Notebook_printed

February 5, 2025

1 16825 - Learning for 3D Vision

1.1 Homework 1 - vinayakp

[78]:
```
# Imports
import sys
sys.path.append('C:/Users/VinayakKP/Documents/Spring 25/Learning_For_3D_Vision/
 ↪Homework/Homework_1/assignment1/starter')
import torch
import pytorch3d
from pytorch3d.structures import Meshes
import pytorch3d.renderer as rdr
from pytorch3d.io import load_objs_as_meshes
import numpy as np
import matplotlib.pyplot as plt
import imageio
from tqdm.notebook import tqdm
import camera_transforms
import dolly_zoom
import render_generic
import render_mesh
import utils
from IPython.display import Image
from PIL import Image, ImageDraw

device = torch.device('cuda:0')
```

1.2 Q1 Practicing with Cameras:

1.2.1 1.1. 360-degree Renders (5 points):

[79]:
```
def get_mesh_renderer(image_size=512):
    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

    R, T = rdr.look_at_view_transform(2.7, 0, 0)
    cameras = rdr.FoVPerspectiveCameras(device=device, R=R, T=T)

    raster_set = rdr.RasterizationSettings(
```

```
        image_size=image_size,
        blur_radius=0.0,
        faces_per_pixel=1
    )

    lights = rdr.PointLights(device=device, location=[[0.0, 0.0, -3.0]])

    render = rdr.MeshRenderer(
        rasterizer=rdr.MeshRasterizer(
            cameras=cameras,
            raster_settings=raster_set
        ),
        shader=rdr.HardPhongShader(
            device=device,
            cameras=cameras,
            lights=lights
        )
    )
    return render

def render_360_degree_mesh(mesh, device, image_size=512, num_views=72,␣
 ↪distance=2.75, elevation=30):
    renderer = utils.get_mesh_renderer(image_size=image_size)
    angles = torch.linspace(-180, 180, num_views)
    lights = rdr.PointLights(location=[[0, 0, -3]], device=device)
    images = []

    for angle in tqdm(angles):
        R, T = rdr.look_at_view_transform(dist=distance, elev=elevation,␣
 ↪azim=angle)
        cameras = rdr.FoVPerspectiveCameras(R=R, T=T, device=device)

        render = renderer(mesh, cameras=cameras, lights=lights)
        image = render[0, ..., :3].cpu().numpy()
        image = (image * 255).astype(np.uint8)
        images.append(image)

    return images

def save_gif(images, output_path, fps=24):
    duration = 1000 // fps
    imageio.mimsave(
        output_path,
        images,
        duration=duration,
        loop=0
    )
```

```
obj_filename = "../data/cow.obj"
mesh = load_objs_as_meshes([obj_filename], device=device)

render_img = render_360_degree_mesh(
    mesh,
    device=device,
    image_size=512,
    num_views=120,
    distance=2.7,
    elevation=30
)

save_gif(render_img, 'mesh_360_hardphongshader.gif', fps=30)
```

```
  0%|          | 0/120 [00:00<?, ?it/s]
```

1.2.2 1.2. Re-creating the Dolly Zoom (10 points):

[80]:
```
dolly_zoom.dolly_zoom(
        image_size=512,
        num_frames=30,
        duration=3,
        output_file="dolly.gif",
    )
```

```
  0%|          | 0/30 [00:00<?, ?it/s]
```

1.3 Q2. Practicing with Meshes:

1.3.1 2.1. Constructing a Tetrahedron (5 points):

