An Implementation of Gradual Point Cloud Processing

Phuc Tran

Hamburg University of Applied Sciences, Berliner Tor 7, 20099 Hamburg, Germany

Abstract. This paper presents the implementation of Gradual Point Cloud Processing (GPCP) – a point cloud processing technique for mobile robots proposed in the previous work [1]. The implementation was experimented in object segmentation tasks using the SUN RGB-D dataset. The experiments showed that GPCP enables the robot to handle 3D point clouds using a 2D model.

Keywords: Point Cloud Segmentation · Gradual Point Cloud Processing.

1 Introduction

Point cloud, a set of point in 3D space, has become one of the most representation format for 3D data thanks to the increasing availability of 3D data acquisition devices. Over the past decade, various machine learning methods have been developed to work with point clouds such as PointNet [4], PointNet++ [5] and VoxelNet [6]. However, methods that perform the computation directly on the point set require high computing capability.

Gradual Point Cloud Processing (GPCP) [1] is a technique that uses projection to process point clouds gradually. GPCP was originally developed for mobile robots to navigate and understand the surroundings. The major arguments for applying GPCP are as follows:

1. In many robot vision tasks such as 3D object detection or segmentation, the robot often receives rather RGB-D than only depth data.
2. The robot constructs the point cloud gradually using SLAM as it navigates around. Therefore, the point cloud does not contain more information about an object that the robot is currently seeing than an RGB-D image since the object can only be partially observed.

Since the methodology was already explained in the previous work [1], this paper focuses mainly on the implementation of GPCP and the experiments.

2 Implementation

This section presents the implementation of GPCP. Open3D – an open-source library for 3D data processing [9] was used in this work for point cloud import and visualization.

2.1 Dataset & point cloud construction

The original idea was to utilize a ZED 2 Camera and construct a point cloud of the lab using SLAM to create a test dataset for this project. Unfortunately, the camera failed to produce usable results. The problem could be due to the poor factory calibration of the camera as other users also reported accuracy issues; or due to the error margin of the sensors inside the camera which may add up and lead to duplicated objects and false loop closure.

The SUN RGB-D [7] was used for the evaluation. The datasets contain RGB and depth images, allowing to reconstruct a point cloud of the scenes.

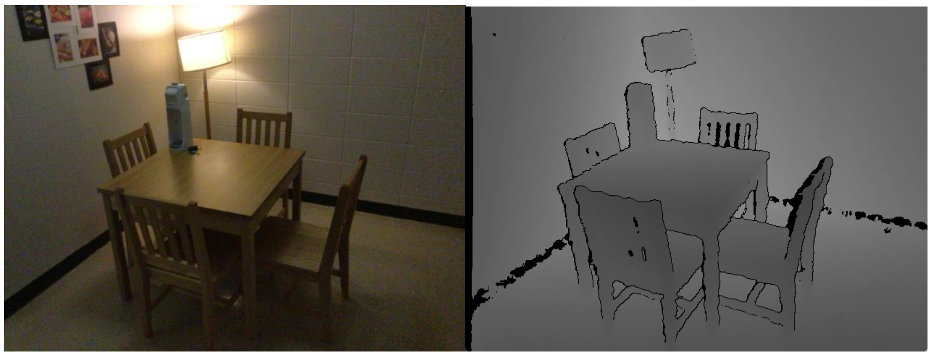

(a) RGB image (b) Depth image

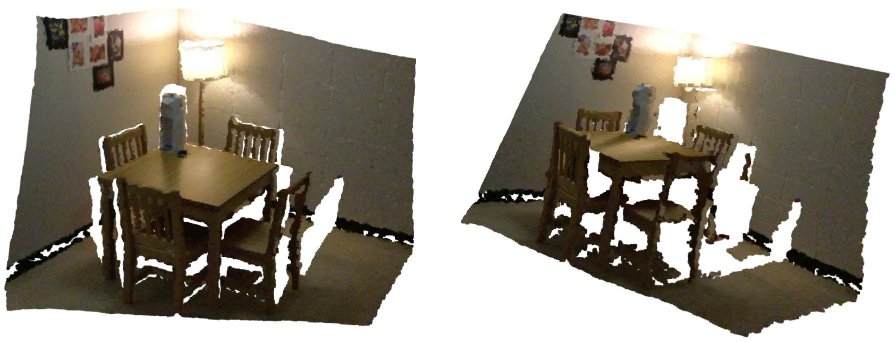
Fig. 1. A sample from the SUN RGB-D Dataset

