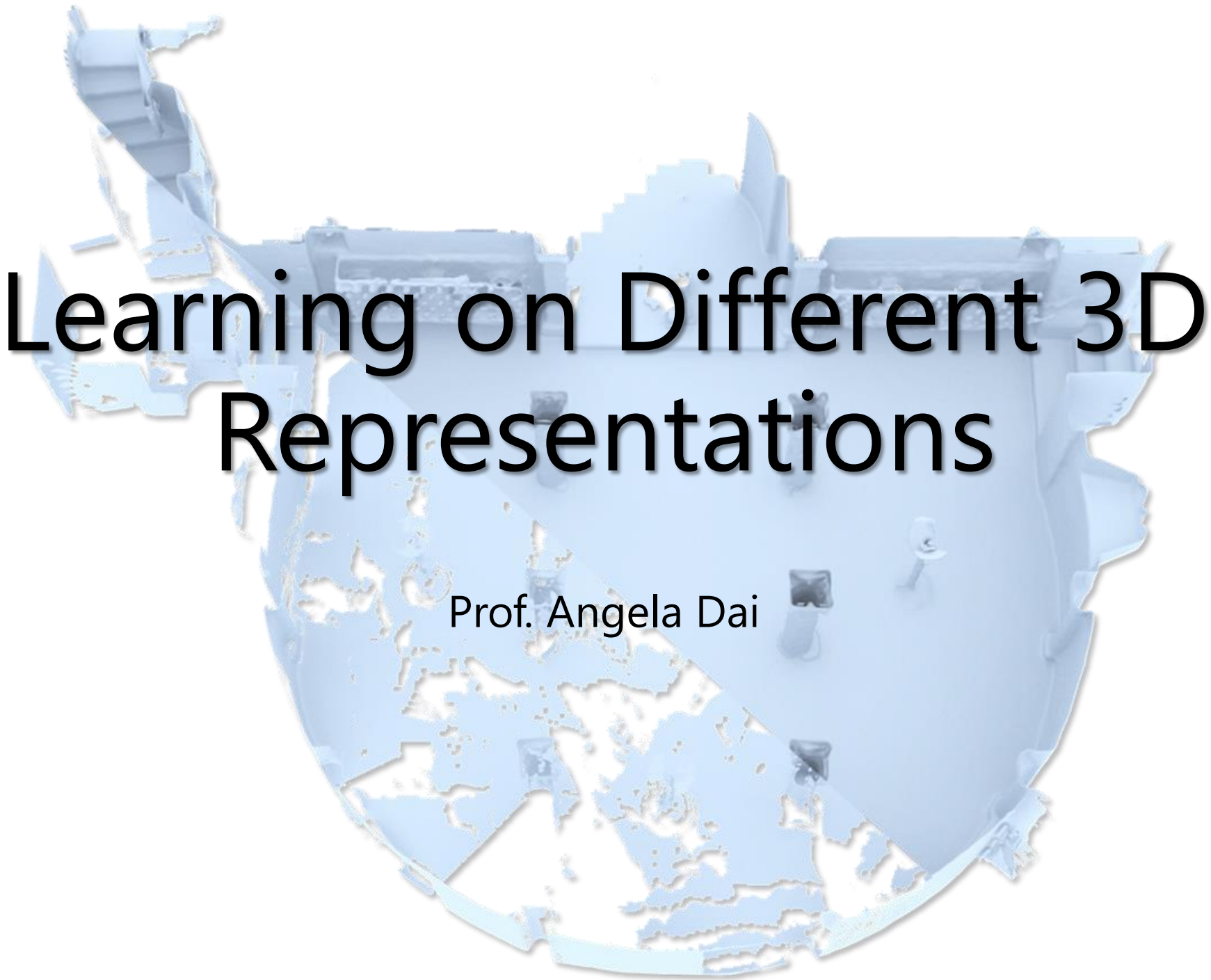
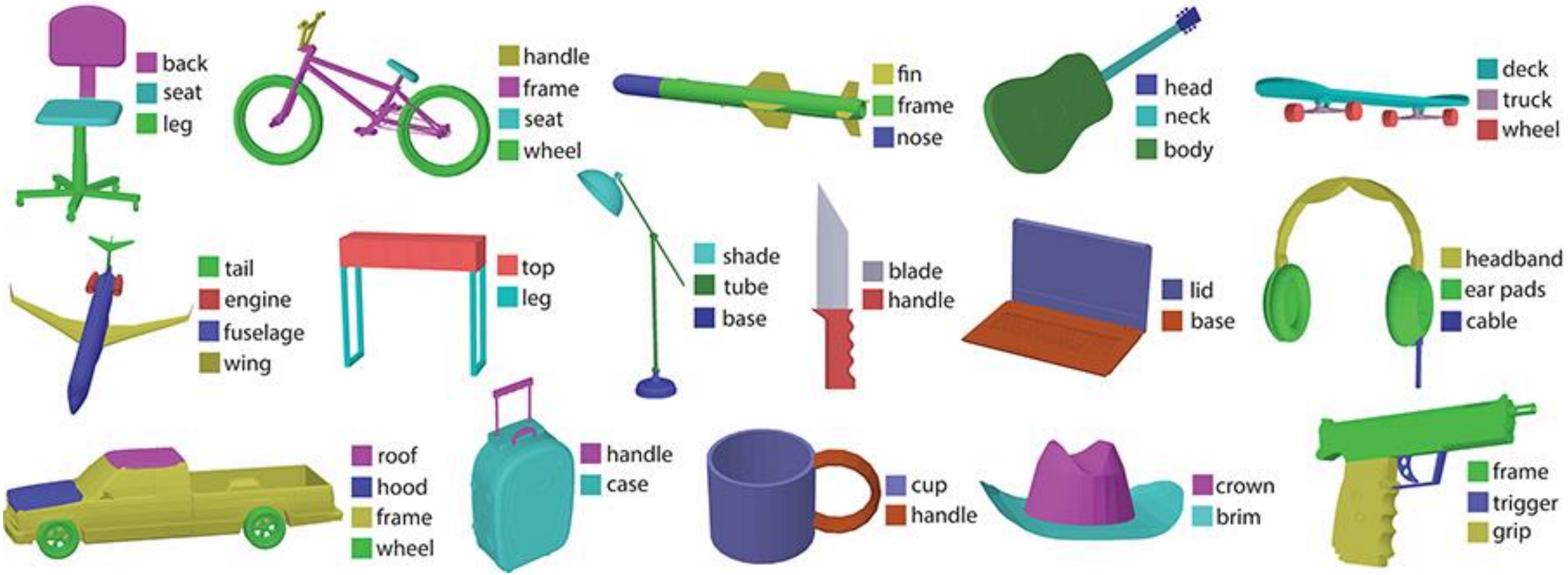


Learning on Different 3D Representations

Prof. Angela Dai

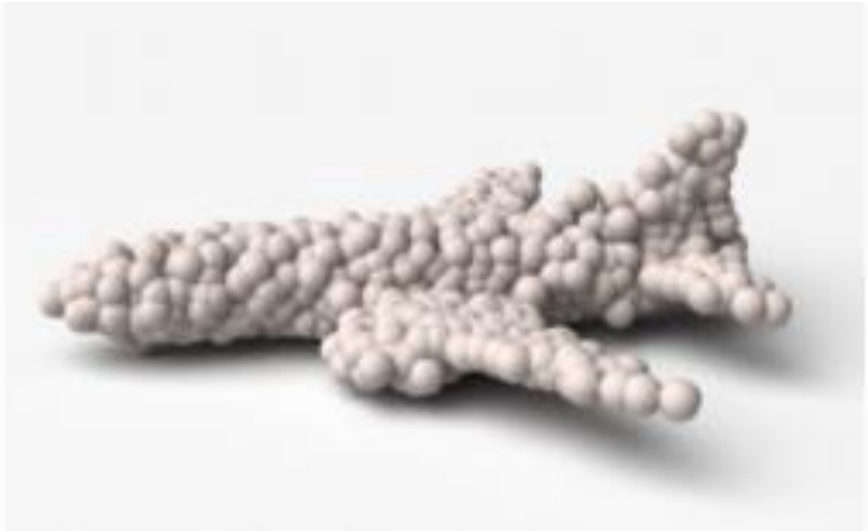
Brief Recap

Shape segmentation into parts

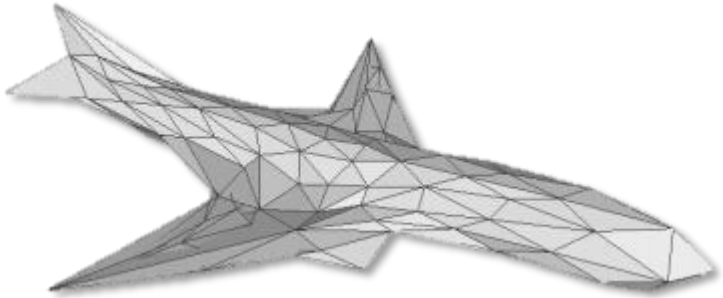


Generating Shapes

Signed Distance Fields

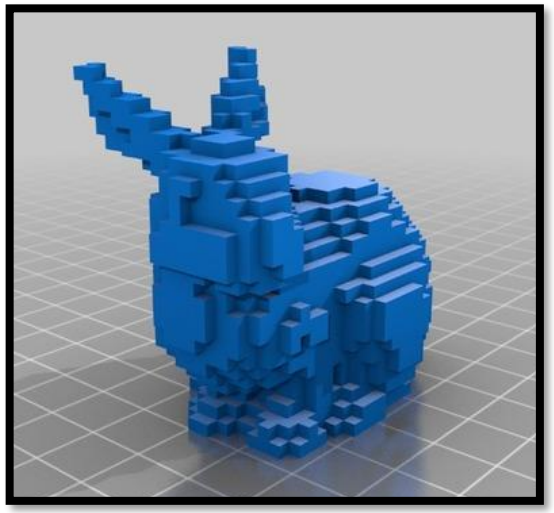


Point Clouds

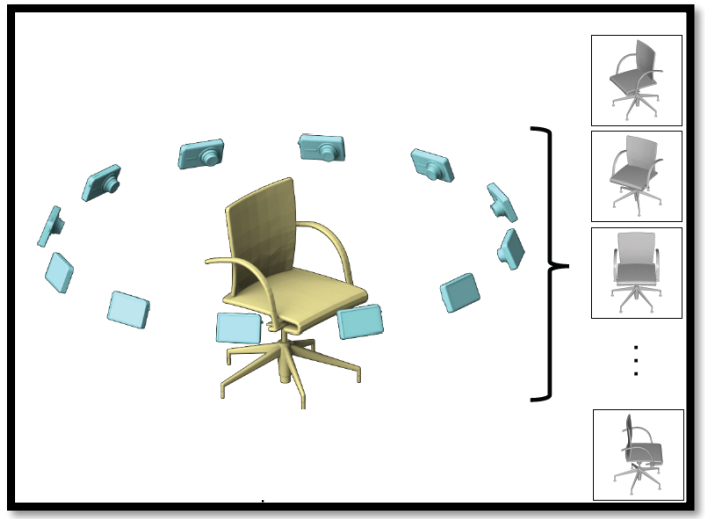


Meshes

3D Deep Learning by Representations

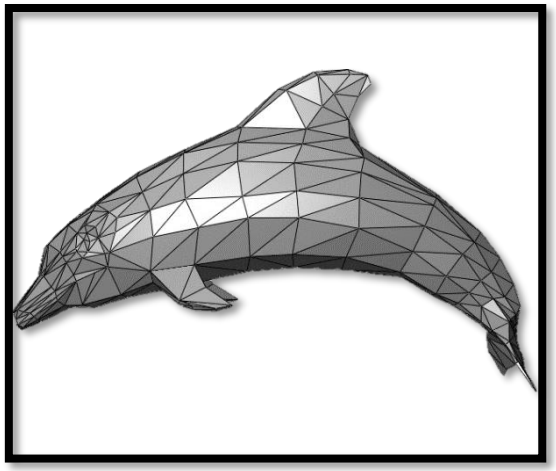


Volumetric
3D CNNs: Dense, Hierarchical, Sparse



Multi-View
(also: multi-view + volumetric/point/mesh)

Point Cloud

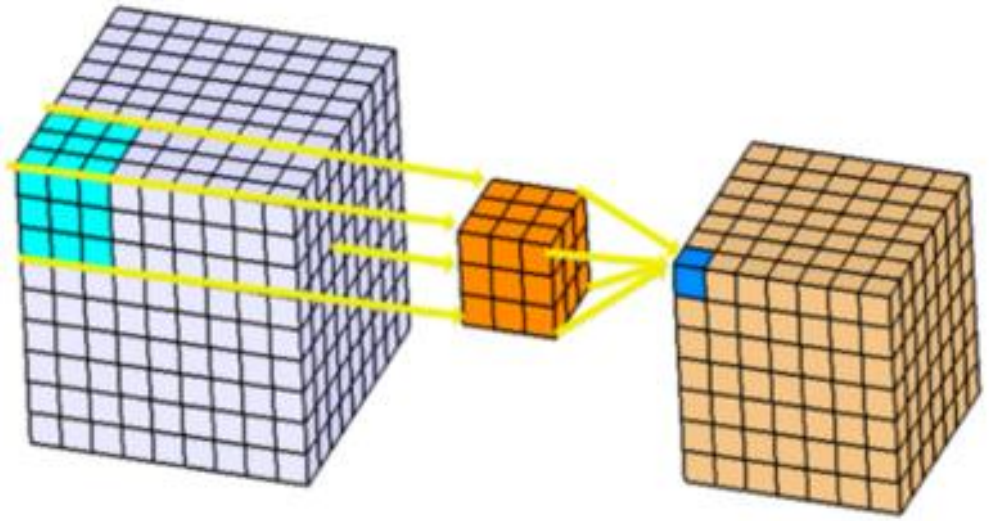


Mesh
Graph Neural Networks

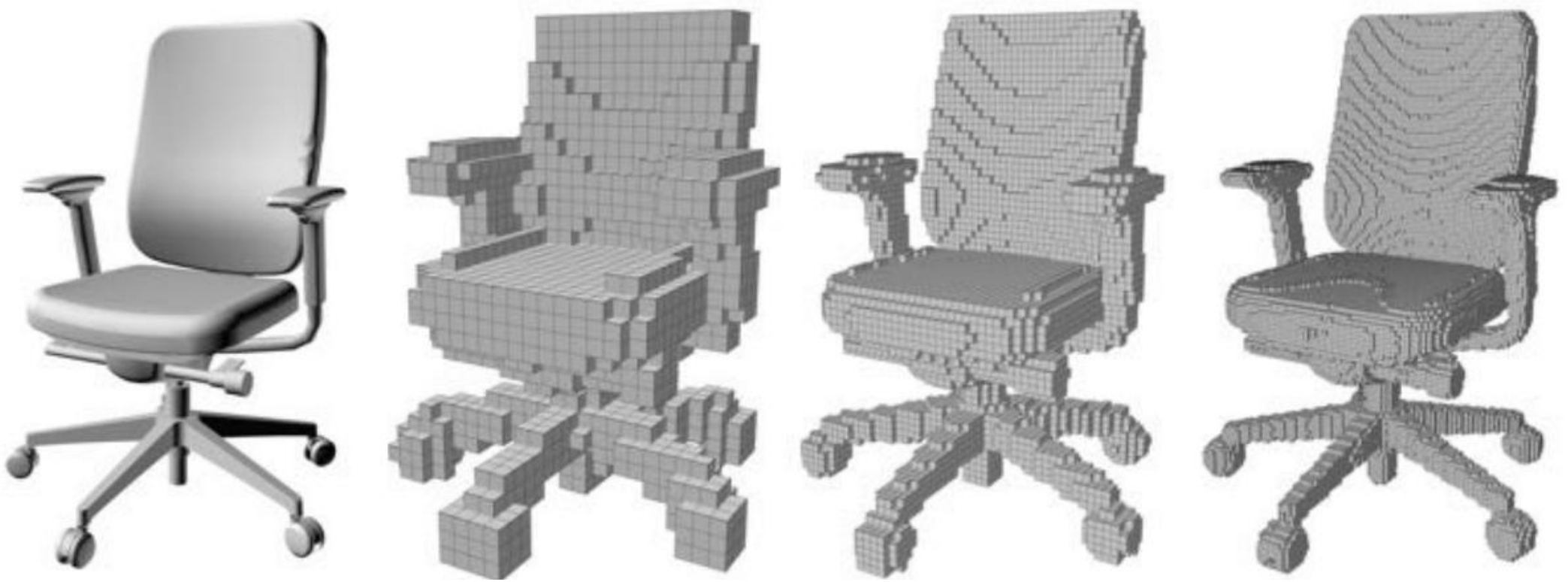
and more!

Volumetric Representations

- 3D Convolutions
  - Direct extension of 2D convolutions
- Often: occupancy grid, signed/unsigned distance fields
- Operate on regular grid structures
  - Get neighborhood structures and spatial propagation
  - Well-defined pooling and unpooling for hierarchical processing
  - Cubic growth in memory



Dense Volumetric Grids



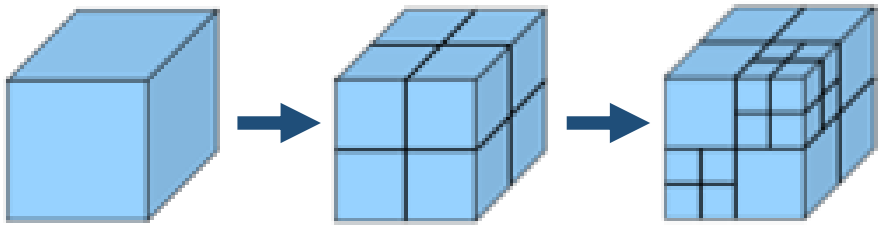
Percentage Occupancy:	10.4%	5.1%	2.4%
	32^3	64^3	128^3

Volumetric Representations

- Can we be more efficient than a dense volumetric grid?

