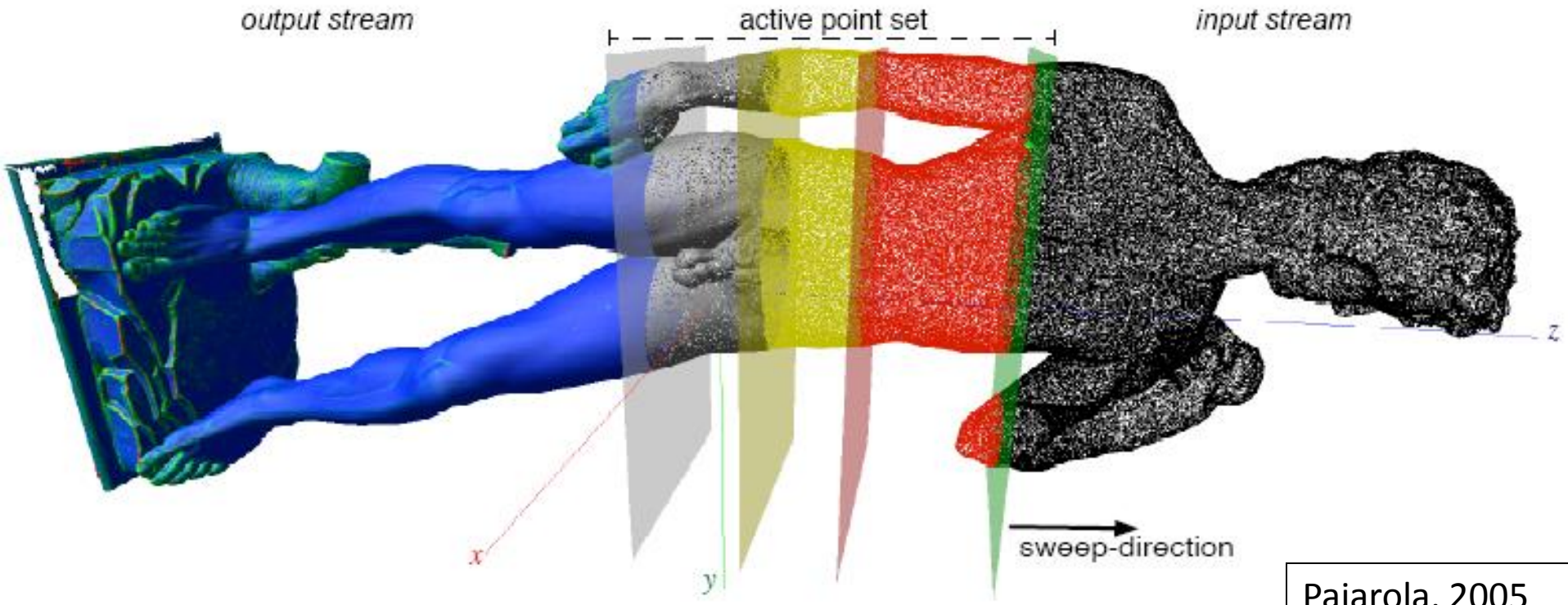
- We can adapt the octree to the sampling density to better handle non-uniform samples



Properties

Advantages:

– Sorting the octree nodes by *x*-index, we can design a streaming and parallel implementation.

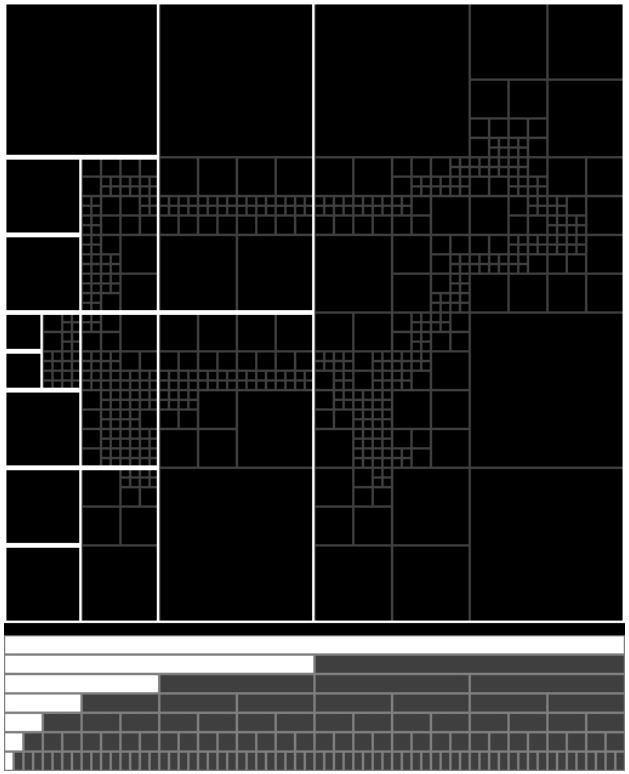


Pajarola, 2005

Properties

Advantages:

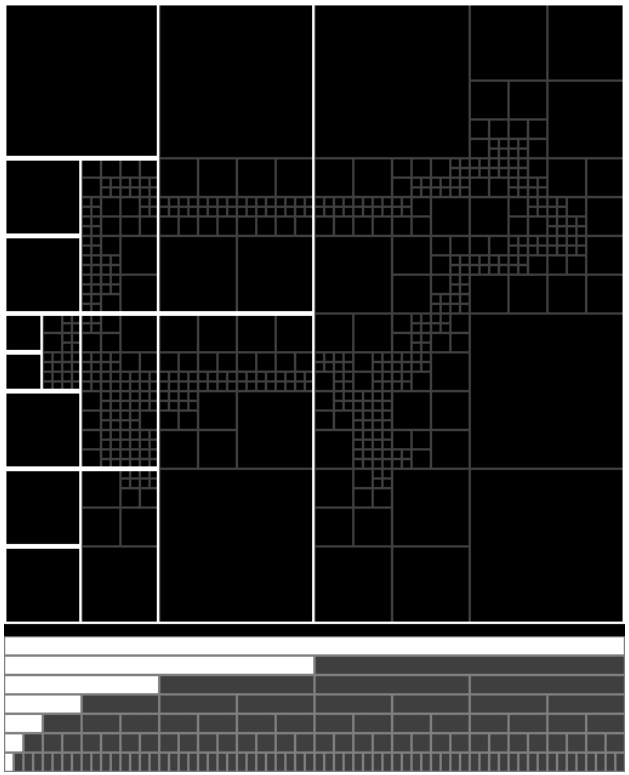
- Sorting the octree nodes by *x*-index, we can get a streaming and parallel implementation.

