CS 598 3DV: Representations

Shenlong Wang
UIUC

ILLINOIS

Some materials borrowed from Angjoo Kanazawa and Shubham Tulsiani

Seeing the World in 3D

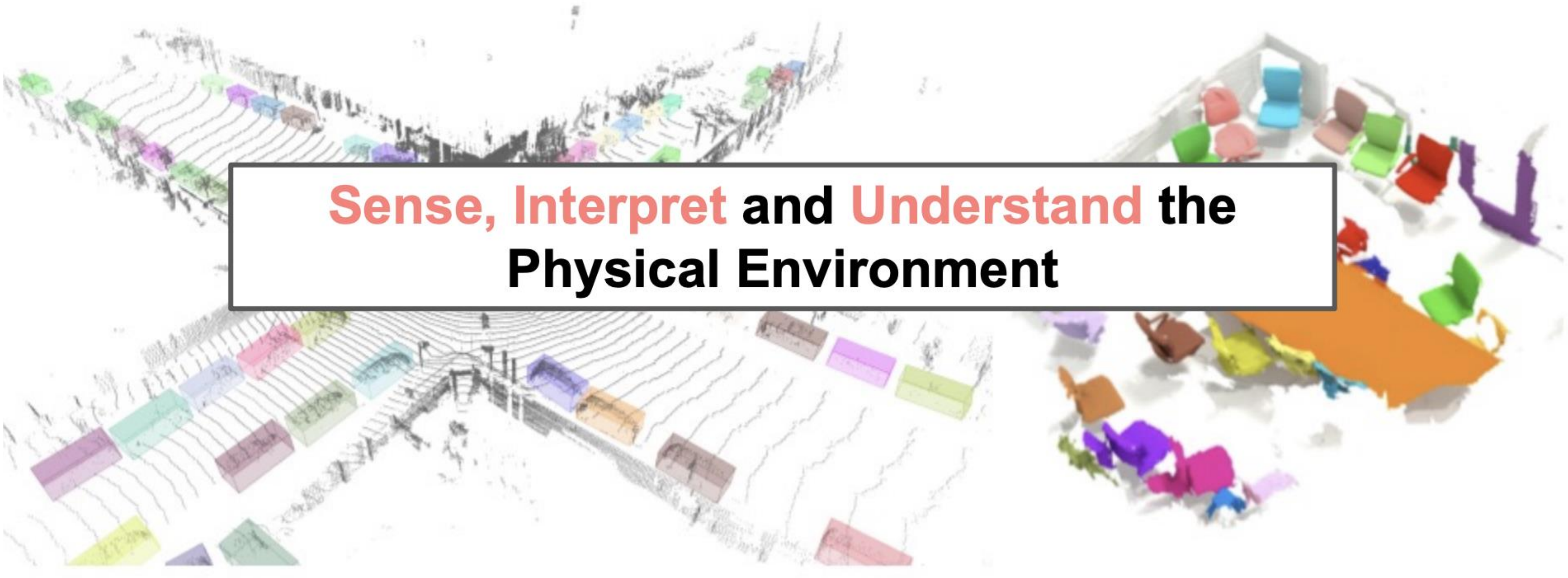


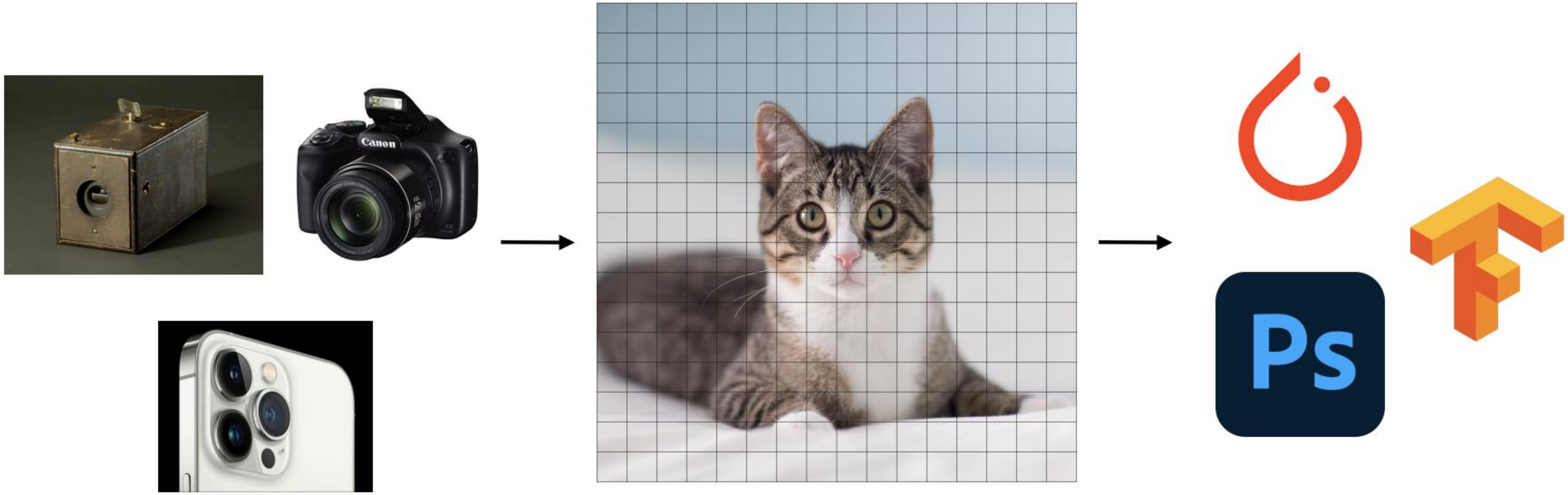
Image credit: Google, Waymo Open Dataset, ScanNet

Creating the World in 3D

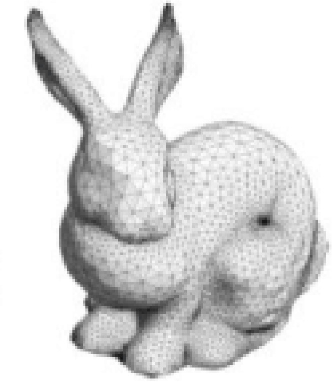
Throw a basketball with fire towards vase with flowers and break the vase with collision.

Key Challenge: How to Represent 3D?

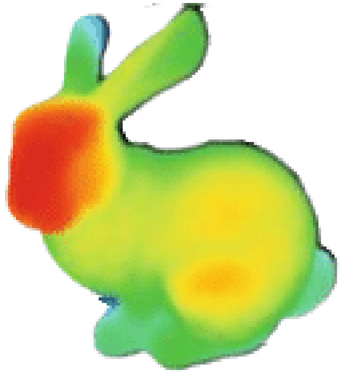
Life is good in 2D world



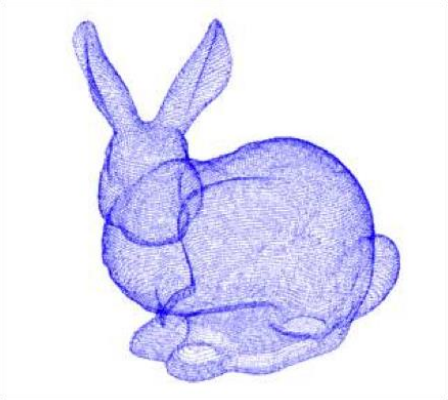
Meanwhile in 3D…



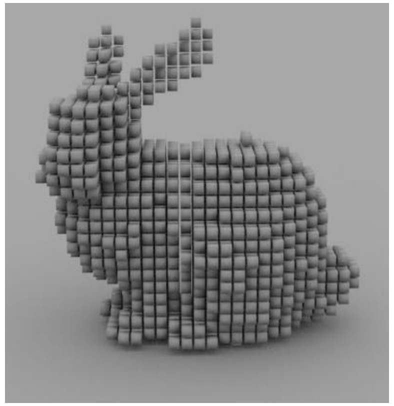
Mesh



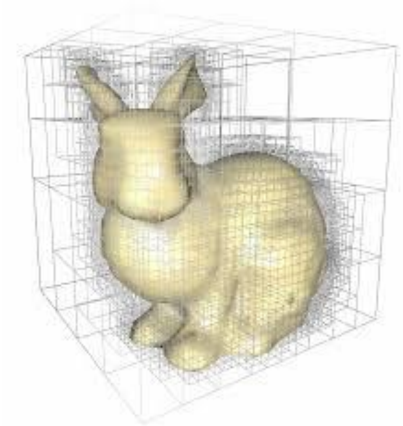
2.5D



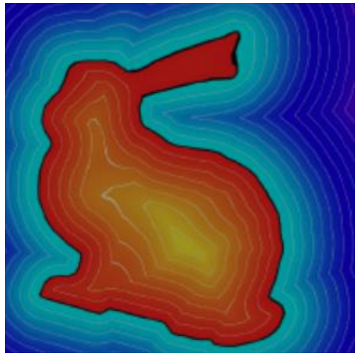
Point Cloud



Voxel

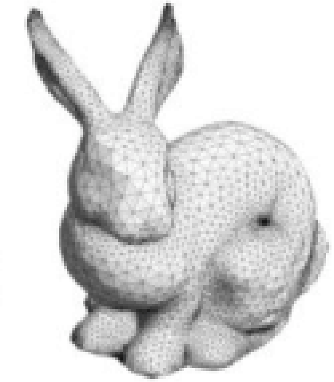


Octree



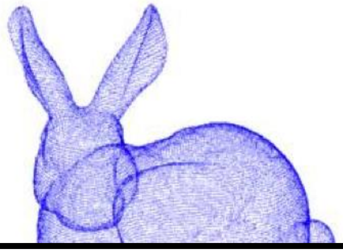
SDF

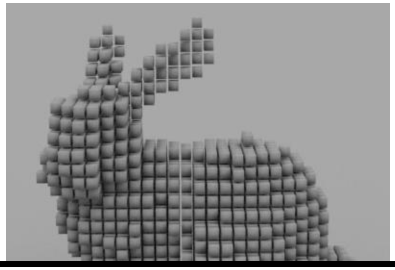
Meanwhile in 3D…

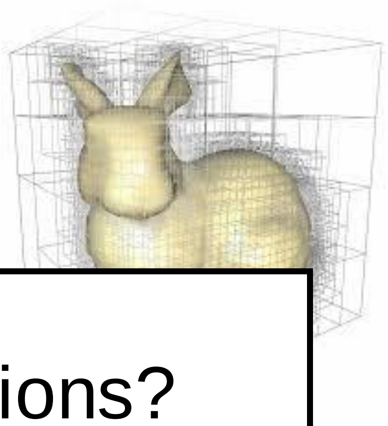


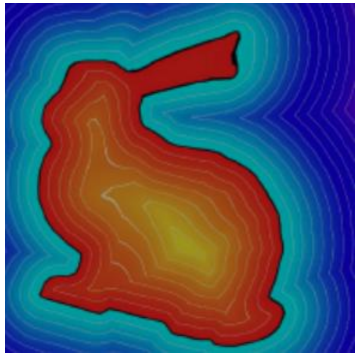
Mesh

2.5





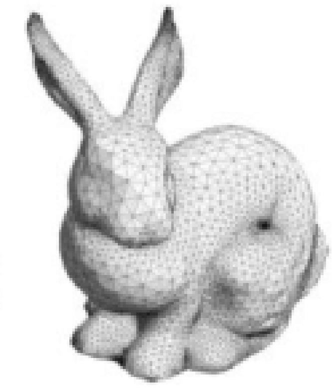




SDF

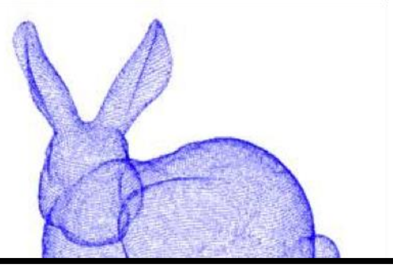
What is the ***best*** representations?

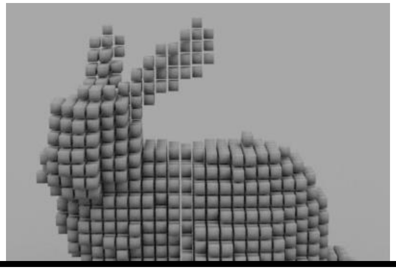
Meanwhile in 3D…

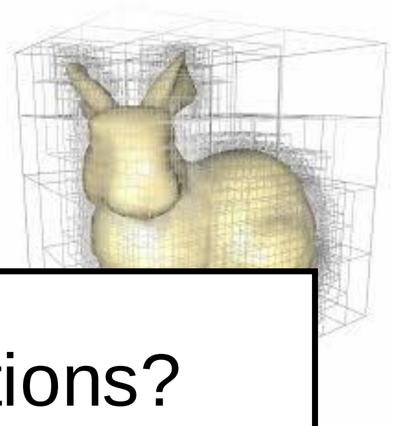


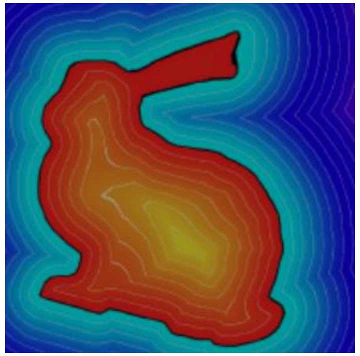
Mesh

2.5









SDF

What is the ***best*** representations?

Answer: well, it really depends…

Today's Agenda

Understand different 3D representations

- Case studies
- Pros and cons

- 2.5D, Points, Meshes, Voxels, Octree, SDFs, etc.

2.5D Representation

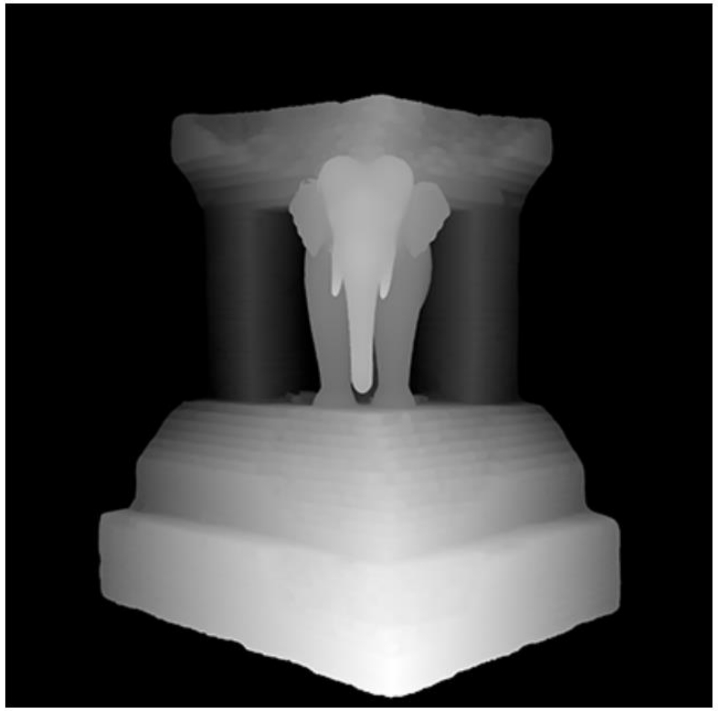
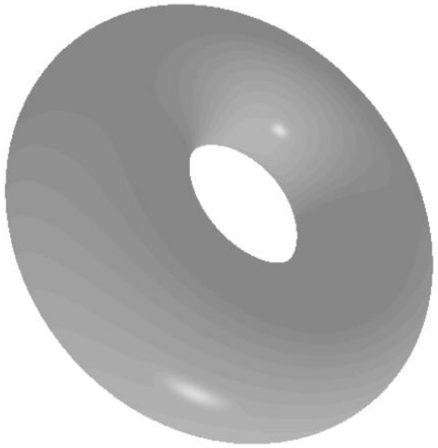
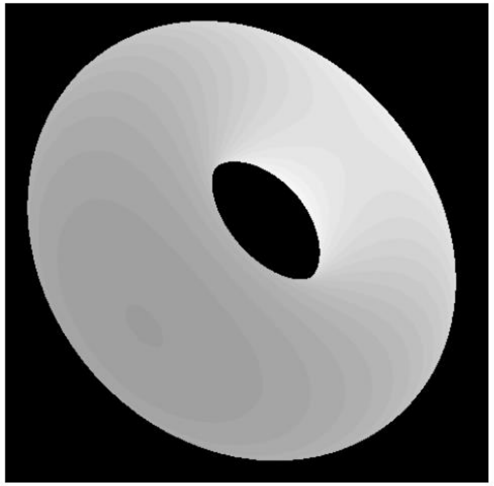
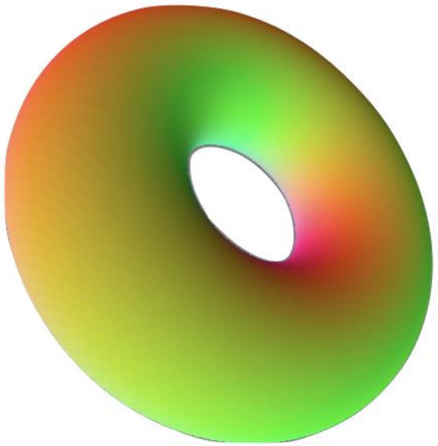


Image credits: Paul Bourke

2.5D Representation



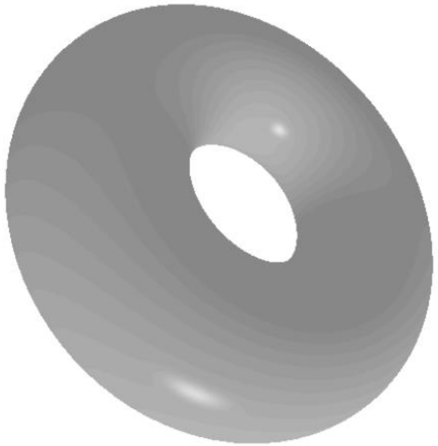


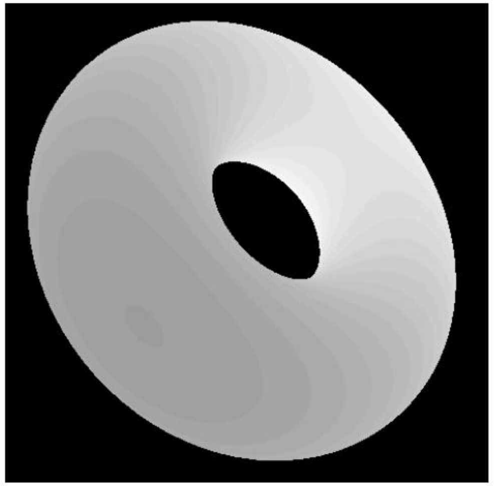


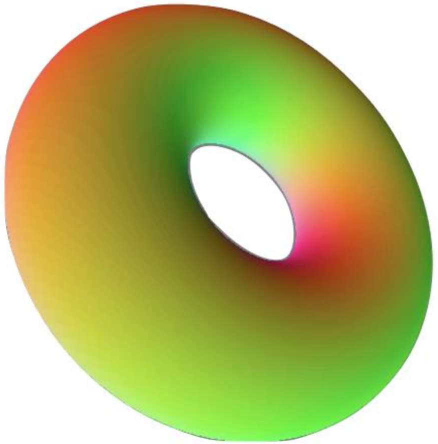
$N[\mathbf{p}] \in \mathbb{S}^2$

Image credits: Paul Bourke

2.5D Representation







$N[\mathbf{p}] \in \mathbb{S}^2$

Quiz: is it possible to get normal from depth?

Image credits: Paul Bourke

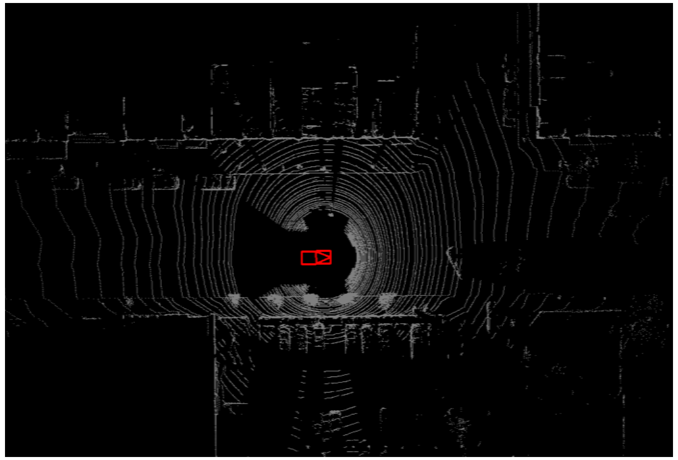
2.5D Representation

2D Tensor, Each element encode distance (optionally other attributes: such as color, reflectance, etc.)

