Paved2Paradise: Cost-Effective and Scalable LiDAR Simulation by Factoring the Real World

Michael A. Alcorn[1*] Noah Schwartz[1,2]
[1]Bear Flag Robotics
[2] John Deere

Abstract

To achieve strong real world performance, neural networks must be trained on large, diverse datasets; however, obtaining and annotating such datasets is costly and time-consuming, particularly for 3D point clouds. In this paper, we describe Paved2Paradise[1], a simple, cost-effective approach for generating fully labeled, diverse, and realistic lidar datasets from scratch—all while requiring minimal human annotation. Our key insight is that, by deliberately collecting separate "background" and "object" datasets (i.e., "factoring the real world"), we can intelligently combine them to produce a combinatorially large and diverse training set. The Paved2Paradise pipeline thus consists of four steps: (1) collecting copious background data, (2) recording individuals from the desired object class(es) performing different behaviors in an isolated environment (like a parking lot), (3) bootstrapping labels for the object dataset, and (4) generating samples by placing objects at arbitrary locations in backgrounds. To demonstrate the utility of Paved2Paradise, we generated synthetic datasets for two tasks: (1) human detection in orchards[2] (a task for which no public data exists) and (2) pedestrian detection in urban environments. Qualitatively, we find that a model trained exclusively on Paved2Paradise synthetic data is highly effective at detecting humans in orchards, including when individuals are heavily occluded by tree branches. Quantitatively, a model trained on Paved2Paradise data that sources backgrounds from KITTI performs comparably to a model trained on the actual dataset. These results suggest the Paved2Paradise synthetic data pipeline can help accelerate point cloud model development in sectors where acquiring lidar datasets has previously been cost-prohibitive.

*Corresponding author: michael@bearflagrobotics.com.

[1]Code for the Paved2Paradise pipeline can be found at: https://github.com/airalcorn2/paved2paradise .

[2]The name "Paved2Paradise" refers to the fact that we move human point clouds from parking lot scenes (i.e., paved environments) to orchard backgrounds (i.e., "paradises" like the Garden of Eden) for this task.

1. Introduction and Related Work

To achieve strong real world performance, neural networks must be trained on large, diverse datasets; however, obtaining and annotating such datasets is costly and time-consuming. For 3D object detection and semantic segmentation on point clouds, the annotation process is particularly slow due to the inherent challenges of working in 3D environments (e.g., confusing perspectives) [2]. As a result, most lidar benchmark datasets are quite small by modern deep learning standards, particularly when compared to image datasets like ImageNet [5] and COCO [20]. The KITTI [9, 10], SemanticKITTI [1], and nuScenes [4] datasets contain 7,481, 23,201, and 40,157 annotated frames, respectively. Further, the frames for each of these datasets were collected in short sequences, i.e., the frames in a sequence are highly correlated, and thus the environmental variability reflected in each dataset is considerably lower than what might be suggested by the total frame count. While the Waymo Open Dataset [34] is much larger in comparison (with 230,000 annotated frames), the dataset has a restrictive license.

An additional challenge with lidar datasets is that—unlike image datasets, where models generally transfer well when applied to images captured on different cameras—point cloud models perform poorly when applied to data captured with different sensors [31, 42]. Lastly, importantly, *all* of the aforementioned lidar datasets are focused on autonomous driving, so their utility in other sectors (e.g., agriculture, construction, and mining) is extremely limited. A method making it possible to acquire large, labeled lidar datasets in a cost-effective way would thus be valuable in many industries.

To address the limited sample sizes and sample diversity in existing lidar datasets, a number of techniques incorporating data augmentations and/or computer-generated data have been proposed. Many of the data augmentation approaches involve some variation of "mixing" scenes, whether that be "cutting" objects from one scene and "pasting" them into another scene [14, 17, 30, 39], or combin-

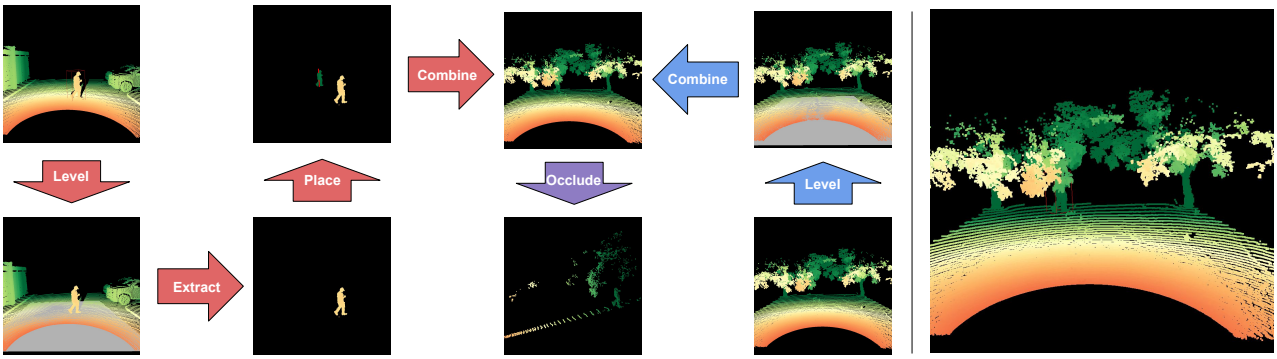

Figure 1. An overview of the Paved2Paradise pipeline for generating synthetic scenes (left) and an example synthetic scene (right). **Left**: The pipeline begins by randomly sampling an object scene (top left) and a background scene (bottom right) and "leveling" both scenes (Section 3.4). The leveling step ensures the object point cloud extracted from the object scene will be placed on the ground in the background scene. Next, the object point cloud is extracted from the object scene and placed at a randomly sampled location in the detection region (Section 3.5). During the placement step, the object point cloud is rotated to ensure perspective consistency relative to the sensor's location. The repositioned object point cloud is then combined with the background point cloud (Section 3.6), and a final occluding procedure removes points from the scene based on their visibility from the sensor's perspective (Section 3.7). **Right**: The resulting synthetic scene is both realistic and automatically annotated.

ing entire scenes [26, 30]. While these approaches can produce novel *combinations* of input signals, the fundamental *diversity* of the input signals (e.g., the number of distinct street scenes in an autonomous driving dataset) is still limited in small datasets. To make this point clear, consider the Waymo Open Dataset, which features data from San Francisco, Phoenix, and Mountain View. No amount of mixing in this dataset would allow a model to encounter a peacock crossing the road during training.[3] Likewise, the diversity and quality of computer-generated scenes are a function of the 3D assets and graphics engines made available through the rendering pipelines (e.g., [6, 7, 15, 18, 32, 38, 41]), which makes them unsuitable for many applications. Importantly, *all of these approaches rely on previously labeled data*, whether those labels are provided by human annotators or are automatically available via a computer-generated environment.

In this paper, we introduce Paved2Paradise (**P2P**; Figure 1), a simple, cost-effective approach for assembling large, diverse, realistic, and fully annotated lidar datasets *from scratch*—all while requiring minimal human annotation. Our key insight is that, by deliberately collecting *separate* "background" and "object" datasets (i.e., "factoring the real world"), we can intelligently combine them to produce a combinatorially large and diverse training set. To demonstrate the efficacy of our approach, we used **P2P** to generate a dataset for a challenging task in which public data does not exist: human detection in orchards. A PointPillars [17] model trained exclusively on the **P2P** synthetic dataset is highly effective at detecting humans in orchards, including when individuals are occluded by tree branches. We further performed a quantitative analysis of **P2P** by generating a pedestrian detection dataset that sources pedestrian-less backgrounds from KITTI, and we find that a **P2P**-trained model performs comparably to a model trained on the full KITTI dataset.