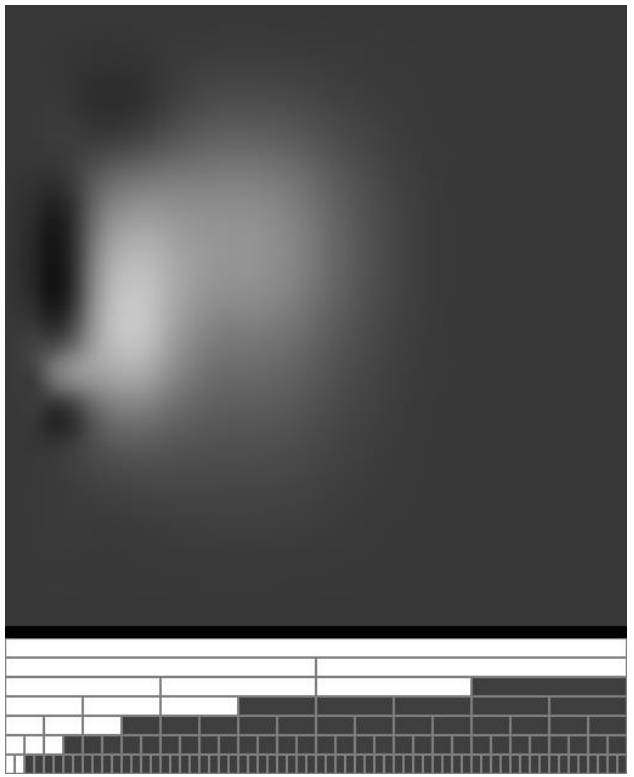


Properties

Advantages:

– Sorting the octree nodes by *x*-index, we can get a streaming and parallel implementation.

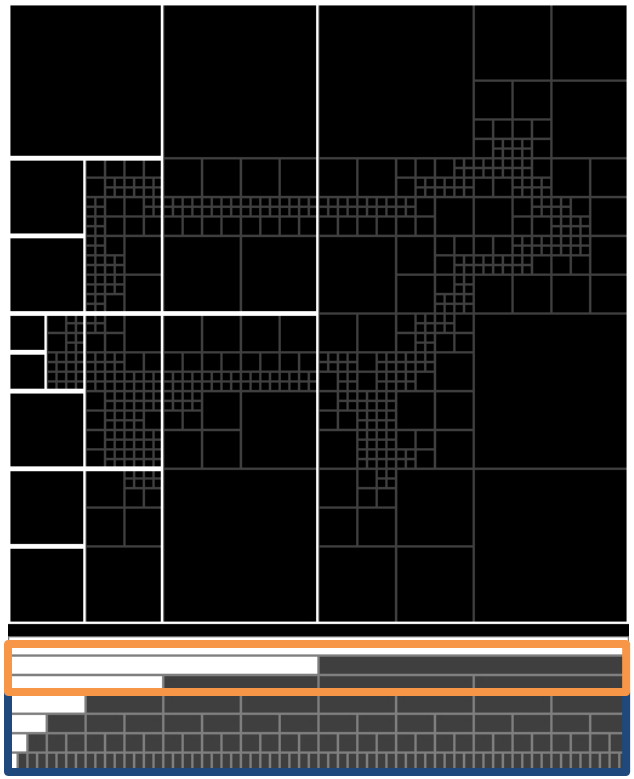


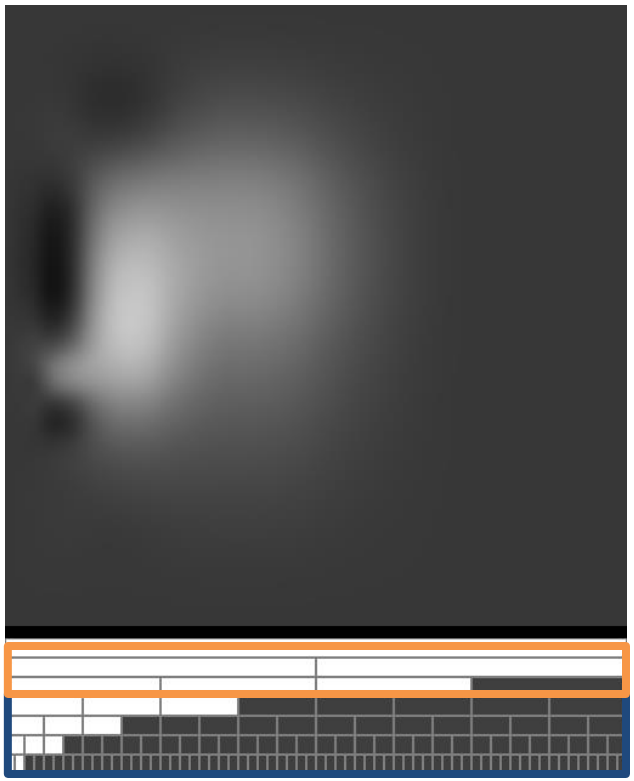


Properties

Advantages:

– Sorting the octree nodes by *x*-index, we can get a streaming and parallel implementation.

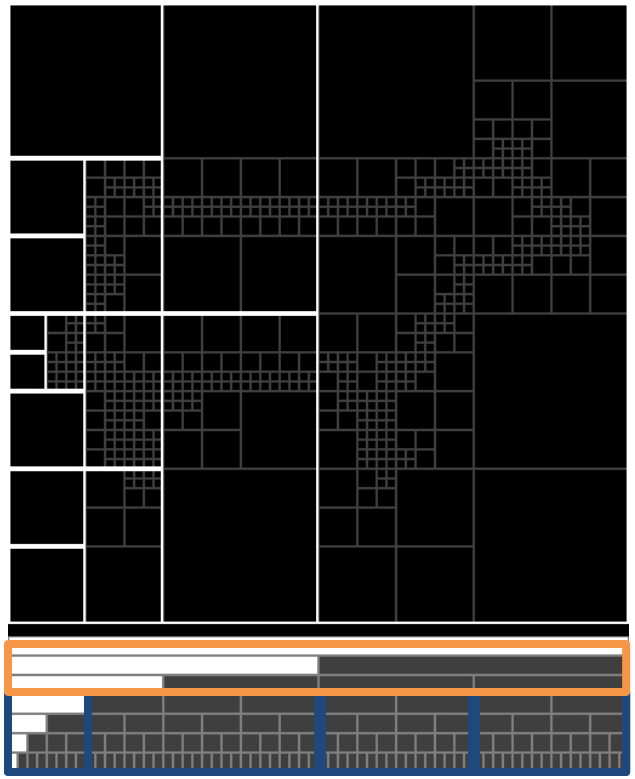




Properties

Advantages:

– Sorting the octree nodes by *x*-index, we can get a streaming and parallel implementation.



Properties

What is the difference between the FFT and Poisson solvers?

Properties

What is the difference between the FFT and Poisson solvers?

- The convolution kernel in the FFT approach is the operator that computes the divergence of a vector field and then applies the inverse of the Laplacian.

$$\vec{F}_{lmn}(x,y,z) = \frac{1}{-2\pi i\left(l^2+m^2+n^2\right)} \begin{pmatrix} l \\ m \\ n \end{pmatrix} e^{-2\pi i(lx+my+nz)}$$

Inverse Laplacian Divergence

Properties

What is the difference between the FFT and Poisson solvers?