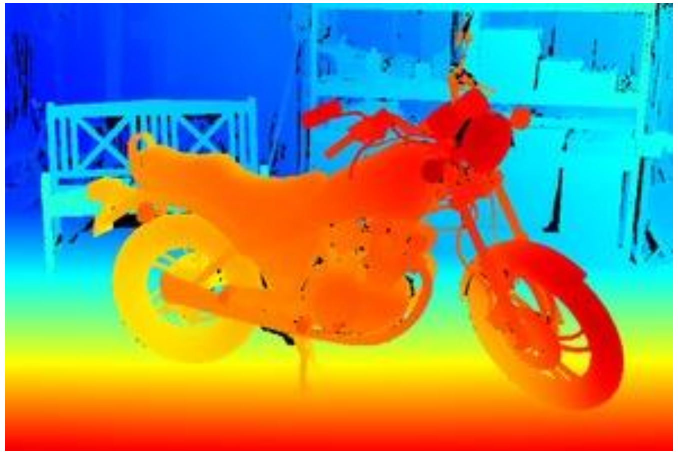
equirectangular LiDAR

Pros:

- 2D tensor, compact and efficient
- Off-the-shelf CNN perception
- Coupled with state/action space (Birds' eye view)
- Coupled with raw sensor measures

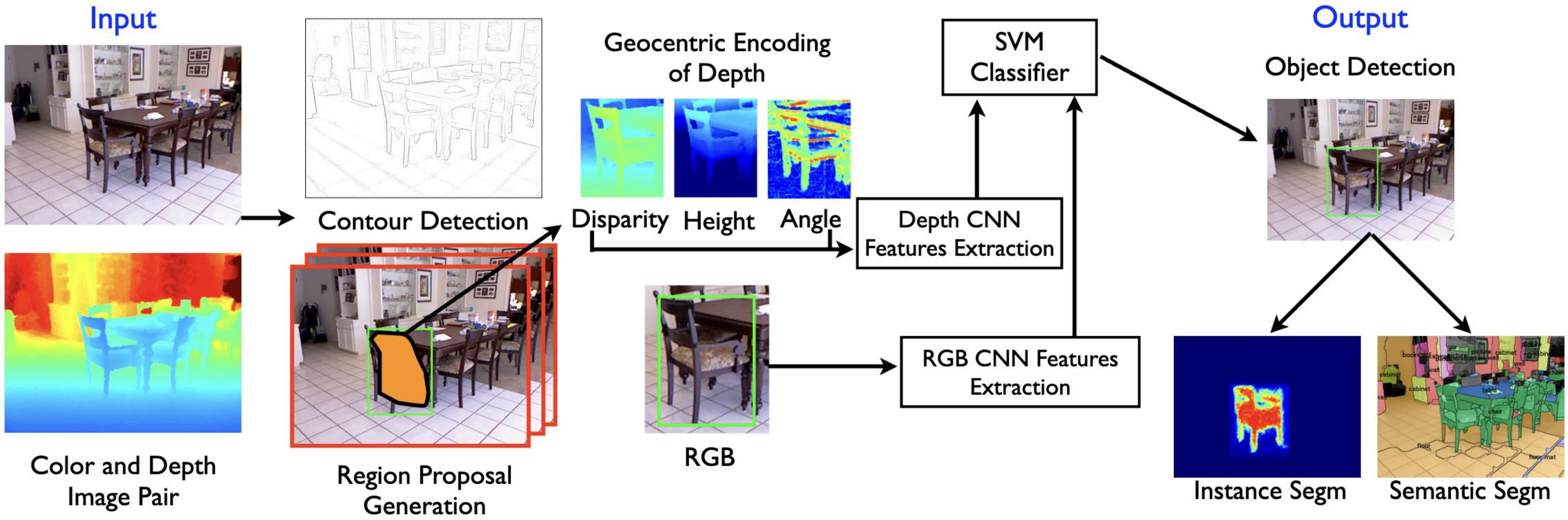


Birds-eye-view LiDAR



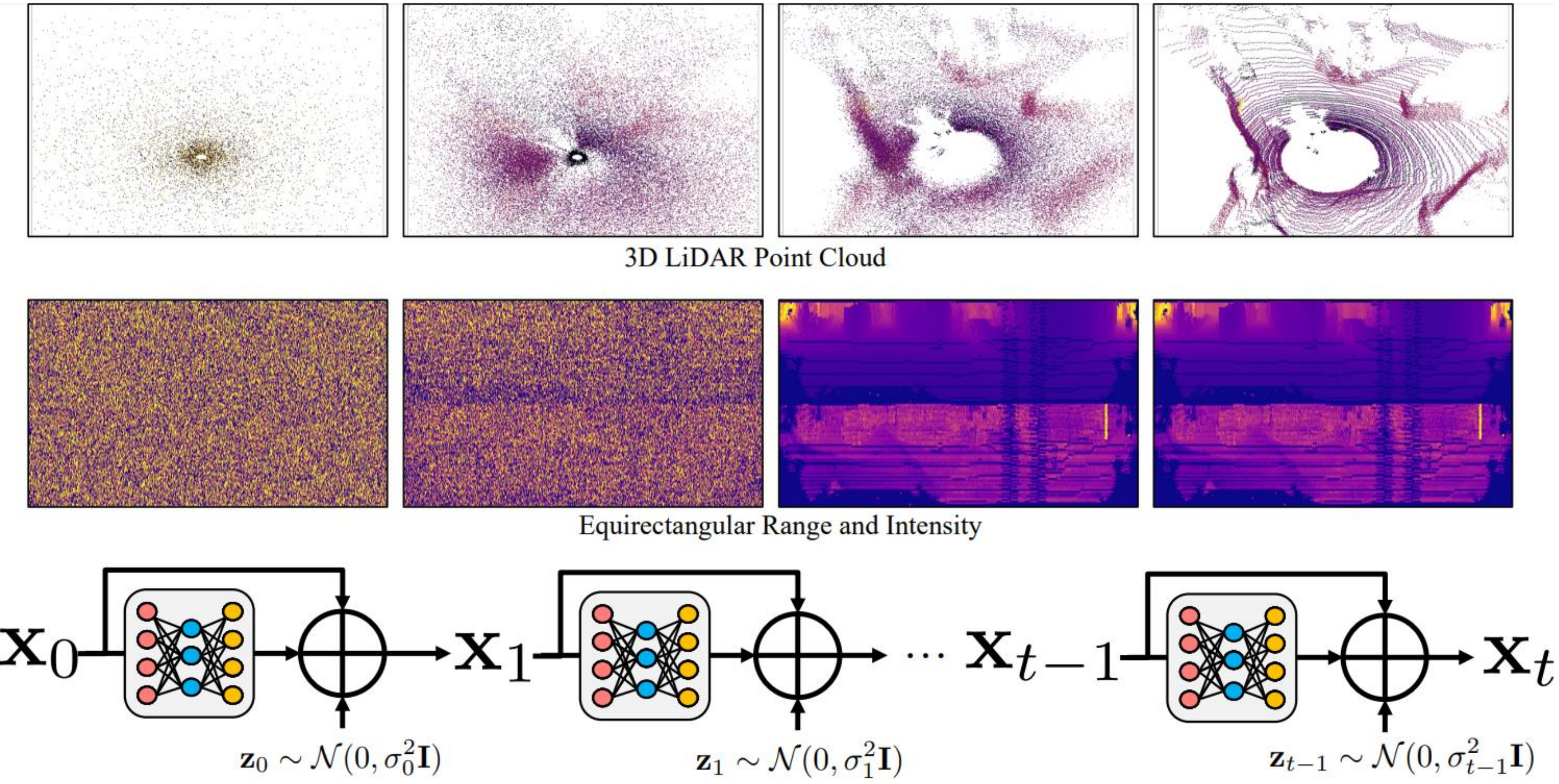
Perspective Depth Image

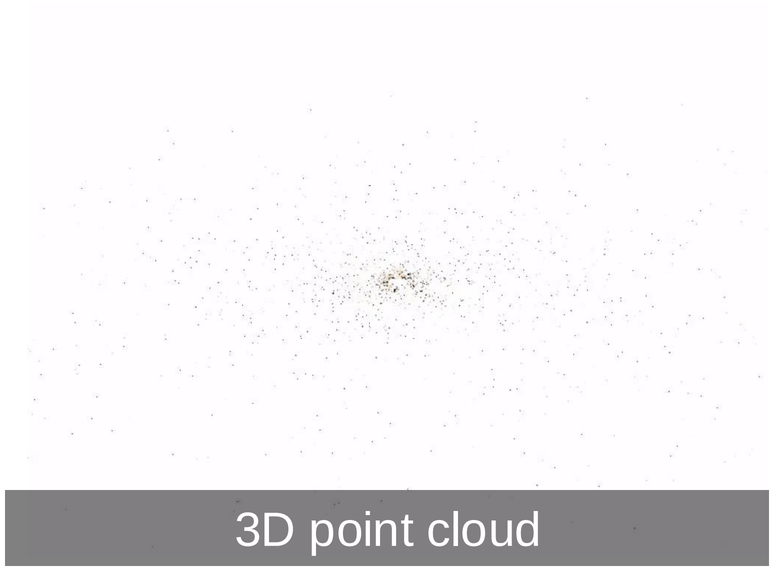
2.5D can be processed as images



Saurabh Gupta, Learning Rich Features from RGB-D Images for Object Detection and Segmentation, ECCV 2014

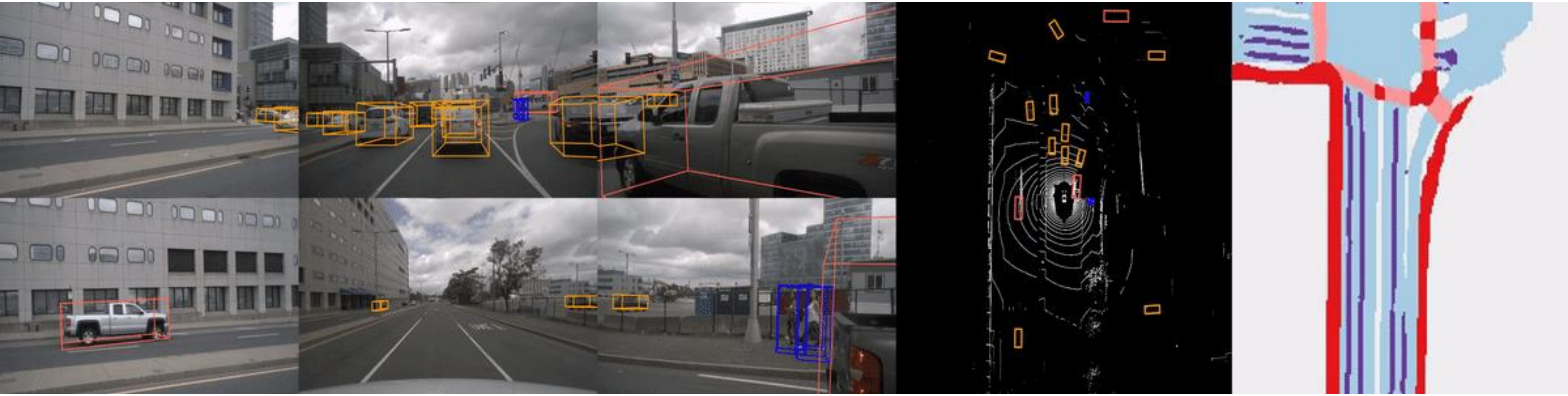
2.5D can be generated as images





Zrianov, et al. Learning to Generate Realistic LiDAR Point Clouds, ECCV 2022

BEV 2.5D is coupled with state space



Which space is easier for motion planner?

Liu, et al. BEVFusion: Multi-Task Multi-Sensor Fusion with Unified Bird's-Eye View Representation, ICRA 2023

2.5D Representation

2D Tensor, Each element encode distance (optionally other attributes: such as color, reflectance, etc.)

Cons:

- Information loss along a dimension
- Resolution loss due to rasterization
- Neighbor pixels can be far in 3D

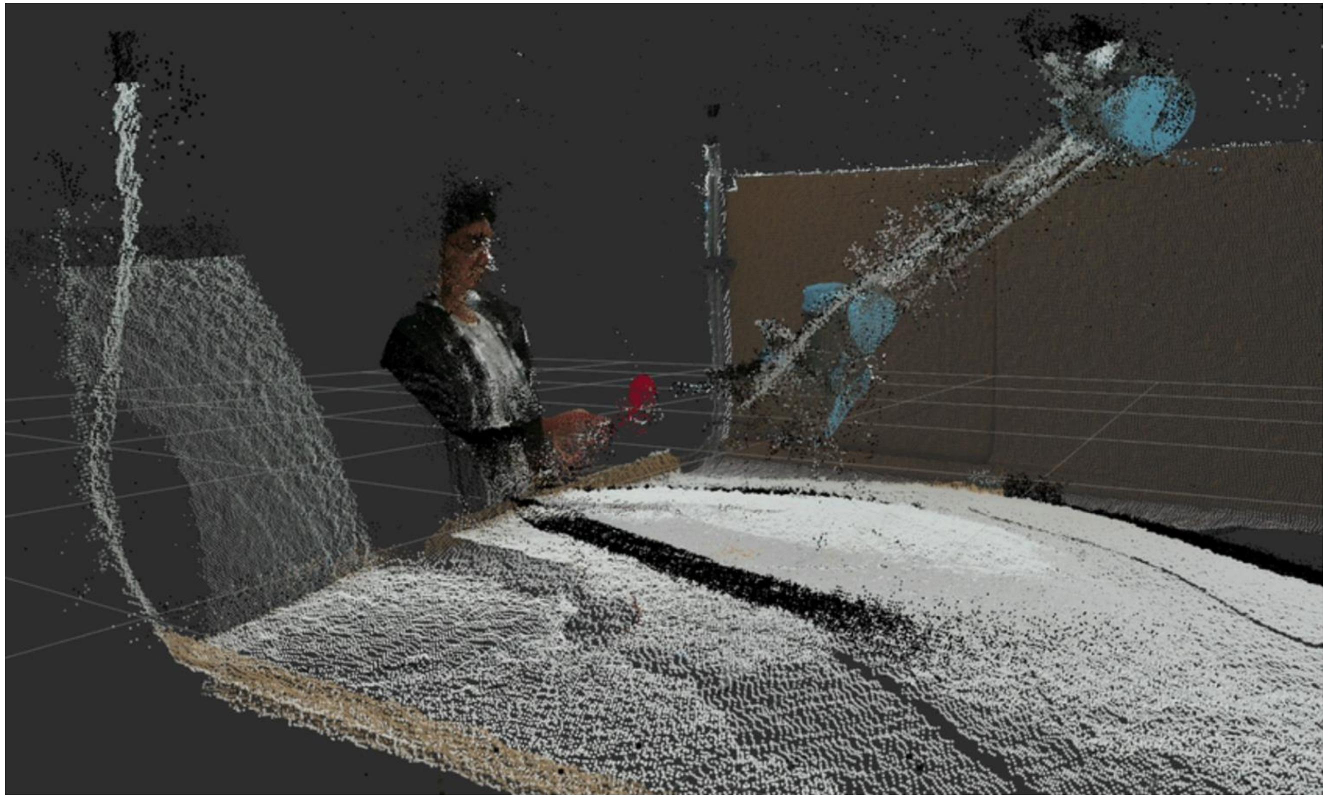


Image credits: Shubham Tulsiani

2.5D Representation

2D Tensor, Each element encode distance (optionally other attributes: such as color, reflectance, etc.)

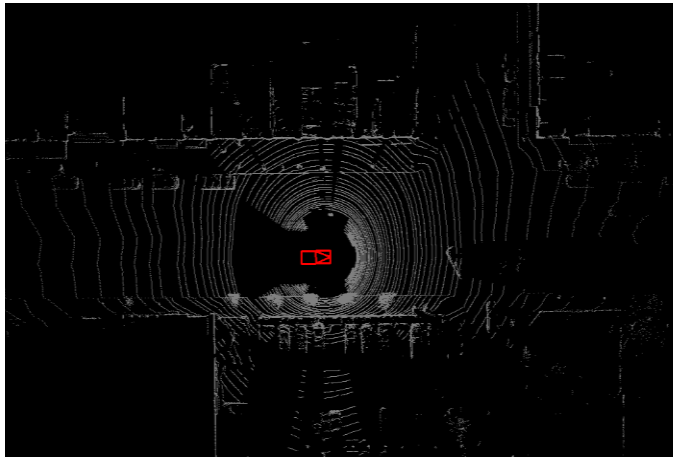
Pros:

- 2D tensor, compact and efficient
- Off-the-shelf CNN perception
- Coupled with state/action space (Birds' eye view)
- Coupled with raw sensor measures

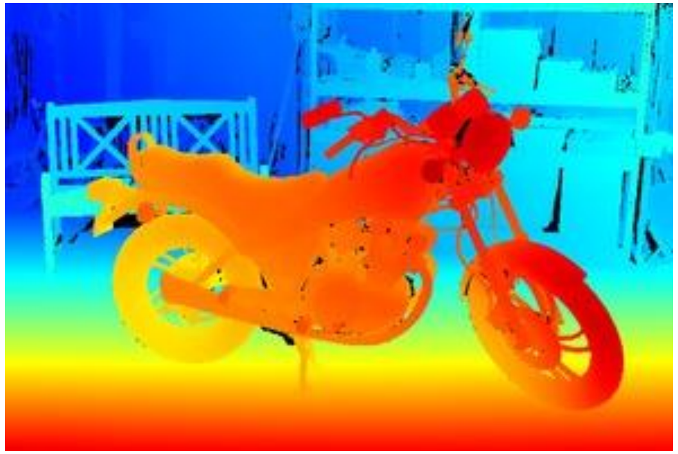
Cons:

- Information loss along a dimension
- Resolution loss due to rasterization
- Neighbor pixels can be far in 3D

equirectangular LiDAR



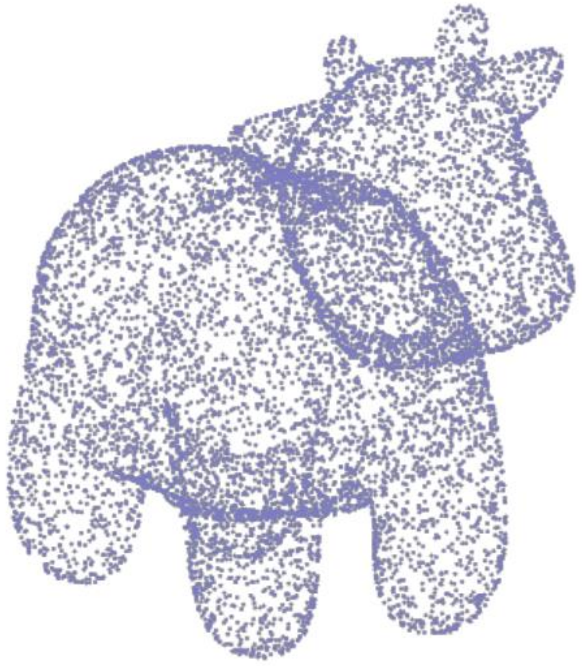
Birds-eye-view LiDAR



Perspective Depth Image

3D Point Cloud

3D unordered point, each encodes spatial location



$$\{\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_N\}$$

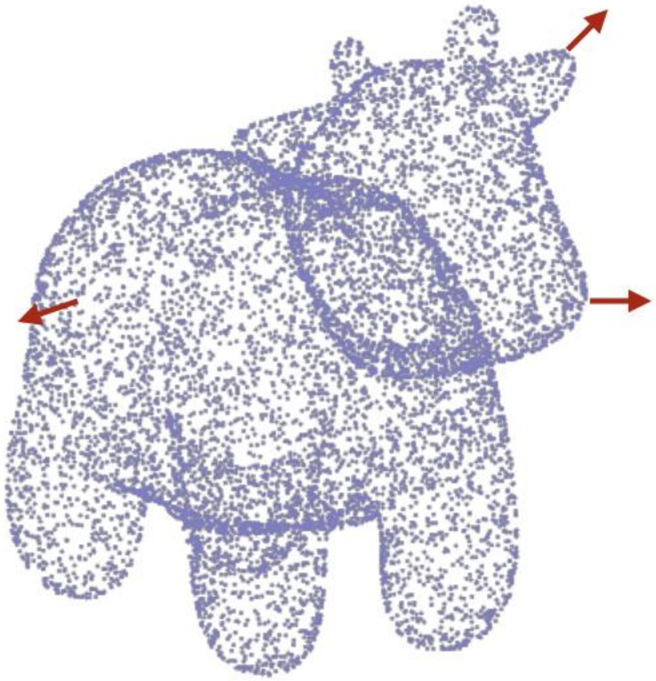
Unordered set of points

Stored as Nx3 matrix, but keep in mind they are **permutation invariant**!

