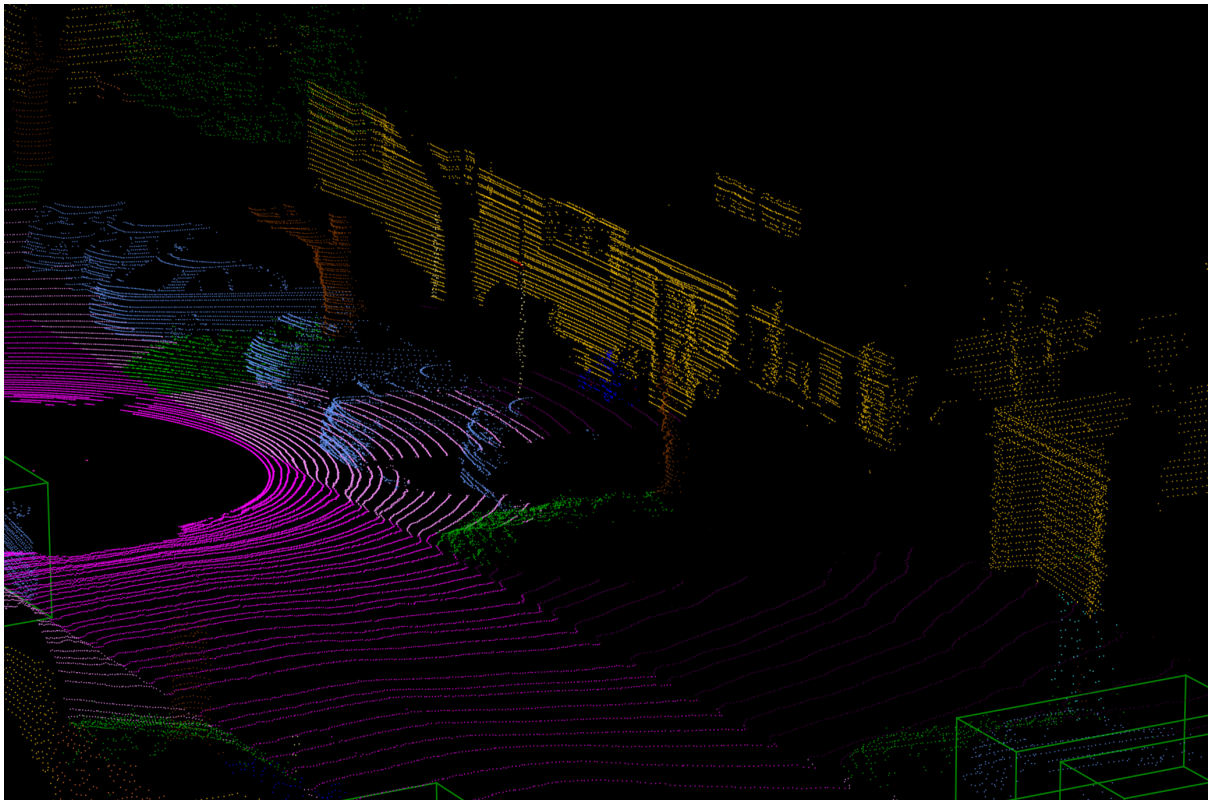
CVAIAC 2024

Project 2

Task 1

Group 05

Carl Brander
Tanguy Dieudonné
Han Xi



December 20, 2024

Problem 1: Multi-Modal Data Visualization

Task 1.1: 3D Point cloud in 2D Image

Result Images of Task 1.1: Both the Demo and the Data image. In this task, the Lidar point cloud was projected into the RGB Image FoV and colored based on the ground-truth semantic labels.