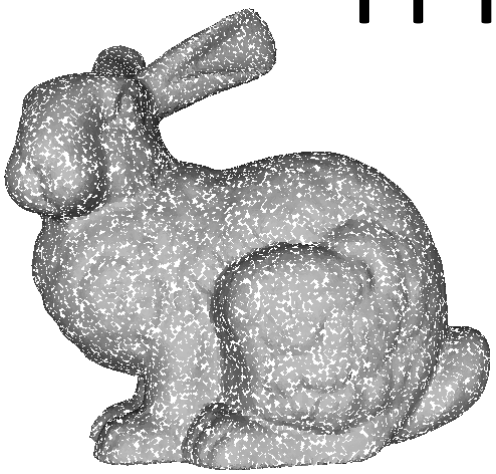
- Alternatively, the equivalence of the normals with the gradient of the indicator function derives from the Divergence Theorem.

$$\begin{aligned}\frac{\partial}{\partial x}(\chi_D * F)(q) &= \int_D \frac{\partial F}{\partial x}(q-p)dp \\ &= \int_D \nabla\cdot(F,0,0)(q-p)dp \\ &= \int_{\partial D}(F,0,0)(q-p)\vec{N}(p)dp \\ &= \int_{\partial D} F(q-p)\vec{N}_x(p)dp\end{aligned}$$

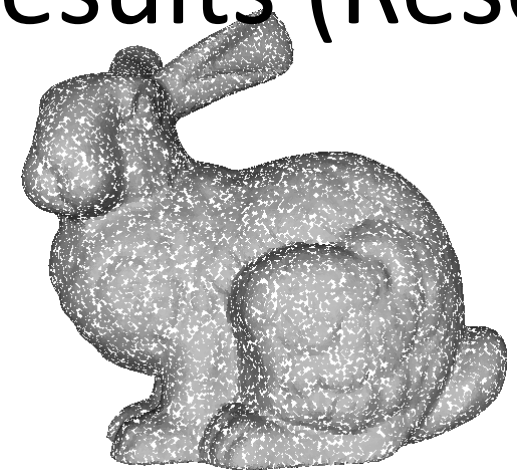
Outline

- Introduction
- Related Work
- Two Approaches
  - FFT: What can we compute?
  - Poisson: What are we representing?
- Results

FFT Results (Resolution)