2. Comparison to LiDARsim

LiDARsim [22] is conceptually similar to **P2P** in motivation, but the two pipelines differ considerably in execution with **P2P** being both simpler and more flexible than LiDARsim. LiDARsim builds a catalog of surfel [28] meshes consisting of "static maps" and "dynamic objects" that serve as assets for creating synthetic scenes as in a graphics engine. To create a static map for a specific area, LiDARsim requires (on average) three lidar passes over the area. The lidar scans are then fused with wheel-odometry, IMU, and GPS measurements and converted into a single merged point cloud using GraphSLAM [36], i.e., the merging step is rather complex and difficult to use.[4] Finally, the merged point cloud is converted into a surfel mesh, which also involves parameter tuning to achieve optimal results.

For their dynamic objects, Manivasagam et al. [22] focused exclusively on vehicles (non-rigid objects—like humans—were explicitly omitted), which is reflected in the LiDARsim asset creation process. Specifically, to create a dynamic object, LiDARsim first accumulates all of the points found in an object's bounding box over a short time sequence and then mirrors the points over the object's heading (which exploits the bilateral symmetry commonly found

[3] A common occurrence on the streets of Miami: https://www.nbcmiami.com/news/local/peacocks-in-miami-should-they-stay-or-should-they-go/2702779/.

[4] The hdl_graph_slam ROS package (https://github.com/koide3/hdl_graph_slam) has 116 configurable parameters and notes that "mapping quality largely depends on the parameter setting".

in vehicles). The point cloud is then refined with an iterative closest point algorithm [27] (using lidar intensity as the color value) and finally converted into a surfel mesh. Importantly, dynamic object creation in LiDARsim depends on lidar scenes that have *already been annotated* with bounding boxes, which are unavailable in many applications and are costly and time-consuming to obtain. Further, the diversity of behaviors reflected in the dynamic object dataset are left to chance, i.e., many behaviors are likely to be missing.

In contrast to LiDARsim, **P2P** leverages the raw lidar scans directly, which means: (1) new background data is immediately available for synthetic scene creation (i.e., without requiring additional preprocessing or access to auxiliary inputs) and (2) the approach works equally well with rigid and non-rigid objects. Further, **P2P** only requires a single scan per static map, so **P2P** can collect at least three times as many static maps as LiDARsim for a given data collection time budget. Lastly, because dynamic object data collection is a separate and deliberate process in **P2P**, the diversity of object behaviors is directly under the researcher's control instead of being left to serendipity.

3. Paved2Paradise

> ♫ Don't it always seem to go
> That the data that you want would take too long?
> With Paved2Paradise
> You only need people and a parking lot ♫
>
> "Little Red MULE" - *JonAI Mitchell*[5]

To motivate the **P2P** pipeline, note that the "ideal" 3D object detection dataset (i.e., one that reflects the diversity of scenarios a model may experience in the wild) will vary across four factors: (1) **instances** of the object class(es), (2) **poses** each object can take on, (3) **locations** where an object can be found in a scene relative to a lidar sensor, and (4) environmental **backgrounds** where the objects can be seen. Clearly, directly acquiring and annotating a dataset that fully reflects this combinatorial diversity is impractical. Our key insights are that: (1) **instance** and **background** datasets can be collected *separately* with the **background** dataset collected passively, (2) the **location** factor can be entirely simulated, which means (3) **poses** only need to be collected from a single location with respect to the lidar sensor. The full **P2P** pipeline thus consists of the following four steps:

1. Collecting copious background data that reflects the variety of environmental conditions a model will encounter in the wild (Section 3.1).
2. Recording individuals from the desired object class(es) performing different behaviors at an isolated, flat location (Section 3.2).
3. Bootstrapping labels for the object dataset (Section 3.3).
4. Generating training samples by combining randomly sampled objects with randomly sampled backgrounds in such a way that the synthetic scenes are consistent with the sensor's perspective (Sections 3.4, 3.5, 3.6, and 3.7).

[5] If you find these lyrics funny, they were written by the authors. If not, they were written by ChatGPT.

3.1. Collecting the background scenes

The goal of the background collection step of **P2P** is to capture as much environmental data as possible. The only requirement for this step is that the collected scenes do *not* contain any objects that the model will be trained to detect. Because these environmental captures will serve as the backgrounds into which objects will be placed, any scenes that contain *unlabeled* objects are in fact incorrectly labeled and will hurt model performance.

In many cases, the backgrounds can be collected passively because (presumably) the task that needs to be automated has not yet been automated. As a result, data can be recorded during standard vehicle/equipment operation to produce a large and diverse background dataset that spans environmental conditions—the operator simply needs to diligently note moments when an object from a detection category is observed so that the surrounding frames can be removed from the background pool. In the case of pedestrian detection for autonomous driving, background data could be collected by driving in neighborhoods when they experience their lowest pedestrian traffic.

3.2. Collecting the object scenes

Note that, if given a mesh of a 3D object, simulating the point cloud for that object from the perspective of a lidar sensor is trivial using ray casting (ignoring occlusion and sensor effects for the time being). Indeed, this is an effective approach as demonstrated by LiDAR-Aug [8] and LiDARsim [22], and can be taken to the extreme by assembling datasets that are *entirely* computer-generated (e.g., [6, 15, 32, 38]). But, for non-rigid objects, simple meshes are insufficient because their real-life analogs undergo both rigid *and* non-rigid transformations. Using high-quality "rigged" meshes that can be animated is one potential solution to the non-rigidity issue. However, such meshes are generally expensive to acquire. Further, animating each separate mesh (of which there should be many to reflect the natural diversity of the object class) requires employing/contracting 3D animators (although deep learning may be able to automate animation in the future [24]).

The goal of the object collection step in **P2P** is to match the flexibility and diversity of this hypothetical animation library in a cost-effective and time-efficient way. To motivate our approach, we first describe a concrete example of a naive strategy for collecting the object dataset. Imagine that a researcher would like to be able to detect humans in a 200×200 square grid located directly in front of a lidar sen-

sor. Further, assume that these particular humans come in 200 different body types and can take on one of 200 different poses. There are thus $200^4 = 1.6$ *billion* unique human point clouds (in terms of the point coordinates) that a model could encounter for this particular task (again, ignoring any sensor effects).

In theory, recording an individual of each body type performing each pose in each grid cell at the collection location would provide the researcher with a full object dataset that, following annotation, could be mixed with background data. Of course, collecting and annotating 1.6 billion samples is prohibitively costly and/or time-consuming. **P2P** uses a simpler, far less time-intensive approach—recording individuals performing a variety of natural behaviors at a *single* location relative to the sensor[6] in an isolated, flat setting (such as an empty area of a parking lot). This strategy immediately reduces the data collection workload by a factor of 40,000, i.e., only 40,000 samples (the scale of the SemanticKITTI dataset) would need to be labeled in this hypothetical example.