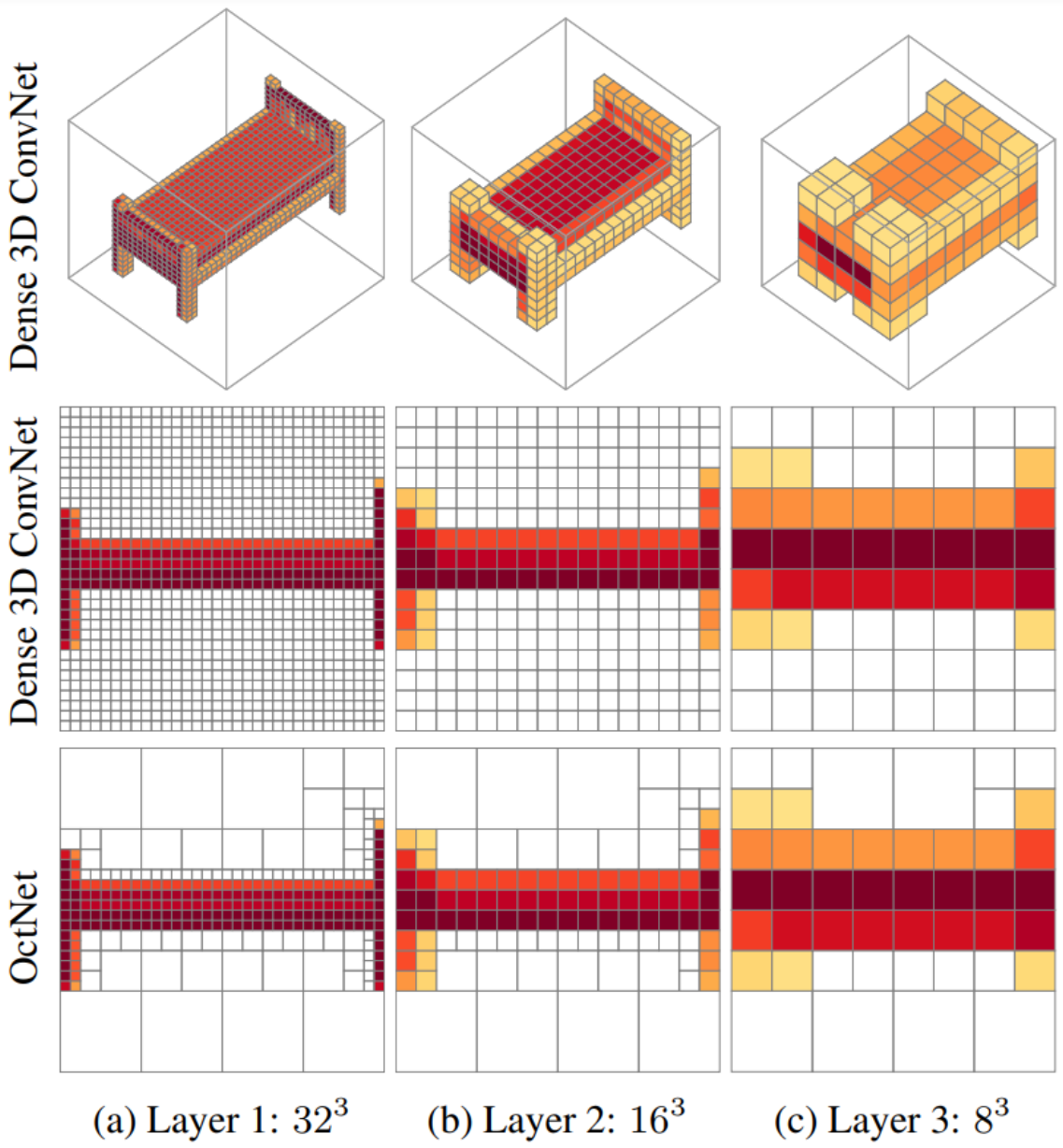
➢Hierarchical data structures

➢Operate on octree
  - Be careful about efficient neighbor queries



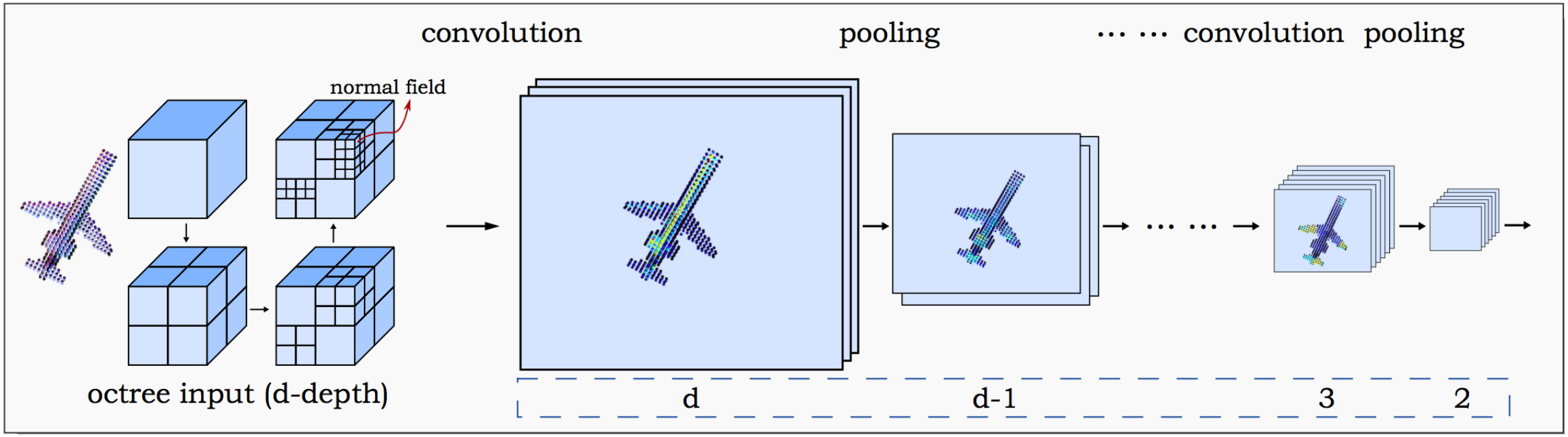
OctNet

- Internal octree representation
  - Limited max depth (e.g., 3) and regular grid structure at each level to aid neighbor queries
- Standard 3D convolutions but using octree data structure for memory efficiency
- Requires known octree structure
  - No generative tasks



[Riegler et al. '17]

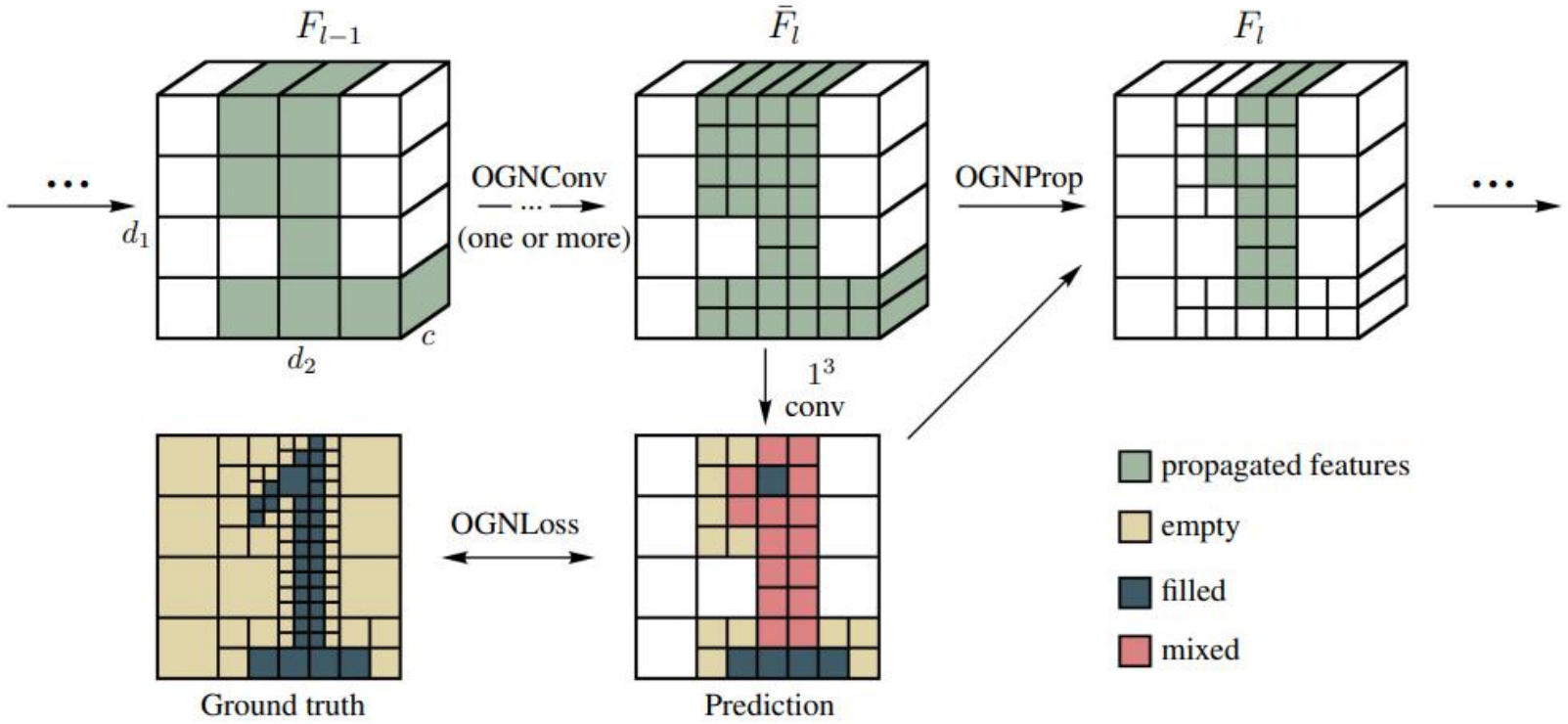
OctNet



[Riegler et al. '17]

Octree Generating Networks

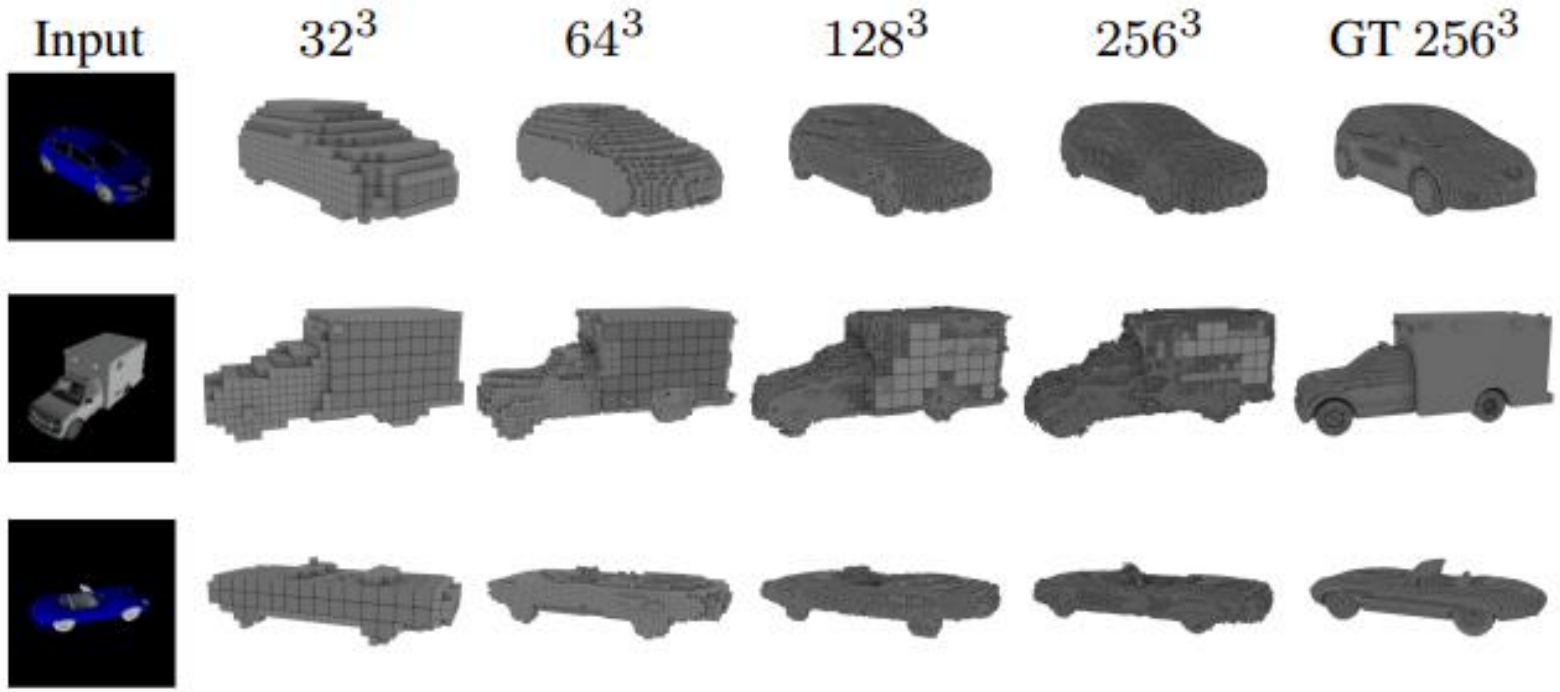
- Enable generating octree structures
- Predict if a cell is empty, filled, or mixed



[Tatarchenko et al. '17]

Octree Generating Networks

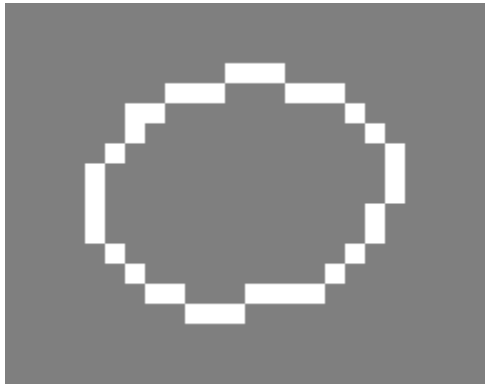
- Enable generating octree structures
- Predict if a cell is empty, filled, or mixed



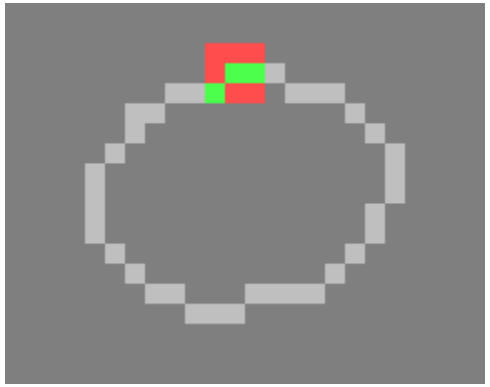
[Tatarchenko et al. '17]

Submanifold Sparse Convolutions

- Regular 3×3 convolution



- Submanifold sparse 3×3 convolution

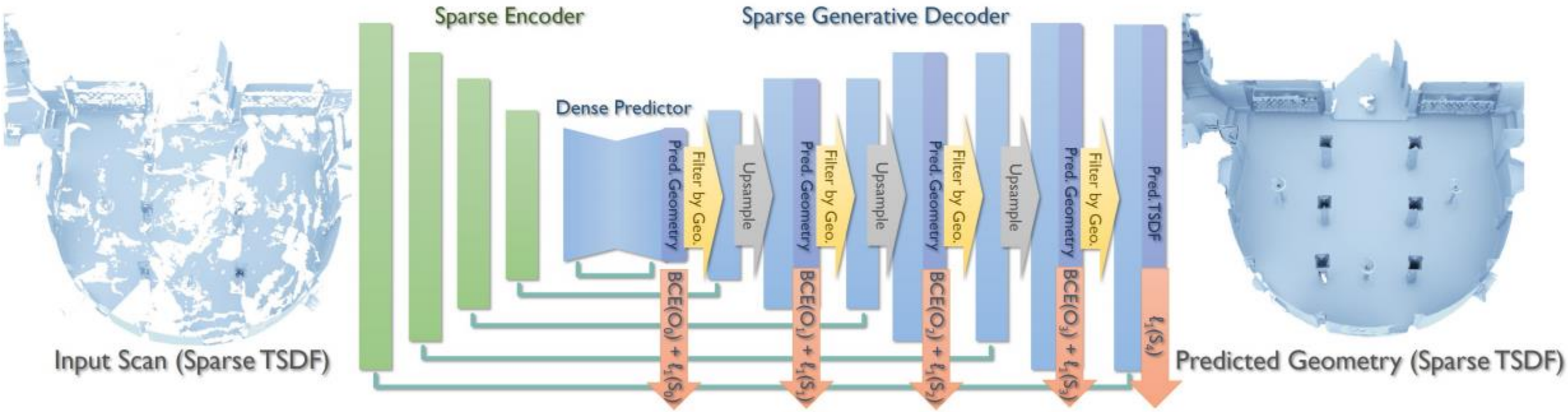


Submanifold Sparse Convolutions

- Very memory-efficient representation
- Hashing for storing sparse set of voxel locations
  - Efficient lookups
- Enabled processing large scenes at much higher resolution (1cm, 2cm for room-sized scenes)
- Significant performance improvements over previous
- See also: MinkowskiEngine [Choy et al ’19]

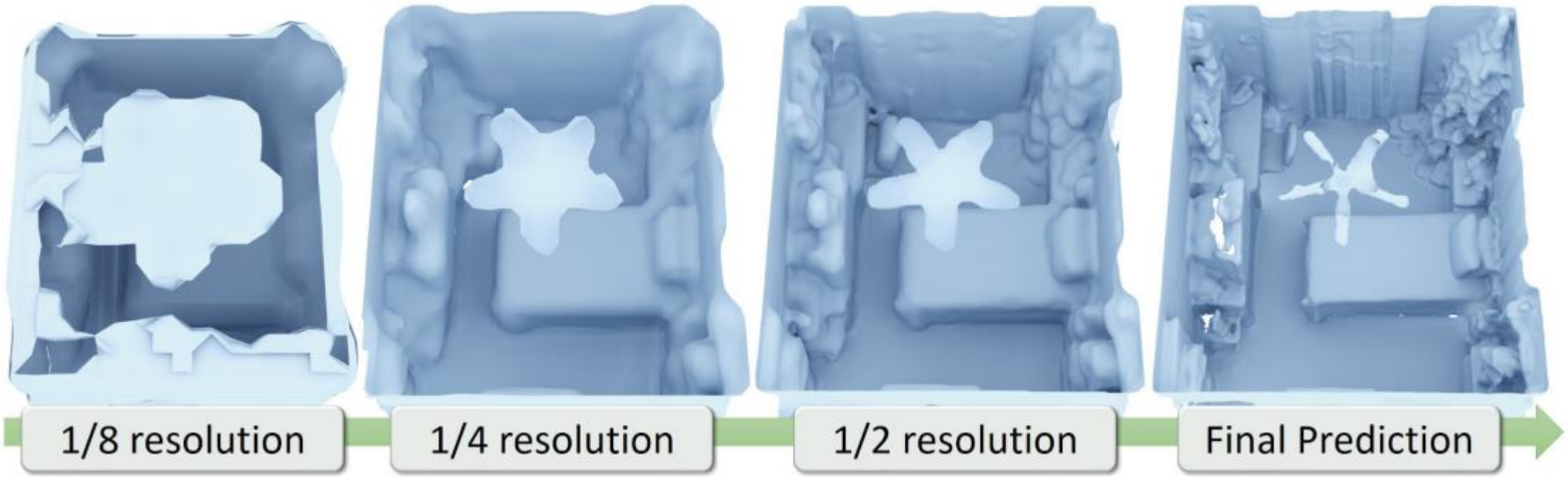
[Graham et al. ’17, Graham et al ’18]

Sparse Generative Neural Networks



[Dai et al. '20]

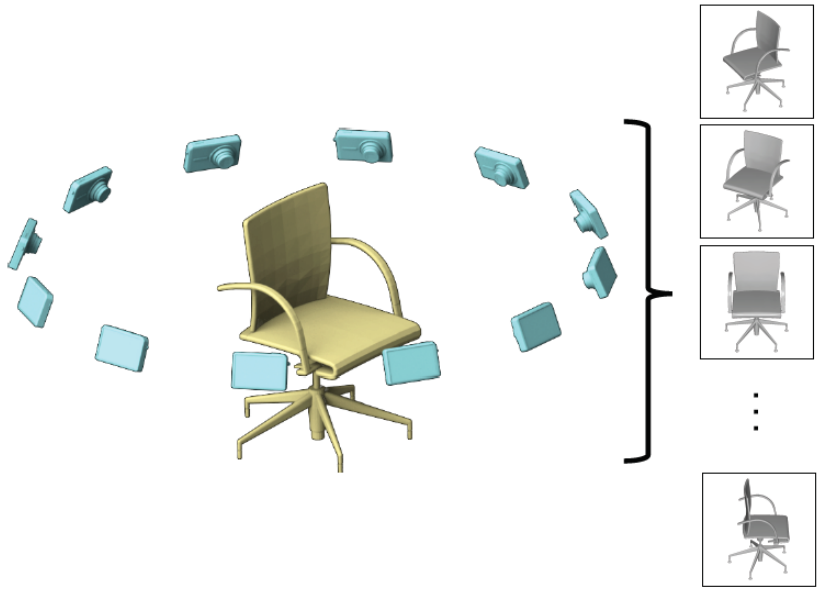
Sparse Generative Neural Networks



[Dai et al. '20]

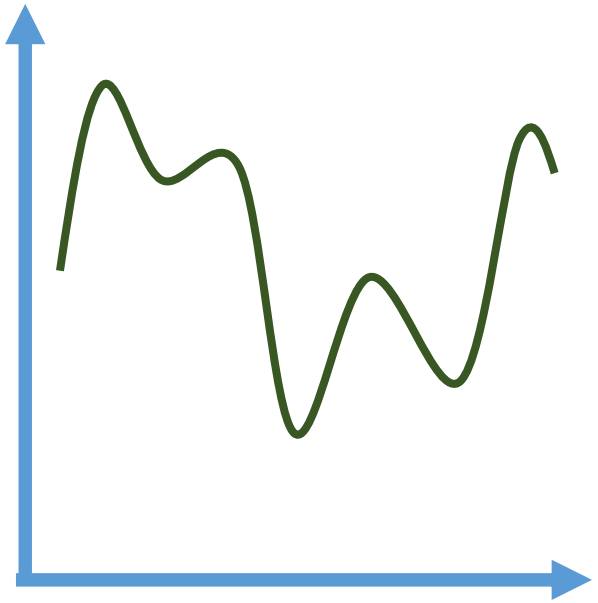
Multi-View Representations

- Rather than operate on explicit 3D, represent with multiple image views of a 3D object or scene
- Exploit powerful 2D CNN works
- Leverage high resolution 2D images



