AdaSplats: Adaptive Splats from Semantic Point Cloud for Fast and High-Fidelity LiDAR Simulation

Jean Pierre Richa\textsuperscript{1,2}, Jean-Emmanuel Deschaud\textsuperscript{1}, François Goulette\textsuperscript{1} and Nicolas Dalmasso\textsuperscript{2}

\textsuperscript{1} MINES ParisTech, PSL University, Centre for Robotics, 75006 Paris, France
\textsuperscript{2} ANSYS France, 15 Pl. Georges Pompidou, 78180 Montigny-le-Bretonneux, France

Figure 1: Our pipeline to replicate the real world: from a raw outdoor point cloud acquired by a mobile mapping system, we perform automatic semantic segmentation, remove dynamic objects, and model the static environment with our adaptive splat modeling. The LiDAR sensor is simulated by sending rays through our splat model in an online fashion.

Abstract

LiDAR sensors provide rich 3D information about surrounding scenes and are becoming increasingly important for autonomous vehicles’ tasks, such as semantic segmentation, object detection, and tracking. The ability to simulate a LiDAR sensor will accelerate the testing, validation, and deployment of autonomous vehicles while reducing the cost and eliminating the risks of testing in real-world scenarios. To tackle the issue of simulating LiDAR data with high fidelity, we present a pipeline that leverages real-world point clouds acquired by mobile mapping systems. Point-based geometry representations, more specifically splats, have proven their ability to accurately model the underlying surface in very large point clouds. We introduce an adaptive splats generation method that accurately models the underlying 3D geometry, especially for thin structures. We have also developed a faster-than-real-time LiDAR simulation by ray casting on GPU while focusing on efficiently handling large point clouds. We test our LiDAR simulation in real-world conditions, showing qualitative and quantitative results compared to basic splatting and meshing techniques, demonstrating the superiority of our modeling technique.

1. Introduction

The last decade witnessed significant advances in Autonomous Vehicles (AVs), increasing the need for testing and validating new algorithms for safe navigation. Resource exhaustion and increased risk, time, and cost are some of the main drawbacks of testing AVs \cite{14}. Many algorithms used by AVs are data hungry and have to be tested in different scenarios, especially events that rarely appear in the real world. For this, a shift towards simulated environments \cite{13} was necessary. Although handcrafted simulators provide the leverage of testing AV algorithms, they introduce a large domain gap stemming from the almost perfect geometry in such simulators, and geometrical imperfection and higher diversity present in the real world. This gave rise to more realistic simulators \cite{23, 14} using real-world point clouds collected using LiDAR scanners mounted on a Mobile Mapping System (MMS). These methods model the real-world from outdoor point clouds using well-known splatting techniques \cite{40, 25} to reduce the domain gap. They simulate the LiDAR sensor in the splatted environment, but they neither demonstrate accurate geometric modeling of reality from MMS point clouds nor address the time aspect in LiDAR simulation, which is vital to accelerate testing and deployment of AVs.
LiDAR sensors on mobile platforms produce a huge amount of data containing noise caused by sensor calibration, localization errors, and so on. Surface reconstruction methods [18, 17] can be used for geometry representation, but they suffer from a performance drop (see section 4) when used on outdoor LiDAR point clouds. Point-based modeling and rendering [19], more specifically splatting [40], achieves high-quality 3D modeling while reducing the number of generated geometric primitives. Some approaches use planar geometries or hybrid mesh splat surface representation [12, 24], but they either do not focus on accurate geometry representation, or, in the case of the latter, they use very expensive mesh generation techniques. Rendering of points without connectivity information [28, 40, 34, 6] primarily focused on the use of accelerating data structures for efficiently rendering point-based geometry at interactive frame rates, in image space, while achieving a hole-free approximation of the geometry. Moreover, rendering a high number of splats [3] and high-quality shading [40, 39] can be achieved, as well as ray tracing splats [20]. All of these methods mainly work on carefully scanned small objects, but they fail to model large outdoor noisy point clouds.

LiDAR point clouds from MMS are highly anisotropic [27]. Resampling the point cloud reduces the anisotropy and increases uniformity by redistributing the points. Previous upsampling methods [18, 17] address this problem, but they require passing through computationally expensive representations. Deep learning methods for point cloud upsampling [36, 39] are used to increase the uniformity of the points’ distribution, but these methods are data hungry and usually limited to small objects or scenes. We propose a novel point cloud resampling, eliminating the need for extensive data preprocessing and training while achieving isotropic resampling in a splat-only approach, which further reduces the computational complexity.

Our contributions can be summarized as follows:

- AdaSplats: a novel adaptive splatting approach for accurate 3D geometry modeling of large outdoor noisy point clouds.
- A splat-based point cloud resampling, dealing with highly varying densities and scalable to large data.
- Faster-than-real-time GPU ray casting in the splat model for LiDAR sensor simulation.