[81]:
```
def make_reg_tetrahedron(device, edge_length):
    vertices = (edge_length/5)*torch.tensor([
        [2.0412, 2.0412, 2.0412],
        [-2.0412, -2.0412, 2.0412],
        [-2.0412, 2.0412, -2.0412],
        [2.0412, -2.0412, -2.0412]
     ], dtype=torch.float32, device=device)

    faces = torch.tensor([
        [0, 1, 2],
        [0, 1, 3],
        [0, 2, 3],
        [1, 2, 3]
     ], dtype=torch.int64, device=device)

    color = torch.tensor([0.0, 0.0, 1.0], device=device)
```

```
    vertex_colors = torch.ones_like(vertices)[None] * color
    textures = rdr.TexturesVertex(verts_features=vertex_colors)

    mesh = Meshes(
      verts=[vertices],
      faces=[faces],
      textures=textures
     )

    return mesh

tetra_mesh = make_reg_tetrahedron("cuda:0", 5)

rendered_images = render_360_degree_mesh(
    tetra_mesh,
    device=device,
    image_size=512,
    num_views=120,
    distance=8,
    elevation=30
)

save_gif(rendered_images, 'tetrahedron_360.gif', fps=30)
```

```
  0%|          | 0/120 [00:00<?, ?it/s]
```

1.3.2 2.2. Constructing a Cube (5 points):

[82]:
```
def make_cube(device, edge_length):
    vertices = (edge_length)*torch.tensor([
        [-0.5, 0.5, -0.5],
        [0.5, 0.5, -0.5],
        [0.5, 0.5, 0.5],
        [-0.5, 0.5, 0.5],
        [0.5, -0.5, -0.5],
        [0.5, -0.5, 0.5],
        [-0.5, -0.5, 0.5],
        [-0.5, -0.5, -0.5]
     ], dtype=torch.float32, device=device)

    faces = torch.tensor([
        [0, 1, 3],
        [1, 2, 3],
        [1, 2, 5],
        [1, 4, 5],
        [4, 5, 7],
        [5, 6, 7],
```

```
        [3, 6, 7],
        [3, 0, 7],
        [2, 6, 3],
        [2, 5, 6],
        [0, 4, 7],
        [0, 1, 4]
    ], dtype=torch.int64, device=device)

    color = torch.tensor([0.0, 0.0, 1.0], device=device)
    vertex_colors = torch.ones_like(vertices)[None] * color
    textures = rdr.TexturesVertex(verts_features=vertex_colors)

    mesh = Meshes(
      verts=[vertices],
      faces=[faces],
      textures=textures
     )

    return mesh

cube_mesh = make_cube("cuda:0", 5)

rendered_images = render_360_degree_mesh(
    cube_mesh,
    device=device,
    image_size=512,
    num_views=120,
    distance=9,
    elevation=30
)

save_gif(rendered_images, 'cube_360.gif', fps=30)
```

```
  0%|          | 0/120 [00:00<?, ?it/s]
```

1.4 3. Retexturing a Mesh (10 points):

[83]:
```
def apply_color_gradient(mesh, color1, color2, device):
    verts = mesh.verts_packed()

    z_coords = verts[:, 2]

    z_min, z_max = z_coords.min(), z_coords.max()
    alpha = (z_coords - z_min) / (z_max - z_min)

    alpha = alpha.unsqueeze(-1)
```

```
    color1 = torch.tensor(color1, device=device)
    color2 = torch.tensor(color2, device=device)

    vertex_colors = alpha * color2 + (1 - alpha) * color1
    vertex_colors = vertex_colors.unsqueeze(0)

    textures = rdr.TexturesVertex(vertex_colors)

    color_mesh = Meshes(
        verts=mesh.verts_list(),
        faces=mesh.faces_list(),
        textures=textures
    )

    return color_mesh

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

obj_filename = "../data/cow.obj"
mesh = load_objs_as_meshes([obj_filename], device=device)

color1 = [1.0, 0.0, 0.0]
color2 = [0.0, 0.0, 1.0]

colored_mesh = apply_color_gradient(mesh, color1, color2, device)

rendered_images = render_360_degree_mesh(
    colored_mesh,
    device=device,
    image_size=512,
    num_views=120,
    distance=2.7,
    elevation=30
)

save_gif(rendered_images, 'color_gradient_cow_360.gif', fps=30)
```

```
  0%|          | 0/120 [00:00<?, ?it/s]
```

1.5 4. Camera Transformations (10 points):