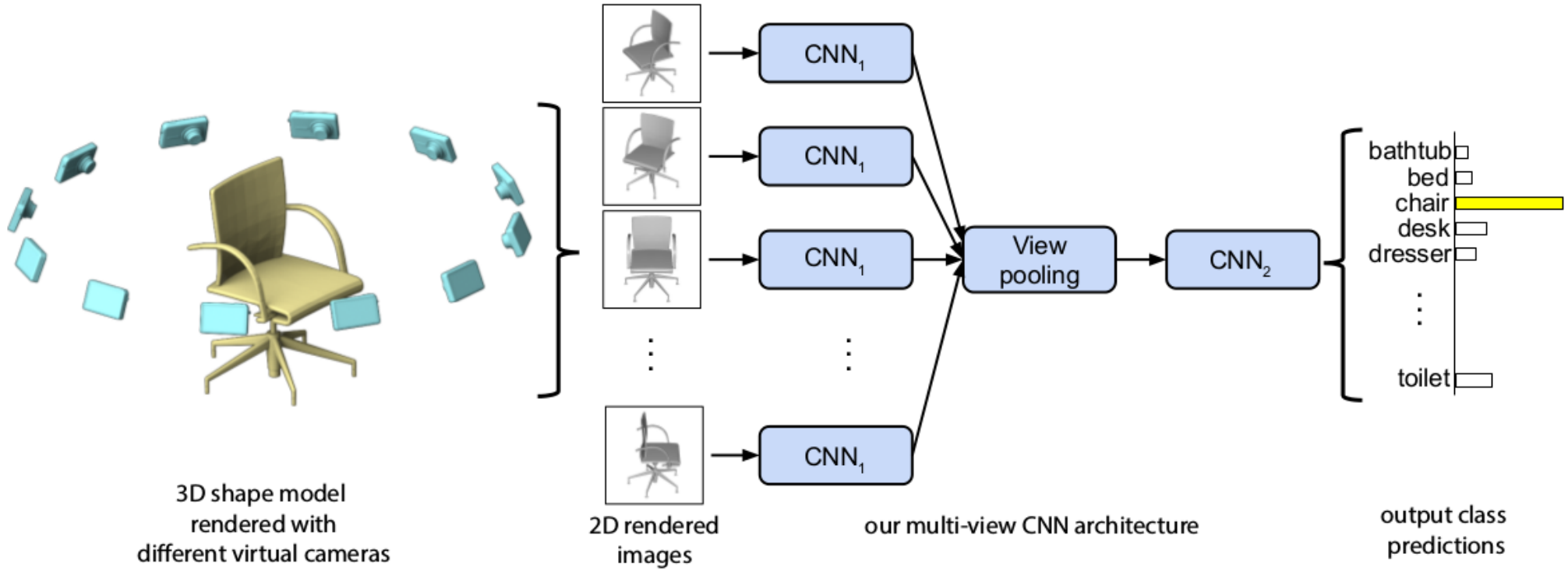
Exploit powerful 2D CNNs: Fine-tuning

- Neural network optimization is very non-convex
- More training data helps converge to better local min
- If train data is small, leverage larger dataset with some shared characteristics
- Train on large dataset, then fine-tune on train data



Multi-View Shape Recognition

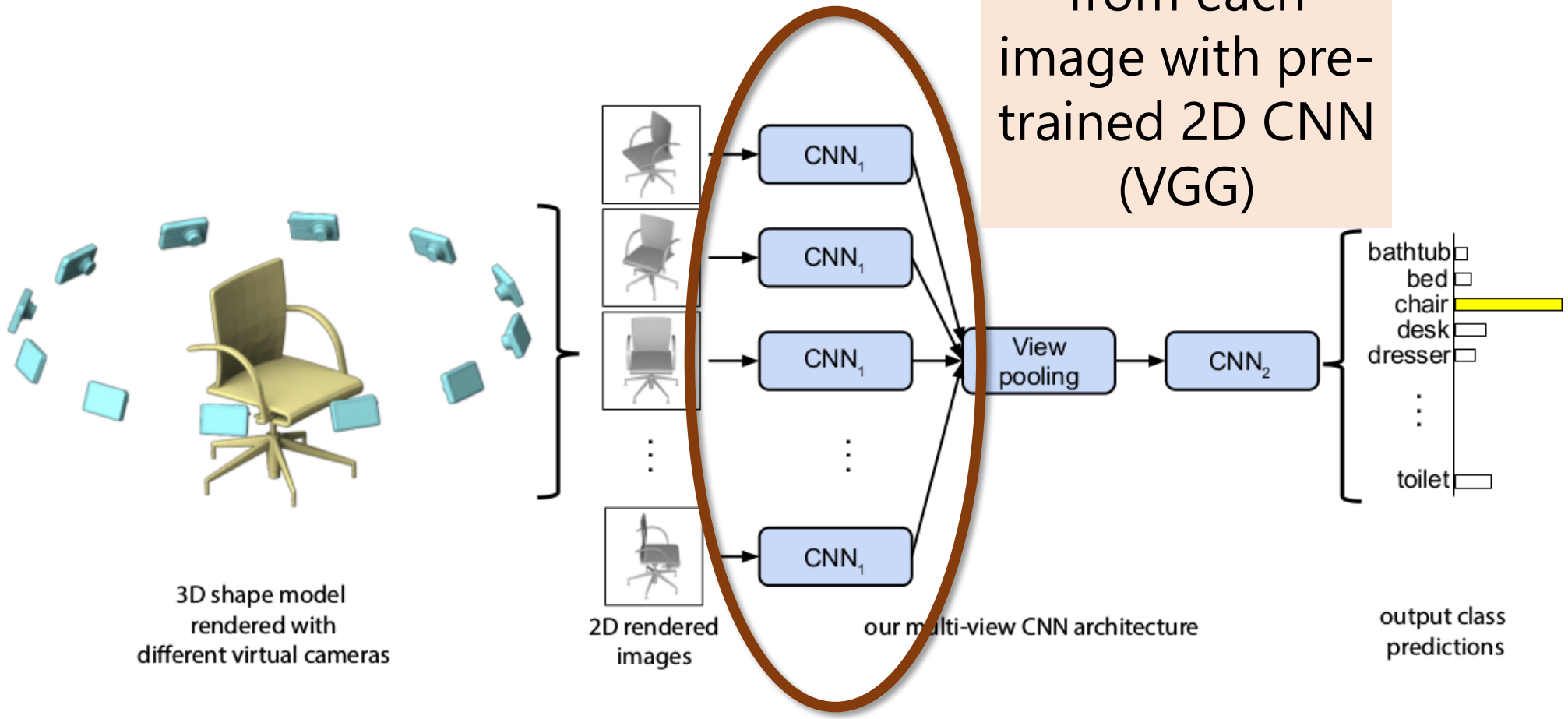
- Input: 3D Shape; Output: class category
- Render shape with multiple virtual cameras



[Su et al. '15]

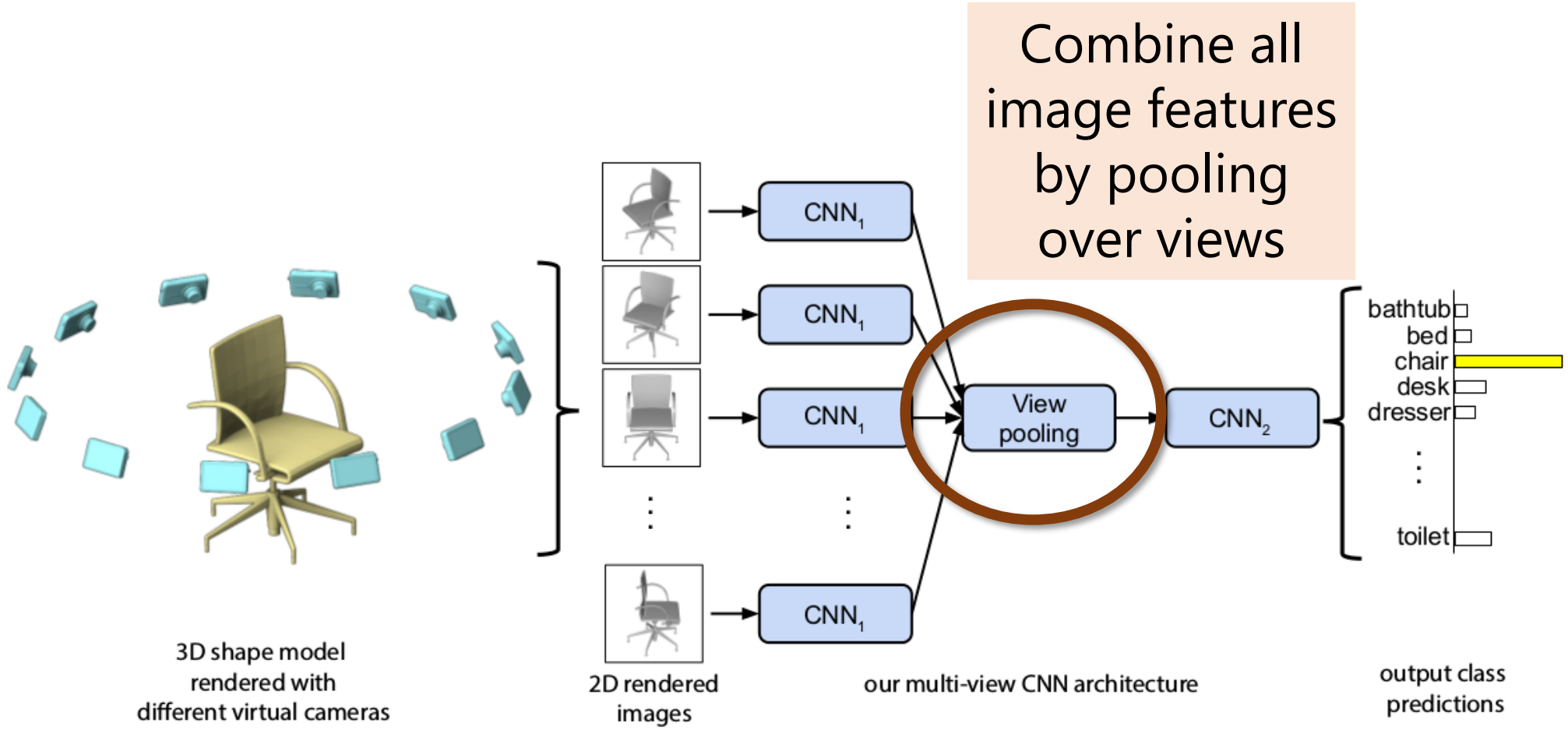
Multi-View Shape Recognition

Extract features from each image with pre-trained 2D CNN (VGG)



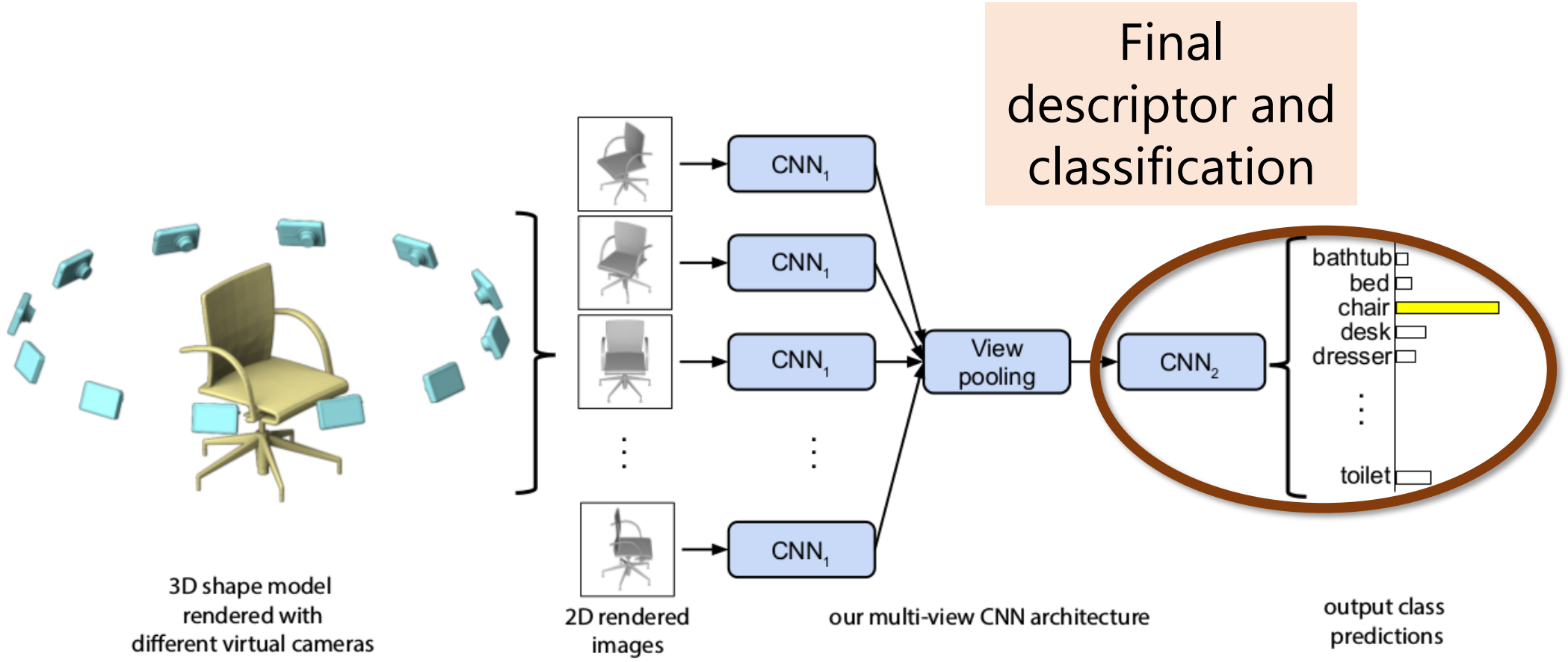
[Su et al. '15]

Multi-View Shape Recognition



[Su et al. '15]

Multi-View Shape Recognition



[Su et al. '15]

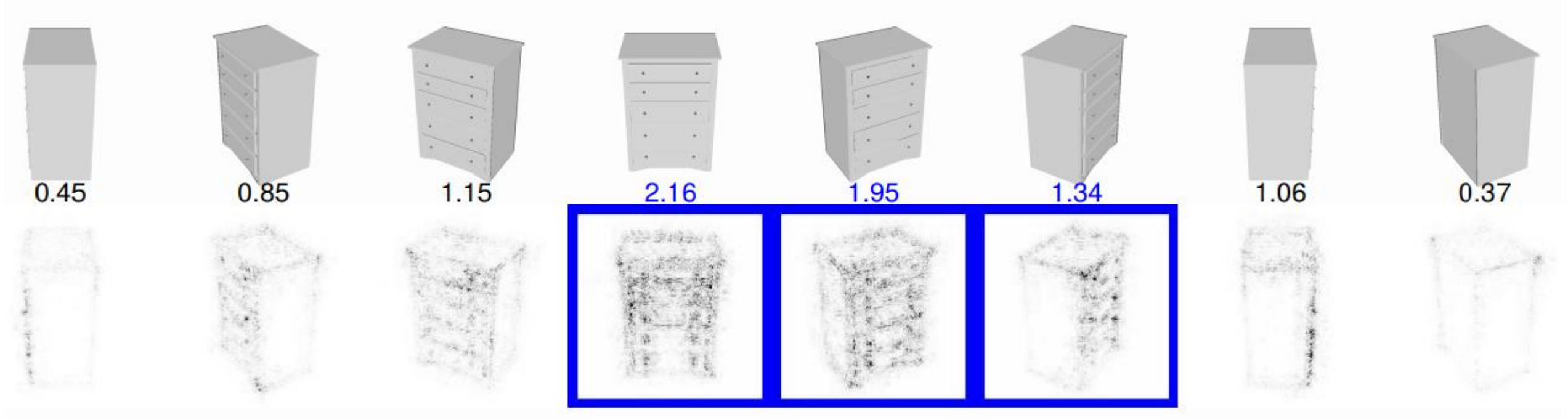
Multi-View Shape Recognition

Method	Classification (Accuracy)
SPH [16]	68.2%
LFD [5]	75.5%
3D ShapeNets [37]	77.3%
FV, 12 views	84.8%
CNN, 12 views	88.6%
MVCNN, 12 views	**89.9%**
MVCNN, 80 views	**90.1%**

[Su et al. '15]

Multi-View Shape Recognition

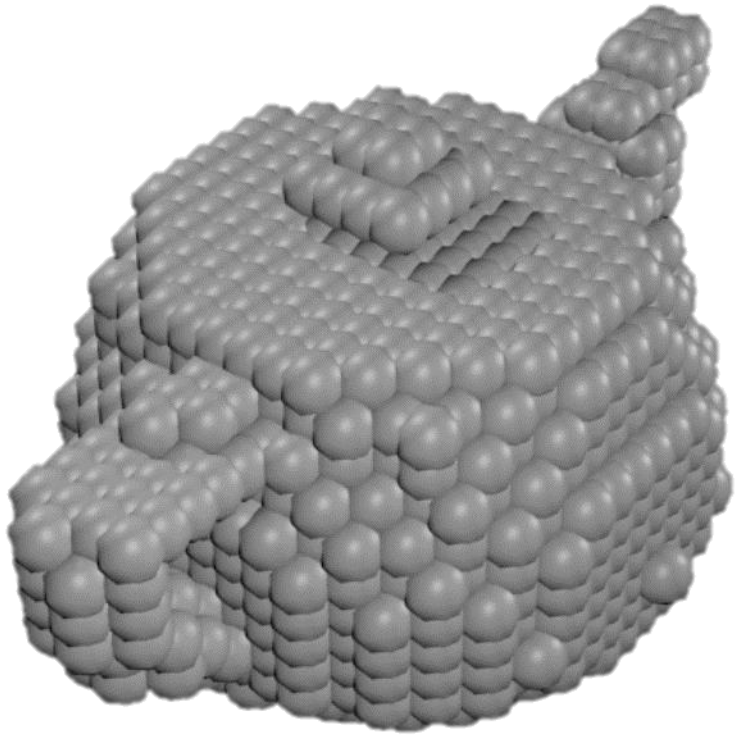
- Most salient views



[Su et al. '15]

Multi-View Point Cloud Recognition

- Render point clouds as spheres
  - Spheres are view-invariant primitives
- Apply multi-view CNN



[Qi et al. ’16]

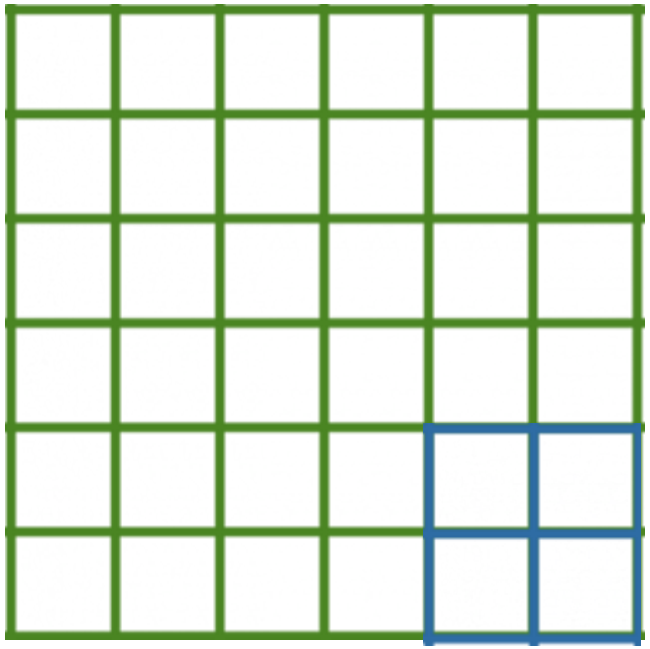
Multi-View in Practice

- Powerful use of 2D feature extraction
- Important questions:

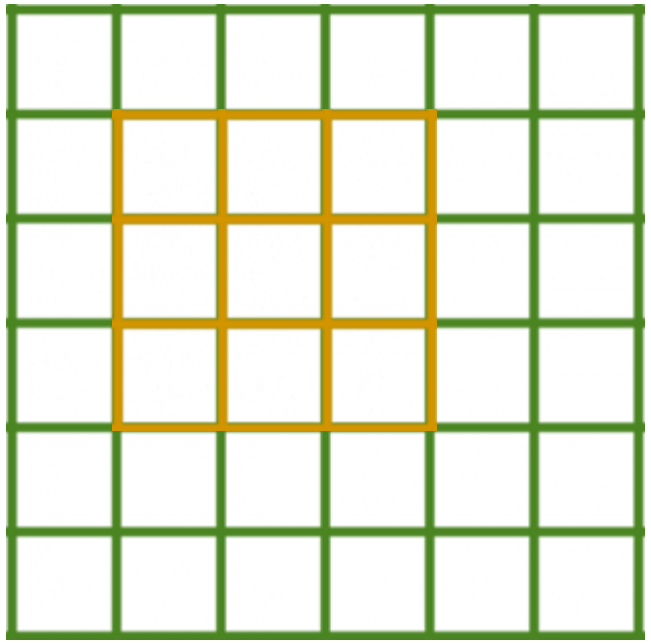
- What viewpoints to use?
- How many viewpoints?
- How to handle noisy/incomplete data?

