Graph-Based Point Tracker for 3D Object Tracking in Point Clouds

Minseong Park, Hongje Seong, Wonje Jang, Euntai Kim[*]

School of Electrical and Electronic Engineering, Yonsei University, Seoul, Korea
{msp922, hjseong, jangwj1256, etkim}@yonsei.ac.kr

Abstract

In this paper, a new deep learning network named as graph-based point tracker (GPT) is proposed for 3D object tracking in point clouds. GPT is not based on Siamese network applied to template and search area, but it is based on the transfer of target clue from the template to the search area. GPT is end-to-end trainable. GPT has two new modules: graph feature augmentation (GFA) and improved target clue (ITC) module. The key idea of GFA is to exploit one-to-many relationship between template and search area points using a bipartite graph. In GFA, edge features of the bipartite graph are generated by transferring the target clues of template points to search area points through edge convolution. It captures the relationship between template and search area points effectively from the perspective of geometry and shape of two point clouds. The second module is ITC. The key idea of ITC is to embed the information of the center of the target into the edges of the bipartite graph via Hough voting, strengthening the discriminative power of GFA. Both modules significantly contribute to the improvement of GPT by transferring geometric and shape information including target center from target template to search area effectively. Experiments on the KITTI tracking dataset show that GPT achieves state-of-the-art performance and can run in real-time.

Introduction

The goal of 3D object tracking is to estimate the 3D bounding box of a target within the search area of the current frame when the 3D bounding box of the target in the template of the previous frame is given. 3D object tracking is attracting considerable attention within the society of computer vision and robotics since it is used in various applications ranging from autonomous driving to mobile robotics. To realize 3D object tracking, we need a sensor or a combination of multiple sensors. Among the various sensors or their combinations, the two most popular sensors for 3D object tracking are RGB-D sensor and LiDAR.

RGB-D sensor can provide visual as well as geometric information about the target, and it has been widely used for 3D object tracking in indoor applications (Song and Xiao 2013; Held et al. 2016; Bibi, Zhang, and Ghanem 2016; Kart, Kamarainen, and Matas 2018; Xiao et al. 2018; Kart et al. 2019). However, RGB-D has a drawback; that the maximum range of the sensor is relatively short and visual information is sensitive to illumination changes, preventing them from being used in outdoor applications.

LiDAR is also being used widely in outdoor applications such as autonomous driving. LiDAR outputs a point cloud that captures the contour of the nearby environment and is robust to illumination changes. However, when using LiDAR, it is difficult to train the features from the unstructured point cloud using CNNs, because of the sparsity and disorder of the point cloud. In this paper, we will focus on 3D object tracking in point clouds from LiDAR.

3D tracking in the point cloud was initially motivated by the success of Siamese network in 2D object trackers (Bertinetto et al. 2016; Guo et al. 2017; He et al. 2018; Wang et al. 2018; Li et al. 2018a). As a pioneering work, Giancola, Zarzar, and Ghanem (Giancola, Zarzar, and Ghanem 2019) proposed a 3D Siamese tracker that encodes model and candidate shapes into a latent representation regularized by shape completion. Based on this 3D Siamese tracker baseline, Zarzar, Giancola, and Ghanem (Zarzar, Giancola, and Ghanem 2020) and F-Siamese tracker (Zou et al. 2020) which take double Siamese networks were proposed. They first generate proposals and then refine the proposals. More specifically, Zarzar, Giancola, and Ghanem generate a small number of proposals from bird eye view (BEV) Siamese network using BEV representation of point clouds. F-Siamese tracker generates frustum based proposals from the result of 2D Siamese tracker using images. They improved the performance of the 3D Siamese tracker baseline with an efficient region proposal network, but they still suffer from the loss of geometric details due to the one-dimensional representation of each proposal. Recently, P2B (Qi et al. 2020), a new method based on the transfer of information from the template to the search area, was proposed as an alternative to the Siamese approach. P2B augments the features by transferring the target clue from the template to the search area and localizes the 3D target bounding box using the augmented feature in the search area. P2B demonstrated excellent performance compared to the previous 3D Siamese tracker. Despite its success in 3D object tracking, we believe that there are still some inefficiencies in the feature augmentation in P2B. Specifically, (1) the similarity used in P2B is not sufficient to capture the details of the one-to-one relationship

[*]Corresponding author.

interwoven between a template point and a search area point because each similarity is represented only as a scalar value. (2) A one-to-all relationship between a single search area point and all the template points is considered in P2B, but it may include uninformative features or cause ambiguity in search area feature representations.

To address the above concerns in P2B, we propose a new deep learning network named graph-based point tracker (GPT). Our GPT is motivated by P2B and thus, it is based on the transfer of target clue from the template to the search area. The main difference between GPT and P2B lies in how the features from the template and the search area are combined. Unlike P2B, our GPT exploits the one-to-many relationship between a search area point and its nearby template points, augmenting features more effectively than P2B. Our GPT has two new modules for effective feature augmentation: (1) graph feature augmentation (GFA) and (2) improved target clue (ITC). In the GFA module, we model the one-to-many relationship between a search area point and its nearby template points using a bipartite graph and extract features from the edges of the graph using edge convolution. In the ITC module, we move the information of the target from the template seed to the template center via Hough voting (Qi et al. 2019) and embed the template center information into the edges of the bipartite graph. Finally, the outputs from the two modules are combined to regress the final 3D box.

The main contributions of our GPT are threefold. (1) We propose a graph feature augmentation module (GFA) that builds one-to-many relationships between the template and search area, transferring target clues effectively from the template to search area. (2) We propose a novel improved target clue (ITC) that captures the information of the template target center. (3) Our GPT achieves state-of-the-art performance on the KITTI tracking dataset (Geiger, Lenz, and Urtasun 2012) and is suitable for real-time applications (38 FPS on a single GPU).

Related Works

Deep Learning on Point Clouds. Deep learning on point clouds is challenging because of the sparsity and disorder of unstructured point clouds. Ealy studies (Su et al. 2015; Maturana and Scherer 2015) addressed this issue by converting a point cloud to structured data, such as a 2D image or voxel grid. PointNet (Qi et al. 2017a) presented a method that learns a global representation without input data converting, using point-wise MLPs. To learn local representation, some studies (Qi et al. 2017b; Xu et al. 2018; Li et al. 2018b; Zhao et al. 2019; Thomas et al. 2019) have proposed using sampling and grouping to aggregate the information of neighbor points. Others (Wang et al. 2019; Xie et al. 2018; Yang et al. 2019; Yan et al. 2020) have proposed capturing global context using k-NN graphs in feature space or an attention mechanism. Recently, deep learning is widely adopted for several tasks with point clouds such as 3D registration (Aoki et al. 2019; Wang and Solomon 2019), 3D object detection (Shi, Wang, and Li 2019; Shi et al. 2020; Qi et al. 2019; Pan et al. 2021), and 3D object tracking (Giancola, Zarzar, and Ghanem 2019; Qi et al. 2020).