[84]:
```
# I am copying this function and removing the get_device() part here instead of␣
 ↪calling it from camera_transforms.py as my device is cuda and the default␣
 ↪implementation is giving me some problems

def render_textured_cow(
    cow_path="../data/cow.obj",
```

```
    image_size=256,
    # default case 0
    # R_relative=[[1, 0, 0], [0, 1, 0], [0, 0, 1]],
    # T_relative=[0, 0, 0],
    # CW 90-degree roation about z_axis case 1
    # R_relative=[[0, 1, 0], [-1, 0, 0], [0, 0, 1]],
    # T_relative =[0, 0, 0],
    # CW 90-degree rotation about y_axis case 2
    # R_relative=[[0, 0, 1], [0, 1, 0], [-1, 0, 0]],
    # T_relative=[-3, 0, 3],
    # Zoom out (z direction) case 3
    # R_relative=[[1, 0, 0], [0, 1, 0], [0, 0, 1]],
    # T_relative=[0, 0, 3],
    # Translation in x-y plane case 4
    R_relative=[[1, 0, 0], [0, 1, 0], [0, 0, 1]],
    T_relative=[0.5, -0.5, 0],
    device=device,
):
    meshes = pytorch3d.io.load_objs_as_meshes([cow_path]).to(device)
    R_relative = torch.tensor(R_relative).float()
    T_relative = torch.tensor(T_relative).float()
    R = R_relative @ torch.tensor([[1.0, 0, 0], [0, 1, 0], [0, 0, 1]])
    T = R_relative @ torch.tensor([0.0, 0, 3]) + T_relative
    renderer = get_mesh_renderer(image_size=256)
    cameras = pytorch3d.renderer.FoVPerspectiveCameras(
        R=R.unsqueeze(0), T=T.unsqueeze(0), device=device,
    )
    lights = pytorch3d.renderer.PointLights(location=[[0, 0.0, -3.0]],␣
 ↪device=device,)
    rend = renderer(meshes, cameras=cameras, lights=lights)
    return rend[0, ..., :3].cpu().numpy()

R_relative1 = torch.tensor([[0, 1, 0], [-1, 0, 0], [0, 0, 1]])
T_relative1 = torch.tensor([0, 0, 0])
Img1 = render_textured_cow(R_relative=R_relative1, T_relative=T_relative1)
plt.imsave("textured_cow1.jpg", Img1)

R_relative2 = torch.tensor([[0, 0, 1], [0, 1, 0], [-1, 0, 0]])
T_relative2 = torch.tensor([-3, 0, 3])
Img2 = render_textured_cow(R_relative=R_relative2, T_relative=T_relative2)
plt.imsave("textured_cow2.jpg", Img2)

R_relative3 = torch.tensor([[1, 0, 0], [0, 1, 0], [0, 0, 1]])
T_relative3 = torch.tensor([0, 0, 3])
Img3 = render_textured_cow(R_relative=R_relative3, T_relative=T_relative3)
plt.imsave("textured_cow3.jpg", Img3)
```

```
R_relative4 = torch.tensor([[1, 0, 0], [0, 1, 0], [0, 0, 1]])
T_relative4 = torch.tensor([0.5, -0.5, 0])
Img4 = render_textured_cow(R_relative=R_relative4, T_relative=T_relative4)
plt.imsave("textured_cow4.jpg", Img4)
```

```
C:\Users\VinayakKP\AppData\Local\Temp\ipykernel_28356\3654917673.py:24:
UserWarning: To copy construct from a tensor, it is recommended to use
sourceTensor.clone().detach() or
sourceTensor.clone().detach().requires_grad_(True), rather than
torch.tensor(sourceTensor).
  R_relative = torch.tensor(R_relative).float()
C:\Users\VinayakKP\AppData\Local\Temp\ipykernel_28356\3654917673.py:25:
UserWarning: To copy construct from a tensor, it is recommended to use
sourceTensor.clone().detach() or
sourceTensor.clone().detach().requires_grad_(True), rather than
torch.tensor(sourceTensor).
  T_relative = torch.tensor(T_relative).float()
```

1.6 5. Rendering Generic 3D Representations:

1.6.1 5.1. Rendering Point Clouds from RGB-D Images (10 points):

