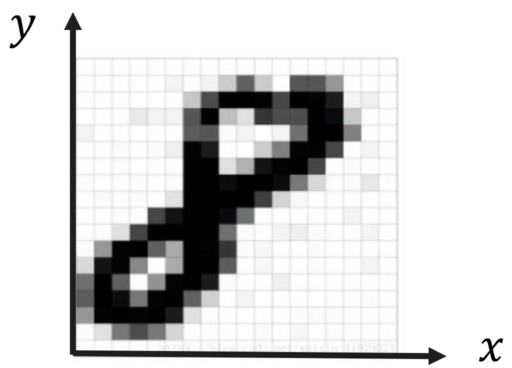
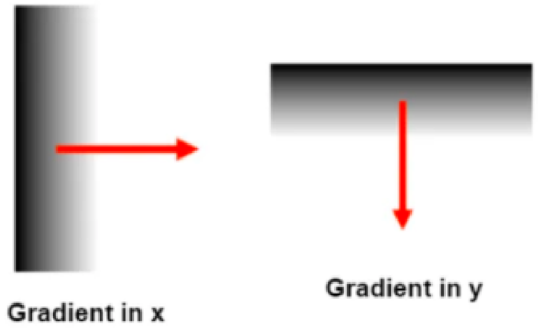
Research Progress

2022.01.21

Guan Yunyi

1. Digital images





- 2D digital image can be represented by an discrete function $F(x, y)$

- x and y are the coordinates
- $F(x, y)$ represents the grayscale or color value at pixel (x, y)

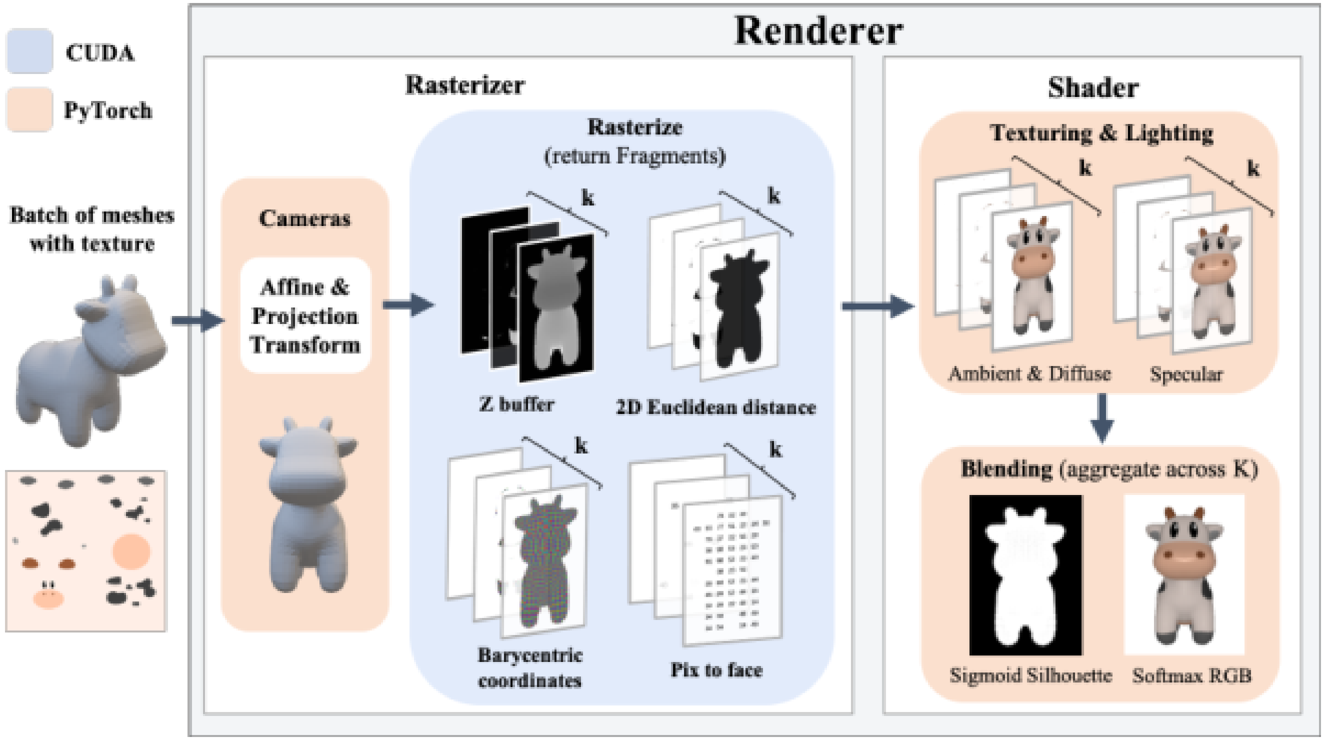
-> To determine (x, y), first determine <u>coordinate system</u>