3.3. Bootstrapping labels for the object scenes

Because the **P2P** object scenes are so simple and consistent, a PointNet++ [29] regression model can be trained to accurately predict bounding box attributes from a relatively small number of human-labeled samples. Specifically, a sample is created by (1) extracting only those points in the performance area that are *above the ground plane* and (2) centering the extracted points based on the median coordinate values. Once the PointNet++ model reaches an acceptable level of accuracy, the remaining samples can be automatically labeled.

3.4. Leveling the scenes

At this point, **P2P** has a collection of M background point clouds $\mathcal{B} = \{\mathbf{B}_m\}_{m=1}^{M}$ where $\mathbf{B}_m = \{\mathbf{b}_{m,p}\}_{p=1}^{P_m}$ is a set of P_m points with $\mathbf{b}_{m,p} = (x_{\mathbf{b}_{m,p}}, y_{\mathbf{b}_{m,p}}, z_{\mathbf{b}_{m,p}})$. Likewise, **P2P** has a collection of N object *scene*[7] point clouds $\mathcal{O} = \{\mathbf{O}_n\}_{n=1}^{N}$ where $\mathbf{O}_n = \{\mathbf{o}_{n,q}\}_{n=1}^{Q_n}$ is a set of Q_n points with $\mathbf{o}_{n,q} = (x_{\mathbf{o}_{n,q}}, y_{\mathbf{o}_{n,q}}, z_{\mathbf{o}_{n,q}})$. Further, each object scene has an associated bounding box for the object defined by its center $\mathbf{c}_n = (x_{\mathbf{c}_n}, y_{\mathbf{c}_n}, z_{\mathbf{c}_n})$, extent $\mathbf{d}_n = (x_{\mathbf{d}_n}, y_{\mathbf{d}_n}, z_{\mathbf{d}_n})$, and rotation matrix $\mathbf{R}_n$. For both collections of point clouds, the points are in the sensor's coordinate frame, i.e., the sensor is located at the origin with the x-axis pointing forward out of the sensor, the y-axis pointing to the sensor's left, and the z-axis pointing up.

Because the sensors used to obtain the background and object scenes may not be perfectly level with their respective ground planes nor the same heights off the ground, naively manipulating and combining the point clouds can produce unrealistic scenes (Figure 2). Therefore, it is necessary to "level" the point clouds before proceeding (Figure 3). The leveling procedure begins by setting up a grid of $G \times G$ points in a predetermined region $x \in [x_{\text{min}}, x_{\text{max}}]$ and $y \in [-y_{\text{max}}, y_{\text{max}}]$ where x_{min}, x_{max}, and y_{max} are provided by the user, and the z value for each grid point is the minimum z value of the scene points in the region. Next, the "ground points" are defined as the scene points in the grid region that are nearest neighbors with the grid points.

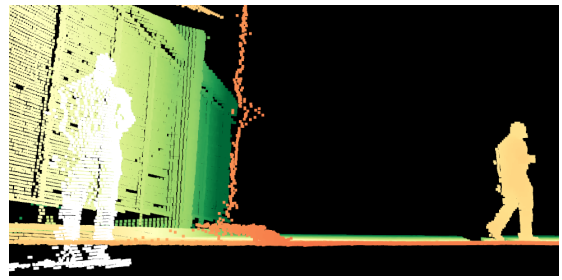

Figure 2. Because this sensor's xy-plane was not level with the ground plane, naively rotating the human point cloud around the sensor's z-axis results in the human's feet being underground.

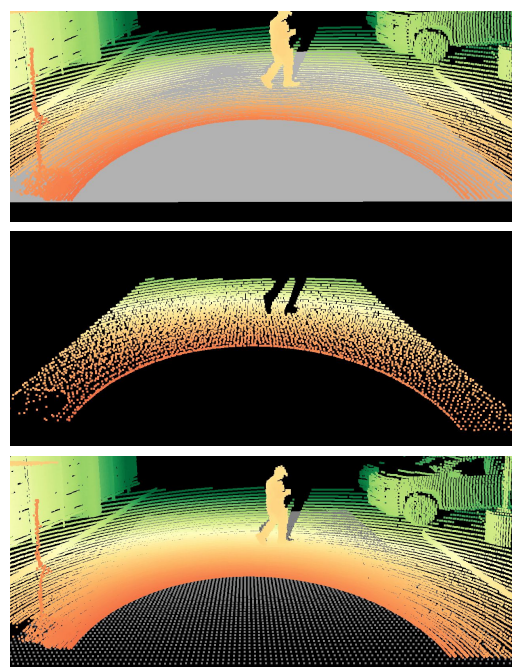

Figure 3. The Paved2Paradise leveling procedure estimates the ground plane for a scene (top) by performing linear regression on ground points (middle) obtained by finding the nearest neighbors to a set of grid points beneath the scene (bottom).

A linear regression model of the form $z = b_0 + b_1 x + b_2 y$ is then fit to the ground points to estimate the ground plane. The normal vector $\mathbf{h}_{\mathbb{1}}$ for the estimated ground plane is thus $\mathbf{h}_{\mathbb{1}} = \frac{\mathbf{h}}{\|\mathbf{h}\|}$ where $\mathbf{h} = [-b_1, -b_2, 1]$. The rotation matrix

[6] Specifically, as close to the sensor as possible while maintaining full visibility of the human.

[7] Note that we are explicitly distinguishing an object *scene* point cloud from an the isolated object point cloud.

$\mathbf{R}_{\{\mathbf{O},\mathbf{B}\}}$ for leveling the estimated ground plane can be obtained using Rodrigues' rotation formula, i.e., $\mathbf{R}_{\{\mathbf{O},\mathbf{B}\}} = \mathbf{I} + [\mathbf{v}]_\times + [\mathbf{v}]_\times^2 \frac{1}{1+\mathbf{h}[3]}$ where:

$$[\mathbf{v}]_\times \stackrel{\text{def}}{=} \begin{bmatrix} 0 & -v_3 & v_2 \\ v_3 & 0 & -v_1 \\ -v_2 & v_1 & 0 \end{bmatrix}$$

and $\mathbf{v} = \mathbf{h}_{\mathbb{1}} \times [0, 0, 1]$.[8] **P2P** thus levels an object scene point cloud $\mathbf{O}$[9] such that its estimated ground plane is aligned with the xy-plane at $z = 0$ using the following transformation:

$$\widetilde{\mathbf{O}} = \mathbf{R}_{\mathbf{O}}\mathbf{O} - \mathbf{t}_{\mathbf{O}} \tag{1}$$

where $\mathbf{R}_{\mathbf{O}}$ is the leveling rotation matrix for the object scene and $\mathbf{t}_{\mathbf{O}} = [0, 0, b_0]$. Using the same procedure, **P2P** can convert a background point cloud $\mathbf{B}$ into a leveled background $\widetilde{\mathbf{B}}$.

3.5. Placing an object

While a large dataset can be assembled by combining unmodified object point clouds with background point clouds, the true diversity of the dataset will be relatively low because the object point clouds will always be found in the same locations. If an object point cloud could instead be placed at an *arbitrary* location in a background point cloud, the diversity of the synthetic dataset would dramatically increase. However, due to the sensor's perspective of the scene, placing an object point cloud at an arbitrary location in a background point cloud is not as simple as translating the object point cloud to the new location.

