
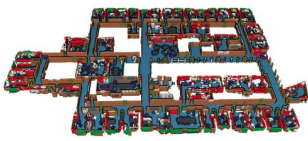
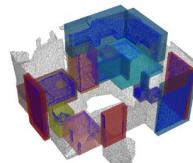
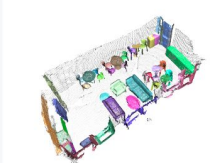


Densify Point Clouds Sprint 2 Technical

By Group A2_9: Chris, Risa, and Brian

Available Tasks

Tasks	Examples
Classification / Part Segmentation	
Segmentation	
Object Detection	
Panoptic Segmentation	
Registration	

- *Classification*: ModelNet from Zhirong Wu 3D ShapeNets: A Deep Representation for Volumetric Shapes

- *Segmentation/Detection*: ScanNet: Richly-annotated 3D Reconstructions of Indoor Scenes

- *Registration*: The IRALab Benchmark from Simone Fontana: Benchmark for point Cloud Registration Algorithms

Different dataset visualization

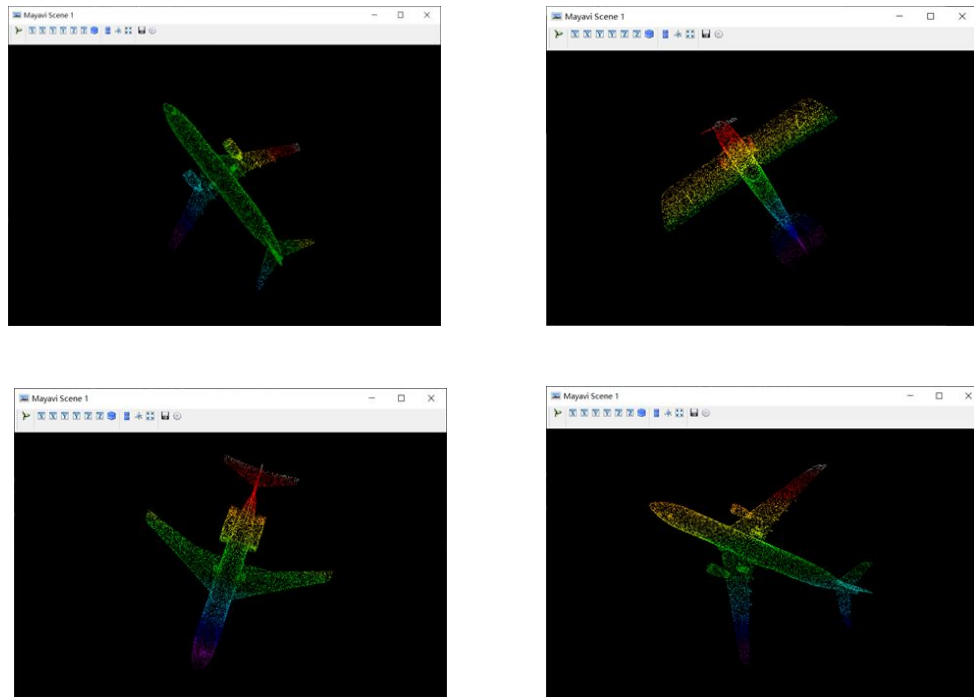


Fig1. point cloud images

Demo for Point Cloud 3D Model

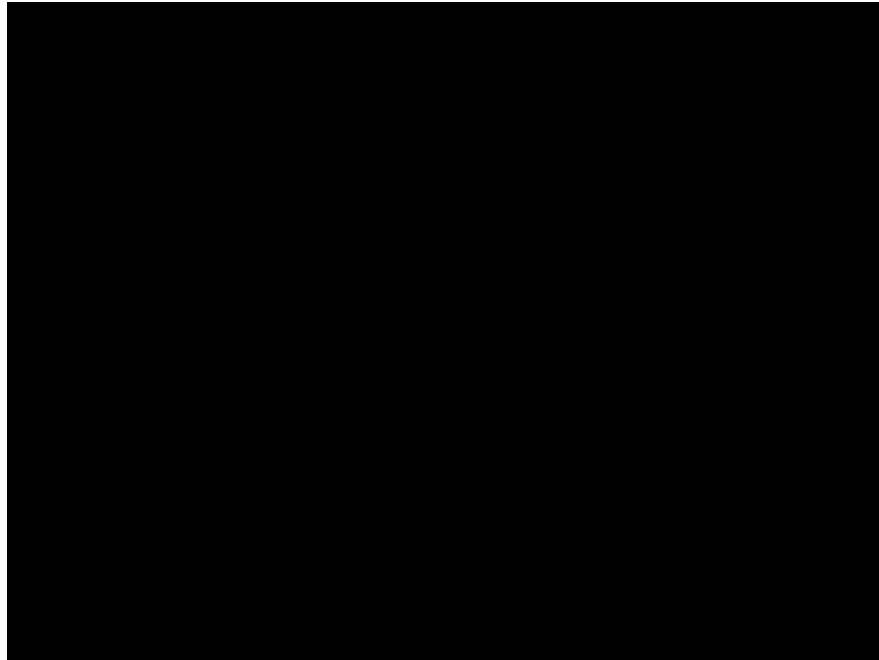
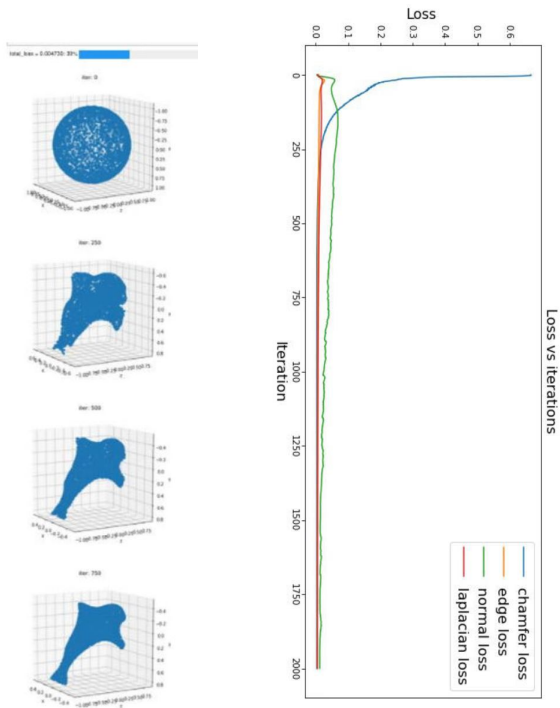


Fig2. Point Cloud Illustration

Dense deformation and Training Process



- Using Pytorch 3d modules to train and deform a ball into a dolphin
- GPU version == cuda 11.1
- Torch version == 1.8.1

- **Chamfer Loss:** compare the two sets of Point Clouds by computing the chamfer loss