Meaning of <u>image gradient</u>: change in the grayscale at that pixel

2. Pytorch3D Rendering pipeline

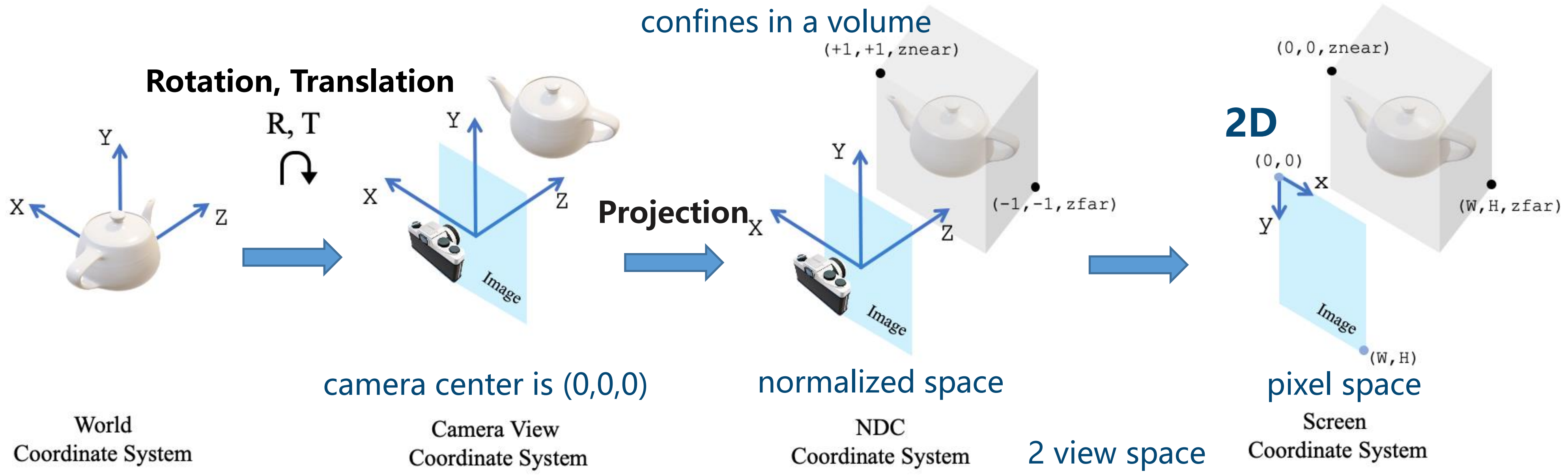
Rendering:

transform 3D continuous objects to 2D discrete pixels and shade them



- **Rasterizer**: transform coordinate from 3D to 2D
  -> just like taking pictures with a virtual camera
- **Shader**: texturing, lighting, blending…
  -> just like painting the white image delicately

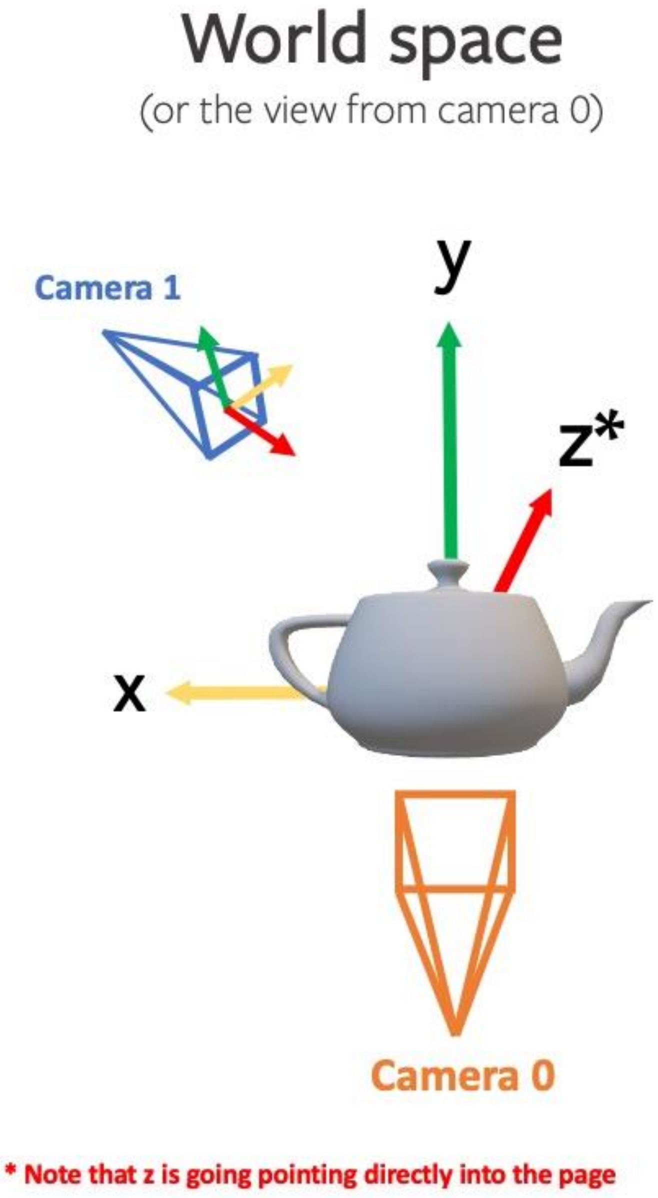
2.1 Rasterizer - Transforms in 4 Coordinate Systems