For example, consider a scene where a cone has been placed directly in front of the sensor with the base oriented perpendicular to the sensor's x-axis (Figure 4). In this case, the sensor will see a disk of points. If the cone was then physically placed—without rotating the object—directly to the left of the sensor, the sensor would see a curved triangle of points. However, naively translating the disk of points from the original scene to the side of the sensor would result in an impossible point cloud from the sensor's perspective. Not only is the side of the cone missing, but a side view of a disk should produce a line of points, i.e., the new point cloud is wrong in two different ways.

A straightforward way to enforce perspective-consistency is by restricting object movements to (1) translations along the direction defined by the object's center and (2) rotations around the z-axis of the sensor. Therefore, given an arbitrary location on the leveled ground plane $\rho = (x_\rho, y_\rho, 0)$, the extracted and repositioned object point cloud $\widetilde{\mathbf{O}}_\rho$ is obtained with:

[8] See: https://en.wikipedia.org/wiki/Rodrigues%27_rotation_formula#Matrix_notation and: https://math.stackexchange.com/a/476311/614328.

[9] We omit the subscripts here for succinctness.

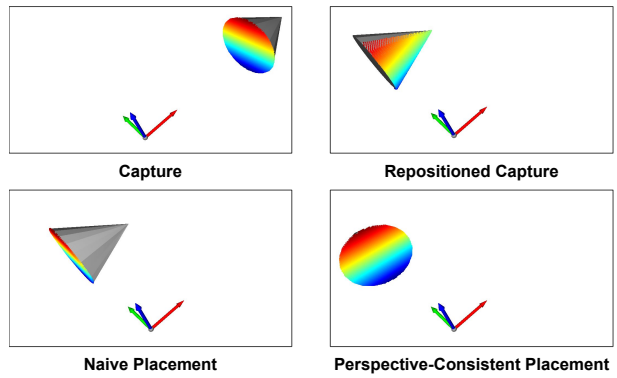

Figure 4. Naively placing a point cloud at an arbitrary location in a scene can produce a synthetic lidar scan that is physically impossible. **Top Left**: In the original scene, the cone is directly in front of the sensor (indicated by the axes) with the cone's base perpendicular to the sensor's x-axis. **Top Right**: Physically moving the cone to the left of the sensor (without rotating the object) results in an entirely different lidar scan. **Bottom Left**: However, naively translating the point cloud from the original scene to the left of the sensor leads to an impossible lidar scan. **Bottom Right**: Rotating the translated point cloud produces a perspective-consistent synthetic lidar scan.

$$\widetilde{\mathbf{O}}_\rho = \mathbf{R}_\rho(\psi(\widetilde{\mathbf{O}}, \widetilde{\mathbf{c}}, \widetilde{\mathbf{d}}, \widetilde{\mathbf{R}}) + \mathbf{t}_\rho) \tag{2}$$

where $\mathbf{R}_\rho = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}$, $\theta = \text{atan2}(y_\rho, x_\rho) - \text{atan2}(y_{\tilde{\mathbf{c}}}, x_{\tilde{\mathbf{c}}})$, ψ is a cropping function, $\widetilde{\mathbf{c}}$, $\widetilde{\mathbf{d}}$, and $\widetilde{\mathbf{R}}$ are the leveled bounding box parameters, $\mathbf{t}_\rho = \|\rho\| \frac{\tilde{\mathbf{c}}_0}{\|\tilde{\mathbf{c}}_0\|} - \tilde{\mathbf{c}}_0$, and $\tilde{\mathbf{c}}_0 = (x_{\tilde{\mathbf{c}}}, y_{\tilde{\mathbf{c}}}, 0)$.

3.6. Combining an object with a background

At this point, the repositioned object can be placed into a leveled background by combining the two sets of points, i.e., $\widetilde{\mathbf{S}} = \widetilde{\mathbf{B}} \cup \widetilde{\mathbf{O}}_\rho$. The leveled synthetic composite scene $\widetilde{\mathbf{S}}$ can then be transformed back into the original background coordinate frame with $\mathbf{R}_{\mathbf{B}}^\top(\widetilde{\mathbf{S}} + \mathbf{t}_{\mathbf{B}}) = \mathbf{S}_u$. While these steps conceptually describe how **P2P** combines an object with a background, they are inefficient because the background points are simply returned to their original positions. As a result, in practice, the background inverse transformation is only applied to $\widetilde{\mathbf{O}}_\rho$, i.e.:

$$\widehat{\mathbf{O}} = \mathbf{R}_{\mathbf{B}}^\top(\widetilde{\mathbf{O}}_\rho + \mathbf{t}_{\mathbf{B}}) \tag{3}$$

The points in $\widehat{\mathbf{O}}$ are then re-indexed such that $i = \text{rank}(q, \hat{\mathbf{q}})$ where $\hat{\mathbf{q}} = \{q \in \{1 \ldots Q\} : \hat{\mathbf{o}}_q \in \widehat{\mathbf{O}}\}$ and $\text{rank}(q, \hat{\mathbf{q}})$ returns the ranking of q in the set of indexes $\hat{\mathbf{q}}$. The unoccluded composite scene $\mathbf{S}_u$ is thus:

$$\mathbf{S}_u = \mathbf{B} \cup \widehat{\mathbf{O}} \tag{4}$$

3.7. Simulating the final LiDAR scan

To make the final point cloud appear more realistic, **P2P** simulates occlusion and sensor effects. To simulate oc-

clusion, **P2P** uses an *approximate* ray intersection algorithm (similar to Schaufler and Jensen [33]), which occludes points in both the object point cloud and the background point cloud to create an occluded point cloud scene. Because an object is only occluded when lidar beams in its sector are blocked, the occlusion algorithm begins by extracting a subset of background points $\mathbf{B}_\alpha \subset \mathbf{B}$ that are in the same sector as $\widehat{\mathbf{O}}$, i.e.:

$$\mathbf{B}_\alpha = \{\mathbf{b}_p \in \mathbf{B} : \min(\alpha(\widehat{\mathbf{O}})) - \epsilon < \alpha(\mathbf{b}_p) < \max(\alpha(\widehat{\mathbf{O}})) + \epsilon\}$$

where α is a function that returns the azimuth of a point, $\alpha(\widehat{\mathbf{O}}) = \{\alpha(\hat{\mathbf{o}}_i)\}_{i=1}^{|\widehat{\mathbf{O}}|}$, and ϵ is a user-defined tolerance factor. Further, because an object can only be occluded by a second object if the second object is *closer* to the sensor than the object, the occlusion algorithm extracts two additional subsets of points from $\mathbf{B}_\alpha$:

$$\mathbf{B}_\beta = \{\mathbf{b}_p \in \mathbf{B}_\alpha : \|\mathbf{b}_p\| \leq \max(\|\widehat{\mathbf{O}}\|)\}$$
$$\mathbf{B}_\gamma = \{\mathbf{b}_p \in \mathbf{B}_\alpha : \|\mathbf{b}_p\| \geq \min(\|\widehat{\mathbf{O}}\|)\}$$

where $\|\widehat{\mathbf{O}}\| = \{\|\hat{\mathbf{o}}_i\|\}_{i=1}^{|\widehat{\mathbf{O}}|}$. $\mathbf{B}_\beta$ will be used to occlude points in $\widehat{\mathbf{O}}$, while $\widehat{\mathbf{O}}$ will be used to occlude points in $\mathbf{B}_\gamma$.