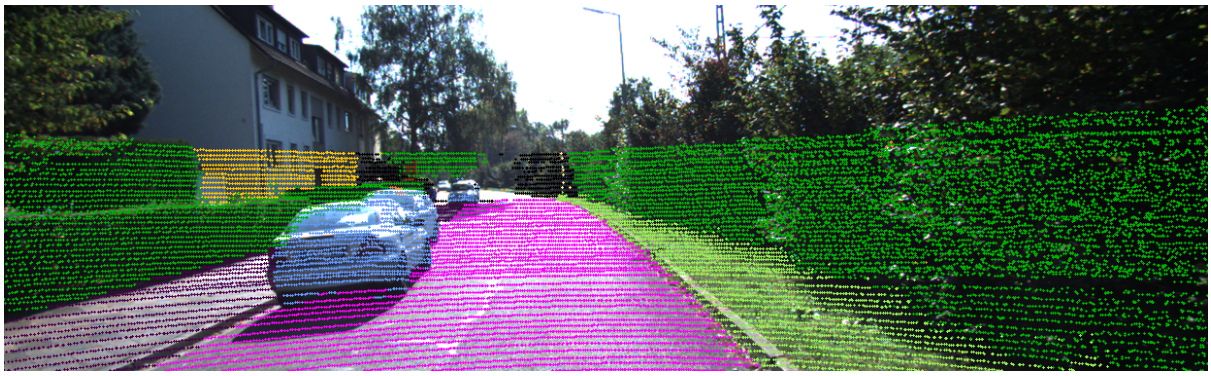


Figure 1: Demo Image, Lidar point cloud overlayed on FoV of RGB camera with points color-coded according to their ground-truth classes

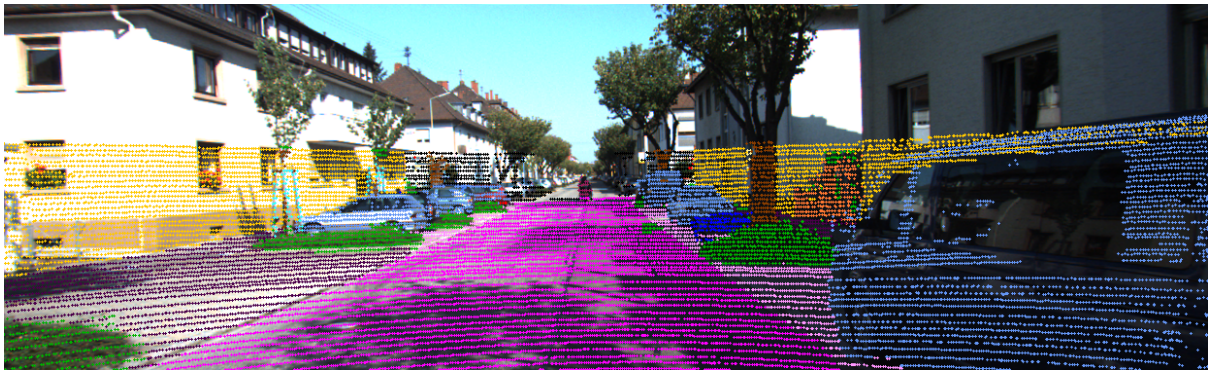


Figure 2: Data Image, Lidar point cloud overlayed on FoV of RGB camera with points color-coded according to their ground-truth classes

Task 1.2: 3D Bounding Boxes in 2D Image

Result Images of Task 1.2: Both the Demo and the Data image. In this task, the ground-truth 3D bounding boxes of the cars in the scene were added to the 2D RGB Image. This can be seen in Figure 3 and 4.