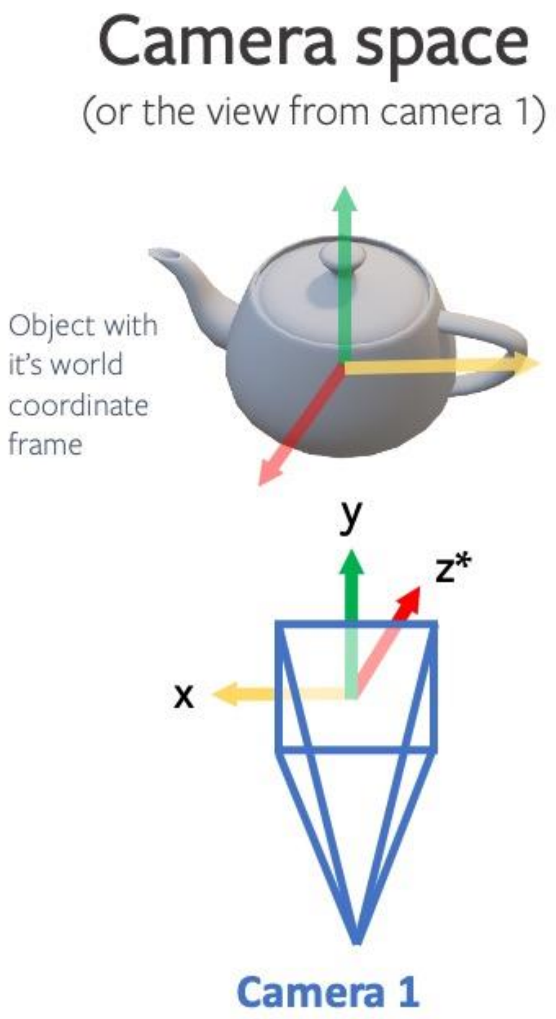


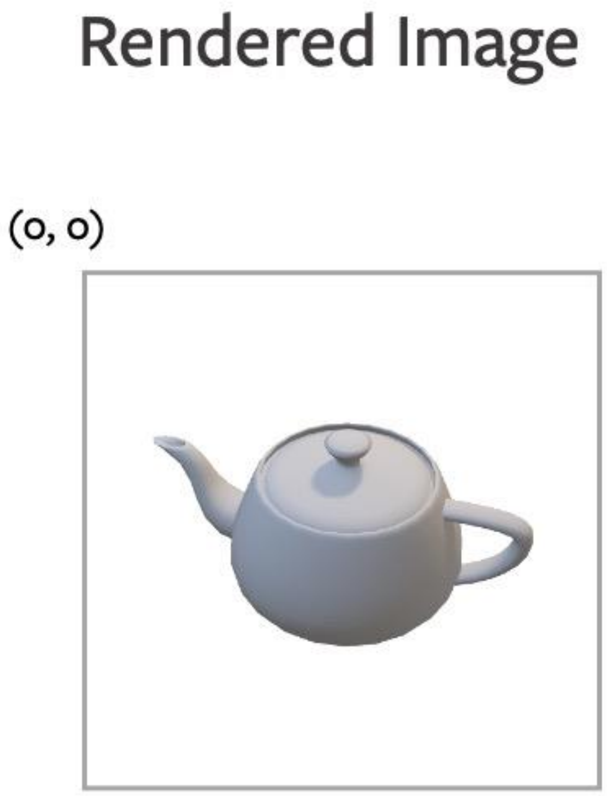
- Cameras transform a 3D object to 2D by:
- transforming it from world system to camera system via <u>extrinsic (R,T)</u>
- projecting it to view space S via projection $P = K[R|T]$,
  where <u>intrinsic</u> camera parameters <u>K</u> define S.
- All the transforms are defined purely by R, T and K.
  -> users can define cameras in any coordinate system and for any transforms

