Urban Scene Modeling From 3D Point Clouds and Massive LiDAR Simulation for Autonomous Vehicles

Jean Pierre Richa

PSL
UNIVERSITÉ PARIS

THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PSL

Préparée à Mines Paris-PSL

Modélisation de Scènes Urbaines à Partir de Nuages de Points 3D et Simulation LiDAR Massive pour Véhicules Autonomes
Urban Scene Modeling From 3D Point Clouds and Massive LiDAR Simulation for Autonomous Vehicles

Soutenue par

Jean Pierre RICHA

Le 08 Décembre 2022

École doctorale n°621

ISMME

Spécialité

Informatique temps réel, robotique et automatique - Paris

Composition du jury :

Raphaëlle Chaine Professeur, Université de Lyon	*Présidente*
Bruno Vallet Chargé de recherche, IGN	*Rapporteur*
Giorgio Grisetti Associate professor, Sapienza Université de Rome	*Rapporteur*
Jean-Emmanuel Deschaud Chargé de recherche, MINES-PSL	*Examinateur*
François Goulette Professeur, MINES-PSL	*Directeur de thèse*

MINES PARIS | PSL

Acknowledgements

I believe that will, persistence and believing on oneself are key to achieve the unimaginable. This is true, however, I was also lucky to be surrounded by amazing people even before the PhD without whom, the success of this work would not have been possible. Having said that, I would like to take the time to thank the incredible people who made the completion of this work a reality.

First and foremost, I would like to thank my Director François Goulette and co-director Jean-Emmanuel Deschaud for their invaluable advice, and continuous support and assistance at every stage of this thesis. Their profound knowledge and immense experience in combination with a great intellectual support made the success of this work possible. Moreover, their incredible moral support helped me through difficult times as they made everything easier, even when it got tough. I was lucky to have chosen them as my advisers and I cannot thank them enough for everything they did and I hope that I made a relationship that would last a lifetime with them.

I would also like to thank my industrial advisers at ANSYS, namely, Nicolas Dalmasso, Vincent Hourdin, Nassim Jibai, David Ganieux and Gilles Gallee for their technical support, which brought many years of high quality technical experience to my work. I would also like to thank them for facilitating the collaboration to make the completion of this thesis as smooth as possible, and a special thank you to Nicolas Dalmasso for always pushing my technical capabilities.

When i first embarked on this journey I was not alone, in fact my fiancée, Désirée Tannous and I defended our PhDs within 4 days, so we went through the differet phases together and I am thankful for that. I would like to thank her for being there through good and bad, even during my masters degree when we were in different countries, she was incredibly supportive even when she was going through difficult times and I can't thank her enough for that.

The journey started in my home country Lebanon, when I decided to continue my studies in a different country. At the time, I crossed paths with my university Doctor, Iyad Al Khatib who I approached to find an internship abroad, because I wanted to live an international experience. A few days later, we got an offer for an internship in the lab of Prof. Fabio Curti with whom he collaborated at La Sapienza University of Rome, so I went there and passed an amazing summer internship with incredible people. This was the first step towards my PhD and I cannot thank Dr. Iyad enough for the amazing opportunity, because I found my masters program through the internship and this was a turning point in my life.

At the end of my masters program, I did my final year project with my dear Professor Giorgio Grisetti from whom I learned a lot. My journey with Prof. Giorgio did not stop there, and I was happy when I learned that he accepted to be a "rapporteur" for my PhD thesis. I could not be more honored for such an amazing opportunity, so thank you Giorgio.

Last but not least, this work could not have been possible without the support of my family and friends, especially my mom Sylvie Bou Mansour, dad Sami Richa and my brother Charbel Richa. Moreover, I spent 3 amazing years with my colleagues Sofiane Horache, Pierre Dellenbach, Agapius Bou Ghosn, Jules Sanchez, Louis Soum-Fontez, Nathan Sanchiz-Viel, Hugo Blanc and the rest of the CAOR center to whom I am thankful for the incredible

support.

Abstract

The development of autonomous vehicles has seen increased interest from the research community and industry over the last decade. Developing algorithms used by autonomous vehicles requires retrofitting vehicles with multiple sensors, which has a high cost. Moreover, the vehicles daily operation and maintenance raise the cost even further and introduce a high risk on other vehicles and people in their surrounding. The development and testing can be also achieved by simulating the autonomous vehicle and the different sensors in a virtual environment. However, manually handcrafted virtual environments fail to generalize to real-world scenes, due to the domain gap that arises from the over-simplified models used in such environments. In this thesis, we propose to reduce this domain gap by leveraging real-world scans of urban scenes in the form of 3D point clouds acquired using a LiDAR sensor mounted on a mobile mapping system. Toward this end, we create a new annotated 3D point cloud dataset and propose an automatic simulation pipeline. The pipeline introduces a new intermediate representation to complete the scene geometry through deep 3D architectures, an accurate scene modeling method based on semantic adaptive splatting to create the virtual environment and finally, a real-time LiDAR simulation method. Our pipeline is fast, automatic and can be used to augment the scenarios domain and generate massive simulations.

Contents

1 Introduction **1**
- 1.1 General Introduction . . . 2
- 1.2 REPLICA Project . . . 5
- 1.3 Thesis Scope . . . 5
  - 1.3.1 Employed Sensors . . . 6
  - 1.3.2 Required Tasks . . . 6
- 1.4 Team . . . 7
  - 1.4.1 Center of Robotics . . . 7
    - 1.4.1.1 NPM3D Team and L3D2 Mobile Mapping System . . . 7
  - 1.4.2 ANSYS . . . 7
- 1.5 Identified Problems . . . 7
- 1.6 Proposed Solutions . . . 8
- 1.7 Automatic Reality Replication Pipeline . . . 9
- 1.8 Thesis Structure . . . 11
- 1.9 Contributions . . . 12

2 Datasets **13**
- 2.1 Introduction . . . 14
- 2.2 Related Work . . . 16
- 2.3 Employed Datasets . . . 17
  - 2.3.1 M-City . . . 17
  - 2.3.2 SemanticKITTI . . . 18
  - 2.3.3 Matterport3D . . . 18
- 2.4 Paris-CARLA-3D Dataset Creation . . . 19
  - 2.4.1 Acquisition Platform and Used Sensors . . . 20
  - 2.4.2 Real Paris . . . 21
  - 2.4.3 Synthetic CARLA . . . 26
- 2.5 Paris-CARLA-3D Dataset Properties . . . 27
  - 2.5.1 Classes Statistics . . . 27
  - 2.5.2 Split for training . . . 28
  - 2.5.3 Interest in having both synthetic and real data . . . 29
  - 2.5.4 Transfer learning . . . 29
- 2.6 Scene Completion Task . . . 29
  - 2.6.1 Task Protocol . . . 30
  - 2.6.2 Experiments: setting a baseline . . . 31
    - 2.6.2.1 Qualitative results . . . 32
    - 2.6.2.2 Quantitative Results . . . 32
    - 2.6.2.3 Transfer learning with scene completion . . . 33
- 2.7 Tasks per dataset . . . 34
- 2.8 Conclusion . . . 34

3 Scene Completion **35**
3.1 Introduction . . . 36
3.2 Related Work . . . 40
3.2.1 Distance Functions . . . 40
3.2.1.1 Signed Distance Function . . . 40
3.2.1.2 Unsigned Distance Function . . . 41
3.2.2 Implicit Function Learning . . . 41
3.2.3 Point Cloud Completion . . . 41
3.2.4 Scene Completion . . . 43
3.2.4.1 Scene Completion from RGB-D Sensors . . . 43
3.2.4.2 Scene Completion from LiDAR Sensors . . . 45
3.3 Scene Representation . . . 46
3.3.1 Unsigned Weighted Euclidean Distance (UWED) . . . 46
3.3.2 Point Cloud Extraction . . . 47
3.3.2.1 Extraction from SDF . . . 47
3.3.2.2 Extraction from UDF . . . 47
3.4 Experiments and Results . . . 48
3.4.1 Distance Functions for Scene Completion . . . 48
3.4.2 Dataset Preparation . . . 49
3.4.3 Evaluation Metrics for Scene Completion . . . 50
3.4.4 Implementation Details . . . 50
3.4.5 Validity of our UDF as Scene Representation . . . 50
3.4.6 Results of Distance Functions for Scene Completion . . . 53
3.4.7 Scene Completion on Target . . . 58
3.4.8 Limitations and Perspectives . . . 60
3.5 Conclusion . . . 60

4 Scene Modeling **63**
4.1 Introduction . . . 64
4.2 Related Work . . . 67
4.2.1 Surface Reconstruction . . . 67
4.2.2 Point-based Surface Modeling . . . 68
4.2.3 Neural Radiance Fields . . . 71
4.2.4 Resampling . . . 71
4.3 Adaptive Splatting . . . 72
4.3.1 Basic-Splatting . . . 72
4.3.2 Adaptive Splatting . . . 73
4.3.3 Splat-based Resampling and Denoising . . . 73
4.4 Splat Ray Tracing . . . 77
4.4.1 Ray-splat Intersection . . . 77
4.4.2 OptiX . . . 77
4.4.3 Blending and Shading . . . 78
4.5 Experiments and Results . . . 79
4.5.1 Experiments . . . 79
4.5.1.1 Surface Representation . . . 79
4.5.1.2 Datasets . . . 79
4.5.2 Results . . . 80
4.5.2.1 Paris-Carla-3D . . . 81
4.5.2.2 SemanticKITTI . . . 83
4.5.2.3 M-City . . . 84
4.5.2.4 Blending . . . 85
4.5.2.5 Modeling the Completed Scene . . . 87

4.6 Conclusion 90

5 LiDAR Simulation **91**
5.1 Introduction 92
5.2 Related Work 94
5.3 LiDAR Simulation 96
5.3.1 New Trajectory Simulation 96
5.3.2 Firing Sequence Simulation 97
5.3.2.1 Velodyne HDL-64E 97
5.3.2.2 Firing Sequence Rays Generation 97
5.4 Experiments and Results 97
5.4.1 LiDAR Simulation with offset trajectories in PC3D, SemanticKITTI and M-City 98
5.4.1.1 PC3D-Paris 99
5.4.1.2 SemanticKITTI 103
5.4.1.3 M-City 105
5.4.2 Simulation on the Completed PC3D-Paris 106
5.4.3 LiDAR Simulation integration in a full simulation software (ANSYS sensor SDK with SCANeR Simulator) 109
5.4.3.1 Dynamic Objects Addition and Scenarios Generation 109
5.5 Conclusion 110

6 Conclusions and Future Works **113**
6.1 Conclusions 113
6.2 Future Works 114

List of Figures

1.1 Autonomous vehicles autonomy levels standard as defined by the Society of Automotive Engineers. . . . 2
1.2 A LiDAR frame shown from the vehicles' perspective (left) and a bird's eye view (right). . . . 4
1.3 REPLICA partners. . . . 5
1.4 sense-plan-act robotic paradigm 6
1.5 The L3D2 Mobile Mapping System of the center of robotics at MINES ParisTech. 8
1.6 Showing the difference between a LiDAR frame acquired in mapping configuration (left) and a LiDAR frame acquired in AV configuration (right) 9
1.7 With a mobile mapping system, we acquire a point cloud on which we perform semantic segmentation to remove the dynamic objects. In the second step, an SC algorithm is deployed to complete the missing geometry. The resulting point cloud is then used to perform 3D modeling that is consequently used for sensor simulation. Dynamic objects, such as cars and pedestrians, in the form of triangular meshes or splats (see chapters 4 and 5) can be added after the modeling step, to generate different scenarios for testing. . . . 10

2.1 The first step in our automatic reality replication pipeline consists of acquiring a dataset using a mobile mapping system (MMS) and performing manual data annotation, or semantic segmentation using deep learning methods 14
2.2 M-City dataset. To the left, we show a part of the original point cloud that is used in this thesis. In the middle, we show the full manually reconstructed and textured scene. To the right, we show the semantic information that we have along with the dataset. . . . 17
2.3 Example of the SemanticKITTI dataset [Behley et al. 2019]. . . . 18
2.4 Matterport3D dataset samples. Source [Chang et al. 2017b] 19
2.5 Prototype acquisition system used to create the PC3D dataset in the city of Paris. Sensors: Velodyne HDL32 LiDAR, Ladybug5 360° camera, Photonfocus MV1 16-band VIR and 25-band NIR hyperspectral cameras (hyperspectral data is not available in this dataset; they cannot be used in mobile mapping due to a limited exposure time). . . . 20
2.6 An example of open surfaces. The structures are scanned from the mapping system's side only 21
2.7 Full PC3D-Paris dataset in RGB. Showing the train, test and val parts, acquired in Soufflot street. . . . 22
2.8 Full PC3D-Paris dataset. Showing the semantic classes in the train, test and val parts, acquired in Soufflot street. . . . 23
2.9 PC3D-Paris training set. . . . 23
2.10 PC3D-Paris validation set. . . . 24
2.11 PC3D-Paris test set. . . . 24

2.12 Street view in PC3D-Paris. Showing the dataset in RGB, instances, and semantic classes going from left to right, respectively 25

2.13 Another street view in PC3D-Paris. Showing the dataset in RGB, instances, and semantic classes going from left to right, respectively 25

2.14 PC3D-CARLA training set. 26

2.15 PC3D-CARLA validation set. 27

2.16 PC3D-CARLA test set. 27

2.17 Paris data after removal of vehicles and pedestrians. Zones in red circles shows the interest in performing scene completion for 3D mapping: attempt to fill holes from removed pedestrians, parked cars, and to improve sampling of points in areas far from the LiDAR. 30

2.18 Given an incomplete scan of a scene, the scene completion task predicts a more complete geometry. Source [Dai et al. 2020] 31

2.19 Scene completion task for one chunk point cloud in Town1 (T_1) of CARLA test data (training on CARLA data). 32

2.20 Scene completion task for one chunk point cloud in Soufflot0 (S_0) of Paris test data (training on Paris data). 32

3.1 The second step in our simulation pipeline consists of removing dynamic objects from the scene and completing missing geometry resulting from object occlusion. 37

3.2 Self-supervised 3D Scene Completion (SC) comparing several Signed Distance Functions (SDFs) and Unsigned Distance Functions (UDFs). Small point cloud blocks are extracted from the original point cloud; random frames (for RGB-D) or scans (for LiDAR) are removed for the input, while all of the points are used for the target. All point clouds are then transformed into Distance Function (DF) representations on discretized grids; then, the network is trained to learn the completion of the representations. Our Unsigned Weighted Euclidean Distance (UWED) function eliminates the sign ambiguities resulting from SDFs while achieving higher accuracy on the SC task. 38

3.3 A 2D example of an open surface (non-closed surface). Signed distance functions (SDF) do not handle open surfaces, and even if the truncation limits the side effects, the extraction to point clouds will generate border effects. Using an unsigned distance function (UDF), eliminates the problem of negative and positive sides. However, the difficulty lies in the extraction of a point cloud. Here, the point clouds extraction from SDF and UDF follows the methods described in sections 3.3.2.1 and 3.3.2.2 respectively. We also show flipped versions of TSDF and TUDF that remove strong gradients at truncation distance. 39

3.4 Point completion networks example predictions. Source [Yuan et al. 2018] . . 42

3.5 Sparse Generative Neural Network SG-NN architecture. Source: [Dai et al. 2020] 44

3.6 Comparing the inferences of LMSCNet [Roldão et al. 2020] and SSCNet [Song et al. 2017] networks on the SemanticKITTI dataset. Source: LMSCNet [Roldão et al. 2020] 45

3.7 Example of the inference of S3CNet. Source: S3CNet [Cheng et al. 2020] . . . 46

3.8 Comparison of different implicit functions representations. From point clouds, we compute Signed Distance Functions (Hoppe and IMLS) or our Unsigned Distance Function (UWED), then we extract back a point cloud (without completion) to compare the accuracy of the extracted point clouds to the original point cloud. With UWED, we are able to extract smooth point clouds: the advantages can be seen all over the extracted point clouds, such as on the stair bar in Matterport and the window in the Paris dataset (Hoppe and IMLS dilate the geometry because of the sign and point-to-plane distance functions). 51

3.9 Sparse search 2D example. Starting from a grid encapsulating the point cloud (green quadrants), where points are converted into voxel coordinates. We use a kernel (red squares, here 5x5) of shape (Truncation $(T) * 2 + 1)^2$ here T = 2. We extract two voxels from all sides of the occupied voxels (green quadrants), which includes the current voxel and the 4 neighboring voxels $[-2, +2]$ on x and y. These coordinates contain overlaps, so we filter out the repeated ones and padded coordinates, then perform the search on the rest, where the final tensor containing the coordinates is flattened and queried from a KD-Tree. . 53

3.10 Scene completion inference of SG-NN using Hoppe, IMLS, and our UWED function on the PC3D-Paris test set (input has 10% of the original points). . 55

3.11 Scene completion inference of SG-NN using Hoppe, IMLS, and our UWED function on the Matterport3D test set (input has 50% of the original points). 55

3.12 Scene Completion inference of SG-NN using Hoppe, IMLS, and our UWED function on the PC3D-CARLA test set (input has 10% of the original points). 56

3.13 Scene completion inference of MinCompNet using Occupancy, IMLS, and our UWED function as input on the PC3D-Paris test set (input has 10% of the original points) and predicting occupancy. 56

3.14 Scene completion inference of MinCompNet using Occupancy, IMLS, and our UWED function as input on the Matterport3D test set (input has 50% of the original points) and predicting occupancy. 57

3.15 Scene completion inference of MinCompNet using Occupancy, IMLS, and our UWED function as input on the PC3D-CARLA test set (input has 10% of the original points) and predicting occupancy. 57

3.16 Scene completion performed on the target point cloud. 58

3.17 More results of scene completion performed on the target point cloud. 59

3.18 Scene completion performed on the target point cloud with point-wise semantic information taken from the nearest neighbor. 59

3.19 More results of scene completion performed on the target point cloud with point-wise semantic labels taken from the nearest neighbor. 60

4.1 The third step in our automatic reality replication pipeline consists of generating a hole-free approximation of the surface from a point cloud and rendering the generated primitives as a first step toward camera simulation. 65

4.2 Surface splatting of a scan of a human face, textured terrain, and a complex point-sampled object with semi-transparent surfaces. Source [Zwicker et al. 2001] 69

4.3 Left and right: Textured Splats rendering of point cloud. Source [Botsch et al. 2003] and [Botsch et al. 2005]. 69

4.4 Splat-based ray tracing from method [Linsen et al. 2008]. Showing the results on the Skeleton Hand, Fuel, and Happy Buddha datasets. Source [Linsen et al. 2008] 70

4.5 Qualitative evaluation of the resampling algorithm. We show the original and resampled point clouds at the top and bottom, respectively. Our resampling algorithm is able to re-distribute the point density by reducing the number of points in dense regions and increasing the number of points in sparse regions, while preserving the underlying surface structure. 75
4.6 Another qualitative evaluation of the resampling algorithm. We show the original and resampled point clouds at the top and bottom, respectively. . . . 76
4.7 Rendering results on different choices of K-nn on PC3D-Paris dataset. A small K (e.g., 10 or 20) results in holes on the surface and ground groups, while resulting in a better approximation on the non-surface and linear groups. A large K (40 to 120) results in a hole-free approximation of the surface and ground semantic groups, while creating artefacts on small structures, belonging to the linear and surface groups. 81
4.8 Rendering the different surface representations on PC3D-Paris. The first row shows the meshed scene using Poisson (left) and IMLS (right). The second shows the splatted scene using basic splats (left) and AdaSplats using KPConv semantics (right). The splatted scene using AdaSplats-GT, which contains the ground truth point-wise semantic information (left) and the original point cloud (right). 82
4.9 Rendering the different surface representations on SemanticKITTI. The first row shows the meshed scene using Poisson (left) and IMLS (right). The second shows the splatted scene using basic splats (left) and AdaSplats using KPConv semantics (right). The splatted scene using AdaSplats-GT, which contains the ground truth point-wise semantic information (left) and the original point cloud (right). 83
4.10 Rendering the different surface representations on M-City. The first row shows the manually meshed scene (left) and basic splats (right). The second shows the results of rendering AdaSplats using GT semantics (left) and the original point cloud (right). 84
4.11 Results on rendering the splatted environment with splat-wise color information. We show a rendering of the splatted Soufflot-0 and Soufflot-1 from PC3D-Paris dataset. To the left, we show the rendering without blending. To the right we show the rendering results with blending. 85
4.12 Showing the splat ray tracing results on the full PC3D-Paris dataset. The scene contains 8.7M million splats, 43 light sources and rendered at a resolution of 2560 x 1440, obtaining a real-time performance. 86
4.13 Combining the scene completion and scene modeling steps to obtain a more complete geometry. 87
4.14 Original PC3D-Paris without dynamic objects (first view). 88
4.15 Results of ray tracing the AdaSplats model generated on the completed PC3D-Paris scene (first view). 88
4.16 Original PC3D-Paris without dynamic objects (second view). 89
4.17 Results of ray tracing the AdaSplats model generated on the completed PC3D-Paris scene (second view). 89