2. Related Work

LiDAR Simulation Simulating the LiDAR sensor provides the ability to reduce the time and risk involved in testing AVs, through data generation in different scenarios created in virtual environments. The data is mostly used for training and/or testing the algorithms used by AVs to increase their decision-making capabilities. The availability of handcrafted simulated environments, such as CARLA [13] or BlenSor [15], or game engines (GTA-V), offers the ability to simulate the LiDAR sensor and collect scans [32, 16, 37], which are later used for data augmentation, cost free, and require minimal time. This technique leverages the huge amount of data that can be collected from such environments, but it introduces a large domain gap between synthetic and real-world data. While this gap can be reduced with domain adaptation strategies [33, 38], it still limits the ability of the algorithms used by AVs to generalize to the real world when trained on the simulated LiDAR data. A previous approach [11] extended in [29] focuses on accurate interaction between the LiDAR beam and the environment, which is modeled from real LiDAR data to reduce the domain gap. They introduce permeability to sample the points of intersection from 3D gaussian kernels contained in volumetric grids. Although good results can be achieved, a volumetric representation is not accurate for modeling the underlying surface, and the approach is computationally expensive, so it cannot be used for real-time applications. More recent approaches [14, 23] acquire data using a LiDAR mounted on an MMS and model the 3D geometry using splatting techniques after removing the dynamic objects in the foreground (e.g., pedestrians, cars, etc.). In the first approach [14] they do not take into account the physical model of the LiDAR since they do not cast rays; instead they use cube maps rasterization to accelerate the simulation. In the second [23] approach, they use Embree [31] to accelerate the ray-primitive intersection, but Embree runs on CPU and is still very far from real-time sensor simulation. Furthermore, although they achieve higher performance from their simulated data compared with simulated data in CARLA on object detection and semantic segmentation tasks, they do not focus on demonstrating accurate modeling of the static background, which they prove to play the most important part in elevating the deep neural networks’ performance in their experiments.

LiDAR simulation can be done offline, but achieving real-time simulation is important for accelerating AV testing and validation. Ray casting is used for physics-based LiDAR simulation by casting rays from the virtual sensor placed in the virtual scene. Ray casting is highly parallelizable and can be further accelerated through the use of accelerating structures. Embree [31] is a ray-tracing engine working on CPU that builds an acceleration structure that arranges the geometric primitives into bounded spatial positions and reduces the computation time by accelerating the ray-primitive intersection. Although it drastically reduces the ray-tracing time, it is still limited to the use of CPU cores to parallelize the ray casting. OptiX [25], on the other hand, builds a Bounding Volume Hierarchy (BVH) structure to arrange the geometric primitives in a tree and
benefits from the parallel architectures of GPUs to further accelerate the ray-casting task, reducing the time even further compared with Embree. Both Embree and OptiX offer the ability to define a custom geometric primitive and the corresponding ray-primitive intersection algorithm, making them good choices for simulating LiDAR sensors in a splatted scene. For this, we choose to use OptiX and implement the ray-splat intersection using CUDA to accelerate the intersection process and achieve real-time LiDAR simulation.

Splatting The race toward efficiently rendering point clouds began with the rise of 3D scanners and their ability to produce a very large number of points, giving a lot of information about the scanned object or environment. Point-based rendering gained interest after the report published by Levoy and Whitted on using points as display primitives [19]. The first works on rendering such primitives without any connectivity information focused on achieving a low rendering time and interactively displaying large amounts of points through the use of accelerating hierarchical data structures [28,26]. Following these early works, point-based rendering started shifting to a more clearly defined surface representation, which is the definition of splat primitives. Although these approaches could render large amounts of points at interactive frame rates, they still did not provide high rendering quality. Surface Splatting [40] introduced a point-rendering and texture-filtering technique to achieve high-quality anisotropic anti-aliasing. It combines oriented 2D reconstruction kernels (circular or elliptical splats) with a band-limiting image-space Elliptical Weighted Average (EWA) texture filter. Other works in this area [4,3] exploit the programmability of GPUs and detail the best practices for rendering point-based methods using elliptical splats, achieving a high frame rate and containing more details than similar scenes based on polygonal meshes. Surfels [26] is another approach that focuses on how to accurately map textures to splats to increase the visual details of the rendered objects at interactive rates. Whereas these approaches focused on rendering splats in image space, other works moved toward efficiently ray tracing them [20] using an octree to accelerate the ray-splat intersection. Leveraging this last method, which was implemented to work only on CPUs, we implement an efficient GPU ray-splat intersection, improve their splats generation method and use semantic information obtained from deep neural networks to build adaptive splats.

Resampling Data sparsity and nonuniformity dominate point clouds collected using an MMS, as shown in [27]. Isotropic resampling to increase the uniformity of the points’ distribution across the point cloud facilitates the splats generation and improves the normals estimation. A previous upsampling approach [11] works directly on the geometry inferred from the local neighborhood of points in the point cloud, in which the authors compute an approximation of the Voronoi diagram of neighborhood points at a random point, choose the Voronoi vertex whose circle has the largest radius, and project the vertex on the surface with the Moving Least Squares (MLS) projection, repeating the process until the radius of the largest circle is less than a defined threshold. This results in an upsampling that accurately approximates the surface geometry, but the Voronoi diagram approximations are expensive to compute. Other methods use deep learning techniques directly on point clouds to achieve higher density on the sparse point clouds through upsampling [36,39]. Depth completion can also be used to infer the completed depth map from an incomplete one, which can later be re-projected into 3D to upsample the point cloud [7,35]. However, current deep learning methods are limited to small scenes and suffer from a performance drop with unseen real-world data. Mesh-based resampling techniques can be achieved by reconstructing the surface from the acquired point cloud, simplifying the mesh while preserving the local underlying surface structure [22], and then sampling points on the reconstructed surface. Although they can achieve a good approximation of the surface, they are dependent on geometry errors introduced by surface reconstruction methods. Inspired by [1], we introduce a novel splat-based point cloud resampling approach, which increases the uniformity of points distribution. With the resampled point cloud, we achieve high-quality, hole-free surface modeling using our adaptive splats approach.