3D Object Tracking in Point Clouds. As a pioneering work, Giancola, Zarzar, and Ghanem (Giancola, Zarzar, and Ghanem 2019) presented 3D Siamese tracker. This approach generates target proposals using Kalman filter and leverages shape completion for latent vector regularization, but it cannot be trained end-to-end. Motivated by the aforementioned study, Zarzar, Giancola, and Ghanem (Zarzar, Giancola, and Ghanem 2020) and F-Siamese tracker (Zou et al. 2020) apply double Siamese networks, one each for a better region proposal from bird eye view representation of point clouds and 2D image. However, they depend more on generating the search area than the performance of the tracker itself. Recently, P2B (Qi et al. 2020), which aims to first embed the information of the template into the search area, and then to localize the target center in the search area, has been proposed.

Feature Augmentation. There have been several recent methods for one-shot learning (Dixit et al. 2017; Chen et al. 2019), domain adaptation (Volpi et al. 2018), imbalanced classification (Zhang et al. 2019), visual recognition (Chen et al. 2021), and semantic segmentation in point cloud (Qiu, Anwar, and Barnes 2021), which applied feature augmentation to avoid lack of clues and to enrich features. P2B (Qi et al. 2020) first applied feature augmentation in the 3D object tracking. It builds relationship between the template and the search area using similarity matrix. This approach, however, overlooks the reliable relationship building, which is paramount to accurately transfer tracking clues from template to search area. Specifically, P2B relates each relationship using only a scalar value (similarity), and considers all template points for a search area point, which causes the loss of details for the relationship and builds redundant relationship. To address this issues, We introduce one-to-many relationship via bipartite graph embedding.

Method

Overview

When a target is given in the form of a template point cloud $P_{tmp} = \{x_{T,i} \in \mathbb{R}^3\}_{i=1}^{N_1}$, the problem considered herein is 3D tracking and it aims at tracking a 3D bounding box in the search area point cloud $P_{sea} = \{x_{S,j} \in \mathbb{R}^3\}_{j=1}^{N_2}$ of the search area so that the bounding box encompasses the target in P_{sea}, where N_1 and N_2 denote the number of points in the two clouds P_{tmp} and P_{sea}, respectively.

To solve the problem, a new framework named graph-based point tracker (GPT) is proposed. GPT consists of two parts: (1) feature augmentation and (2) 3D box regression. The architecture of GPT is motivated by P2B (Qi et al. 2020) and it consists of two parts. GPT is also end-to-end trainable.

The first part of GPT is graph feature augmentation (GFA). This part aims to transfer the target clue of the template P_{tmp} to the search area P_{sea} via a bipartite graph embedding, which is the key contribution of this paper. The target clue of a template point is embedded into the corresponding search area point (later, the point is called a node) in the graph by edge convolution (Wang et al. 2019). Furthermore, the center of the target is also embedded into the search area P_{sea} as an additional target clue via the improved

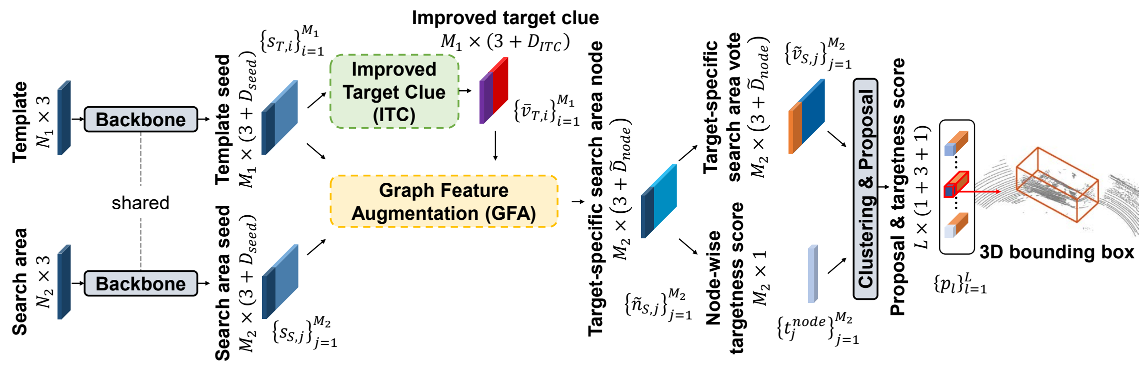
Template
$N_1 \times 3$
Backbone
shared
Search area
$N_2 \times 3$
Backbone
Template seed
$M_1 \times (3 + D_{seed})$
$\{s_{T,i}\}_{i=1}^{M_1}$
Search area seed
$M_2 \times (3 + D_{seed})$
$\{s_{S,j}\}_{j=1}^{M_2}$
Improved target clue
$M_1 \times (3 + D_{ITC})$
Improved Target Clue (ITC)
$\{\tilde{v}_{T,i}\}_{i=1}^{M_1}$
Graph Feature Augmentation (GFA)
Target-specific search area node
$M_2 \times (3 + \tilde{D}_{node})$
$\{\tilde{n}_{S,j}\}_{j=1}^{M_2}$
Target-specific search area vote
$M_2 \times (3 + \tilde{D}_{node})$
$\{\tilde{v}_{S,j}\}_{j=1}^{M_2}$
Node-wise targetness score
$M_2 \times 1$
$\{t_j^{node}\}_{j=1}^{M_2}$
Clustering & Proposal
Proposal & targetness score
$L \times (1 + 3 + 1)$
$\{p_l\}_{l=1}^{L}$
3D bounding box

Figure 1: Pipeline of GPT for 3D object tracking in point clouds. Given two input point clouds template and search area, the backbone network samples seed points. The search area seeds are augmented to target-specific search area nodes using graph feature augmentation module with improved target clue which provides additional information of template. target-specific search area nodes are processed as target proposals by voting and classification branches. Finally, the target proposal with highest targetness score is verified as final 3D bounding box.

target clue (ITC) module. ITC aims to strengthen the discriminative power of a point (= node) in the search area P_{sea} by providing the information about the center of the target.

The second part of GPT regresses a 3D bounding box using the augmented features from the first part. In this part, a point (=node) in the search area P_{sea} is projected to the potential center of the target via Hough voting. This projection lowers the likelihood of there being no search area nodes in the vicinity of the target center, and the projection improves the accuracy of the 3D tracking. Then, the potential centers are grouped into L clusters, making L target proposals. The architecture of GPT is summarized in Fig. 1. The tensor symbols used in the subsequent sections are also depicted in the figure.

Feature Augmentation

Here, we transfer feature information from the target template to the search area. Specifically, the geometric and shape information of the template seeds are transferred to the search area seeds so that search area seeds can find the corresponding template seeds with similar geometric and shape information effectively. This step is motivated by the target specific feature augmentation in P2B (Qi et al. 2020) but its details are completely different. The main difference between them is how the geometric information among individual template seeds is used. *The key idea of this paper is that edges are connected between the template and search area seeds in a bipartite graph, and the clues of template seeds are transferred to search area seeds through edge convolution*. Details will be provided at the end of the section.