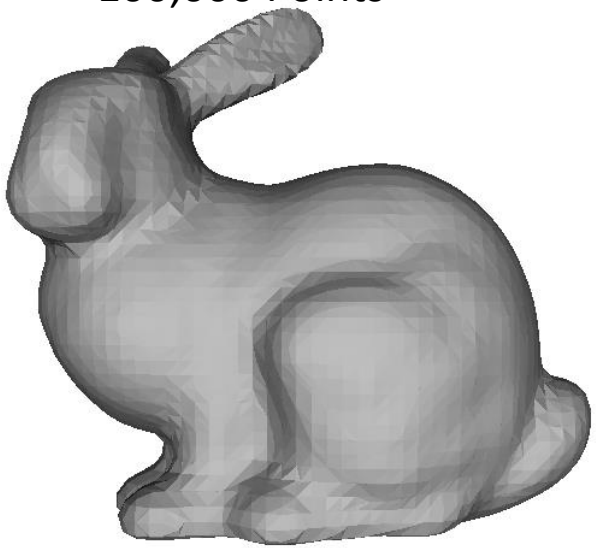


100,000 Points

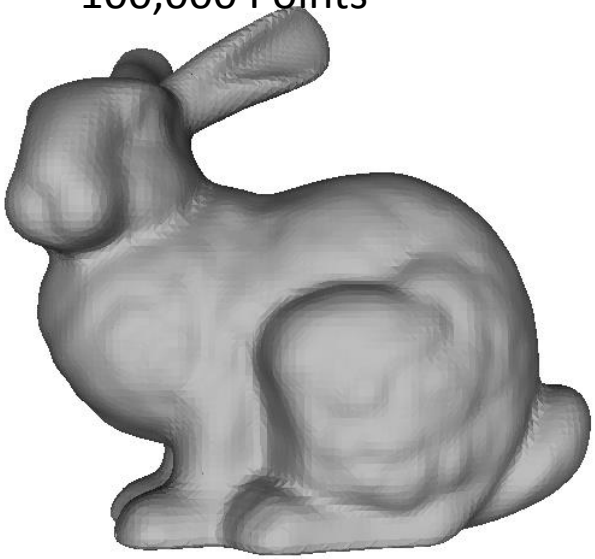


100,000 Points

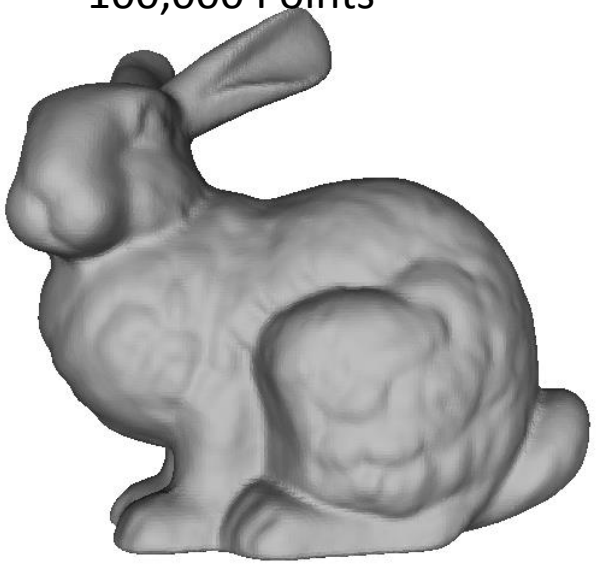
100,000 Points



res=64^3
tris=11,900
time=0:01

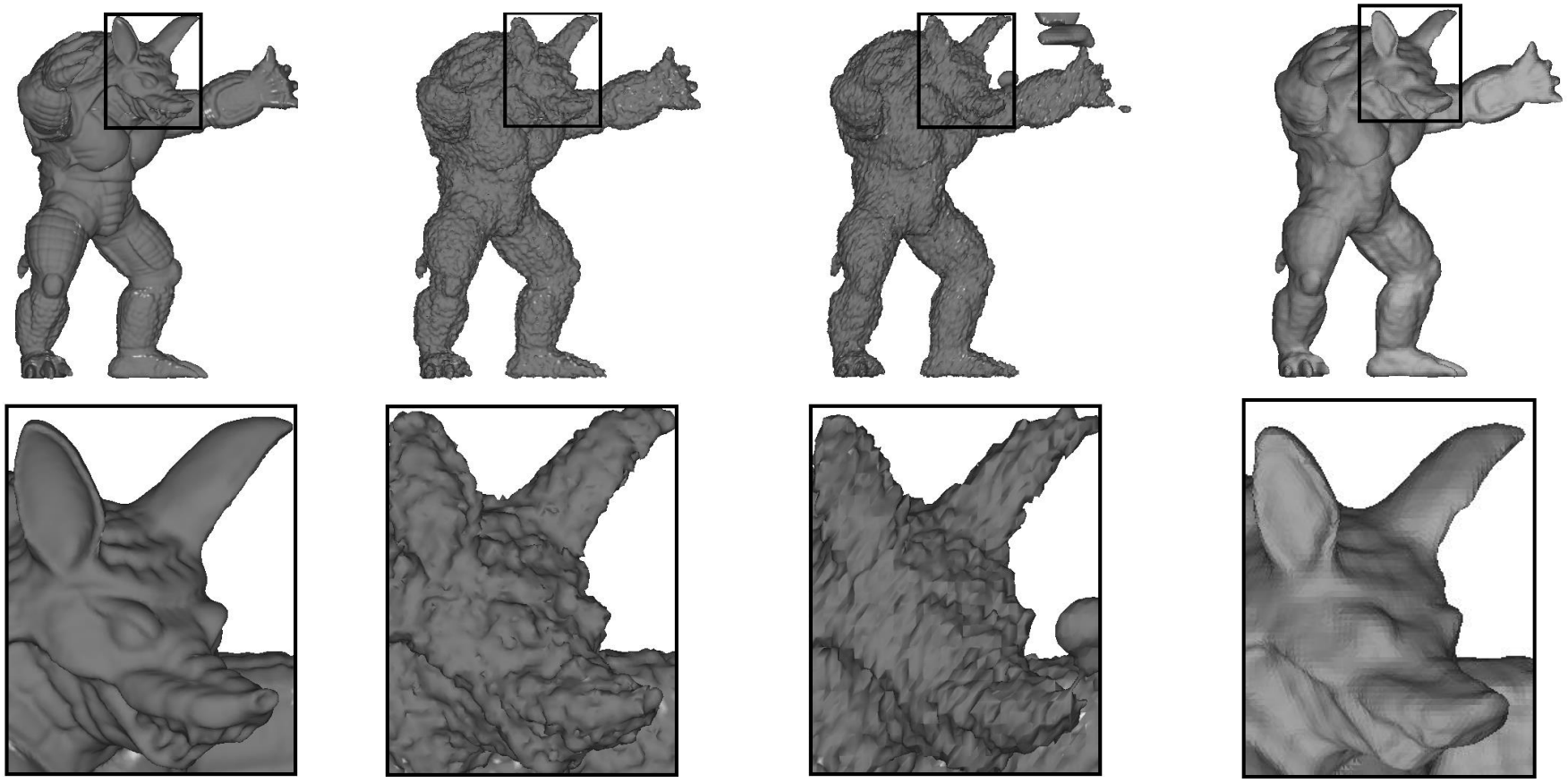


res=128^3
tris=49,556
time=0:03



res=256^3
tris=200,692
time=0:17

Results (Noise / Related Work)



Original

RBF Reconstruction

res=256^3
tris=200,000
time=24:10
points=100,000

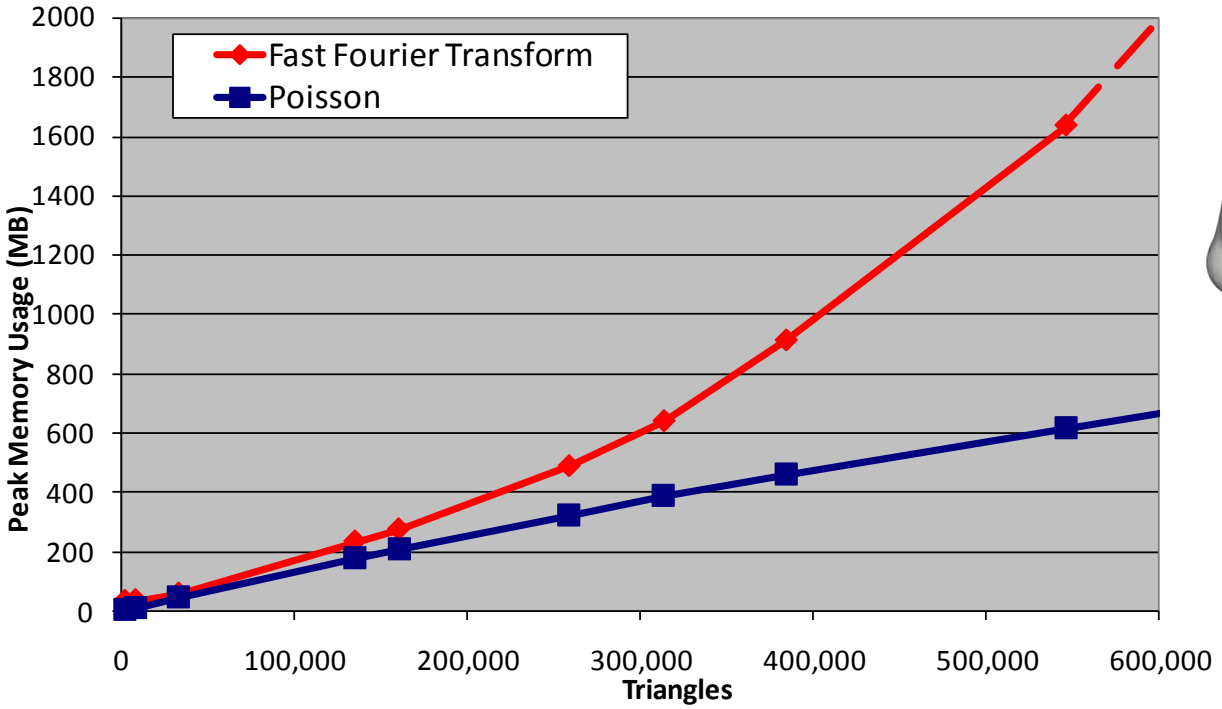
MPU Reconstruction

res=256^3
tris=205,000
time=2:14
points=100,000

FFT Reconstruction

res=256^3
tris=177,000
time=0:16
points=100,000

Memory Usage: FFT vs Poisson



Michelangelo's David

216×10^6 points (4.8 GB)

Effective Resolution: 2048^3

Projected FFT Recon:

Time: ~6hrs

Memory: 512GB

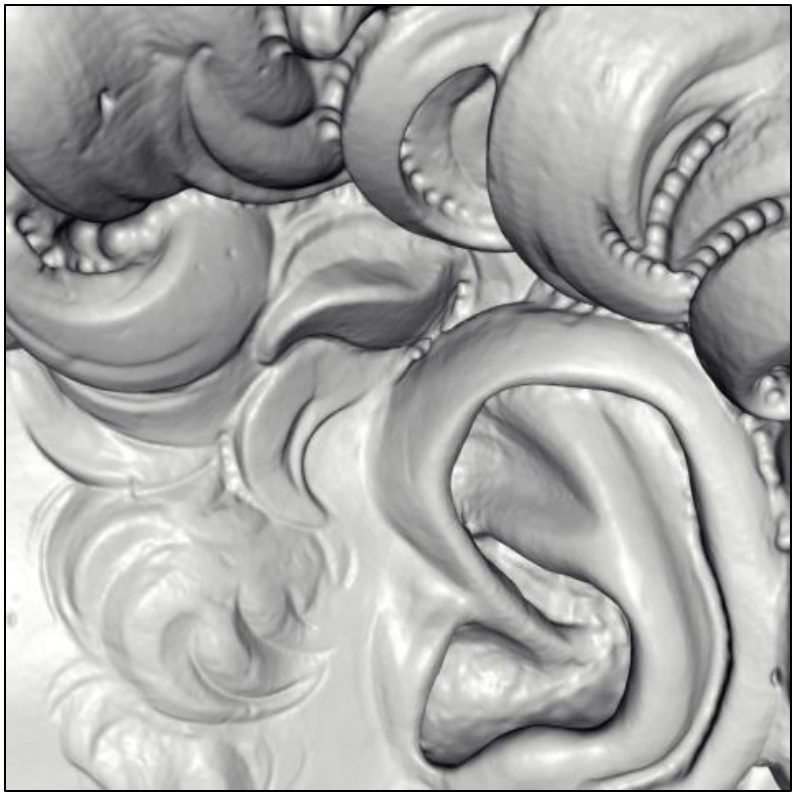
Poisson Recon:

Time: ~2.5hrs

Memory: 4.4GB

Triangles: 22 million

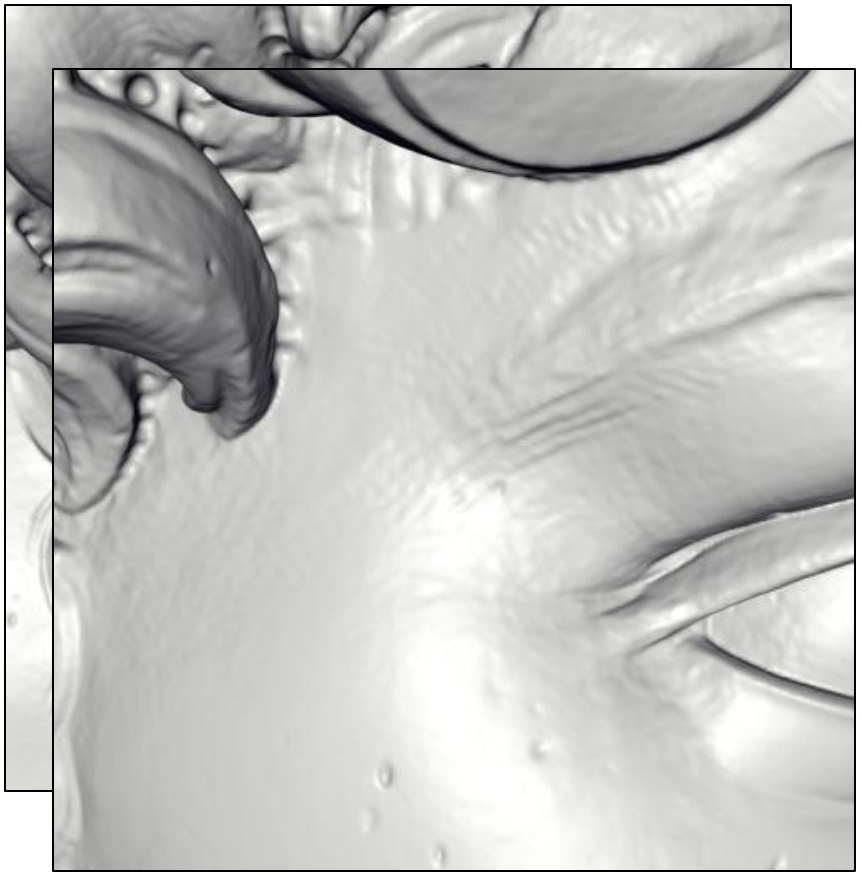
Michelangelo's David



$216x10^6$ points (4.8 GB)

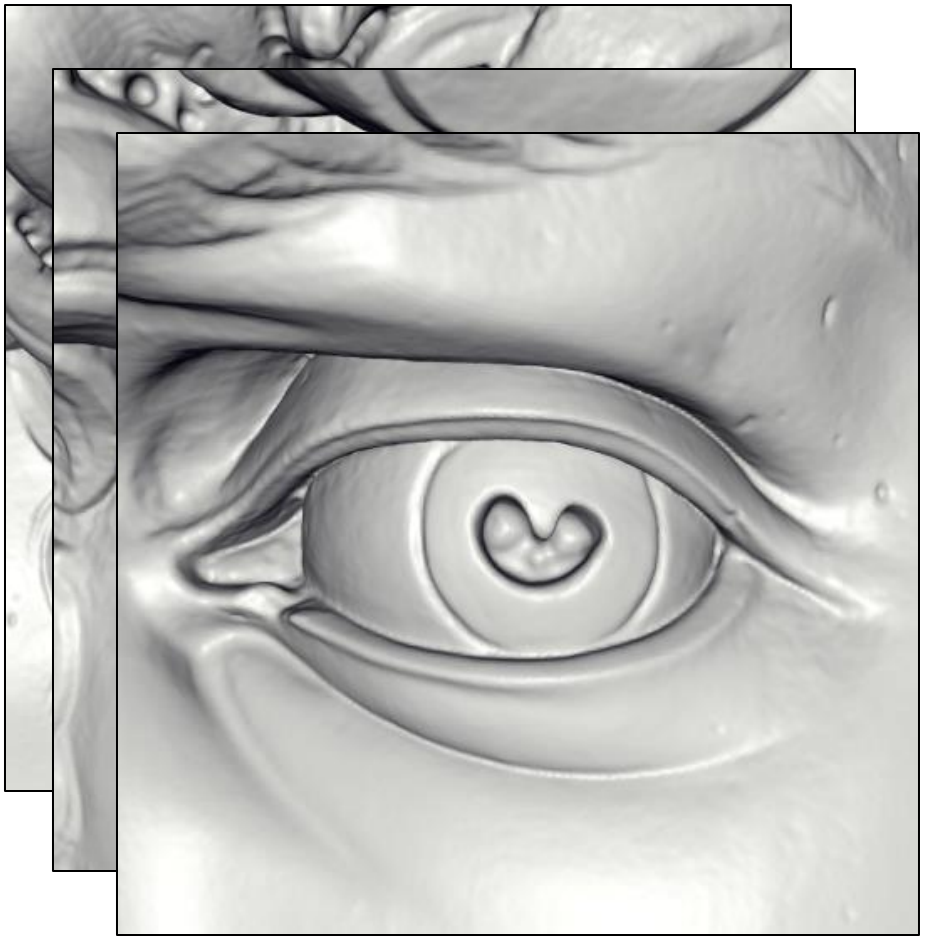
Michelangelo’s David

$216x10^6$ points (4.8 GB)



Michelangelo’s David

216×10^6 points (4.8 GB)



Streaming Poisson Results

$216x10^6$ points (4.8 GB)