[Qi et al. ’16]

CNNs leverage grid structures



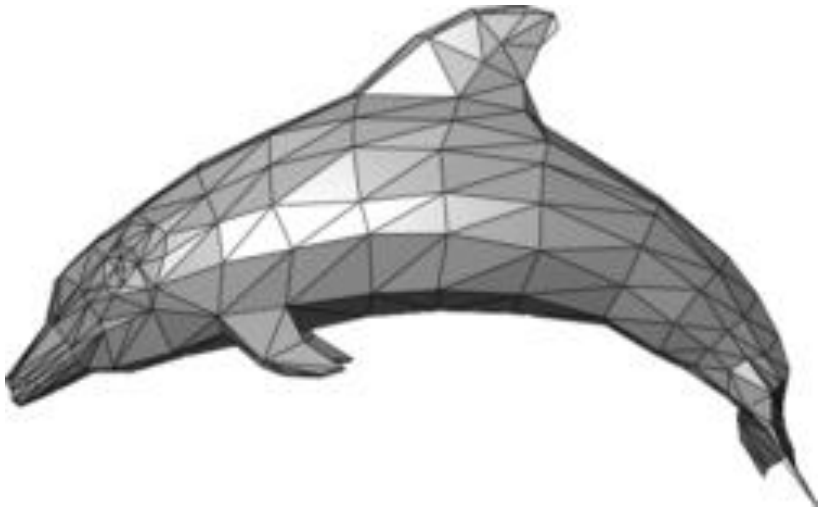
2D (or 3D) Grid:
Image or Volumetric



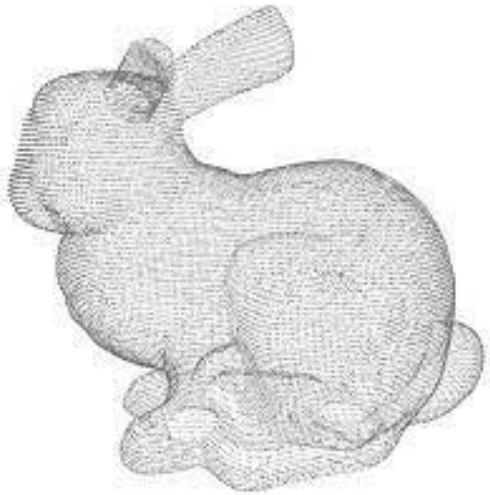
Locally Supported
Filters

Natural
Downsampling:
Multi-Scale
Analysis

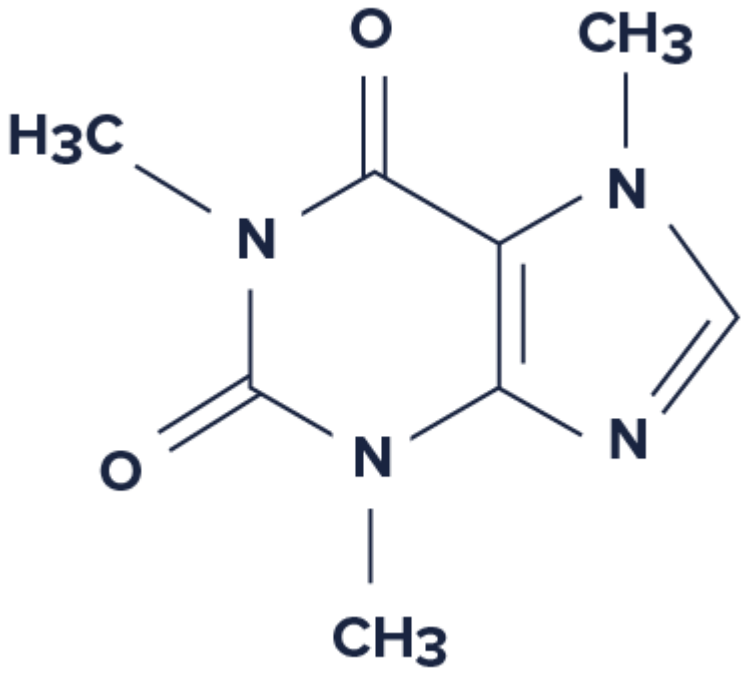
What about irregular structures?



Mesh (Graph)



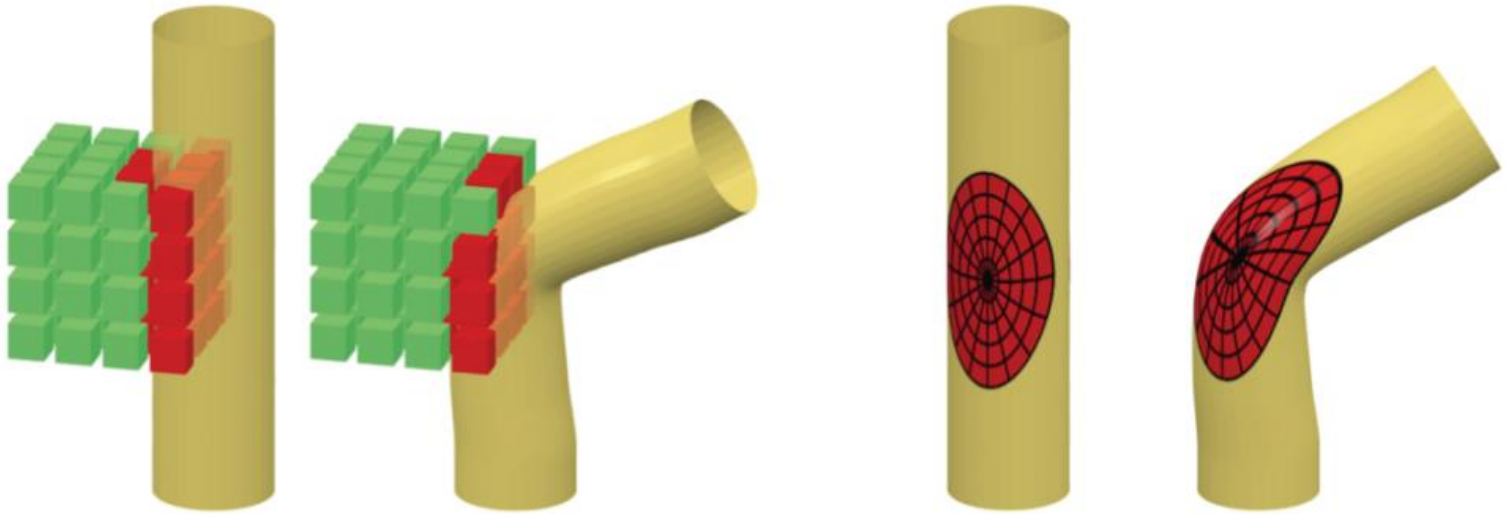
Point Set



Molecule (Graph)

What about irregular structures?

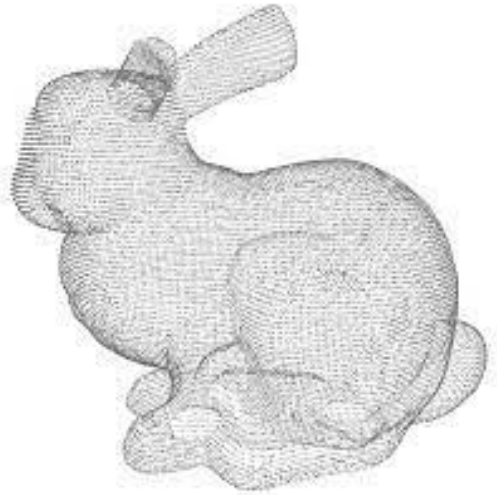
- Surface geometry: 2D surfaces in 3D space
- Operate on surface structures only?
  - Sparse convolutions
  - Other operators?



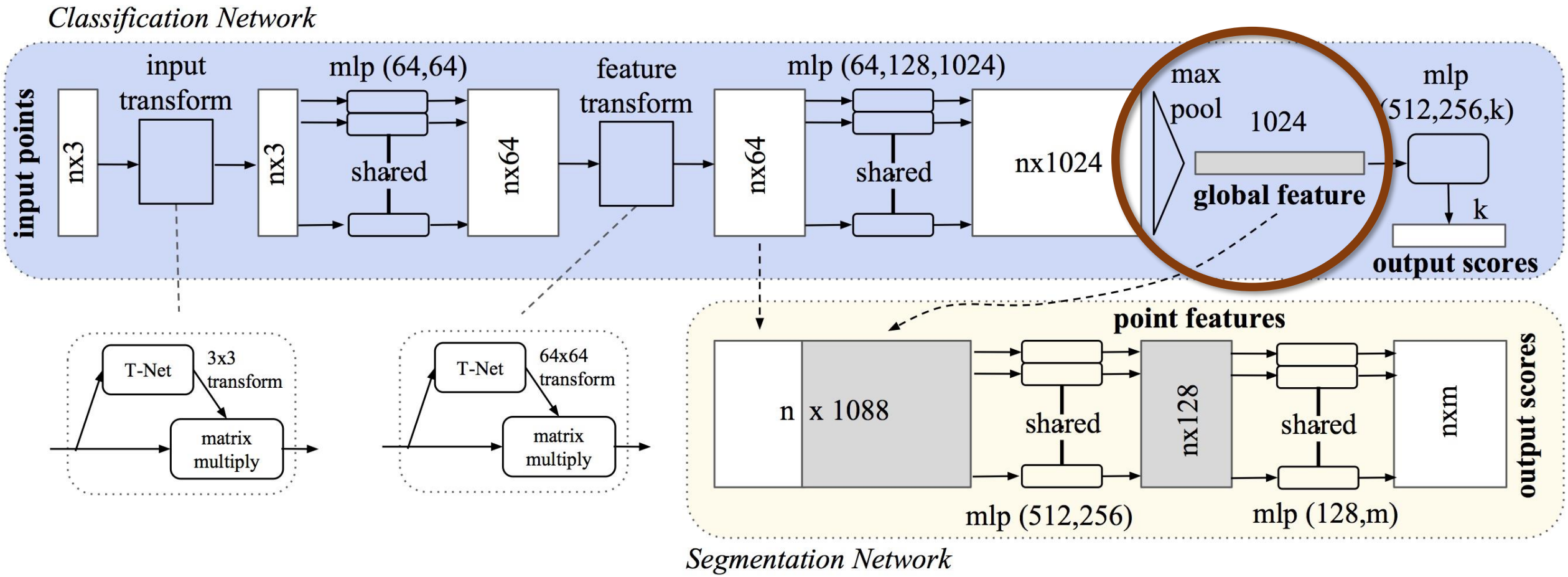
[Boscaini et al. ‘16]

Point Clouds

- Efficiency: represents surface points only
- Irregular: unordered, not tied to grid structure
- Often found from raw sensor measurements (e.g., LIDAR)



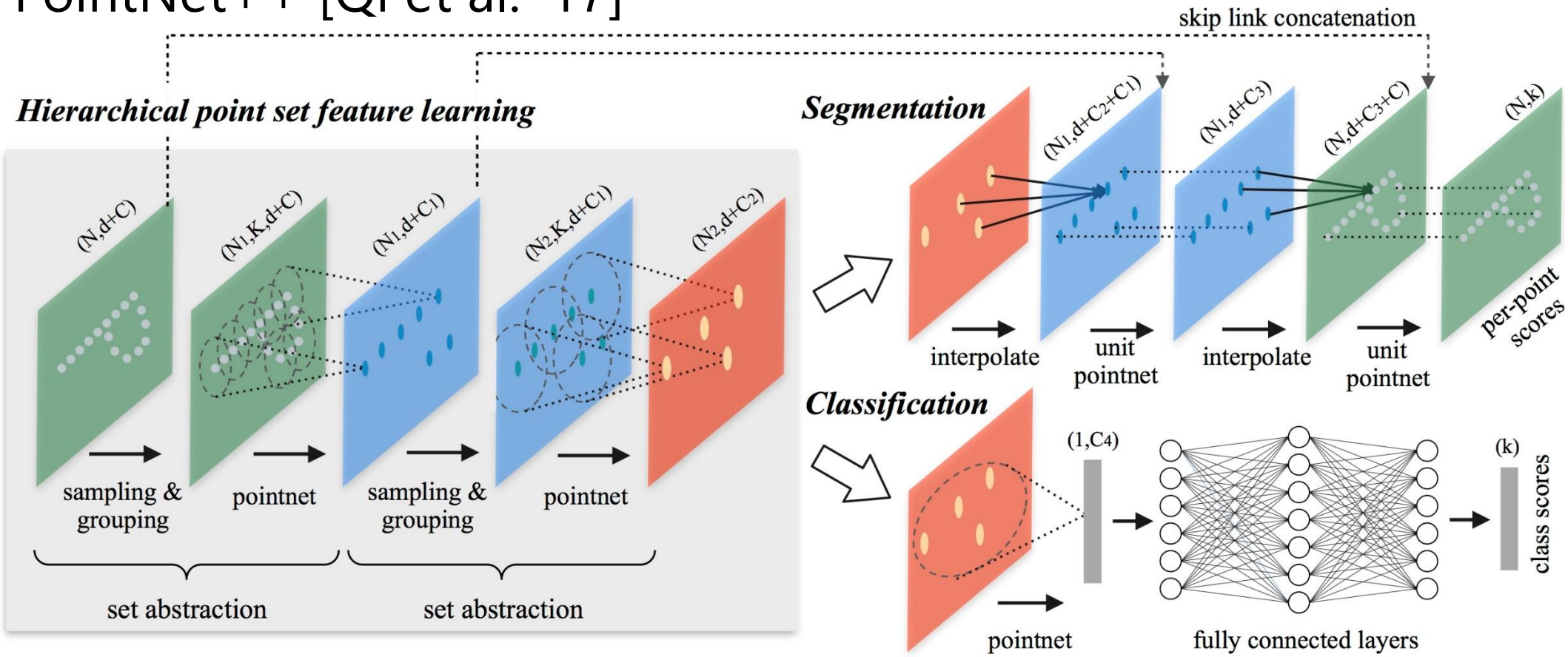
Recall: PointNet



[Qi et al. '17]

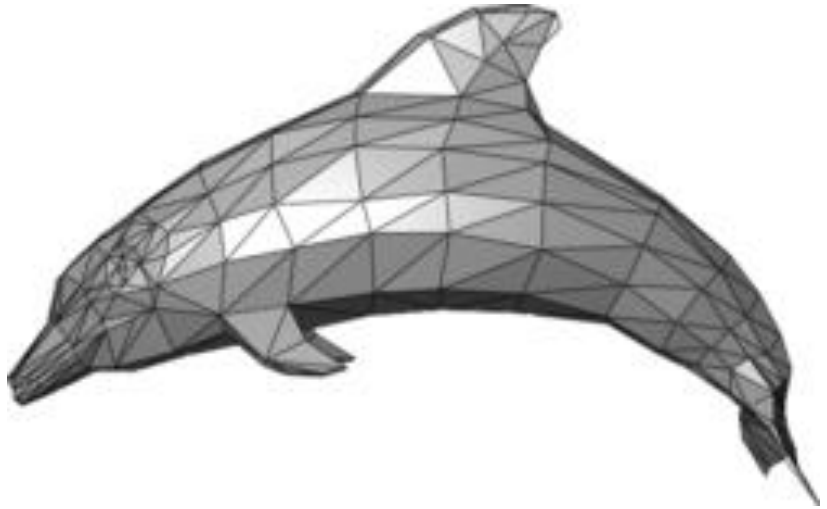
PointNet + Local Neighborhood Structure

- PointNet++ [Qi et al. '17]



Meshes

- Collection of vertices, edges, and faces
- Defines surface geometry



- Widely used in practical applications

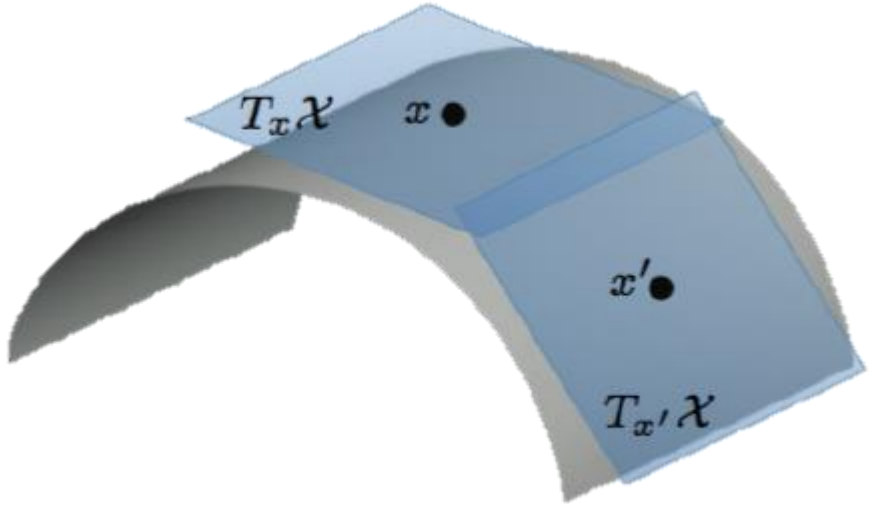
- Interpret meshes as graphs
- How to define a convolution over a graph?

How to define convolution over a mesh?

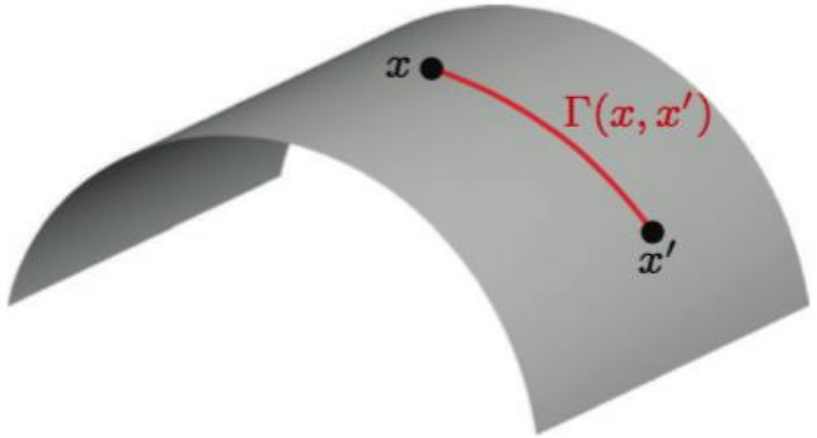
- Ideally:
  - Convolution kernel has local support
  - Weight sharing across different coordinates
  - Enable multi-scale analysis
  - Generalizability across different connectivity structures, number of vertices, edges, etc.