Cameras are Python objects, and can compute gradients via autograd

2.1 Example - Transform from World to Screen space



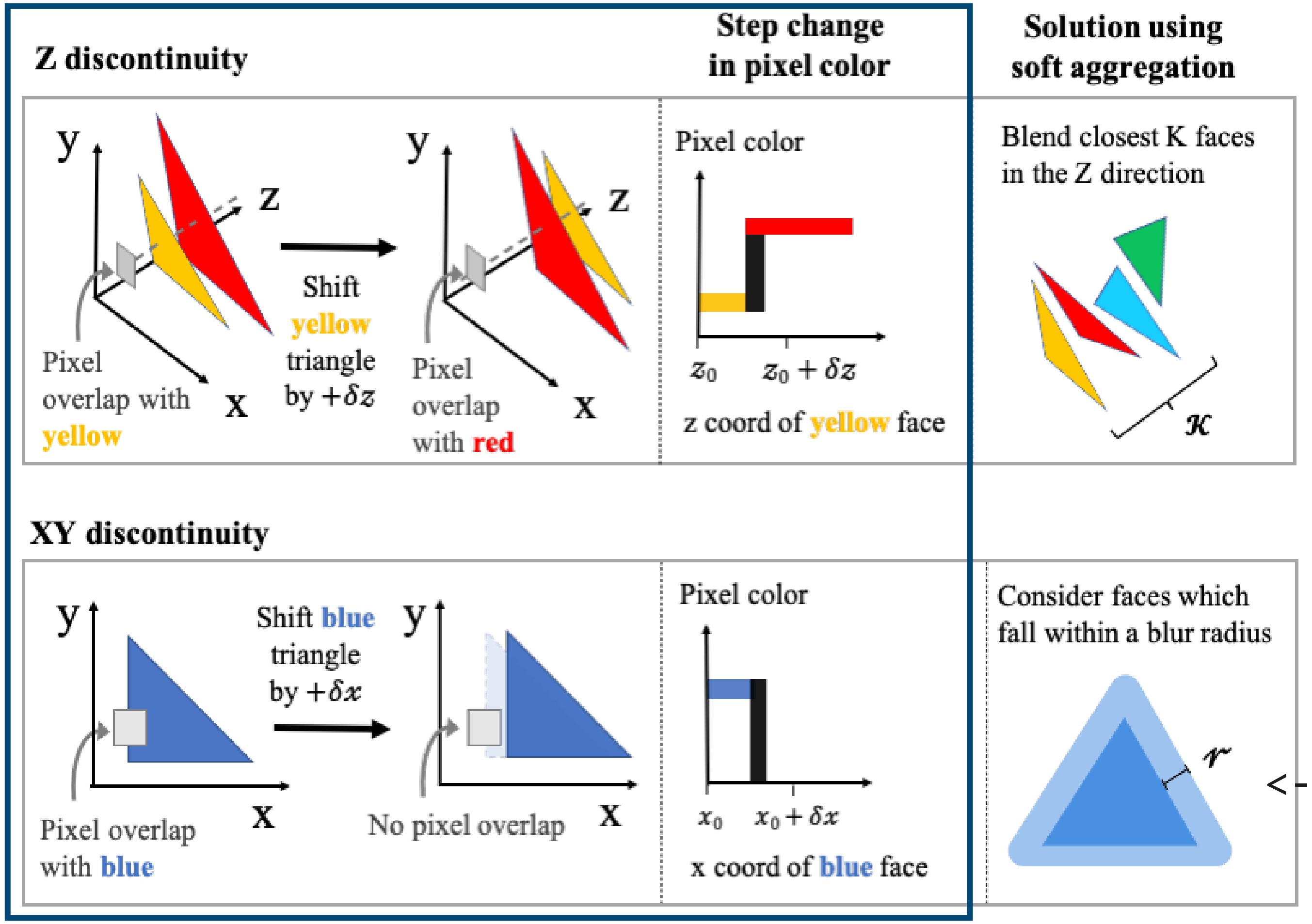




2.2 Shader – 2 stages

- Shaders consume the Fragment data produced by the rasterizer, and compute pixel values of the rendered image:
- first computing K values for the pixel, then blending them to give a final pixel