The corresponding 3D point of a pixel at the coordinate (u, v) that has a depth value d can be computed using the intrinsic parameters of the camera as follows:

$$\begin{aligned} z &= d \\ x &= \frac{(u - c_x)z}{f_x} \\ y &= \frac{(v - c_y)z}{f_y} \end{aligned}$$

where f_x, f_y describe the focal length, and c_x, c_y describe the location of the principle point. The SUN RGB-D Dataset was recorded using a Kinect V2. Therefore, the intrinsic parameters are:

$$\begin{aligned} c_x &= 254.878 \\ c_y &= 205.395 \\ f_x &= 365.456 \\ f_y &= 365.456 \end{aligned}$$



(a) Front view (b) Side view

Fig. 2. The constructed point cloud from different view angles

2.2 Perspective projection

As described in the concept paper [1], perspective projection is employed to construct a 2D image from the point cloud. Since the point cloud was generated from RGB-D data, the point cloud is incomplete. Therefore, this work choosed the camera position as center of projection. However, arbitrary center of projection can be chosen when working with a complete point cloud.

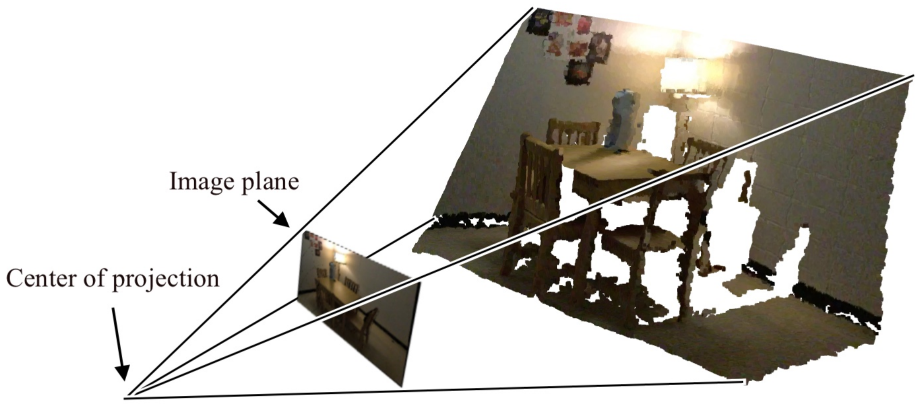


Fig. 3. Illustration of the perspective projection

The projection matrix consists of two components, namely the intrinsic matrix and the extrinsic matrix:

$$x_p = Px = K\,[R|t]\,x$$

The intrinsic matrix K used in this work is the intrinsic parameters of the Kinect V2. In order to simulate the sensor inaccuracy, this work added a small random rotation and translation in all axes when constructing the extrinsic matrix $[R|t]$. The symbol | denotes a horizontal concatenation.

The rotation matrix R is simply a chain of rotations in 3 axes,

$$R = \begin{bmatrix} cos(\theta_x) & -sin(\theta_x) & 0 \\ sin(\theta_x) & cos(\theta_x) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos(\theta_y) & 0 & sin(\theta_y) \\ 0 & 1 & 0 \\ -sin(\theta_y) & 0 & cos(\theta_y) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\theta_z) & -sin(\theta_z) \\ 0 & sin(\theta_z) & cos(\theta_z) \end{bmatrix}$$

and the translation vector t is $\begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$. The projection was implemented using Numpy as follows:

```
"""
:param X: List of points (N, 3)
:param K: Intrinsic matrix (3, 3)
:param R: Rotation matrix (3,3)
:param t: Translation vector (3, 1)
"""
# Extrinsic matrix
Rt = np.hstack([R, t])
# Projection matrix
P = K @ Rt
# Projection
X = X.swapaxes(0, 1)  # (N, 3) -> (3, N)
X = np.vstack([X, np.ones((1, X.shape[1]))]) # (4, N)
Xp = P @ X
# Normalize by Z
Z = Xp[2, :]
Xp = np.swapaxes((Xp / Z)[:2, :], 0, 1).astype(int)
```