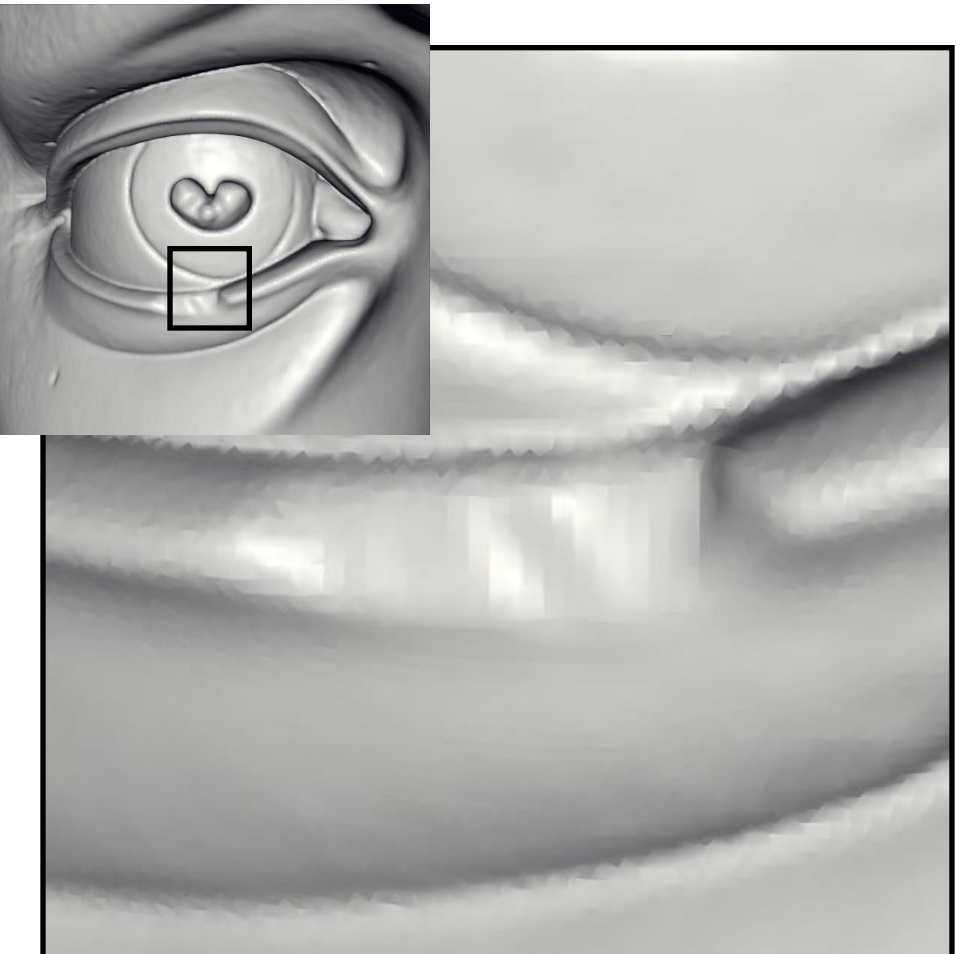
Res.	Octree Memory	Peak Memory	Running Time
256	48 49	521 309	0.53 0.50
512	168 188	278 442	0.68 0.65
1024	702 818	213 1,285	1.20 1.05
2048	3,070 3,695	212 4,442	3.33 2.65
4096	13,367 N/A	427 N/A	12.6 N/A
8192	39,452 N/A	780 N/A	32.3 N/A