Image credits: Angjoo Kanazawa

3D Point Cloud

3D unordered point, each encodes spatial location



$$\{(\mathbf{p}_1, \mathbf{n}_1), (\mathbf{p}_2, \mathbf{n}_2), \cdots, (\mathbf{p}_N, \mathbf{n}_N)\}$$

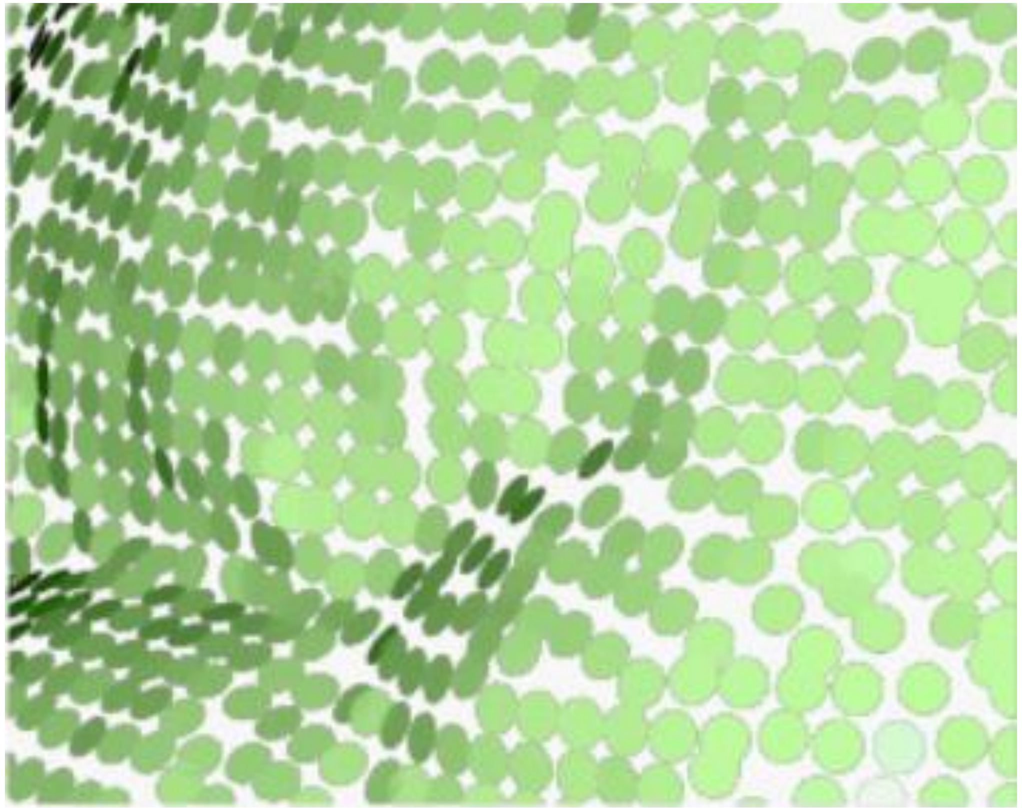
Quiz: how to get normal from point?

Could be extended to carry additional information, e.g. color, or normal

Image credits: Angjoo Kanazawa

3D Point Cloud

3D unordered point, each encodes spatial location

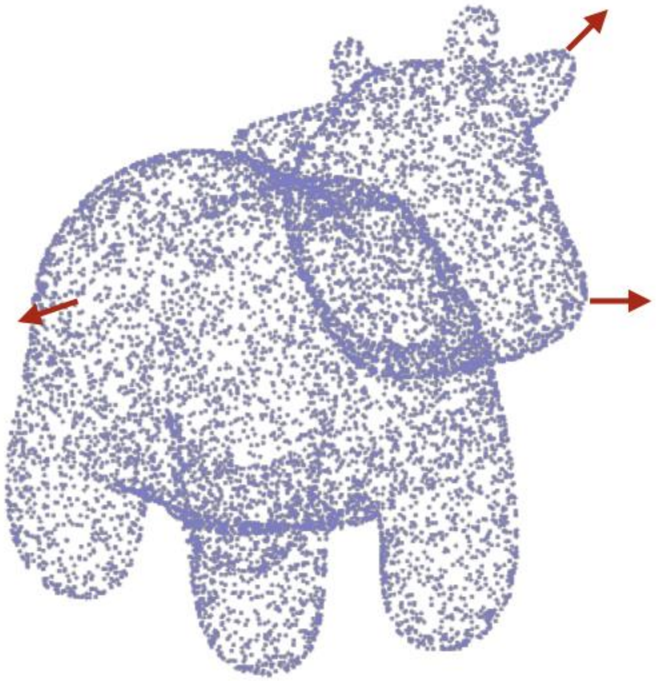


Could be further extended to be a set of small disks. **Why?**

Image credits: Surfel Meshing

3D Point Cloud

3D unordered point, each encodes spatial location



Quiz: how to apply deep learning?

$$\{(\mathbf{p}_1, \mathbf{n}_1), (\mathbf{p}_2, \mathbf{n}_2), \cdots, (\mathbf{p}_N, \mathbf{n}_N)\}$$

Could be extended to carry additional information, e.g. color, or normal

3D Point Cloud

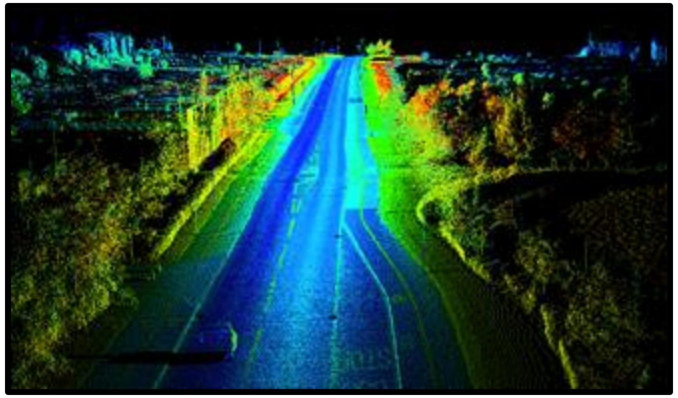
3D unordered point, each encodes spatial location

Pros:

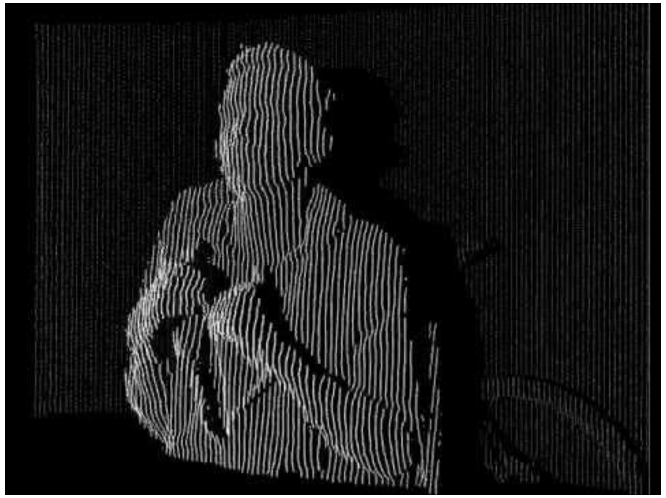
- No geometry loss
- Memory and computational efficient processing

Cons:

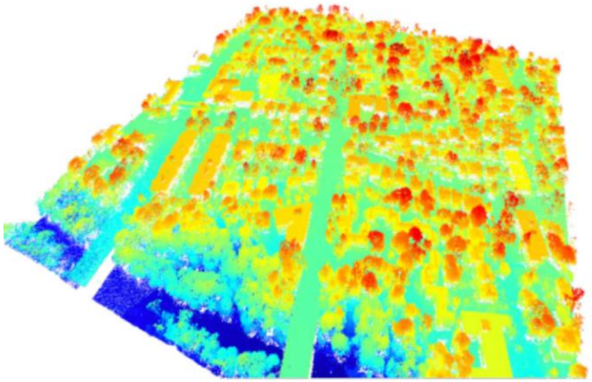
- **No topology**; No occupancy/surface
- Need to splat or hole filling for rendering
- Hard to retrieve neighbors (need kd-tree, r-tree, octree, etc)



Point Cloud from Surveying Lidar



Point Cloud from Kinect



Point Cloud from Surveying Lidar

Meshes

- A mesh is a set of vertices with faces that defines the topology
- Mesh = {Vertices, Faces}
  - Vertices: N x 3
  - Faces: |F| x {3, 4, …} specifying the edges of a polygon
  - Triangle faces most common but tetrahedrons (tets) are also.
- Surface is explicitly modeled by the faces
- Most common modeling representation

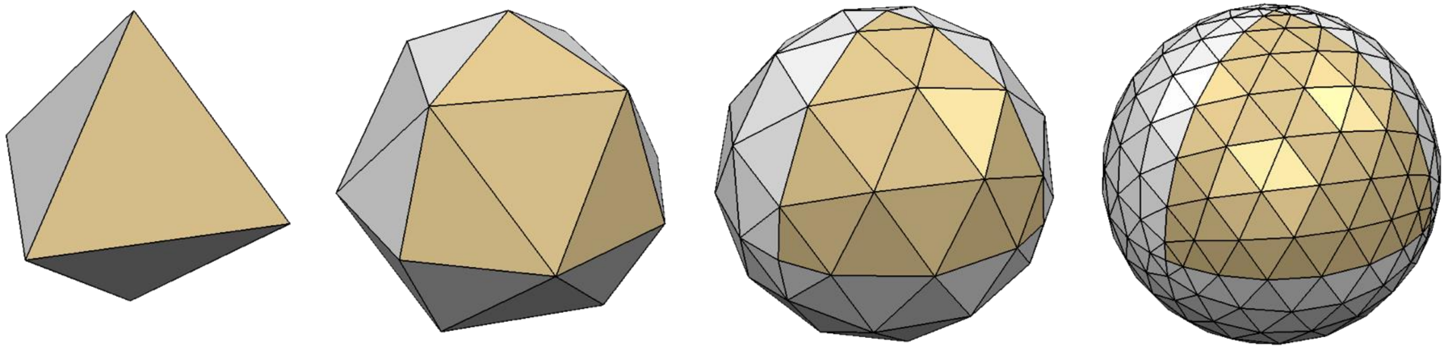
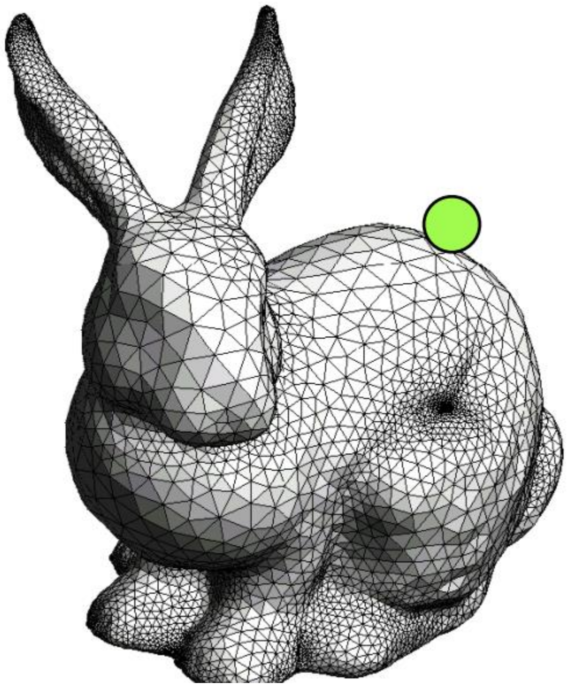


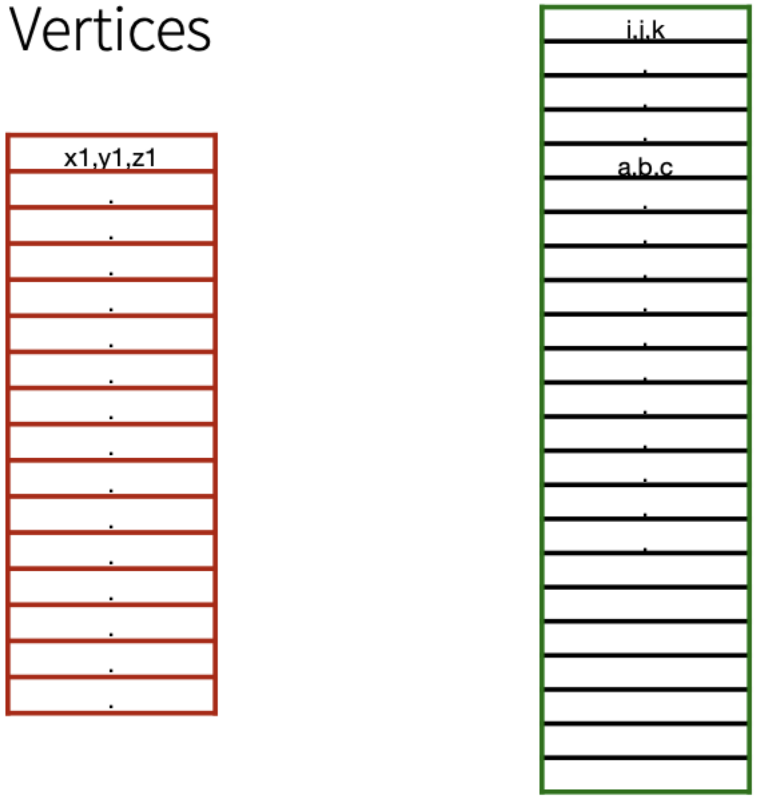
Image credits: Angjoo Kanazawa

Meshes



Faces

Vertices



Positions of Vertices

Connectivity
(indices of **three** vertices
that make a 'face')

Meshes are great for texturing

UV Image

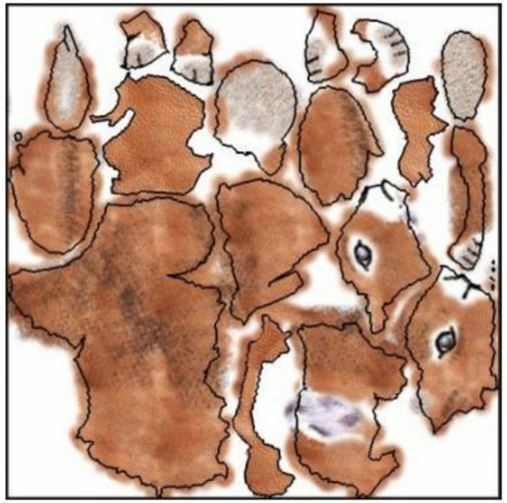


Image credits: Angjoo Kanazawa

UV Mapping

- Defined by UV mapping : (x,y,z) → (u,v)
- “texture coordinates”

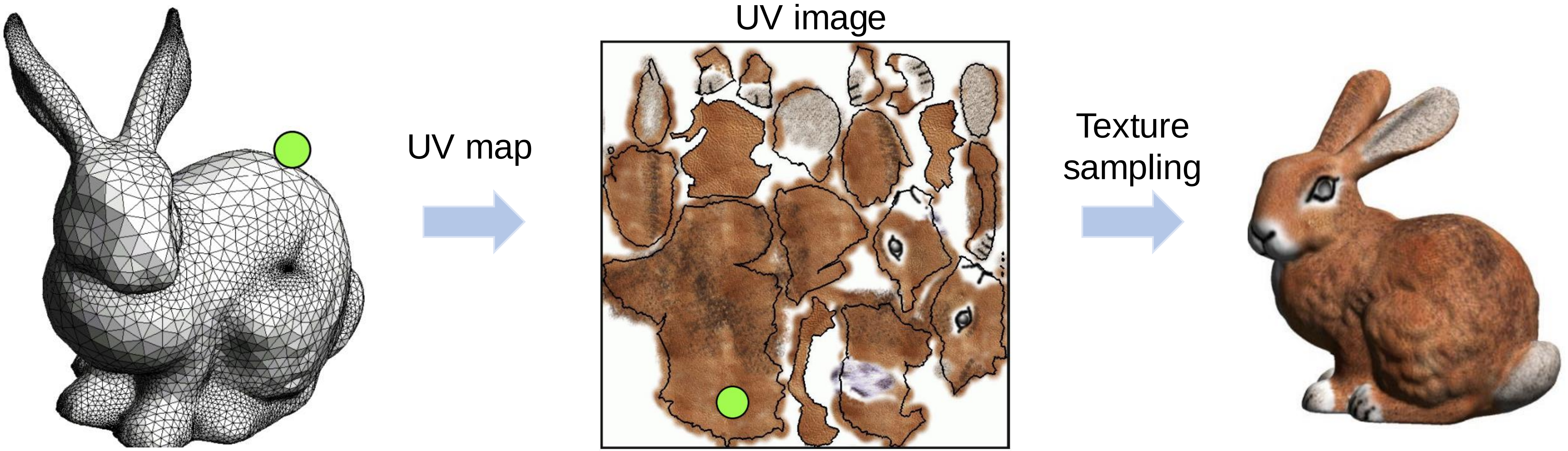


Image credits: Angjoo Kanazawa

Mesh Representation

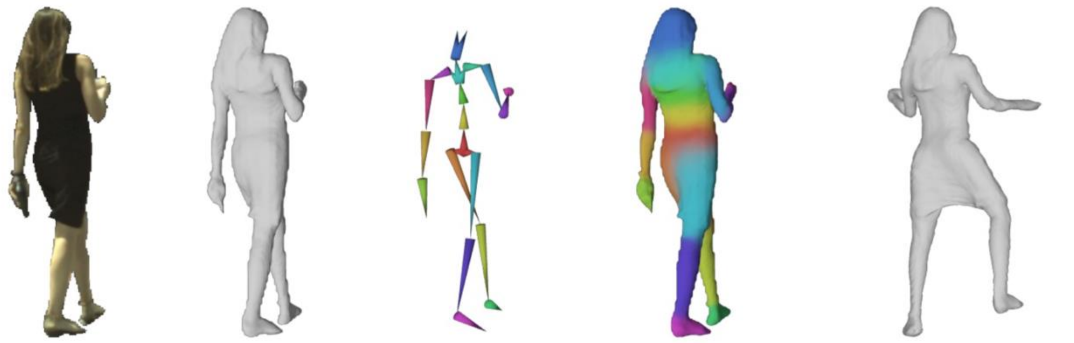
A collection of vertices and faces that defines the shape of a polyhedral object

Pros:

- Memory efficient
- Easy to deform, easy to texturing
- Explicit surface

Cons:

- Topology restrictions
- Hard for ML (parametric shape, template, GNNs)



Voxel Representation

- Expressive power dependent on voxel resolution

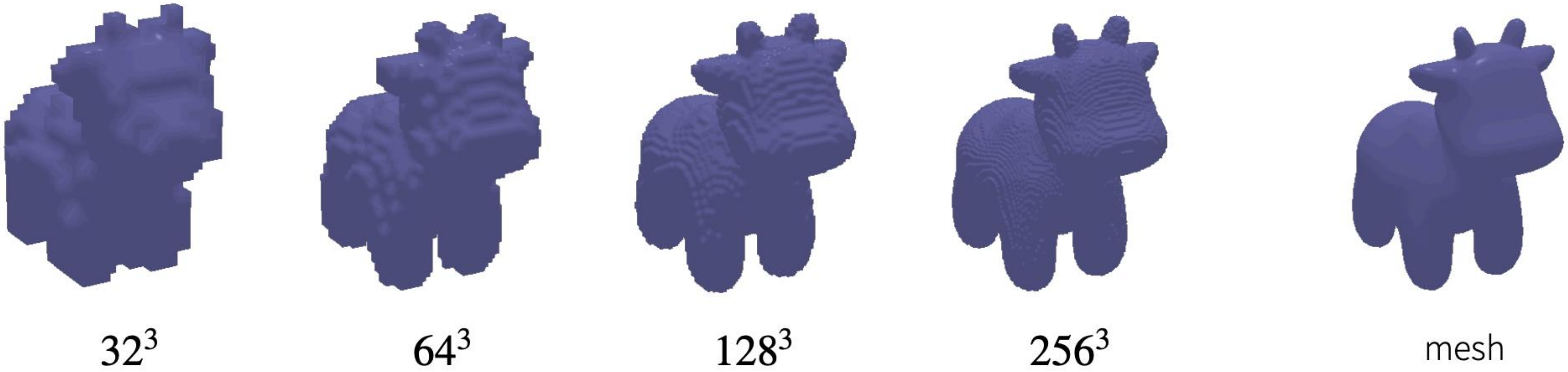


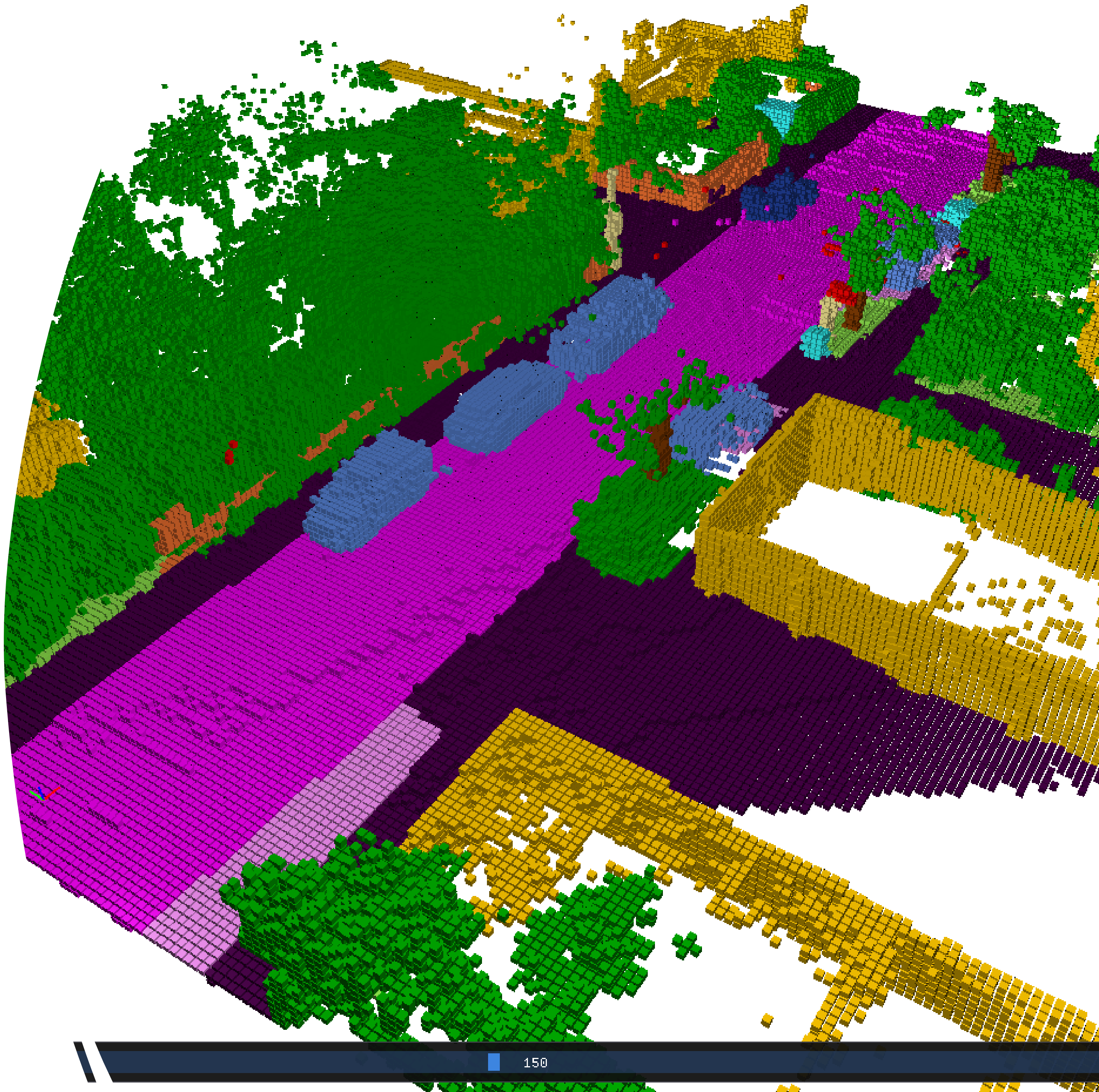
Image credits: Shubham Tulsiani

Voxel Representation

Dense grid, each voxel encodes occupancy

Pros:

- Easy to learn/process (3D CNNs)
- Can be accurate (with very high-resolution)
- Easy to compute occupancy/freespace



Voxel Representation

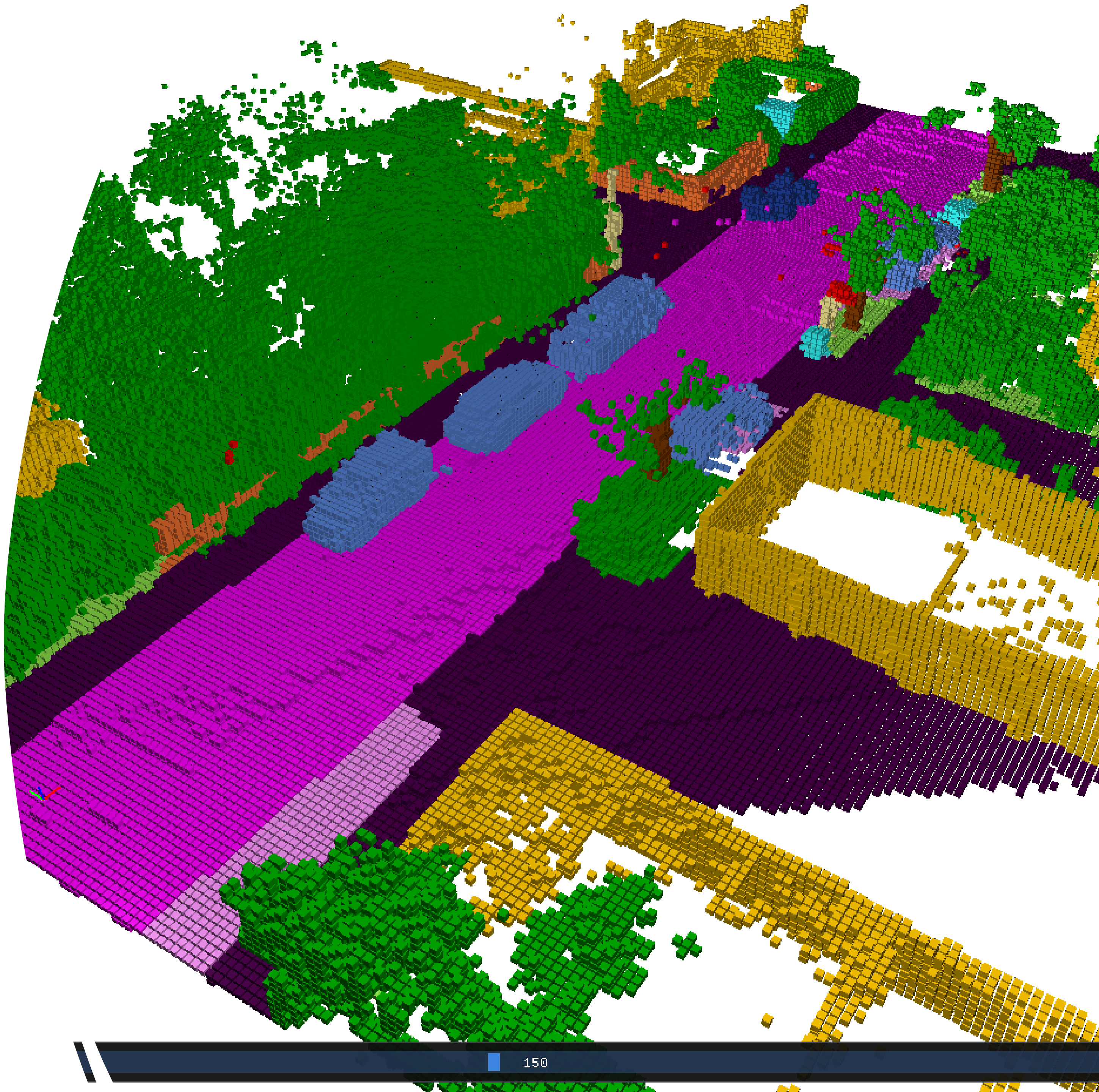
Dense grid, each voxel encodes occupancy

Pros:

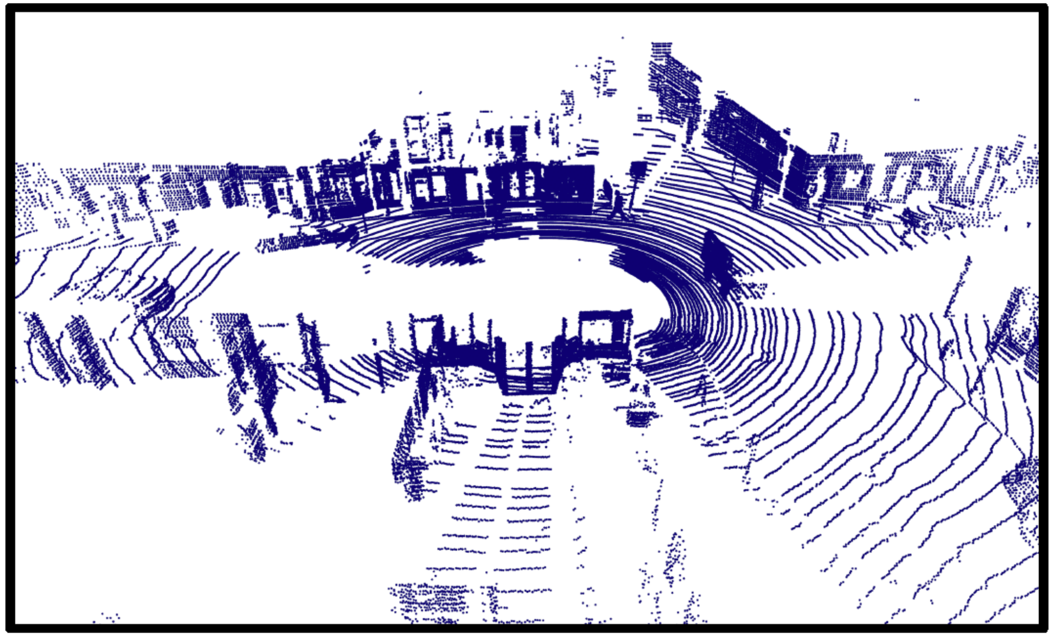
- Easy to learn/process (3D CNNs)
- Can be accurate (with very high-resolution)
- Easy to compute occupancy/freespace

Cons:

- Intensive memory, requires special data structure to scale up
- Hard to render (volume rendering)