Seed Encoding. First, we feed the two sets of points $\{x_{T,i}\}_{i=1}^{N_1}$ and $\{x_{S,j}\}_{j=1}^{N_2}$ to a backbone network to obtain seed points $\{s_{T,i} = [x_{T,i}; f_{T,i}^{seed}] \in \mathbb{R}^{3+D_{seed}}\}_{i=1}^{M_1}$ and $\{s_{S,j} = [x_{S,j}; f_{S,j}^{seed}] \in \mathbb{R}^{3+D_{seed}}\}_{j=1}^{M_2}$, respectively, where M_1 and M_2 denote the number of seeds taken from the two sets of clouds P_{tmp} and P_{sea}, respectively; $M_1 < N_1$ and $M_2 < N_2$; and the indices i and j in the sets $\{s_{T,i}\}_{i=1}^{M_1}$ and $\{s_{S,j}\}_{j=1}^{M_2}$ are slightly abused, but they do not cause confusion; D_{seed} is the dimension of the output from the backbone network. Here, PointNet++ (Qi et al. 2017b) is used as a feature backbone, but the backbone is not restricted to it.

Graph Feature Augmentation. After seed encoding, we feed two sets of features $\{f_{T,i}^{seed}\}_{i=1}^{M_1}$ and $\{f_{S,j}^{seed}\}_{j=1}^{M_2}$ to feature transformation functions $\phi : \mathbb{R}^{D_{seed}} \to \mathbb{R}^{D_{node}}$ and $\varphi : \mathbb{R}^{D_{seed}} \to \mathbb{R}^{D_{node}}$, and make two sets of nodes $\mathcal{N}_T = \{n_{T,i} = [x_{T,i}; f_{T,i}^{node}] \in \mathbb{R}^{3+D_{node}}\}_{i=1}^{M_1}$ and $\mathcal{N}_S = \{n_{S,j} = [x_{S,j}; f_{S,j}^{node}] \in \mathbb{R}^{3+D_{node}}\}_{j=1}^{M_2}$, respectively, as shown in Fig. 2, where

$$\begin{aligned} f_{T,i}^{node} &= \phi\left(f_{T,i}^{seed}\right), i = 1, 2, \cdots, M_1, \\ f_{S,j}^{node} &= \varphi\left(f_{S,j}^{seed}\right), j = 1, 2, \cdots, M_2; \end{aligned} \tag{1}$$

D_{node} is the dimension of the output from the two transformation functions ϕ and φ; and $D_{node} < D_{seed}$. Functions ϕ and φ are implemented using 1×1 convolutions. They aim at reducing the feature dimension of point clouds from D_{seed} to D_{node}, reducing computational resources. By connecting the directed edges from $\mathcal{N}_T$ to $\mathcal{N}_S$ using k-nearest neighbor (k-NN), we build a bipartite graph $\mathcal{G} = (\mathcal{N}_T, \mathcal{N}_S, \mathcal{E})$. That is, all the edges in $\mathcal{G}$ are directed from $\mathcal{N}_T$ to $\mathcal{N}_S$, and thus the set of edges are represented by $\mathcal{E} = \mathcal{N}_T \times \mathcal{N}_S$. When building a bipartite graph $\mathcal{G}$, k-NN is performed based on the Euclidean distance of $\{f_{T,i}^{node}\}_{i=1}^{M_1}$ and $\{f_{S,j}^{node}\}_{j=1}^{M_2}$ in the feature space. This implies that the bipartite graph $\mathcal{G}$ embodies relationship between similar-looking points. In the bipartite graph $\mathcal{G}$, the number of incoming edges of the nodes in $\mathcal{N}_S$ is k ($\text{indeg}(n_{S,j}) = k$) but the number of incoming edges of the nodes in $\mathcal{N}_T$ is zero ($\text{indeg}(n_{T,i}) = 0$).

Next, target clues from the template nodes $\mathcal{N}_T$ are transferred to the search area nodes $\mathcal{N}_S$ using edge convolution (EdgeConv) (Wang et al. 2019) through a bipartite graph $\mathcal{G}$. Specifically, we define the edge features between two nodes $n_{T,i}$ and $n_{S,j}$ as $e_{ij} = h_\Theta(n_{T,i}, n_{S,j})$ and compute the output of EdgeConv at the j-th node $n_{S,j}$ in the search area $\mathcal{N}_S$ by

$$\tilde{f}_{S,j}^{node} = \bigoplus_{i:(i,j)\in\mathcal{E}} e_{ij} = \bigoplus_{i:(i,j)\in\mathcal{E}} h_\Theta\left(n_{T,i}, n_{S,j}\right), \tag{2}$$

where $h_\Theta : \mathbb{R}^{3+D_{node}} \times \mathbb{R}^{3+D_{node}} \to \mathbb{R}^{\tilde{D}_{node}}$ is a nonlinear function with a set of learnable parameters Θ and $\oplus$ is a

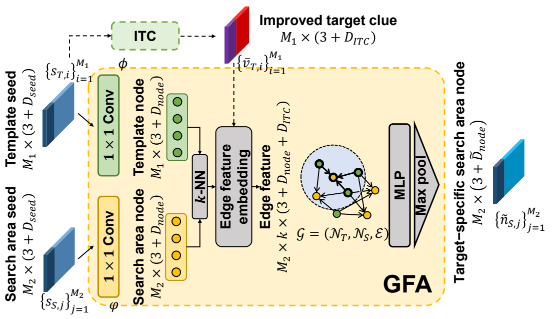
ITC
Improved target clue
$M_1 \times (3 + D_{ITC})$
$\{\bar{v}_{T,i}\}_{i=1}^{M_1}$
$\{s_{T,i}\}_{i=1}^{M_1}$
ϕ
Template seed
$M_1 \times (3 + D_{seed})$
1 × 1 Conv
Template node
$M_1 \times (3 + D_{node})$
k-NN
Edge feature embedding
Edge feature
$M_2 \times k \times (3 + D_{node} + D_{ITC})$
$\mathcal{G} = (\mathcal{N}_T, \mathcal{N}_S, \mathcal{E})$
MLP
Max pool
Target-specific search area node
$M_2 \times (3 + \tilde{D}_{node})$
$\{\tilde{n}_{S,j}\}_{j=1}^{M_2}$
Search area seed
$M_2 \times (3 + D_{seed})$
$\{s_{S,j}\}_{j=1}^{M_2}$
φ
1 × 1 Conv
Search area node
$M_2 \times (3 + D_{node})$
GFA

Figure 2: Graph feature augmentation.

simple symmetric aggregation function and makes up for the disorder of edges in a graph. Then, using the output of EdgeConv $\tilde{f}_{S,j}^{node}$, the *target-specific search area node* is defined by

$$\tilde{n}_{S,j} = \left[x_{S,j}; \tilde{f}_{S,j}^{node}\right] \in \mathbb{R}^{3+\tilde{D}_{node}}. \quad (3)$$