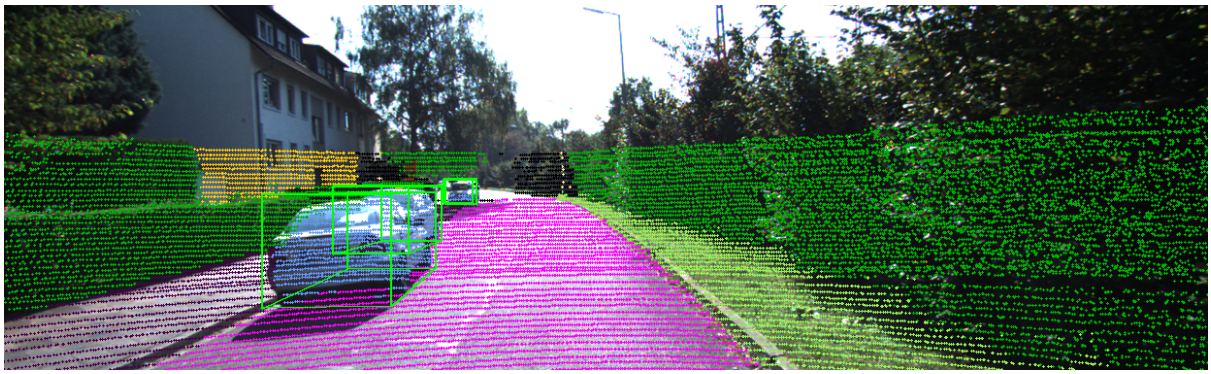


Figure 3: Demo Image with semantically labelled pointcloud and the 3D ground-truth bounding boxes of the cars overlayed.

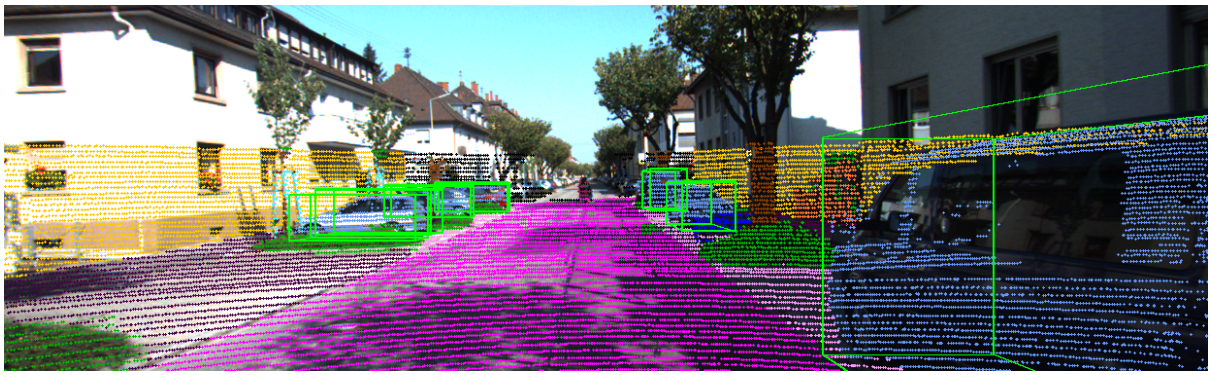


Figure 4: Data Image with semantically labelled pointcloud and the 3D ground-truth bounding boxes of the cars overlayed.

Task 1.3: 3D Visualization

First, both Data and Demo 3D scenes with all points of the lidar colored by their label and 3D bounding boxes for all detected vehicles are shown.

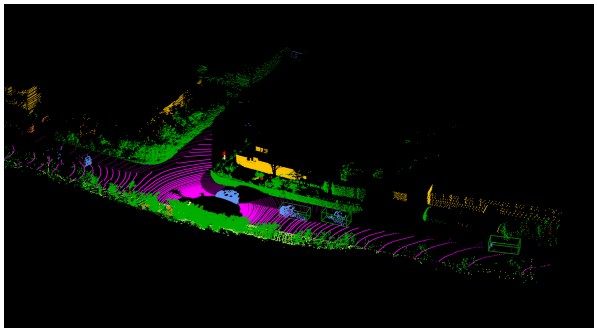


Figure 5: Demo Scene: All points Pointcloud and 3D bounding boxes for cars

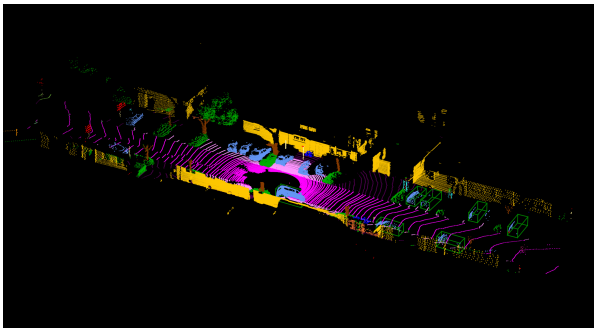


Figure 6: Data Scene: All points Pointcloud and 3D bounding boxes for cars

Next, the 3D Pointcloud is visualized with only points that are in the FoV (Field of View) of the RGB Camera (cam2).