5.1 The final step in our pipeline consists of performing real-time LiDAR simulation in the modeled scene. We first start by simulating the LiDAR in the splatted scene without completion and without dynamic objects addition. . . 93
5.2 In [Fang et al. 2020], first the scene is modeled, then CAD models of dynamic objects, such as vehicles, pedestrians and cyclists are added to generate a new scenario in which the LiDAR is simulated. Source [Fang et al. 2020] 94

5.3 In [Manivasagam et al. 2020], first the scene is modeled, then dynamic objects models collected from the real world, such as vehicles are added to generate a new scenario in which the LiDAR is simulated. Source [Manivasagam et al. 2020] 95
5.4 Point clouds used in the experiments. PC3D-Paris, SemanticKITTI, and M-City, from left to right, respectively. In red, the trajectory used for simulation. 98
5.5 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). 100
5.6 Another view of the 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). 101
5.7 Comparison of simulated LiDAR data using different reconstruction and modeling methods on SemanticKITTI. The top row: the simulation in meshed IMLS (left) and Poisson (right). The middle row: the simulation with Basic Splats (left) and AdaSplats-KPConv (right). The bottom row: the simulation with AdaSplats-GT (left) and the original point cloud (right) 103
5.8 Another comparison of simulated LiDAR data using different reconstruction and modeling methods on SemanticKITTI. The top row: the simulation in meshed IMLS (left) and Poisson (right). The middle row: the simulation with Basic Splats (left) and AdaSplats-KPConv (right). The bottom row: the simulation with AdaSplats-GT (left) and the original point cloud (right) . . . 104
5.9 Comparing the manually meshed (bottom left) and automatically splatted (bottom right). Original M-City point cloud (top image). Modeling vegetation is not an easy task and usually requires different ray-primitive intersection methods for a more accurate rendering. 105
5.10 The simulation pipeline including the scene completion, scene modeling and LiDAR simulation modules. 106
5.11 Close-up comparison. Original point cloud (top). Simulated point cloud (bottom). 107
5.12 Simulated Velodyne HDL-64E LiDAR in PC3D-Paris with scene completion. Output: accumulated point cloud. 108
5.13 Simulated Velodyne HDL-64E LiDAR in PC3D-Paris with scene completion. Output: accumulated point cloud with point-wise semantic labels. 108
5.14 Updated simulation pipeline to include meshed dynamic objects, such as pedestrians and vehicles to generate novel scenarios. 109
5.15 Scenarios generation in PC3D-Paris using the integration at ANSYS. 110

List of Tables

2.1 Point cloud datasets for semantic segmentation (SS), instance segmentation (IS), and scene completion (SC) tasks. RGB means color available on all points of the point clouds. In parentheses for semantic segmentation (SS), we show only the number of classes evaluated (the annotation can have more classes). 16

2.2 Class distribution in Paris-CARLA-3D (PC3D) dataset (in %). Columns headed by S_i stand for Soufflot from PC3D-Paris data and T_j stand for Town from PC3D-CARLA data. 28

2.3 Scene Completion results on Paris-CARLA-3D. CD is the mean Chamfer Distance over 2 000 chunks for the Paris test set (S_0 and S_3) and 6 000 chunks for the CARLA test set (T_1 and T_7). ℓ_1 is the mean ℓ_1 distance between predicted and target TSDF measured in voxel units for 5 cm voxels and $mIoU$ is the mean intersection over union of TSDF occupancy. Both metrics are computed on known voxel areas. *ori* means original point cloud, *in* is the input point cloud containing 10% of the original, *pred* is the predicted point cloud computed from the predicted TSDF. 33

2.4 Results of transfer learning for the scene completion task. CD is the mean Chamfer Distance between point clouds. ℓ_1 is the mean ℓ_1 distance between predicted and target TSDF measured in voxel units for 5 cm voxels and $mIoU$ is the mean intersection over union of TSDF occupancy. The mean is over 2 000 chunks for Paris data. *ori* means original point cloud, *in* is input point cloud (10 % of the original), *pred* for CD is the predicted point cloud (computed from predicted TSDF), *pred* for IoU and ℓ_1 is the predicted TSDF and *tar* is the target TSDF. 33

2.5 The datasets used for each task in the simulation pipeline. PC3D-Paris and PC3D-CARLA were used for the scene completion task, while only PC3D-Paris was used for the scene modeling and LiDAR simulation tasks. 34

3.1 Number of epochs and time taken to train each network on the different datasets. Time is reported in hours. 50

3.2 Mean Chamfer Distance (in cm) between the extracted point clouds from the different DFs and the original point clouds, used to compare the accuracy of each function without passing through SC networks. 52

3.3 Mean Chamfer Distance (in cm) between raw and extracted point clouds from IMLS and UWED representations while varying the parameter σ between one and four times the Voxel Size (VS). 52

3.4 SG-NN scene completion results for different signed and unsigned distance functions on three datasets: Matterport3D (RGB-D frames in indoor), PC3D-CARLA (mobile LiDAR in virtual CARLA simulator), and PC3D-Paris (mobile LiDAR in outdoor urban scene). Results are the mean Chamfer Distance computed over the test sets of the three datasets (6,000, 6,000, and 1,000 points clouds, respectively). 54

3.5 MinCompNet scene completion results of using IMLS, UWED and Occupancy grid as input to predict occupancy grids that are later extracted as point clouds and compared to the original point clouds on three datasets: Matterport3D (RGB-D frames in indoor), PC3D-CARLA (mobile LiDAR in virtual CARLA simulator), and PC3D-Paris (mobile LiDAR in outdoor urban scene). Results are the mean Chamfer Distance computed over the test sets of the three datasets (6,000, 6,000, and 1,000 points clouds, respectively). 54

4.1 Results 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) and rendering frequency (Render Freq) in Hz. 82

4.2 Results 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) and rendering frequency (Render Freq) in Hz. 84

4.3 Results 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) and rendering frequency (Render Freq) in Hz. 85

5.1 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 clouds. 100

5.2 Results of the LiDAR simulation on the PC3D-Paris using AdaSplats with ground truth semantics without resampling (top row), compared to the the simulation on the resampled model (bottom row). We report the time taken (Gen T) in seconds to generate the primitives, 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 clouds. 101

5.3 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. The AdaSplats methods include resampling 102

5.4 Cloud-to-Cloud distance (in cm) computed on PC3D-Paris for points that belong to classes of thin structures, between the simulated using AdaSplats without resampling and original point cloud 102

5.5 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. 103

5.6 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. 105

Glossary

AV *Autonomous Vehicles.* 2

BVH *Bounding Volume Hierarchy: an acceleration data structure. Used to accelerate ray tracing (ray primitive intersection) in the scope of this thesis.* 77

DF *Distance Function.* 38

IS *Instance Segmentation: the task of assigning a point-wise semantic class and a unique instance label.* 19

LiDAR *Light Detection And Ranging.* 4

MMS *Mobile Mapping System: a vehicle equipped with a collection of sensors (e.g., camera, LiDAR, RADAR, etc...) used to map the environment.* 4

NIR *Near-Infrared.* 7

SC *Scene Completion.* Deep Learning task for predicting missing geometric regions. 3

SDF *Signed Distance Function.* 38

SLAM *Simultaneous Localization And Mapping.* 3

SSC *Semantic Scene Completion.* Deep Learning task for predicting missing geometric regions and the corresponding semantic classes. 29

TLS *Terrestrial Laser Scanner: a laser scanner that is used at fixed locations in the environment.* 17

Truncation truncation of the distance function on a 3D grid is removing all of the distance values, in voxel units, from voxels lying farther than the truncation, which is a predefined threshold. 39

UDF *Unsigned Distance Function.* 38

UWED *Unsigned Weighted Euclidean Distance.* It is the euclidean distance between a point in the grid and knn in the point cloud, where the contribution of each point in the knn is weighted using a gaussian kernel. 11

Chapter 1

Introduction

Abstract

The development of autonomous vehicles (AVs) is an emerging technology that is expected to change the future of transportation. AVs can automate a repetitive task that takes up a large part of our day and free up time for more important tasks. While AVs are currently at level 3 autonomy, where human intervention is still required, they are making progress towards level 5 autonomy, where no human intervention is needed. However, testing and validating AVs in the real world is costly, risky, and time-consuming, with human validation being needed. Simulation is a way to reduce the cost and risk involved in testing AVs. The use of simulators has opened up the possibility of simulating large models such as virtual scenes and virtual simulators, which have become more realistic and are still being improved. The development of high-quality datasets is crucial in AVs development, and simulation is a cost-effective and risk-free way to collect data. This work is part of the replica project, which aims at building a realistic REPLICA of the real world and a high fidelity sensor simulation. Toward this end, we propose a fully automatic sensor simulation pipeline that can be used to test and validate autonomous vehicles' perception algorithms.

Résumé

Le développement des véhicules autonomes (VA) est une technologie émergente qui devrait changer l'avenir des transports. Les VA peuvent automatiser une tâche répétitive qui occupe une grande partie de notre journée et permet de libérer du temps pour des tâches plus importantes. Alors que les VA sont actuellement au niveau 3 d'autonomie, où l'intervention humaine est encore nécessaire, ils progressent vers le niveau 5 d'autonomie, où aucune intervention humaine n'est requise. Cependant, tester et valider les VA dans le monde réel est coûteux, risqué et prend du temps. De plus la validation humaine est nécessaire. La simulation est un moyen qui permet de réduire les coûts et les risques liés aux tests des VA. L'utilisation de simulateurs a ouvert la possibilité de simuler de grands modèles tels que des scènes virtuelles et des simulateurs virtuels, qui sont devenus plus réalistes et se sont améliorés. Le développement d'ensembles de données de haute qualité est crucial dans le développement des VA, et la simulation est un moyen rentable et sans risque de collecter des données. Ce travail fait partie du projet REPLICA, qui vise à construire une réplique réaliste du monde réel et une simulation de capteur de haute fidélité. Nous proposons une chaîne de simulation de capteurs entièrement automatique qui peut être utilisé pour tester et valider les algorithmes de perception des VA.

1.1 General Introduction

Technological advances are re-shaping our perception and understanding of the world, and changing how we approach our every day life. With every passing day, we see that technology is not only here to simply help us achieve our daily tasks anymore, but to facilitate repetitive tasks on which we spend a large portion of the day. This lost time can be used to realize more important tasks. For some people, something more important would be reading the last research papers, for others it can be planning their next vacation, etc. We all spend so much time commuting, because of the centralized industrial setting that leads to congested roads, whereas this time can be spent realizing other more important tasks. For this, as well as other issues, automation found its way to vehicles and gave the rise to autonomous vehicles (AVs) among other technologies targeting the same goal: automated repetitive tasks and free-up time for the more important. AVs are becoming increasingly possible with every passing day.

A few decades from now, manually manipulating cars might face some restrictions as the vision of self-driving cars is becoming more clear. From where we are standing now, the future of AVs looks promising thanks to the many advances happening globally across the many fields contributing to the development of AVs. At this moment, AVs are making the transition into level 3 autonomy, where they can perceive the environment and can do most driving tasks, but human intervention is still required. AVs still need improvements to reach level 5 autonomy, where no human intervention is required (see figure 1.1), while being flexible to roads and architectural changes across countries and cities. Testing AVs in the real world is risky and costly due to possible collisions with other cars and/or pedestrians, and the resources needed to operate the vehicles added to the initial assembly cost and continuous maintenance. Moreover, real-world testing requires human validation, which places a human behind the wheel in every car being tested, extending the risk of collision to the human driver and increasing the cost even further.

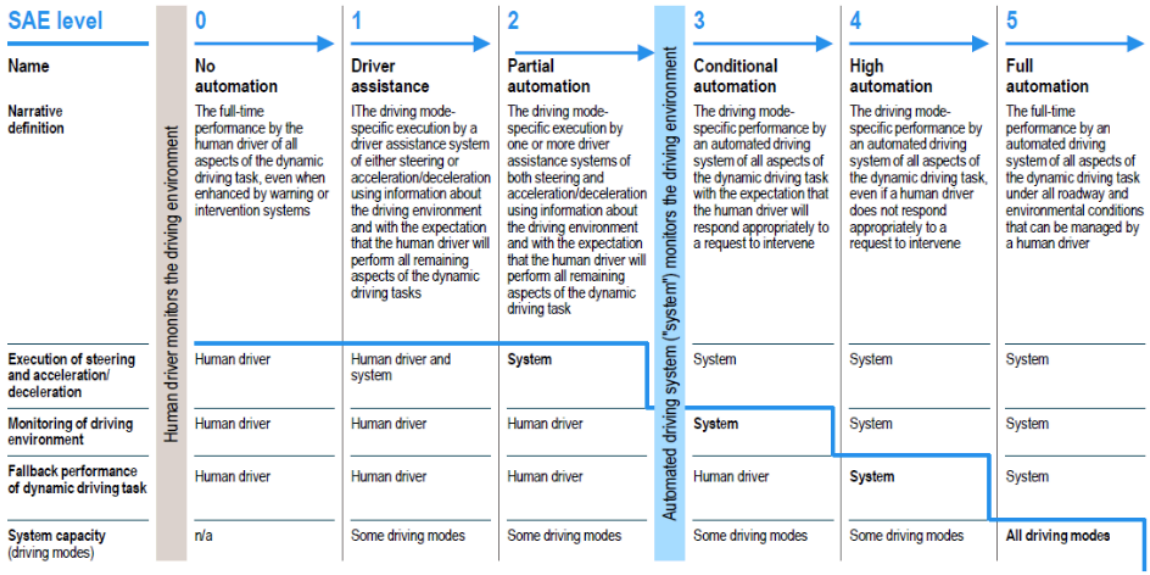


Figure 1.1: Autonomous vehicles autonomy levels standard as defined by the Society of Automotive Engineers.

The last decade witnessed significant advances in AVs influenced by many factors, such as advances in machine learning and deep learning methods targeting the automation of a large set of tasks from which, we narrow our focus down to computer vision. The advances in deep learning methods pushed the capabilities of previously available computing power

and data storage to the limit, increasing the need for more compact, portable and capable computing resources. With more data and more computing power, advances across 2D and 3D computer vision tasks witnessed an unprecedented increase in quantity and quality.

AVs development have seen increasing interest in the last decade. To test the accuracy of algorithms used by AVs, existing vehicles are usually retrofitted with sensors and computer processors to enable them to sense and analyze what is happening in their surrounding, which consequently enables them to plan and finally act based on the algorithms' output. Multiple vehicles can be retrofitted and deployed on different roads, in different cities, during different seasons, at different time of the day and in different weather conditions, to maximize the domain of scenarios covered by the deployed fleet in which the algorithms can be tested. Accelerating the development of AVs is key to achieve level 5 autonomy, where human intervention is no longer required. To do this, increasing the vehicles' fleet size is required, however, the increased number of retrofitted vehicles increases the cost and risk involved in testing AVs. Moreover, as previously stated, there is a large number of different conditions in which AVs should be tested, which makes the testing process more complex. Finally, even with the increased fleet size, it is only possible to cover a small number of cases and even a smaller number of corner cases, which are cases that rarely happen in the real world.

Large and high quality datasets, that can be used for AVs development, is one of the first requirements and is increasingly needed. Testing and validating AVs, or collecting datasets in the real world, is subject to multiple constraints, such as the time needed to increase the robustness of algorithms used by such systems, increased risk of hitting pedestrians or other cars, and high cost because of the needed resources, among others [Fang et al. 2020]. It is becoming very common to drive a fleet of cars for millions of kilometers, before deploying them in real-world scenes. To reduce the collision risk, it is necessary to associate a human control to every vehicle that is being tested, which adds another safety layer, however, it associates the failure of an autonomous system to human error and increases the cost even further.

The scientific community has looked for decades to reduce the cost and risk involved in testing AVs. The first choice that comes to mind in this context is simulation, where a process, or a physical system's behavior is imitated and optimized to reach a desired behavior. The earliest methods [Pomerleau 1988] that used simulation to test algorithms used by AVs were limited to simple tasks, such as lane detection, because of the limited computing resources. The advantage of using such simulators is that they are risk free and the cost of developing AVs is reduced to the cost of a computing hardware, which is non-negligible when compared to the cost of retrofitting a vehicle and taking into account the other costs involved in developing AVs in the real world.

With the advances in computer hardware, more powerful computing systems became more affordable and available, as a consequence, more complex systems could be developed. This opened the possibility of simulating large models, such as large virtual scenes available in game engines (GTA-V) and virtual simulators [Dosovitskiy et al. 2017a; Gschwandtner et al. 2011], which became more realistic and are still being improved as more complexity is added since more computations can be done more quickly. These simulators are virtual worlds, carefully constructed by 3D artists as a collection of virtual models corresponding to real-life objects. Using these simulators, virtual models of vehicles and sensors can be placed in the constructed scene to simulate the real-world physical models and collect the data needed for testing.

AVs development can be reduced to the algorithms used by the vehicles to complete assigned tasks, such as perception algorithms, namely, semantic segmentation (SS) [Landrieu et al. 2018; Thomas et al. 2019], object detection [Weng et al. 2020; Yin et al. 2021], scene completion (SC) [Song et al. 2017; Dai et al. 2020] and SLAM [Campos et al. 2021; Deschaud 2018], to name a few. Developing such algorithms can be achieved through the simulation

of a virtual model of the vehicle equipped with the set of sensors used by AVs, including, but not limited to LiDARs (see Figure 1.2 for an example of a LiDAR frame) and cameras. These sensors provide rich information about the scene and can be used to understand the surrounding of the vehicle and make decisions based on their output. The ability to simulate these sensors enables the generation of corner cases to test the accuracy of the output of the algorithms used by AVs under different circumstances. One drawback of using handcrafted simulators is the domain gap they introduce. This gap is caused by the perfect geometry resulting from the manual meshing done by the 3D artists and the difficulty in approximating complex real-world objects, such as vegetation, resulting in changes in the simulated scene, which affects the output of the algorithms.

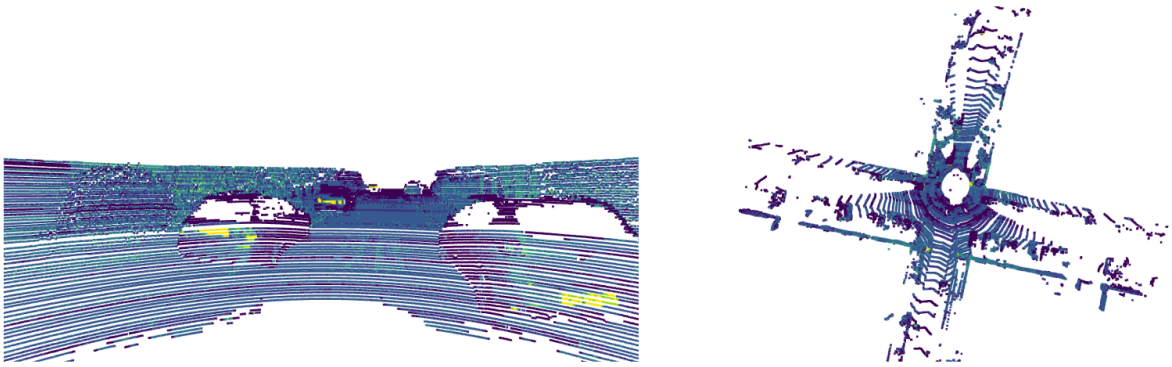


Figure 1.2: A LiDAR frame shown from the vehicles' perspective (left) and a bird's eye view (right).

This gap made the shift toward more realistic simulators necessary, and methods leveraging real-world data, in the form of point clouds, which are accumulated LiDAR frames along the acquisition path, started appearing in a move to reduce the gap [Fang et al. 2020; Manivasagam et al. 2020]. Indeed, leveraging real-world data (e.g., point clouds acquired using a LiDAR sensor) helps in making the simulations more realistic, since the constructed environment is trying to replicate the real world. However, data collected in urban regions usually contain a large number of temporarily static object, such as parked vehicles on the sides of the roads, which results in occluded regions that are inaccessible by the LiDAR pulses. Moreover, the density of the point cloud is proportional to the distance between the LiDAR and the surface, which becomes sparser in farther regions. Mobile Mapping System (MMS) fleets are usually deployed to perform multiple acquisitions of point clouds representing the same street, in order to fill the missing regions and densify sparser areas. An alternative is to use scene completion (SC) methods to predict missing geometry and create the virtual environment on the completed scene.

Realistic sensor simulation in a virtual environment already provides a safer alternative to testing AVs in the real world. However, accelerating the simulation is crucial to reduce the testing and deployment time. A wide variety of simulation engines exist, however, current reliable engines performing automatic modeling and physics-based simulation leveraging real-world data are limited to few processes that can be executed on CPU, which can be used to parallelize the sensor simulation algorithm, taking into account the physical model of the sensor. Simulating the LiDAR and camera models can be achieved by using ray casting, where a ray is cast from the origin of the sensor (the sensor position) through the virtually constructed environment, then a ray-primitive intersection algorithm returns the intersection point or color, for the LiDAR or camera sensors simulation, respectively. Along with the advances in computer hardware, GPUs are becoming more powerful and more affordable as well, consequently, GPU resources can be used to parallelize the ray-casting algorithm to obtain real-time sensor simulation.