3. Method

In this section, we describe the main contributions in our pipeline, first introducing the adaptive splats generation algorithm, then our resampling method, and finally our real-time LiDAR simulation based on GPU ray casting. In our approach, we leverage the accuracy of the state-of-the-art deep learning method KPConv [30] in semantic segmentation of 3D outdoor point clouds. Using the semantic information, we adapt the splat growing and generation to better model the geometry. We also use the semantic information in the resampling process.

3.1. Adaptive Splats Generation

Basic splatting We describe here a variant of the splatting method of Linsen et al. [20], the basic algorithm on which we will develop our adaptive method. Let a vector \( \mathbf{x} \) with \( \hat{x} \) be a unit vector.

We consider as input data a point cloud \( P = \{ \mathbf{p}_i \in \mathbb{R}^3 \mid 0 \leq i \leq N \} \). We first compute an average radius \( \bar{R} \) of points in the \( K \)-nearest neighbors neighborhood of every point, \( \mathbf{p}_i \). For all experiments, we choose \( K = 80 \). We then define \( N_{\mathbf{p}_i} \), the neighborhood of \( \mathbf{p}_i \) to be the smallest neighborhood between \( K \)-nn and sphere of radius \( \bar{R} \), to be...
robust to the highly variable density of points. We perform Principal Component Analysis (PCA) on \( N_p \), to obtain the normal \( \hat{n}_i \) at each point \( p_i \), and reorient it with respect to the LiDAR sensor position.

Each splat \( S_i \) is defined by \( (c_{S_i}, \hat{n}_{S_i}, r_{S_i}) \), with \( c_{S_i} \) being the center of the splat, \( \hat{n}_{S_i} \) its unit normal vector, and \( r_{S_i} \) its radius. A splat \( S_i \) centered at \( p_i \) in \( P \) initially has \( \hat{n}_{S_i} = \hat{n}_i \) and \( r_i = 0 \), which is increased by including points in \( N_p \) (sorted in the order of increasing distance to \( p_i \)). We compute the signed point-to-plane distance of each neighbor \( p^k_i \in N_p \):

\[
\epsilon^k_i = \hat{n}_i \cdot (p^k_i - p_i) \tag{1}
\]

We stop the growing in \( N_p \) when \( |\epsilon^k_i| \) exceeds an error bound \( \bar{E} \) (see below). When the growing is done, we update the splat’s center position by moving it along the normal:

\[
c_{S_i} = p_i + \epsilon_i \hat{n}_i \tag{2}
\]

with \( \epsilon_i \) being the average signed point-to-plane distance of the points included in its generation. Then, we set the radius of the splat as the projected distance of the farthest point \( p^{k_{last}}_i \) from \( c_{S_i} \):

\[
r_{S_i} = ||(p^{k_{last}}_i - c_{S_i}) - \hat{n}_i \cdot (p^{k_{last}}_i - c_{S_i}) \hat{n}_i||_2 \tag{3}
\]

Then, all points in the neighborhood \( N_p \), inside the sphere of radius \( \alpha r_{S_i} \), are discarded from the splat generation. \( \alpha \) is a global parameter in \([0, 1]\) that allows to cover the entire surface without holes while minimizing the number of splats generated (we used \( \alpha = 0.2 \) for all experiments).

Before starting the generation process, we compute the error bound \( E \) as the average unsigned point-to-plane distance of points in all \( N_p \). Finally, we keep the \( m \) splats with radius \( r_{S_i} > 0 \), \( m \), where \( m \) is much lower than the number of points \( N \) in the point cloud.

Now that we have introduced the method that we call Basic Splats generation (that we use as a baseline), We move to explain our adaptive splats generation using semantic information.

**Adaptiveness** As shown in Figure 4, we first perform a semantic segmentation of the raw point cloud using deep learning [30] to obtain the semantic classes in the point cloud. We then remove the detected points classified as dynamic objects. Using the semantic information, we divide the points into four main groups:

- Ground: road and sidewalk.
- Surface: buildings and other similar classes that locally resemble a surface.
- Linear: poles, traffic signs, and similar objects.
- Non-surface: vegetation, fences, and similar objects.

In the adaptive splats generation, based on the group of the starting point \( p_i \), we change the neighborhood \( N_p \), with parameters \( K \) and \( \bar{R} \) (as a reminder, \( N_p \) is the smallest neighborhood between \( K \)-nn and a sphere of radius \( \bar{R} \)), and we change the error bound parameter \( \bar{E} \), the criterion that stops the growth of the splats. The parameters used for the four groups are as follows:

- Ground: \( 4K, 4\bar{R}, 4\bar{E} \)
- Surface: \( K, \bar{R}, \bar{E} \) (no change compared to basic splat)
- Linear: \( 0.33K, 0.33\bar{R}, 0.33\bar{E} \)
- Non-surface: \( 0.25K, 0.25\bar{R}, 0.25\bar{E} \)

We also stop growing splat \( S_i \) when a new point \( p^k_i \in N_p \), has a semantic class different from the class of \( p_i \).

The different parameters of the four groups were initially chosen by approximating the difference of the average local plane size for each class, and then they were finetuned to obtain the best modeling results.