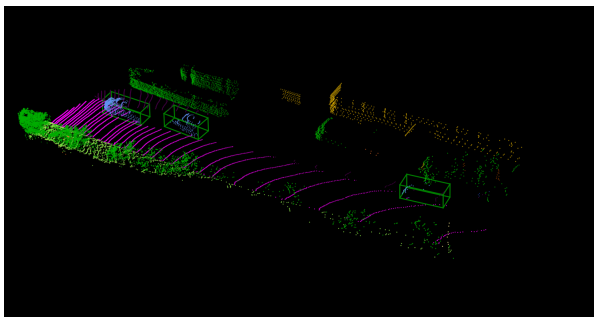


Figure 7: Demo Scene: Camera FoV of points in the Pointcloud including 3D bounding boxes for cars

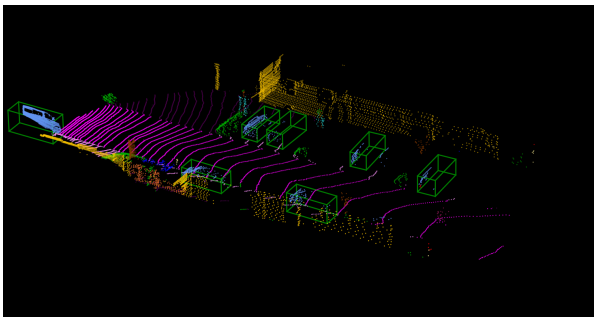


Figure 8: Data Scene: Camera FoV of points in the Pointcloud including 3D bounding boxes for cars

When filtering out all Lidar points without the class "car" (class nr. 10) we can count the number of cars in the ground truth a lot easier:
Here for only the FoV: In the demo image, we can see that all 3 out of the 3 cars are detected with 3D bounding boxes. In the data image, within the camera FoV we can make out 7 detections and 2 possible cars that have not been detected.

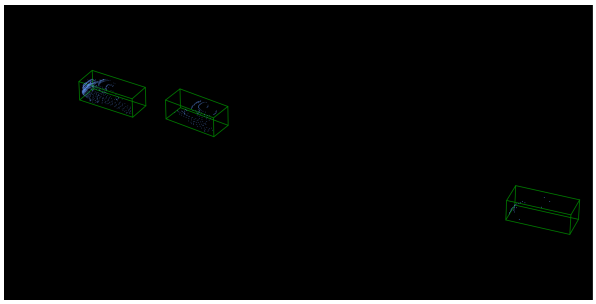


Figure 9: Demo Image: Points in the Camera FoV with the class car in Point cloud and their 3D bounding boxes

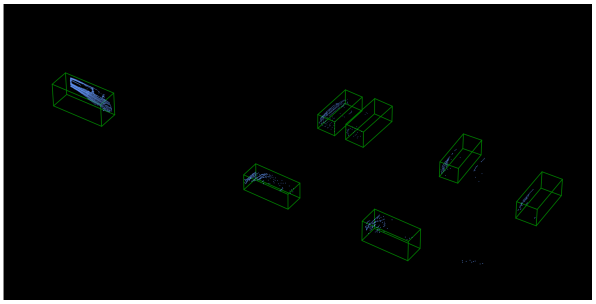


Figure 10: Data Image: Points in the Camera FoV with the class car in Point cloud and their 3D bounding boxes

Here are the results for all available lidar points: For the demo scene, one can make out a lot more cars on the side and behind the Lidar equipt car which cannot be seen from the camera. A total of around

7 cars can be made out of which only 3 in the front were detected. For the data scene, one can make out a total of 17 cars, most of them parked on the side of the road. And only 7 of them (in front of the Lidar car) have been correctly detected.