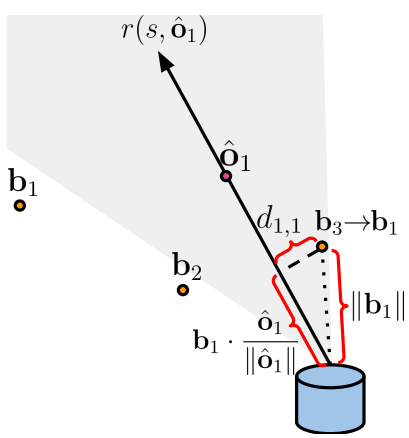

Figure 5. The Paved2Paradise object occluding procedure. Given a set of object points $\widehat{\mathbf{O}}$ (here, $\widehat{\mathbf{O}}$ contains a single point, i.e., $\widehat{\mathbf{O}} = \{\hat{\mathbf{o}}_1\}$), Paved2Paradise extracts a subset of (reindexed; see Section 3.7) background points $\mathbf{B}_\beta \subset \mathbf{B}$ such that each $\mathbf{b}_j \in \mathbf{B}_\beta$ is *both* in the same sector as $\widehat{\mathbf{O}}$ (delineated by the gray cone) *and* closer to the sensor (the blue cylinder) than at least one point in $\widehat{\mathbf{O}}$. Next, for each object point $\hat{\mathbf{o}}_i$, Paved2Paradise calculates the minimum distances $d_{i,j}$ between the points in $\mathbf{B}_\beta$ and the ray $r(s, \hat{\mathbf{o}}_i) = s\frac{\hat{\mathbf{o}}_i}{\|\hat{\mathbf{o}}_i\|}$. Because the closest point along $r(s, \hat{\mathbf{o}}_i)$ to $\mathbf{b}_j$ is the projection of $\mathbf{b}_j$ onto the ray, i.e., $(\mathbf{b}_j \cdot \frac{\hat{\mathbf{o}}_i}{\|\hat{\mathbf{o}}_i\|})\frac{\hat{\mathbf{o}}_i}{\|\hat{\mathbf{o}}_i\|}$, each $d_{i,j}$ can be calculated using the Pythagorean theorem. Finally, if any of the $d_{i,j}$ values are less than some user-defined threshold, then $\hat{\mathbf{o}}_i$ is considered "occluded" and dropped from the final point cloud for the scene.

We now describe the **P2P** approximate ray intersection function ϕ, which has a signature of $\mathbf{D}_\phi = \phi(\mathbf{D}, \mathbf{E}, F)$ where $\mathbf{E}$ is a point cloud that will be used to occlude points in another point cloud $\mathbf{D}$ and F is a user-defined threshold. Here, we provide details for the case of $\widehat{\mathbf{O}}_\phi = \phi(\widehat{\mathbf{O}}, \mathbf{B}_\beta, F_o)$ (Figure 5), but the procedure is identical for $\mathbf{B}_\phi = \phi(\mathbf{B}_\gamma, \widehat{\mathbf{O}}, F_b)$. First, the points in $\mathbf{B}_\beta$ are re-indexed such that $j = \text{rank}(p, \mathbf{p})$ where $\mathbf{p} = \{p \in \{1 \ldots P\} : \mathbf{b}_p \in \mathbf{B}_\beta\}$. Next, for each object point $\hat{\mathbf{o}}_i$, the procedure calculates the minimum distances $d_{i,j}$ between the points in $\mathbf{B}_\beta$ and the ray $r(s, \hat{\mathbf{o}}_i) = s\frac{\hat{\mathbf{o}}_i}{\|\hat{\mathbf{o}}_i\|}$ where $s \in (0, \infty)$. Because the closest location on a ray to a given point is the projection of that point onto the ray, each $d_{i,j}$ can be calculated using the Pythagorean theorem, i.e.:

$$d_{i,j} = \sqrt{\|\mathbf{b}_j\|^2 - (\mathbf{b}_j \cdot \frac{\hat{\mathbf{o}}_i}{\|\hat{\mathbf{o}}_i\|})^2} \tag{5}$$

If $\min(\{d_{i,j}\}_{j=1}^{J}) < F_o$, $\hat{\mathbf{o}}_i$ is considered "occluded" and dropped from the final scene.

With regards to sensor effects, note that the number and locations of points detected on an object is a function of its position relative to a sensor. For example, when an object is moved closer to a sensor, more lidar beams strike the object, which produces a point cloud with more points. To simulate the appearance of the object point cloud at different distances, **P2P** uses an "unoccluding" function ζ (Figure 6) that in some sense is the reverse of the **P2P** occluding procedure.

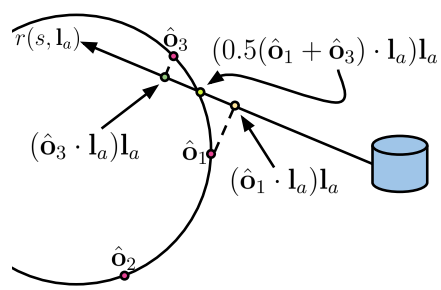

Figure 6. To simulate the appearance of an object's point cloud at different distances relative to the sensor, Paved2Paradise uses an "unoccluding" procedure that is essentially the reverse of the occluding approach shown in Figure 5. Specifically, an object's simulated surface point for a specific lidar beam $\mathbf{l}_a$ is calculated by taking the average of the projected object points $(\hat{\mathbf{o}}_i \cdot \mathbf{l}_a)\mathbf{l}_a$ on the ray $r(s, \mathbf{l}_a)$ for the two closest object points.

Specifically, given a ray $r(s, \mathbf{l}_a) = s\mathbf{l}_a$ where $\mathbf{l}_a$ is the direction (i.e., a unit vector) corresponding to the lidar beam indexed by a, the procedure calculates the distances $d_{a,i}$ to each object point $\hat{\mathbf{o}}_i$ using the same approach described for the occluding function. If no object points have a distance less than some user-defined threshold L to the ray, then the ray does not intersect with the object and no surface point is generated for that lidar beam. In the case where at least one $d_{a,i} < L$, the simplest strategy would be to take the point on the ray $(\hat{\mathbf{o}}_i \cdot \mathbf{l}_a)\mathbf{l}_a$ corresponding to the minimum $d_{a,i}$ as

the surface point. However, this point is often relatively far from the lidar beam's true intersection point, which causes the simulated object point cloud to appear "jagged". Instead, the **P2P** procedure takes *up to* two of the closest object points to the ray. If only one point meets this criterion, then that point is used as the object's surface point if $d_{a,i} < \frac{L}{2}$. If there are two close points, the surface point is their average. The final simulated scene $\mathbf{S}$ is thus:[10]

$$\mathbf{S} = (\mathbf{B} \setminus \mathbf{B}_\alpha) \cup (\mathbf{B}_\alpha \setminus \mathbf{B}_\gamma) \cup \mathbf{B}_\phi \cup \widehat{\mathbf{O}}_\zeta \tag{6}$$

A last practical consideration is that of storage. For our **orchard** dataset (Section 4), each synthetic scene contained ∼70,000 points, which would require ∼850 kB of storage per sample if each scene was saved in the PCD format. Therefore, our 500,000 synthetic scenes would require ∼425 GB of storage if each point cloud was stored in full. Such a storage rate introduces inconvenient dataset accessibility challenges (e.g., requiring datasets to be downloaded from a remote server whenever they are needed). Fortunately, the storage footprint for a **P2P** synthetic dataset can be considerably reduced by saving only: (1) the final transformed and occluded object point clouds and (2) the *indexes* of the occluded background points. Because the number of object points and occluded background points is much smaller than the total number of points in the final scene, the storage requirements are greatly reduced. During training, the synthetic scene for a given sample is assembled on the fly by: (1) loading the (transformed) object and (original) background point clouds, (2) selecting the *unoccluded* background points based on the occluded indexes, and (3) combining the two point clouds.