[85]:
```
data = render_generic.load_rgbd_data(path="../data/rgbd_data.pkl")

points1, colors1 = utils.unproject_depth_image(torch.tensor(data["rgb1"]),
                                               torch.tensor(data["mask1"]),
                                               torch.tensor(data["depth1"]),
                                               data["cameras1"])

pc1 = pytorch3d.structures.Pointclouds(points=points1.unsqueeze(0),␣
 ↪features=colors1.unsqueeze(0)).to(device)
num_views=120
angles = torch.linspace(-180, 180, num_views)
image_size = 256

R, T = rdr.look_at_view_transform(dist = 10, elev = 0, azim = angles)

R = torch.tensor([[-1, 0, 0], [0, -1, 0], [0, 0, 1]]).float()@R

cameras = rdr.FoVPerspectiveCameras(R=R, T=T, device=device)
lights = rdr.PointLights(location=[[0, 0, -3]], device=device)
renderer = utils.get_points_renderer(image_size=image_size, device=device)

images = renderer(pc1.extend(num_views), cameras = cameras, lights = lights)
images = images.cpu().numpy()[..., :3]
images = (images * 255).clip(0, 255).astype(np.uint8)
imageio.mimsave("pc1.gif", images, fps=30, loop = 0)
```

```
points2, colors2 = utils.unproject_depth_image(torch.tensor(data["rgb2"]),
                                               torch.tensor(data["mask2"]),
                                               torch.tensor(data["depth2"]),
                                               data["cameras2"])

pc2 = pytorch3d.structures.Pointclouds(points=points2.unsqueeze(0),␣
 ↪features=colors2.unsqueeze(0)).to(device)

images = renderer(pc2.extend(num_views), cameras = cameras, lights = lights)
images = images.cpu().numpy()[..., :3]
images = (images * 255).clip(0, 255).astype(np.uint8)
imageio.mimsave("pc2.gif", images, fps=30, loop = 0)

pc3 = pytorch3d.structures.Pointclouds(points=torch.cat((points1,points2), 0).
 ↪unsqueeze(0), features=torch.cat((colors1,colors2), 0).unsqueeze(0),).
 ↪to(device)

images = renderer(pc3.extend(num_views), cameras= cameras, lights= lights)
images = images.cpu().numpy()[..., :3]
images = (images * 255).clip(0, 255).astype(np.uint8)
imageio.mimsave("pc3.gif", images, fps=30, loop = 0)
```

1.6.2 5.2. Parametric Functions (10 + 5 points):

Part 1: Torus

[86]:
```
def get_points_renderer(
    image_size=512, radius=0.01, background_color=(1, 1, 1), device=None
):
    if device is None:
        if torch.cuda.is_available():
            device = torch.device("cuda:0")
        else:
            device = torch.device("cpu")
    raster_settings = rdr.PointsRasterizationSettings(image_size=image_size,␣
 ↪radius=radius,)
    renderer = rdr.PointsRenderer(
        rasterizer= rdr.PointsRasterizer(raster_settings=raster_settings),
        compositor= rdr.AlphaCompositor(background_color=background_color),
    )
    return renderer

def get_torus_points(num_samples, c=3, a=2):
    u = torch.linspace(0, 2 * torch.pi, num_samples)
    v = torch.linspace(0, 2 * torch.pi, num_samples)
    u, v = torch.meshgrid(u, v)
```

```
    x = (c + a * torch.cos(v)) * torch.cos(u)
    y = (c + a * torch.cos(v)) * torch.sin(u)
    z = a * torch.sin(v)

    points = torch.stack((x.flatten(), y.flatten(), z.flatten()), dim=1).
 ↪unsqueeze(0)
    return points

def parametric_torus_colors(num_samples):
    u = torch.linspace(0, 2 * torch.pi, num_samples)
    v = torch.linspace(0, 2 * torch.pi, num_samples)
    u, v = torch.meshgrid(u, v)

    colors_r = torch.cos(u).flatten()
    colors_b = torch.sin(v).flatten()
    colors_g = torch.zeros_like(colors_r)

    colors = torch.stack([colors_r, colors_g, colors_b], dim=1)
    colors = (colors + 1) / 2
    colors = colors.unsqueeze(0)
    return colors

num_samples = 200
torus_pts = get_torus_points(num_samples = num_samples).to(device)
torus_color = parametric_torus_colors(num_samples = num_samples).to(device)

pc_torus = pytorch3d.structures.Pointclouds(points = torus_pts, features =␣
 ↪torus_color).to(device)
R, T = rdr.look_at_view_transform(dist=10, elev=0, azim = angles)
renderer = get_points_renderer(image_size=image_size)
images = renderer(pc_torus.extend(num_views), cameras=cameras)
images = images.cpu().numpy()[..., :3]
images = (images * 255).clip(0, 255).astype(np.uint8)
imageio.mimsave("torus360.gif", images, fps=30, loop=0)
```