Figure 11: Demo scene: visualizing all Points with class car in Point cloud and the 3D bounding box of the detected ones within the FoV

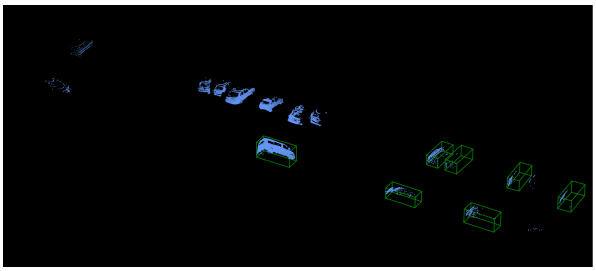


Figure 12: Data scene: visualizing all Points with class car in Point cloud and the 3D bounding box of the detected ones within the FoV

In table 1 one can see a summary of the demo scene car detections within the FoV and the complete point cloud. This results in a detection rate of 100% within the FoV and ≈ 43% of all cars.

	Within Camera FoV	Complete Point Cloud
Detected Cars	3 (100%)	3 (43%)
Non-Detected Cars	0 (0%)	4 (57%)
Total Cars	3	7

Table 1: Summary of Demo scene car detections

In table 2 one can see a summary of the data scene car detections within the FoV and the complete point cloud. This results in a detection rate of ≈ 78% of cars in the FoV and ≈ 41% of all cars.

	Within Camera FoV	Complete Point Cloud
Detected Cars	7 (78%)	7 (41%)
Non-Detected Cars	2 (22%)	10 (59%)
Total Cars	9	17

Table 2: Summary of Data scene car detections

Appendix 1: Code

Code used for Task 1.1

```
import os
from load_data import load_data
from PIL import Image
from PIL import ImageDraw
import numpy as np

DATA_FILENAME = 'data.p'  # TODO: Change to data.p for your final submission


def task_1(data):
    """
    Task 1: Given the intrinsic and extrinsic projection matrices, project the point
    ↪  cloud onto the image of Cam 2. Color each point projected according to their
    ↪  respective semantic label. The color map for each label is provided with the
    ↪  data.
    """

    #the rgb image from camera 2
    rgb_image_cam_2 = data['image_2']
    #intrinsic and extrinsic matrices for cam 2
    intrinsic_cam_2 = data['P_rect_20']
    extrinsic_from_velo_to_cam_2 = data['T_cam2_velo']
    #points (num_points,4)
    points = data['velodyne']
    #semantic labels of points (num_points)
    labels = data['sem_label']
    #color map for each label
    color_map = data['color_map']

    #remove all 3d points with a negative x-axis value (behind the camera) as well as
    ↪  their labels in the label array
    labels = labels[points[:,0] >= 0]
    points = points[points[:,0] >= 0]


    #transform points from velodyne to cam 2
    points = points @ extrinsic_from_velo_to_cam_2.T
    #project points to cam 2
    points = points @ intrinsic_cam_2.T

    #normalize points to get 2D coordinates in image [x,y]
    points[:,0] /= points[:,2]
    points[:,1] /= points[:,2]
    points = points[:,:2]

    #remove all points that are outside the image, same for the labels in the label
    ↪  array
    labels = labels[(points[:,0] >= 0) & (points[:,0] < rgb_image_cam_2.shape[1]) &
    ↪  (points[:,1] >= 0) & (points[:,1] < rgb_image_cam_2.shape[0])]
    points = points[(points[:,0] >= 0) & (points[:,0] < rgb_image_cam_2.shape[1]) &
    ↪  (points[:,1] >= 0) & (points[:,1] < rgb_image_cam_2.shape[0])]

    #draw all points on the image in their correct semantic color
```

```
    for i in range(len(points)):
        x = int(points[i,0])
        y = int(points[i,1])
        label_of_point = labels[i][0]

        colors_of_point = color_map[label_of_point]
        #change color of points from BGR to rgb
        colors_of_point = colors_of_point[::-1]
        #draw small + at the location of the point (like in the example image)
        rgb_image_cam_2[y,x-1:x+2] = colors_of_point
        rgb_image_cam_2[y-1:y+2,x] = colors_of_point

    # Convert the array to uint8
    rgb_image_cam_2 = rgb_image_cam_2.astype(np.uint8)

    #convert to pillow image
    rgb_image_cam_2 = Image.fromarray(rgb_image_cam_2)


    #save the image with pillow
    rgb_image_cam_2.save(f'task1_{DATA_FILENAME}.png')
```