1.2 REPLICA Project

This work is in collaboration with *ANSYS* and *MINES ParisTech - Armines*. Both, among other partners, such as *Renault, PSA and AVSimulation* (see Figure 1.3) are part of the REPLICA project that aims at creating digital twins of real-world scenes and accelerate the simulation of the sensors used by AVs in realistically built virtual environments. This project was built with the target of opening the possibility for massive AV sensor simulation, pushing AV development faster toward level 5 autonomy, while reducing the risk and cost and increasing the simulation realism. Toward this goal, the project identified four major steps:

- Automatic creation of high-definition segmented maps in the form of point cloud datasets
- Automatic creation of test cases, including corner cases
- Creation of dynamic scenarios representing the identified cases
- Realistic 3D modeling and massive sensor simulation

Figure 1.3: REPLICA partners.

This thesis tackles the fourth step in which we aim to develop a fully automatic pipeline, which is able to create a virtual environment, as close as possible to real-world scenes, that is later used to simulate sensors used by AVs, and hence generate new driving scenarios and test the behavior of AVs.

1.3 Thesis Scope

AVs development can be reduced down to the development of the collection of algorithms used by AVs and can be summarized in a paradigm containing the essential steps, namely, sense, plan and act (see Figure 1.4), that imitates the human driving activity. Perception falls under the sensing step, which takes input from the collection of sensors used by AVs and attempts to make sense of these inputs to understand the surrounding of the vehicle using a collection of algorithms, such as semantic segmentation and object detection, among others. The input data used for such algorithms comes in many forms, such as 2D images acquired using monocular cameras, or 3D point clouds acquired using LiDAR sensors that give a 3D representation about the environment. Using the collection of sensors and algorithms, the AV is able to see and make sense out of its surrounding. The output of the perception algorithms is then used to plan and finally act based on the two preceding steps.

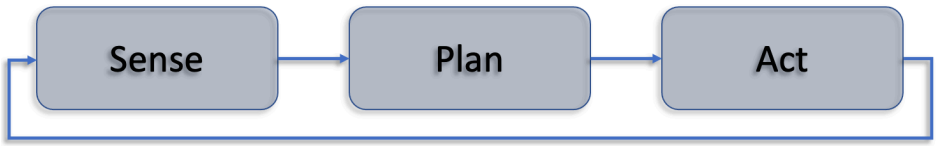


Figure 1.4: sense-plan-act robotic paradigm

1.3.1 Employed Sensors

In our work, we are interested in the monocular camera and LiDAR sensors, consequently, we use these two sensors only throughout this work, however, other sensors, such as RADAR, can be easily integrated.

The monocular camera is a passive sensor that provides rich dense information about the environment in which the AV is operating. Multiple monocular cameras can be placed on the vehicle to provide a 360° view of the surrounding of the car. However, monocular cameras limit the output information provided to the vehicles to 2D, therefore depth information are not directly available using monocular cameras.

The Light Detection And Ranging (LiDAR) sensor is an active sensor that provides sparse 3D information about the surrounding of the AV and outputs the representation of the scene in the form of a point cloud that can be used to deduce the distance to objects inside the scene. Leveraging the data coming from both sensors, their advantages can be combined to overcome the weaknesses introduced by each sensor. This combination leads to obtaining a more accurate representation of the scene, increasing data redundancy, and the accuracy of the output of perception algorithms.

1.3.2 Required Tasks

There exist a large collection of AVs tasks in the different steps of the robotic paradigm. Our work focuses on the use of the LiDAR sensor to enable a more realistic virtual testing of the algorithms used throughout the whole paradigm. To do this, we create a digital copy of real-world environments, by automatically modeling a hole-free surface representation, using point clouds collected from the real world. We validate our scene modeling technique by simulating perception sensors, namely, LiDAR and camera. Our pipeline can be easily extended to test the algorithms used in the other steps as well. Below, we detail the list of tasks that are involved in the creation of our realistic reality replication pipeline for sensor simulation.

- Simultaneous Localization and Mapping (SLAM) is the task of using perception sensors for building a map of the environment and localizing and re-localizing the vehicle. This is used in the dataset creation, which is the first step in our pipeline.

- Semantic Segmentation (SS) is the task of assigning pixel-wise (on 2D camera images), or point-wise (on 3D LiDAR point clouds) semantic information. We deploy an SS algorithm to segment the input point cloud and then use the semantic classes to remove dynamic objects, or temporarily static objects from the scene. Moreover, semantic information play a key role in the modeling step of our pipeline.

- Scene Completion (SC) is the task of completing the 3D geometry from incomplete, sparse scans. We use SC to complete the missing geometries caused by the dynamic objects removal

- 3D Scene Modeling is the task of generating a hole-free approximation of a 3D surface, from a set of points. The modeled scene is used as the virtual world in which the sensors

can be simulated. A resampling step is included in the modeling algorithm to overcome the data sparsity caused by the physical model of the LiDAR.

- Sensor Simulation is the task of simulating the physical model of a sensor. In this work, we will show results on simulating LiDAR sensors, knowing that our model could be used to simulate the camera sensor as well.

1.4 Team

1.4.1 Center of Robotics

CAOR, the center of robotics at *MINES ParisTech* focuses its interest on basic and applied research and tackles a wide variety of tasks that are essential for the development of AVs. Researchers at the center address problems that fall in the different steps of the sense-plan-act paradigm ranging from perception tasks to planning and complex kinematic and dynamic models control.

1.4.1.1 NPM3D Team and L3D2 Mobile Mapping System

The NPM3D "Nuage de Points et Modélisation 3D", which translates to "Point Cloud and 3D Modeling" team has experts in scanning large environments using MMSs, such as the L3D2 system [Goulette et al. 2006] (see Figure 1.5). The system is equipped with a 360° poly-dioptric camera Ladybug5 (composed of 6 cameras), a Velodyne HDL-32E LiDAR sensor, Photonfocus MV1 16-band VIR and 25-band NIR hyperspectral cameras. Moreover, a LiDAR SLAM system based on IMLS-SLAM [Deschaud 2018] uses only LiDAR data for the map building.

1.4.2 ANSYS

ANSYS is a world leader in multi-physics engineering simulation. The vision and perception team at *ANSYS* invests in real-time, high-fidelity sensor simulation, more specifically camera, LiDAR and Radar sensors. Prior to this work, the team worked on manual, or semi-automatic methods to construct virtual environments to enable sensor simulation. They achieve real-time sensor simulation by simulating the complete physics-based sensor model. However, simulating in manually constructed environments has its limitations and cannot be used directly for testing algorithms used by AVs. By integrating this work, we are able to reduce the domain gap through using real-world data to create realistic virtual environment.

1.5 Identified Problems

In this thesis, we started by tackling the problem of accurate scene modeling for camera simulation. However, while working on improving the scene modeling, we identified several other problems that we introduce below.

As a first step, we started by addressing the problem of accurate 3D scene modeling of the static background of the scene, originally represented in the form of point cloud, that would result in realistic rendering and re-illumination of the scene. Using previous 3D point cloud modeling (in this work, we use splats as geometric primitive), or surface reconstruction techniques result in a poor representation quality of the point cloud surface.

When working on the modeling step, we encountered another problem, which is the presence of missing regions from the scene, caused by dynamic objects removal. This could be

Figure 1.5: The L3D2 Mobile Mapping System of the center of robotics at MINES ParisTech.

partially solved by driving the MMS multiple times in the same environment and concatenating the different acquisitions to obtain a more complete point cloud. However, multiple sweeps acquisition is time and money consuming, and only partially solves the problem.

Testing the accuracy of the scene modeling method is not straightforward as rendering the modeled scene can only give us a qualitative evaluation. To overcome this limitation, a LiDAR sensor can be simulated inside the modeled scene and a point cloud can be obtained on which an error metric can be calculated with respect to the original point cloud, providing us with a quantitative evaluation. Previous methods on 3D modeling with splats parallelize the ray casting process using CPUs, which provide a low number of cores that can be used in parallel. However, being limited to a few rays that can be cast in parallel places limitations on real-time simulation.

1.6 Proposed Solutions

Following the identified problems, in this section, we detail the proposed solutions that we will detail in the following chapters.

For the discussed reasons, in our work, we first introduce a new 3D point clouds dataset that contains a real and a synthetic part. The real part was collected in the heart of Paris using a MMS, it contains a wide variety of objects and was manually annotated. The synthetic part leverages the CARLA [Dosovitskiy et al. 2017a] simulator in which we simulate the same MMS that was used to acquire the real part and acquire the synthetic point clouds. The synthetic part contains a large amount of points that are freely annotated, since the simulator contains information about the intersected geometry. We leverage the large synthetic dataset and the real dataset to train and finetune the SS and SC tasks, respectively, in a deep

learning framework. LiDAR sensors mounted on AVs are usually configured to give as much information about the surrounding of the vehicle without caring much about objects on higher elevation, such as building facades, we call this LiDAR configuration "AV configuration", or AV mode. The LiDAR used to acquire the dataset was configured in mapping mode (see Figure 1.6) to maximize the acquisition of geometric surfaces in all directions.

Previous 3D modeling techniques were designed to work on carefully scanned dense 3D objects. However, they fail in modeling large scenes represented by a sparse collection of points, for this, we introduce a 3D modeling technique that uses splats, which are oriented 2D disks, as geometric primitives. We use semantic information along with local geometric cues to change the orientation and radii of the splats resulting in a high-quality hole-free approximation of the surface.

Missing regions are the result of occlusions caused by the physical model of the LiDAR, which limits the points return and laser surface intersection to the point of view of the LiDAR. Data sparsity is proportional to the distance between the LiDAR and the surface being scanned. To overcome this problem, we propose the use of the SC task to predict the missing geometry. Moreover, we introduce a novel method to overcome the data sparsity problem, which we embed in the 3D modeling algorithm and obtain a more uniformly distributed point cloud.

Testing AVs in a virtual environment is essential for accelerating AVs deployment. However, not achieving real-time sensor simulation limits the use of the constructed 3D environment. Having said that, we detail how we achieve real-time ray casting using the GPU architecture with an acceleration data structure that we use to accelerate the ray-splat intersection. Moreover, we demonstrate how we can add dynamic objects on the static background to generate new scenarios, while achieving real-time rendering and LiDAR simulation.

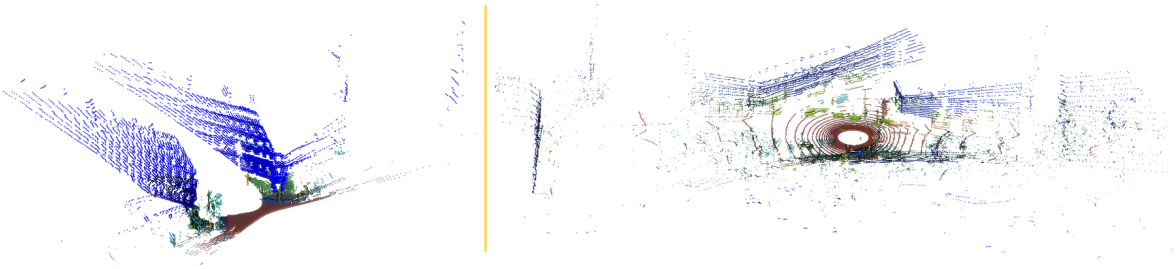


Figure 1.6: Showing the difference between a LiDAR frame acquired in mapping configuration (left) and a LiDAR frame acquired in AV configuration (right)

1.7 Automatic Reality Replication Pipeline

With the proposed solutions, we introduce a pipeline that is able to automatically produce a realistic replication of real-world scenes, unlocking a massive high-fidelity sensor simulation.

The pipeline (see Figure 1.7) consists of five main steps that we list below:

- Point cloud acquisition
- Semantic Segmentation
- Scene Completion
- 3D modeling
- Real-Time Sensor Simulation

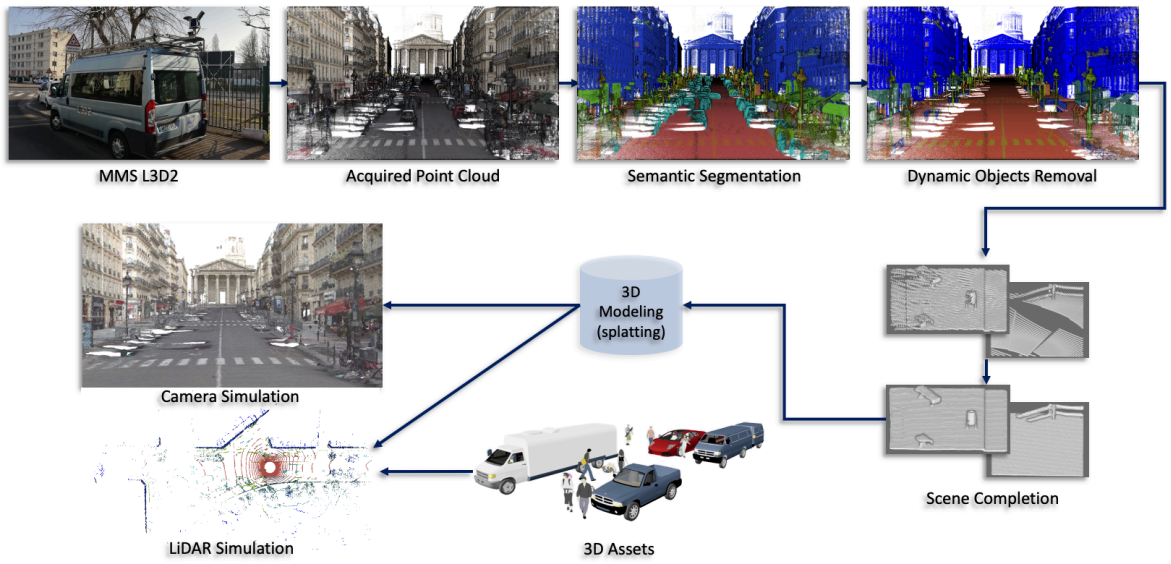


Figure 1.7: With a mobile mapping system, we acquire a point cloud on which we perform semantic segmentation to remove the dynamic objects. In the second step, an SC algorithm is deployed to complete the missing geometry. The resulting point cloud is then used to perform 3D modeling that is consequently used for sensor simulation. Dynamic objects, such as cars and pedestrians, in the form of triangular meshes or splats (see chapters 4 and 5) can be added after the modeling step, to generate different scenarios for testing.

The acquired point cloud is used to perform SS, which gives point-wise semantic information that are used in two steps throughout the pipeline (dynamic object removal and 3D modeling). After obtaining the semantic classes, we remove dynamic objects to obtain the static background, since our work focuses only on accurate background modeling. The obtained static background contains holes because of objects occlusions (e.g., parked cars), so we use SC to complete the missing geometry and decrease the data sparsity in the point cloud. After completing the point cloud, we perform 3D modeling, where we generate a hole-free approximation of the surface, which is finally used to perform real-time sensor simulation.

1.8 Thesis Structure

In this section, we list the contributions of the thesis:

- To generate a replica of a real-world scene, we first need to have a dataset that we can use to perform the tests. To this end, we collect a dataset using our MMS and call it Paris-CARLA-3D [Deschaud et al. 2021]. This dataset contains a real part collected in Paris and a synthetic part collected using a virtual MMS, having the same configuration as our real-world MMS, in CARLA simulator [Dosovitskiy et al. 2017a]. The dataset is introduced and detailed in chapter 2
- When working on point clouds acquired using an MMS, the occluded regions are represented as missing areas in the 3D space, which is not the case in the real world. To solve this problem, we use a virtually cost-free approach, which is the scene completion (SC) and introduce UWED, an unsigned distance function that improves the completion over all other distance functions used for the task of SC. Although we do not complete all the missing regions, we still obtain the best geometry completion among other approaches. Our SC approach is introduced and detailed in chapter 3
- Reconstructing the point cloud surface is not an easy task and while there exists methods that tackle this problem, high-quality surface reconstruction from such noisy data is still a challenge. For the task of scene representation, we introduce AdaSplats and a real-time splats rendering algorithm, which is detailed in chapter 4. AdaSplats generates a hole-free approximation of the surface using splats [Linsen et al. 2008] that can be adapted following the semantics of the local region and other geometric cues.
- Real-time LiDAR simulation is an important task. We develop a real-time physics-based LiDAR simulation method, leveraging the high parallelization capability of GPUs, that can be used in the splats model generated in the modeling step. Our LiDAR simulation method is detailed in chapter 5.

Our pipeline is fully automatic and can quickly replicate a real-world scene, without the need of manual reconstruction, which takes months and does not obtain the same level of realism as our method.

1.9 Contributions

Our contributions resulted in the following publications and submissions:

- Deschaud, J.E., Duque, D., Richa, J.P., Velasco-Forero, S., Marcotegui, B. and Goulette, F., 2021. Paris-CARLA-3D: A real and synthetic outdoor point cloud dataset for challenging tasks in 3D mapping. Remote Sensing, 13(22), p.4713.
- Richa, J.P., Deschaud, J.E., Goulette, F. and Dalmasso, N., 2022. AdaSplats: Adaptive Splatting of Point Clouds for Accurate 3D Modeling and Real-Time High-Fidelity LiDAR Simulation. Remote Sensing, 14(24), p.6262.
- Richa, J.P., Deschaud, J.E., Goulette, F. and Dalmasso, N., 2022. UWED: Unsigned Distance Field for Accurate 3D Scene Representation and Completion. arXiv preprint arXiv:2203.09167.

Additionally, our work resulted in two communications that we list below:

- Richa, J.P., "Splats Ray Tracing of Outdoor Point Clouds for Autonomous Vehicles Sensors Simulation.", GDR IG-RV Rendering Working Group, 7th July 2020. http://gtrendu.blogspot.com/
- Richa, J.P., Dalmasso, N., "Urban Scene Completion and Modeling From 3D Point Clouds and Massive LiDAR Simulation for Autonomous Vehicles.", ANSYS TechCon, 17 November 2022.

Chapter 2

Datasets

Contents

2.1 Introduction **14**
2.2 Related Work **16**
2.3 Employed Datasets **17**
2.3.1 M-City 17
2.3.2 SemanticKITTI 18
2.3.3 Matterport3D 18
2.4 Paris-CARLA-3D Dataset Creation **19**
2.4.1 Acquisition Platform and Used Sensors 20
2.4.2 Real Paris 21
2.4.3 Synthetic CARLA 26
2.5 Paris-CARLA-3D Dataset Properties **27**
2.5.1 Classes Statistics 27
2.5.2 Split for training 28
2.5.3 Interest in having both synthetic and real data 29
2.5.4 Transfer learning 29
2.6 Scene Completion Task **29**
2.6.1 Task Protocol 30
2.6.2 Experiments: setting a baseline 31
2.6.2.1 Qualitative results 32
2.6.2.2 Quantitative Results 32
2.6.2.3 Transfer learning with scene completion 33
2.7 Tasks per dataset **34**
2.8 Conclusion **34**

Abstract

The first step in our pipeline consists of using a 3D point cloud dataset that can be used for the different sub-tasks we want to realize. In this chapter, we list the datasets that we are going to use throughout the thesis and introduce a new dataset, containing a real part acquired using our MMS and a synthetic part acquired by a virtual replica of our MMS, in mapping configuration. After acquisition, the dataset is manually annotated, which offers ground truth point-wise semantic information. In the rest of this thesis, we make use of this dataset, along with others, for the scene completion, scene modeling and LiDAR simulation tasks.

Résumé

La première étape de notre pipeline consiste à utiliser un ensemble de données de nuages de points 3D qui peuvent être utilisé pour les différentes sous-tâches que nous souhaitons réaliser. Dans ce chapitre, nous listons les jeux de données que nous allons utiliser tout au long de la thèse et introduisons un nouveau jeu de données, contenant une partie réelle acquise à l'aide de notre MMS et une partie synthétique acquise par une réplique virtuelle de notre MMS, en configuration de cartographie. Après l'acquisition, l'ensemble de données est annoté manuellement, ceci offre des informations sémantiques de vérité de terrain. Dans le reste de cette thèse, nous utilisons ces données, ainsi que d'autres, pour les tâches de complétion de scène, modélisation de scène et simulation du LiDAR.

2.1 Introduction

In this chapter, we detail the first step in our automatic reality replication pipeline for sensor simulation (see Figure 2.1), which is data acquisition.