These two adaptations in the growing of splats help to better model the geometry depending on the group and the semantics of points (e.g., improving splats for fine structures or the vegetation); they also improve the geometry at the intersection of different semantic areas and provide the ability to recover larger missing regions in ground and sidewalk neighborhoods.

Preserving sharp features in such noisy point clouds is not an easy task. Every splat in the generation phase with a normal \( \hat{n}_i \) will include a neighboring point \( p^k_i \) with a normal \( \hat{n}^k_i \) only if it passes the smoothness check \( \hat{n}_i \cdot \hat{n}^k_i > \beta \), (we took \( \beta = 0.6 \)). Once a point fails to pass this check, we stop growing the splat.

### 3.2. Splat-Based Point Cloud Resampling

Point clouds from MMSs have highly varying densities, proportional to the distance between the sensor and the scanned surface. A high level of local anisotropy also dominates the point clouds, which is caused by the physical model of the LiDAR (sweeps of lasers) and can be observed from the high density of points along the sweep lines of the LiDAR and sparser in other directions. To reduce local anisotropy, we resample the point cloud based on the approximation of the surface from a first splats generation. After resampling, we restart the whole process of adaptive splats generation on the new point cloud.

First, we generate splats from point cloud \( P \) with our adaptive variant. To obtain prior information on the acceptable local density throughout the splat surface, we compute the average splats density \( \delta \) as the average number of splats in a spherical neighborhood of radius \( \bar{R} \), excluding splats belonging to the non-surface group. For a splat \( S_i \), we start the resampling process whenever \( \delta_i < \delta \). We select the
farthest splat $S_j$ in the neighborhood of radius $R$. We verify whether both splats belong to the same semantic class and if they pass the smoothness check $\hat{n}_{S_i} \cdot \hat{n}_{S_j} > \beta$. If both checks are passed, we interpolate a new point that lies at the center of the segment connecting the splats’ centers. If one of the checks fails, we iterate through the neighboring splats in descending order of distance to splat $S_i$ and re-check both smoothness and semantic class equality. We repeat the same procedure until the desired local density is achieved. Because we need both the LiDAR sensor position and the semantic class for the splats generation, we assign to the new point the semantic class and LiDAR position of $p_i$ used to build splat $S_i$.

After the resampling step, adding the new points to the original point cloud $P$, we get a more uniformly distributed point cloud $P'$, which we use to restart our adaptive splats generation, able to fill small holes present before in the splat model.

3.3. LiDAR Simulation on GPU

Simulating the LiDAR sensor helps to accelerate the testing and deployment of AVs, but to speed up the testing phase, it is necessary to accelerate the simulation process and not be limited to offline simulation. Previous approaches do not tackle the real-time aspect of LiDAR simulation. We focus our work on accelerating the sensor simulation and achieve real-time LiDAR simulation. To do this, we use OptiX due to its ability to highly parallelize the ray-casting and intersection processes through the use of GPUs and an efficient acceleration data structure, respectively. OptiX does not provide a ray-splat intersection implementation, but it provides the ability to define a custom geometric primitive and its corresponding intersection. We have therefore defined our own splat primitive in the OptiX ray-tracing engine and implemented an efficient occurrence, as we will see in the results of the ray-splat intersection, allowing us to simulate millions of LiDAR rays per second. We cast the rays in the splatted environment, compute the ray-splat intersection by intersecting the ray with the plane defined by the splat’s center and the corresponding normal vector, and check whether the ray falls within the radius of the intersected splat. If an intersection is reported, we return the hit distance, compute the 3D points, and save the accumulated point cloud. With the implementation of ray-casting we can simulate any LiDAR type (even if the LiDAR has a different pattern from the LiDAR used to model the environment as we will see in section 4).

For our experiments we do not add meshes of dynamic objects in the scene, because our task is to see how accurately we can model the static environment. However, the addition of such objects can be integrated into our pipeline, knowing that we can achieve a hybrid ray-primitive intersection with OptiX to intersect triangles and splats.

4. Experiments And Results

Validating the correct modeling of the environment to simulate a LiDAR is a complex task. LiDARsim and AugmentedLiDAR approaches only do as experiments comparisons of learning results of deep networks between their simulated 3D data and data simulated under a simplified synthetic environment from CARLA. In this section, we detail the task protocol and how we compare the performance of the LiDAR simulation using our splatting technique (AdaSplats) and other surface representations.

4.1. Experiments

To be able to precisely compare different modeling techniques for the simulation, we choose three different datasets: two acquired by mobile LiDARs and one built from a Terrestrial Laser Scanner (TLS) (see Figure 2). More specifically, we choose (1) the Paris dataset from PC3D, which we call PC3D-Paris. This is an outdoor dataset acquired using a Velodyne HDL-32E in mapping configuration (pitched at 45 degrees). The dataset was acquired in the heart of Paris, a dense urban center with many complex objects (barriers, street lamps, traffic lights, vegetation, facades with balconies), allowing us to compare the modeling capacities of the different techniques in real situations. (2) SemanticKITTI acquired using a Velodyne HDL-64E in AV configuration in suburban roads around Karlsruhe (3) M-City, a dataset acquired in an autonomous vehicle testing site in Michigan using a TLS at fixed points in a controlled environment with no dynamic objects.