Edge Loss: the edge length of the predicted mesh

Normal Loss: mesh normal consistency

Laplacian Loss: mesh laplacian smoothing

Fig3. Dense deform

ICCV-2019 Mesh R-CNN

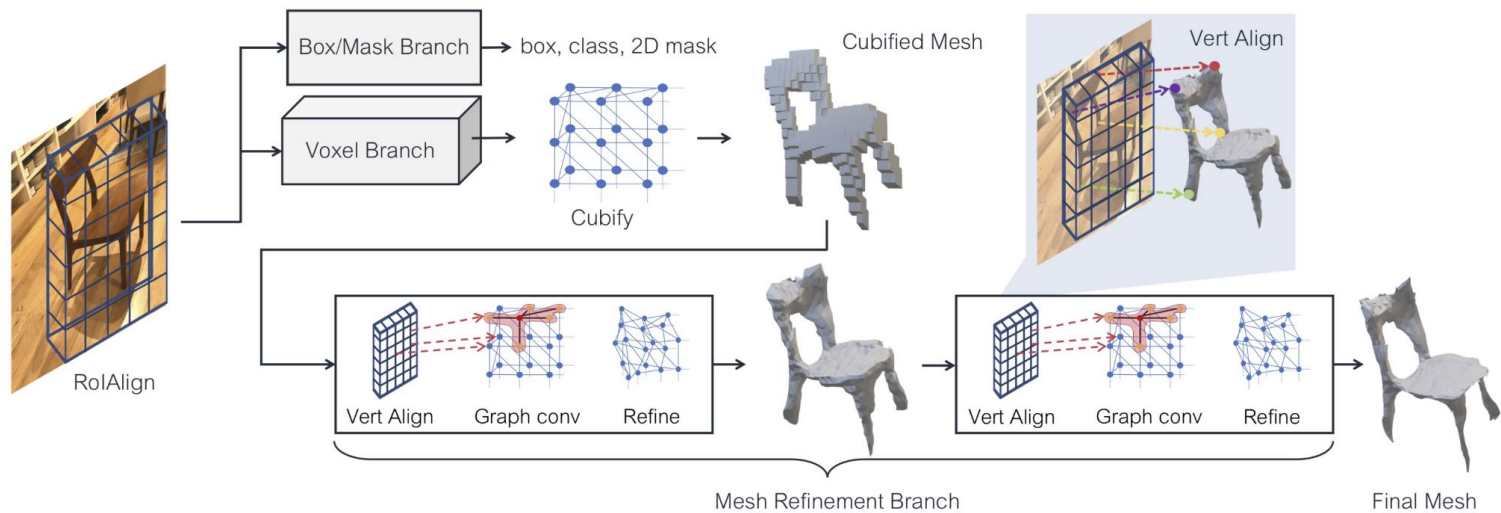


Fig4. Principle for mesh R-CNN

ICCV-2019 Mesh R-CNN

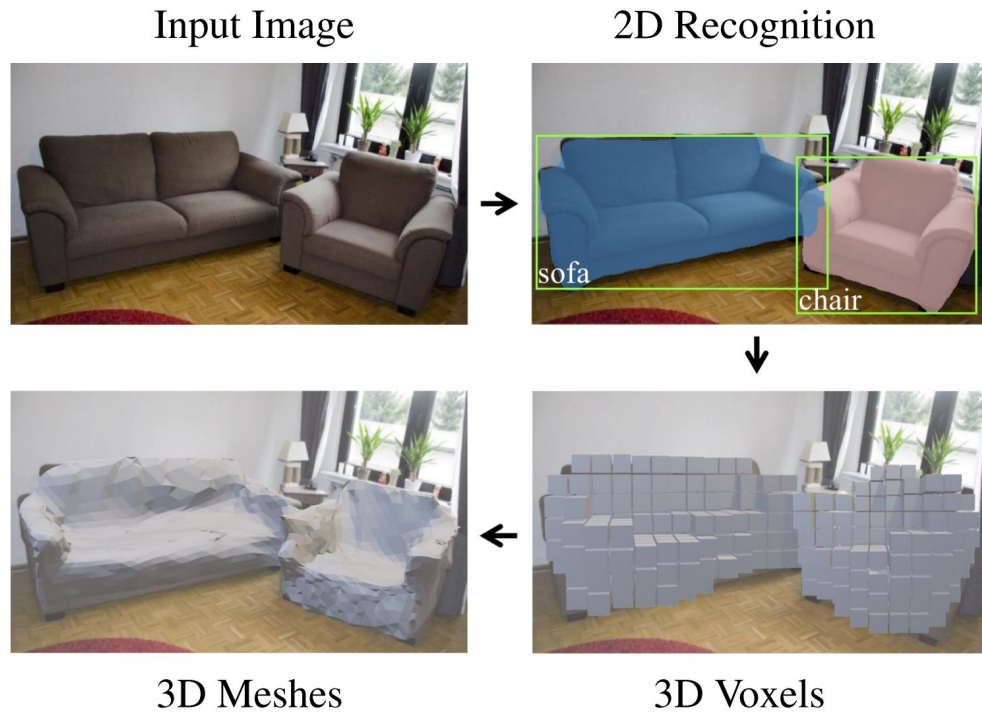
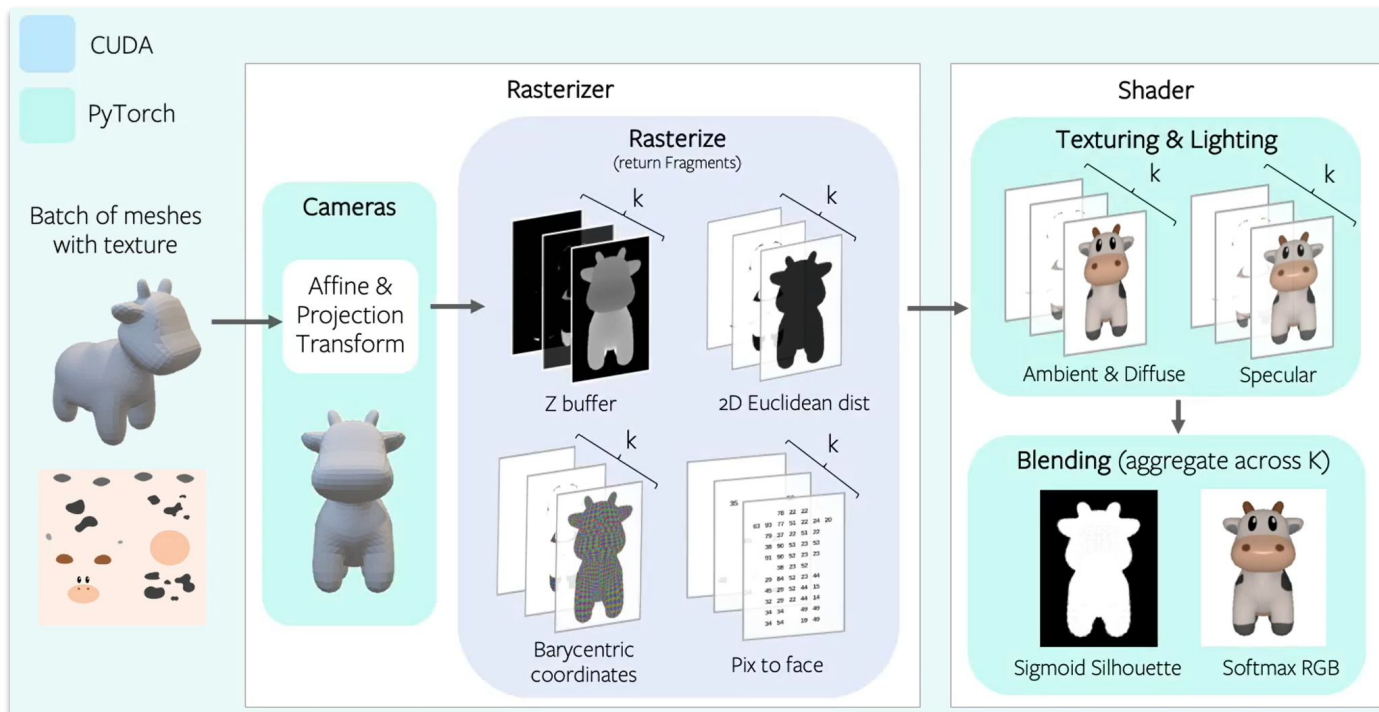