Out-of-Core (Streaming) Reconstruction

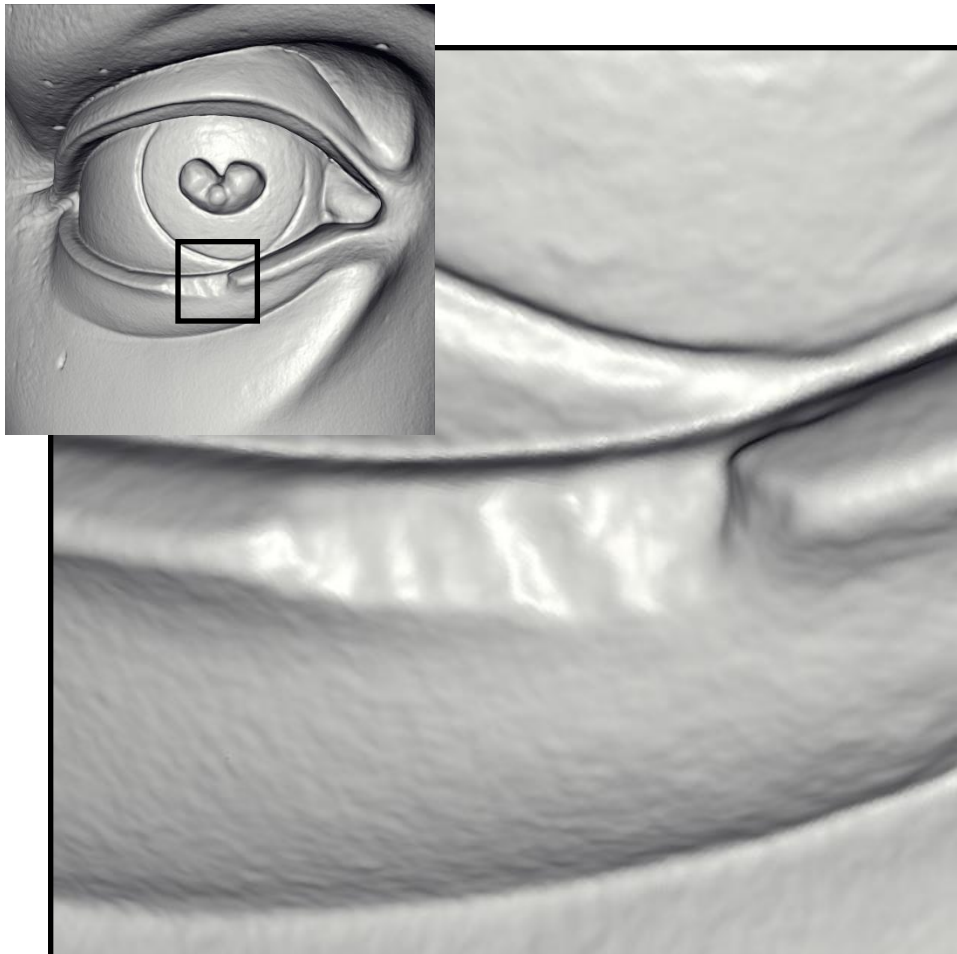
In-Core Reconstruction

Streaming Poisson Results

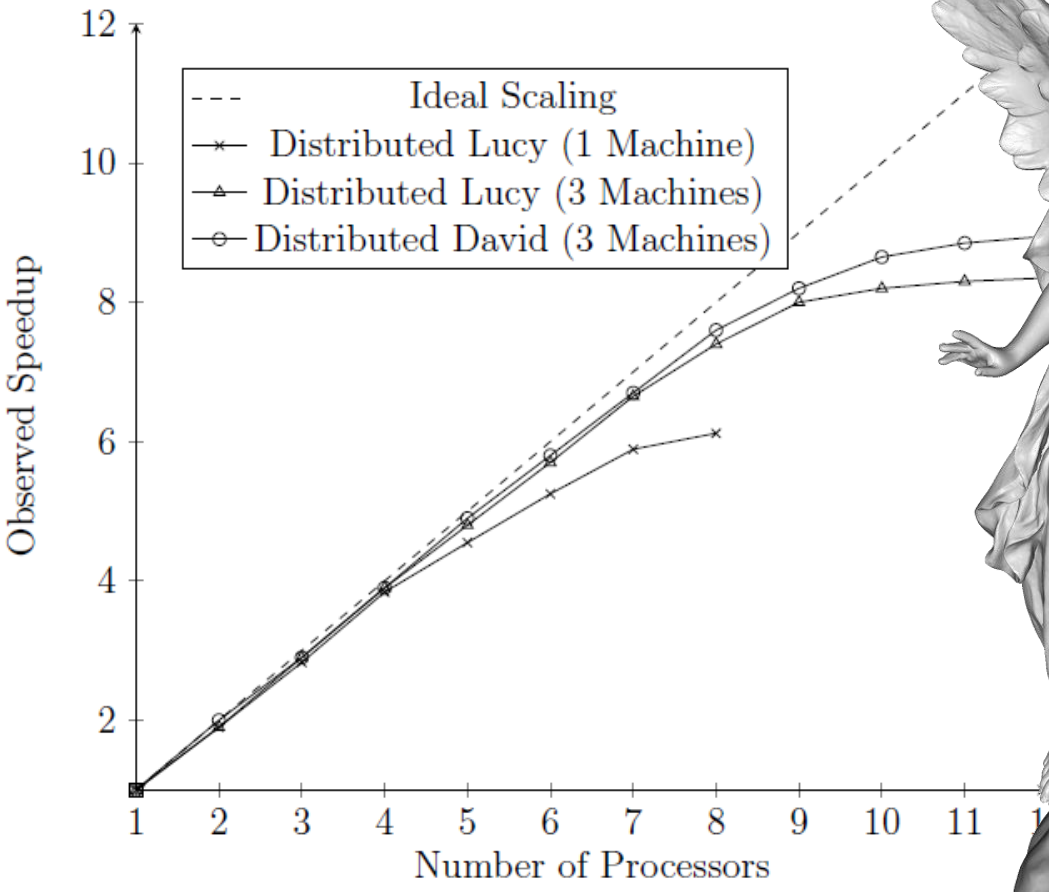
In-Core Reconstruction
Peak Mem: 4.4 GB (Depth 11)

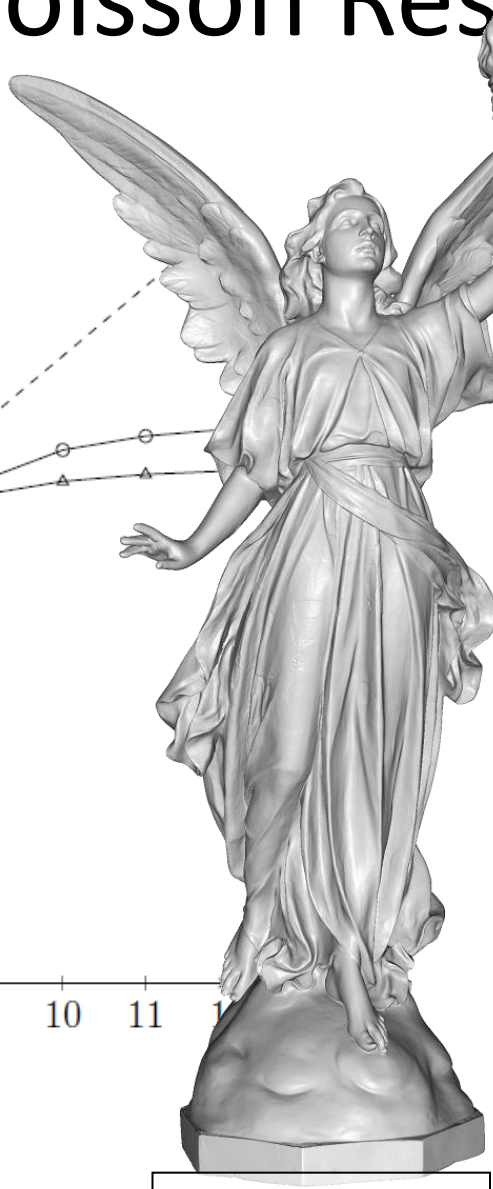


Out-of-Core (Streaming) Reconstruction
Peak Mem: 0.8 GB (Depth 13)

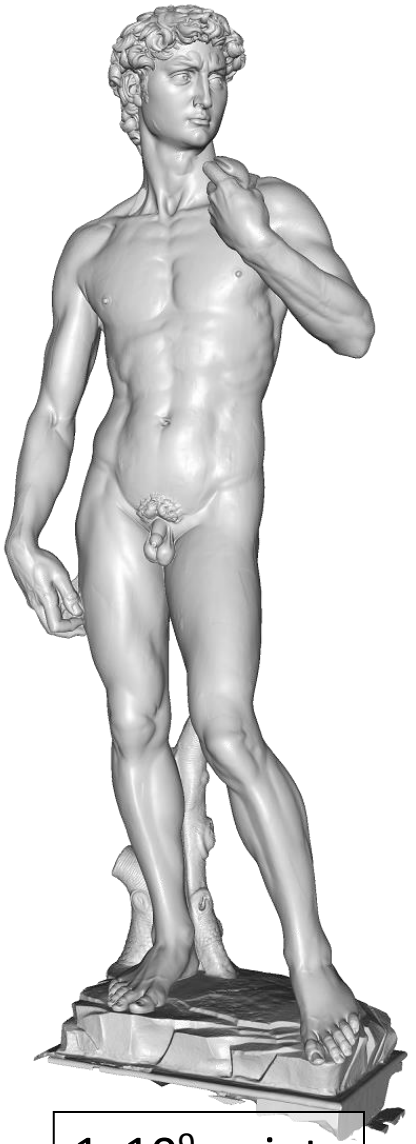


Parallel Poisson Results





$94x10^6$ points



$1x10^9$ points

Conclusion

Two different interpretations of the input samples:

- FFT: Samples as a tool for volume integration – informs how we handle non-uniform sampling
- Poisson: Samples as the gradient field – supports the design of an efficient solver.