![Figure 2: Point clouds used in the experiments. PC3D-Paris, SemanticKITTI, and M-City, from left to right, respectively. In red, the trajectory used for simulation.](image)
face reconstruction (version 13.72) to mesh the point cloud, obtaining the finest mesh with an octree depth of 13. For IMLS, we use a voxel size of 7 cm and perform a sparse grid search, where we fill the IMLS signed distance function values only in voxels near the surface and use the marching cubes algorithm [21] to extract the iso-surface.

Compared with concurrent LiDAR simulation methods, such as LiDARsim [23] and AugmentedLiDAR [14], our Basic Splats modeling already offers finer models (LiDARsim produces surfels only in a voxel subsampling of the point cloud, and AugmentedLiDAR performs rendering by rasterization of splats in cube maps).

For all tested scenarios, we simulate the Velodyne HDL-64E LiDAR in AV mode (meaning positioned vertically), modeled with the complete firing sequence, returning around 150,000 points per scan, within a range of 120 meters. We generate new trajectories for each dataset and simulate the moving LiDAR inside the different scenes. As described in section 3.3, we implement our own ray-splat intersection in OptiX and use the original ray-triangles intersection of OptiX for meshed models.

For PC3D-Paris and SemanticKITTI, we take the original sensor positions provided with the datasets and offset them on the three axes to change the original LiDAR scan pattern (see Figure 2). More precisely, the offset for PC3D-Paris is $[1.0, 1.0, 0.0]$ along the $x$, $y$, and $z$ axes and $[1.0, 1.0, 0.0, -0.5]$ for SemanticKITTI. For M-City, we generate a linear trajectory across the dataset. For all of our LiDAR simulation experiments, we use a Nvidia GeForce RTX2070 SUPER GPU.

We provide different qualitative and quantitative results to validate the accuracy of our modeling and LiDAR simulation approach. To measure the accuracy of the different models, we compute the Cloud-to-Cloud (C2C) distance between the simulated point clouds (accumulation of all simulated scans) and the original point clouds (used to model the environment):

$$C2C(P_{sim}, P_{ori}) = \frac{1}{|P_{sim}|} \sum_{x \in P_{sim}} \min_{y \in P_{ori}} ||x - y||_2$$

where $P_{sim}$ and $P_{ori}$ are the simulated and original point clouds, respectively.

4.2. Results

We split the results according to the different datasets and show the performance of the LiDAR simulation using the different surface representations. Moreover, we show on PC3D-Paris that using deep learning for semantic segmentation [30] achieves similar results to AdaSplats using ground truth semantic information.

4.2.1 PC3D-Paris

Figure 3 shows renderings of the different models on the PC3D-Paris dataset (IMLS, Screened Poisson, Basic Splats and AdaSplats). We are able to achieve a more realistic scene modeling using our AdaSplats method.

Figure 4 illustrates qualitative results comparing the accumulated point clouds from the LiDAR simulation with the different surface representations. The last image is the original point cloud used to model the environment. The other images are an accumulation of simulated scans (in blue one simulated LiDAR scan). AdaSplats is able to generate higher-quality LiDAR data when compared to Basic Splats and other meshing techniques.

Figure 3: Rendering of the different surface representations on PC3D-Paris. The top row shows the reconstructed surface using IMLS (left) and Screened Poisson (right) methods. The bottom row shows the splatted scene with Basic Splats (left) and AdaSplats (right).

Figure 4: Comparison of simulated LiDAR data using different reconstruction and modeling methods on PC3D-Paris. Top: simulation in meshed IMLS (left) and Poisson (right). Middle: the simulation with Basic Splats (left) and AdaSplats-KPConv (right). Bottom: the simulation with AdaSplats-GT (left) and original point cloud (right).
The simulation in meshed or basic splat environments does not work well on thin objects containing few points, such as fences, poles, and traffic signs. Basic splatting techniques are not able to adapt to the local sparsity without semantic information. Screened Poisson [17] and IMLS [18] do not work well on outdoor noisy LiDAR data, especially on thin objects. These surface reconstruction methods have issues with open shapes: borders are diluted because these functions try to close the surface, as it is performing inside/outside classification. To limit this effect, we truncate the IMLS function at 2 voxels and perform surface trimming with Poisson, but we can still see artifacts on Figure 4 in the red and orange areas.

Our method is also verified quantitatively (see Table 1). We compare the meshing time of Poisson and IMLS, and the splatting time of Basic Splats and AdaSplats. AdaSplats-KPConv includes KPConv for automatic semantic segmentation (trained on the training set of PC3D dataset). AdaSplats-GT uses the Ground Truth semantic (manual annotation). We see that splatting is much faster than surface reconstruction. With KPConv, our adaptive splatting and resampling technique, it increases modeling time but remains within order of magnitude of Poisson reconstruction (the inference time of KPConv is around 600 seconds for 10 million points).

The simulation frequency is the simulation time of one LiDAR scan (full 360 degrees azimuth turn of the Velodyne, including generating the LiDAR rays at a given position, host (CPU)-to-device (GPU) communication, ray-casting, primitives intersection, and reporting back the buffer containing points of intersection. Our ray-splat intersection is very fast (very close to the GPU hard coded ray-triangles intersection) and our simulation with AdaSplats is around 20 times faster than real time.

We also report the C2C distance between the simulated and original point cloud (see equation 4). AdaSplats-KPConv improves the accuracy over Basic Splats and AdaSplats-GT shows that with improved semantics, the simulated data can be the closest to the original.