Figure 4 shows the result of the perspective projection. One can notice that the perspective projection caused ”holes” in the 2D image. There are two types of holes:

1. Small holes that are caused by numerical error in the projection, and
2. Larger holes that are caused by lack of depth information.

The same issue is to be expected when working solely with point cloud data. It can be observed in that the quality of the 2D projection is still sufficient for object segmentation. However, depending on the 2D model, the degradation in quality of 2D image may cause poor accuracy, which will be discussed in the next section.

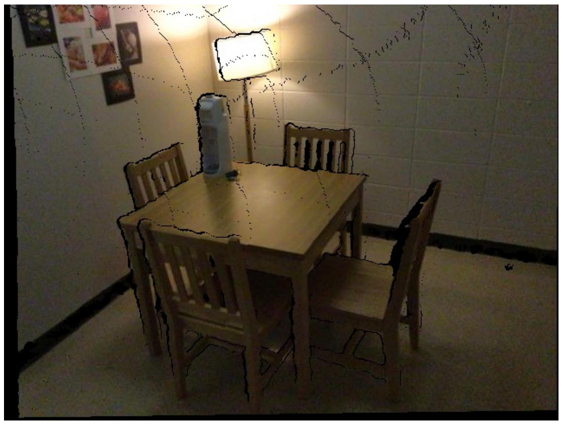


Fig. 4. The result of the perspective projection

2.3 Segmentation

This work uses a pre-trained DeepLabV3 [10] from Pytorch Hub [1] for the segmentation task. The backbone of the model is ResNet-101 [11].

This work normalized the input image using $mean = [0.485\ 0.456\ 0.406]$ and $std = [0.229\ 0.224\ 0.225]$.

The segmentation task was performed as usual using the following snippet:

```
preprocess = transforms.Compose([
transforms.ToTensor(),
transforms.Normalize(
  mean=[0.485, 0.456, 0.406],
  std=[0.229, 0.224, 0.225]),
])
input_tensor = preprocess(input_image)
input_batch = input_tensor.unsqueeze(0)  # create a mini-
                                          batch as expected by the model
if torch.cuda.is_available():
input_batch = input_batch.to('cuda')
model.to('cuda')
with torch.no_grad():
output = model(input_batch)['out'][0]
output = output.argmax(0)
```

Figure 5 shows the result of the segmentation, which is good since the model was able to cleanly segment the objects from the background and from other classes.

As mentioned in the previous section, the quality of the projection does have impact on the segmentation result on some samples. Figure 6 shows another

[1] https://pytorch.org/hub/pytorch_vision_deeplabv3_resnet101/

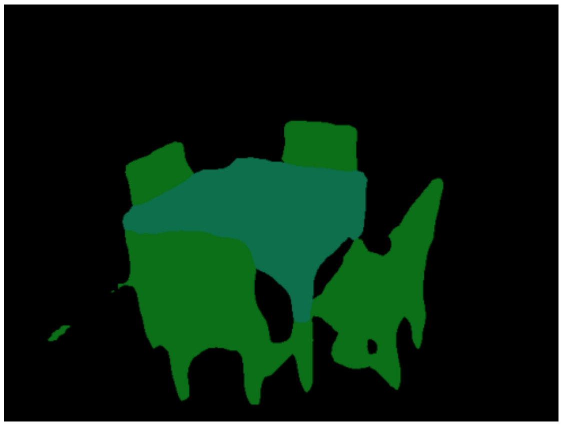


Fig. 5. The result of the segmentation task

image and the output of the segmentation task when performed directly on the RGB image.