Part 2: Square Torus

[87]:
```
def get_square_torus_points(num_samples, R=2, a=1, n=8):
    u = torch.linspace(0, 2 * torch.pi, num_samples)
    v = torch.linspace(0, 2 * torch.pi, num_samples)
    u, v = torch.meshgrid(u, v)

    r = (torch.abs(torch.cos(v))**n + torch.abs(torch.sin(v))**n)**(-1/n)

    x = (R + a * r * torch.cos(v)) * torch.cos(u)
    y = (R + a * r * torch.sin(v)) * torch.sin(u)
    z = a * r * torch.sin(v)
```

```
    points = torch.stack([x.flatten(), y.flatten(), z.flatten()], dim=1).
 ↪unsqueeze(0)
    return points

def color_square_torus(num_samples):
    u = torch.linspace(0, 2 *torch.pi, num_samples)
    v = torch.linspace(0, 2 *torch.pi, num_samples)
    u, v = torch.meshgrid(u, v)

    colors_r = torch.abs(torch.cos(v)).flatten()
    colors_g = torch.abs(torch.sin(u)).flatten()
    colors_b = torch.abs(torch.sin(v + u)).flatten()

    colors =torch.stack([colors_r, colors_g, colors_b], dim=1).unsqueeze(0)
    return colors

num_samples = 200
sq_torus_pts = get_square_torus_points(num_samples = num_samples).to(device)
sq_torus_colors = color_square_torus(num_samples = num_samples).to(device)

pc_sq_torus = pytorch3d.structures.Pointclouds(points = sq_torus_pts, features␣
 ↪= sq_torus_colors).to(device)
R, T = rdr.look_at_view_transform(dist=10, elev=0, azim = angles)
renderer = get_points_renderer(image_size=image_size)
images = renderer(pc_sq_torus.extend(num_views), cameras=cameras)
images = images.cpu().numpy()[..., :3]
images = (images * 255).clip(0, 255).astype(np.uint8)
imageio.mimsave("squaretorus360.gif", images, fps=30, loop=0)
```

1.6.3 5.3. Implicit Surfaces (15 + 5 points):

Part 1: Torus Mesh:

[88]:
```
import mcubes

def torus_implicit_fxn(grid_size=128, R=1.0, r=0.4):
    x = torch.linspace(-2, 2, grid_size)
    y = torch.linspace(-2, 2, grid_size)
    z = torch.linspace(-2, 2, grid_size)

    X, Y, Z = torch.meshgrid(x, y, z, indexing='ij')
    squared_term = (R - torch.sqrt(X**2 + Y**2))**2
    F = squared_term + Z**2 - r**2
    volume = F.numpy()
    vertices, faces = mcubes.marching_cubes(volume, 0)
    vertices = vertices / grid_size * 4 - 2
```

```
    vertices = torch.from_numpy(vertices).float()
    faces = torch.from_numpy(faces).long()
    return vertices, faces

grid_size = 128
vertices, faces = torus_implicit_fxn(grid_size=grid_size)
textures = rdr.TexturesVertex(vertices.unsqueeze(0))
mesh = pytorch3d.structures.Meshes([vertices], [faces], textures=textures).
 ↪to(device)

renderer = get_mesh_renderer(image_size=image_size)
lights = rdr.PointLights(location=[[0, 0.0, -4.0]], device=device)
R, T = rdr.look_at_view_transform(dist=5, elev=0, azim = angles)
cameras = rdr.FoVPerspectiveCameras(R=R, T=T, device=device)

images = renderer(mesh.extend(num_views), cameras=cameras, lights = lights)
images = images.cpu().numpy()[..., :3]
images = (images * 255).clip(0, 255).astype(np.uint8)
imageio.mimsave("torus_fxn.gif", images, fps=30, loop=0)
```