In this paper, tilde ($\sim$) implies that the corresponding variable is a character related to target-specific quantities. In EdgeConv, the edge features $e_{ij} = h_\Theta(n_{T,i}, n_{S,j})$ between two nodes $n_{T,i} = [x_{T,i}; f_{T,i}^{node}]$ and $n_{S,j} = [x_{S,j}; f_{S,j}^{node}]$ are implemented using MLP by

$$e_{ij} = h_\Theta\left(n_{T,i}, n_{S,j}\right) = MLP\left(\begin{bmatrix} x_{T,i} \\ f_{T,i}^{node} - f_{S,j}^{node} \\ f_{T,i}^{node} \end{bmatrix} \Big| \Theta\right) \in \mathbb{R}^{\tilde{D}_{node}} \quad (4)$$

and max pooling is used as a symmetric aggregation $\oplus$. That is, a target-specific search area node $\tilde{n}_{S,j} = [x_{S,j}; \tilde{f}_{S,j}^{node}]$ is realized by

$$\tilde{f}_{S,j}^{node} = \bigoplus_{i:(i,j)\in\mathcal{E}} e_{ij} = \max_{i:(i,j)\in\mathcal{E}} MLP\left(\begin{bmatrix} x_{T,i} \\ f_{T,i}^{node} - f_{S,j}^{node} \\ f_{T,i}^{node} \end{bmatrix} \Big| \Theta\right) \quad (5)$$

and the input of the MLP h_Θ is $[x_{T,i}; f_{T,i}^{node} - f_{S,j}^{node}; f_{T,i}^{node}] \in \mathbb{R}^{3+2D_{node}}$. Here, it should be noted that the target-specific search area node $\tilde{n}_{S,j}$ obtained from the bipartite graph $\mathcal{G}$, and it includes the geometric position of nearby template nodes, global template shape, and local shape difference between the template and search area nodes. The geometric position is captured by $x_{T,i}$; and the global template shape, and local shape difference between target and search area nodes are captured by $f_{T,i}^{node}$ and $f_{T,i}^{node} - f_{S,j}^{node}$, respectively. This step is named Graph Feature Augmentation (GFA).

Improved Target Clue. We believe that there is still room to exploit in the bipartite graph embedding. Our approach considers the center of the template seeds $s_{T,i}$. The template seeds $s_{T,i}$ only capture the limited local clue and they do not have global information about the their center. In addition, there are often few template seeds $s_{T,i}$ around the template center and thus the center is very likely to fall in empty space due to the inherent sparsity of point cloud. To address this issue, we use a voting module (Qi et al. 2019) to generate potential template center $v_{T,i}$ which lie closer to the template center than template seeds $s_{T,i}$ while maintaining the template features. The voting module is implemented using another MLP. It takes the template seed $s_{T,i} = [x_{T,i}; f_{T,i}^{seed}]$ and outputs the geometric space offset

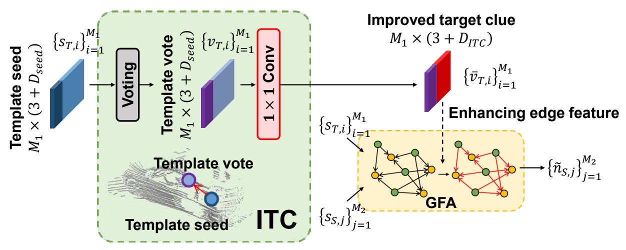
Template seed
$M_1 \times (3 + D_{seed})$
$\{s_{T,i}\}_{i=1}^{M_1}$
Voting
Template vote
$M_1 \times (3 + D_{seed})$
$\{v_{T,i}\}_{i=1}^{M_1}$
1 × 1 Conv
Template vote
Template seed
ITC
Improved target clue
$M_1 \times (3 + D_{ITC})$
$\{\bar{v}_{T,i}\}_{i=1}^{M_1}$
Enhancing edge feature
$\{s_{T,i}\}_{i=1}^{M_1}$
$\{s_{S,j}\}_{j=1}^{M_2}$
GFA
$\{\tilde{n}_{S,j}\}_{j=1}^{M_2}$

Figure 3: Improved target clue module.

$\Delta x_{T,i} \in \mathbb{R}^3$ and a feature offset $\Delta f_{T,i} \in \mathbb{R}^{D_{seed}}$, generating a template vote $v_{T,i} = [y_{T,i}; f_{T,i}^{vote}]$, where $y_{T,i} = x_{T,i} + \Delta x_{T,i}$ and $f_{T,i}^{vote} = f_{T,i}^{seed} + \Delta f_{T,i}$. Finally, we apply 1×1 convolution to the vote to make $\bar{v}_{T,i} = [y_{T,i}; \bar{f}_{T,i}^{vote}] \in \mathbb{R}^{3+D_{ITC}}$, as shown in Fig. 3, and the set of new points $\{\bar{v}_{T,i}\}_{i=1}^{M_1}$ are actually used to represent the global shape of the template point cloud P_{tmp}. In other words, the feature $f_{T,i}^{node}$ of the template node in Eq. 5 is replaced with the feature $\bar{f}_{T,i}^{vote}$ of the template vote and target-specific search area node $\tilde{n}_{S,j} = [x_{S,j}; \tilde{f}_{S,j}^{node}]$ is computed using edge convolution:

$$\tilde{f}_{S,j}^{node} = \bigoplus_{i:(i,j)\in\mathcal{E}} e_{ij} = \max_{i:(i,j)\in\mathcal{E}} MLP\left(\begin{bmatrix} x_{T,i} \\ f_{T,i}^{node} - f_{S,j}^{node} \\ \bar{f}_{T,i}^{vote} \end{bmatrix} \Big| \Theta\right) \quad (6)$$

Target Proposal

As in P2B (Qi et al. 2020), we move target-specific search area nodes $\{\tilde{n}_{S,j}\}_{j=1}^{M_2}$ to the potential target center in the search area via Hough voting as shown in Fig. 1. Specifically, the voting module (which is different from the one in ITC) takes the target-specific search area node $\tilde{n}_{S,j}$ and outputs the geometric space offset $\Delta x_{S,j} \in \mathbb{R}^3$ and a feature offset $\Delta \tilde{f}_{S,j}^{node} \in \mathbb{R}^{\tilde{D}_{node}}$, generating a target-specific search area vote $\tilde{v}_{S,j} = [y_{S,j}; \tilde{f}_{S,j}^{vote}]$ where $y_{S,j} = x_{S,j} + \Delta x_{S,j}$ and $\tilde{f}_{S,j}^{vote} = \tilde{f}_{S,j}^{node} + \Delta \tilde{f}_{S,j}^{node}$. The classification module takes the feature $\tilde{f}_{S,j}^{node}$ and outputs the search area node-wise targetness-score $t_j^{node} \in [0,1]$. From the set of target-specific search area votes $\{\tilde{v}_{S,j}\}_{j=1}^{M_2}$, we use random sampling to choose a subset of votes $\{\tilde{v}_{S,\ell}\}_{\ell=1}^{L}$, where L is the number of target proposals. Then, for each target-specific search area vote $\tilde{v}_{S,\ell}$, we apply the ball query (Qi et al. 2017b) to generate a cluster $C_\ell = \{\tilde{v}_{S,j} \| \|y_{S,j} - y_{S,\ell}\|_2 < R\}$ with a radius R. Then, MLP followed by max pooling is applied to each cluster to obtain the cluster-wise feature vector, and another MLP takes the cluster-wise feature vector to generate proposal offset $\Delta y_{S,\ell}$, rotation θ_ℓ and proposal-wise targetness score t_ℓ^{prop}:

$$\begin{bmatrix} t_\ell^{prop} \\ \Delta y_{S,\ell} \\ \theta_\ell \end{bmatrix} = MLP\left\{\max_{j:\tilde{v}_{S,j}\in C_\ell} MLP\left(\begin{bmatrix} t_j^{node} \\ y_{S,j} \\ \tilde{f}_{S,j}^{vote} \end{bmatrix}\right)\right\}. \quad (7)$$

Then, the box proposal is represented by $p_\ell = [t_\ell^{prop}; y_{S,\ell} + \Delta y_{S,\ell}; \theta_\ell] \in \mathbb{R}^{1+3+1}$. Finally, the box proposal p_ℓ with highest t_ℓ^{prop} is verified as final target bounding box.

Loss

The loss functions of GPT consists of voting regression loss, the loss about seed-wise targetness and proposal-wise targetness, and the loss about the accuracy of target bounding box.

First, let us consider the voting regression loss that is used in the voting in template and search area. Unlike P2B, VoteNet is applied not only to the search area but also to the template. First, the search area voting regression loss $\mathcal{L}_{\text{sea-reg}}$ defined as

$$\mathcal{L}_{\text{sea-reg}} = \frac{1}{M_{SN}} \sum_{j} \left\| \Delta x_{S,j} - \Delta gt_{S,j} \right\| \cdot \mathbb{I}\left(\tilde{n}_{S,j} \text{ on target}\right) \quad (8)$$

is used, and it is the same as in P2B, where $\Delta gt_{S,j}$ is the ground-truth offset from $\tilde{n}_{S,j}$ to the target center in the search area and M_{SN} denotes the number of trained nodes in the search area, where $M_{SN} = \sum \mathbb{I}(\tilde{n}_{S,j} \text{ on target})$. Second, the template voting regression loss $\mathcal{L}_{\text{temp-reg}}$ is defined similarly, as

$$\begin{aligned}\mathcal{L}_{\text{tmp-reg}} &= \frac{1}{M_{TS}} \sum_{i} \left\| \Delta x_{T,i} - \Delta gt_{T,i} \right\| \cdot \mathbb{I}\left(s_{T,i} \text{ on target}\right) \\ &= \frac{1}{M_1} \sum_{i=1}^{M_1} \left\| (x_{T,i} + \Delta x_{T,i}) - (x_{T,i} + \Delta gt_{T,i}) \right\| \quad (9) \\ &= \frac{1}{M_1} \sum_{i=1}^{M_1} \left\| x_{T,i} + \Delta x_{T,i} \right\|.\end{aligned}$$

In the above equation, $\mathbb{I}(s_{T,i} \text{ on target})$ is removed because all the points $s_{T,i}$ in P_{tmp} are on the target in template and M_{TS} denotes the number of trained seeds in the template and $M_{TS} = M_1$. $\Delta gt_{T,i}$ is the offset from $x_{T,i}$ to template center; $x_{T,i} + \Delta gt_{T,i} = 0$ as it denotes that the template center is the origin of the coordinate. Then the regression loss $\mathcal{L}_{\text{reg}}$ is computed as $\mathcal{L}_{\text{reg}} = \lambda_{\text{sea-reg}}\mathcal{L}_{\text{sea-reg}} + \lambda_{\text{temp-reg}}\mathcal{L}_{\text{temp-reg}}$, where $\mathcal{L}_{\text{sea-reg}}$ and $\mathcal{L}_{\text{tmp-reg}}$ are loss weights of search area and template regression. In addition, we use a node-wise targetness score loss $\mathcal{L}_{\text{cls}}$, a proposal-wise targetness score loss $\mathcal{L}_{\text{prop}}$, and a target bounding box proposal loss $\mathcal{L}_{\text{box}}$, as in P2B, where $\mathcal{L}_{\text{cls}}$ and $\mathcal{L}_{\text{prop}}$ are defined using binary cross entropy and $\mathcal{L}_{\text{box}}$ is defined in terms of the 3D center of bounding box and yaw angle using smooth-L1 loss. Then the final training loss $\mathcal{L}$ is computed as

$$\mathcal{L} = \mathcal{L}_{\text{reg}} + \lambda_{\text{cls}}\mathcal{L}_{\text{cls}} + \lambda_{\text{box}}\mathcal{L}_{\text{box}} + \lambda_{\text{prop}}\mathcal{L}_{\text{prop}}, \quad (10)$$

where λ_{cls}, λ_{box}, and λ_{prop} are loss weights of seed-wise targetness score, target bounding box proposal, and proposal-wise targetness score, respectively.

Comparison with P2B

The main difference between our GPT and P2B (Qi et al. 2020) is how the template information is combined with the search area information. In P2B, the clues from the template seeds are transferred to search area seeds through two kinds of channels. The first one is the local tracking clue and the second one is the global target clue. P2B tries to cover the entire spectrum of clues by combining local and global clues. Unfortunately, however, we believe that some inefficiency still remains in the combination. First, let us consider the problem of the local tracking clue. The local tracking clue is implemented using a similarity in P2B. The clue implies a one-to-one relationship (similarity) between all the possible pairs of a single template seed and a single search area seed. Each one-to-one relationship is represented only as a scalar value, so the details about difference between the related pairs are lost.

Next, let us consider the problem of the global target clue. This clue can be considered a one-to-all (not one-to-many) relationship. As before, if all the template seeds are considered in the association, uninformative and ineffective seeds that are far from a certain search area seed will be used in the association, which will degrade the performance. In summary, the local tracking clue and the global target clue in P2B can be considered as a one-to-one and a one-to-all relationship, respectively, but they are ineffective. What we really need is one-to-many relationship with reliable one-to-one relationships, and our GPT is the actual realization of this via bipartite graph embedding.

The ITC is an aspect that represents the difference between P2B and our GPT. In P2B, only the seeds in the search area are moved to the center of the target in the search area by VoteNet, whereas not only the seeds in the search area but also those in the template are moved to their respective centers by VoteNet in GPT. The basic idea on the use of VoteNet in the template is that the center of the template provides another important clue on the target in the search area, and it should be transferred to the search area nodes.