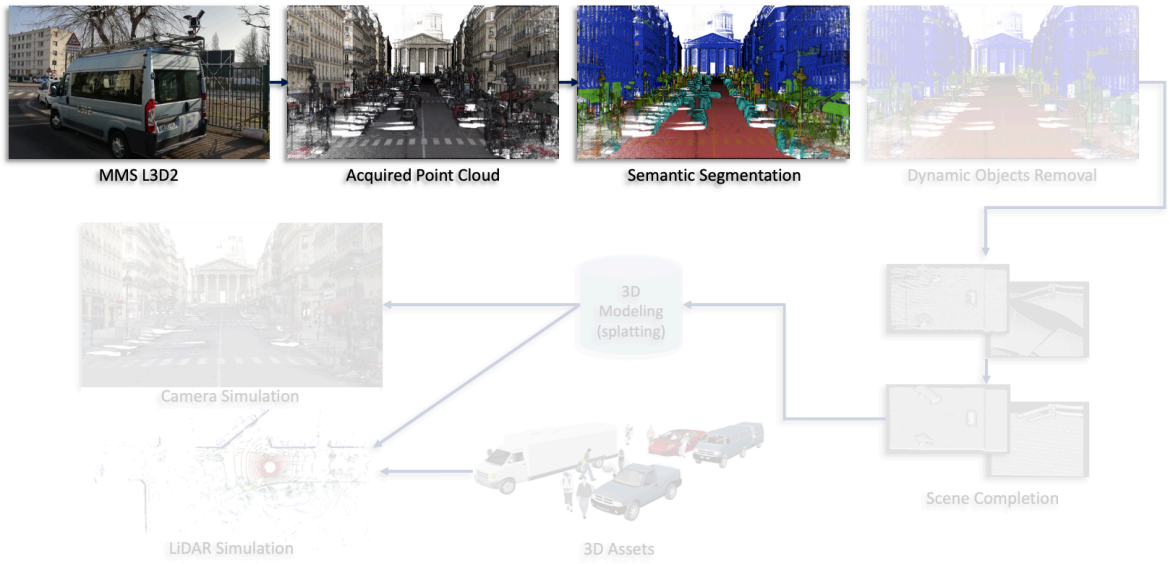


Figure 2.1: The first step in our automatic reality replication pipeline consists of acquiring a dataset using a mobile mapping system (MMS) and performing manual data annotation, or semantic segmentation using deep learning methods

3D point cloud datasets are gaining more interest with the maturity of 3D vision and perception algorithms and the need for bigger datasets is increasing to satisfy the needs of

data hungry deep learning methods. 3D data acquisition can be accomplished with three main approaches, using different sensors:

- photogrammetry - Structure from Motion and Multi-View Stereo from photos
- RGB-D or structured light scanners for small objects or indoor scenes
- static or mobile LiDARs for outdoor scenes

The advantage of mobile LiDARs lies in their ability to acquire large volumes of data with high precision. There are already many datasets published in the first two families. However, fewer are available on outdoor mapping and none contains real and synthetic parts with semantic classes mapping between real and synthetic data, which can be leveraged to reduce the domain gap resulting from the difference in geometry between both parts (see section 2.5.3). There are still many challenges in the ability to analyze outdoor environments from mobile LiDARs. Indeed, the data contains a lot of noise mainly caused by the physical model of the sensor, which leads to significant local anisotropy, and by the imperfect localization system. Moreover, the data contains missing regions caused by objects occluding other objects in the scene, such as parked vehicles causing missing geometry in the static background.

Real-world 3D point cloud datasets provide rich scene information and can be used to test Autonomous Vehicles (AVs) algorithms. The first step (data acquisition) in our pipeline is vital for a high-fidelity simulation. A better acquisition quality and a higher diversity result in better modeling, leading to improved sensor simulation. Moreover, a higher objects diversity results in more classes to train SS algorithms, and/or a higher geometric diversity to train and test SC algorithms. Furthermore, real-world dataset annotation costs a lot of time and money, therefore, choosing an urban area containing a high object diversity leads to a high annotation cost. Having said that, we need to maximize the quality and number of annotated classes, while increasing the dataset size to make more data available and reducing the annotation cost. For this, we leverage the CARLA simulator [Dosovitskiy et al. 2017a] to generate a large synthetic dataset, which augments the size of the dataset cost-free.

Synthetic datasets introduce a domain gap, because of the handcrafted geometry available in manually constructed environments, not based on real data. We introduce a semantic class mapping between the synthetic and real parts in the dataset, making it possible to use the synthetic part for pre-training before fine-tuning deep learning methods on the real part. This transfer learning can help reducing the domain gap introduced by the handcrafted environments available in CARLA.

3D point clouds are becoming increasingly important in many current and potential applications, such as semantic segmentation (SS), scene completion (SC), object detection, city mapping, road infrastructure management and construction of HD-maps for AVs. An important step in testing AVs is the nature and the size of the used dataset. The bigger the dataset, and the higher the diversity available in such datasets, the more scenarios are available for testing. A dataset acquired on a highway, or in a village, results in a lower geometric diversity and a reduced complexity in the acquired scene, which leads to testing on a reduced domain.

We introduce a new annotated dataset containing a real and a synthetic part acquired in outdoor environments, using a real MMS and its virtual replica, in a real and a virtual (CARLA) environment, respectively. Moreover, we perform experiments and provide a baseline for the Scene Completion (SC) task on that dataset.

The main contributions of this chapter are:

- A new dataset, called Paris-CARLA-3D (PC3D in short) containing synthetic and real point clouds of outdoor environments.
- The protocol and experiments with a baseline on the SC task using this dataset.

2.2 Related Work

The increasing demand on 3D point cloud data is leading the research community to join efforts and make datasets more and more available. Several public datasets exist and can be organized in two main categories: indoor and outdoor. Table 2.1 shows a list of 3D point clouds datasets.

First, in Table 2.1, we separate the datasets according to the environment: indoor and outdoor datasets mainly acquired using RGB-D and LiDAR sensors, respectively. For outdoor datasets, we also made the distinction between perception datasets that are used for AVs perception tasks and mapping datasets that are used for environment mapping tasks. For example, the well-known SemanticKITTI [Behley et al. 2019] consists of a set of LiDAR scans from which it is possible to produce a dense point cloud of the environment with the poses provided by SLAM or GPS/IMU, but the associated tasks, such as SS or SC, are only centered on a LiDAR scan for the perception of the vehicle. This is very different from the dense point clouds of mapping systems such as Toronto-3D [Tan et al. 2020] or our Paris-CARLA-3D dataset.

Table 2.1 thus shows that Paris-CARLA-3D is the only dataset to offer annotations and protocols that allow for working on semantic, instance, and SC tasks on dense point clouds for outdoor mapping.

Scene	Type	Dataset (Year)	World	# Points	RGB	Tasks		
						SS	IS	SC
Indoor	Mapping	SUN3D [Xiao et al. 2013] (2013)	Real	8M	Yes	✓(11)	✓	
		SceneNet [McCormac et al. 2017] (2015)	Synthetic	-	Yes	✓(11)	✓	✓
		S3DIS [Armeni et al. 2016] (2016)	Real	696M	Yes	✓(13)	✓	✓
		ScanNet [Dai et al. 2017a] (2017)	Real	5 581M	Yes	✓(11)	✓	✓
		Matterport3D [Chang et al. 2017a] (2017)	Real	24M	Yes	✓(11)	✓	✓
Outdoor	Perception	PreSIL [Hurl et al. 2019a] (2019)	Synthetic	3 135M	Yes	✓(12)	✓	
		SemanticKITTI [Behley et al. 2019] (2019)	Real	4 549M	No	✓(25)	✓	✓
		nuScenes-Lidarseg [Caesar et al. 2020] (2019)	Real	1 400M	Yes	✓(32)	✓	
		A2D2 [Geyer et al. 2020] (2020)	Real	1 238M	Yes	✓(38)	✓	
		SemanticPOSS [Pan et al. 2020] (2020)	Real	216M	No	✓(14)	✓	
		SynLiDAR [Xiao et al. 2021] (2021)	Synthetic	19 482M	No	✓(32)		
		KITTI-CARLA [Deschaud 2021] (2021)	Synthetic	4 500M	Yes	✓(23)	✓	
	Mapping	Oakland [Munoz et al. 2009] (2009)	Real	2 M	No	✓(5)		
		Paris-rue-Madame [Serna et al. 2014] (2014)	Real	20M	No	✓(17)	✓	
		iQmulus [Vallet et al. 2015] (2015)	Real	12 M	No	✓(8)	✓	
		Semantic3D [Hackel et al. 2017] (2017)	Real	4 009M	Yes	✓(8)		
		Paris-Lille-3D [Roynard et al. 2018a] (2018)	Real	143M	No	✓(9)	✓	
		SynthCity [Griffiths et al. 2019] (2019)	Synthetic	368M	Yes	✓(9)		
		Toronto-3D [Tan et al. 2020] (2020)	Real	78M	Yes	✓(8)		
		TUM-MLS-2016 [Zhu et al. 2020] (2020)	Real	41M	No	✓(8)		
		Paris-CARLA-3D (2021)	**Synthetic+Real**	**700+60M**	**Yes**	✓(23)	✓	✓

Table 2.1: Point cloud datasets for semantic segmentation (SS), instance segmentation (IS), and scene completion (SC) tasks. RGB means color available on all points of the point clouds. In parentheses for semantic segmentation (SS), we show only the number of classes evaluated (the annotation can have more classes).

We have listed only datasets available in the form of a point cloud and not datasets such as NYUv2 [Silberman et al. 2012], which do not contain the poses (trajectory of the RGB-D sensor) that do not allow for producing a dense point cloud of the environment. We are also only interested in terrestrial datasets, which is why we have not listed aerial datasets such as DALES [Varney et al. 2020], Campus3D [Li et al. 2020] or SensatUrban [Hu et al. 2021].

2.3 Employed Datasets

Testing AV tasks is usually done on datasets collected from real-world environments, simulated environments, or controlled real-world environments, which are closed areas configured specifically for new scenarios generation to test AVs in specific cases. Throughout this thesis, we use several datasets for the different tasks. More specifically, we make use of Paris-CARLA-3D (PC3D) (see section 2.4) and Matterport3D for the SC task, while we use PC3D [Deschaud et al. 2021], M-City and SemanticKITTI [Behley et al. 2019] for the scene modeling and sensor simulation tasks.

2.3.1 M-City

M-City is an in-house dataset created at ANSYS. M-City was acquired in an AV testing site in Michigan using a Terrestrial Laser Scanner (TLS), namely, the Leica RTC360 at fixed points in a controlled environment, without dynamic objects. The acquired point cloud was then used by 3D artists to perform manual surface reconstruction, which took 4 months. In Figure 2.2 we show the full reconstructed and textured dataset in the middle and a part containing 17.5 million points from the original point cloud, which is around 25% of the original point cloud and used in this thesis for the surface modeling and sensor simulation tasks.

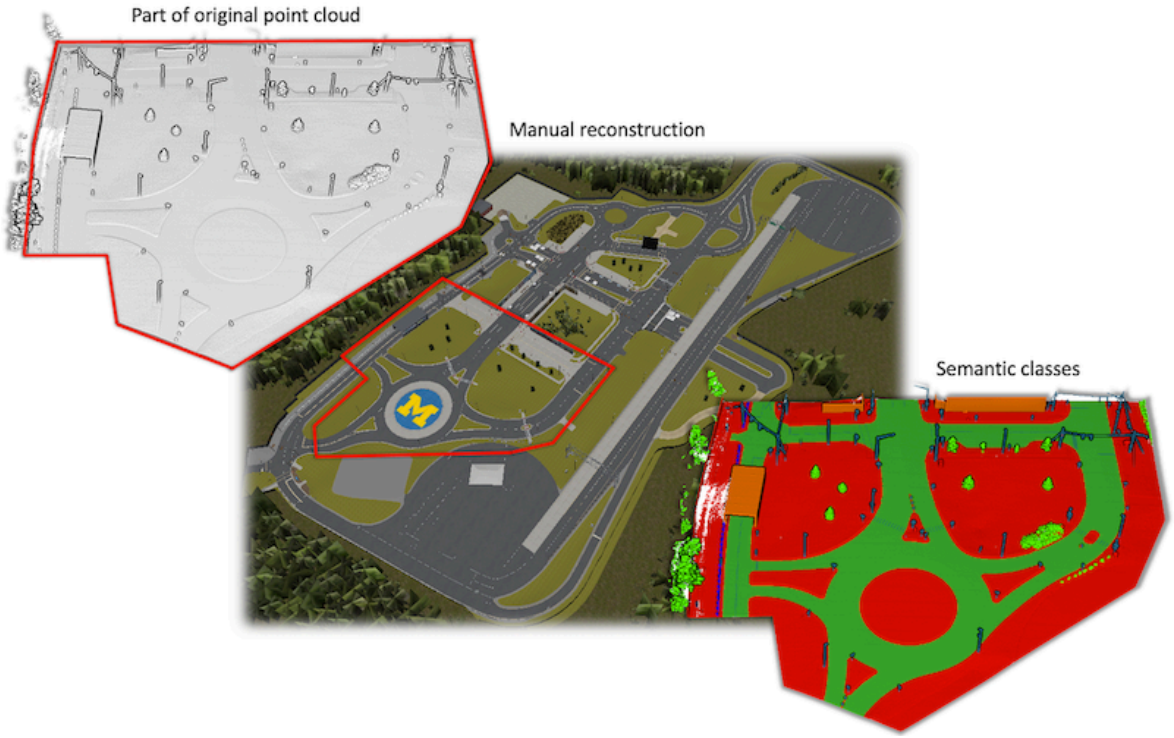


Figure 2.2: M-City dataset. To the left, we show a part of the original point cloud that is used in this thesis. In the middle, we show the full manually reconstructed and textured scene. To the right, we show the semantic information that we have along with the dataset.

2.3.2 SemanticKITTI

SemanticKITTI [Behley et al. 2019] is an outdoor point cloud dataset that was acquired using a Velodyne HDL-64 LiDAR in AV configuration in the suburban roads around Karlsuhe. The dataset was acquired using a sensor (Velodyne HDL-64) that is different from PC3D (Velodyne HDL-32) and configured in AV mode rather than mapping mode, which changes the scanning pattern of the LiDAR and provides more information about the immediate surrounding of the vehicle rather than building facades and higher structures. A part of the dataset contains ground truth point-wise semantic labels obtained from manual annotation consisting of 25 classes.

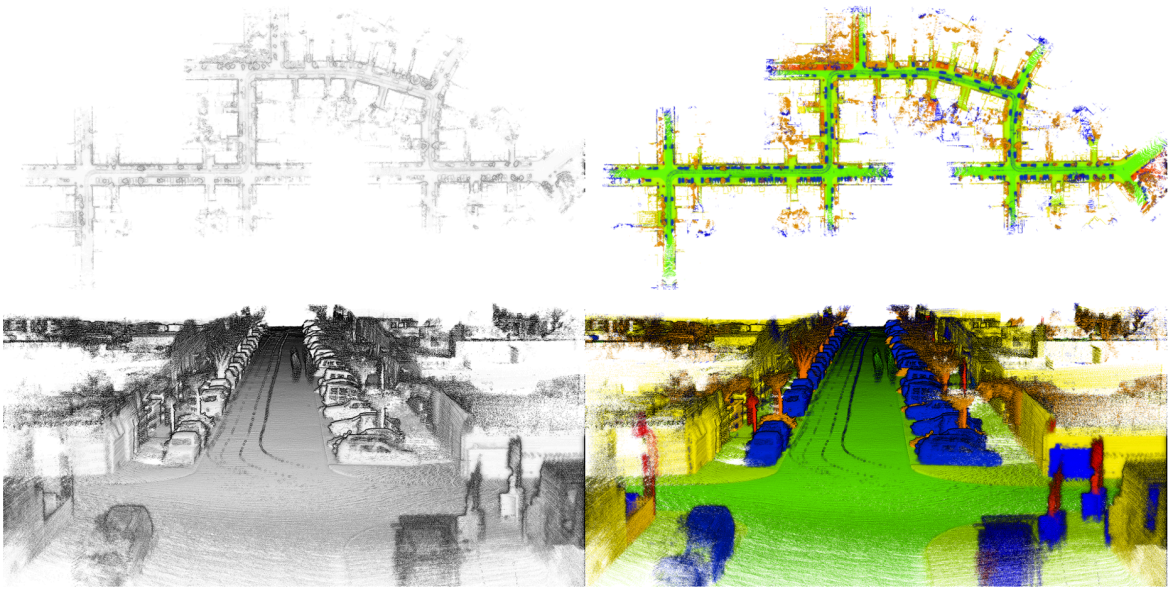


Figure 2.3: Example of the SemanticKITTI dataset [Behley et al. 2019].

2.3.3 Matterport3D

Matterport3D [Chang et al. 2017b] is an indoor RGB-D dataset covering 90 buildings that was acquired using a camera rig mounted on a tripod consisting of three color and three depth cameras pointing from slightly down to slightly up. The rig rotates around the vertical axis to 6 different orientations and stops at each orientation to acquire an HDR photo from each of the three color cameras. The three depth cameras continuously acquire the data while rotating, then the depth images are aligned with the color images of the same size.

Figure 2.4: Matterport3D dataset samples. Source [Chang et al. 2017b]

2.4 Paris-CARLA-3D Dataset Creation

The creation of the dataset follows the need of a point cloud that can be used to perform three main AV tasks, namely, SS, Instance Segmentation (IS) and SC. Moreover, as a part of the REPLICA project, the dataset should also be used for realistic 3D scene modeling in which we can later perform sensor simulation. Furthermore, the dataset contains a real and

a synthetic part acquired using a real-world MMS and its virtual replica. The choice of using the same MMS for both parts was made to reduce the domain gap between synthetic and real-world datasets, which is not provided by other datasets. Most of the existing outdoor point cloud datasets are usually collected with the LiDAR in AV configuration [Behley et al. 2019], where the field of view is limited to the surrounding of the vehicle, so the data can be used to perform AV tasks. This type of datasets includes points from objects that are few degrees above the vehicle's height and does not provide any information about objects that are above this level, such as building facades. Our dataset was collected in mapping configuration, with the LiDAR pitched to face the horizon, in order to include as much data as possible about all objects in the scene. Consequently, our dataset can be used for the AV tasks, including SC, listed above, and for 3D scene modeling as well.

The dataset is two folds, the first is a real point cloud containing 60M points, the second is a synthetic point cloud containing 700M points collected using CARLA open source 3D simulator.

2.4.1 Acquisition Platform and Used Sensors

Figure 2.5: Prototype acquisition system used to create the PC3D dataset in the city of Paris. Sensors: Velodyne HDL32 LiDAR, Ladybug5 360° camera, Photonfocus MV1 16-band VIR and 25-band NIR hyperspectral cameras (hyperspectral data is not available in this dataset; they cannot be used in mobile mapping due to a limited exposure time).

For the dataset creation, we retrofitted a vehicle with multiple sensors to obtain a MMS that can be used for the data collection. This system is equipped with a Velodyne HDL-32E LiDAR pitched to the horizon at 45°, a 360° poly-dioptric camera ladybug5 composed of

6 cameras and a Photonfocus MV1 16-band VIR and 25 band NIR hyperspectral cameras. Hyperspectral data are not available in the dataset as another work [Duque-Arias 2021] demonstrates that they cannot be used in mobile mapping due to the limited exposure time. To visualize the different sensors see Figure 2.5

2.4.2 Real Paris

The dataset should be used for several AV tasks, scene modeling and sensor simulation. To this end, we chose to acquire the real-world data in a dense urban scene in the heart of Paris, in parts of Saint-Michel Avenue and Soufflot Street, containing a wide variety of static and dynamic object classes, which represent challenges for 3D scene understanding tasks. Moreover, the acquired geometry contains many complex "*open surface*" shapes (see figure 2.6), which challenges existing 3D modeling and surface reconstruction methods.