4. Experiments

To demonstrate the utility of **P2P**, we used the pipeline to generate synthetic datasets for two tasks: (1) human detection in orchards and (2) pedestrian detection in urban environments. We refer to these tasks as the **orchard** and **urban** tasks. The **orchard** task is an excellent case study for **P2P** because public datasets of humans in orchards do not exist. Further, the task is inherently challenging due to: (i) the wide variety of poses individuals working in an orchard can be found in and (ii) the frequency with which individuals are partially occluded by trees.

For the **orchard** task, we collected 6,717 proprietary background scenes by driving a Kawasaki MULE equipped with an Ouster OS1 (REV7) lidar sensor for ∼20 minutes through a walnut orchard located in Central California. For the **urban** task, we sourced 5,581 background scenes from the KITTI training dataset. Specifically, we randomly split the 7,481 KITTI training frames 90/10 into training and test sets consisting of 6,732 and 749 frames, which we refer to as the **urban** training and test sets. We then used all of the frames from the **urban** training set that did not have any annotated `Pedestrian` bounding boxes in our detection region as background scenes.

For both the **orchard** and **urban** tasks, we sourced objects from the same proprietary set of 21,263 object scenes, but we excluded crouching poses from the **urban** synthetic dataset leaving 11,054 object scenes for that task. The object scenes were collected by recording 48 different individuals performing a pose protocol in a parking lot over the span of eight hours. The pose protocol began with the individual standing on a flat rubber dot located 5 m in front of a lidar sensor. The individual was then instructed to slowly perform one full rotation (either clockwise or counterclockwise as determined by coin flip) in place while maintaining a neutral stance. Next, the individual was instructed to stand at a randomly selected location on a 1 m radius chalk circle surrounding the dot and then walk in a full circle in the clockwise direction. The individual was then instructed to stand at a different randomly selected location on the chalk circle and walk in a full circle in the *counterclockwise* direction. Finally, the previous two steps were repeated, but with the individual receiving the additional instruction to pantomime walking through trees (e.g., crouching and/or moving their arms as if they were moving branches out of the way). Once the human data was collected, we annotated a subset of the samples with bounding boxes using Segments.ai. We then trained a PointNet++ model on the annotated subset and used the trained model to generate the bounding boxes for the remaining samples (see Section 9 in the supplement for training details).

For each task, we generated 500,000 synthetic samples that were split 99.5/0.5 into training and validation sets of 497,500 and 2,500 samples. For the **orchard** task, the detection region bounds were $(0, 12)$, $(-4.625, 4.625)$, and $(-1, 5)$ for the x, y, and z dimensions, and the simulated lidar consisted of beams at 128 evenly spaced elevation angles from -22.5° to 22.5° and 2,048 evenly spaced azimuth angles from 0° to 360°. For the **urban** task, the detection region bounds were $(0, 19)$, $(-9, 9)$, and $(-2.5, 4)$ for the x, y, and z dimensions, and the simulated lidar consisted of beams at 64 evenly spaced elevation angles from -24.8° to 2° and 2,083 evenly spaced azimuth angles from 0° to 360°. For each task, we set $F_b = 3$ cm and $L = 4$ cm. We set $F_o = 4$ cm and $F_o = 8$ cm for the **orchard** and **urban** tasks, respectively. We doubled the number of background and object scenes for each task by mirroring each point cloud across the x-axis of the sensor. We further augmented the object scenes by placing the human point clouds at randomly selected locations on the ground plane in the detection prism. Lastly, for the **orchard** task, we only in-

[10] Note that multiple objects can be incorporated into a scene by following a two step algorithm: (1) apply **P2P** to a sampled object and background, (2) sample a new object and apply **P2P** while treating the previously created P2P scene as the background.

serted a single object into each synthetic scene, while for the **urban** task we inserted up to 10 objects in a scene.

For each task, we trained a neural network to perform a simplified object detection task—predicting the cell in a 2D grid that contains the center of the object's bounding box. Specifically, the target $\mathbf{Y}$ is a 2D binary grid full of zeros except for the cells corresponding to bounding box centers, which contain ones. This task is equivalent to pixel-wise binary classification and thus a model can be trained with the cross-entropy loss and evaluated threshold-free by calculating the average negative log-likelihood (NLL) and area under the precision-recall curve (AUC). For the **orchard** task, we evaluated the model on a two minute recording of a drive through a walnut orchard that features two individuals acting naturally and coming in and out of view. For the **urban** task, we evaluated both models on our **urban** test set.

Recently, Li et al. [19] showed PointPillars models can match the performance of newer architectures with minor architectural and/or training modifications, so we used PointPillars as our neural network for all experiments. Specifically, we trained three models: (1) a **P2P orchard** model, (2) a **P2P urban** model, and (3) a **baseline urban** model trained on our **urban** training set (which we further split 90/10 into training and validation sets consisting of 6,058 and 674 samples). See Section 10 in the supplement for training details.

5. Results

When using a detection probability threshold of 0.23 on the **orchard** evaluation sequence, the **P2P** model obtained 100% recall with only one false detection across 1,247 frames (741 of which included at least one human). The **P2P** model consistently detected humans under challenging conditions, such as when individuals were heavily occluded by tree branches as seen in Figure 7.[11] Notably, the model effortlessly generalized to detecting two humans in scenes even though the synthetic samples only ever contained a single individual at a time.

For the **urban** task, we report the NLL and AUC-PR at the grid level and scene level for our **urban** test set in Table 1. For the scene-level metrics, we assigned a point cloud a label of one if it contained at least one pedestrian and zero otherwise, and we used the model's *maximum* logit as its prediction. Note that our **urban** task setup *heavily favors* the **baseline** model because of how the datasets were created. Specifically, because the KITTI training set was derived from *sequences* of data, certain locations are repeated numerous times in the dataset, and even specific *individuals* appear in multiple samples; i.e., the **baseline** model is effectively allowed to "peek" at pedestrian samples in the **urban** test set.[12] Further, because the **P2P** pedestrian dataset was not labeled by the same individuals who labeled the KITTI dataset, the **P2P** labels may systematically differ from the KITTI labels in a way that favors the **baseline** model. Despite these significant disadvantages, the **P2P** model performed admirably, matching the **baseline** model's NLL and AUC at the scene level. Like the **orchard P2P** model, the **urban P2P** model can effectively detect pedestrians in challenging conditions, such as crowded scenes (see Figure 8 in the supplement).

[11] A video of the full sequence can be viewed at: https://drive.google.com/file/d/1iyyELpPYilfiQLS1UK_4fLTU-YTXtani/view?usp=sharing.