We notice that point clouds contain a huge amount of points on the ground, which is the easiest class to model and has a higher effect on the computed distance, but thin structures contain fewer points and are important for AV simulation. To measure the modeling of thin structures, we pick three classes from PC3D-Paris, compute the C2C distance on these classes, and report the results in Table 2.

We observe that AdaSplats obtains much better results than IMLS, Poisson or Basic Splats. AdaSplats-KPConv is able to achieve a C2C distance very close to the model constructed with ground truth semantic information. We achieve a lower C2C distance on poles and traffic signs with KPConv due to misclassifications, leading to the generation of smaller splats.

| Model            | Gen T (in s) | Gen Prim (#) | Sim Freq (in Hz) | C2C (in cm) |
|------------------|--------------|--------------|------------------|-------------|
| Mesh - Poisson   | 797          | 5.20M        | 232 Hz           | 2.3 cm      |
| Mesh - IMLS      | 3216         | 4.95M        | 233 Hz           | 2.0 cm      |
| Basic Splats     | 200          | 5.40M        | 135 Hz           | 2.3 cm      |
| AdaSplats-KPConv | 1064         | 1.75M        | 203 Hz           | 2.2 cm      |
| AdaSplats-GT*    | 169          | 2.84M        | 180 Hz           | 1.99 cm     |
| AdaSplats-GT     | 451          | 1.72M        | 205 Hz           | 1.97 cm     |

Table 1: Comparison of LiDAR simulation on PC3D-Paris. We report the time taken (Gen T) in seconds to generate the primitives (triangular mesh, or splats), the number of generated primitives (Gen Prim) in millions (M), simulation frequency (Sim Freq) in Hz, and the Cloud-to-Cloud Distance (C2C) in cm between simulated and original point cloud.

| Model            | Fences | Poles | Traffic Signs | Average |
|------------------|--------|-------|---------------|---------|
| Mesh - Poisson   | 5.9    | 6.1   | 6.7           | 6.2     |
| Mesh - IMLS      | 4.6    | 3.5   | 2.9           | 3.7     |
| Basic Splats     | 4.7    | 4.3   | 3.4           | 4.1     |
| AdaSplats-KPConv | 5.5    | 2.1   | 1.1           | 2.9     |
| AdaSplats-GT*    | 2.5    | 2.4   | 1.8           | 2.3     |
| AdaSplats-GT     | 2.4    | 2.3   | 1.8           | 2.2     |

Table 2: Cloud-to-Cloud distance (in cm) computed on PC3D-Paris for points that belong to classes of thin structures, between the simulated and original point cloud.

In Table 1 and Table 2, AdaSplats-GT* is our adaptive splatting method without the resampling process. By comparison, we get less primitives (1.72M) with resampling than without (2.84M). We have also a better repartition of splats on thin objects with resampling (2.2 cm) than without (2.3 cm).

AdaSplats is a method that uses, but does not require perfect semantics as can be seen with the results using KPConv’s inferences (which have errors). On the contrary, modeling methods that have specific models for semantic objects (e.g., a specific model for traffic lights) are highly dependent on the quality of the semantics and no longer work with the slightest error. Compared to mesh-based models using surface reconstruction, splats are independent surface elements whose parameters can be easily changed according to semantics (unlike methods based on SDFs, like IMLS, or on an indicator function, like Poisson). As our proposal is not focused on the semantic segmentation part, we will only use the semantic ground truth for the following experiments.

4.2.2 SemanticKITTI

Figure 5 and Table 3 show qualitative and quantitative results comparing the simulated point clouds using the different scene representations for the SemanticKITTI dataset.
Viewing the results of simulation on SemanticKITTI, we can see that our modeling method is not limited to a specific LiDAR sensor, or configuration since SemanticKITTI was acquired with a Velodyne HDL-64E in AV configuration, which is different from PC3D-Paris.

Table 3: Comparison of LiDAR simulation on SemanticKITTI. We report the time taken (Gen T) in seconds to generate the primitives (triangular mesh, or splats), the number of generated primitives (Gen Prim) in millions (M), simulation frequency (Sim Freq) in Hz, and the Cloud-to-Cloud (C2C) distance (in cm) between simulated and original point cloud.

| Model        | Gen T (in s) | Gen Prim (#) | Sim Freq (in Hz) | C2C (in cm) |
|--------------|--------------|--------------|------------------|-------------|
| Mesh - Poisson | 796          | 9.97M        | 229 Hz           | 2.6 cm      |
| Mesh - IMLS  | 1380         | 7.05M        | 222 Hz           | 3.0 cm      |
| Basic Splats | 185          | 7.77M        | 144 Hz           | 2.6 cm      |
| AdaSplats-GT | 544          | 4.56M        | 180 Hz           | 2.0 cm      |

Table 4: Comparison of LiDAR simulation on M-City. We report the time taken (Gen T) in seconds to generate the primitives (triangular meshes, or splats), the number of generated primitives (Gen Prim) in thousands (K) or millions (M), simulation frequency (Sim Freq) in Hz, and the Cloud-to-Cloud Distance (C2C) in cm between simulated and original point cloud.

| Model          | Gen T (in s) | Gen Prim (#) | Sim Freq (in Hz) | C2C (in cm) |
|----------------|--------------|--------------|------------------|-------------|
| Mesh - Manual  | 3 days       | 71.5K        | 259 Hz           | 7.0 cm      |
| Basic Splats   | 199          | 5.82M        | 81 Hz            | 1.7 cm      |
| AdaSplats-GT   | 513          | 3.01M        | 204 Hz           | 1.5 cm      |

Figure 5: Comparison of simulated LiDAR data using different reconstruction and modeling methods on SemanticKITTI. The top row: the original point cloud. The middle row: the simulation in meshed IMLS (left) and Poisson (right). The bottom row: the simulation with Basic Splats (left) and AdaSplats (right).