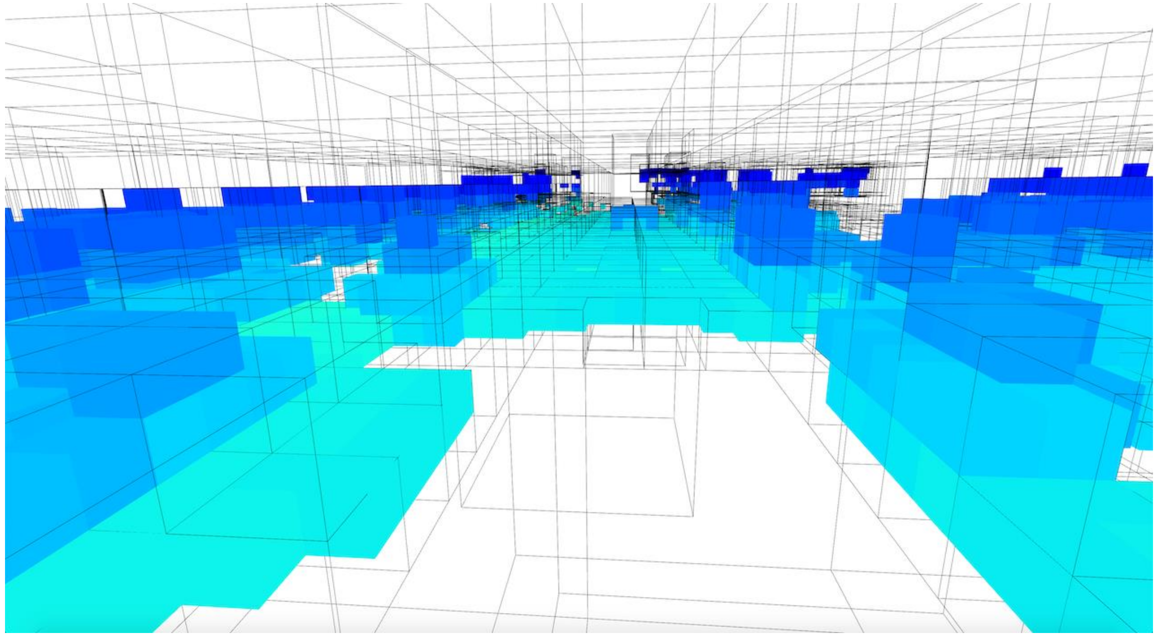


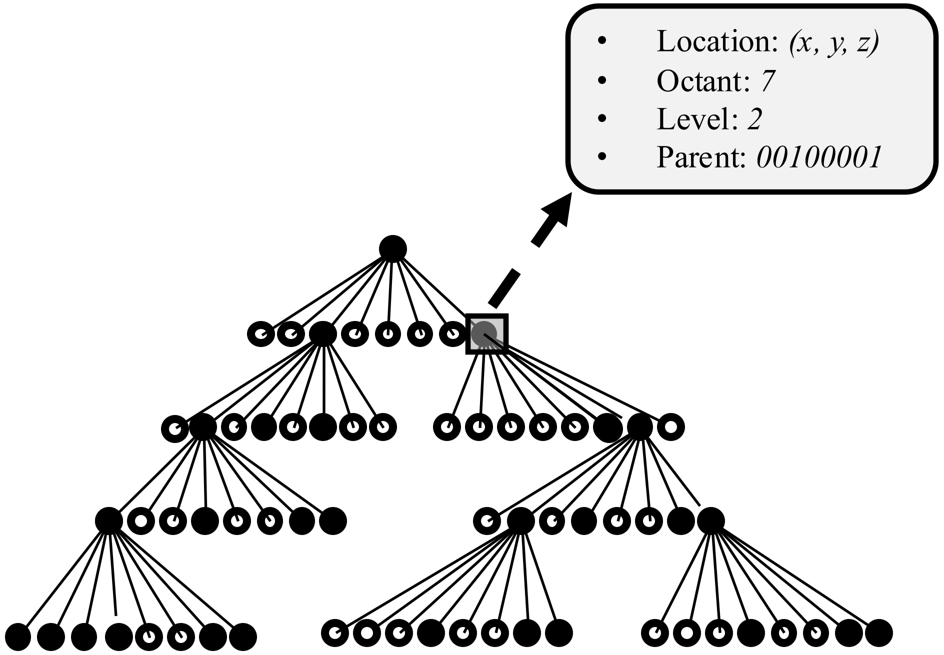
3D Sensors Require Storage



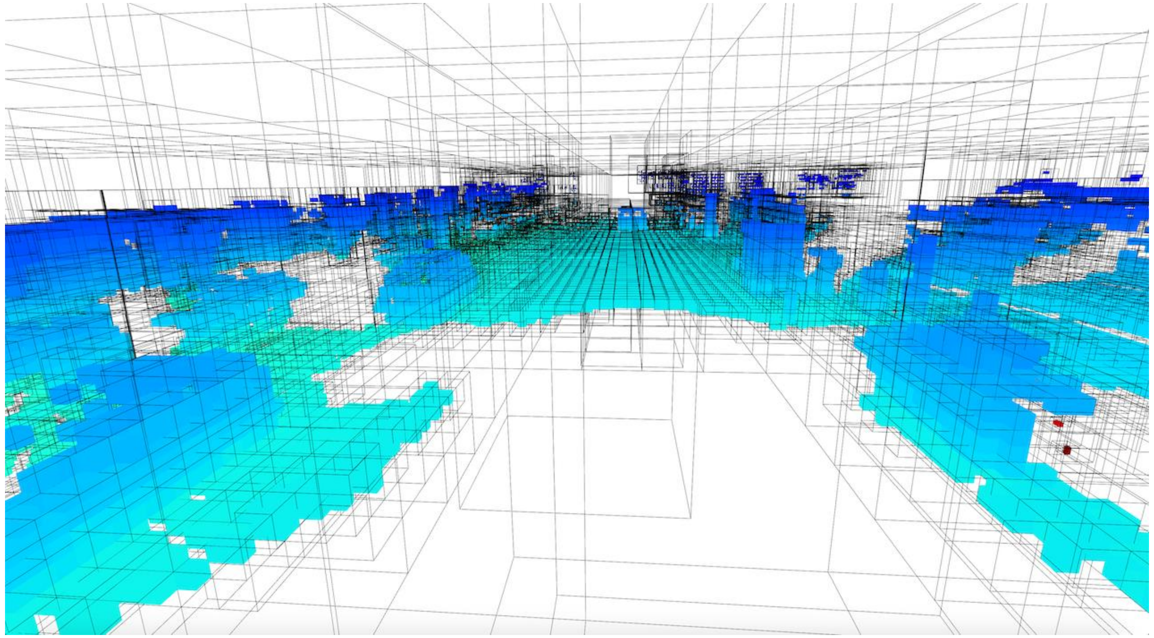
> 1TB / 8hr

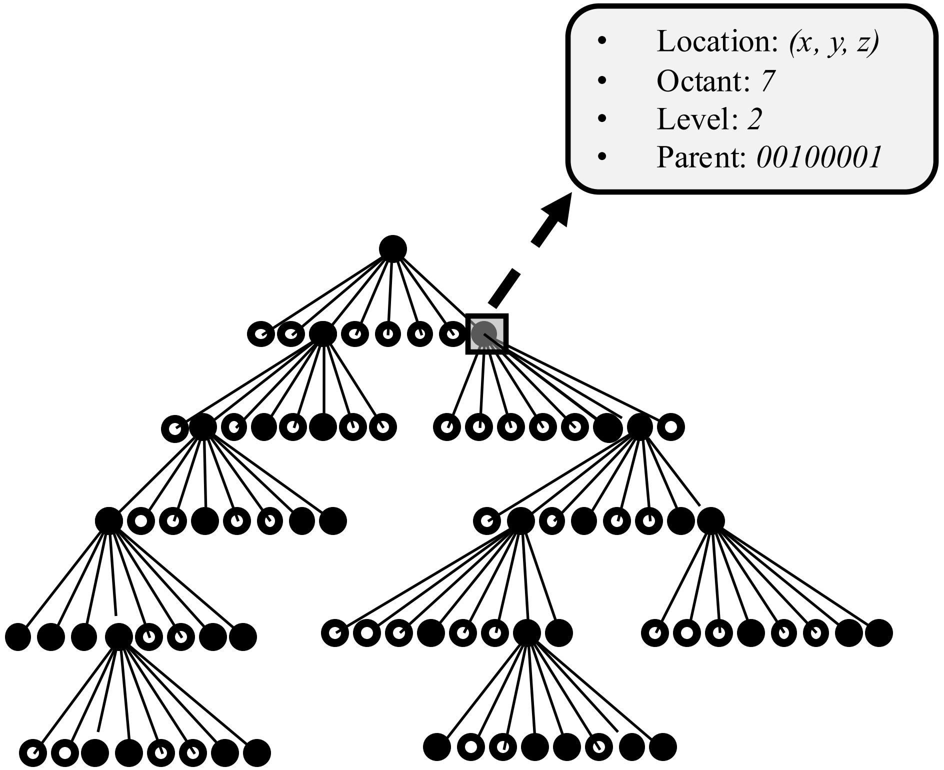
Octree



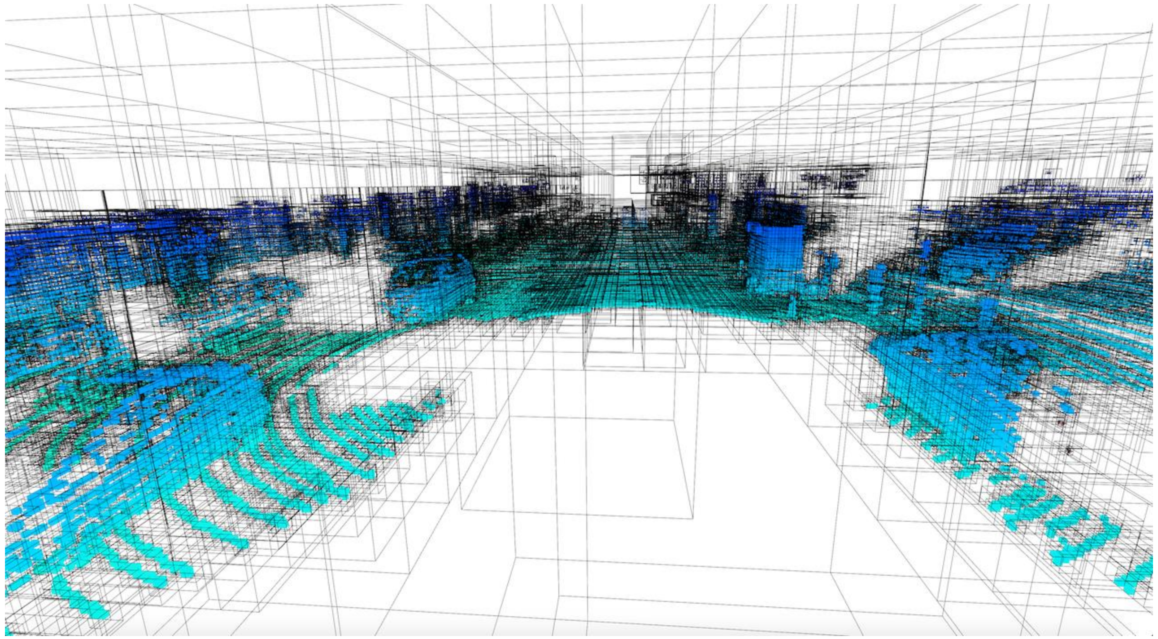


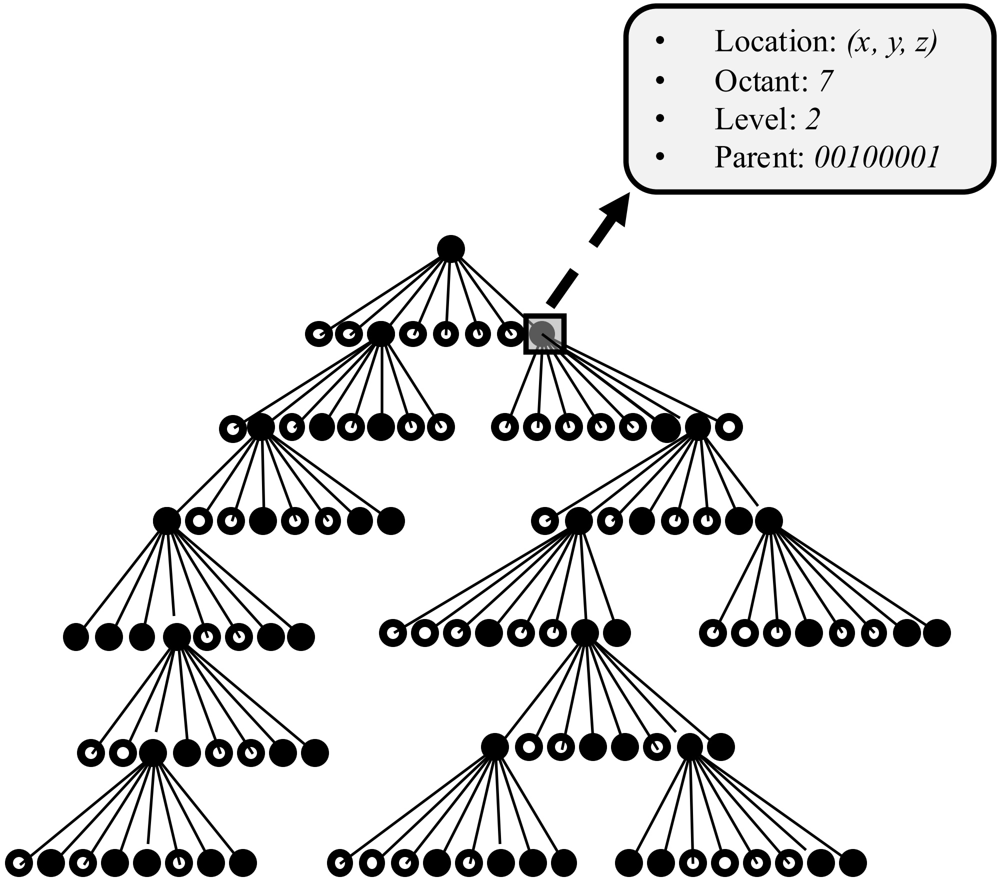
Octree



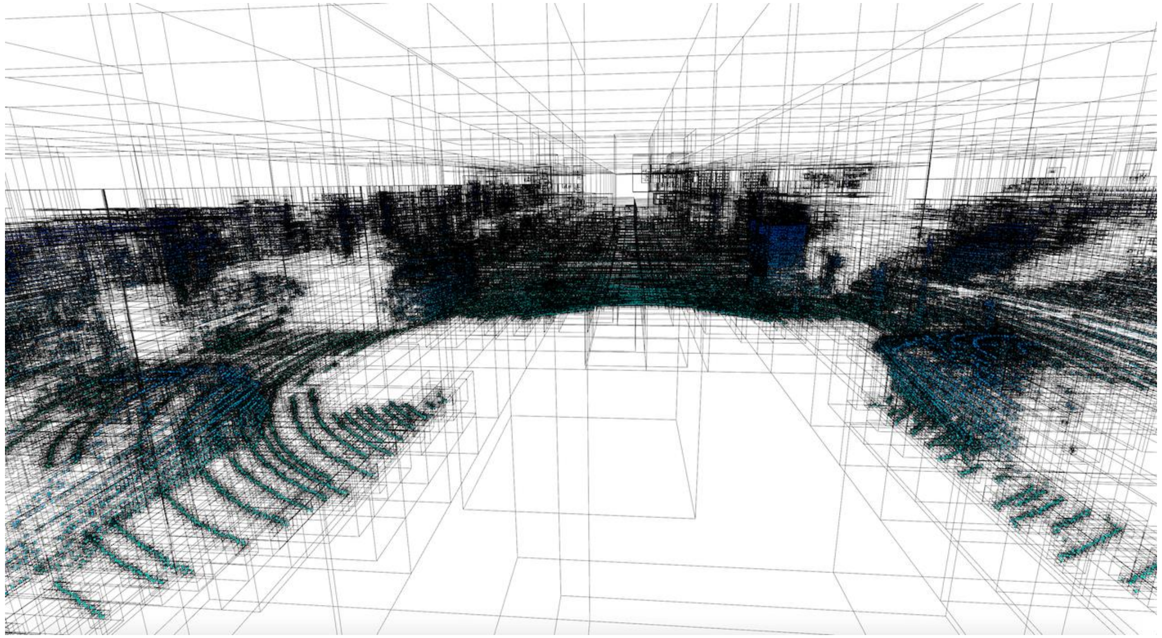


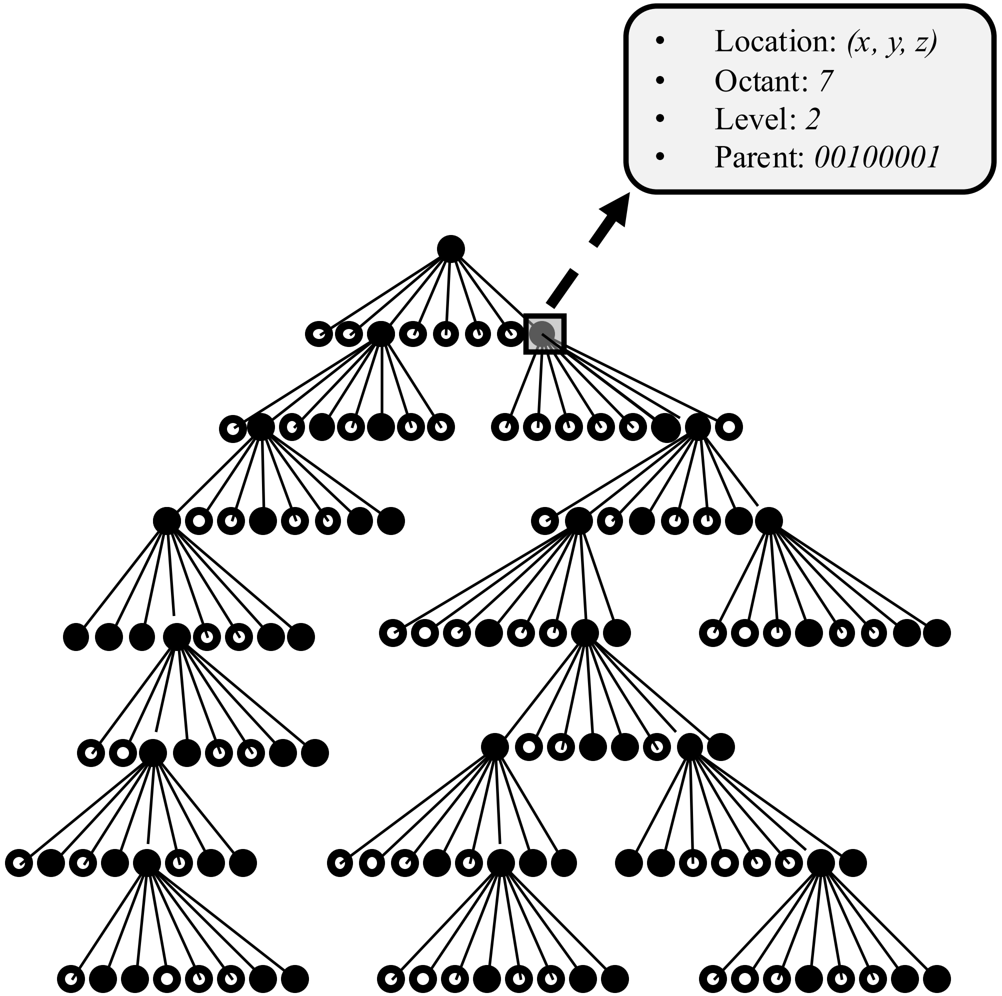
Octree



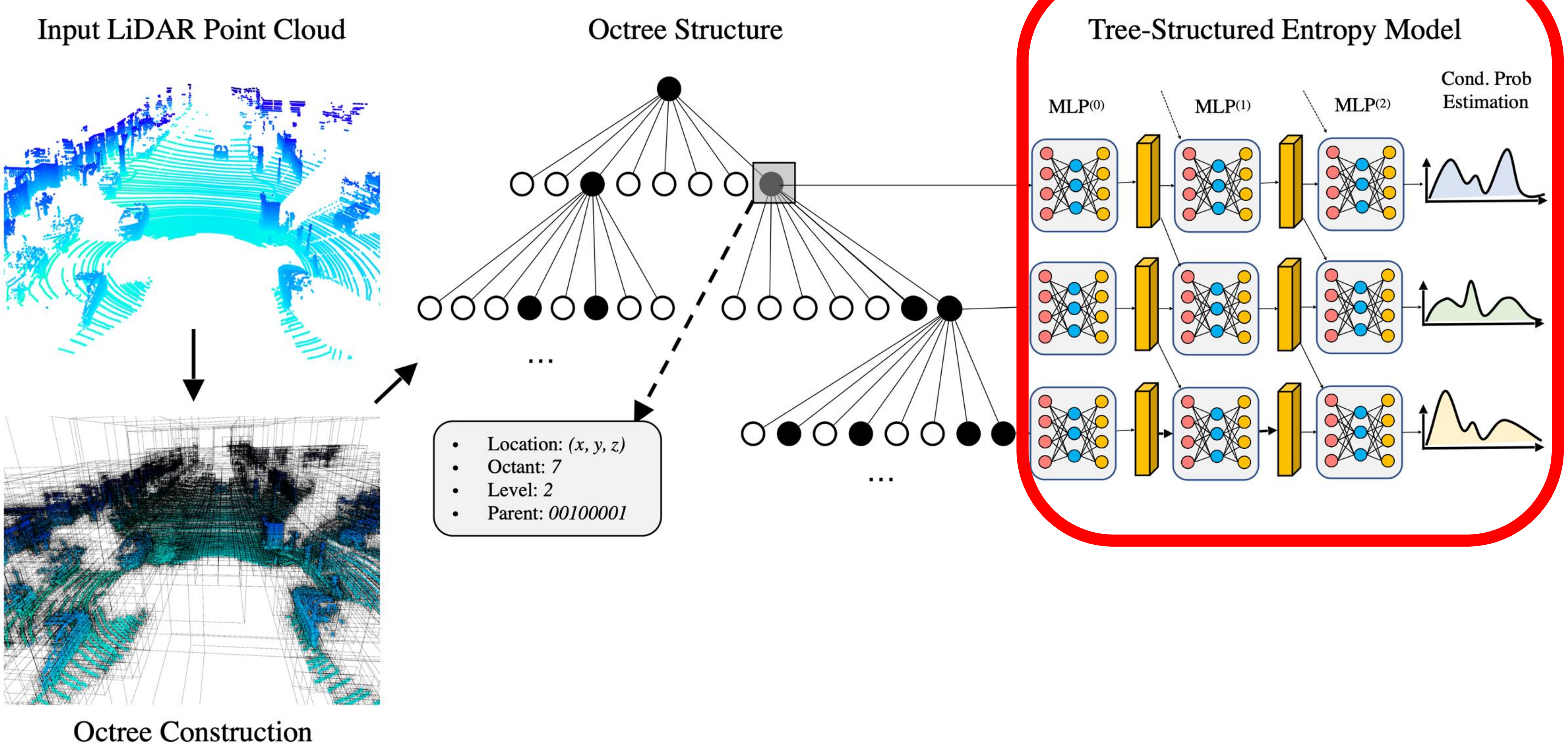


Octree

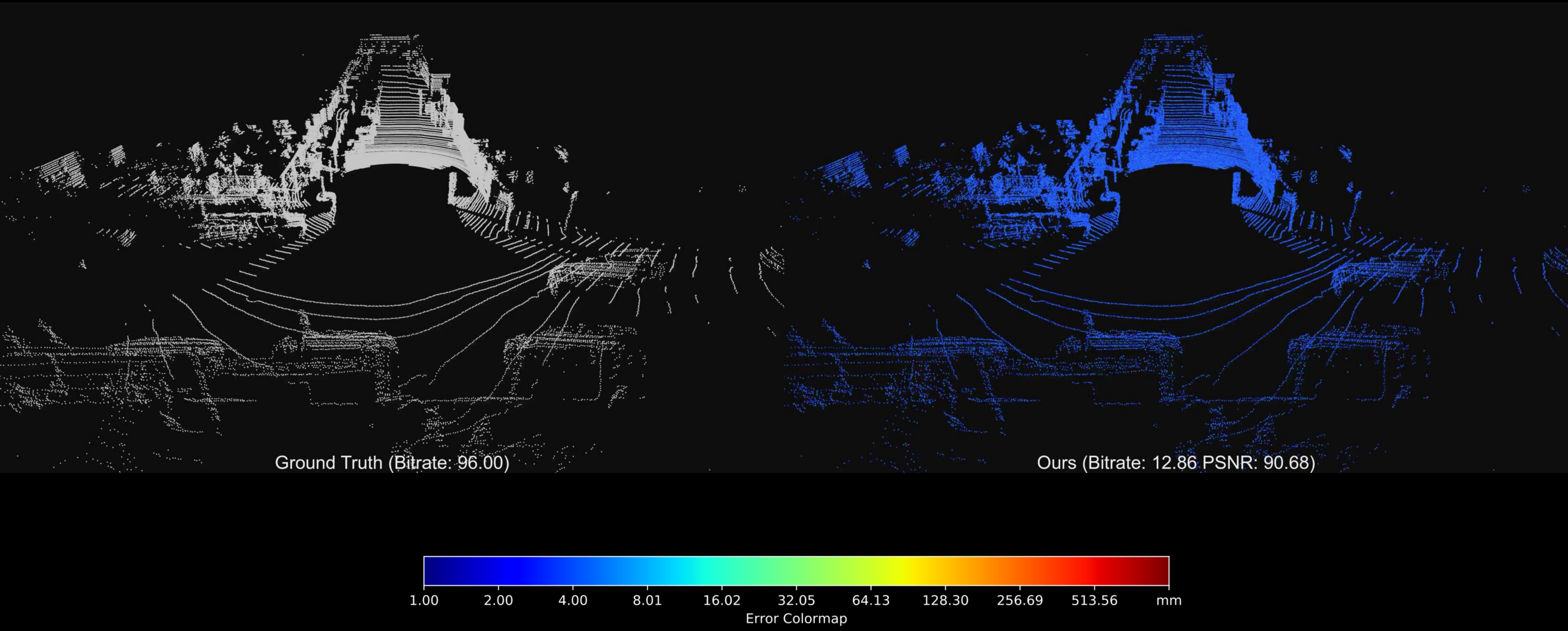


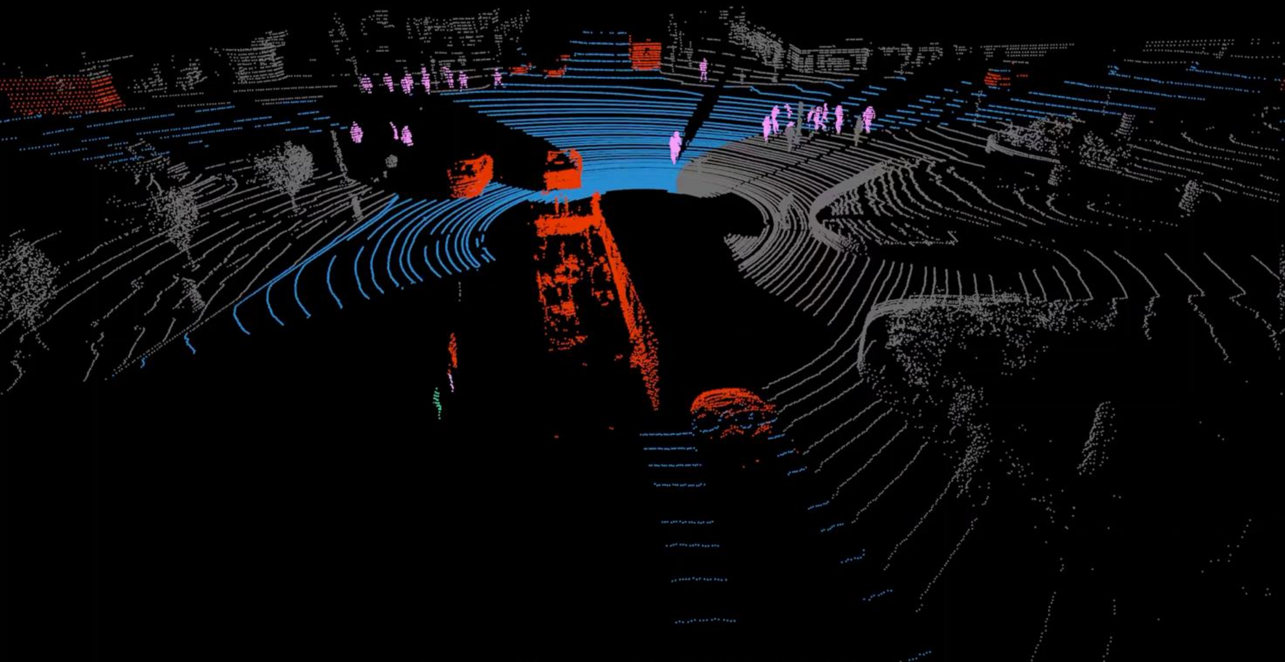


OctSqueeze

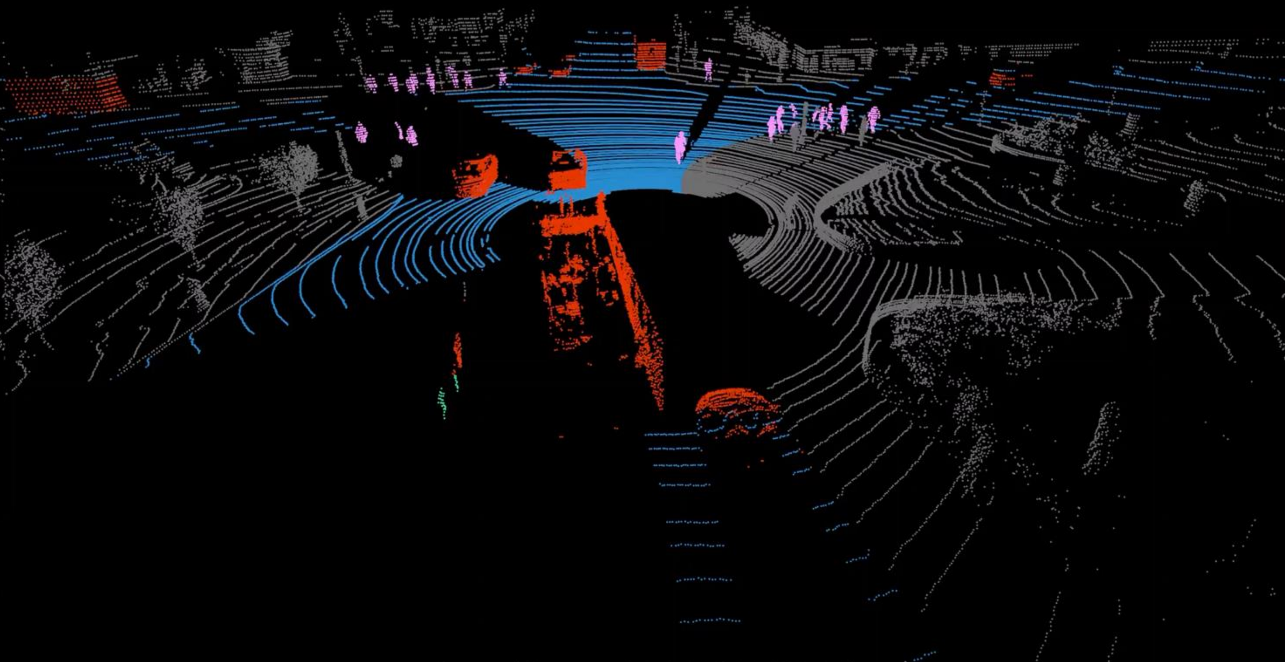


Input LiDAR Point Cloud
Octree Construction
Octree Structure
Location: (x, y, z)
Octant: 7
Level: 2
Parent: 00100001
Tree-Structured Entropy Model
MLP(0)
MLP(1)
MLP(2)
Cond. Prob Estimation


Ground Truth (Bitrate: 96.00)
Ours (Bitrate: 12.86 PSNR: 90.68)
1.00
2.00
4.00
8.01
16.02
32.05
64.13
128.30
256.69
513.56
mm
Error Colormap



Oracle (Bitrate: 96.00 IOU: 73.99)



Ours (Bitrate: 14.61 IOU: 70.94)