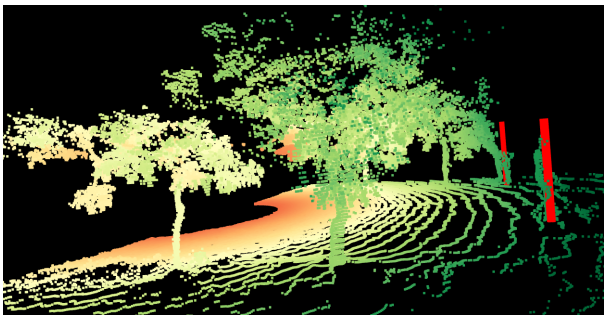

Figure 7. A PointPillars model trained *exclusively* on synthetic data generated by Paved2Paradise can reliably detect humans in orchards—including individuals who are heavily occluded by tree branches. The vertical red bars indicate the model's estimates of the bounding box centers for the humans.

Table 1. For our pedestrian detection task, a PointPillars model trained *exclusively* on Paved2Paradise synthetic data (P2P) matched the performance—as measured by the average negative log-likelihood (NLL) and area under the precision-recall curve (AUC)—of a model trained on the full KITTI dataset (Base) at the scene level (see Section 5).

	Grid		Scene	
Dataset	NLL	AUC	NLL	AUC
Base	1.11	0.34	0.21	0.96
P2P	1.52	0.23	0.21	0.95

6. Conclusion

Our results show **P2P** is a cost-effective and scalable pipeline for generating diverse and realistic synthetic lidar datasets. As a result, **P2P** could help accelerate point cloud model development in sectors where acquiring lidar datasets has previously been cost-prohibitive. While the minimalist lidar simulator we used here was effective, a more sophisticated simulator that incorporates (potentially learned) sensor physics [3, 11, 18, 22, 25, 40, 43] and/or weather effects [12, 13, 16] could produce even more realistic scenes.

[12] Among the 113 frames in the KITTI training set that have greater than five pedestrians in our detection region, only nine different locations are represented, with the top three locations being found in 30, 27, and 20 samples, respectively.

7. Author Contributions

MAA conceived and implemented the pipeline, designed and ran the experiments, and wrote the manuscript. NS led data collection efforts, discussed research ideas, and facilitated publication.

8. Acknowledgments

We would like to thank Aaron Avila, Drew Avila, Samir Chowdhury, Will Nitsch, and Joey Passarello for collecting the parking lot and orchard data. We would like to further thank Samir Chowdhury for his detailed feedback on the manuscript. Lastly, we would like to thank all of the volunteers who participated in the parking lot data collection.

References

[1] Jens Behley, Martin Garbade, Andres Milioto, Jan Quenzel, Sven Behnke, Cyrill Stachniss, and Jurgen Gall. Semantickitti: A dataset for semantic scene understanding of lidar sequences. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9297–9307, 2019. 1

[2] Jens Behley, Martin Garbade, Andres Milioto, Jan Quenzel, Sven Behnke, Jürgen Gall, and Cyrill Stachniss. Towards 3d lidar-based semantic scene understanding of 3d point cloud sequences: The semantickitti dataset. *The International Journal of Robotics Research*, 40(8-9):959–967, 2021. 1

[3] Lucas Caccia, Herke Van Hoof, Aaron Courville, and Joelle Pineau. Deep generative modeling of lidar data. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5034–5040. IEEE, 2019. 8

[4] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11621–11631, 2020. 1

[5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 1

[6] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. In *Conference on robot learning*, pages 1–16. PMLR, 2017. 2, 3

[7] Jin Fang, Dingfu Zhou, Feilong Yan, Tongtong Zhao, Feihu Zhang, Yu Ma, Liang Wang, and Ruigang Yang. Augmented lidar simulator for autonomous driving. *IEEE Robotics and Automation Letters*, 5(2):1931–1938, 2020. 2

[8] Jin Fang, Xinxin Zuo, Dingfu Zhou, Shengze Jin, Sen Wang, and Liangjun Zhang. Lidar-aug: A general rendering-based augmentation framework for 3d object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4710–4720, 2021. 3

[9] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3354–3361. IEEE, 2012. 1

[10] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013. 1

[11] Benoît Guillard, Sai Vemprala, Jayesh K Gupta, Ondrej Miksik, Vibhav Vineet, Pascal Fua, and Ashish Kapoor. Learning to simulate realistic lidars. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8173–8180. IEEE, 2022. 8

[12] Martin Hahner, Christos Sakaridis, Dengxin Dai, and Luc Van Gool. Fog simulation on real lidar point clouds for 3d object detection in adverse weather. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15283–15292, 2021. 8

[13] Martin Hahner, Christos Sakaridis, Mario Bijelic, Felix Heide, Fisher Yu, Dengxin Dai, and Luc Van Gool. Lidar snowfall simulation for robust 3d object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16364–16374, 2022. 8

[14] Xuzhong Hu, Zaipeng Duan, Xiao Huang, Ziwen Xu, Delie Ming, and Jie Ma. Context-aware data augmentation for lidar 3d object detection. In *2023 IEEE International Conference on Image Processing (ICIP)*, pages 11–15. IEEE, 2023. 1

[15] Braden Hurl, Krzysztof Czarnecki, and Steven Waslander. Precise synthetic image and lidar (presil) dataset for autonomous vehicle perception. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 2522–2529. IEEE, 2019. 2, 3

[16] Velat Kilic, Deepti Hegde, Vishwanath Sindagi, A Brinton Cooper, Mark A Foster, and Vishal M Patel. Lidar light scattering augmentation (lisa): Physics-based simulation of adverse weather conditions for 3d object detection. *arXiv preprint arXiv:2107.07004*, 2021. 8

[17] Alex H Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12697–12705, 2019. 1, 2

[18] Chenqi Li, Yuan Ren, and Bingbing Liu. Pcgen: Point cloud generator for lidar simulation. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11676–11682. IEEE, 2023. 2, 8

[19] Jinyu Li, Chenxu Luo, and Xiaodong Yang. Pillarnext: Rethinking network designs for 3d object detection in lidar point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17567–17576, 2023. 8

[20] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014. 1

[21] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. 1

[22] Sivabalan Manivasagam, Shenlong Wang, Kelvin Wong, Wenyuan Zeng, Mikita Sazanovich, Shuhan Tan, Bin Yang, Wei-Chiu Ma, and Raquel Urtasun. Lidarsim: Realistic lidar simulation by leveraging the real world. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11167–11176, 2020. 2, 3, 8

[23] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020. 1

[24] Lucas Mourot, Ludovic Hoyet, François Le Clerc, François Schnitzler, and Pierre Hellier. A survey on deep learning for skeleton-based human animation. In *Computer Graphics Forum*, pages 122–157. Wiley Online Library, 2022. 3

[25] Kazuto Nakashima and Ryo Kurazume. Learning to drop points for lidar scan synthesis. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 222–229. IEEE, 2021. 8

[26] Alexey Nekrasov, Jonas Schult, Or Litany, Bastian Leibe, and Francis Engelmann. Mix3d: Out-of-context data augmentation for 3d scenes. In *2021 International Conference on 3D Vision (3DV)*, pages 116–125. IEEE, 2021. 2

[27] Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Colored point cloud registration revisited. In *Proceedings of the IEEE international conference on computer vision*, pages 143–152, 2017. 3

[28] Hanspeter Pfister, Matthias Zwicker, Jeroen Van Baar, and Markus Gross. Surfels: Surface elements as rendering primitives. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 335–342, 2000. 2