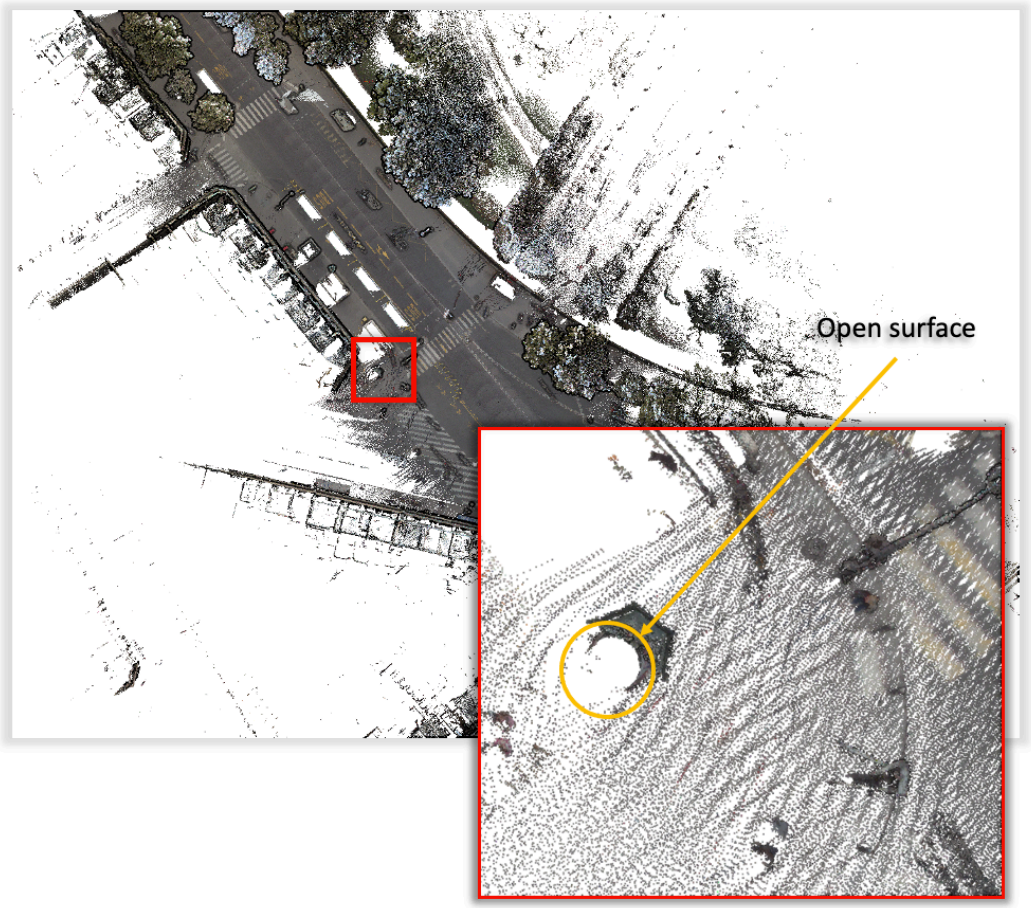


Figure 2.6: An example of open surfaces. The structures are scanned from the mapping system's side only

To create the accumulated point cloud, we aggregate the LiDAR scans and build a full map of the environment by using IMLS-SLAM [Deschaud 2018], which is a precise LiDAR-based SLAM that uses only LiDAR data for the scans aggregation. Although our platform is equipped with a high precision LANDINS iXblue IMU and an RTK GPS, an IMU and GPS based localization does not outperform a precise SLAM system, such as IMLS-SLAM in a dense environment, even with offline post processing, thanks to the geometric information that are present on the buildings. The hyperparameters used for IMLS-SLAM on Paris-CARLA-3D are $n = 30$ scans, $s = 600$ keypoints and $r = 0.50\,\text{m}$ for the neighbor search, we refer the reader to [Deschaud 2018] for explanations about the parameters. The drift of

the IMLS-SLAM odometry is less than 0.40 % with no failure (failure = no convergence of the algorithm) cases. Having said that, the quality of the used odometry makes it possible to consider the localization used for scan aggregation outputting the final map as "ground truth". The final aggregated point cloud is colored by projecting each 3D point coming from the LiDAR scans onto the 360° camera that is synchronized and calibrated with the LiDAR. For the projection, we use the camera frame timestamp that is closest to its LiDAR counterpart.

Annotating the point cloud is not an easy process as it requires time, re-iterations, checks and validations. The annotation was done manually by its entirety using CloudCompare [*CloudCompare (version 2.11.1) [Anoia]* 2021], it involved 3 persons, including me, and consisted of 3 phases to ensure consistency and high-quality. In the first phase, the dataset was split into two, with one person annotating each part. The first annotation took approximately 100 hours per person. The second phase consisted of each person verifying the annotation of the other with feedback and corrections. In the third phase, a third person who was not involved in the annotation process performed a verification of the annotation of the whole dataset, while verifying the consistency with the annotation in CARLA. The data annotation was carried out to add the semantic class attribute to each 3D point, which finally led to a total of 23 semantic classes throughout the entire dataset. Moreover, vehicles instances were separated so that the dataset can be used for the IS task as well. Semantic classes used to annotate the point cloud are the same as the semantic classes defined in CARLA simulator, this mapping can thus be used for transfer learning methods, which can leverage the large number of points available in synthetic CARLA, before fine-tuning on the real data in Paris. The total manual annotation time is about 300 hours, which resulted in a very high quality visible in Figures 2.8, 2.9, 2.10, 2.11, 2.12, 2.13.

We split the final aggregated point cloud into six parts containing 10M points each and saved in binary ply format. Every point holds several attributes that we list below:

- *x*, *y*, *z*, *x_lidar_position*, *y_lidar_position*, *z_lidar_position*, *intensity*, *timestamp*, *scan_index*, *scan_angle*, *vertical_laser_angle*, *laser_index*, *red*, *green*, *blue*, *semantic*, *instance*

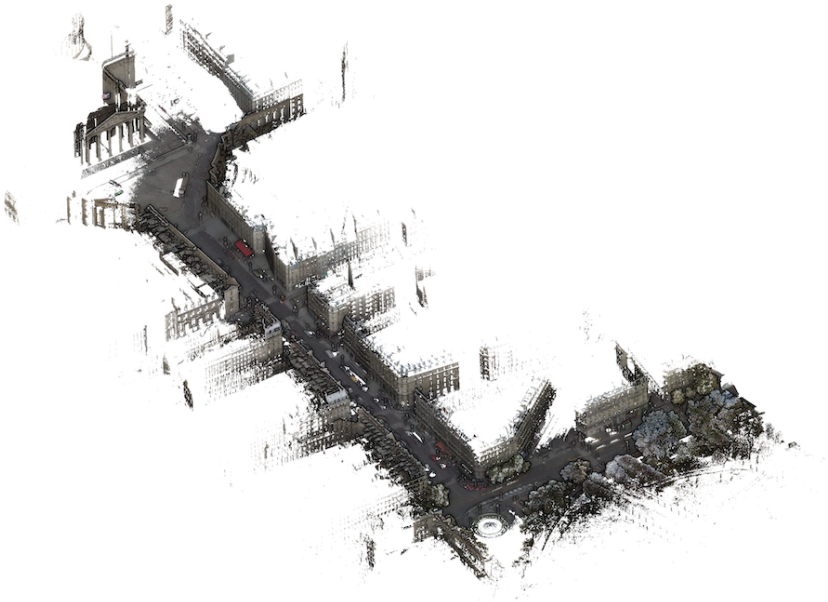


Figure 2.7: Full PC3D-Paris dataset in RGB. Showing the train, test and val parts, acquired in Soufflot street.

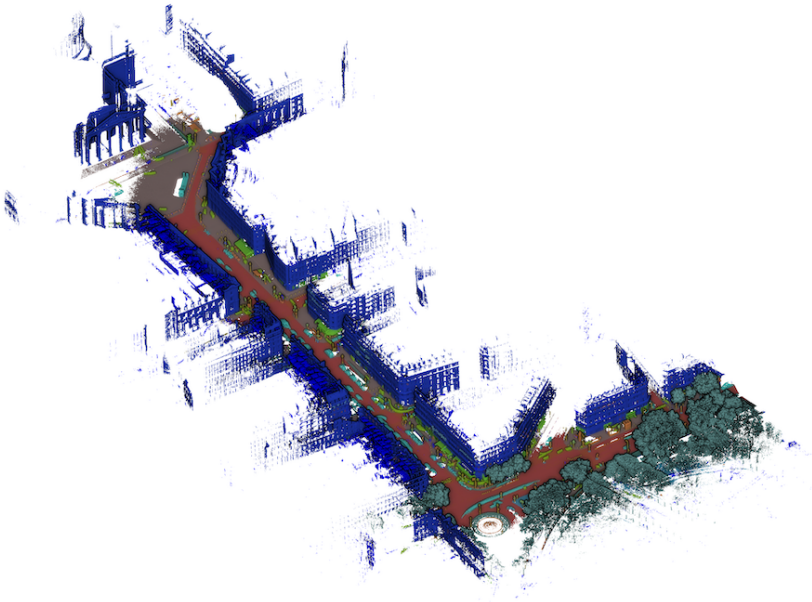


Figure 2.8: Full PC3D-Paris dataset. Showing the semantic classes in the train, test and val parts, acquired in Soufflot street.

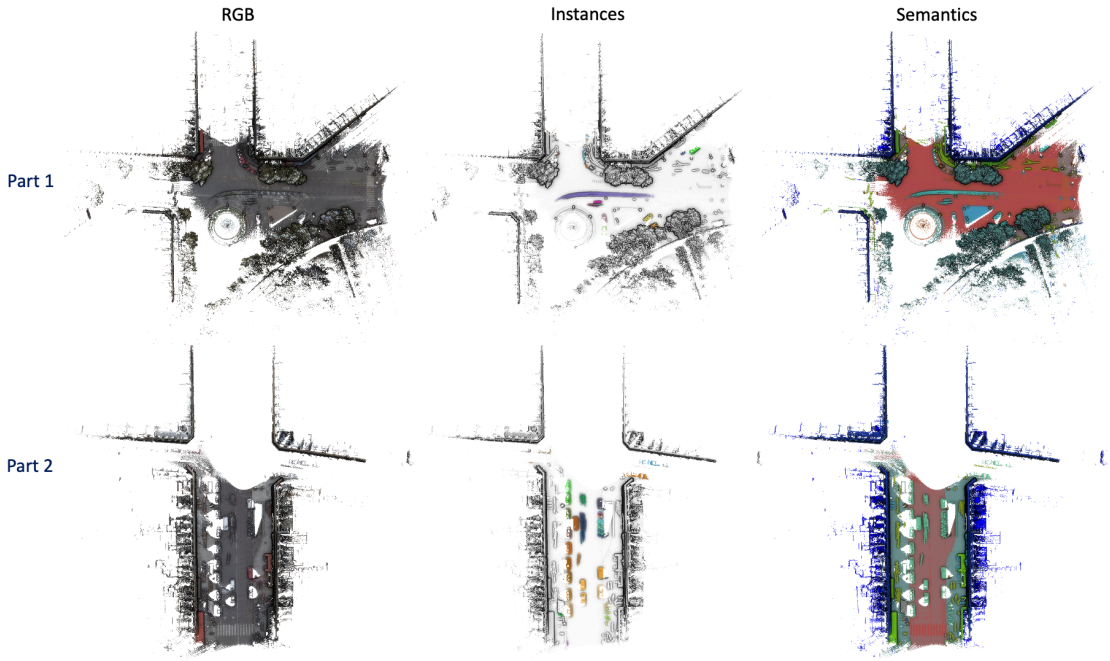


Figure 2.9: PC3D-Paris training set.

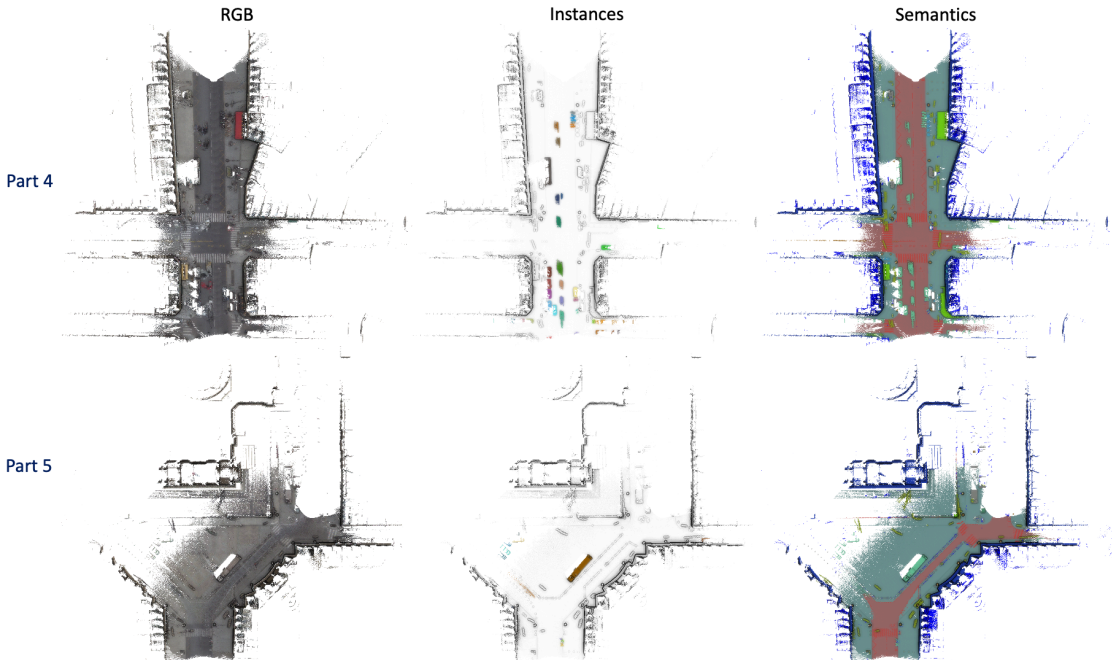


Figure 2.10: PC3D-Paris validation set.

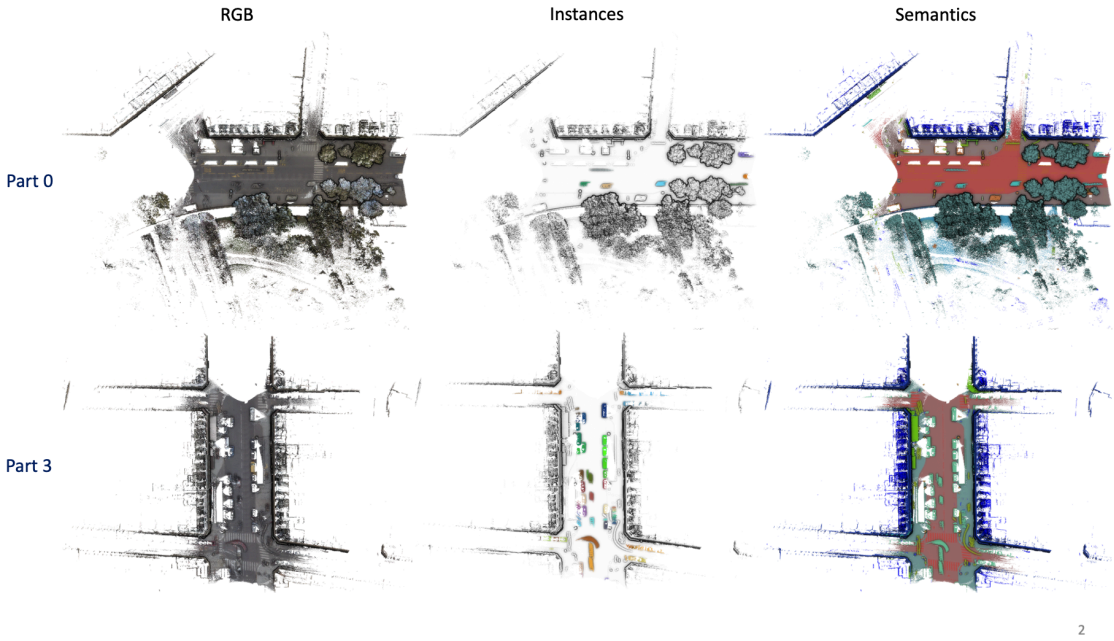


Figure 2.11: PC3D-Paris test set.

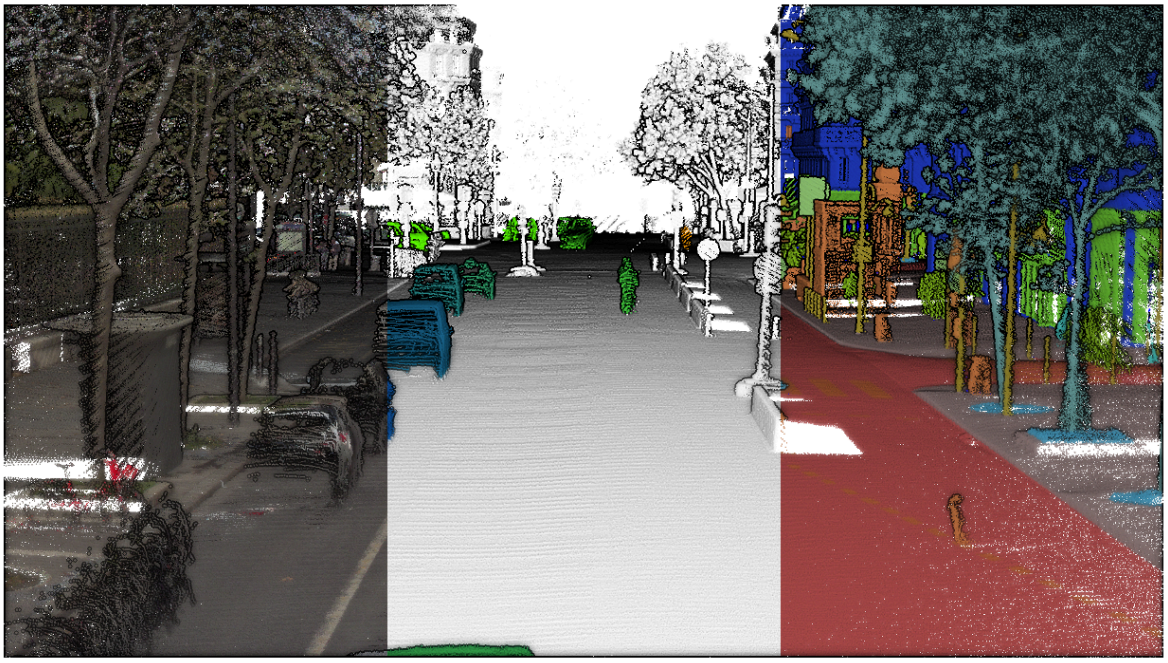


Figure 2.12: Street view in PC3D-Paris. Showing the dataset in RGB, instances, and semantic classes going from left to right, respectively

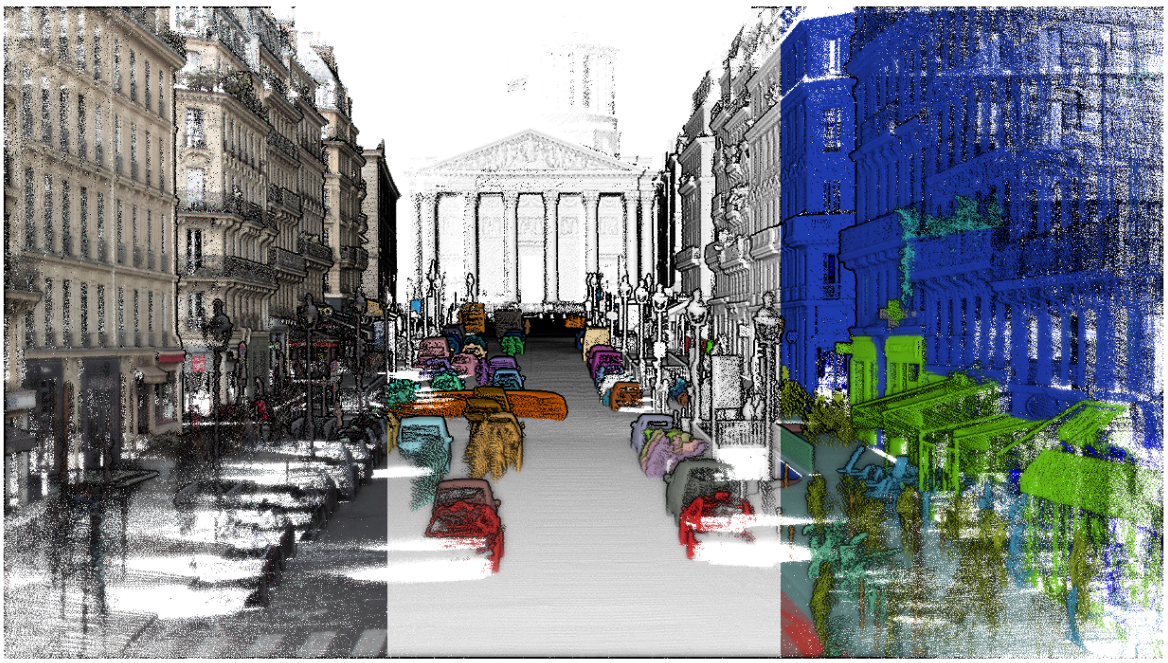


Figure 2.13: Another street view in PC3D-Paris. Showing the dataset in RGB, instances, and semantic classes going from left to right, respectively

2.4.3 Synthetic CARLA

CARLA [Dosovitskiy et al. 2017b] is an open source simulator that can be used to simulate different sensors, such as the LiDAR and camera. We used CARLA v0.9.10 that contains a set of seven virtual towns, ranging from "Town01" to "Town07", in which a vehicle can be placed along with the different sensors, a trajectory can be then generated and finally the physical sensor model can be simulated to obtain the final simulated data. Since we only need the sensors output from the simulation, we can create a virtual platform corresponding to the real one used for the collection of the Paris data, using only the sensors positions with respect to the reference point of the vehicle. Having said that, we place the virtual sensors at the same positions as the real ones and pitch the LiDAR to put it in mapping configuration, define trajectories in the different towns, simulate the camera and LiDAR sensors and finally obtain the output data in the form of point clouds and RGB images. The point clouds were aggregated using the ground truth trajectories. The final point clouds were separated by town and each one contained 100M point.