Experiments

Experimental Setting

Dataset. To validate our GPT, we use the KITTI tracking dataset as a benchmark (Geiger, Lenz, and Urtasun 2012). As the ground truth (GT) of its test set is not available, we divide its training set into three sets, and use them for training, validation, and testing. The KITTI training set has 21 scenes. Following the settings in previous works (Giancola, Zarzar, and Ghanem 2019; Qi et al. 2020), we use scenes 0-16 for training, scenes 17-18 for validation and scenes 19-20 for testing. Although there are eight types of targets in the dataset, we use only four types (Car, Pedestrian, Van, and Cyclist) due to the lack of training data for the other types. For each scene, we generate tracklets by concatenating the frames containing the same target instance of the above four types by time order for their target instances.

In training, template point cloud P_{tmp} is generated by accumulating the points within the first and the immediately preceding GT boxes. Search area point cloud P_{sea} is selected by lengthening each edge of the previous GT box by 4 m (i.e., by 2 m in each direction from the center of the edge) and sampling points from the enlarged box (including background points). The template and search area point clouds are normalized to $N_1 = 512$ and $N_2 = 1024$, respectively, by randomly abandoning or duplicating points. When more training points are needed, random offset is applied to the center of the GT box. Similarly, in testing, template P_{tmp} is generated by accumulating the points within the first GT box and the immediately preceding tracking result. The search area is also selected by enlarging the previous tracking result (=box) by 4 m on each edge and sampling points from the enlarged tracking box.

Evaluation Metric. As in other papers regarding the tracking (Giancola, Zarzar, and Ghanem 2019; Qi et al. 2020), we use One Pass Evaluation (OPE) (Wu, Lim, and Yang 2013) as an evaluation metric to compare GPT with

		Car 6424	Ped 6088	Van 1248	Cyc 308	Mean
Success	SC3D	41.3	18.2	40.4	41.5	31.2
	BEV-3D	36.3	17.9	-	43.2	-
	F-Siam	37.1	16.3	-	**47.0**	-
	P2B	56.2	28.7	40.8	32.1	42.4
	GPT	**59.1**	**35.2**	**49.6**	34.3	**47.4**
Precision	SC3D	57.9	37.8	47.0	70.4	48.5
	BEV-3D	51.0	47.8	-	**81.2**	-
	F-Siam	50.6	32.3	-	77.3	-
	P2B	72.8	49.6	48.4	44.7	60.0
	GPT	**75.6**	**63.6**	**60.6**	46.3	**68.4**

Table 1: 3D object tracking results. The numbers under types of targets are the number of frames. Note that F-Siam uses point clouds and RGB images both as input but other methods use point clouds only.

other methods. OPE measures the Success and Precision of the tracking results. Success is defined as the IoU between the tracking result and the GT bounding box. Precision is defined as the AUC for distance between the centers of tracking result and GT from 0 to 2 m.

Implementation Details

As in P2B (Qi et al. 2020), PointNet++ (Qi et al. 2017b) with three set abstraction (SA) layers is used as a backbone in GPT. In the three SA layers, the radii of their receptive fields are set to 0.3, 0.5, and 0.7 m, respectively. The size of the range query is set to 32, and the number of points is cut in half by random down-sampling in each SA layer. Thus, after applying three SA layers, the backbone outputs $M_1 = N_1/2^3 = 64$ template seeds and $M_2 = N_2/2^3 = 128$ search area seeds with $D_{seed} = 256$ feature dimension. In GFA, template and search area seeds are transformed to nodes with $D_{node} = D_{seed}/2 = 128$ feature dimension by ϕ and φ, respectively, and the bipartite graph $\mathcal{G}$ is constructed with $k = 16$. Moreover, the edge feature can be strengthened by the ITC feature with $D_{ITC} = 128$, and GPT generates the target-specific search area node $\tilde{v}_{S,j} \in \mathbb{R}^{3+\tilde{D}_{node}}$ with $\tilde{D}_{node} = 256$. $\tilde{v}_{S,j}$ is fed into regression (=voting) and classification branches and $L = 64$ clusters with the radius $R = 0.3$ m and 16 query points are generated. The clusters are then used as proposals. When we train our GPT, the relative weights of the loss are set to $\lambda_{\text{sea-reg}} = 0.9$, $\lambda_{\text{tmp-reg}} = 0.1$, $\lambda_{\text{cls}} = 0.2$, $\lambda_{\text{box}} = 0.2$, and $\lambda_{\text{prop}} = 1.5$. An Adam optimizer is used, the batch size is 48, and the learning rate is initially set to 0.001, which is reduced by a rate of 0.2 every 12 epochs.

Experimental Result

Our GPT is compared with Giancola, Zarzar, and Ghanem (Giancola, Zarzar, and Ghanem 2019) (SC3D), Zarzar, Giancola,and Ghanem (Zarzar, Giancola, and Ghanem 2020) (BEV-3D), F-Siamese tracker (Zou et al. 2020) (F-Siam) and P2B (Qi et al. 2020). SC3D, BEV-3D, and F-Siam are Siamese trackers, whereas P2B is a tracker that fuses the template and search area features by target-specific feature augmentation.

	f_S^{node}	f_T^{node}	$\bar{f}_T^{vote}$	Success	Precision
(i)	✓			58.1	73.7
(ii)		✓		58.9	75.3
(iii)			✓	**59.1**	**75.6**
(iv)	✓	✓		58.7	75.0
(v)	✓		✓	**59.1**	75.5
(vi)		✓	✓	58.9	75.4
(vii)	✓	✓	✓	58.8	75.2

Table 2: Target-Specific Search Area Node $\tilde{f}_{S,j}^{node}$.

	Success	Precision
GPT w/o ITC	58.9	75.3
GPT w/ ITC	**59.1**	**75.6**
P2B w/o ITC	56.2	72.8
P2B w/ ITC	57.9	73.5

Table 3: Analysis of improved target clue.

Tab. 1 presents the comparison results of our GPT with the previous methods. GPT outperforms all the previous methods. In case of rigid targets (=Car, Van), the Success and Precision rates of GPT are higher than those of the previous methods by more than 2.8 and 2.9, respectively. In case of deformable target (Pedestrian), Success and Precision rates of GPT are higher by more than 8.8 and 12.2, respectively. Only in Cyclist, not our GPT but Siamese trackers take the best. This might be that the number of training samples for Cyclist is much less than that of other targets.

Ablation Studies

Target-Specific Search Area Node $\tilde{f}_{S,j}^{node}$. We conduct further experiments to check the variation of the performance when we change the inputs to the target-specific search area node $\tilde{f}_{S,j}^{node}$ in Eq. 6. x_T and $f_T^{node} - f_S^{node}$ are used as default inputs in the edge feature, and $\bar{f}_T^{vote}$ from the ITC is replaced with other variables. The results are summarized in Tab. 2. OPE metrics using f_T^{node} are higher than those using f_S^{node}. This might be because the search area contains many more clutter points than template; thus template node f_T^{node} is more reliable than the search area node f_S^{node}. When we replace f_T^{node} with $\bar{f}_T^{vote}$ from ITC, the performance is improved further. This is because the template center provides additional information that was not available in template node.