Code used for Task 1.2

```
import os
from load_data import load_data
from PIL import Image
from PIL import ImageDraw
import numpy as np

DATA_FILENAME = 'data.p'  # TODO: Change to data.p for your final submission



def task_2(data):
    """
    Task 2: In addition to the points with semantic labels, project the 3D bounding
    ↪  boxes of all given vehicles onto the Cam 2 image.
    """

    image = data['image_2']
    P_cam_2 = data['P_rect_20']

    # Create a draw object
    image = image.astype(np.uint8)
    image = Image.fromarray(image)
    draw = ImageDraw.Draw(image)

    for obj in data['objects']:

        #skip objects with negative x center value (behind the camera)
        if (obj[13] < 0):
```

```
            continue
        else:

            h = obj[8] #height
            l = obj[10] #width
            w = obj[9] #length

            x = obj[11] #center x
            y = obj[12] #center y
            z = obj[13] #center z

            ry = obj[14] #rotation around vertical axis

            # 3D bounding box corners in the object coordinate frame
            x_corners = [ l/2,  l/2, -l/2, -l/2,  l/2,  l/2, -l/2, -l/2 ]
            y_corners = [    0,    0,    0,    0,   -h,   -h,   -h,   -h ]
            z_corners = [ w/2, -w/2, -w/2,  w/2,  w/2, -w/2, -w/2,  w/2 ]

            corners = np.array([x_corners, y_corners, z_corners])

            #get rotation matrix according to obj[14] (vertical rotation value)

            R = np.array([[np.cos(ry), 0, np.sin(ry)],
                          [0, 1, 0],
                          [-np.sin(ry), 0, np.cos(ry)]])

            #rotate the corners
            corners_3d = R @ corners
            #add the center of the car to the rotated corners
            corners_3d += np.array([[x],[y],[z]])

            corners_3d_hom = np.vstack((corners_3d, np.ones((1,
            ↪  corners_3d.shape[1]))))

            # Project the 3D corners onto the image plane
            corners_2d_hom = P_cam_2 @ corners_3d_hom

            # Normalize to get pixel coordinates
            corners_2d = corners_2d_hom[:2, :] / corners_2d_hom[2, :]

            # Prepare the coordinates for drawing
            corners_2d = corners_2d.T  # Transpose to (8, 2)

            # Draw the bounding box edges
            # Define the connections between corners to form the 3D bounding box
            edges = [
                (0, 1), (1, 2), (2, 3), (3, 0),  # Bottom face
                (4, 5), (5, 6), (6, 7), (7, 4),  # Top face
                (0, 4), (1, 5), (2, 6), (3, 7)   # Vertical edges
            ]

            # Draw each edge
            for edge in edges:
                x1, y1 = corners_2d[edge[0]]
                x2, y2 = corners_2d[edge[1]]
                draw.line([(x1, y1), (x2, y2)], fill=(0, 255, 0), width=2)

    #save the image with pillow
```

```
    image.save(f'task1_2_{DATA_FILENAME}.png')
```