As done for the real part of the dataset, the final aggregated point cloud is colored by projecting each 3D point coming from the LiDAR scans onto the 360° camera, using the closest timestamps, that is synchronized and calibrated with the LiDAR.

We store the final aggregated point clouds in seven parts, each part corresponding to one map (town), containing 100M points each and save them in binary ply format. Every point holds several attributes that we list below:

- *x*, *y*, *z*, *x_lidar_position*, *y_lidar_position*, *z_lidar_position*, *timestamp*, *scan_index*, *cos_angle_lidar_surface*, *red*, *green*, *blue*, *semantic*, *instance*, *semantic_image*.

In CARLA, each geometric shape contains the object class, at the ray-primitive intersection time, the semantic class information can be retrieved, which eliminates the need for manual annotation, making the annotation process automatic. CARLA contains 23 semantic classes and instances of *vehicles* and *pedestrians*, which we also keep as attributes. Moreover, during the colorization process, we keep the semantic information available in the images (attribute *semantic_image*), for each 3D point.

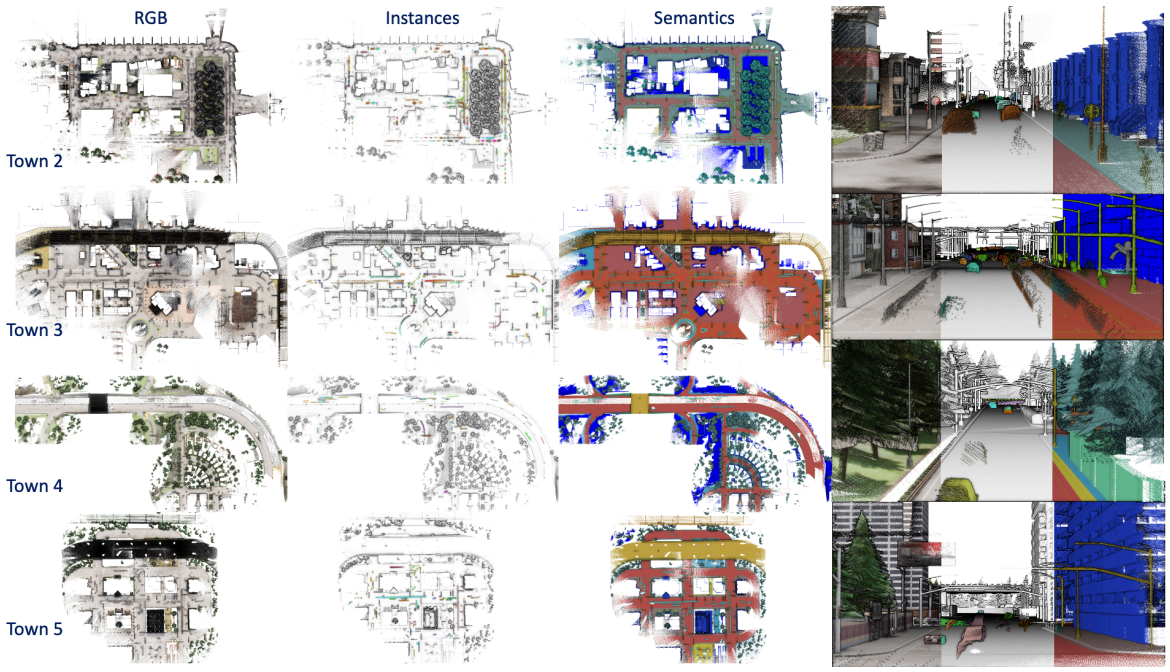


Figure 2.14: PC3D-CARLA training set.

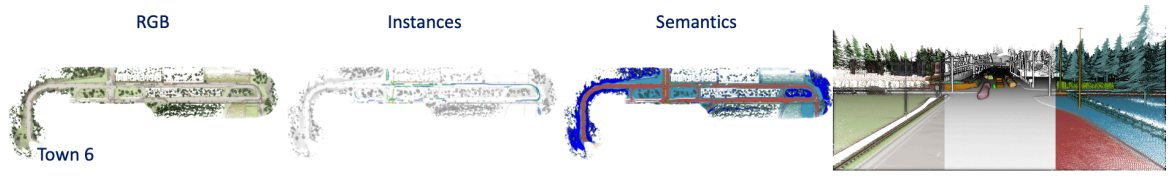


Figure 2.15: PC3D-CARLA validation set.

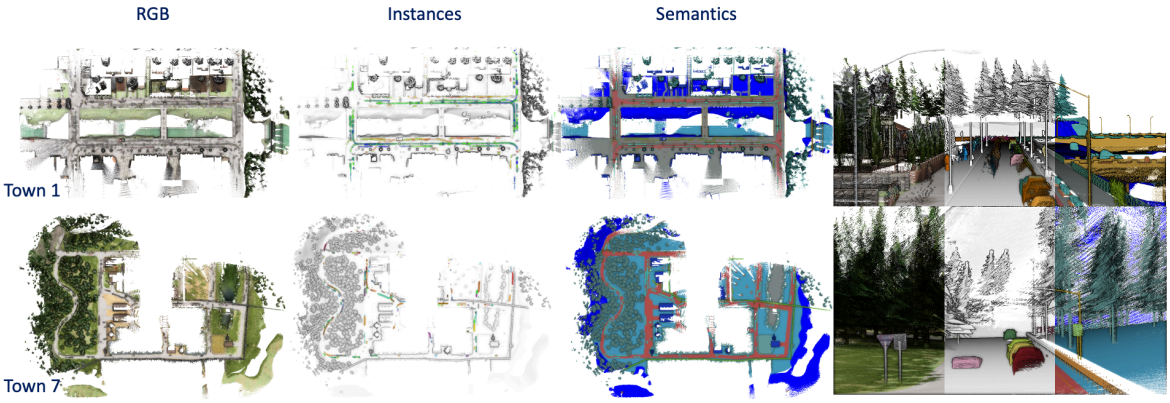


Figure 2.16: PC3D-CARLA test set.

2.5 Paris-CARLA-3D Dataset Properties

Paris-CARLA-3D has a linear distance of 550 m in Paris and about 5.8 km in CARLA (the same order of magnitude as the number of points (x10) between synthetic and real). For the real part, this represents three streets in the center of Paris. The area coverage is not big but the number and variety of urban objects, pedestrian movements and vehicles is important: it is precisely this type of dense urban environment that is challenging to analyze.

2.5.1 Classes Statistics

Paris-CARLA-3D dataset is split into 2 parts. The real *Paris* part contains six point clouds, namely, Soufflot0 (S_0) to Soufflot5 (S_5) and the synthetic *CARLA* part contains seven point clouds, namely, Town01 (T_1) to Town07 (T_7).

The CARLA dataset can be divided into groups following the Town configuration. Two main groups are available in CARLA and the dataset can be split accordingly. The urban group, consists of towns T_1, T_2, T_3 and T_5 and the rural group consists of towns T_4, T_6 and T_7. As it occurs in real world scenarios, not every class is present in every town, as we detail below:

- eleven classes are present in all towns (*road, building, sidewalk, vegetation, vehicles, road-line, fence, pole, static, dynamic, traffic-sign*)
- three classes in six towns (*unlabeled, wall, pedestrian*)
- three classes in five towns (*terrain, guard-rail, ground*)
- two classes in four towns (*bridge, other*)
- one class in three towns (*water*)
- two classes in two towns (*traffic-light, rail-track*).

The Paris dataset can be divided into two main groups containing different sets of semantic classes each. The first group consists of point clouds S_0 and S_1 close to the Luxembourg garden, which contains vegetation and wide roads. The second group consists of point clouds S_2, S_3, S_4 and S_5 containing denser urban areas with building on both sides of the road, vehicles, pedestrians, etc. The classes variability in Paris is smaller than in CARLA, however, this is an expected and a desired feature, because the point clouds correspond to the same town and were successively acquired by a MMS. Even though, as it occurs with CARLA towns, not every class is present in every point cloud as we detail below:

- twelve classes are present in all point clouds (*road, building, sidewalk, road-line, vehicles, other, unlabeled, static, pole, dynamic, pedestrian, traffic-sign*)
- three classes in five point clouds (*vegetation, fence, traffic-light*)
- one class in two point clouds (*terrain*)
- seven classes in any point cloud (*wall, sky, ground, bridge, rail-track, guard-rail, water*).

Table 2.2 shows the detailed statistics of the classes in the Paris-CARLA-3D dataset.

	Paris						**CARLA**						
Class	S_0	S_1	S_2	S_3	S_4	S_5	T_1	T_2	T_3	T_4	T_5	T_6	T_7
unlabeled	0.9	1.5	3.9	3.2	1.9	0.9	5.8	2.9	-	7.6	0.0	6.4	1.8
building	14.9	18.9	34.2	36.6	33.1	32.9	6.8	22.6	15.3	4.5	16.1	2.6	3.3
fence	2.3	0.6	0.7	0.8	-	0.4	1.0	0.6	0.0	0.5	3.8	1.5	0.6
other	2.1	3.4	6.7	2.2	2.5	0.4	-	-	-	-	0.1	0.1	0.1
pedestrian	0.2	1.0	0.6	1.0	0.7	0.7	0.1	0.2	0.1	0.0	-	0.1	0.0
pole	0.6	0.9	0.6	0.8	0.7	1.1	0.6	0.6	4.2	0.8	0.8	0.4	0.3
road-line	3.8	3.7	2.4	4.1	3.5	3.4	0.2	0.2	2.9	1.6	2.2	1.3	1.7
road	41.0	49.7	35.0	37.6	40.6	27.5	47.8	37.2	53.1	52.8	44.7	58.0	42.8
sidewalk	10.1	4.2	7.3	6.7	11.9	29.4	22.5	17.5	10.3	1.7	10.5	3.1	0.4
vegetation	18.5	9.0	0.1	0.3	0.1	-	8.7	10.8	2.7	12.8	4.6	8.1	23.1
vehicles	1.3	1.8	6.5	6.5	3.3	1.6	1.7	3.1	0.9	0.5	3.1	4.2	0.9
wall	-	-	-	-	-	-	1.9	3.6	1.4	5.4	5.3	3.4	-
traffic-sign	0.1	0.4	0.1	0.1	0.3	0.1	-	0.0	-	0.1	0.0	0.0	0.1
sky	-	-	-	-	-	-	-	-	-	-	-	-	-
ground	-	-	-	-	-	-	-	0.0	0.2	1.4	0.3	0.1	-
bridge	-	-	-	-	-	-	1.7	-	-	0.7	6.6	-	-
rail-track	-	-	-	-	-	-	-	-	7.6	-	0.5	-	-
guard-rail	-	-	-	-	-	-	0.0	-	-	4.3	-	1.2	0.5
static	2.6	2.3	0.3	0.1	0.7	1.5	0.1	0.1	-	-	-	-	-
traffic-light	0.1	0.2	0.1	0.1	0.1	-	0.8	0.5	0.3	0.3	0.3	-	-
dynamic	0.3	1.6	1.5	0.2	0.7	0.0	0.1	0.1	0.1	0.3	0.1	0.1	0.1
water	-	-	-	-	-	-	0.4	-	0.0	-	-	-	0.6
terrain	1.4	0.8	-	-	-	-	-	-	0.9	4.8	1.1	9.6	23.8
# Points	**60M**						**700M**						

Table 2.2: Class distribution in Paris-CARLA-3D (PC3D) dataset (in %). Columns headed by S_i stand for Soufflot from PC3D-Paris data and T_j stand for Town from PC3D-CARLA data.

2.5.2 Split for training

In the original work, we have set baselines for different AV tasks, such as SS, IS and SC. The SC task was realized by me and the SS and IS tasks were realized by another co-author [Duque-Arias 2021; Duque-Arias et al. 2021; Ku et al. 2020]. For the different tasks presented, we have chosen to split the dataset into the following Train/Val/Test, according to the distribution of the classes:

- **Training data:** S_1, S_2 (Paris); T_2, T_3, T_4, T_5 (CARLA)

- **Validation data:** S_4, S_5 (Paris); T_6 (CARLA)
- **Test data:** S_0, S_3 (Paris); T_1, T_7 (CARLA)

2.5.3 Interest in having both synthetic and real data

Using simulated data to train AV tasks have proven to introduce a large domain gap when testing on real-world data. Paris-CARLA-3D was created with the reduction of this domain gap in mind. Having a dataset containing a real part and a synthetic part that was acquired with a virtual platform that is the closest possible to the real one already contributes to the domain gap reduction. Moreover, it allows the reproduction of certain issues, such as the difference in point of view of the LiDAR and camera sensors, creating color artefacts on the colored point clouds. Synthetic data is relatively easy to produce and a virtually infinite quantity can be automatically created containing pixel-wise, or point-wise semantic and instance information almost cost free for 2D and 3D vision tasks. Having said that, it is appealing to develop methods leveraging the large quantity of synthetic data, however, it is not guaranteed that they generalize when tested on real data, due to the domain gap. Using the same annotation that we have in Paris-CARLA-3D, a task can be learned on the synthetic data and tested on the real data, but as previously said, this would help in reducing the domain gap, however, it does not close it as we will see in section 2.6.2. A more promising approach is to leverage the large amount of data available in the synthetic datasets and develop domain adaptation strategies that quickly adapt the learned tasks to the real world, using small amounts of data.

2.5.4 Transfer learning

Simulators are becoming more realistic with the increased availability of high performance computing hardware, which makes it possible to increase the complexity of virtual worlds making them closer to reality. We argue that if we could bring the synthetic and real data closer to each other, we can obtain more robust results using transfer learning methods. Another benefit of doing so is that any scenario can be generated in simulated environments, which eliminates the limitation of the domain of scenarios collected from the real world. Synthetic-to-real transfer learning methods is a line of research that could bring many advantages in the future.

We will now describe the Scene Completion (SC) task using Paris-CARLA-3D dataset.

2.6 Scene Completion Task

The Scene Completion (SC) task consists of predicting the missing parts of a scene (see Figure 2.17), which can be in the form of a depth image, a point cloud, or a mesh. This is an important problem in 3D Mapping which is caused by the presence of holes resulting from occlusions and removal of unwanted objects, such as vehicles or pedestrians. It can be solved in the form of 3D reconstruction [Gomes et al. 2014], scan completion [Xu et al. 2019], or more specifically, methods to fill holes in a 3D model [Guo et al. 2018a].

Semantic Scene Completion (SSC) is the task of filling the geometry as well as predicting the semantics of the points, with the aim that the two tasks carried out simultaneously benefit each other (survey of SSC in [Roldao et al. 2021]). It is also possible to jointly predict the geometry and color during SC, as in SPSG [Dai et al. 2021]. For now, we will only evaluate the geometry prediction and leave the prediction of simultaneous geometry, semantics and color for future work.

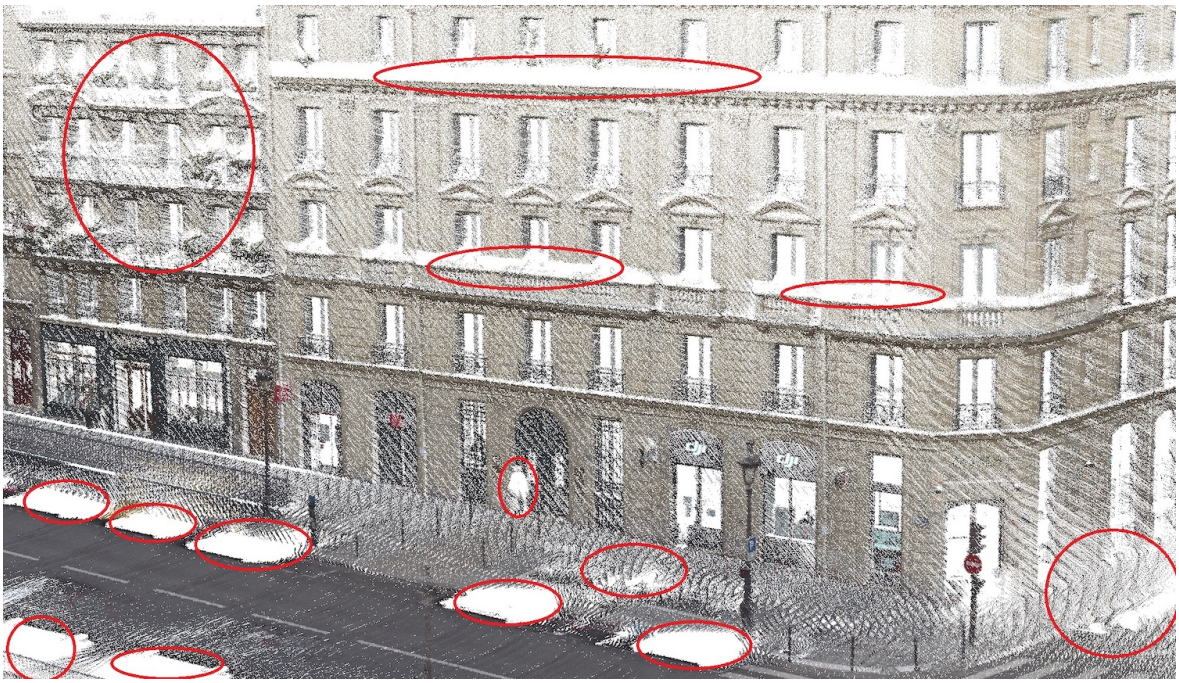


Figure 2.17: Paris data after removal of vehicles and pedestrians. Zones in red circles shows the interest in performing scene completion for 3D mapping: attempt to fill holes from removed pedestrians, parked cars, and to improve sampling of points in areas far from the LiDAR.

The vast majority of existing SC methods are performed on small indoor scenes, while in our case we have a dense outdoor environment with Paris-CARLA-3D. Completing outdoor LiDAR point clouds is more challenging than data obtained from RGB-D images acquired in indoor environments, due to the sparsity of points obtained using LiDAR sensors. Moreover, larger occluded areas are present in outdoor scenes caused by static and temporary foreground objects, such as parked vehicles, moving vehicles, etc... The SemanticKITTI [Hackel et al. 2017] dataset is used to perform SC and SSC on LiDAR data, however, they use only a single scan as input to the SC network, with the target, ground truth, being the accumulation of all LiDAR scans. In our dataset, we seek to complete the "holes" from the accumulation of all LiDAR scans.

2.6.1 Task Protocol

We introduce the task protocol to perform SC on PC3D. First, we extract random small chunks from the original point cloud. Each chunk is then transformed into a discretized regular 3D grid representation containing the Truncated Signed Distance Function TSDF values, which express the distance from each voxel to the surface represented by the point cloud. For the TSDF representation, we use the classical signed point-to-plane distance to the closest point in the point cloud as in [Hoppe et al. 1992] and we call this function hoppe. Our original point cloud is already incomplete due to the occlusions caused by static objects and the sparsity of the scans. To overcome this incompleteness, we make the point cloud more incomplete by removing 90% of the points (by *scan_index*), and use the incomplete data to compute the TSDF input to the neural network. Moreover, we use the original point cloud containing all of the points as the ground truth and compute the target TSDF. Our approach is inspired by the work done by SG-NN [Dai et al. 2020] and we do this in order to have more complete data that is used as target for the network (see Figure 2.18). Removing points according to their *scan_index* allows us to create larger "holes" than removing points

at random. For the chunks, we used a grid size of 128x128x128 and a voxel size of 5 cm (compared to a voxel size of 2 cm used for indoor scenes in SG-NN [Dai et al. 2020]). Dynamic objects, pedestrians, vehicles and unlabeled points are first removed from the data using the ground-truth semantic information.

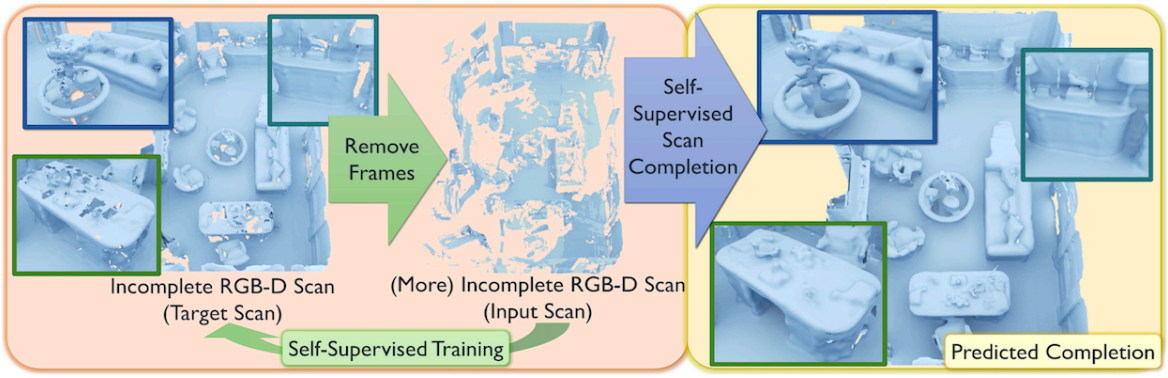


Figure 2.18: Given an incomplete scan of a scene, the scene completion task predicts a more complete geometry. Source [Dai et al. 2020]

To evaluate the completed scene, we use the Chamfer Distance (CD) between the original P_1 and predicted P_2 point clouds:

$$\mathcal{CD}(\mathcal{P}_1, \mathcal{P}_2) = \frac{1}{|\mathcal{P}_1|} \sum_{x \in \mathcal{P}_1} \min_{y \in \mathcal{P}_2} ||x - y||_2 + \frac{1}{|\mathcal{P}_2|} \sum_{y \in \mathcal{P}_2} \min_{x \in \mathcal{P}_1} ||y - x||_2 \tag{2.1}$$

In a self-supervised context, not having the ground truth and having the predicted point cloud more complete than the target places some limitations on using the CD metric. To overcome this limitation, we introduce a mask that is used to compute the CD only on the points that were originally available. The mask is simply a binary occupancy grid on the original point cloud.