4.2.3 M-City

For M-City, we do not have the sensor positions to re-orient the normals. Not having a correct normal orientation makes it difficult to properly reconstruct the surface using automatic meshing techniques (Poisson and IMLS). Instead, we do have a mesh manually reconstructed by 3D artists (in three days of work). So we use this manual meshed model, performing a comparison against a carefully reconstructed scene (which correspond to the model used for many applications). Even without orientation of the normals, our AdaSplats method works. Figure 6 and Table 4 show qualitative and quantitative results comparing the simulated point clouds, using M-City, inside the different scene models.

Figure 6: Comparing the manually meshed (bottom left) and automatically splatted (bottom right). Original M-City point cloud (top image). Modeling vegetation (in red) is not an easy task and usually requires different ray-primitive intersection methods for a more accurate rendering.

With M-City, we demonstrate that our pipeline can also be used on point clouds collected using a TLS, achieving LiDAR simulation results that are closer to reality than a manually reconstructed model. This is due to the modification of the local geometry done by 3D artists to simplify the reconstruction task (e.g., on the vegetation or some traffic signs).

5. Conclusions

We have presented a fully automatic processing pipeline enabling the modeling of a static environment from real outdoor data with only a few minutes of preprocessing. Then, we simulate a LiDAR in an adaptive splats model to produce high-fidelity simulated LiDAR data, achieving better results than with meshing models or classic splatting. Our LiDAR simulation is much faster than real time, which makes it possible to create massive simulations for testing AV algorithms based on real data.
References

[1] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C.T. Silva. Computing and rendering point set surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 9(1):3–15, 2003.

[2] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, C. Stachniss, and J. Gall. SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences. In *Proc. of the IEEE/CVF International Conf. on Computer Vision (ICCV)*, 2019.

[3] M. Botsch, A. Hornung, M. Zwicker, and L. Kobbelt. High-quality surface splatting on today’s gpus. In *Proceedings Eurographics/IEEE VGTC Symposium Point-Based Graphics, 2005.*, pages 17–141, 2005.

[4] Mario Botsch and Leif Kobbelt. High-quality point-based rendering on modern gpus. In *Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, PG '03, page 335, USA, 2003. IEEE Computer Society.

[5] Mario Botsch, Michael Spernat, and Leif Kobbelt. Phong splatting. In *Proceedings of the First Eurographics Conference on Point-Based Graphics*, SPBG'04, page 25–32, Goslar, DEU, 2004. Eurographics Association.

[6] Hassan Bouchiba, Jean-Emmanuel Deschaud, and François Goulette. Raw Point Cloud Deferred Shading Through Screen Space Pyramid Operators. In Olga Diamanti and Amir Vaxman, editors, *EG 2018 - Short Papers*. The Eurographics Association, 2018.

[7] Yun Chen, Bin Yang, Ming Liang, and Raquel Urtasun. Learning joint 2d-3d representations for depth completion. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 10022–10031, 2019.

[8] Zhonggui Chen, Tieyi Zhang, Juan Cao, Yongjie Jessica Zhang, and Cheng Wang. Point cloud resampling using centroidal voronoi tessellation methods. *Computer-Aided Design*, 102:12–21, 2018. Proceeding of SPM 2018 Symposium.

[9] Jean-Emmanuel Deschaud. Imls-slam: Scan-to-model matching based on 3d data. In *2019 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2480–2485, 2018.

[10] Jean-Emmanuel Deschaud, David Duque, Jean Pierre Richa, Santiago Velasco-Forero, Beatriz Marcotegui, and François Goulette. Paris-carla-3d: A real and synthetic outdoor point cloud dataset for challenging tasks in 3d mapping. *Remote Sensing*, 13(22), 2021.

[11] Jean-Emmanuel Deschaud, David Prasser, M. Freddie Dias, Brett Browning, and Peter Rander. Automatic data driven vegetation modeling for lidar simulation. In *2012 IEEE International Conference on Robotics and Automation*, pages 5030–5036, 2012.

[12] Alexandre Devaux and Mathieu Brédif. Realtime projective multi-texturing of pointclouds and meshes for a realistic street-view web navigation. In *Proceedings of the 21st International Conference on Web3D Technology*, Web3D ’16, page 105–108, New York, NY, USA, 2016. Association for Computing Machinery.

[13] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[14] 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.

[15] Michael Gschwandtner, Roland Kwitt, Andreas Uhl, and Wolfgang Pree. Blensor: Blender sensor simulation toolbox. In George Bebis, Richard Boyle, Bahram Parvin, Darko Koracin, Song Wang, Kim Kyungnam, Bedrich Benes, Kenneth Moreland, Christoph Borst, Stephen DiVerdi, Chiang Yi-Jen, and Jiang Ming, editors, *Advances in Visual Computing*, pages 199–208, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[16] 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, 2019.