Geometric Operators

- Tangent plane

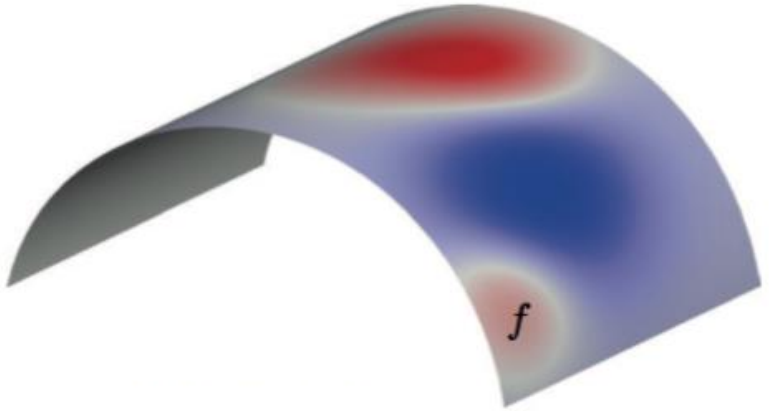


- Geodesic

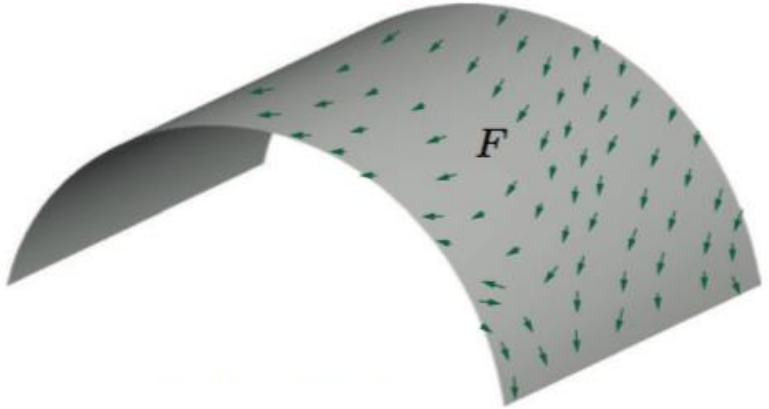


Geometric Operators

- Surface: S
- Scalar field: $f: S \to \mathbb{R}$

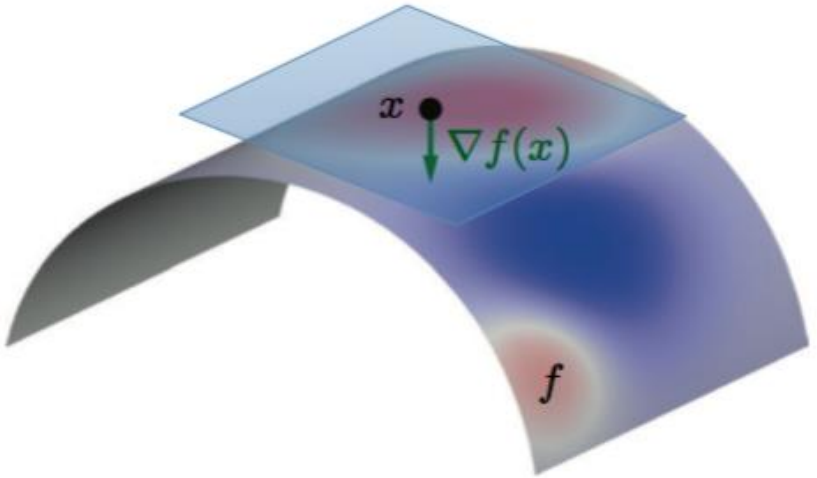


- Vector field: $F: S \to TS$

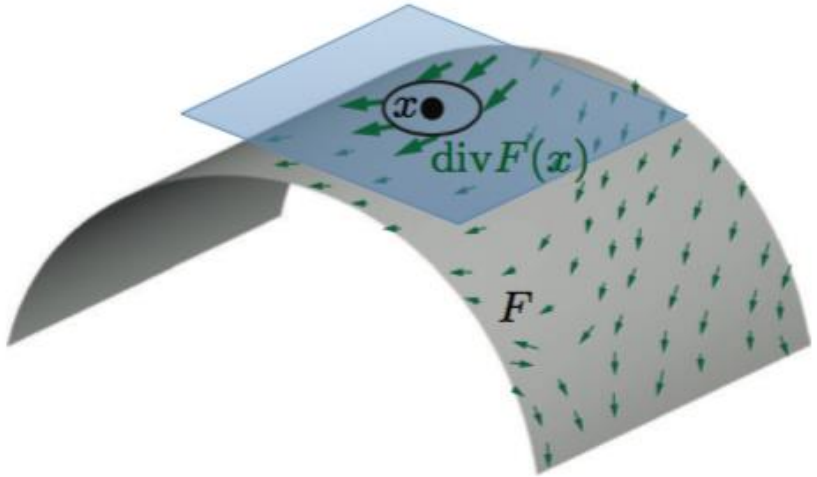


Geometric Operators

- Surface: S
- Scalar field: $f: S \to \mathbb{R}$
  - Gradient operator



- Vector field: $F: S \to TS$
  - Divergence operator



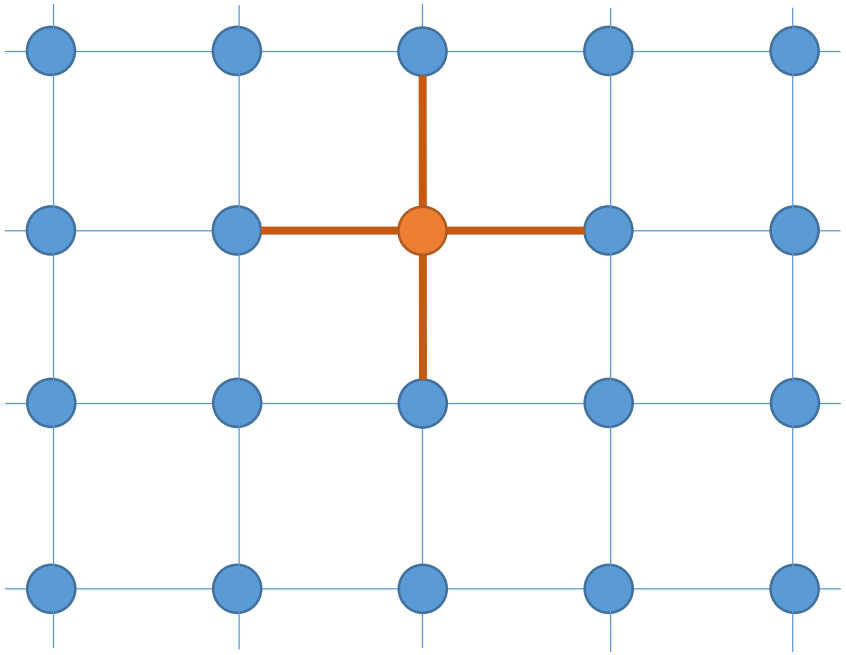
Geometric Operators: Laplacian

- Gradient: $\nabla f(x)$
  - Direction of steepest increase of f at x
- Divergence: $\mathrm{div}\big(F(x)\big)$
  - Density of an outward flux of F from an infinitesimal volume around x
- Laplacian: $\Delta f(x) = -\mathrm{div}(\nabla f(x))$
  - Difference between $f(x)$ and the average of f on an infinitesimal sphere around x

Discrete Laplacian

In 1-D:

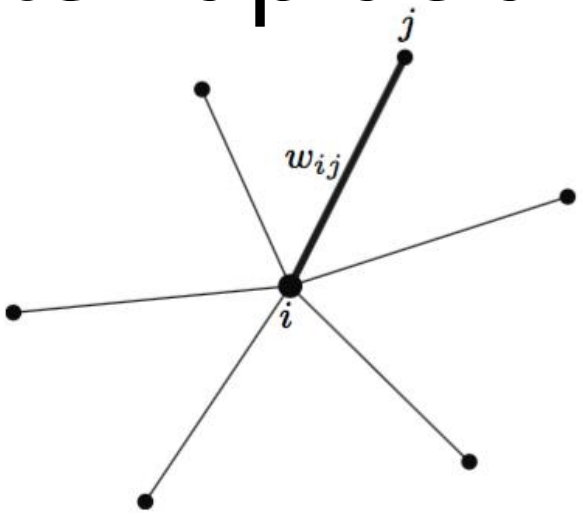
$$(\Delta f)_i \approx 2f_i - f_{i-1} - f_{i+1}$$



In 2-D:

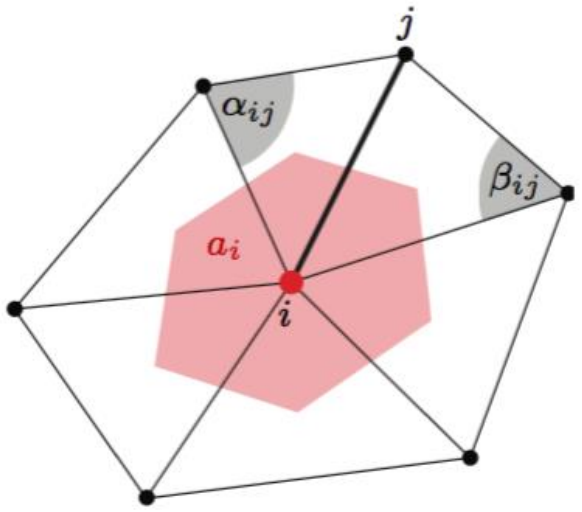
$$(\Delta f)_{i,j} \approx 4f_{i,j} - f_{i-1,j} - f_{i+1,j} - f_{i,j-1} - f_{i,j+1}$$

Discrete Laplacian



Undirected graph (V, E)

$$(\Delta f)_i \approx \sum_{(i,j)\in E} w_{ij}(f_i - f_j)$$



Triangular mesh (V, E, F)

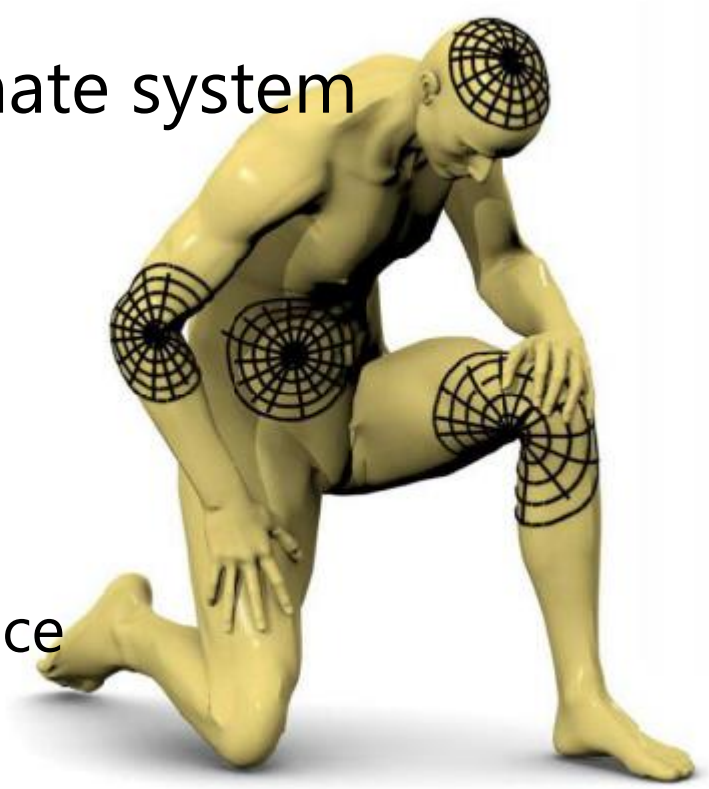
$$(\Delta f)_i \approx \frac{1}{a_i} \sum_{(i,j)\in E} \frac{\cot \alpha_{ij} + \cot \beta_{ij}}{2}(f_i - f_j)$$

$a_i =$ local area element

Relation to Fourier analysis: Graph Fourier transform $\widehat{f}$ of f on vertices as the expansion of f in terms of the eigenvectors of the graph Laplacian

Geodesic CNN

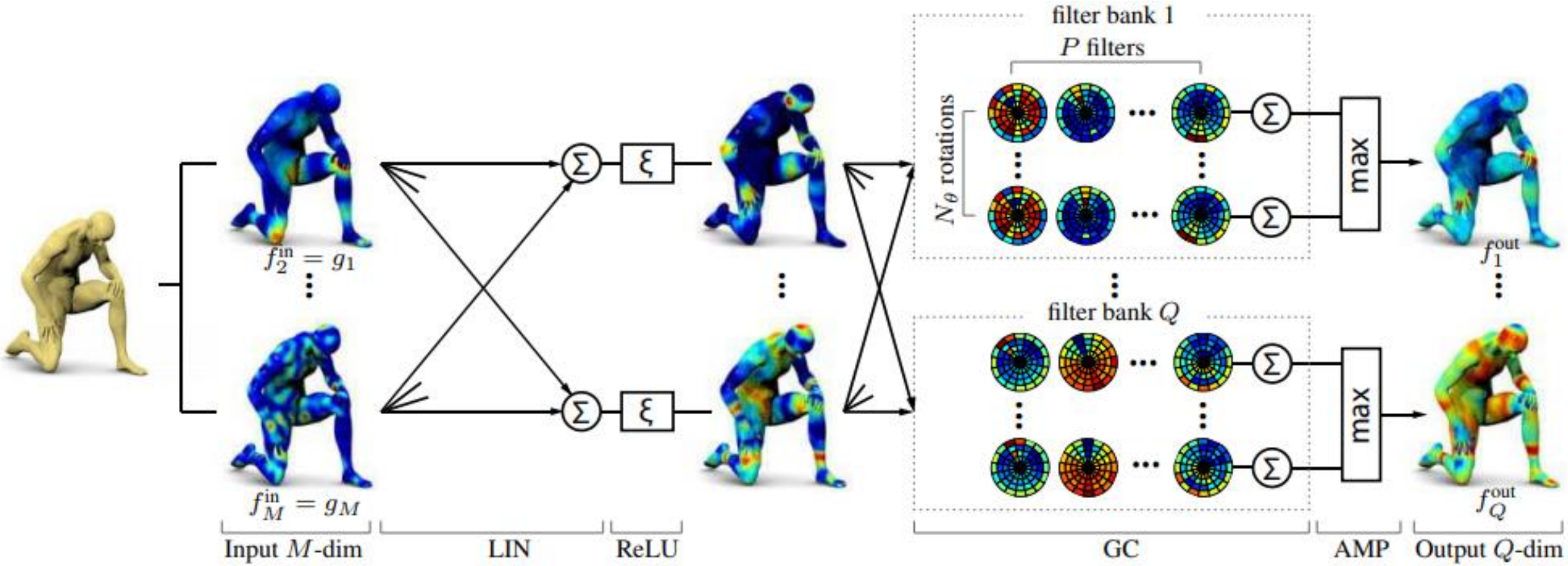
- Convolutions based on local geodesic coordinate system
- Geodesic polar coordinate system
- Extract a small patch at each point x
  - Radial coordinate ρ: geodesic distance (truncated)
  - Angular coordinate θ: direction of geodesic distance



- Geodesic convolution: apply filter to local geodesic patches
  - Rotation ambiguity: use N_θ rotations of the filter

[Masci et al. '15]

Geodesic CNN



[Masci et al. '15]

Geodesic CNN

- In practice:
  - Computing the local patches relies on a fast marching-like procedure that requires a triangle mesh
  - Choice of radius of geodesic patches
  - No natural pooling operation; relies on convolutions only to increase receptive field

[Masci et al. ‘15]

Spectral Graph Convolutions

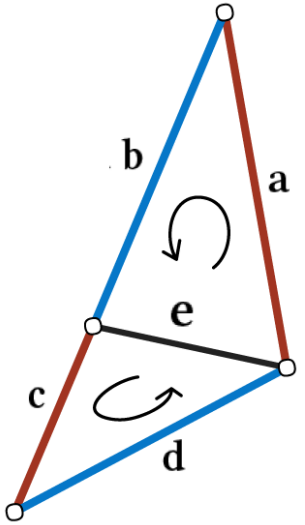
- Convolution Theorem: Convolution in the spatial domain is multiplication in the frequency domain
- Transform to frequency domain by eigenvectors of graph Laplacian
  - In practice, use a few eigenvectors
- Multiply with kernel for convolution

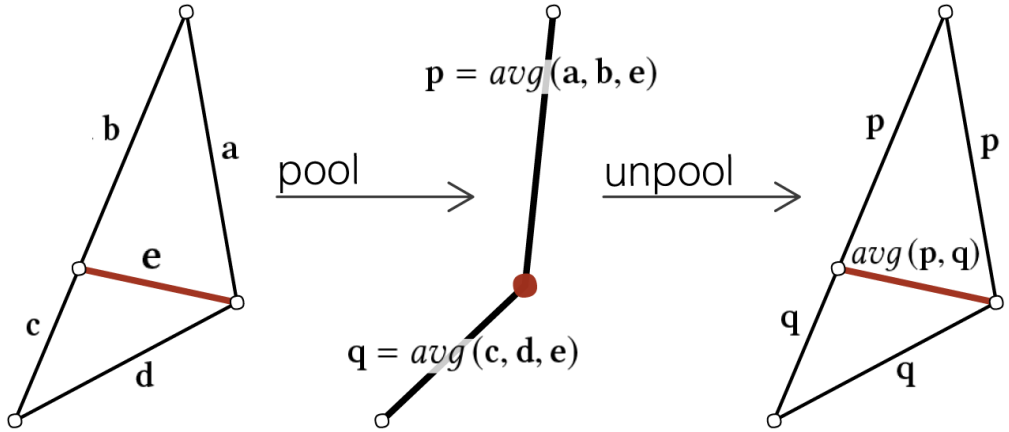
Spectral Graph Convolutions

- Considerations:
  - No guarantee that filters will have local support on the graph
  - No shift-invariance for convolution kernels
  - No natural pooling operators
  - Filter weights depend on Fourier basis; difficult to generalize to new domains

MeshCNN

- A CNN for triangle meshes
- Define convolution and pooling operators on mesh edges
- Each edge has a feature
- Each edge has four edge neighbors (from 2 incident faces)
- Convolution filters applied to each ed its 4 neighbors
- Pooling by edge-collapse

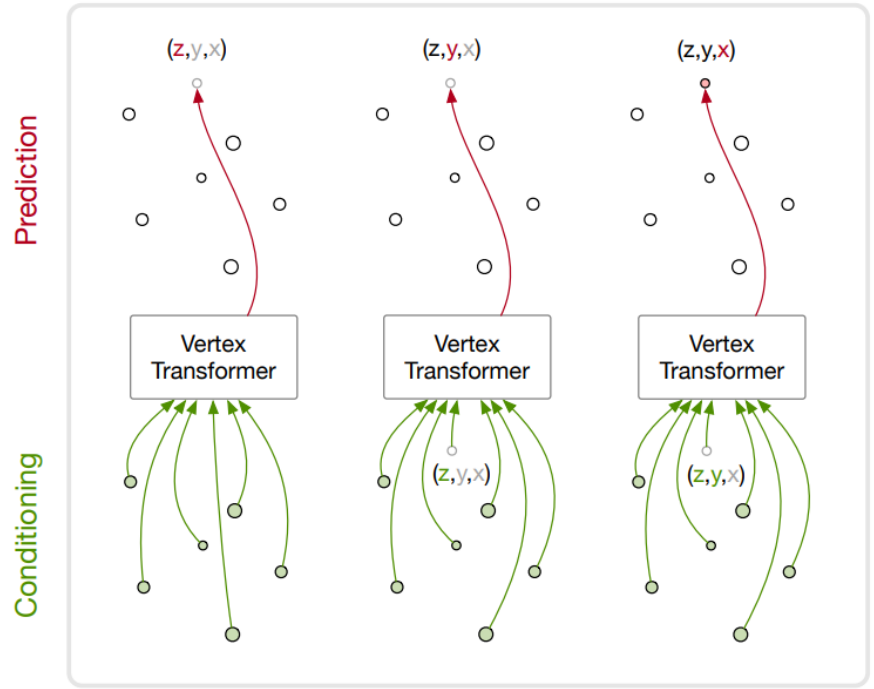


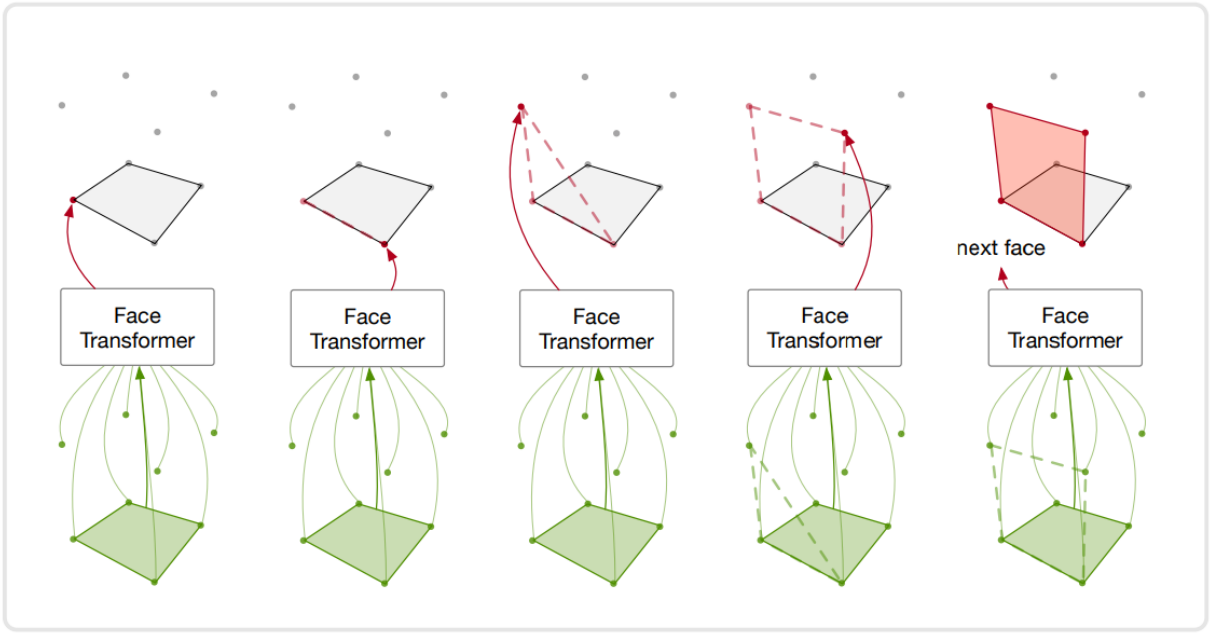


[Hanocka et al. '19]