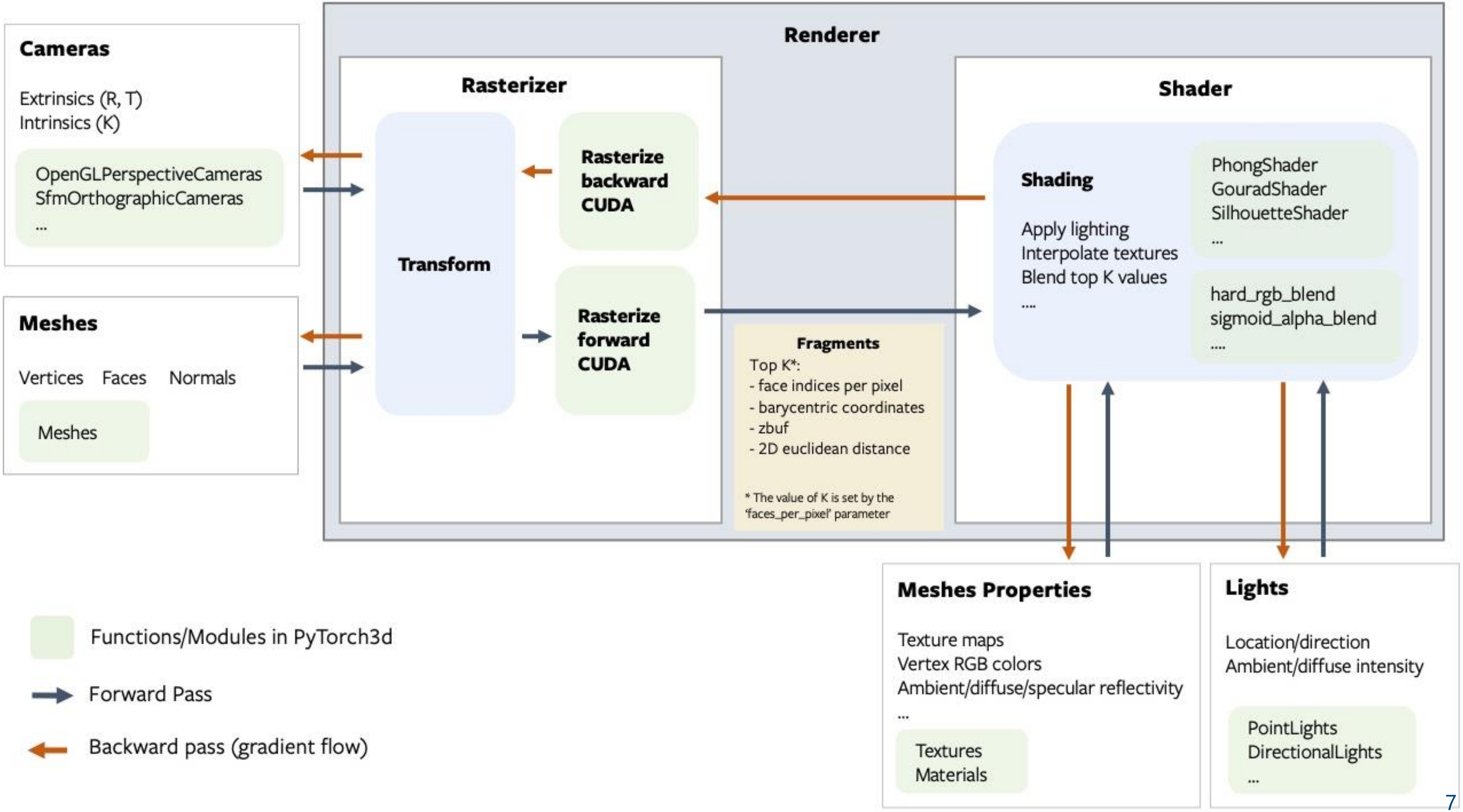
Problems in traditional rendering



Shaders are Python objects and compute gradients via autograd.