Fig.5 Visualization

Principle of Rendering Texture

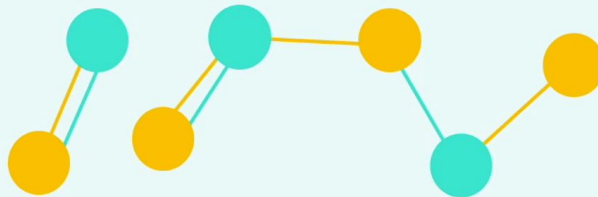
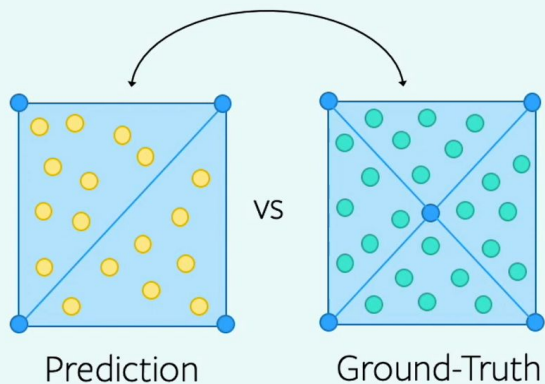


Loss Function

CHAMFER LOSS

Chamfer distance is the sum of L2 distance to each point's nearest neighbor in the other set

$$d_{CD}(S_1, S_2) = \sum_{x \in S_1} \min_{y \in S_2} \|x - y\|_2^2 + \sum_{y \in S_2} \min_{x \in S_1} \|x - y\|_2^2$$