Octree Representation

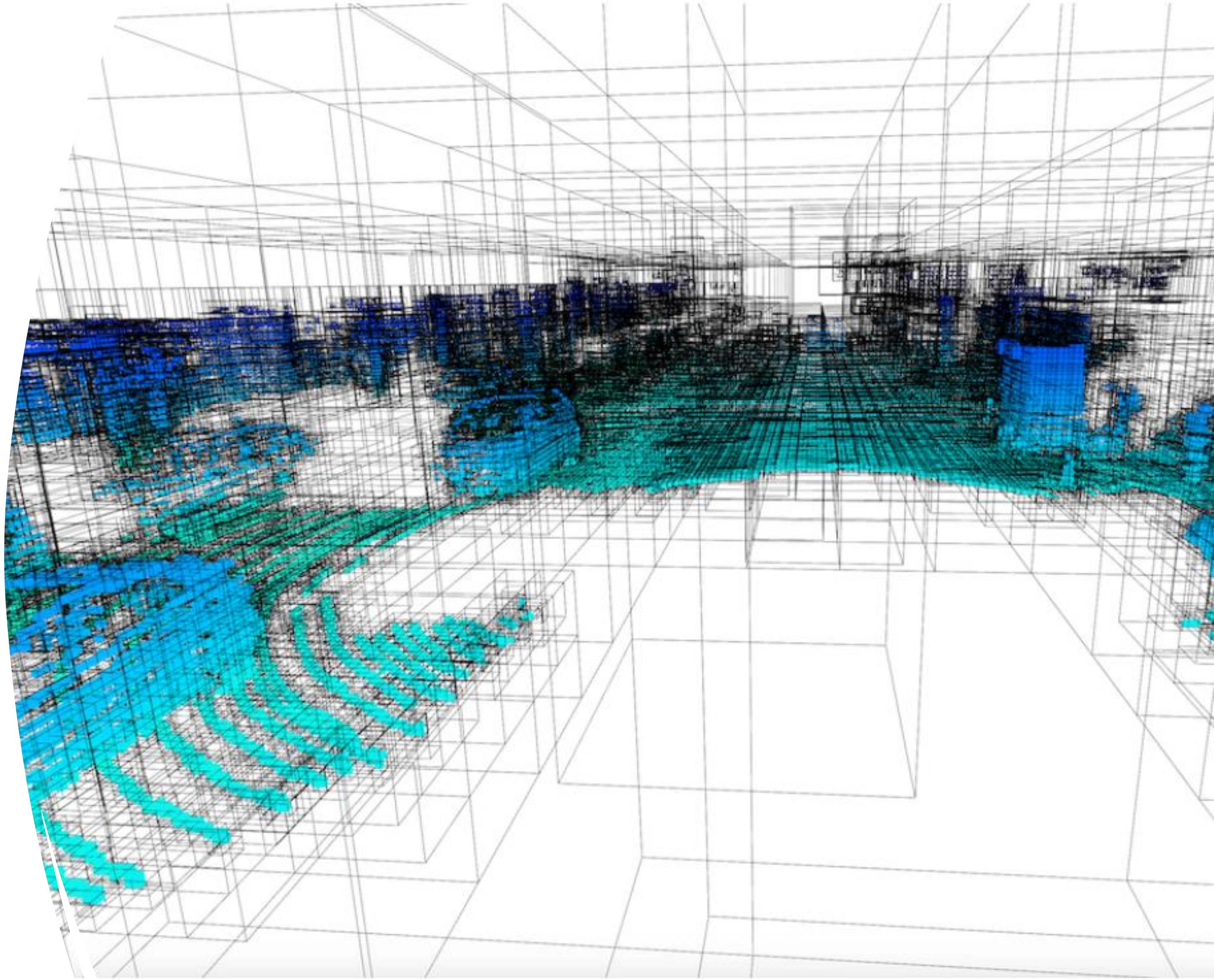
Hierarchical occupancy representation (other options, KD-Tree)

Pros:

- Compressive
- Hard to render (volume rendering)
- Coarse-to-fine representation

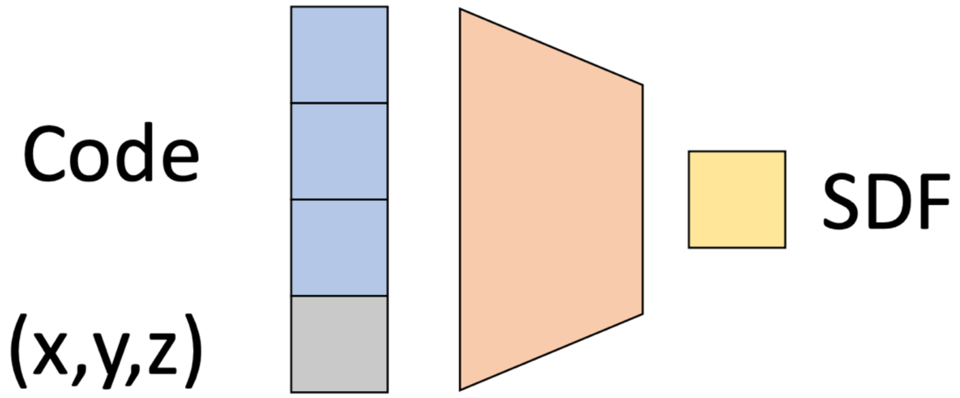
Cons:

- Non-trivial to learn/process (OctNet, Tree-structured Network)
- Expensive to update (KD-Tree)



Implicit Representation

Learning implicit function in the 3D continuous space to represent surfaces



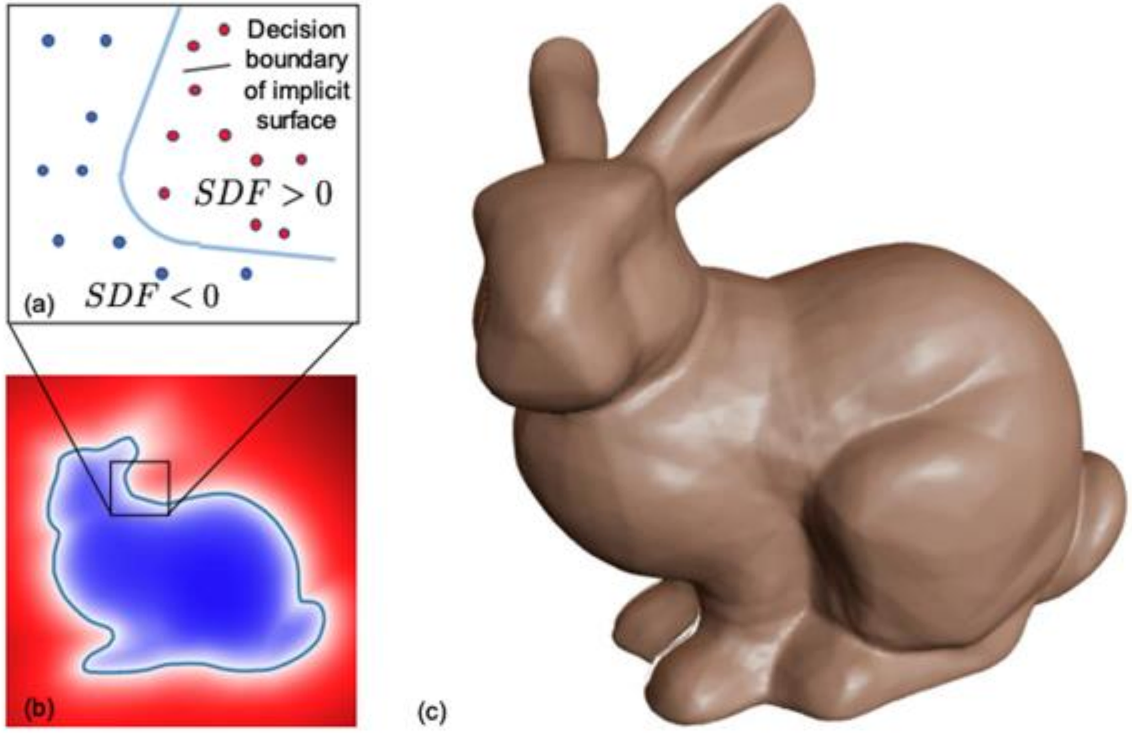


Image from: DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation

Implicit Representation

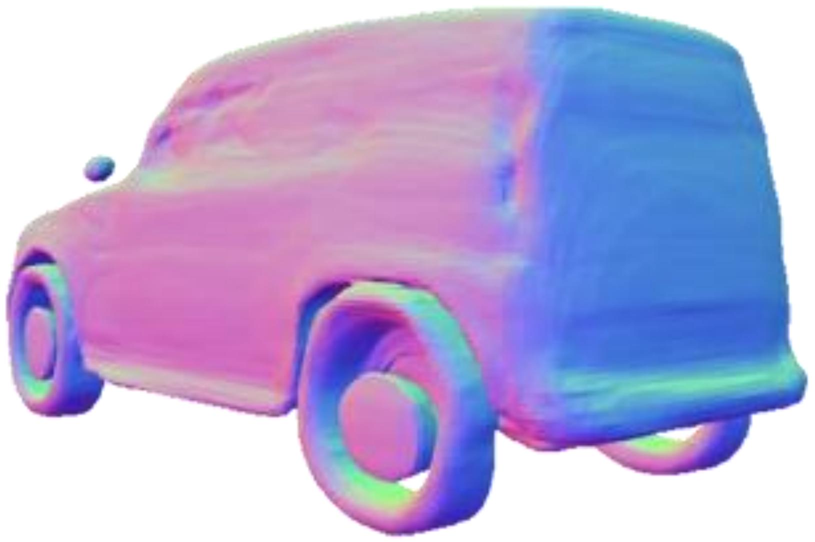
Signed distance function determines the distance of a given point x from the boundary of a shape

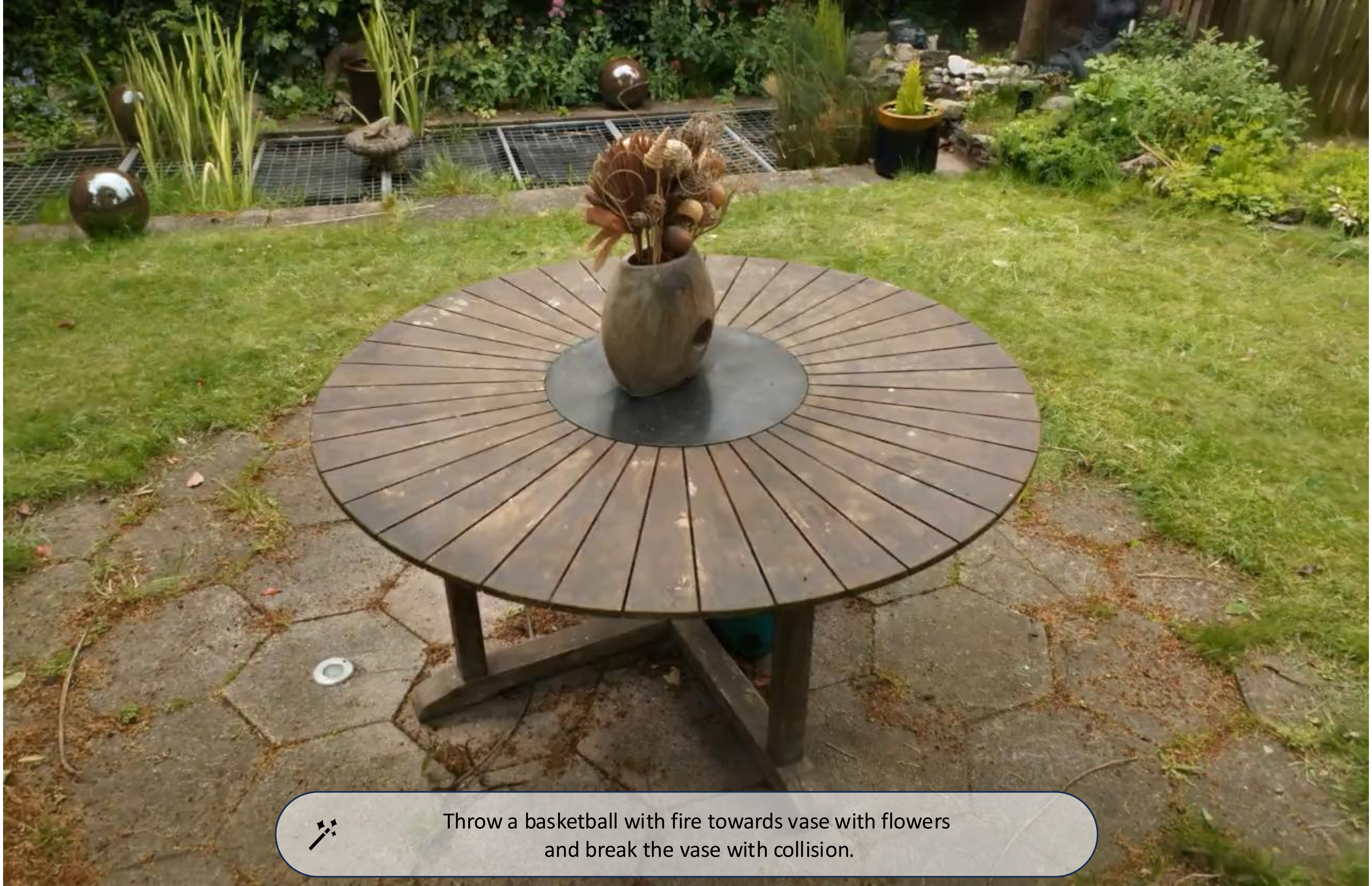
Pros:

- Flexible, easy to compose
- Expressive
- Easy to change topology
- Dense in space, no resolution loss

Cons:

- Hard to render (ray marching)
- Additional steps to extract surface (marching cube)



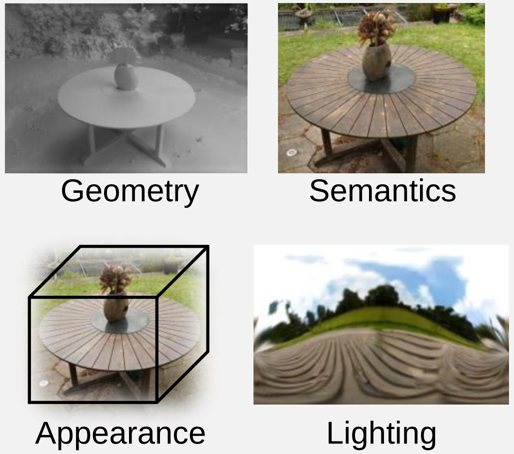

Throw a basketball with fire towards vase with flowers
and break the vase with collision.

AutoVFX: Let's make LLM code for y

Input Video

Scene Modeling
(*SfM, GSplats, Neural SDF, SAM, HDR light*)

Holistic Scene Models



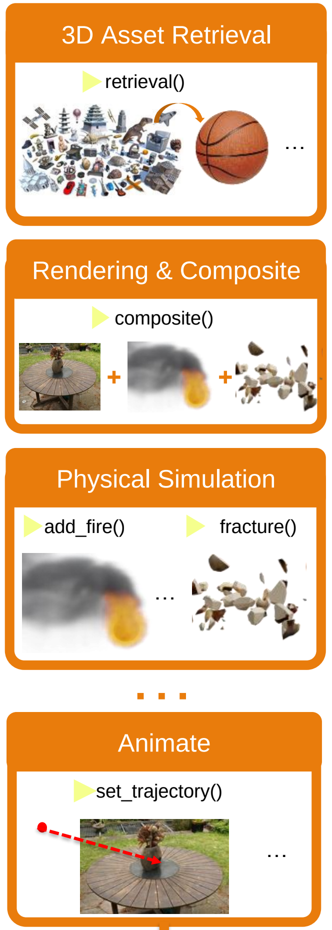
Input Instruction

"Throw a ***basketball with fire*** *towards the* ***vase****"*

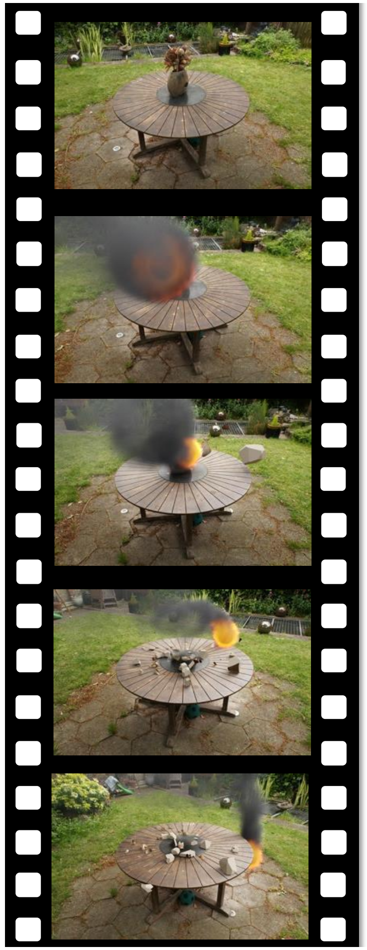
LLM Code Generation

```python
def auto_vfx(video):

    # scene modeling and understanding
    scene = scene_modeling(video)

    # generate object asset
    ball = retrieve(”basketball”)
    if ball is None:
        ball = generate(”basketball”)
    fireball = add_fire(ball)

    # segment and localize existing objects from the scene
    vase = localize_object(scene, ”vase”)
    vase = enable_fracture(vase)

    # animate new objects
    traj = set_trajectory(fireball, vase)
    animated_objs = set_animation(fireball, vase, traj)

    # physical interaction with existing scene
    output_scene = simulate(scene, animated_objs)

    # render and composite
    output_video = composite(video, render(output_scene))

    return output_video
```

VFX Modules



blender



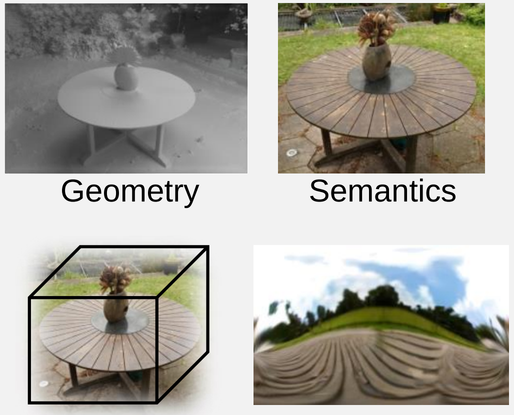
AutoVFX: Physically Realistic Video Editing from Natural Language Instructions, arXiv soon

AutoVFX: Let's make LLM code for you

Input Video

Scene Modeling
(*SfM, GSplats, Neural SDF, SAM, HDR light*)

Holistic Scene Models



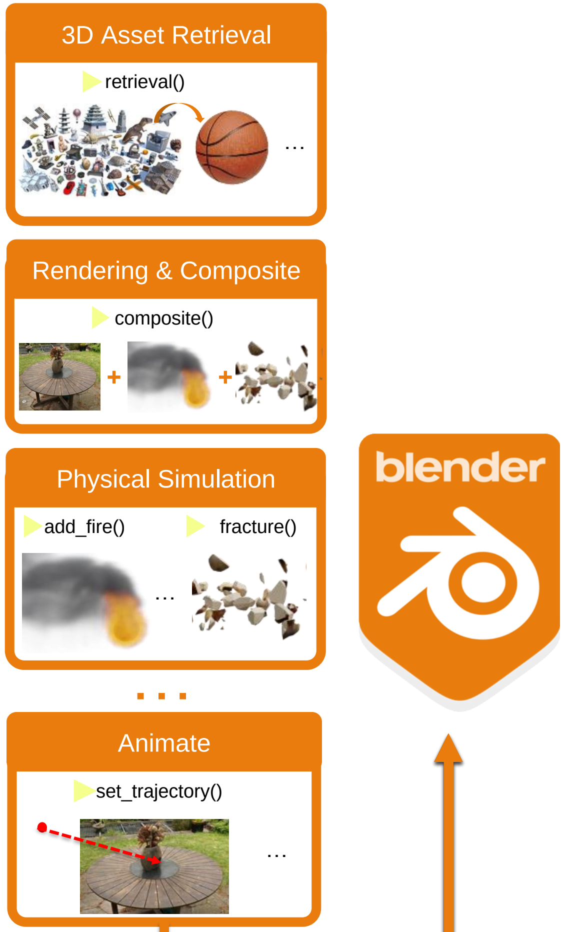
Input Instruction

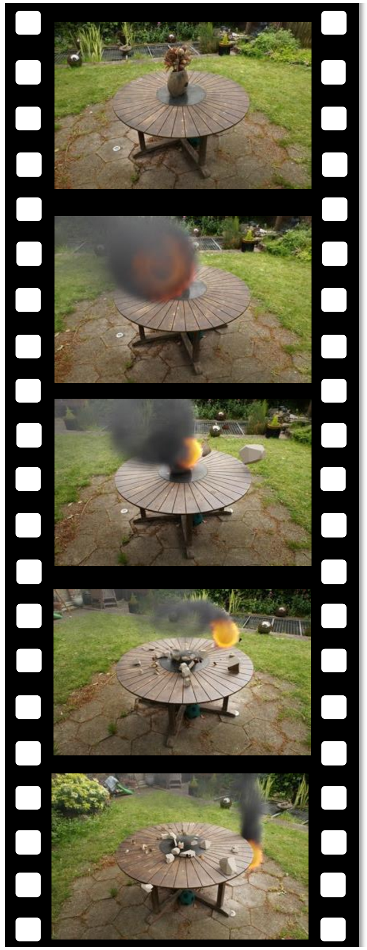
*"Throw a **basketball with fire** towards the **vase**"*