Effectiveness of Improved Target Clue. To demonstrate the effectiveness of the ITC module, we applied the module not only to GPT but also to P2B. The results are summarized in Tab. 3. In the table, "w/o ITC" denotes the case in which either the template seed or node is used instead of the output from the ITC. In both methods, the ITC module improves the tracking performance. This implies that the information about target center obtained from ITC plays an important roles in localizing the target precisely in the search area.

Analysis of Feature Transform Function. We tested three different cases of the feature transformation functions ϕ and φ that are used on the front of the GFA: none, sharing weight ($\phi = \varphi$), and not sharing weight ($\phi \neq \varphi$). We also

		None	$\phi = \varphi$	$\phi \neq \varphi$
Success	w/o ITC	56.9	57.0	58.9
	w/ ITC	57.5	57.3	**59.1**
Precision	w/o ITC	73.0	72.6	75.3
	w/ ITC	73.3	72.8	**75.6**

Table 4: Analysis of feature transformation function.

		Training					
		Success			Precision		
	k	1	16	64	1	16	64
Inference	1	55.4	49.2	41.7	71.5	61.2	54.5
	4	55.2	54.5	45.4	71.3	68.9	58.6
	16	55.6	**59.1**	53.2	72.4	**75.6**	69.0
	32	52.9	57.8	56.0	68.9	73.7	71.8
	64	50.0	58.4	58.4	65.1	75.0	74.9

Table 5: Graph construction with various k.

tested two cases of using and not using ITC. The results are summarized in Tab. 4. When $\phi = \varphi$, tracking performance is not improved from "none" (=no transform); This implies that the feature transformation functions only reduce the feature dimensions. When $\phi \neq \varphi$, however, tracking performance is improved from "none" (=no transform); This implies that the feature transformation functions treat the template and search area separately in such a way that their difference is represented well in a bipartite graph $\mathcal{G}$.

Graph Construction with Various k. Some experiments are conducted to see the effect of k in GFA on the tracking performance. We trained and tested GPT while varying k from 1 to 64. As summarized in Tab. 5, GPT achieves the best Success and Precision when $k = 16$ in training and testing. Here, $k = 1$ is one-to-one matching; it is too local, and it does not capture one-to-many relationship. $k = 64$ is similar to one-to-all matching and is ineffective due to the redundant relations. One-to-many relationship with $k = 16$ in both training and testing outperforms all other settings.

Template Generation. We evaluated four different settings of template generation during testing, as presented in Tab. 6. We used first frame GT (First), previous result (Prev.), concatenation of first frame GT and previous result (F&P) and concatenation of all previous results (All) to generate a template for testing. "F&P" is the best in GPT and P2B, whereas "All" is the best in SC3D. Note that SC3D is trained with "All" whereas P2B and GPT is trained with "F&P" to reduce preprocessing time. This implies that the Siamese approach SC3D needs as much information as possible. GPT outperforms SC3D and P2B across all settings of template generation.

		First	Prev.	F & P	All
Success	SC3D	31.6	25.7	34.9	41.3
	P2B	46.7	53.1	56.2	51.4
	GPT	**52.6**	**56.7**	**59.1**	**57.0**
Precision	SC3D	44.4	35.1	49.8	57.9
	P2B	59.7	68.9	72.8	66.8
	GPT	**66.1**	**72.0**	**75.6**	**72.7**

Table 6: Ways for template generation.

		Prev. result	Prev. GT	Curr. GT
Success	SC3D	41.3	64.6	76.9
	P2B	56.2	82.4	84.0
	GPT	**59.1**	**82.8**	**84.4**
Precision	SC3D	57.9	74.5	81.4
	P2B	72.8	**90.1**	90.3
	GPT	**75.6**	**90.1**	**90.4**

Table 7: Ways for search area generation.

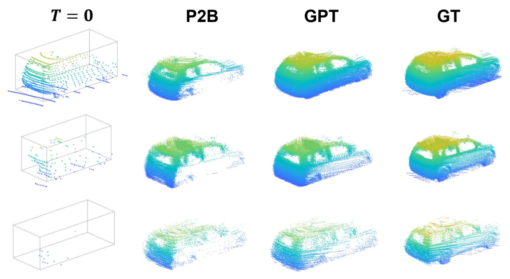

Figure 4: Qualitative results on Car. From left to right, the GT box in the first frame of each tracklet, GT model, P2B, and our GPT model generated by tracking results.

Search Area Generation. We evaluated three different settings of search area generation during testing. The results are summarized in Tab. 7. The three settings involve using previous result (Prev. result), previous GT (Prev. GT), and current GT (Curr. GT). The last two settings are unrealistic but are also evaluated, as in other papers. Compared to the state-of-the-art method P2B (Qi et al. 2020), GPT shows improved performance in all three settings. Specifically, GPT demonstrates higher improvement in Prev. GT than in the other unrealistic settings (Prev. GT and Curr. GT), possibly because the two unrealistic settings excessively simplify 3D target tracking problems.

Qualitative Results

For a qualitative comparison of the competing methods, we accumulated all the points within the tracking result (=box) over the entire sequence, as shown in Fig. 4. More results will be illustrated in supplementary material.

Conclusion

In this work, we present a graph-based point tracker (GPT) that builds reliable and efficient one-to-many relationship via a bipartite graph embedding. Our GPT achieves state-of-the-art performance by exploiting two key modules, GFA and ITC. Experimental results demonstrate the effectiveness of our proposed modules from various perspectives. We believe that our approach can be extended to more relation-based applications such as registration, moving object segmentation and sensor fusion.

Acknowledgements

This work was supported in part by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (NRF-2019R1A2C1007153). This work also supported in part by the Industry Core Technology Development Project, 20005062, Development of Artificial Intelligence Robot Autonomous Navigation Technology for Agile Movement in Crowded Space, funded by the Ministry of Trade, industry Energy (MOTIE, Republic of Korea).

References

Aoki, Y.; Goforth, H.; Srivatsan, R. A.; and Lucey, S. 2019. PointNetLK: Robust & Efficient Point Cloud Registration Using PointNet. In *CVPR*.

Bertinetto, L.; Valmadre, J.; Henriques, J. F.; Vedaldi, A.; and Torr, P. H. S. 2016. Fully-Convolutional Siamese Networks for Object Tracking. In *ECCVW*, 850–865.

Bibi, A.; Zhang, T.; and Ghanem, B. 2016. 3D Part-Based Sparse Tracker With Automatic Synchronization and Registration. In *CVPR*.

Chen, T.; Cheng, Y.; Gan, Z.; Wang, J.; Wang, L.; Wang, Z.; and Liu, J. 2021. Adversarial Feature Augmentation and Normalization for Visual Recognition. arXiv:2103.12171.

Chen, Z.; Fu, Y.; Zhang, Y.; Jiang, Y.-G.; Xue, X.; and Sigal, L. 2019. Multi-Level Semantic Feature Augmentation for One-Shot Learning. *IEEE Transactions on Image Processing*, 28(9): 4594–4605.