Part 2: Double-Bubble Surface:

[89]:
```
def double_bubble(grid_size=128):
    x = torch.linspace(-2, 2, grid_size)
    y = torch.linspace(-2, 2, grid_size)
    z = torch.linspace(-2, 2, grid_size)

    X, Y, Z = torch.meshgrid(x, y, z, indexing='ij')

    sphere1 = (X + 0.5)**2 + Y**2 + Z**2 - 1
    sphere2 = (X - 0.5)**2 + Y**2 + Z**2 - 1

    F = torch.minimum(sphere1, sphere2)

    volume = F.numpy()
    vertices, faces = mcubes.marching_cubes(volume, 0)
    vertices = vertices / grid_size * 4 - 2
    vertices = torch.from_numpy(vertices).float()
    faces = torch.from_numpy(faces).long()
    return vertices, faces

grid_size = 128
vertices, faces = double_bubble(grid_size=grid_size)
textures = rdr.TexturesVertex(vertices.unsqueeze(0))
mesh = Meshes([vertices], [faces], textures=textures).to(device)

renderer = get_mesh_renderer(image_size=image_size)
```

```
lights = rdr.PointLights(location=[[0, 0.0, -4.0]], device=device)
R, T = rdr.look_at_view_transform(dist=5, elev=0, azim = angles)
cameras = rdr.FoVPerspectiveCameras(R=R, T=T, device=device)


images = renderer(mesh.extend(num_views), cameras=cameras, lights = lights)
images = images.cpu().numpy()[..., :3]
images = (images * 255).clip(0, 255).astype(np.uint8)
imageio.mimsave("double_bubble_fxn.gif", images, fps=30, loop=0)
```

1.7 Q6. Do Something Fun (10 points):

1.7.1 Morphing between sphere and torus:

[90]:
```
def morphing_shapes(t, grid_size=128):
    x = torch.linspace(-2, 2, grid_size)
    y = torch.linspace(-2, 2, grid_size)
    z = torch.linspace(-2, 2, grid_size)

    X, Y, Z = torch.meshgrid(x, y, z, indexing='ij')

    sphere = X**2 + Y**2 + Z**2 - 1

    R = 1.0
    r = 0.4
    torus = (R - torch.sqrt(X**2 + Y**2))**2 + Z**2 - r**2

    F = (1-t) * sphere + t * torus

    volume = F.numpy()
    vertices, faces = mcubes.marching_cubes(volume, 0)
    vertices = vertices / grid_size * 4 - 2

    return torch.from_numpy(vertices).float(), torch.from_numpy(faces).long()

grid_size = 128
vertices, faces = morphing_shapes(grid_size=grid_size, t = 0.5)
textures = rdr.TexturesVertex(verts_features=torch.ones_like(vertices).
 ↪unsqueeze(0))
mesh = Meshes([vertices], [faces], textures=textures).to(device)

renderer = get_mesh_renderer(image_size=image_size)
lights = rdr.PointLights(location=[[0, 0.0, -4.0]], device=device)
R, T = rdr.look_at_view_transform(dist=5, elev=0, azim = angles)
cameras = rdr.FoVPerspectiveCameras(R=R, T=T, device=device)

images = renderer(mesh.extend(num_views), cameras=cameras, lights = lights)
```

```
images = images.cpu().numpy()[..., :3]
images = (images * 255).clip(0, 255).astype(np.uint8)
imageio.mimsave("morphing_shapes.gif", images, fps=15, loop=0)
```