LLM Code Generation

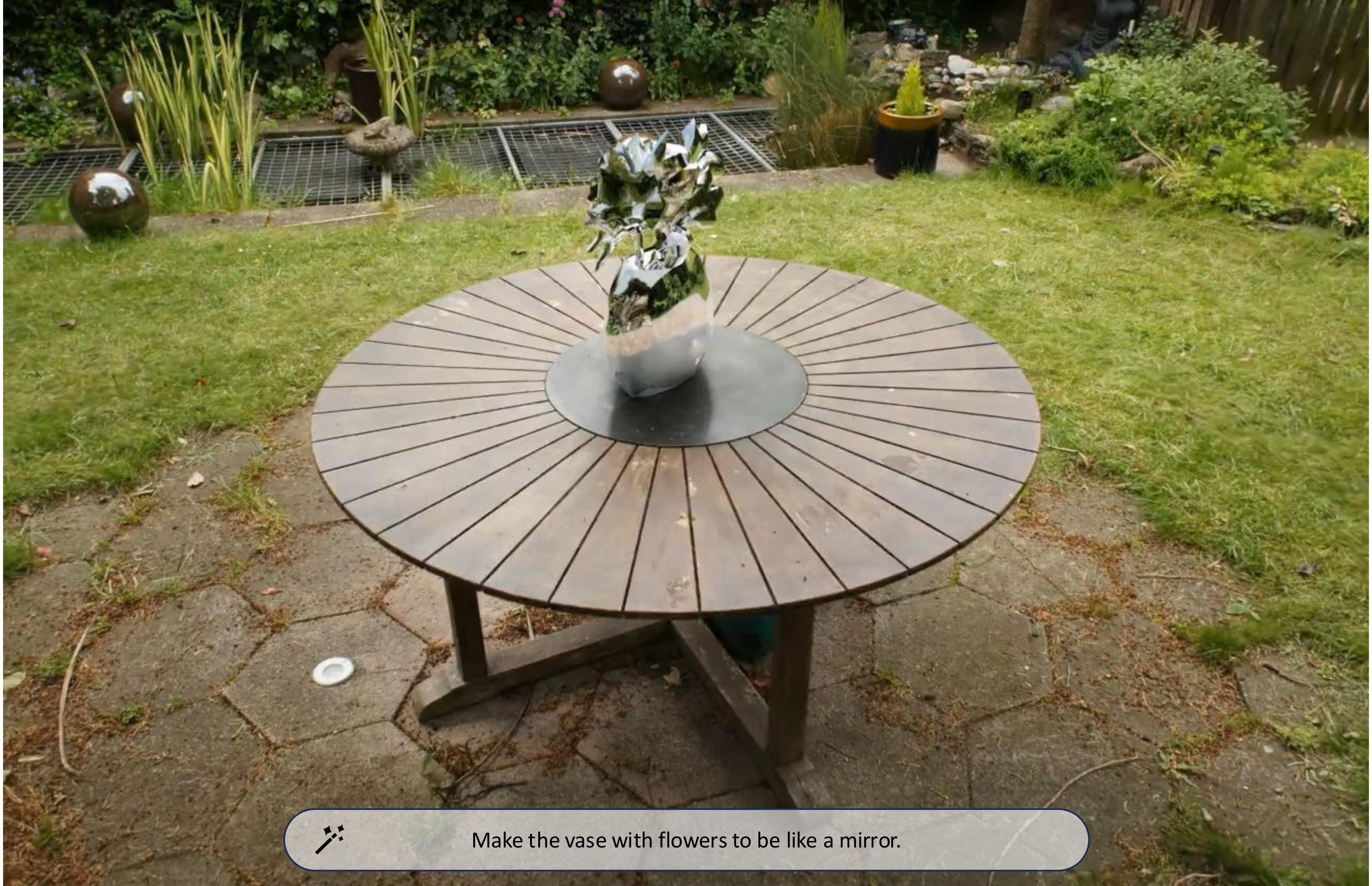
```python
def auto_vfx(video):

    # scene modeling and understanding
    scene = scene_modeling(video)

    # generate object asset
    ball = retrieve("basketball")
    if ball is None:
        ball = generate("basketball")
    fireball = add_fire(ball)

    # segment and localize existing objects from the scene
    vase = localize_object(scene, "vase")
    vase = enable_fracture(vase)

    # animate new objects
    traj = set_trajectory(fireball, vase)
    animated_objs = set_animation(fireball, vase, traj)

    # physical interaction with existing scene
    output_scene = simulate(scene, animated_objs)

    # render and composite
    output_video = composite(video, render(output_scene))

    return output_video
```

VFX Modules

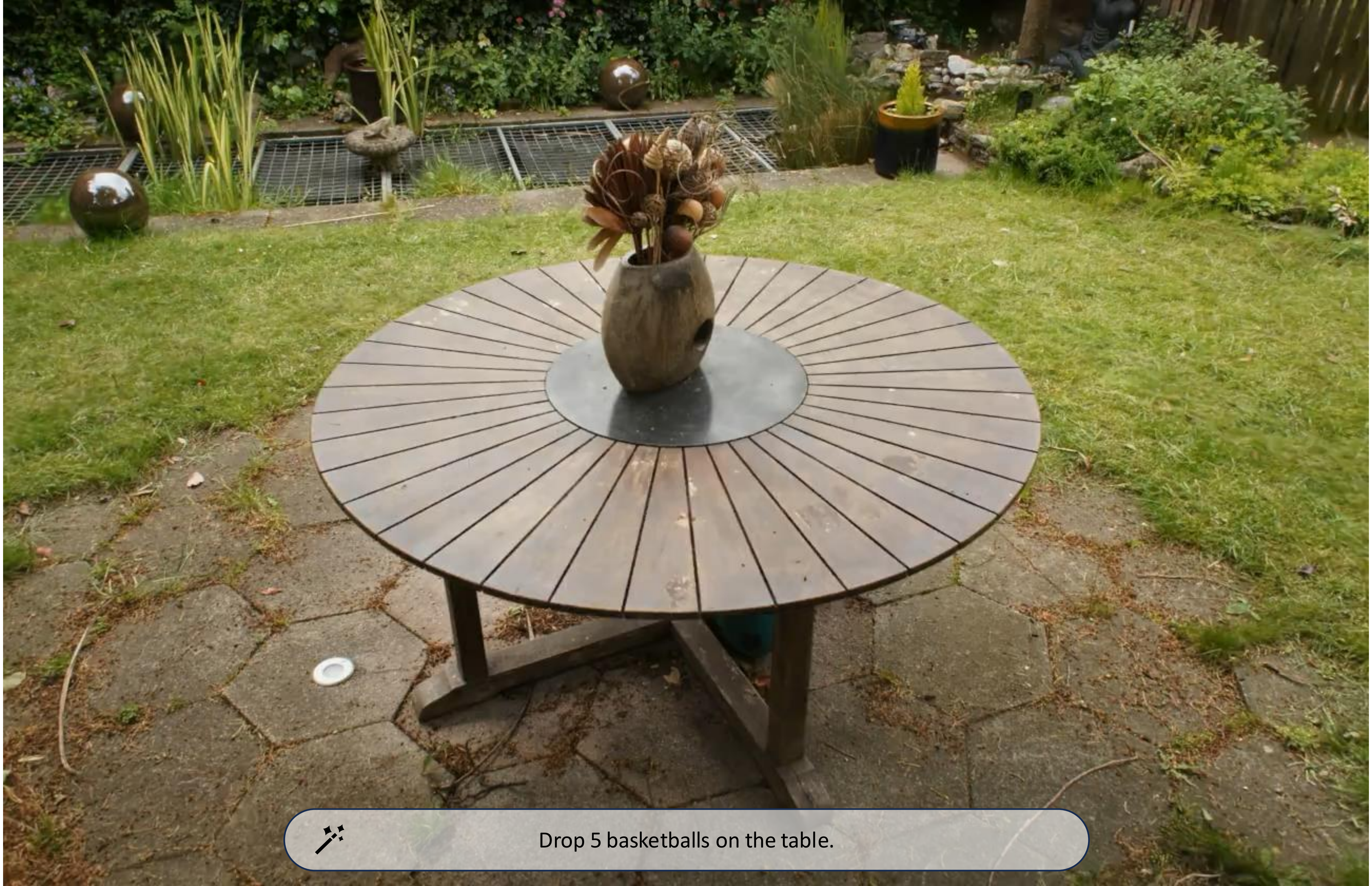




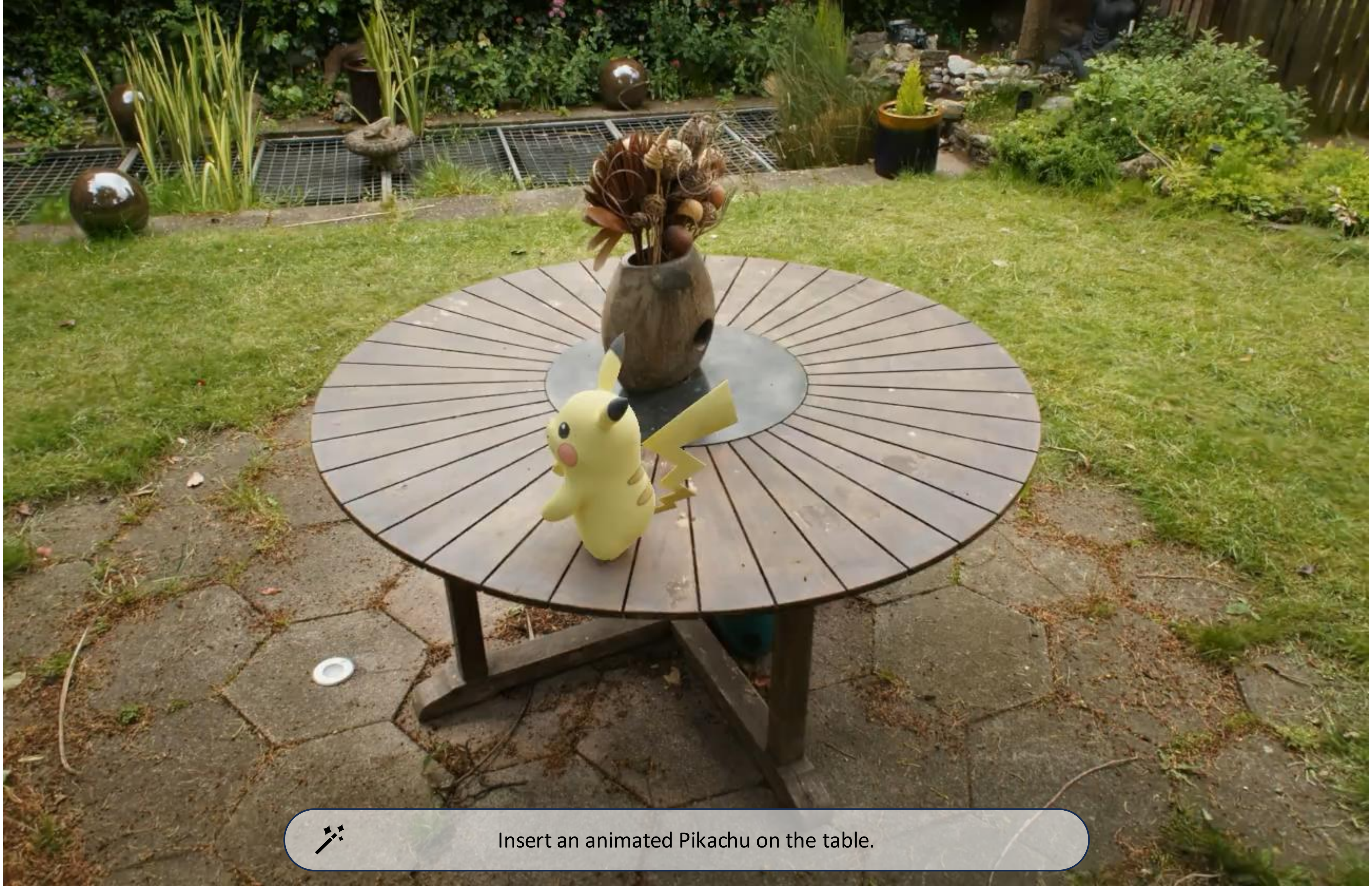
AutoVFX: Physically Realistic Video Editing from Natural Language Instructions, arXiv soon



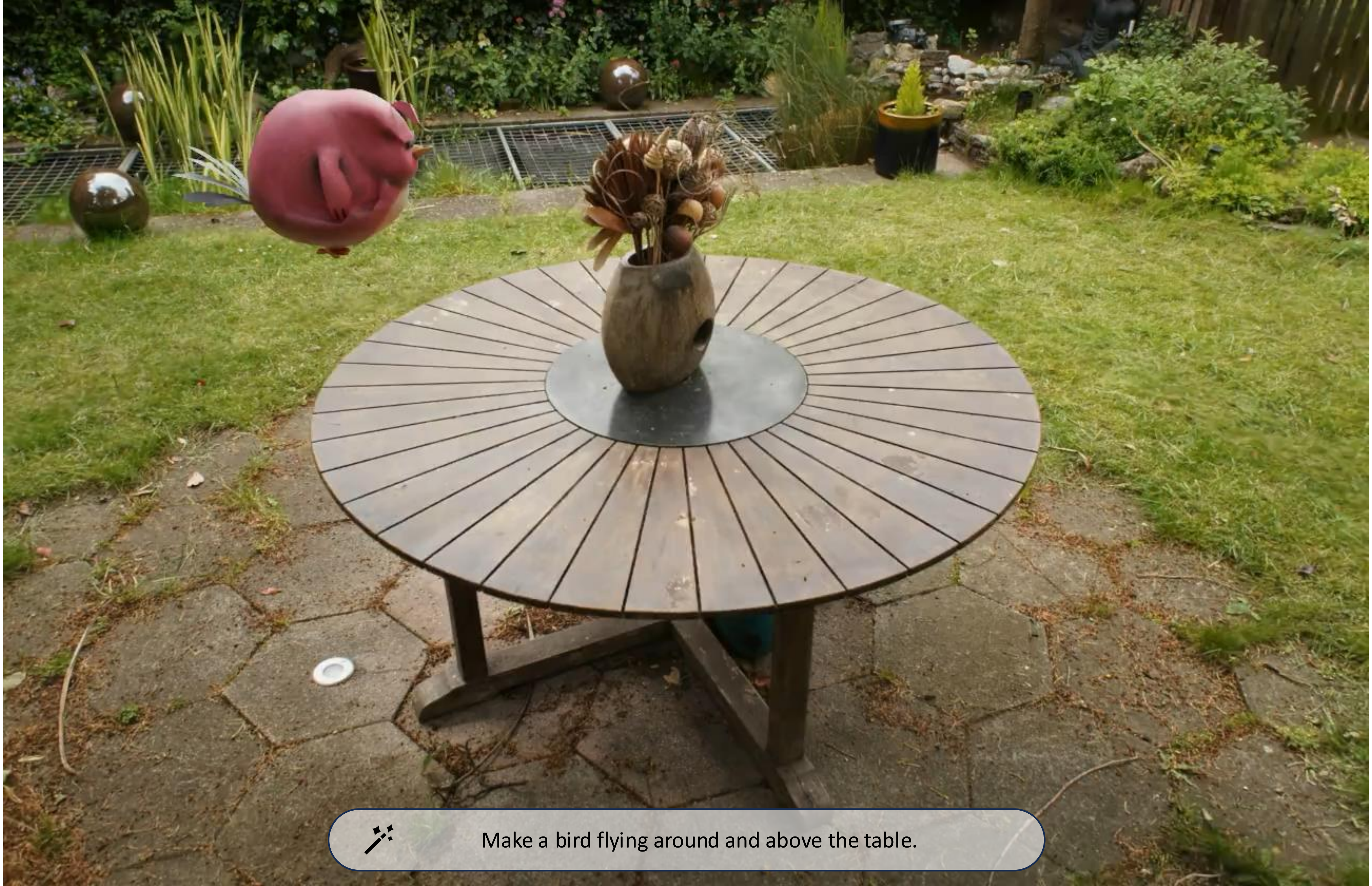
Make the vase with flowers to be like a mirror.

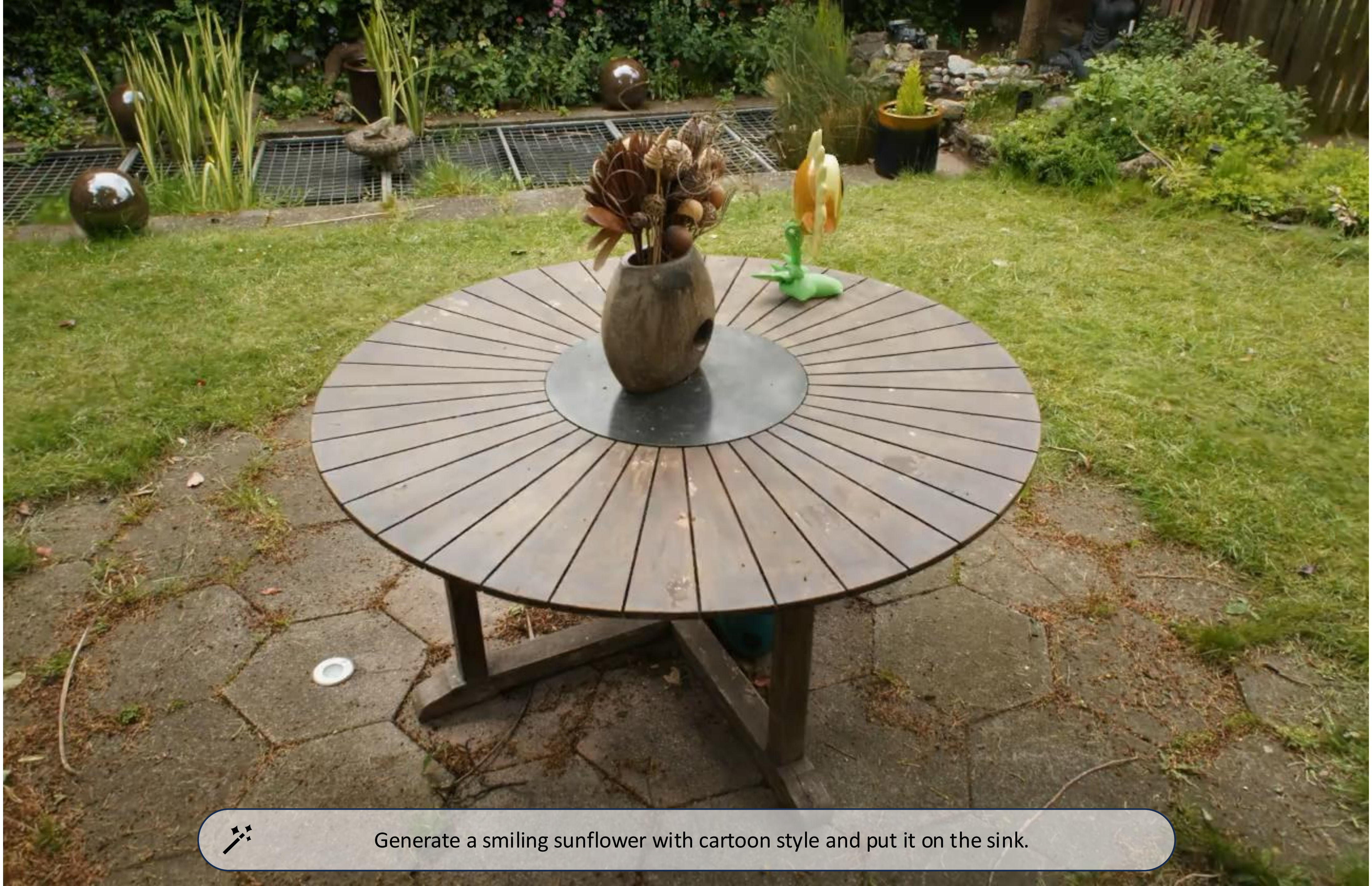


Drop 5 basketballs on the table.



Insert an animated Pikachu on the table.


Make a bird flying around and above the table.


Generate a smiling sunflower with cartoon style and put it on the sink.


Insert an animated dragon moving above and around the floor.

Put a Tony Stark on the floor covered with smoke.


Simulate books falling from the sofa.

Setup a camp fire in the middle of the floor.

Drop four barrels onto the floor: one mirror-like, one with fabric textures,
one resembling pavement, and one unchanged.


Break the sculpture.


Insert a physics-enabled Benz G 20 meters in front of us with random 2D rotation. Add a Ferrari moving forward.