Polygon Mesh Generation: PolyGen

- Autoregressive generation of vertices, then faces conditioned on vertices





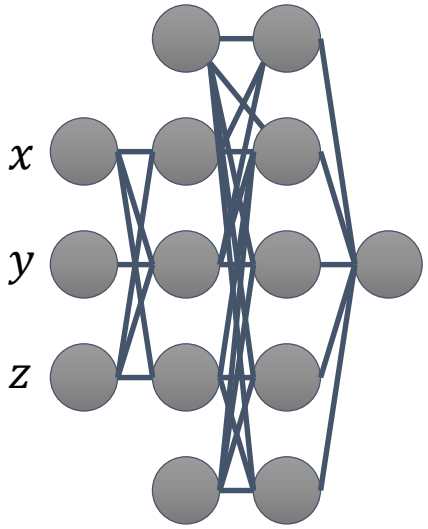
[Nash et al. '20]

Coordinate-field Models for Shapes

- Recall: implicit surfaces

$$S = \{x \in \mathbb{R}^3 \mid f(x) = 0\}$$

- What if we model f as a deep neural network?



f

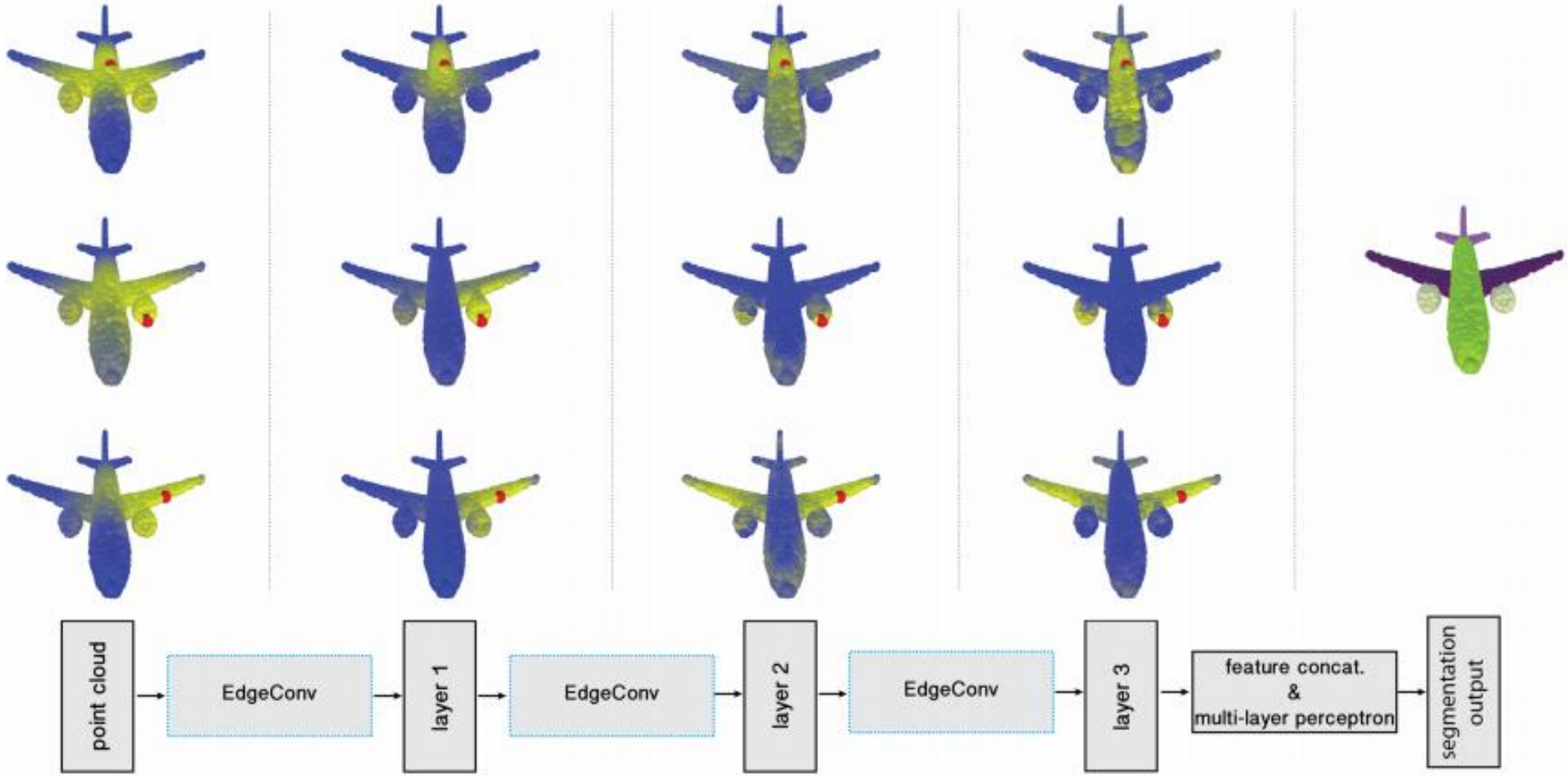
Extract a mesh representation with Marching Cubes!

Must sample f many times – MC grid resolution

Combining Representations

- Leverage different data modalities (e.g., color, geometry)
- Exploit different advantages of various representations

Dynamic Graph CNN for Learning on Point Clouds



Dynamic Graph CNN for Learning on Point Clouds

- Compute local neighborhood graph for point in a point cloud
  - k-nn graph
- Apply convolution on graph
  - For a vertex x_i: aggregation of features of neighboring vertices x_j in the graph
  - Use features from $x_i, x_j - x_i$ to capture more shape structure
- Re-compute new local neighborhood graph based on new features

2D-3D Projection

- Concepts:
  - World space: global coordinate system of 3D scene
  - Camera space: coordinate system relative to the camera
  - Screen space: pixel coordinates (+z depth)

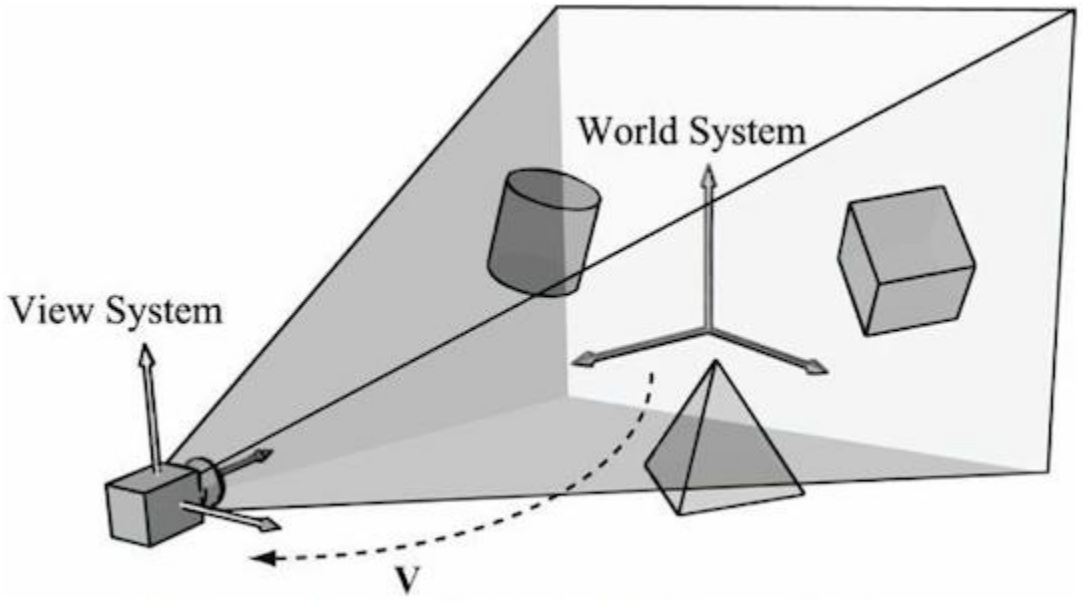
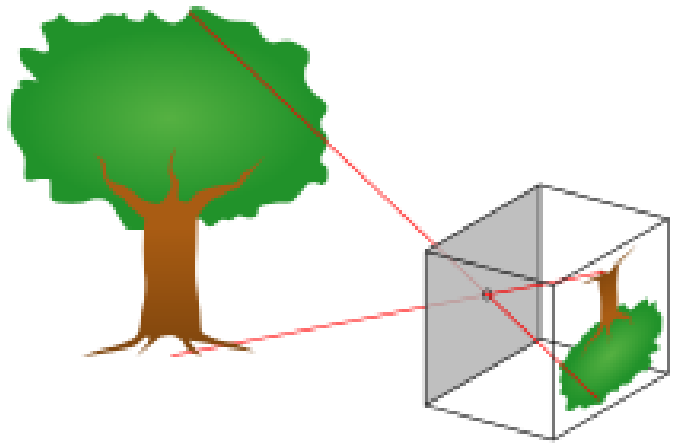


Figure 5.19. Convert the coordinates of vertices relative to the world space to make them relative to the camera space.

http://ycpcs.github.io/cs470-fall2014/labs/lab07-2.html

2D-3D Projection

- (Simple) Pinhole camera model:
  - Focal length f_x, f_y
  - Principal point m_x, m_y



Camera Intrinsic Matrix: $K = \begin{pmatrix} f_x & 0 & m_x \\ 0 & f_y & m_y \\ 0 & 0 & 1 \end{pmatrix}$

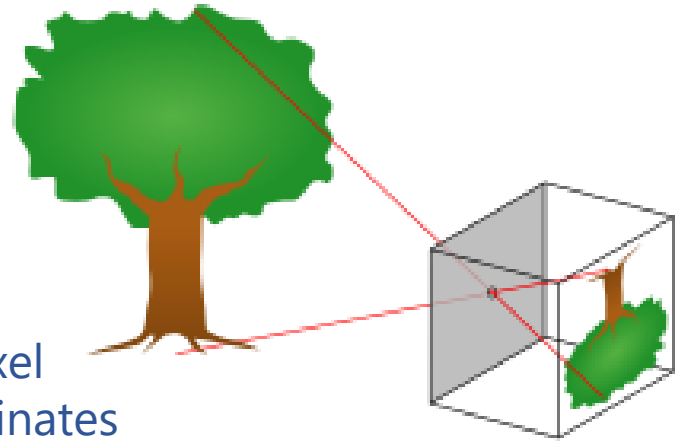
2D-3D Projection

- Apply projection

camera space

$$\begin{pmatrix} f_x & 0 & m_x \\ 0 & f_y & m_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_c \\ y_c \\ z_c \end{pmatrix} = z_c \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$$

pixel coordinates



$$K \begin{pmatrix} x_c \\ y_c \\ z_c \end{pmatrix} = z_c \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$$

2D-3D Projection

- From world space
  - R, t: rotation, translation from world space to camera space

$$\begin{pmatrix} f_x & 0 & m_x \\ 0 & f_y & m_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} R & t \\ 0 & 0 \end{pmatrix} \begin{pmatrix} x_w \\ y_w \\ z_w \end{pmatrix} = z_c \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} R & t \\ 0 & 0 \end{pmatrix} \begin{pmatrix} x_w \\ y_w \\ z_w \end{pmatrix} = \begin{pmatrix} x_c \\ y_c \\ z_c \end{pmatrix}$$

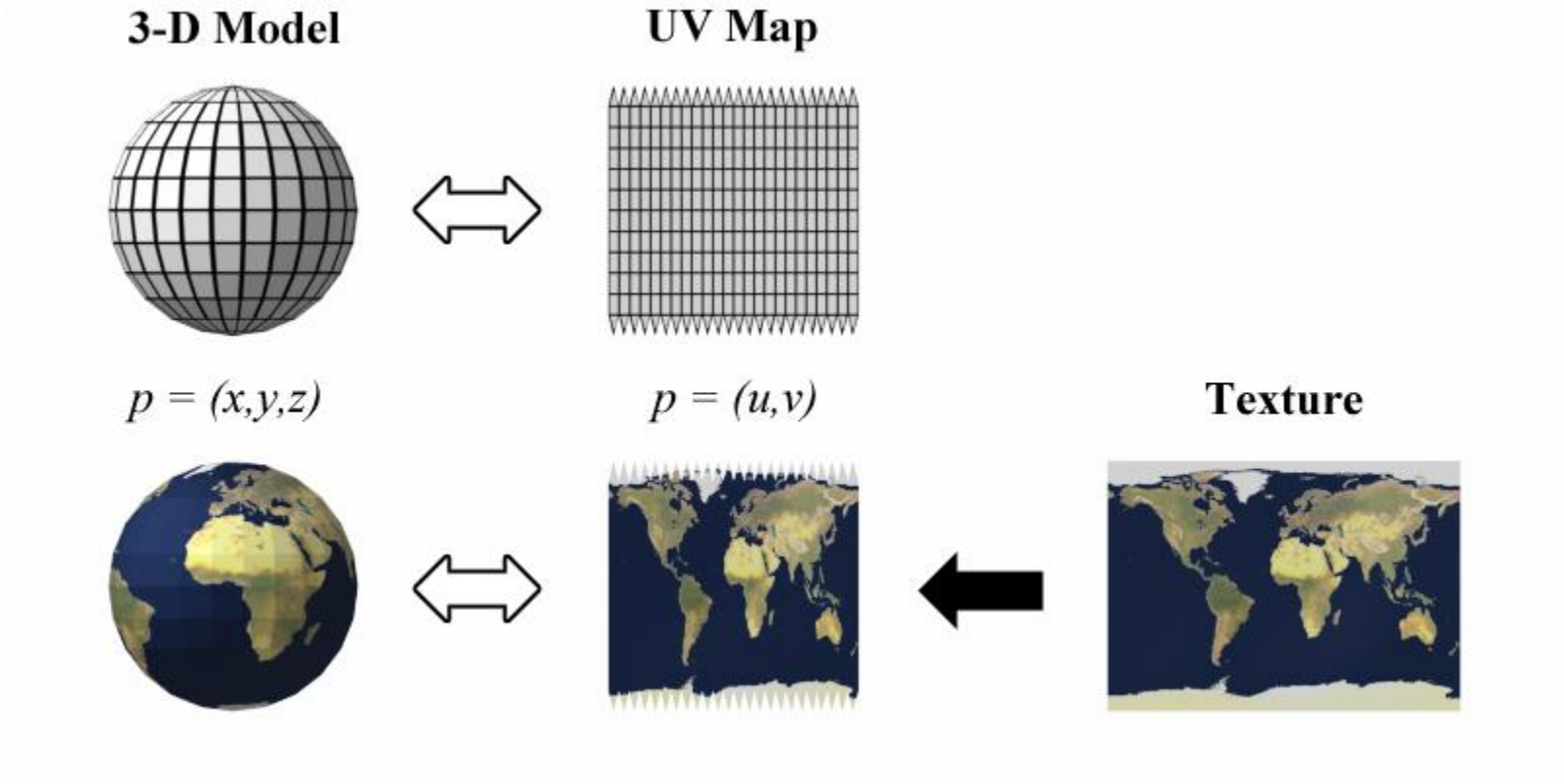
2D-3D Projection

- 3D to 2D: projection
- 2D to 3D: back-projection

- What are the differences?

- Consider relative resolution and dimensionality of representations
  - Usually 2D is higher resolution: 2D to 3D more expensive, but retain higher resolution info

2D-3D Relation: UV Mapping



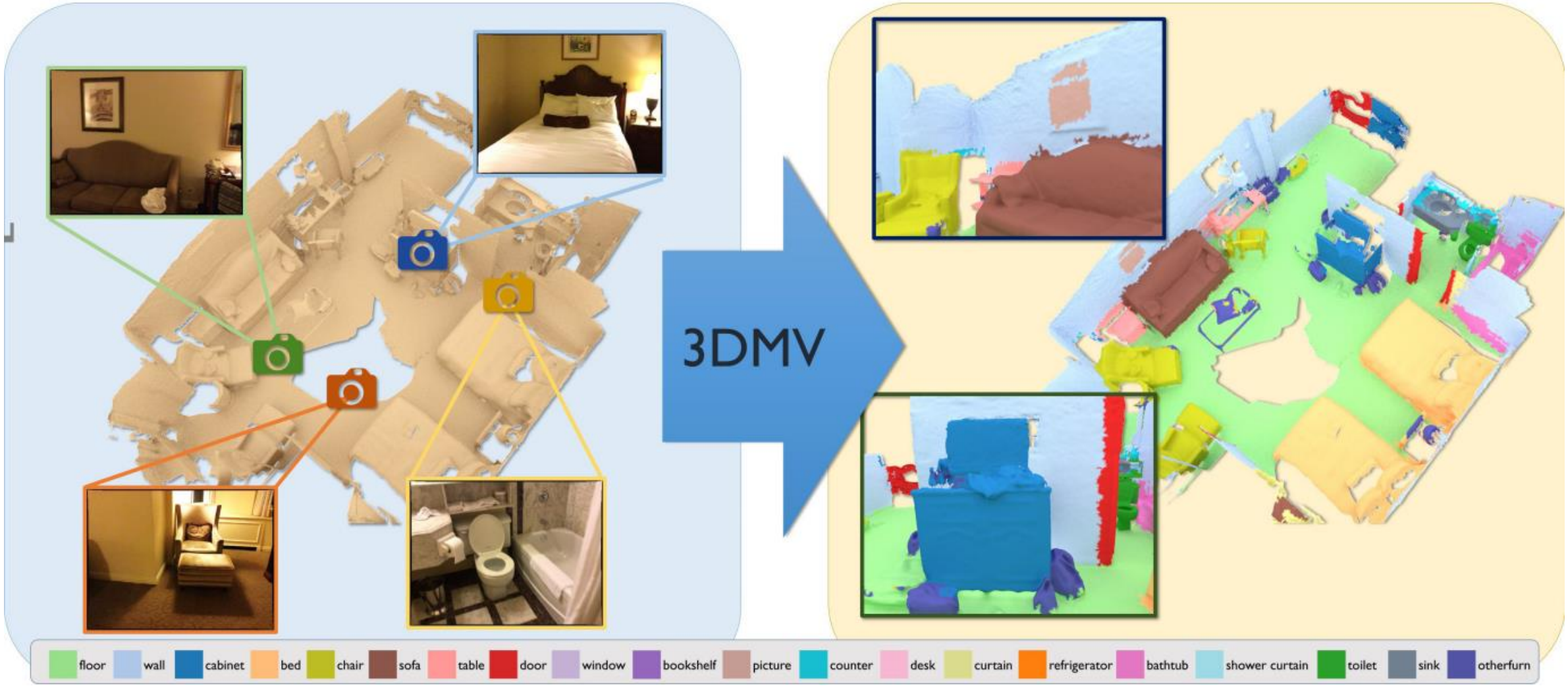
https://en.wikipedia.org/wiki/UV_mapping