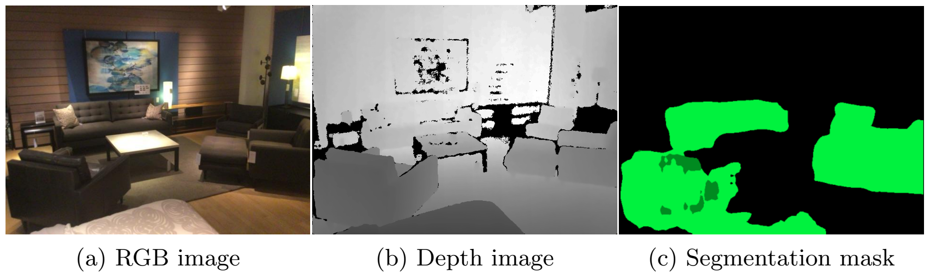


Fig. 6. The original image and the desired output

However, bad result of the segmentation on the projection can be observed in Figure 7. The main reason for the bad result is that the 2D segmentation model is not robust to noises and holes in the image. A possible solution is to employ another model that is more robust, or to use a denoising technique. In case of large holes in input images that denoising methods cannot help to reconstruct a good image, Autoencoder can by used to learn the latent representation of the image. This method, however, requires an additional self-supervised training stage for the Autoencoder.

2.4 Back-projection

After retrieving the segmentation mask, the last step is to project the result back to the original point cloud.

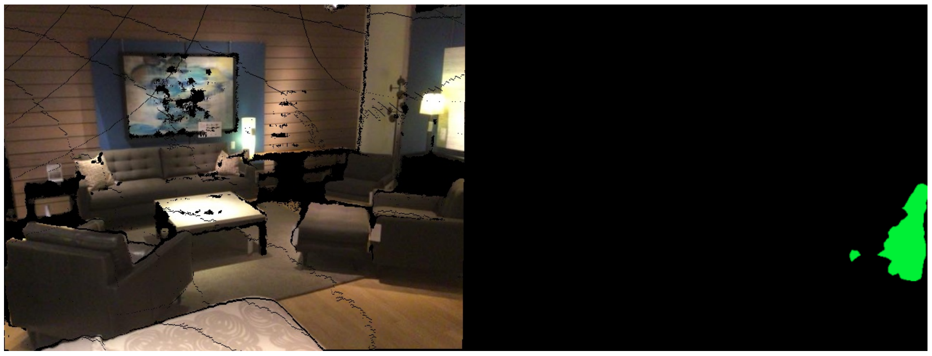

(a) Projection (b) Segmentation mask

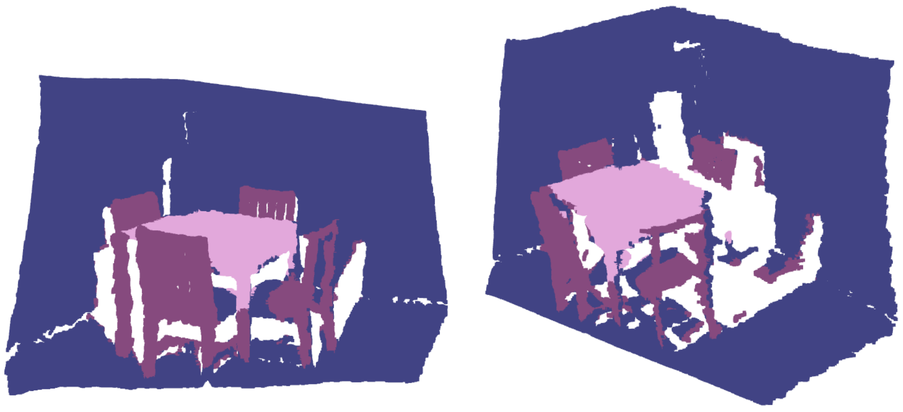
Fig. 7. The projection and the actual output

Instead of performing the actual back-projection, this work utilizes the reference of the points to perform label lookup with constant time complexity $\mathcal{O}(1)$ for each point.