Dixit, M.; Kwitt, R.; Niethammer, M.; and Vasconcelos, N. 2017. AGA: Attribute-Guided Augmentation. In *CVPR*.

Geiger, A.; Lenz, P.; and Urtasun, R. 2012. Are we ready for autonomous driving? The KITTI vision benchmark suite. In *CVPR*, 3354–3361.

Giancola, S.; Zarzar, J.; and Ghanem, B. 2019. Leveraging Shape Completion for 3D Siamese Tracking. In *CVPR*.

Guo, Q.; Feng, W.; Zhou, C.; Huang, R.; Wan, L.; and Wang, S. 2017. Learning Dynamic Siamese Network for Visual Object Tracking. In *ICCV*.

He, A.; Luo, C.; Tian, X.; and Zeng, W. 2018. A Twofold Siamese Network for Real-Time Object Tracking. In *CVPR*.

Held, D.; Levinson, J.; Thrun, S.; and Savarese, S. 2016. Robust real-time tracking combining 3D shape, color, and motion. *IJRR*, 35(1-3): 30–49.

Kart, U.; Kamarainen, J.-K.; and Matas, J. 2018. How to Make an RGBD Tracker ? In *ECCVW*.

Kart, U.; Lukezic, A.; Kristan, M.; Kamarainen, J.-K.; and Matas, J. 2019. Object Tracking by Reconstruction With View-Specific Discriminative Correlation Filters. In *CVPR*.

Li, B.; Yan, J.; Wu, W.; Zhu, Z.; and Hu, X. 2018a. High Performance Visual Tracking With Siamese Region Proposal Network. In *CVPR*.

Li, Y.; Bu, R.; Sun, M.; Wu, W.; Di, X.; and Chen, B. 2018b. PointCNN: Convolution On X-Transformed Points. In *NIPS*, volume 31. Curran Associates, Inc.

Maturana, D.; and Scherer, S. 2015. VoxNet: A 3D Convolutional Neural Network for real-time object recognition. In *IROS*, 922–928.

Pan, X.; Xia, Z.; Song, S.; Li, L. E.; and Huang, G. 2021. 3D Object Detection With Pointformer. In *CVPR*, 7463–7472.

Qi, C. R.; Litany, O.; He, K.; and Guibas, L. J. 2019. Deep Hough Voting for 3D Object Detection in Point Clouds. In *ICCV*.

Qi, C. R.; Su, H.; Mo, K.; and Guibas, L. J. 2017a. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *CVPR*.

Qi, C. R.; Yi, L.; Su, H.; and Guibas, L. J. 2017b. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. In *NIPS*, volume 30.

Qi, H.; Feng, C.; Cao, Z.; Zhao, F.; and Xiao, Y. 2020. P2B: Point-to-Box Network for 3D Object Tracking in Point Clouds. In *CVPR*.

Qiu, S.; Anwar, S.; and Barnes, N. 2021. Semantic Segmentation for Real Point Cloud Scenes via Bilateral Augmentation and Adaptive Fusion. In *CVPR*, 1757–1767.

Shi, S.; Guo, C.; Jiang, L.; Wang, Z.; Shi, J.; Wang, X.; and Li, H. 2020. PV-RCNN: Point-Voxel Feature Set Abstraction for 3D Object Detection. In *CVPR*.

Shi, S.; Wang, X.; and Li, H. 2019. PointRCNN: 3D Object Proposal Generation and Detection From Point Cloud. In *CVPR*.

Song, S.; and Xiao, J. 2013. Tracking Revisited Using RGBD Camera: Unified Benchmark and Baselines. In *ICCV*.

Su, H.; Maji, S.; Kalogerakis, E.; and Learned-Miller, E. 2015. Multi-View Convolutional Neural Networks for 3D Shape Recognition. In *ICCV*.

Thomas, H.; Qi, C. R.; Deschaud, J.-E.; Marcotegui, B.; Goulette, F.; and Guibas, L. J. 2019. KPConv: Flexible and Deformable Convolution for Point Clouds. In *ICCV*.

Volpi, R.; Morerio, P.; Savarese, S.; and Murino, V. 2018. Adversarial Feature Augmentation for Unsupervised Domain Adaptation. In *CVPR*.

Wang, Q.; Teng, Z.; Xing, J.; Gao, J.; Hu, W.; and Maybank, S. 2018. Learning Attentions: Residual Attentional Siamese Network for High Performance Online Visual Tracking. In *CVPR*.

Wang, Y.; and Solomon, J. M. 2019. Deep Closest Point: Learning Representations for Point Cloud Registration. In *ICCV*.

Wang, Y.; Sun, Y.; Liu, Z.; Sarma, S. E.; Bronstein, M. M.; and Solomon, J. M. 2019. Dynamic Graph CNN for Learning on Point Clouds. *ACM Trans. Graph.*, 38(5).

Wu, Y.; Lim, J.; and Yang, M.-H. 2013. Online Object Tracking: A Benchmark. In *CVPR*.

Xiao, J.; Stolkin, R.; Gao, Y.; and Leonardis, A. 2018. Robust Fusion of Color and Depth Data for RGB-D Target Tracking Using Adaptive Range-Invariant Depth Models and Spatio-Temporal Consistency Constraints. *IEEE Transactions on Cybernetics*, 48(8): 2485–2499.

Xie, S.; Liu, S.; Chen, Z.; and Tu, Z. 2018. Attentional ShapeContextNet for Point Cloud Recognition. In *CVPR*.

Xu, Y.; Fan, T.; Xu, M.; Zeng, L.; and Qiao, Y. 2018. SpiderCNN: Deep Learning on Point Sets with Parameterized Convolutional Filters. In *ECCV*.

Yan, X.; Zheng, C.; Li, Z.; Wang, S.; and Cui, S. 2020. PointASNL: Robust Point Clouds Processing Using Nonlocal Neural Networks With Adaptive Sampling. In *CVPR*.

Yang, J.; Zhang, Q.; Ni, B.; Li, L.; Liu, J.; Zhou, M.; and Tian, Q. 2019. Modeling Point Clouds With Self-Attention and Gumbel Subset Sampling. In *CVPR*.

Zarzar, J.; Giancola, S.; and Ghanem, B. 2020. Efficient Bird Eye View Proposals for 3D Siamese Tracking. arXiv:1903.10168.

Zhang, Y.; Sun, B.; Xiao, Y.; Xiao, R.; and Wei, Y. 2019. Feature augmentation for imbalanced classification with conditional mixture WGANs. *Signal Processing: Image Communication*, 75: 89–99.

Zhao, H.; Jiang, L.; Fu, C.-W.; and Jia, J. 2019. PointWeb: Enhancing Local Neighborhood Features for Point Cloud Processing. In *CVPR*.

Zou, H.; Cui, J.; Kong, X.; Zhang, C.; Liu, Y.; Wen, F.; and Li, W. 2020. F-Siamese Tracker: A Frustum-based Double Siamese Network for 3D Single Object Tracking. In *IROS*, 8133–8139.