<- in Rasterizer

2.3 PyTorch3D rendering pipeline

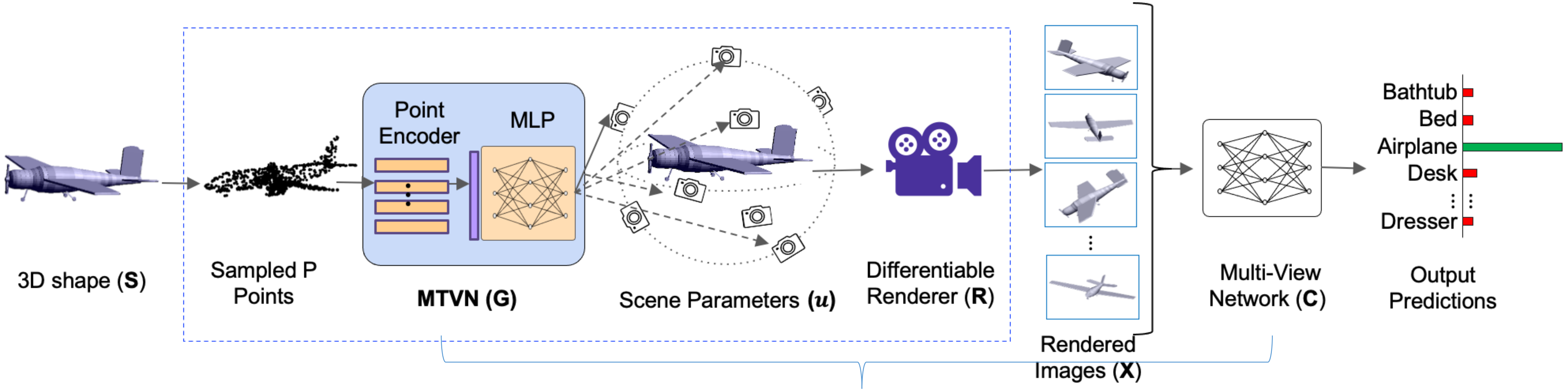


3. Why differentiable rendering?

- Differentiable rendering is like the reverse operation of rendering:

  differentiable -> able to invert rendering step (backpropagation)

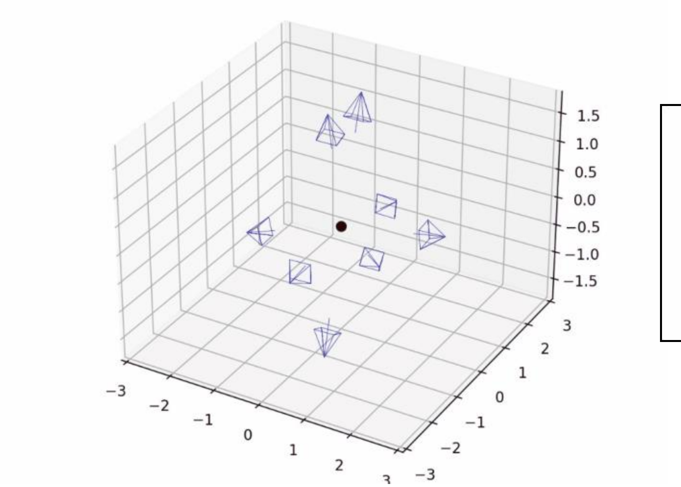
  -> supervise 3D with 2D loss rather than 3D
- Application: tasks need rendering gradient, e.g. 3D reconstruction

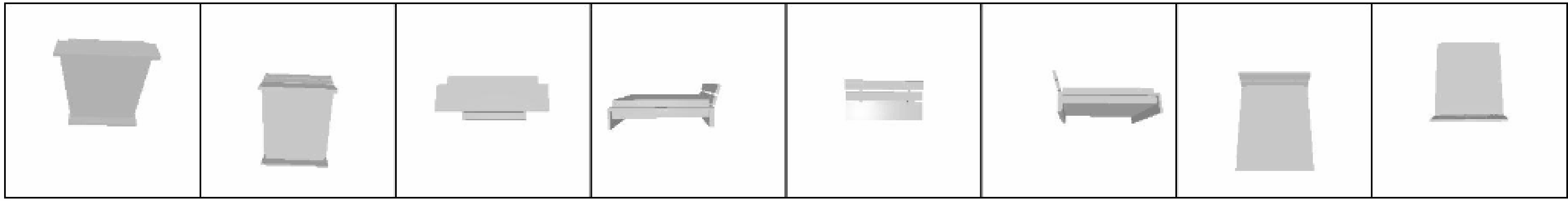


trained on the same loss

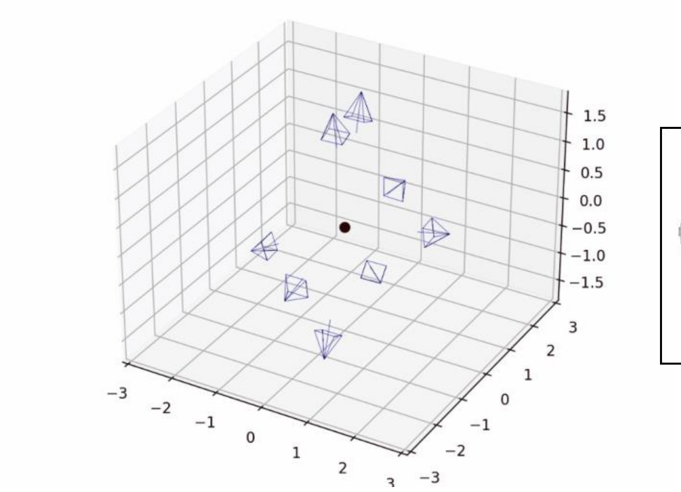
What I did- Result of MVTN of learned_spherical (with MLP)