As previously explained, we extract random chunks for Paris (1 000 chunks per point cloud) and CARLA (3 000 chunks per town) and provide them along with the dataset for future research on SC.

2.6.2 Experiments: setting a baseline

In this section, we present a baseline for SC using the SG-NN network [Dai et al. 2020] to predict the missing points. This chapter only introduces the SC task, however, we provide extensive experiments and detailed analysis on the SC task in chapter 3. SG-NN predicts only the geometry and not the semantics nor the color. In SG-NN, the authors use volumetric fusion [Curless et al. 1996] to compute a TSDF volume representation from RGB-D images, which can't be used on LiDAR point clouds. To obtain a similar representation, we compute a different TSDF directly from point clouds.

Using the cropped chunks, we estimate the normal at each point using PCA as in [Hoppe et al. 1992] with $n = 30$ neighbors and obtain a consistent orientation using the LiDAR sensor position provided with the points. Using the normal information, we use the SDF introduced in [Hoppe et al. 1992], due to its simplicity and the ease of vectorizing, which reduces the data generation complexity. After obtaining the SDF volumetric representation, we convert the values to voxel units and truncate the function at three voxels, which results in a 3D sparse TSDF volumetric representation that is similar to the input of SG-NN [Dai et al. 2020]. For the target, we use all the points available in the original point cloud, and for the input, we keep 10% of points, using the scan indices, in each chunk, in order to obtain the "incomplete" point cloud representation.

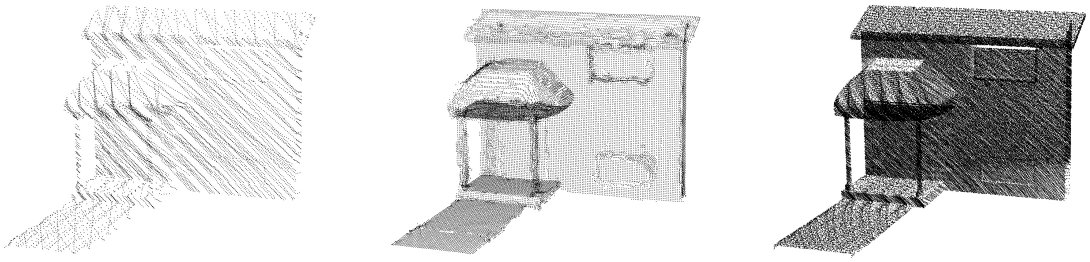
The resulting sparse tensors are then used for training and the network is trained for 20 epochs with ADAM optimizer and a learning rate of 0.001. The loss is a combination of Binary Cross Entropy (BCE) on occupancy and L1 Loss on TSDF prediction. The training is done on a GPU NVIDIA RTX 2070 SUPER with 8Go RAM.

In order to increase the number of samples and prevent overfitting, we perform data augmentation on the extracted chunks: random rotation around z, random scaling between 0.8 and 1.2, and local noise addition with $\sigma = 0.05$.

For a proper quantitative evaluation of the network, we extract a point cloud from the TSDF predicted by the network following an approach that is similar to the marching cubes algorithm, where we interpolate 1 point per voxel. Finally, we compute the CD (see equation 2.1) between the point cloud extracted from the predicted TSDF and the original point cloud without dynamic objects. Moreover, we use the introduced mask to limit the CD computation to known regions, which are represented by voxels where we have points in the original point cloud.

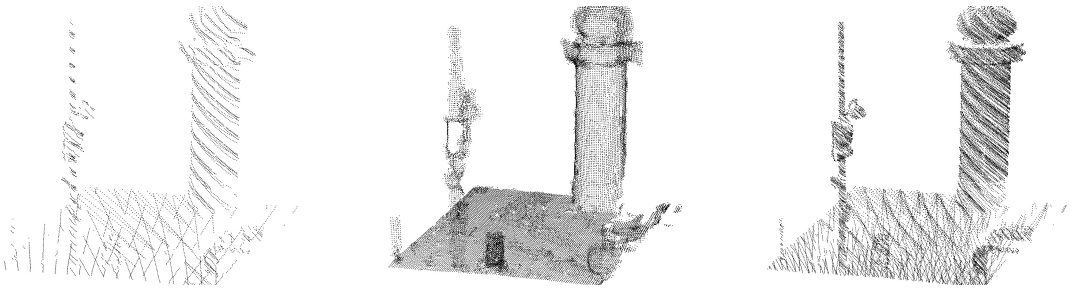
2.6.2.1 Qualitative results

Figure 2.19 shows the SC result on one point cloud chunk from CARLA T_1 test set. Figure 2.20 shows the SC result on one chunk from Paris S_0 test set. We can see that the network manages to produce point clouds quite close visually to the original, despite having a sparse point cloud as input, with only 10 % of points from the original point cloud.



(a) Sparse input point cloud (b) Predicted point cloud (c) Original point cloud

Figure 2.19: Scene completion task for one chunk point cloud in Town1 (T_1) of CARLA test data (training on CARLA data).



(a) Sparse input point cloud (b) Predicted point cloud (c) Original point cloud

Figure 2.20: Scene completion task for one chunk point cloud in Soufflot0 (S_0) of Paris test data (training on Paris data).

2.6.2.2 Quantitative Results

Table 2.3 shows the results of our experiment on Paris-CARLA-3D data. We can see that the network makes it possible to create point clouds whose distances to the original clouds is

clearly smaller.

For further metric evaluation, we provide the mean IoU and ℓ_1 distance between the target and predicted TSDF values on the 2 000 and 6 000 chunks for Paris and CARLA, respectively. The results are also reported in Table 2.3.

Test set	$CD_{in\leftrightarrow ori}$	$CD_{pred\leftrightarrow ori}$	$\ell_{1pred\leftrightarrow tar}$	$mIoU_{pred\leftrightarrow tar}$
S_0 & S_3 (Paris)	16.6 cm	10.7 cm	0.40	85.3 %
T_1 & T_7 (CARLA)	13.3 cm	10.2 cm	0.49	80.3 %

Table 2.3: Scene Completion results on Paris-CARLA-3D. CD is the mean Chamfer Distance over 2 000 chunks for the Paris test set (S_0 and S_3) and 6 000 chunks for the CARLA test set (T_1 and T_7). ℓ_1 is the mean ℓ_1 distance between predicted and target TSDF measured in voxel units for 5 cm voxels and $mIoU$ is the mean intersection over union of TSDF occupancy. Both metrics are computed on known voxel areas. *ori* means original point cloud, *in* is the input point cloud containing 10% of the original, *pred* is the predicted point cloud computed from the predicted TSDF.

2.6.2.3 Transfer learning with scene completion

We argued that using synthetic data for pre-training and then fine-tuning on the real data would improve the inference results. The objective is to do the inference on the test set of the real part "Paris" (S_0 and S_3) of PC3D, to see the effect of transfer learning, we perform three training scenarios, listed below:

- Training only on real data with Paris training set
- Training only on synthetic data with CARLA training set
- Pre-train on synthetic data then fine-tune on real data.

The results are in Table 2.4. We can see that the Chamfer Distance (CD) is better for a model trained only on synthetic CARLA data, compared to the model trained only on Paris data: the network is attempting to fill a local plane in large missing regions and smoothing the rest of the geometry. This is an expected behavior, because of the handcrafted geometry present in CARLA, where planar geometric features are predominantly present. As we can see in table 2.4, pre-training on PC3D-CARLA and fine-tuning on PC3D-Paris, allows us to obtain the best predicted TSDF, seen from the ℓ_1 and $mIoU$ metrics, and the best predicted point cloud, seen from the CD.

Test set: S_0 & S_3 Paris data	$CD_{in\leftrightarrow ori}$	$CD_{pred\leftrightarrow ori}$	$\ell_{1pred\leftrightarrow tar}$	$mIoU_{pred\leftrightarrow tar}$
Trained only on Paris	16.6 cm	10.7 cm	0.40	85.3 %
Trained only on CARLA	16.6 cm	8.0 cm	0.48	84.0 %
Pre-trained CARLA, fine-tuned on Paris	16.6 cm	7.5 cm	0.35	88.7 %

Table 2.4: Results of transfer learning for the scene completion task. CD is the mean Chamfer Distance between point clouds. ℓ_1 is the mean ℓ_1 distance between predicted and target TSDF measured in voxel units for 5 cm voxels and $mIoU$ is the mean intersection over union of TSDF occupancy. The mean is over 2 000 chunks for Paris data. *ori* means original point cloud, *in* is input point cloud (10 % of the original), *pred* for CD is the predicted point cloud (computed from predicted TSDF), *pred* for IoU and ℓ_1 is the predicted TSDF and *tar* is the target TSDF.

2.7 Tasks per dataset

In the following chapters, we will present three main tasks that constitute our reality replication pipeline for sensor simulation and open the possibility of obtaining a massive and high quality simulation. More specifically, we perform the scene completion (SC) task to obtain a more complete static background, after which we model the scene with semantic splats and finally perform the LiDAR simulation. For these tasks, we use the introduced datasets as detailed in Table 2.5

Task	**PC3D**	**M-City**	**SemanticKITTI**	**Matterport3D**
Scene Completion	✓			✓
Scene Modeling	✓	✓	✓	
LiDAR Simulation	✓	✓	✓	

Table 2.5: The datasets used for each task in the simulation pipeline. PC3D-Paris and PC3D-CARLA were used for the scene completion task, while only PC3D-Paris was used for the scene modeling and LiDAR simulation tasks.

2.8 Conclusion

We presented a new dataset called Paris-CARLA-3D. This dataset is made up of both real data (60M points) and synthetic data (700 M points) collected using the same LiDAR and camera sensors mounted on a mobile maping system and its virtual replica, respectively. Using the dataset, we presented the scene completion task, its evaluation protocol and provided a baseline using SG-NN which sets the starting point for the scene completion task on PC3D. SG-NN is used for indoor datasets collected using RGB-D sensors. However, we adapted it to be used for outdoor scenes collected using LiDAR sensors. We could see that even with a simple surface representation, the network was able to learn complex geometries and noticed that using transfer learning, leveraging the large synthetic dataset for pre-training and fine-tuning on the smaller real dataset, we could obtain better learning results.

We also briefly presented the M-City dataset, which is an in-house dataset that will be used for the scene modeling and sensor simulation tasks. Moreover, we will use PC3D-Paris for the task of SC, scene modeling and sensor simulation. SemanticKITTI and M-City along with PC3D will be used for the scene modeling and rendering (see chapter 4), and LiDAR simulation (see chapter 5) tasks, since they were acquired using different sensors, therefore, their use is to validate that our pipeline is not limited to a specific kind of sensors, or sensor configuration.

The work on Paris-CARLA-3D has resulted in a publication in the MDPI Remote Sensing journal:

- Deschaud, J.E., Duque, D., Richa, J.P., Velasco-Forero, S., Marcotegui, B. and Goulette, F., 2021. Paris-CARLA-3D: A real and synthetic outdoor point cloud dataset for challenging tasks in 3D mapping. Remote Sensing, 13(22), p.4713.

Chapter 3

Scene Completion

Contents

3.1 Introduction **36**
3.2 Related Work **40**
3.2.1 Distance Functions 40
3.2.1.1 Signed Distance Function 40
3.2.1.2 Unsigned Distance Function 41
3.2.2 Implicit Function Learning 41
3.2.3 Point Cloud Completion 41
3.2.4 Scene Completion 43
3.2.4.1 Scene Completion from RGB-D Sensors 43
3.2.4.2 Scene Completion from LiDAR Sensors 45
3.3 Scene Representation **46**
3.3.1 Unsigned Weighted Euclidean Distance (UWED) 46
3.3.2 Point Cloud Extraction 47
3.3.2.1 Extraction from SDF 47
3.3.2.2 Extraction from UDF 47
3.4 Experiments and Results **48**
3.4.1 Distance Functions for Scene Completion 48
3.4.2 Dataset Preparation 49
3.4.3 Evaluation Metrics for Scene Completion 50
3.4.4 Implementation Details 50
3.4.5 Validity of our UDF as Scene Representation 50
3.4.6 Results of Distance Functions for Scene Completion 53
3.4.7 Scene Completion on Target 58
3.4.8 Limitations and Perspectives 60
3.5 Conclusion **60**

Abstract

The goal of sensor simulation is to accelerate the development and testing of AVs. Our goal is to accurately model the static environment from real-world data in the form of point clouds. To this end, we make use of the point-wise semantic information to remove the dynamic objects from the scene such as cars and pedestrians. However, this results in large missing areas in the scene, due to occlusions from such objects. Missing geometry places some limitations on accurate surface modeling, to overcome these limitations, we propose to perform scene completion on the incomplete point cloud. Existing methods on point cloud completion or scene completion using an implicit, continuous representation of the scene in the form of signed distance functions fail to accurately represent the geometry, either because of their limitation to small shapes, or their inability to accurately represent open surfaces, being limited to closed objects. In this chapter, we introduce a new implicit unsigned distance function computed on a regular 3D grid, which we name Unsigned Weighted Euclidean Distance (UWED) representation that overcomes the limitation to closed surfaces. Moreover, we propose a method to extract a point cloud from an unsigned distance function (UDF) computed on a regular grid. Using the proposed method, we obtain high quality completion, while preserving the geometry of open surfaces.

Résumé

L'objectif de la simulation de capteurs est d'accélérer le développement et les tests des véhicules autonomes. Notre objectif est de modéliser avec précision l'environnement statique à partir de données réelles sous forme de nuages de points. À cette fin, nous utilisons les informations sémantiques pour supprimer les objets dynamiques de la scène tels que les voitures et les piétons. Cependant, cela se traduit par de grandes zones manquantes dans la scène, en raison des occlusions de ces objets. La géométrie manquante impose certaines limites à la modélisation précise de la surface. Pour surmonter ces limites, nous proposons d'effectuer la complétion de la scène sur le nuage de points incomplet. Les méthodes existantes de complétion de nuage de points ou de complétion de scène utilisant une représentation implicite et continue de la scène sous la forme de fonctions de distance signées ne parviennent pas à représenter avec précision la géométrie, soit en raison de leur limitation aux petites formes, soit de leur incapacité à représenter avec précision les surfaces ouvertes, se limiter aux objets fermés. Dans ce chapitre, nous introduisons une nouvelle fonction de distance non signée implicite calculée sur une grille 3D régulière, que nous nommons "Unsigned Weighted Euclidean Distance" (UWED) ou "distance euclidienne pondérée non signée" qui surmonte la limitation aux surfaces fermées. De plus, nous proposons une méthode pour extraire un nuage de points d'une fonction de distance non signée (UDF) calculée sur une grille régulière 3D. En utilisant la méthode proposée, nous obtenons une finition de haute qualité, tout en préservant la géométrie des surfaces ouvertes.

3.1 Introduction

In this chapter, we tackle the second step in our simulation pipeline, scene completion (see Figure 3.1), which aims to complete missing regions resulting from dynamic objects removal and data sparsity.

The aim of our work is to perform realistic sensor simulation to accelerate AVs testing and validation, as detailed in chapter 1. However, 3D point cloud datasets collected in an outdoor environment, such as PC3D [Deschaud et al. 2021] (the dataset introduced in chapter 2) and SemanticKITTI [Behley et al. 2019], contain many dynamic object and/or

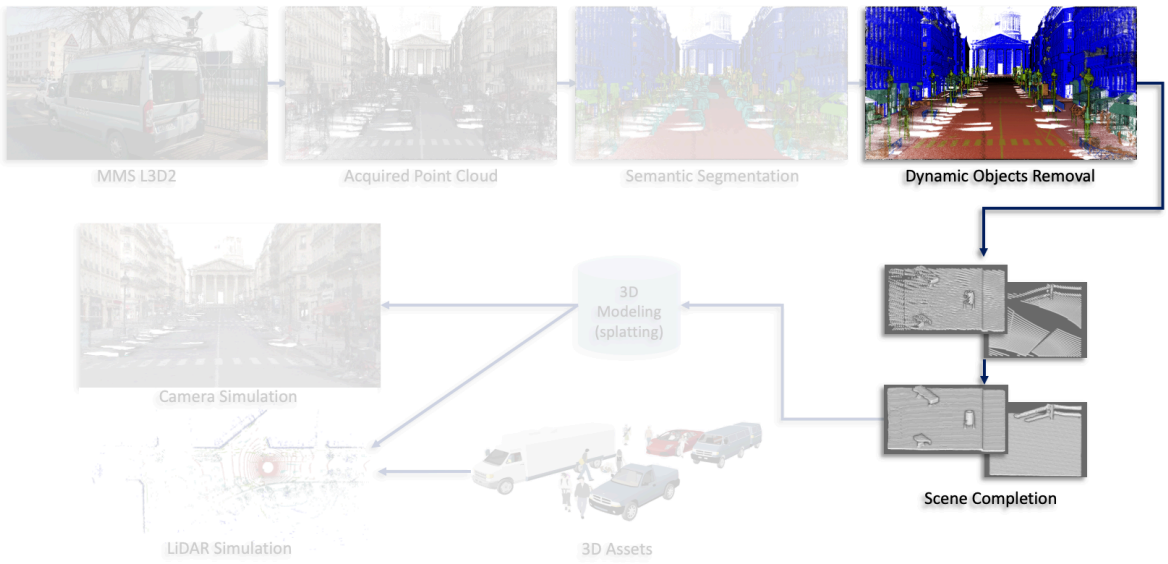


Figure 3.1: The second step in our simulation pipeline consists of removing dynamic objects from the scene and completing missing geometry resulting from object occlusion.

temporarily available static object instances, such as parked vehicles, pedestrians, etc... (see Figures 2.12, 2.13).

Testing AVs requires generating new scenarios, or augmenting the scenarios domain, to test their behavior under a variety of circumstances. Augmenting the scenarios domain requires leaving only the static background by removing non-static objects and inserting new ones while defining their positions or trajectories depending on the scenario that needs to be generated, following the identified edge case in which an AV should be tested.

The separation of the different objects in a given point cloud can be achieved with the SS task [Thomas et al. 2019; Landrieu et al. 2018; Boulch et al. 2020; Choy et al. 2019], which is the task of assigning a point-wise semantic class. We have seen in the previous chapter in sections 2.5.1 and 2.3.2 that PC3D [Deschaud et al. 2021] and SemanticKITTI [Behley et al. 2019] contain point-wise semantic information, resulting from the manual data annotation. Using the semantic information, neural networks can be trained to identify the different classes and generalize to other datasets through the SS task (see SS task protocol in PC3D [Deschaud et al. 2021]), which reduces the cost, in time and money, of data annotation.

Removing the dynamic, or temporarily static, objects from the original point cloud using point-wise semantic information, results in large missing areas in the scene. Moreover, point clouds collected by moving LiDAR sensors are subject to high local anisotropy caused by the physical model of the LiDAR (points are denser along the sweep lines), and data sparsity that is proportional to the distance between the sensor and the surface being scanned, see Figure 2.17. To solve these problems, we make use of the SC task, which is the task of completing the missing geometry from an incomplete, or a partial scan of a scene. Having completed the missing geometry, the new more complete scene can be used to generate our surface representation and simulate the different sensors, such as camera (see chapter 4) and LiDAR (see chapter 5).