2D-3D Relation: UV Mapping

- Enables high-resolution texturing on a 3D mesh
- Efficient representation to look up hi-res texture values for 3D surface locations

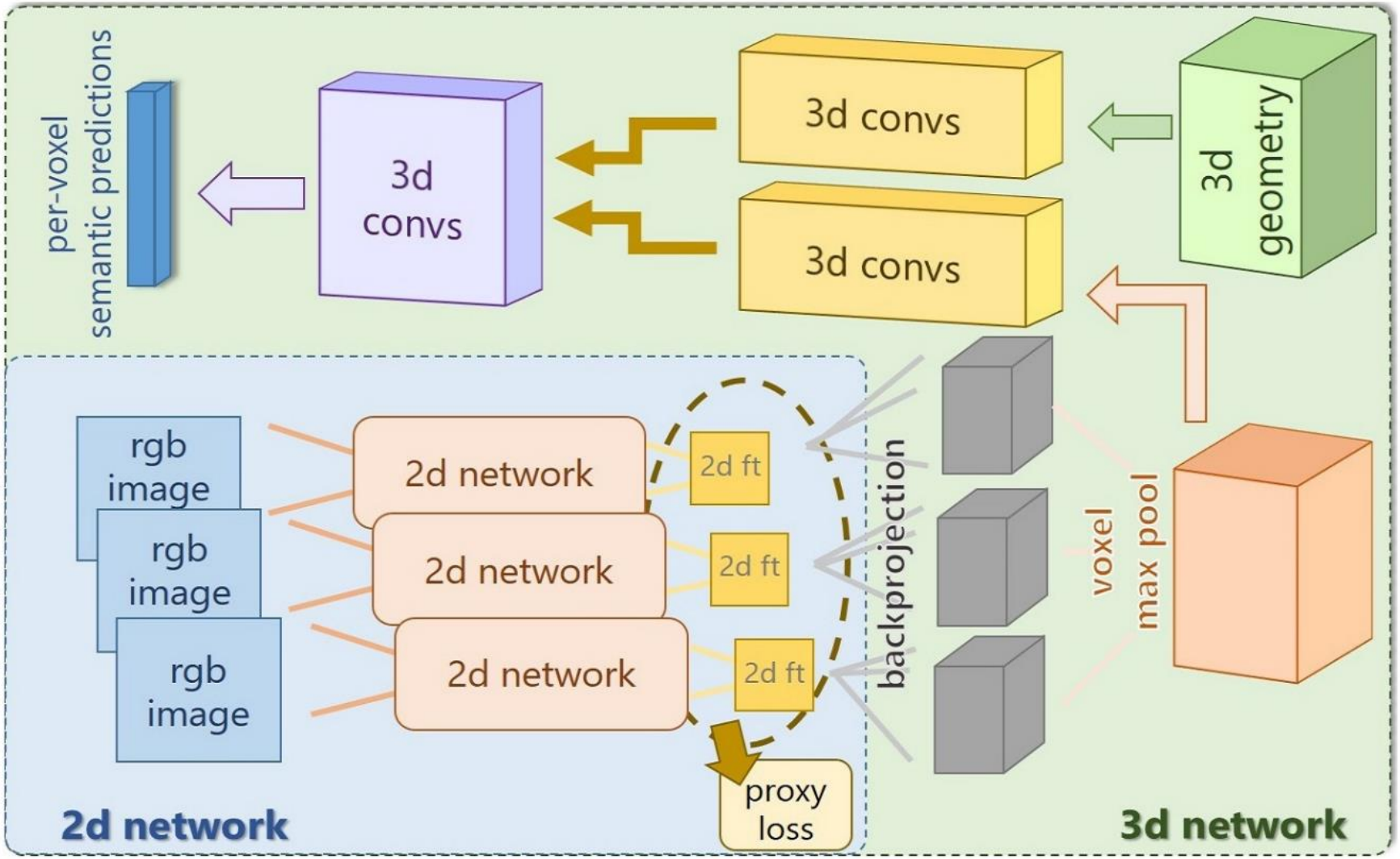
- Note: computing the mapping is often a challenging problem
  - Want to minimize distortion, seams/boundaries, etc.
  - Some existing toolsets: https://github.com/microsoft/UVAtlas

3DMV: Joint 3D-Multi-View Learning



[Dai et al. '18]

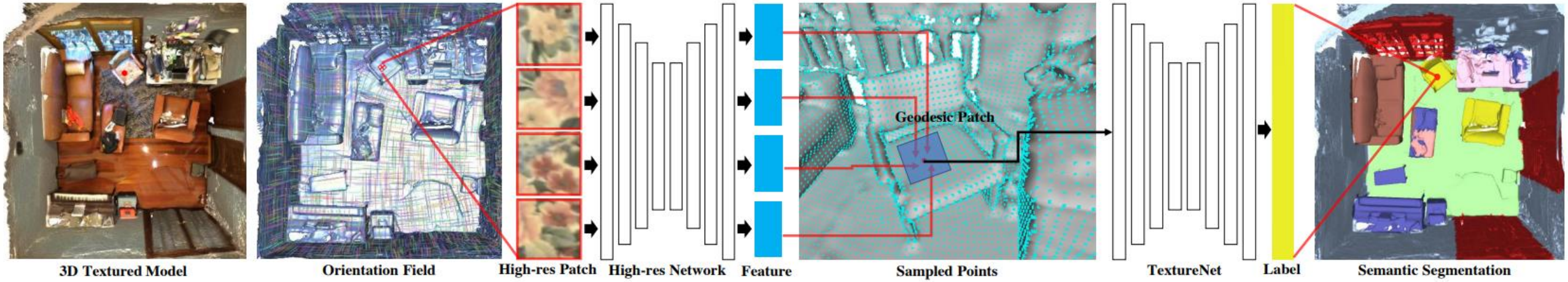
3DMV: Joint 3D-Multi-View Learning



[Dai et al. '18]

TextureNet

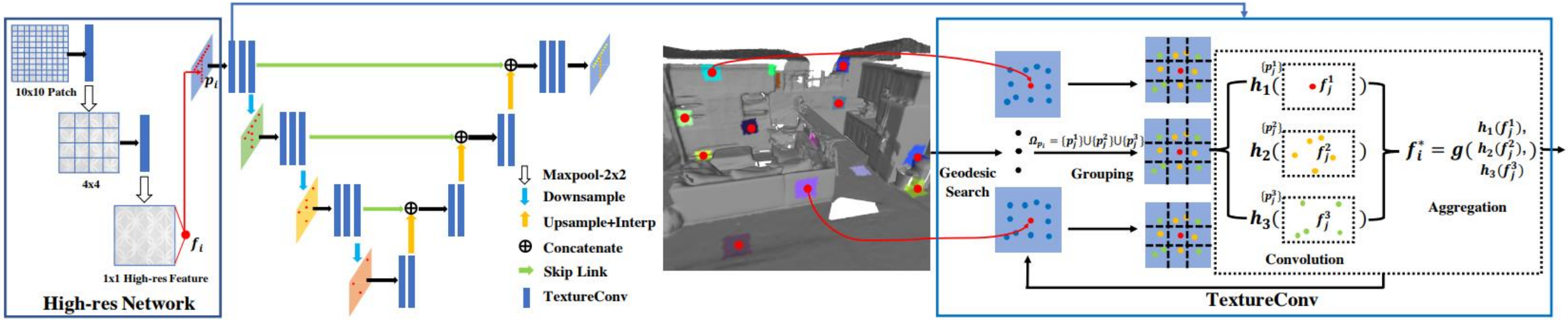
- Semantic labeling of textured meshes
- Leveraging color texture signal along with geometry



[Huang et al. '19]

TextureNet

- Semantic labeling of textured meshes
- Leveraging color texture signal along with geometry



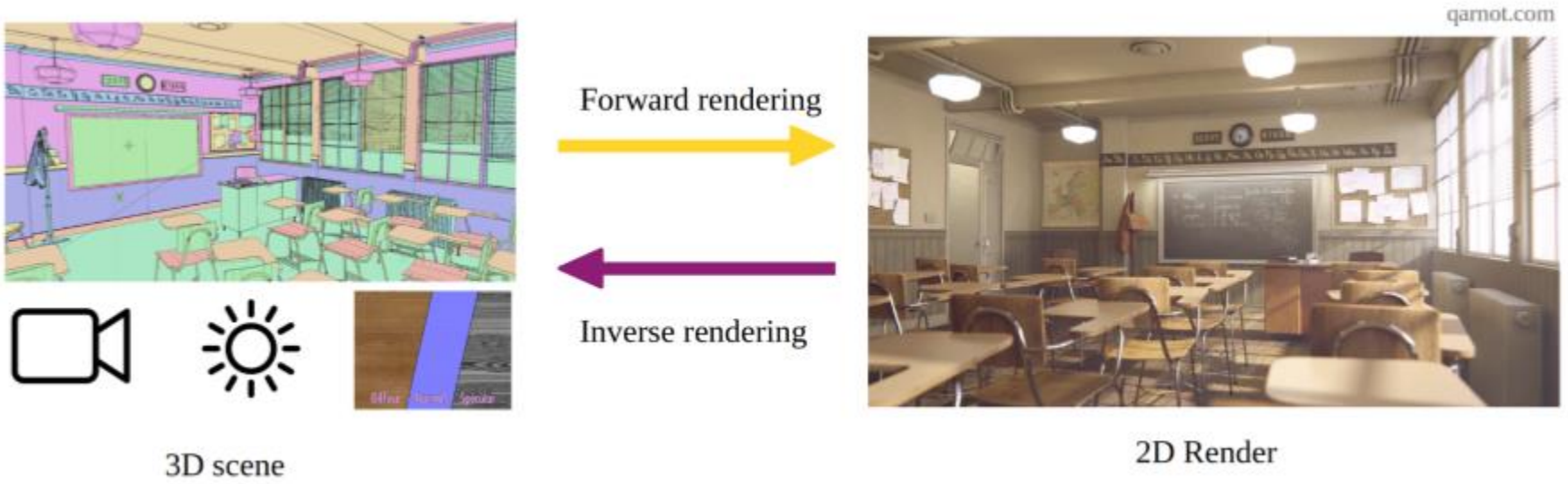
[Huang et al. '19]

Differentiable Rendering

- Bridging 3D scene representations with 2D signal

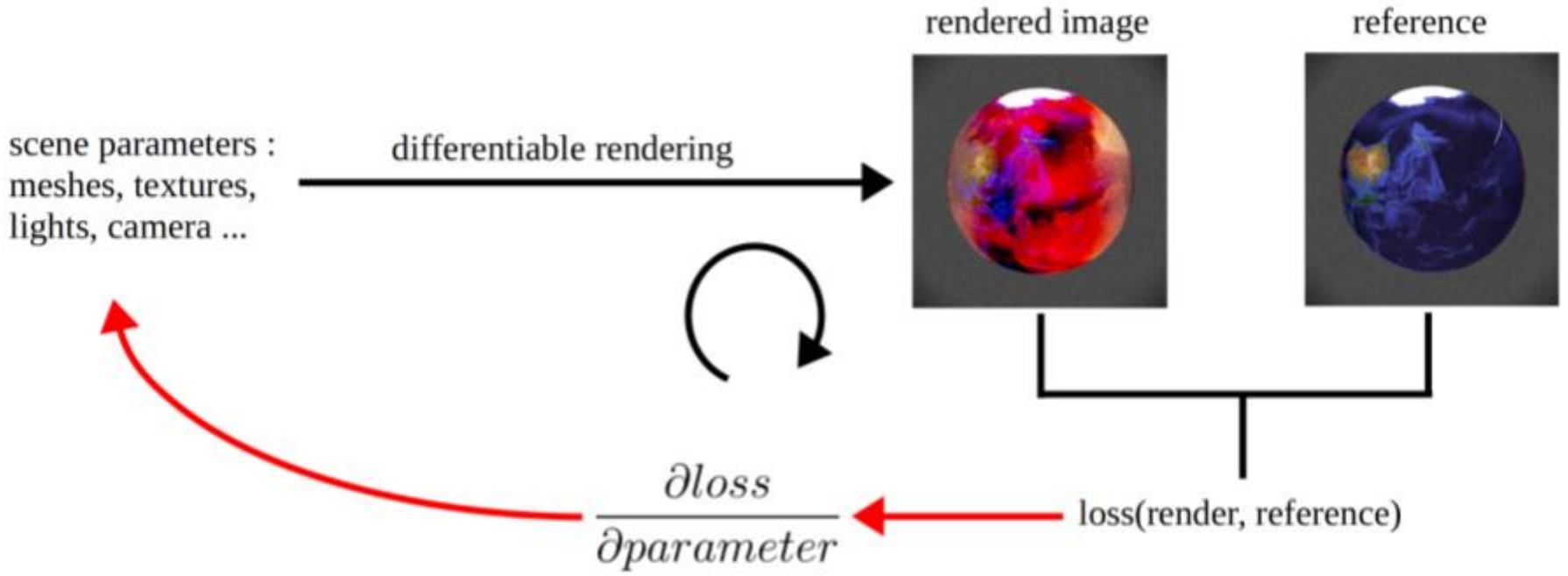
Differentiable Rendering

- Rendering process



Differentiable Rendering

- Optimize through rendering process



https://blog.qarnot.com/an-overview-of-differentiable-rendering/

Differentiable Rendering

- Bridging 3D scene representations with 2D signal
- Can relate 2D image pixels back to 3D scene

- Can use 2D losses
  - 3D data is often more limited than 2D
  - 2D images are often high-resolution and high-quality
  - Can leverage powerful 2D formulations (e.g., pre-trained VGG, StableDiffusion, etc.)

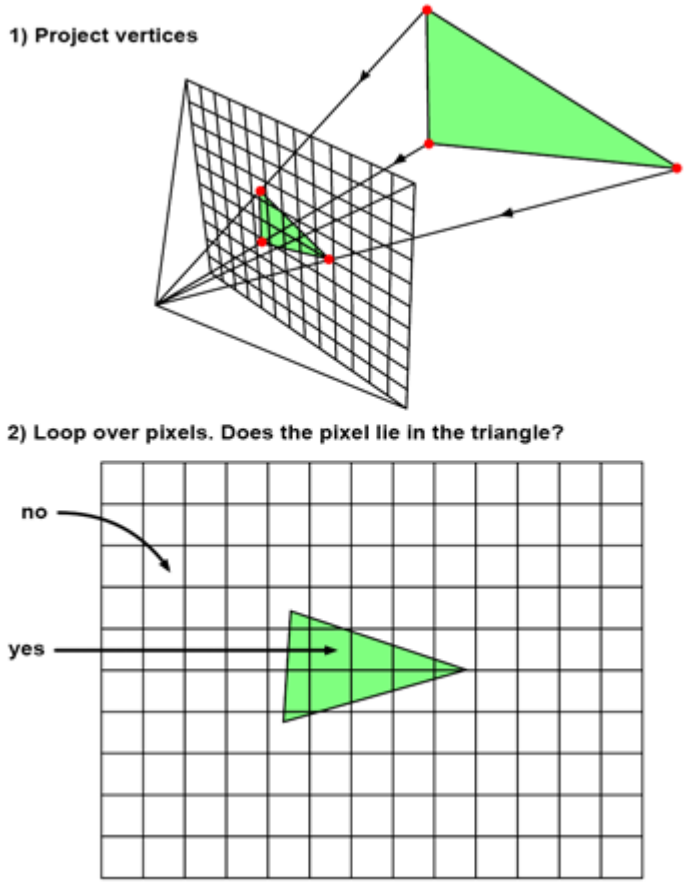
Differentiable Rendering

- How to render?
- Many techniques have been developed in computer graphics
- Consider differentiability, gradient signals

Mesh Rasterization

- Commonly used for rendering
- Developed in 1960s to early 1980s

- Iterate over mesh triangles
- For each mesh triangle:
  - Project to screen
  - Fill in pixels in triangle



Differentiable Rendering: Mesh

- What is the backward gradient?

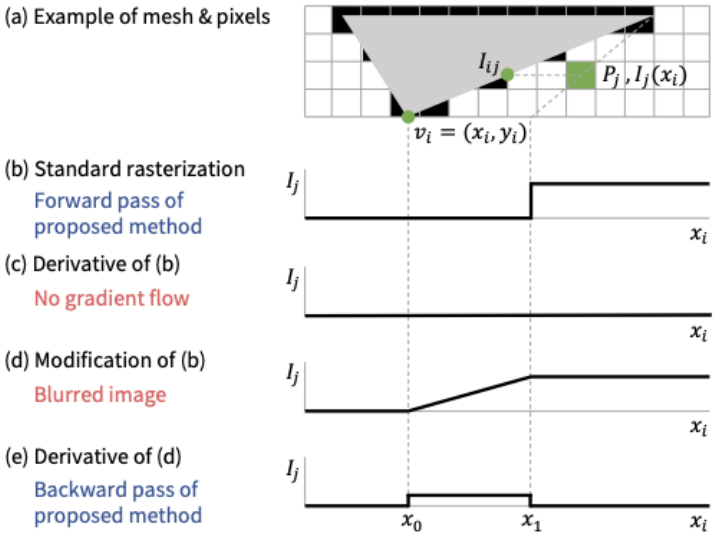


Figure 2. Illustration of our method. $\boldsymbol{v}_i = \{x_i, y_i\}$ is one vertex of the face. I_j is the color of pixel P_j. The current position of x_i is x_0. x_1 is the location of x_i where an edge of the face collides with the center of P_j when x_i moves to the right. I_j becomes I_{ij} when $x_i = x_1$.

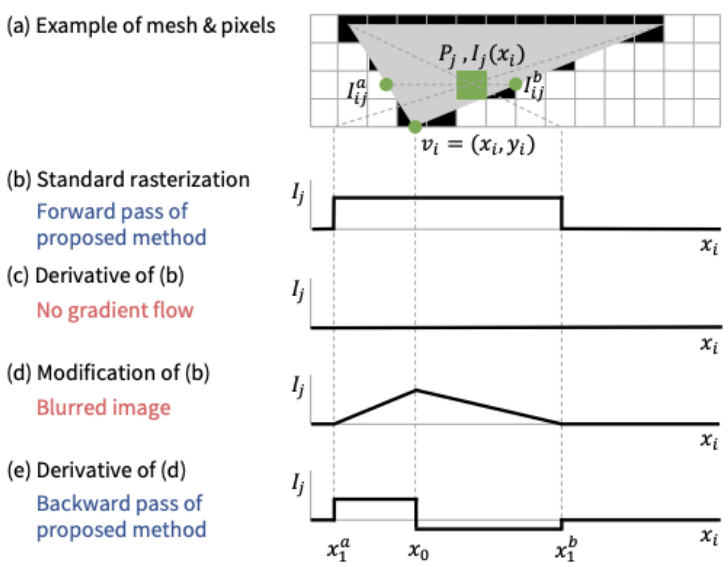
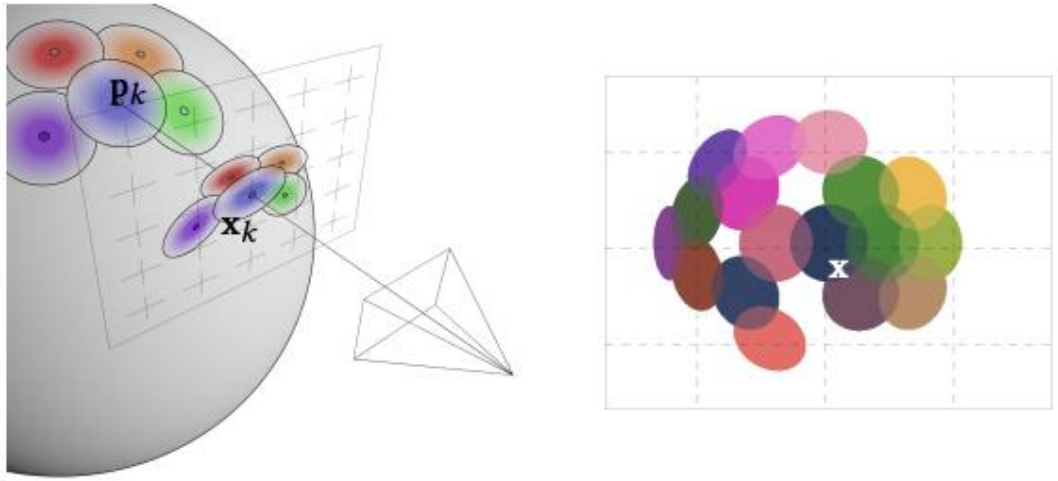


Figure 3. Illustration of our method in the case where P_j is inside the face. I_j changes when x_i moves to the right or left.