Code used for Task 1.3

After many tries, the visualization using vispy (especially over the headless server environment) was abandoned and the plot was generated with Matplotlib locally instead:

```
import numpy as np
import matplotlib.pyplot as plt
from load_data import load_data
from vis import project_point_cloud

def render_point_cloud(data, only_fov,only_car,output_filename='output.png'):
    # get correct colors for each point
    colors = np.array([data["color_map"][label][::-1] for label in
    ↪  data["sem_label"].squeeze()], dtype=np.float32)
    colors /= 255.

    # get points
    points = data["velodyne"]

    if only_car:
        # Filter out all points except those that belong to the "car" class
        keep_only_car = data["sem_label"].squeeze() == 10
        points = points[keep_only_car]
        colors = colors[keep_only_car]

    #used if only showing points that are in cam_2's FoV
    if only_fov:
        # add 1s for homogeneous coordinates
        points[:, 3] = 1

        # Project points to image plane
        projected_points = project_point_cloud(data, points)

        # Get image dimensions
        image = data["image_2"]
        image_shape = image.shape

        # Filter points within image boundaries
        keep = (projected_points[:, 0] >= 0) & (projected_points[:, 0] <
        ↪  image_shape[1]) & \
               (projected_points[:, 1] >= 0) & (projected_points[:, 1] <
               ↪  image_shape[0]) & \
               (points[:, 0] > 0)
        points = points[keep]
        colors = colors[keep]


    # Render 3D point cloud
    #fullscreen
    fig = plt.figure(figsize=(12, 8))
    ax = fig.add_subplot(111, projection='3d')
```

```
    # Plot point cloud
    ax.scatter(points[:, 0], points[:, 1], points[:, 2],
               c=colors,
               s=1)

    # Render bounding boxes
    objects = data["objects"]
    T_inv = np.linalg.inv(data["T_cam0_velo"])

    for obj in objects:
        dimensions = obj[8:11] #height, width, length
        location = obj[11:14] # x, y, z in cam0 coordinates
        rotation = obj[14] #rotation angle of box

        h, w, l = dimensions

        # Define the eight corners of the bounding box in the object's coordinate
        ↪  frame
        x_corners = [ l/2,  l/2, -l/2, -l/2,  l/2,  l/2, -l/2, -l/2]
        y_corners = [ 0,    0,    0,    0,   -h,   -h,   -h,   -h]
        z_corners = [ w/2, -w/2, -w/2,  w/2,  w/2, -w/2, -w/2,  w/2]

        # Combine into a numpy array
        corners = np.array([x_corners, y_corners, z_corners])

        # Define the rotation matrix around the Y-axis
        R = np.array([
            [np.cos(rotation), 0, np.sin(rotation)],
            [0, 1, 0],
            [-np.sin(rotation), 0, np.cos(rotation)]
        ])

        # Rotate the corners
        corners_rotated = np.dot(R, corners)

        # Translate the corners to the object's location in cam0
        corners_rotated[0, :] += location[0]
        corners_rotated[1, :] += location[1]
        corners_rotated[2, :] += location[2]


        # Convert to homogeneous coordinates
        corners_hom = np.vstack((corners_rotated, np.ones((1,
        ↪  corners_rotated.shape[1]))))

        # Transform from cam0 to velo coordinates using T_inv
        corners_velo_hom = np.dot(T_inv, corners_hom)
        corners_velo = corners_velo_hom[:3, :].T  # Shape: (8, 3)

        # Define the edges of the bounding box
        edges = [
            [0, 1], [1, 2], [2, 3], [3, 0],  # Top face
            [4, 5], [5, 6], [6, 7], [7, 4],  # Bottom face
            [0, 4], [1, 5], [2, 6], [3, 7]   # Vertical edges
        ]

        # Plot each edge of the bounding box
```

```
        for edge in edges:
            ax.plot(*corners_velo[edge, :].T, color='green', linewidth=2)

    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')
    ax.set_title('Point Cloud Visualization')

    #make background black
    ax.set_facecolor('black')
    #remove walls of the box for all sides
    ax.xaxis.pane.fill = False
    ax.yaxis.pane.fill = False
    ax.zaxis.pane.fill = False
    #remove grid
    ax.grid(False)
    #remove axis
    ax.axis('off')
    #keep axis scales equal to values
    ax.set_box_aspect([np.ptp(points[:, 0]), np.ptp(points[:, 1]), np.ptp(points[:,
    ↪  2])])

    #center plot on the middle of the point cloud
    ax.auto_scale_xyz(points[:, 0], points[:, 1], points[:, 2])


    plt.tight_layout()
    plt.show()
    #plt.savefig(output_filename, dpi=300, bbox_inches='tight')
    #plt.close()

if __name__ == '__main__':
    data = load_data('data/data.p')
    # Defines if only points within the camera's field of view should be rendered
    only_fov = False
    # Filter out all points except those that belong to the "car" class
    only_car = True
    render_point_cloud(data,only_fov, only_car)
```