Code for the Mesh Algorithm

```
from pytorch3d.utils import ico_sphere
from pytorch3d.ops import sample_points_from_meshes
from pytorch3d.loss import chamfer_distance

sphere_mesh1 = ico_sphere(level=3)
sphere_mesh2 = ico_sphere(level=1)

sample_sphere = sample_points_from_meshes(sphere_mesh1, 5000)
sample_test = sample_points_from_meshes(sphere_mesh2, 5000)

loss_chamfer, _ = chamfer_distance(sample_sphere, sample_test)
```

```
from pytorch3d.renderer import (
    OpenGLPerspectiveCameras, look_at_view_transform,
    RasterizationSettings, BlendParams,
    MeshRenderer, MeshRasterizer, HardPhongShader
)

R, T = look_at_view_transform(2.7, 10, 20)
cameras = OpenGLPerspectiveCameras(device=device, R=R, T=T)

raster_settings = RasterizationSettings(
    image_size=512,
    blur_radius=0.0,
    faces_per_pixel=1, # sets the value of K
)
```

```
renderer = MeshRenderer(
    rasterizer=MeshRasterizer(cameras=cameras, raster_settings=raster_settings),
    shader=HardPhongShader(device=device, cameras=cameras)
)

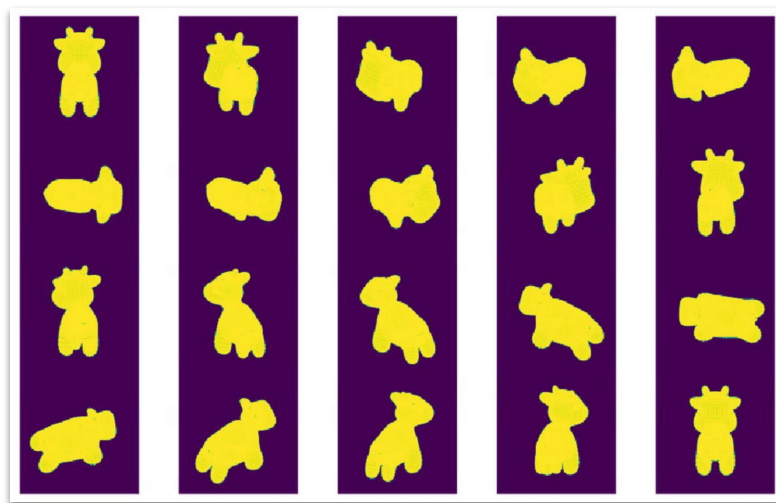
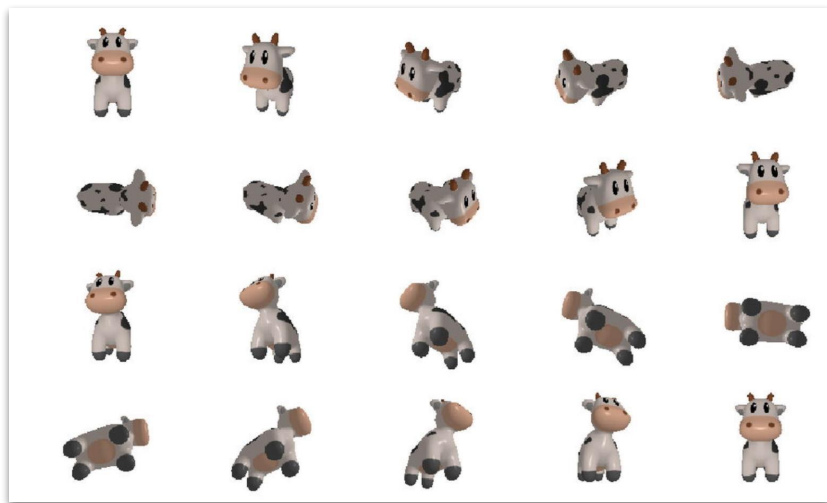
image = renderer(mesh)

# compute a loss given a ground truth image

loss = (gt_image - image).sum()

loss.backward()
```

Rendering Texture



PyTorch3D

Rendering Texture

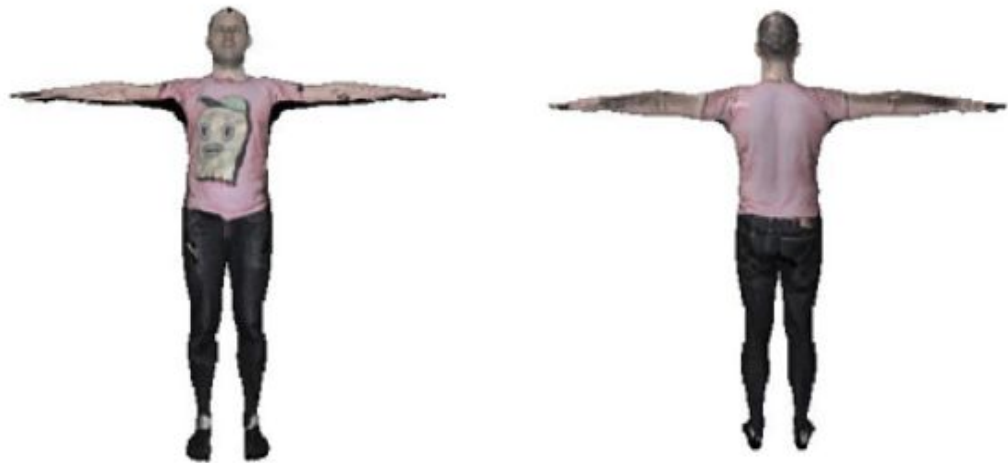


Fig3. Rendering 3D models based on
2D features