[29] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2017. 4, 1

[30] Yuan Ren, Siyan Zhao, and Liu Bingbing. Object insertion based data augmentation for semantic segmentation. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 359–365. IEEE, 2022. 1, 2

[31] Kwonyoung Ryu, Soonmin Hwang, and Jaesik Park. Instant domain augmentation for lidar semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9350–9360, 2023. 1

[32] Cristiano Saltori, Fabio Galasso, Giuseppe Fiameni, Nicu Sebe, Elisa Ricci, and Fabio Poiesi. Cosmix: Compositional semantic mix for domain adaptation in 3d lidar segmentation. In *European Conference on Computer Vision*, pages 586–602. Springer, 2022. 2, 3

[33] Gernot Schaufler and Henrik Wann Jensen. Ray tracing point sampled geometry. In *Rendering Techniques 2000: Proceedings of the Eurographics Workshop in Brno, Czech Republic, June 26–28, 2000 11*, pages 319–328. Springer, 2000. 6

[34] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, et al. Scalability in perception for autonomous driving: Waymo open dataset. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2446–2454, 2020. 1

[35] Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *NeurIPS*, 2020. 1

[36] Sebastian Thrun and Michael Montemerlo. The graph slam algorithm with applications to large-scale mapping of urban structures. *The International Journal of Robotics Research*, 25(5-6):403–429, 2006. 2

[37] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017. 1

[38] Aoran Xiao, Jiaxing Huang, Dayan Guan, Fangneng Zhan, and Shijian Lu. Transfer learning from synthetic to real lidar point cloud for semantic segmentation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 2795–2803, 2022. 2, 3

[39] Yan Yan, Yuxing Mao, and Bo Li. Second: Sparsely embedded convolutional detection. *Sensors*, 18(10):3337, 2018. 1

[40] Ze Yang, Yun Chen, Jingkang Wang, Sivabalan Manivasagam, Wei-Chiu Ma, Anqi Joyce Yang, and Raquel Urtasun. Unisim: A neural closed-loop sensor simulator. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1389–1399, 2023. 8

[41] Ze Yang, Sivabalan Manivasagam, Yun Chen, Jingkang Wang, Rui Hu, and Raquel Urtasun. Reconstructing objects in-the-wild for realistic sensor simulation. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023. 2

[42] Li Yi, Boqing Gong, and Thomas Funkhouser. Complete & label: A domain adaptation approach to semantic segmentation of lidar point clouds. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 15363–15373, 2021. 1

[43] Vlas Zyrianov, Xiyue Zhu, and Shenlong Wang. Learning to generate realistic lidar point clouds. In *European Conference on Computer Vision*, pages 17–35. Springer, 2022. 8

Paved2Paradise: Cost-Effective and Scalable LiDAR Simulation by Factoring the Real World

Supplementary Material

9. PointNet++ Training Details

For our PointNet++ model, we used the same Set Abstraction levels described in Qi et al. [29] for their semantic scene labeling network. Max-pooling was applied to the output of the final Set Abstraction level, and the features were passed through a three layer MLP with 512, 512, and seven nodes, with the final features being passed through a hyperbolic tangent nonlinearity. The model was trained to predict the following seven values: $\Delta x_{\mathbf{c}}$, $\Delta y_{\mathbf{c}}$, $x_{\mathbf{d}}$, $y_{\mathbf{d}}$, $\Delta z_{\mathbf{d}}$, $\cos(\theta)$, and $\text{sign}(\sin(\theta))$, where $\Delta x_{\mathbf{c}}$ is the difference between $x_{\mathbf{c}}$ and the median x value in the extracted point cloud, $\Delta y_{\mathbf{c}}$ is calculated similarly, $\Delta z_{\mathbf{d}}$ is the difference between the height of the bounding box and the height of the human point cloud, and θ is the heading of the bounding box. The target values for $\Delta x_{\mathbf{c}}$, $\Delta y_{\mathbf{c}}$, $x_{\mathbf{d}}$, $y_{\mathbf{d}}$, and $\Delta z_{\mathbf{d}}$ were shifted and scaled so that their ranges lied in $[-1, 1]$. The human-labeled samples were split 97.5/2.5 into training and validation sets, and the model was trained on a single NVIDIA RTX A5500 GPU using the `AdamW` optimizer [21] in PyTorch with the default hyperparameters, mean squared error for the loss function, and a batch size of 32. Every time the validation loss did not improve for five consecutive epochs, we halved the learning rate, and we terminated training if the validation loss did not improve for 10 consecutive epochs.

10. PointPillars Training Details

For all of our PointPillars models, we used the same network hyperparameters described in Lang et al. [17] except we used a 100×100 grid for the **orchard** task and a 200×200 grid for the **urban** task. For the **orchard** task, we additionally passed the point coordinates through a NeRF-like positional encoding function [23, 35, 37] (the first six of the 10 frequencies used in NeRF) before feeding them into the PointPillars model. For both the **orchard** and **urban** tasks, we applied a point dropping augmentation during training for the **P2P** models where, for each sample, we sampled a drop probability $\gamma \sim \mathcal{U}(0, 0.2)$ and then dropped points in the object point cloud with probability γ. For the **urban** task, we additionally applied the training augmentations described in Lang et al. [17]. For the **orchard** task, we applied an additional "bumping" training augmentation where, for each sample, we leveled the point cloud using the $\mathbf{R_B}$ and $\mathbf{t_B}$ for the background and then "unleveled" the point cloud using the $\mathbf{R_{B'}}$ and $\mathbf{t_{B'}}$ for a randomly selected background. The models were trained on a single NVIDIA RTX A5500 GPU using the `AdamW` optimizer in PyTorch with the default hyperparameters and a batch size of eight. Every time the validation loss did not improve for five consecutive epochs, we halved the learning rate, and we terminated training if the validation loss did not improve for 10 consecutive epochs. For the **P2P** model, an "epoch" was defined as 64,000 training samples for both the **orchard** and **urban** tasks.

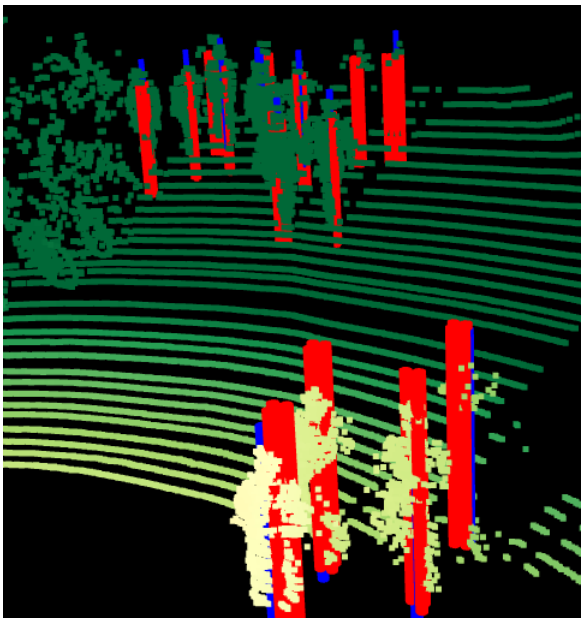

Figure 8. A PointPillars model trained on Paved2Paradise data can effectively detect multiple pedestrians in a crowded scene. The vertical blue bars are the ground truth locations of the bounding box centers for the pedestrians.