[17] Michael Kazhdan and Hugues Hoppe. Screened poisson surface reconstruction. *ACM Trans. Graph.*, 32(3), jul 2013.

[18] Ravikrishna Kolluri. Provably good moving least squares. *ACM Trans. Algorithms*, 4(2), may 2008.

[19] Marc Levoy and Turner Whitted. The use of points as a display primitive. 2000.

[20] Lars Linsen, Karsten Müller, and Paul Rosenthal. Splat-based ray tracing of point clouds. *Journal of WSCG*, pages 51–58, 2008.

[21] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21(4):163–169, aug 1987.

[22] Adrien Maglo, Guillaume Lavoué, Florent Dupont, and Céline Hudelot. 3d mesh compression: Survey, comparisons, and emerging trends. *ACM Comput. Surv.*, 47(3), feb 2015.

[23] Sivabalan Manivasagam, Shenlong Wang, Kelvin Wong, Wenyuan Zeng, Mikita Szanavich, Shuhan Tan, Bin Yang, Wei-Chiu Ma, and Raquel Urtasun. LidarSim: Realistic lidar simulation by leveraging the real world. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11164–11173, 2020.

[24] Rafael Pagés, Sergio García, Daniel Berjón, and Francisco Morán. Splash: A hybrid 3d modeling/rendering approach mixing splats and meshes. In *Proceedings of the 20th International Conference on 3D Web Technology*, Web3D ’15, page 231–234, New York, NY, USA, 2015. Association for Computing Machinery.

[25] Steven G. Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hobrock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, and Martin Stich. Optix: A general purpose ray tracing engine. *ACM Trans. Graph.*, 29(4), jul 2010.

[26] 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*, SIGGRAPH ’00, page 335–342, USA, 2000. ACM Press/Addison-Wesley Publishing Co.
[27] Xavier Roynard, Jean-Emmanuel Deschaud, and François Goulette. Paris-lille-3d: A large and high-quality ground-truth urban point cloud dataset for automatic segmentation and classification. *The International Journal of Robotics Research*, 37(6):545–557, 2018.

[28] Szymon Rusinkiewicz and Marc Levoy. Qsplat: A multiresolution point rendering system for large meshes. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH ’00, page 343–352, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[29] Abhijeet Tallavajhula, Cetin Mericli, and Alonzo Kelly. Off-road lidar simulation with data-driven terrain primitives. In 2018 IEEE International Conference on Robotics and Automation (ICRA), pages 7470–7477, 2018.

[30] Hugues Thomas, Charles R. Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas Guibas. Kpconv: Flexible and deformable convolution for point clouds. In 2019 IEEE/CVF International Conference on Computer Vision (ICCV), pages 6410–6419, 2019.

[31] Ingo Wald, Sven Woop, Carsten Benthin, Gregory S. Johnson, and Manfred Ernst. Embree: A kernel framework for efficient cpu ray tracing. *ACM Trans. Graph.*, 33(4), jul 2014.

[32] Bichen Wu, Alvin Wan, Xiangyu Yue, and Kurt Keutzer. Squeezeseg: Convolutional neural nets with recurrent crf for real-time road-object segmentation from 3d lidar point cloud. In 2018 IEEE International Conference on Robotics and Automation (ICRA), pages 1887–1893, 2018.

[33] Bichen Wu, Xuanyu Zhou, Sicheng Zhao, Xiangyu Yue, and Kurt Keutzer. Squeezesegv2: Improved model structure and unsupervised domain adaptation for road-object segmentation from a lidar point cloud. In 2019 International Conference on Robotics and Automation (ICRA), pages 4376–4382, 2019.

[34] Jainhua Wu and Leif Kobbelt. Optimized sub-sampling of point sets for surface splatting. *Computer Graphics Forum*, 23(3):643–652, 2004.

[35] Yan Xu, Xinge Zhu, Jianping Shi, Guofeng Zhang, Hujun Bao, and Hongsheng Li. Depth completion from sparse lidar data with depth-normal constraints. In 2019 IEEE/CVF International Conference on Computer Vision (ICCV), pages 2811–2820, 2019.

[36] Lequan Yu, Xianzhi Li, Chi-Wing Fu, Daniel Cohen-Or, and Pheng-Ann Heng. Pu-net: Point cloud upsampling network. In 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 2790–2799, 2018.

[37] Xiangyu Yue, Bichen Wu, Sanjit A. Seshia, Kurt Keutzer, and Alberto L. Sangiovanni-Vincentelli. A lidar point cloud generator: From a virtual world to autonomous driving. In *Proceedings of the 2018 ACM on International Conference on Multimedia Retrieval*, ICMR ’18, page 458–464, New York, NY, USA, 2018. Association for Computing Machinery.

[38] Sicheng Zhao, Yezhen Wang, Bo Li, Bichen Wu, Yang Gao, Pengfei Xu, Trevor Darrell, and Kurt Keutzer. epointda: An end-to-end simulation-to-real domain adaptation framework for lidar point cloud segmentation. *CoRR*, abs/2009.03456, 2020.

[39] Kaiyue Zhou, Ming Dong, and Suzan Arslanturk. "zero shot" point cloud upsampling. *CoRR*, abs/2106.13765, 2021.

[40] Matthias Zwicker, Hanspeter Pfister, Jeroen van Baar, and Markus Gross. Surface splatting. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH ’01, page 371–378, New York, NY, USA, 2001. Association for Computing Machinery.