Others

- Surfels / Polygon Soup
- Tetrahedron mesh (tets).
- Stixels
- Radiance Field
- KD-tree
- Voxel hashing

2.5D Representation

2D Tensor, Each element encode distance (optional: intensity)

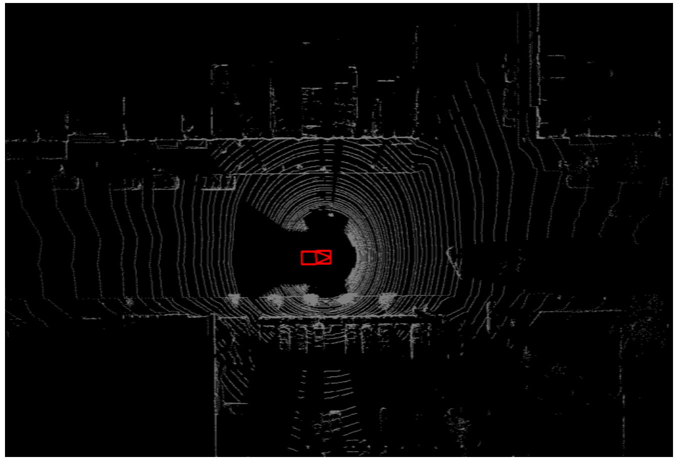
Pros:

- 2D tensor, compact and efficient
- Off-the-shelf CNN perception
- Coupled with state/action space (Birds' eye view)
- Coupled with raw sensor measures

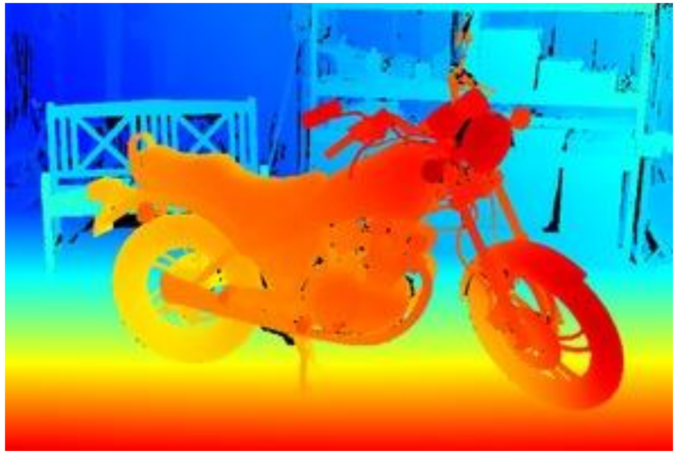
Cons:

- Information loss along a dimension
- Resolution loss due to rasterization
- Neighbor pixels can be far in 3D

equirectangular LiDAR



Birds-eye-view LiDAR



Perspective Depth Image

Point Cloud Representation

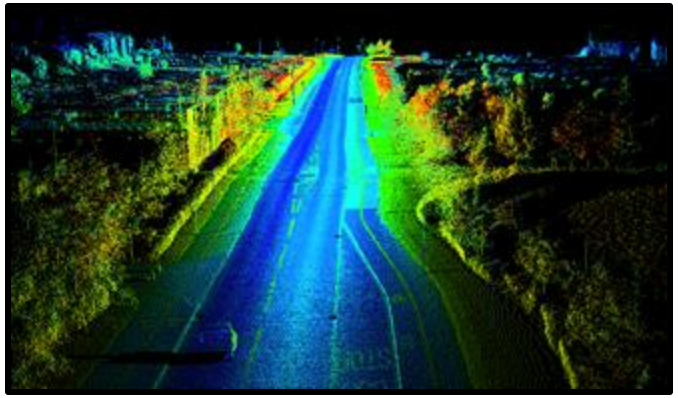
3D unordered point, each encodes spatial location

Pros:

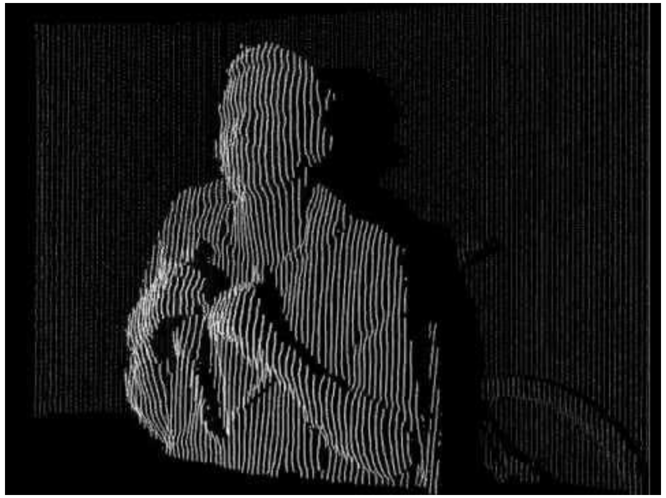
- No geometry loss
- Memory and computational efficient processing

Cons:

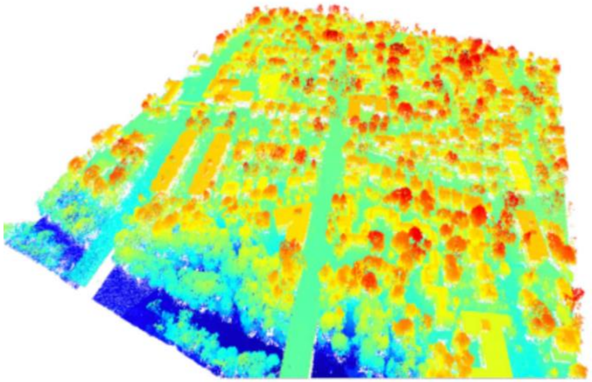
- No topology; No occupancy/surface
- Need to splat or hole filling for rendering
- Hard to retrieve neighbors (need kd-tree, r-tree, octree, etc)



Point Cloud from Surveying Lidar



Point Cloud from Kinect



Point Cloud from Surveying Lidar

Voxel Representation

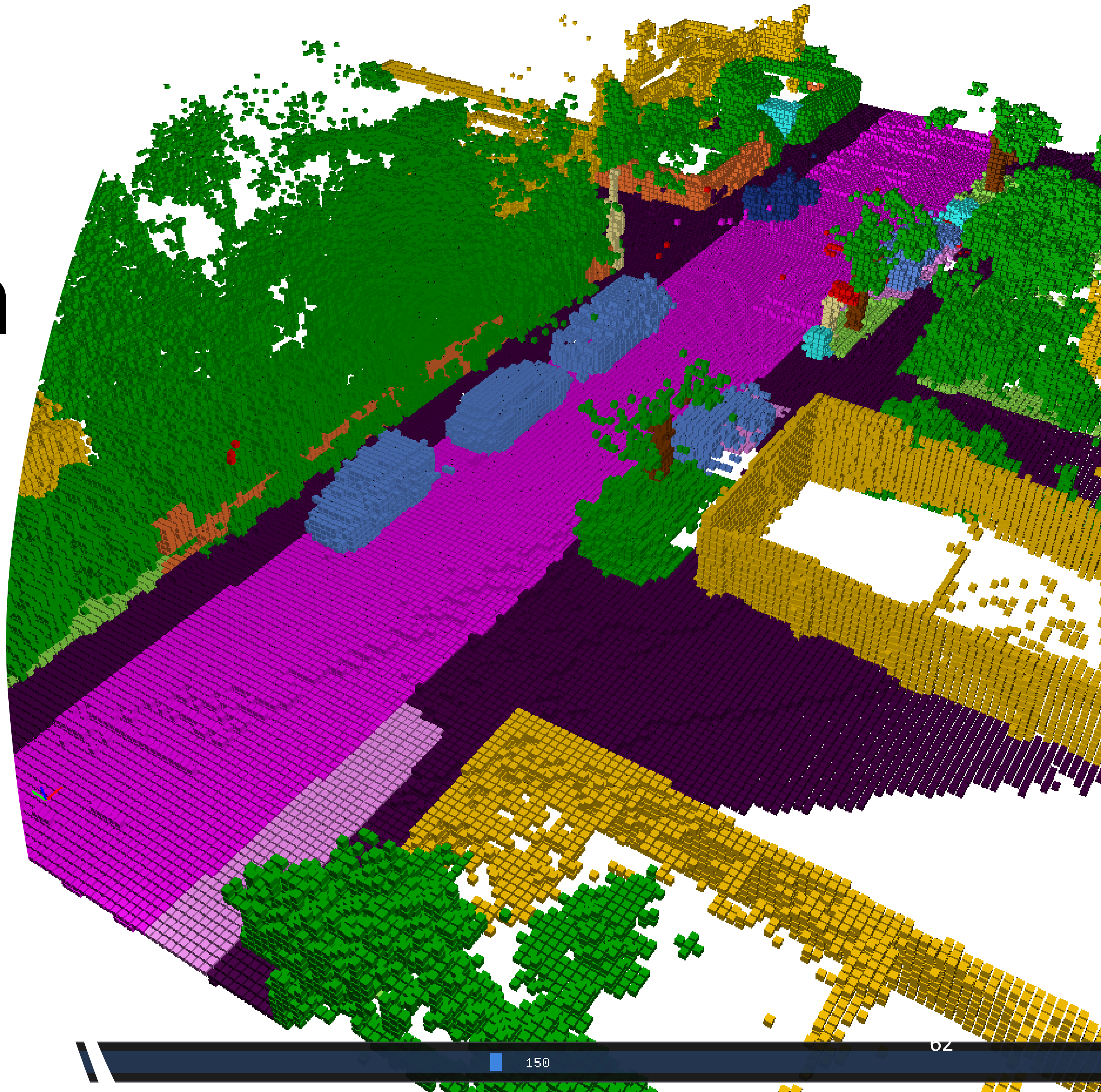
Dense grid, each voxel encodes occupancy

Pros:

- Easy to learn/process (3D CNNs)
- Can be accurate (with very high-resolution)
- Easy to compute occupancy/freespace

Cons:

- Intensive memory, requires special data structure to scale up
- Hard to render (volume rendering)



Octree Representation

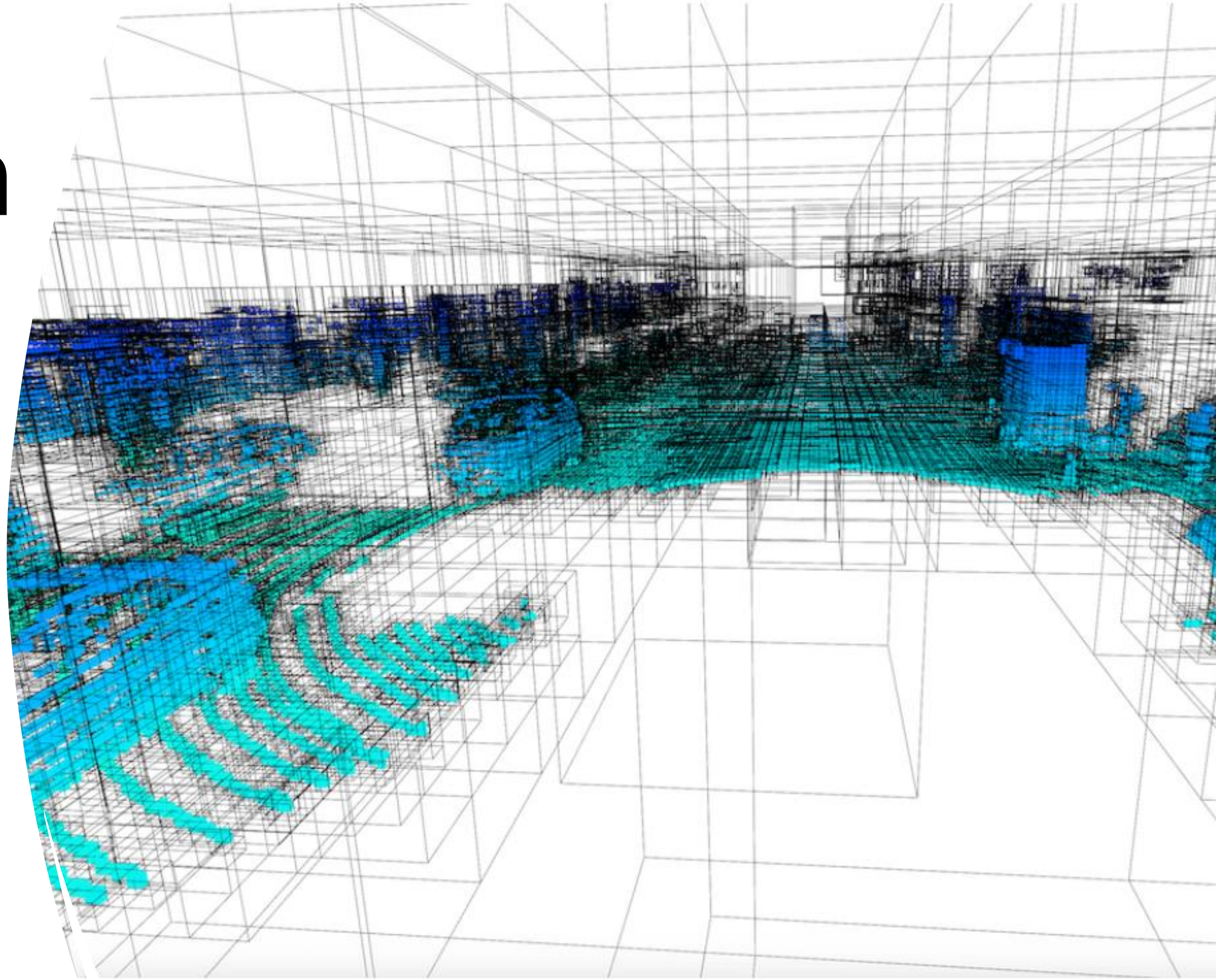
Hierarchical occupancy representation (other options, KD-Tree)

Pros:

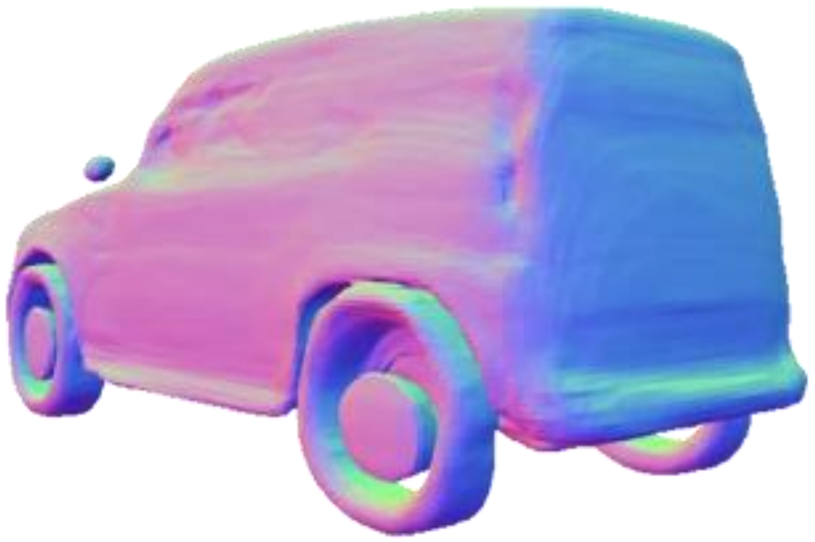
- Compressive
- Hard to render (volume rendering)
- Coarse-to-fine representation

Cons:

- Non-trivial to learn/process (OctNet, Tree-structured Network)
- Expensive to update (KD-Tree)



Implicit Representation



Signed distance function determines the distance of a given point x from the boundary of a shape

Pros:

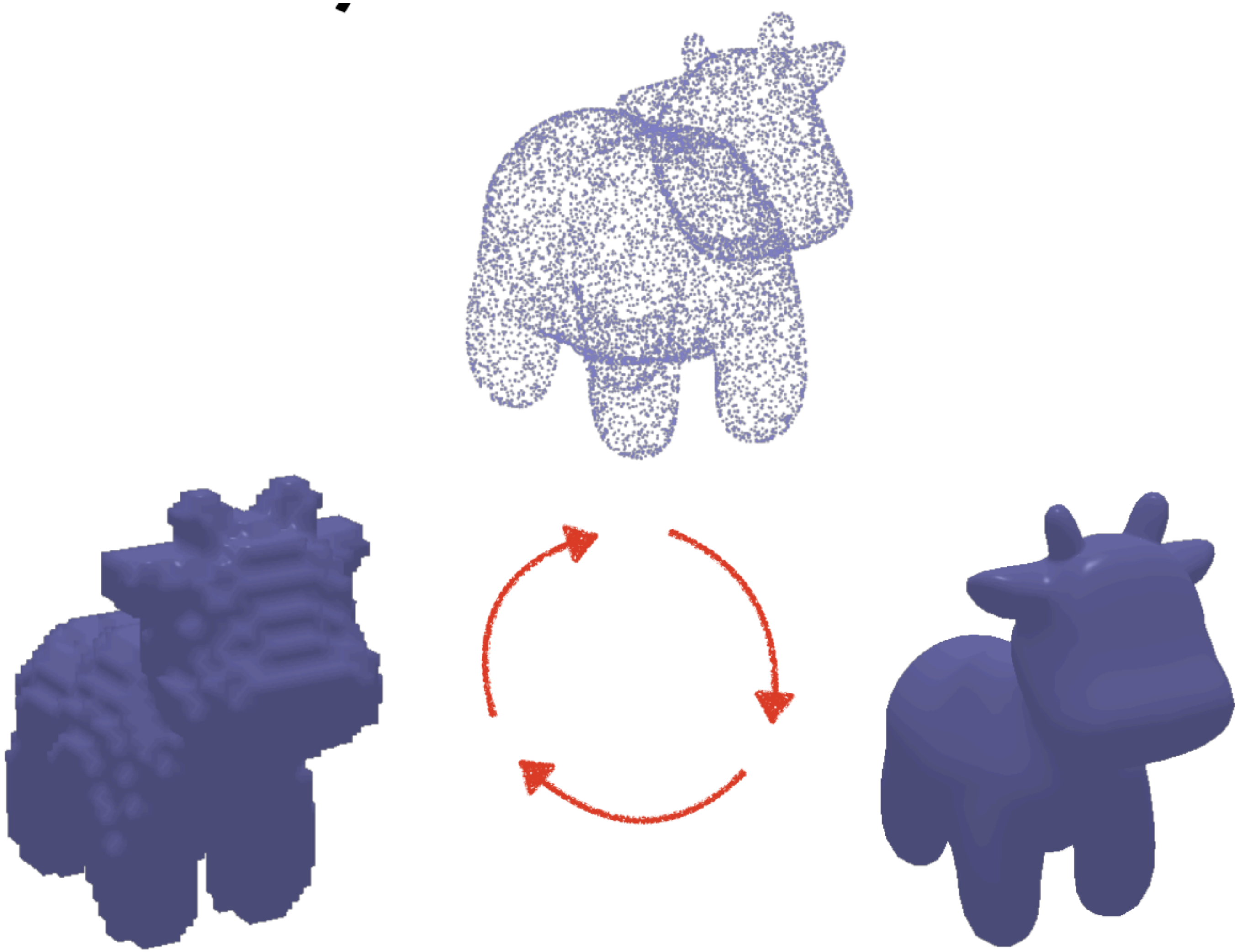
- Flexible, easy to compose
- Expressive
- Easy to change topology
- Dense in space, no resolution loss

Cons:

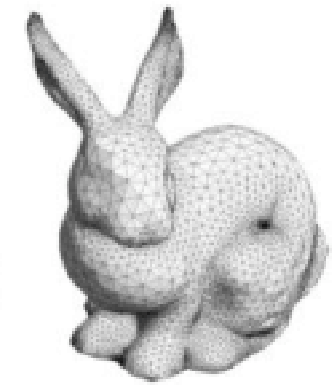
- Hard to render (ray marching)
- Additional steps to extract surface (marching cube)

Quiz 3: Conversions?

- Points
- Voxel
- Mesh
- SDFs

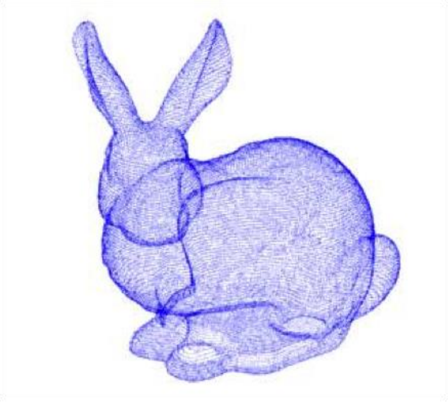


Key Challenge: Representations

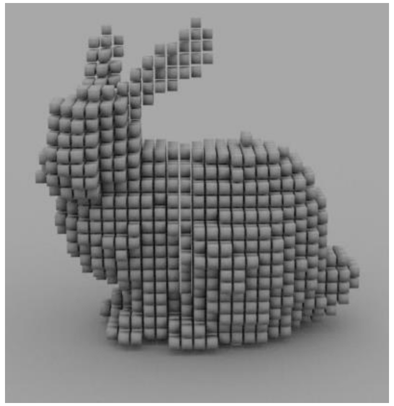


Mesh

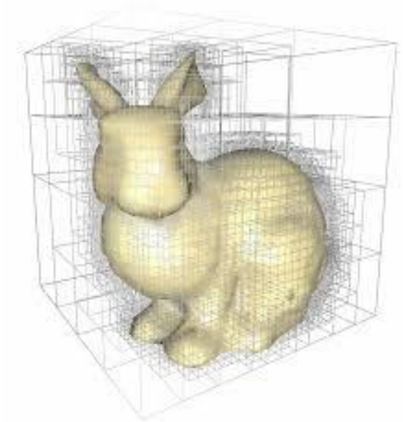
2.5D



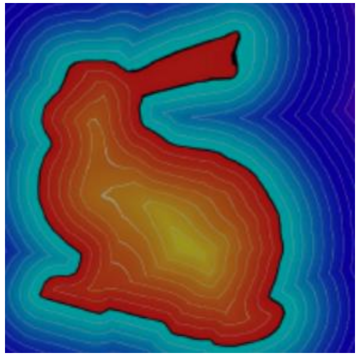
Point Cloud



Voxel



Octree



SDF

- What representation better suits my sensor?
- What representation makes my perception easier?
- What representation helps my downstream tasks?