Xp is the list of 2D points, *segmentation_mask* is the output of the segmentation task. Because the projection reserves the ordering of the points, one can iterate over the point cloud and access the assigned label at the projected (x, y) coordinates. Figure 8 shows the result after the back-projection. It can be observed that the quality of the result is comparable to that of the 2D segmentation.

```
# Back-projection
labels = np.zeros(Xp.shape[0])
for i in range(Xp.shape[0]):
  x, y = Xp[i]
  if x < 0 or x >= w or y < 0 or y >= h:
    labels[i] = 0
  else:
    labels[i] = segmentation_mask[y][x] # (h,w)

# Visualization
# Generate colors for the labels
max_label = labels.max()
palette = plt.get_cmap("tab20b")
colors = palette(labels / (max_label if max_label > 0 else
                              1))
colors[labels < 0] = 0
# Replace color values of the points
point_cloud.colors = o3d.utility.Vector3dVector(colors[:, :
                              3])
o3d.visualization.draw_geometries([point_cloud])
```



(a) Front view (b) Side view

Fig. 8. The point cloud after back-projection

3 Conclusion

Dealing with point cloud often requires high computing capability and training a model that works with point clouds requires a huge amount of data. This work experimentally proved that Gradual Point Cloud Processing has the ability to utilize a pre-trained 2D model to handle 3D data without requiring any training step.

As shown in the section 2.3, the main disadvantage of GPCP is that the quality of the result strongly depends on the 2D model. A model that is not robust to noises and holes in the input images can result to poor performance in some samples. This problem can be solved by employing a more robust model, a denoising method or using Autoencoder to learn the latent-space.

GPCP can bring many advantages. For example, in robotics, GPCP can be integrated with SLAM for map constructing and labeling on the fly. Moreover, GPCP can collect data and pre-label them, which will be more precisely labeled by humans later, so that a rich 3D dataset can be generated with significantly less cost.

References

1. Tran, Phuc: Self-supervised gradual 3D point clouds processing for robots, `https://users.informatik.haw-hamburg.de/~acf530/pub/gradual_processing.pdf`. Last accessed 10 Oct 2021
2. Tran, Phuc: An Implementation of Gradual Point Cloud Processing, `https://users.informatik.haw-hamburg.de/~acf530/pub/gradual_processing_impl.pdf`. Last accessed 05 Apr 2022
3. Bello, Saifullahi Aminu, et al. "Deep learning on 3D point clouds." Remote Sensing 12.11 (2020): 1729.

4. Qi, Charles R., et al. "Pointnet: Deep learning on point sets for 3d classification and segmentation." Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.
5. Qi, Charles R., et al. "Pointnet++: Deep hierarchical feature learning on point sets in a metric space." arXiv preprint arXiv:1706.02413 (2017).
6. Zhou, Yin, and Oncel Tuzel. "Voxelnet: End-to-end learning for point cloud based 3d object detection." Proceedings of the IEEE conference on computer vision and pattern recognition. 2018.
7. Song, Shuran, Samuel P. Lichtenberg, and Jianxiong Xiao. "Sun rgb-d: A rgb-d scene understanding benchmark suite." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.
8. Lai, Kevin, et al. "A large-scale hierarchical multi-view rgb-d object dataset." 2011 IEEE international conference on robotics and automation. IEEE, 2011.
9. Zhou, Qian-Yi, Jaesik Park, and Vladlen Koltun. "Open3D: A modern library for 3D data processing." arXiv preprint arXiv:1801.09847 (2018).
10. Chen, Liang-Chieh, et al. "Rethinking atrous convolution for semantic image segmentation." arXiv preprint arXiv:1706.05587 (2017).
11. He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.
12. Fei-Fei, Li, Jia Deng, and Kai Li. "ImageNet: Constructing a large-scale image database." Journal of vision 9.8 (2009): 1037-1037.