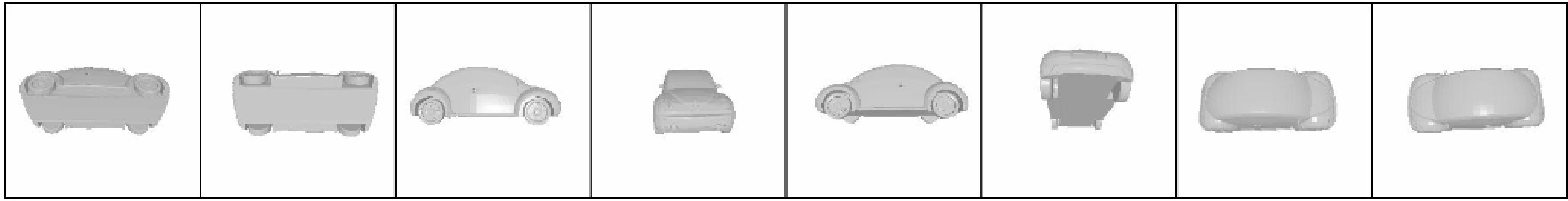
- Cameras and rendering when epoch =3,6,9,12,15





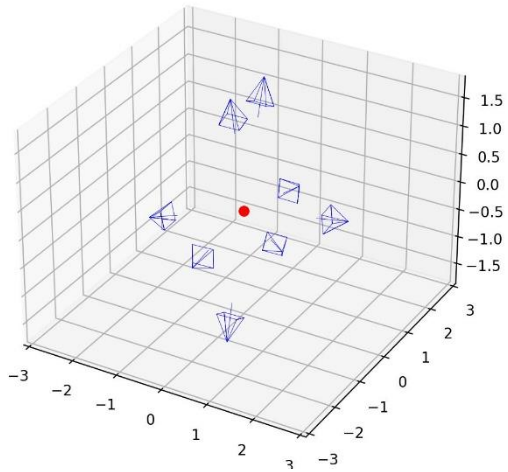
- Best accuracy = 75.61%, best loss = 0.864

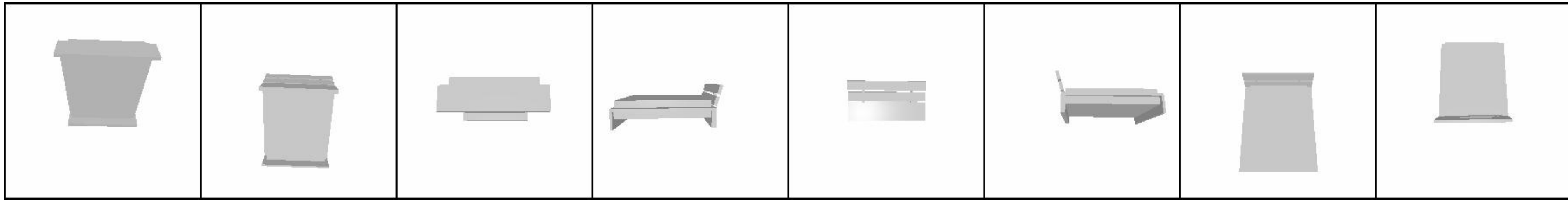




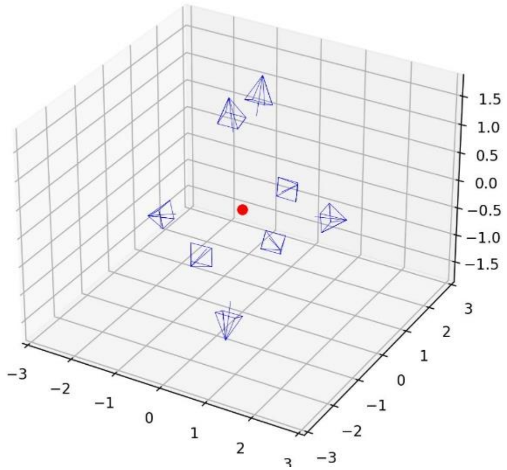
What I did- Result of MVTN of spherical (without MLP, fixed view)