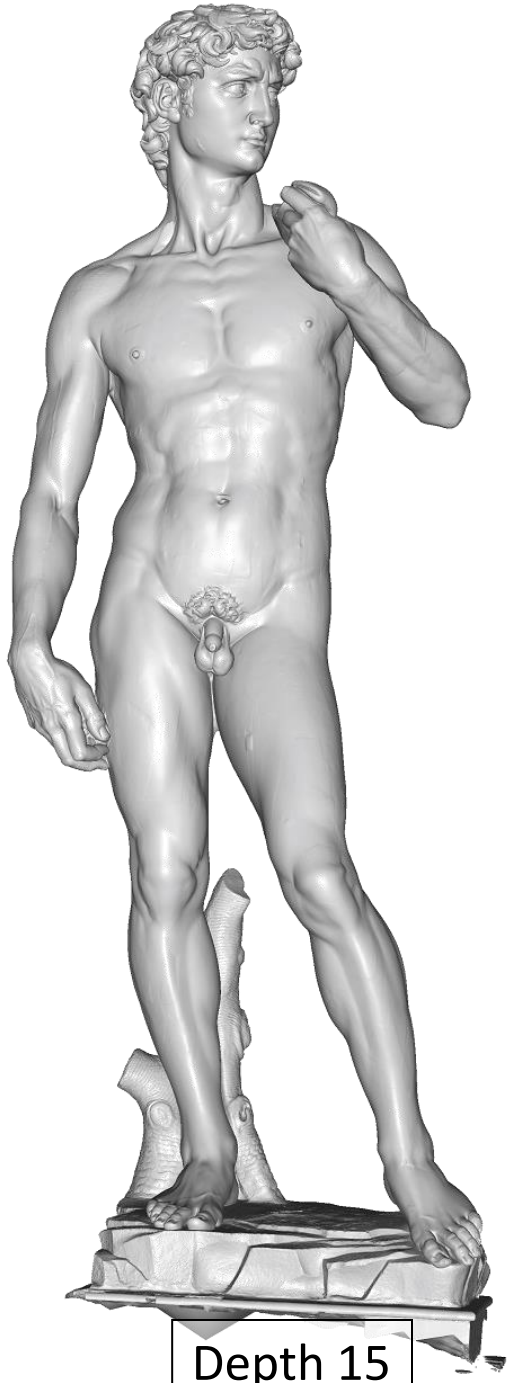
Conclusion

We can robustly reconstruct surfaces in a memory footprint smaller than:

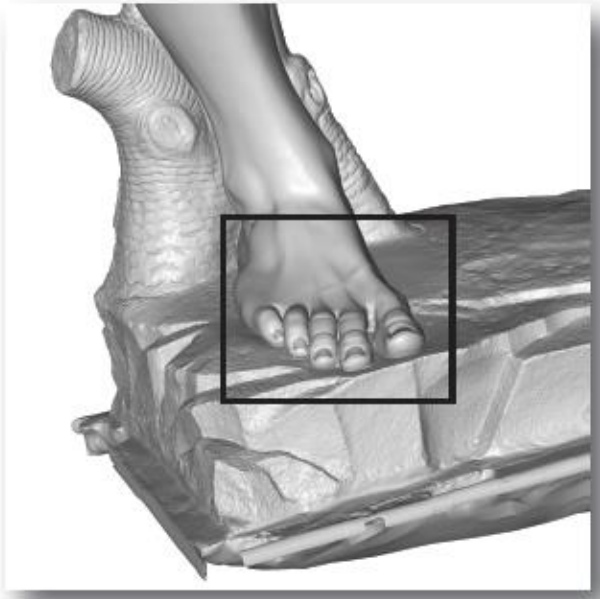
- The total memory used
- The size of the input point set
- The size of the output surface

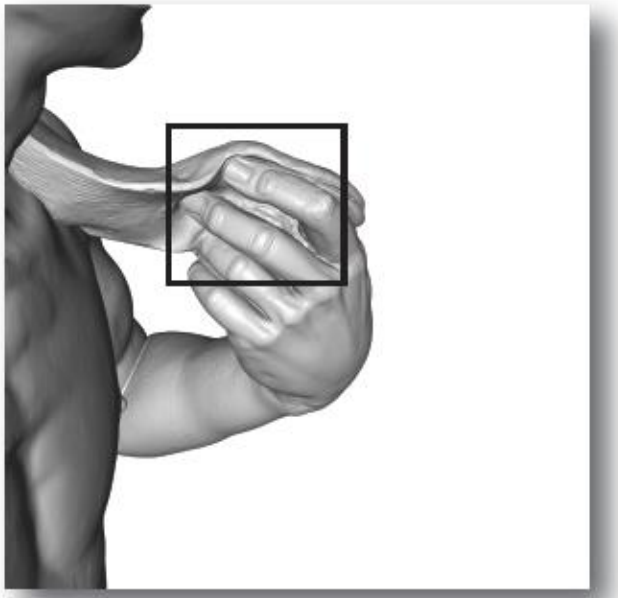
using a streaming and parallel, linear-time algorithm.

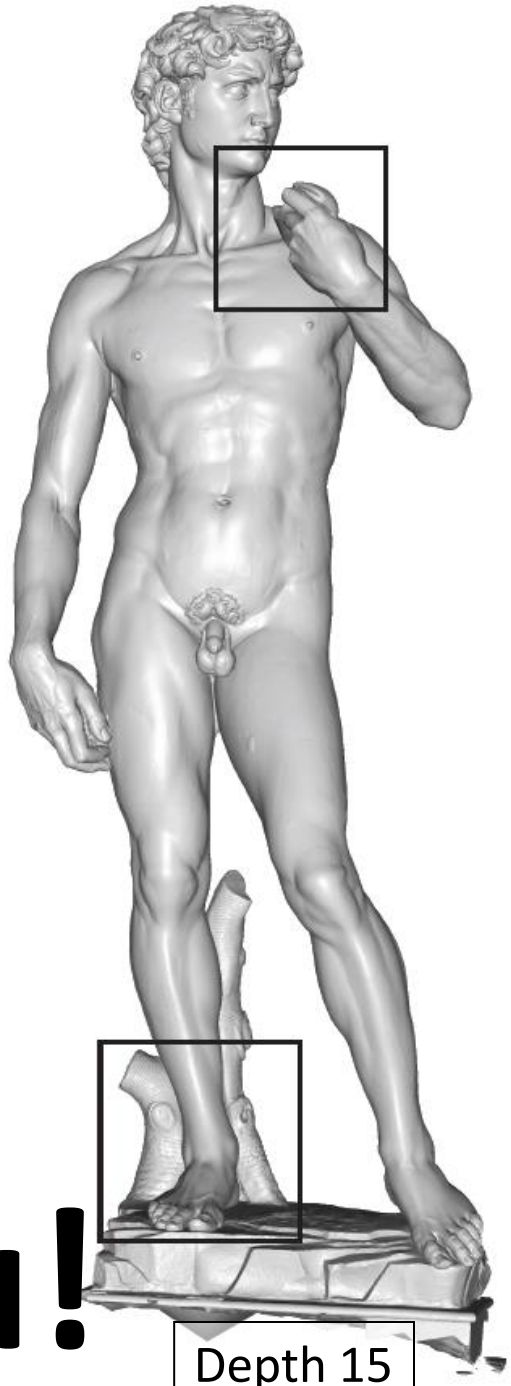
Points ($1x10^9$)	24 GB
Triangles ($1x10^9$)	18 GB
Total Memory	242 GB
Working Memory	2 GB
Time (12 cores)	886 min



Depth 15







Thank You!

Depth 15