[Kato et al. '18]

Point Cloud Splatting

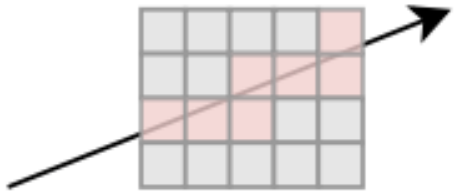
- Render each point as its sphere of influence
- Aggregate overlapping information as weighted sum of influences



Differentiable Surface Splatting for Point-based Geometry Processing, Wang et al. ToG 19
SynSin: Endto-end View Synthesis from a Single Image, Wiles et al. CVPR 20
(and many others)

Voxel Grid Rendering (Simple)

- Occupancy or SDF grid representations
- Find voxels that represent surface
- Aggregate surface voxel and neighboring voxels (e.g., trilinear interpolation) for pixel value



Coordinate Field Rendering (Simple)

- Occupancy or SDF fields
- Cast rays through field and find where surface is hit
- Intersection between ray and surface used to compute camera-to-surface distance

Differentiable Rendering

- Consider: where are gradients propagated in the formulation?

- If only on surface values in the scene representation, problem is ill-constrained
  - No gradients for empty space
  - Can use coarse proxy 3D loss to help constrain

Coordinate-field Models: Radiance Fields

- Radiance field: 5D-function representing color and density

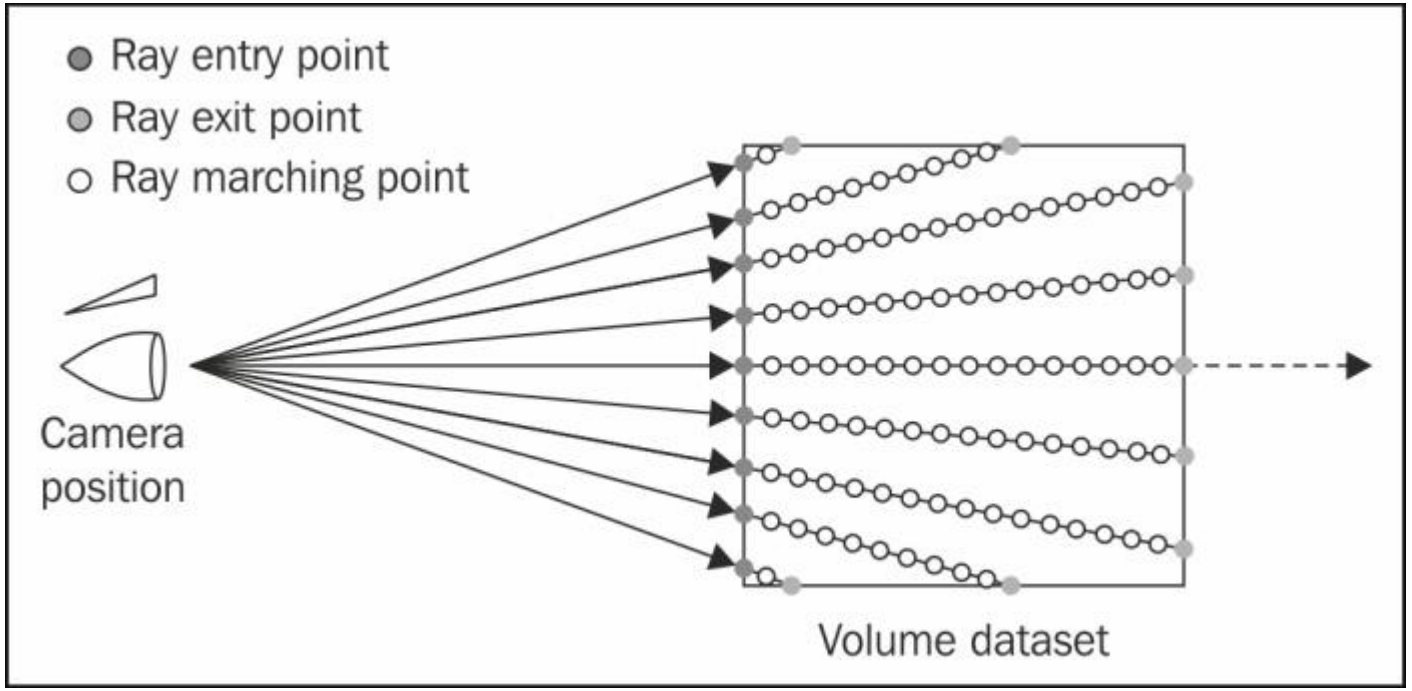
$$f(x, y, z, \theta, \phi) = (RGB, \sigma)$$

- Input: 3D location + viewing direction
- Output: color and density

- Volume rendering

Volume Rendering

- For each image pixel, cast a ray into the scene
- Accumulate the color & density values along the ray for the final pixel result



https://www.oreilly.com/library/view/opengl-build/9781788296724/ch07s03.html

Volume Rendering

- How to compose the RGB value $\hat{C}$ of a pixel?
- Given a ray with samples $i \in [1, N]$ and associated densities σ_i and colors $\boldsymbol{c}_i$,
  - Calculate Transmittance T_i (how much light is transmitted to sample i):
  - $\delta_j = t_{j+1} - t_j$: distance between adjacent samples

$$T_i = \exp\left(-\sum_{j=1}^{i-1} \sigma_j \delta_j\right)$$

Volume Rendering

- How to compose the RGB value $\hat{C}$ of a pixel?
- Given a ray with samples $i \in [1, N]$ and associated densities σ_i and colors $\boldsymbol{c}_i$,
  - Calculate how much light is contributed by sample i with distance δ_i to the next sample

$$\left(1 - \exp(-\sigma_i \delta_i)\right)$$

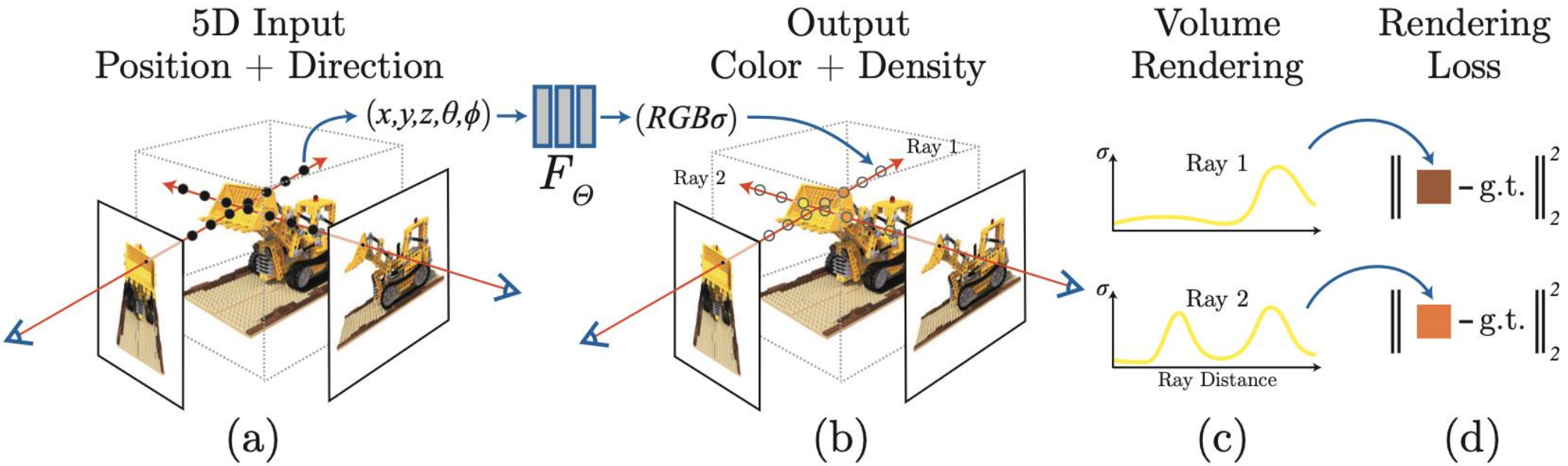
Volume Rendering

- How to compose the RGB value $\hat{C}$ of a pixel?
- Given a ray with samples $i \in [1, N]$ and associated densities σ_i and colors $\boldsymbol{c}_i$,
  - Multiply both and integrate over all samples to get the color of a pixel

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} T_i \big(1 - \exp(-\sigma_i \delta_i)\big) \mathbf{c}_i$$

NeRF: Neural Radiance Fields

- Use MLP to model radiance field



[Mildenhall et al. '20]

NeRF: Neural Radiance Fields

- Training:
  - Input: set of posed images (RGB images + camera parameters)
  - Optimize to match train RGB images when rendered from associated camera parameters
- Inference:
  - Render views from new camera parameters (novel view synthesis)

- View-dependent representation
- No constraints that density represents geometry

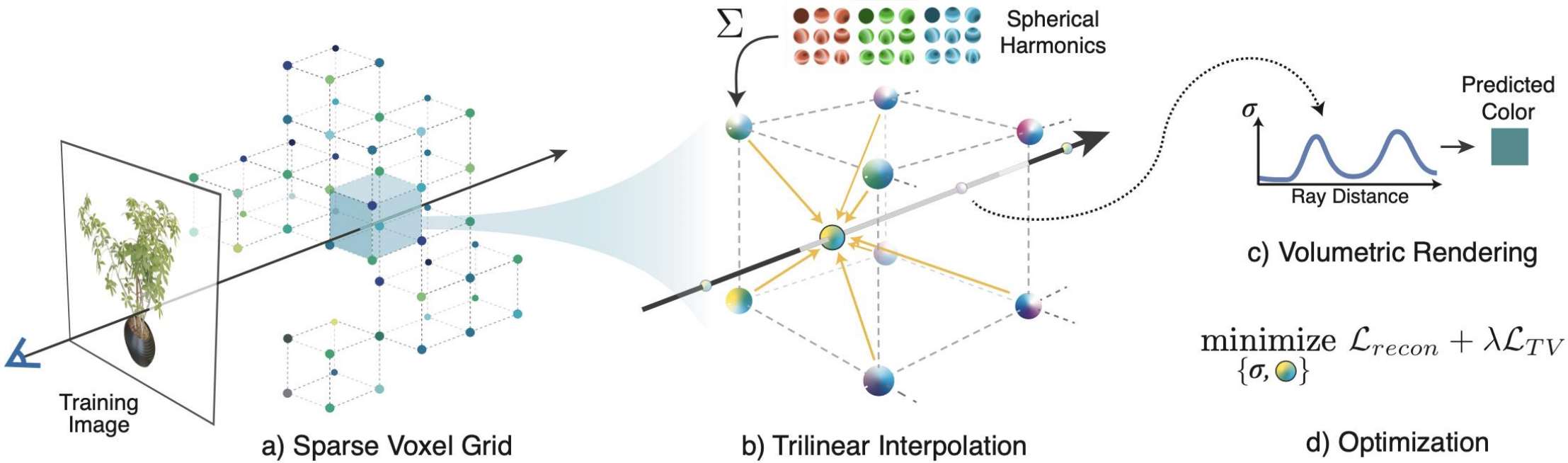
[Mildenhall et al. ‘20]

NeRF: Neural Radiance Fields

[Mildenhall et al. ‘20]

Radiance Fields on Sparse Voxel Grids

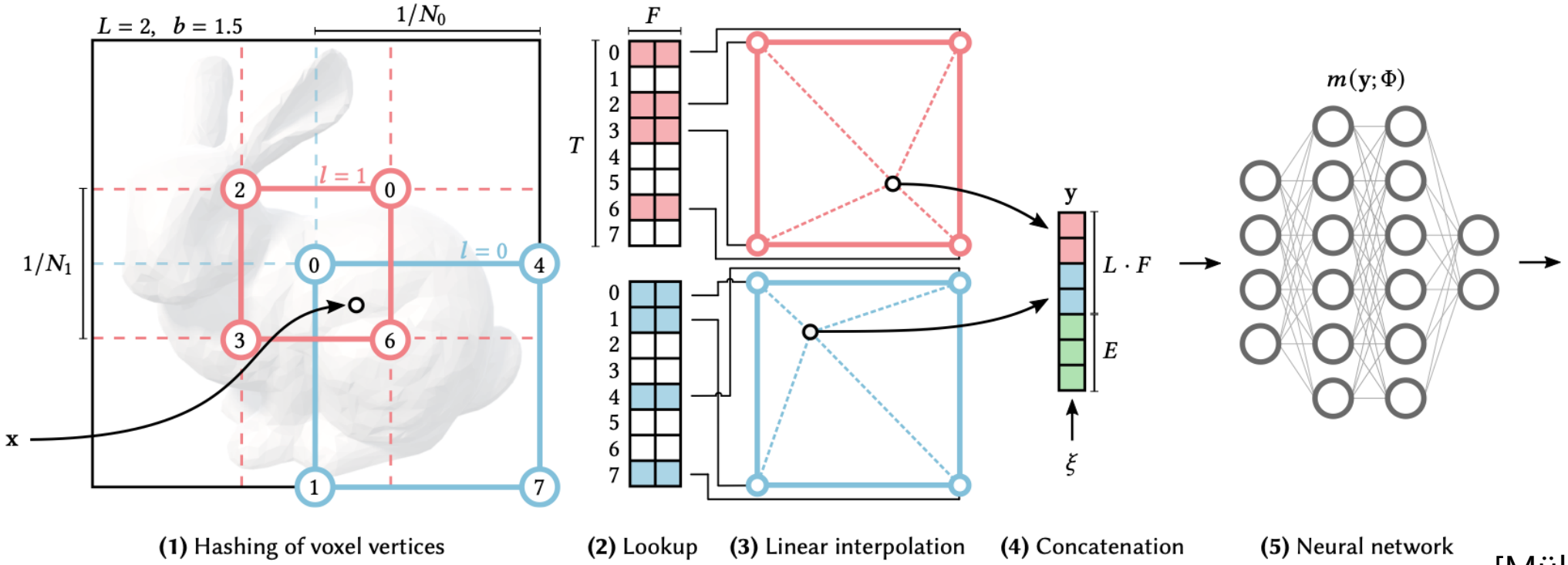
- Plenoxels: much faster optimization



[Fridovich-Keil and Yu et al. '22]

Radiance Fields on Sparse Voxel Grids

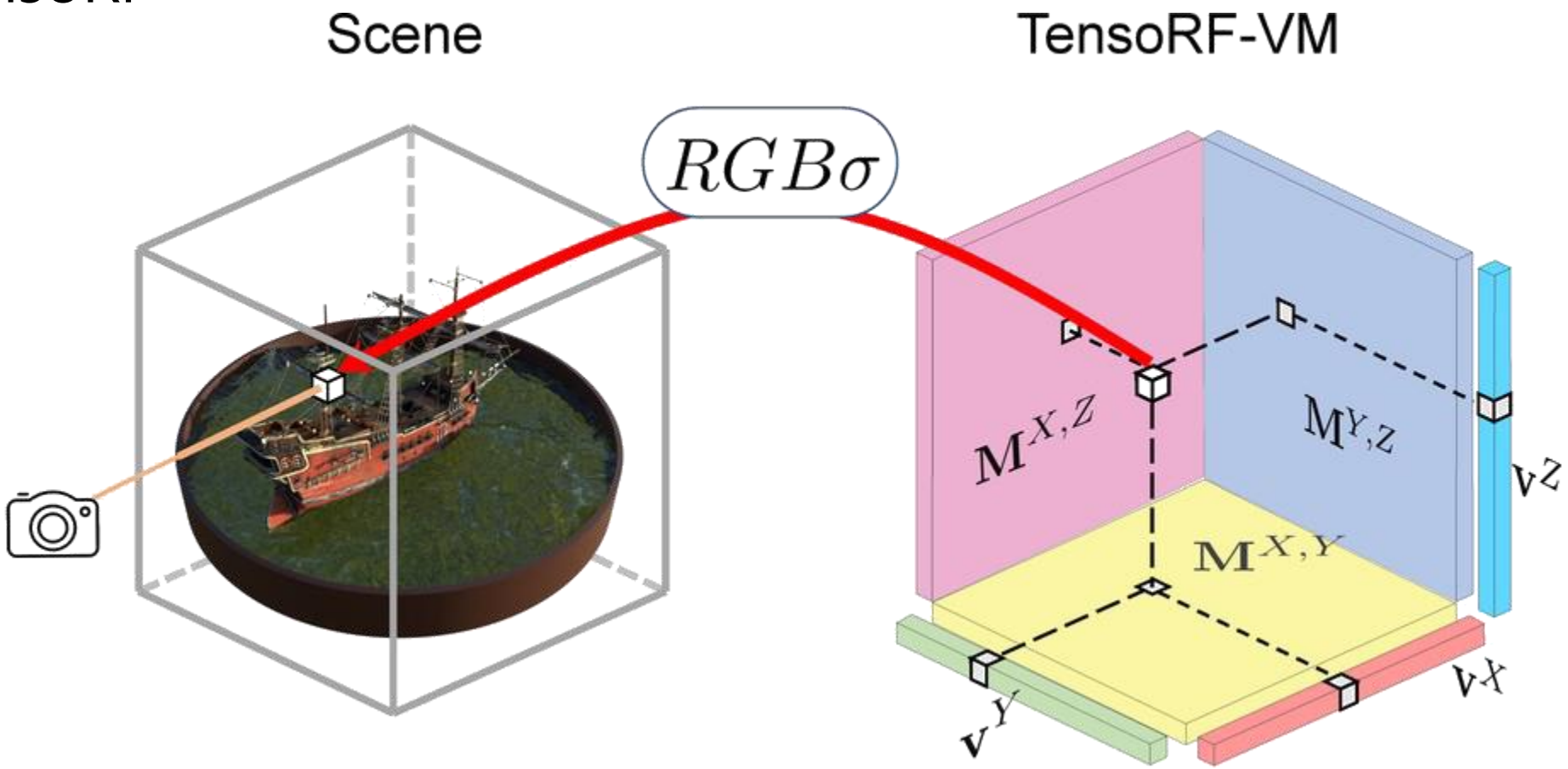
- Instant-NGP: sparse grid + 2 small MLPs (density, color)
- Very efficient optimization (seconds to minutes)



[Müller et al. '22]

Radiance Fields: Tensor Representation

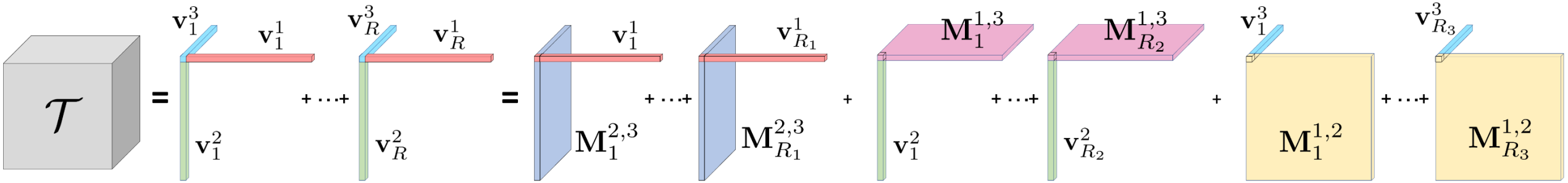
- TensoRF



[Chen and Xu et al. '22]

Factorize Radiance Fields

- TensoRF
- Efficient memory for scene representation + fast optimization



Traditional CANDECOMP/PARAFAC Decomposition:
Very compact $O(n)$; can require many components for complex scenes

Vector-Matrix Decomposition:
Each component has more parameters individually $O(n^2)$, but requires fewer components to model complex scenes

[Chen and Xu et al. '22]

Radiance Fields

- Powerful representation for modeling 3D scenes with view-dependent effects

- Requires very comprehensive, careful data capture for good results
  - Need strong view coverage to constrain new views
  - “floaters” due to ambiguities in train view constraints

Radiance Fields

- Powerful representation for modeling 3D scenes with view-dependent effects
- Requires v r good results
  - Need str
  - “floaters”

Additional References

- 3D tools and visualization:
  - Open3D: http://www.open3d.org/
- Basic 3D operators and differentiable rendering
  - Pytorch3D: https://pytorch3d.org/
- NeRF toolbox
  - Nerfstudio: https://docs.nerf.studio/en/latest/index.html