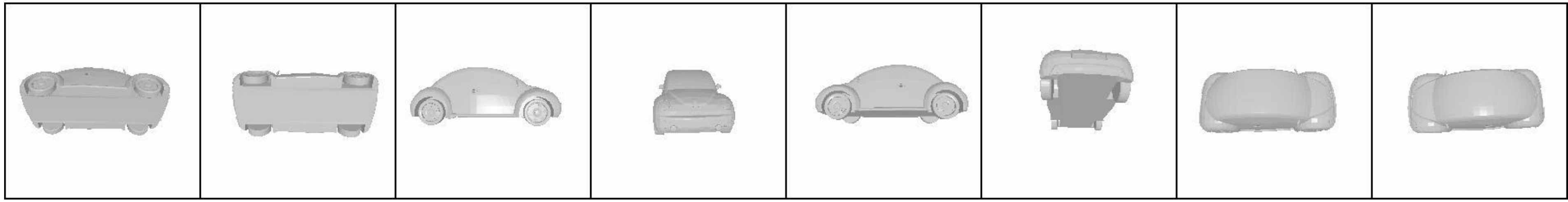
- Cameras and rendering when epoch =3,6,9,12,15





- Best accuracy = 74.80%, best loss = 0.996





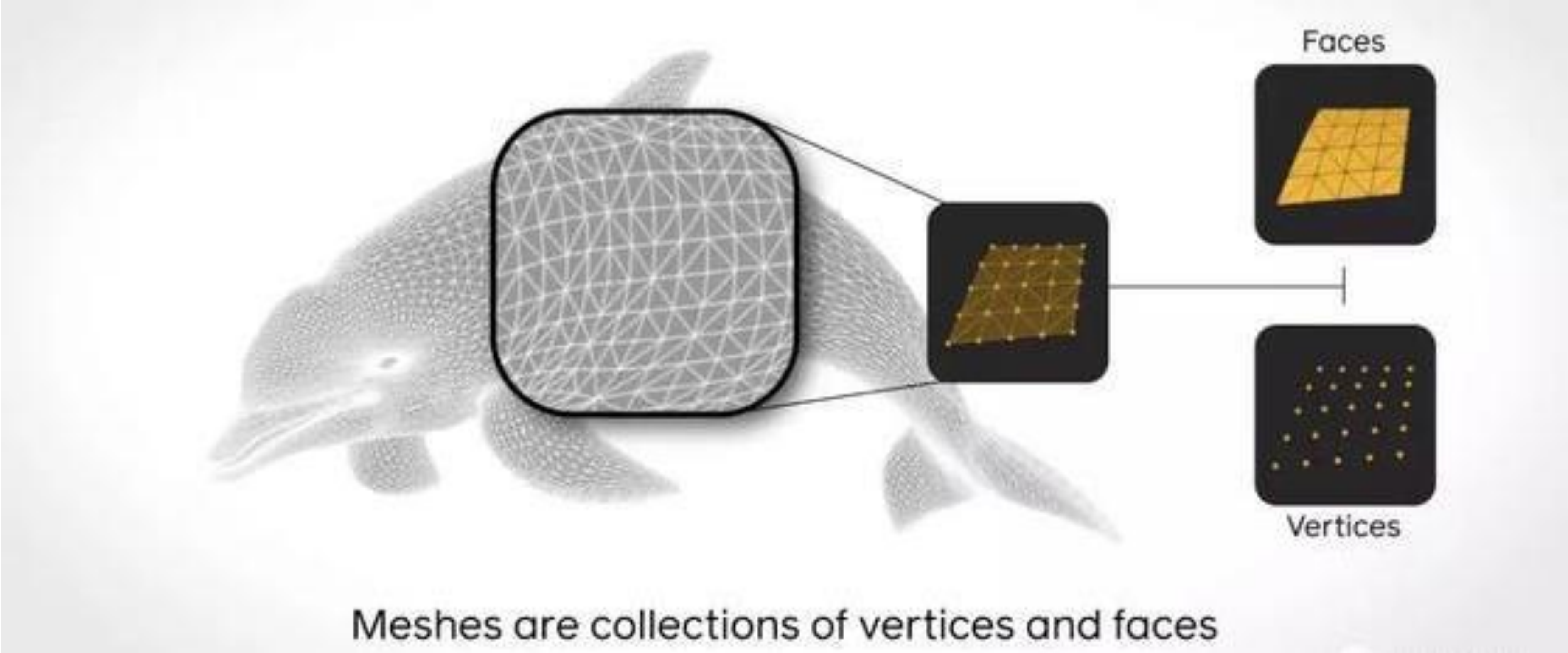
Next to do

- Coding part:

- Modify MVTN to save cameras and rendered images for RotationNet
- How to determine the rank of scene parameters?

  -> Run for `nb_view=1`, just like finding Next Best View

Meshes


Faces
Vertices
Meshes are collections of vertices and faces