The majority of previous SC methods compute an implicit representation from range data (point clouds, or range images) using a TSDF on a 3D grid as input to neural networks. A TSDF representation of the geometry encapsulates the scene in a 3D grid, where each voxel contains the signed distance to the closest sampled surface in the point cloud. Usually, a TSDF representation of the scene is computed from range images [Dai et al. 2020; Dai et al. 2018], or point clouds projected into range images [Cheng et al. 2020]. In the case of computing the TSDF directly from point clouds, the quality of the representation highly depends on

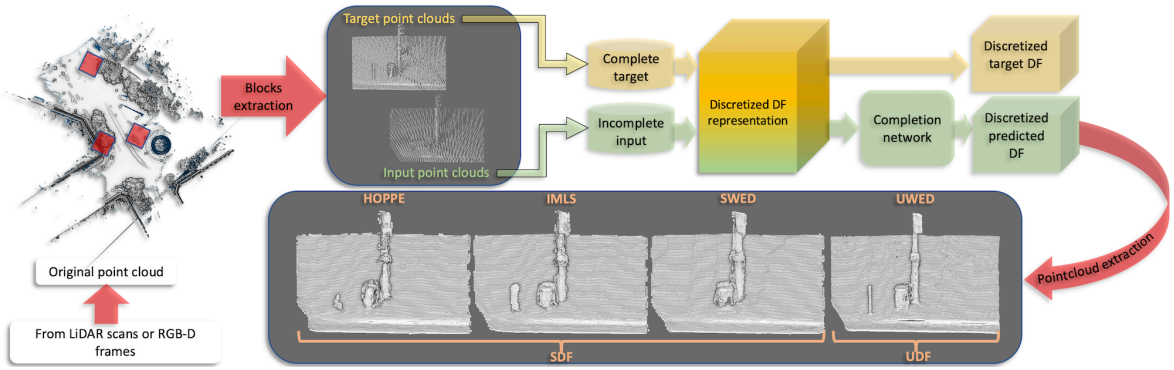


Figure 3.2: Self-supervised 3D Scene Completion (SC) comparing several Signed Distance Functions (SDFs) and Unsigned Distance Functions (UDFs). Small point cloud blocks are extracted from the original point cloud; random frames (for RGB-D) or scans (for LiDAR) are removed for the input, while all of the points are used for the target. All point clouds are then transformed into Distance Function (DF) representations on discretized grids; then, the network is trained to learn the completion of the representations. Our Unsigned Weighted Euclidean Distance (UWED) function eliminates the sign ambiguities resulting from SDFs while achieving higher accuracy on the SC task.

the quality of normals, which depends on the density of the point cloud, registration accuracy, noise level, among other factors.

Completing an implicit representation of the surface enables us to complete larger scenes. However, an implicit representation limits the number of uses of the completed scene. We present a method for extracting an explicit representation of the scene in the form of point cloud from discretized Unsigned Distance Function (UDF) values encoded in a regular 3D grid. We compare different Signed Distance Functions (SDFs) and UDFs for the SC task on indoor and outdoor point clouds collected from RGB-D and LiDAR sensors and show improved completion using the proposed UWED function.

Distance Functions (DFs) computed on 3D regular grids provide rich information about the geometry of a scene. Many DFs have been proposed to tackle a wide variety of tasks, such as surface reconstruction [Hoppe et al. 1992; Öztireli et al. 2009], Simultaneous Localisation And Mapping (SLAM) [Newcombe et al. 2011; Deschaud 2018], path planning [Oleynikova et al. 2017] and SC [Song et al. 2017; Dai et al. 2020]. Traditional surface reconstruction methods [Hoppe et al. 1992; Curless et al. 1996; Kolluri 2008] compute an SDF representation from range images or point clouds on a regular volumetric grid. A zero level set extraction algorithm such as Marching Cubes [Lorensen et al. 1987] can be deployed to extract the final mesh with geometric details proportional to the grid resolution, and a trade-off between a fine resolution and computational complexity is inevitable. These methods fill small holes by local interpolation, but they fail to fill large missing areas resulting from occluded regions caused by the foreground, dynamic objects, or from the high sparsity of points.

With advances in learning-based methods, recent studies have looked to develop approaches tackling the problem of larger missing areas, which gave rise to the SC task. Different data representations have been proposed that focus primarily on 3D binary occupancy grids, point clouds, or implicit functions representing the underlying surface. Occupancy voxel grids do not allow the geometry to be finely represented [Dai et al. 2017b; Dai et al. 2018], since the resolution of the extracted point cloud is limited to the voxel size. Performing point cloud completion like PCN [Yuan et al. 2018] or more recently VRC-Net [Pan et al. 2021] are limited to single objects and are not able to deal with large scenes. On the contrary, implicit surface representation using TSDF has proven to be able to accurately represent and predict the geometry of large scenes [Song et al. 2017; Guo et al. 2018b; Chen et al. 2019b;

Figure 3.3: A 2D example of an open surface (non-closed surface). Signed distance functions (SDF) do not handle open surfaces, and even if the truncation limits the side effects, the extraction to point clouds will generate border effects. Using an unsigned distance function (UDF), eliminates the problem of negative and positive sides. However, the difficulty lies in the extraction of a point cloud. Here, the point clouds extraction from SDF and UDF follows the methods described in sections 3.3.2.1 and 3.3.2.2 respectively. We also show flipped versions of TSDF and TUDF that remove strong gradients at truncation distance.

Zhang et al. 2019; Chen et al. 2019a; Dai et al. 2018; Dai et al. 2020; Dai et al. 2021].

Furthermore, there exist surface reconstruction methods where the geometry is encoded into neural networks, such as Occupancy Network [Mescheder et al. 2019a], DeepSDF [Park et al. 2019] or unsigned geometric data in SAL [Atzmon et al. 2020] and DUDE [Venkatesh et al. 2020]. However, these fully-connected network architectures generalize poorly to unseen data and cannot be applied to large 3D scenes.

Outdoor point clouds collected from urban scenes contain a lot of fine structures. Moreover, using a MMS for point clouds acquisition results in a partial scanning of the surrounding objects, since the MMS does not rotates around every object in the scene. This results in a point cloud filled with open surface geometries. SDFs are used for closed surfaces, because of their need to perform inside/outside classification on the voxels encapsulating the point cloud. Even if the surface is open, SDFs representation of the scene can still be computed using the normals estimation to infer the sign, which is a challenging task when done on noisy point clouds [Schertler et al. 2017; Metzer et al. 2021]. However, SDFs dilate the surface geometry on the border of the open surface. The Truncation of the SDF values results in a Truncated (T)SDF that limits the interference between spatially nearby surfaces and the dilation of the geometry on the boundaries of surfaces. Moreover, the truncation results in a sparse representation of the scene, which reduces the size of the encoded scene and accelerates the training of 3D CNNs using sparse convolutions [Graham et al. 2017; Choy et al. 2019] as well. However, it does not remove the ambiguous cases introduced by the sign for non-closed (open) surfaces and results in many artifacts (see 3D example in Figure 3.2 and 2D example in Figure 3.3).

To address these issues, we present a UDF called Unsigned Weighted Euclidean Distance (UWED), which can be computed on a 3D volumetric grid directly from point clouds, and can be applied on scenes acquired using RGB-D or LiDAR sensors. UWED is simple and efficient as a surface representation, and results in a high quality encoding of point clouds even in presence of noise without the need for high quality normals estimation. Moreover, as we will see in the results section, UWED in able to recover highly detailed fine structures, while SDFs smooth out such fine geometric features.

The extraction of an explicit geometry such as a mesh from a UDF is difficult [Congote et al. 2010]. Thus, we present a method for extracting point clouds from unsigned distance fields. We show that our unsigned distance representation allows different completion networks to better understand the scene's geometry, without managing the sign, and therefore predicts a better completion of the scene.

To validate the advantage of our UDF representation on the SC task, we perform tests on

two different networks. More precisely, for the first network, we adopt a previous neural network architecture, namely SG-NN [Dai et al. 2020] that implements a 3D U-Net architecture using sparse convolutions [Graham et al. 2017] on the encoder side and a custom generative decoder to generate new TSDF values and predict a more complete implicit representation of the scene from which an explicit surface representation can be extracted in the form of meshes or point clouds. To validate that our surface representation is not network specific, we implement another 3D U-Net using Minkowski generalized sparse convolutions [Choy et al. 2019] to encode the feature vector from the input DF representation and generative transposed convolutions (implemented in Minkowski engine) to generate the more complete scene in the form of occupancy grid.

Our contributions are summarized as follows:

- A simple, fast to compute and robust unsigned distance function, UWED, for scene representation, and a method to extract a point cloud from an unsigned distance field computed on a regular 3D grid.

- Extensive comparisons of scene completion using UWED and several other DFs on various environments (indoor and outdoor), tested on different types of datasets (real and synthetic), acquired by different sensors (RGB-D and LiDAR), and validated on two different network architectures, namely SG-NN [Dai et al. 2020] and MinCompNet [Choy et al. 2019].

3.2 Related Work

The increased availability of range sensors and advances in learning-based algorithms have made it possible to obtain large amounts of data and train neural networks for surface reconstruction, and/or to predict larger missing areas caused by occlusions from the foreground and dynamic objects.

For surface reconstruction, Implicit Function Learning (IFL) encodes an implicit representation of the geometry in the latent space and predicts an implicit distance field that can be used to extract the explicit surface in the form of point clouds, or meshes. This family of methods is used for shape representation and completion from sparse scans of small objects.

Two main approaches to geometry completion exist in the literature. Object-level completion [Yuan et al. 2018; Dai et al. 2017b; Yang et al. 2019] focuses on the completion of single objects, such as chairs, cars, and tables. However, they do not scale to large scenes, which limits their use to simple objects. Scene-level completion overcomes the shape completion limitation and attempts to complete the missing geometry in large scenes.

In this section, we first introduce the two main DF representations used for these tasks, and then detail the works done on the different approaches.

3.2.1 Distance Functions

3.2.1.1 Signed Distance Function

Using DFs on 3D volumetric grids for surface reconstruction traces back to the seminal work of Hoppe *et al.* [Hoppe et al. 1992], in which a Signed Distance Field $\phi : \mathbb{R}^3 \rightarrow \mathbb{R}$ was used to implicitly represent the underlying surface in a point cloud:

$$\phi(x) = \mathbf{n}_i \cdot (x - \mathbf{p}_i) \tag{3.1}$$

where x is the voxel coordinates in the 3D grid, $\mathbf{p}_i$ is the nearest neighbor of $\mathbf{x}$ in the point cloud $\mathcal{P}$, and $\mathbf{n}_i$ is the normal unit vector associated with the point $\mathbf{p}_i$.

Having computed the SDF on a 3D regular grid, the Marching Cubes algorithm [Lorensen et al. 1987] extracts the final iso-surface as a mesh. A more detailed description of surface reconstruction methods is available in chapter 4, section 4.2.1.

3.2.1.2 Unsigned Distance Function

UDFs solve the ambiguity introduced by SDFs, since they eliminate the need to define an interior and exterior. However, it is difficult to extract a well-behaved final mesh from such functions. One work [Congote et al. 2010] proposes a variant of the Marching Cubes to be used on an unsigned distance field, but the method results in holes in the reconstructed final mesh. Another work [Mullen et al. 2010] attempts to obtain a consistent signed distance field from unsigned distances on the discretized grid, but this approach comes with the drawback of a high computational cost, not scalable to large scenes. SAL [Atzmon et al. 2020] trains a fully-connected network to infer the sign of a DF from unsigned data but shows this only on object-level data. AnchorUDF [Zhao et al. 2021] trains a network to predict primarly coarse 3D anchors followed by a fine UDF representation on a regular grid to reconstruct open 3D garments surfaces from a single image.

3.2.2 Implicit Function Learning

DeepSDF [Park et al. 2019] learns a continuous SDF representation from point clouds of carefully scanned closed shapes, and unlocks the ability to model multiple classes of objects with the same network, but it is still limited to small and closed shapes. IM-NET [Chen et al. 2019e] designs an implicit decoder that takes as input the point cloud and a latent vector encoding the underlying shape, and predicts an indicator function by performing binary, inside/outside classification and then extracts the final iso-surface using the marching cubes algorithm [Lorensen et al. 1987]. This approach adds another limitation to DeepSDF, which is the reduced accuracy in extracting the iso-surface, caused by predicting an indicator function instead of regressing the DF. DMC [Liao et al. 2018] predicts the final mesh from a point cloud input in an end-to-end fashion instead of predicting the implicit SDF representation and using a surface extraction algorithm. ONeT [Mescheder et al. 2019b] learns the implicit function and predicts binary occupancy, but enables higher resolution explicit surface extraction through subdividing the occupied voxels using an octree structure and predicting a finer inside/outside classification which is later used to extract the final mesh using the marching cubes algorithm. IF-Nets [Chibane et al. 2020a] learns the implicit function from multi-scale point encoding and predicts a binary classification based on local and global shape features.

To solve the problems caused by SDFs, NDF [Chibane et al. 2020b] and DUDE [Venkatesh et al. 2020] learn a volumetric unsigned distance which regresses a UDF from deep features (instead of binary occupancy) and enables the extraction of more accurate sub-voxel resolution when extracting the explicit surface from the predicted distance field. NDF extracts a dense point cloud by projecting points using the gradient field from the back-propagation through the network, while DUDE also learns a normal vector field along with the DF.

Although IFL methods result in high quality final surface representation, while completing small parts, they are limited to small objects, because of their inability to encode large scenes and in case an SDF is predicted, they are also limited to closed surfaces same as SC methods predicting an SDF representation of the scene.

3.2.3 Point Cloud Completion

Point Completion Network [Yuan et al. 2018] (see Figure 3.4) uses an encoder-decoder architecture that extracts local and global feature vectors directly from the input point cloud that is missing some regions. In PCN, 2 levels of details are predicted, first the network

predicts the complete shape by minimizing a combination of the symmetric Chamfer Distance (CD)

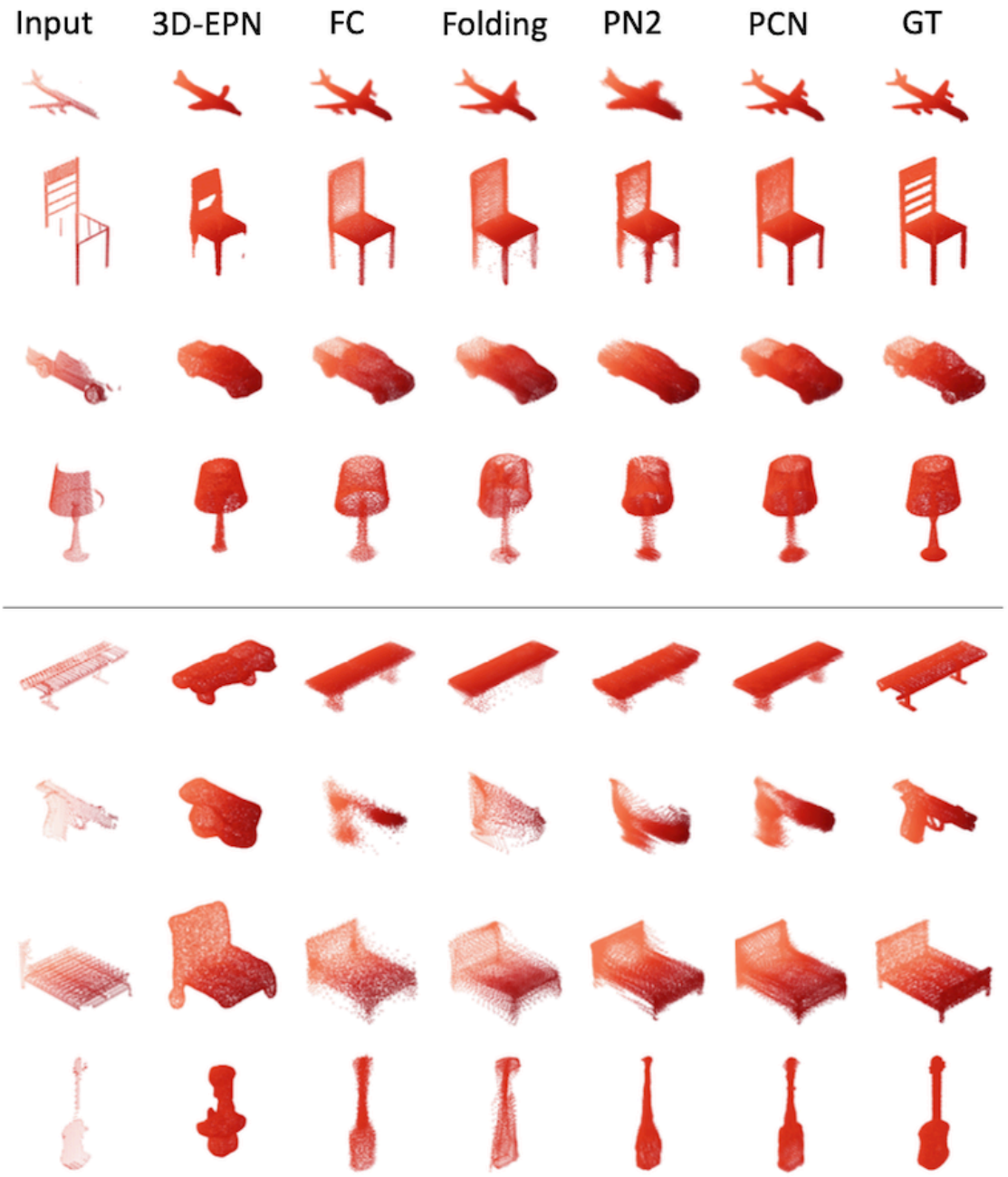


Figure 3.4: Point completion networks example predictions. Source [Yuan et al. 2018]

$$\mathcal{CD}(\mathcal{P}_1, \mathcal{P}_2) = \frac{1}{|\mathcal{P}_1|} \sum_{x \in \mathcal{P}_1} \min_{y \in \mathcal{P}_2} ||x - y||_2 + \frac{1}{|\mathcal{P}_2|} \sum_{y \in \mathcal{P}_2} \min_{x \in \mathcal{P}_1} ||y - x||_2 \tag{3.2}$$

and the Earth Mover's Distance (EMD) (equation 3.3) loss functions

$$\mathcal{EMD}(\mathcal{P}_1, \mathcal{P}_2) = \min_{\phi: \mathcal{P}_1 \to \mathcal{P}_2} \frac{1}{|\mathcal{P}_1|} \sum_{x \in \mathcal{P}_1} ||x - \phi(x)||_2 \tag{3.3}$$

where $\phi : P_1 \to P_2$ is a bijection that finds a flow that minimizes the average distances between matched pairs in the point clouds.

The combined loss is used between a coarse predicted, and sub-sampled target point clouds, to match the resolution of the coarse prediction. For the final detailed predicted and original point clouds, the CD loss is used, due to the high computational complexity required to compute the EMD. Although it provides good completion results, this method does not generalize to other topologies. The authors of TopNet [Tchapmi et al. 2019] tackle the problem of the inability of previous shape completion approaches to take into consideration different topologies. Towards this goal, they design a decoder following a rooted tree to dissect the encoded point cloud and predict the completion on separate parts, before fusing the final full point cloud, which allows the generalization to different topologies. To represent the rooted tree in a deep learning context, they use Multilayer perceptrons to learn embeddings of child nodes from their corresponding parent nodes until the leaf nodes are reached receiving generated feature vectors of dimension 3 representing the generated points in the point cloud. By following this design choice, they are able to increase the domain of represented structures and achieve better completion results on a wider variety of shape classes. In [Wang et al. 2020] the authors propose to use the PCN network architecture and perform 3D features alignment between the complete and incomplete point clouds to learn shape priors and minimize the features matching loss, (e.g., $\mathcal{L}2$ loss, see eq 3.4) and a generative loss at an early stage, and the CD 3.2 loss at inference time to achieve better fine grained completion.

$$\mathcal{L}2_{feat} = \sum_{ij \in \mathcal{F}} ||f_{\mathbf{X}_i} - f_{\mathbf{Y}_i}||_2 \tag{3.4}$$

where $\mathcal{F}$ denotes the pairs of features from the feature sets $f_{\mathbf{X}}, f_{\mathbf{Y}}$. $\mathbf{X}$ and $\mathbf{Y}$ represent the partial and complete point clouds respectively.

SA_Net [Wen et al. 2020] proposes a skip attention mechanism with hierarchical folding to exploit local structure and preserve fine details in the completion. Point cloud completion is an active field of research [Pan et al. 2021; Yu et al. 2021b], but it is used for a different task, which is the completion on small shapes. Many tasks can benefit from this, such as completing vehicles in outdoor environments that can be later used for scenarios generation on pre-existing static backgrounds of urban point clouds, which can be used for data augmentation. In our case, we focus on accurate background completion; therefore, we can not make use of such methods because of their inability to represent large scenes.

3.2.4 Scene Completion

SC, different from point cloud completion methods, extends the completion to large scenes, (1) overcoming the limitation of the former to small shapes and (2) interpolating small missing regions of surface reconstruction.

3.2.4.1 Scene Completion from RGB-D Sensors

SC from a single depth image was introduced to complete missing scenes in indoor settings with SSCNet [Song et al. 2017]. This paved the way for completion beyond traditional surface reconstruction methods, which increased the interest in the SC task [Wang et al. 2018; Zhang et al. 2018]. These earliest works focused on completing the scene from a single depth image and used a TSDF as their scene representation, with TSDF being the signed distance to the closest point on the surface, similar to the implicit representation of Hoppe *et al.* [Hoppe et al. 1992]. They used a flipped version of the TSDF to remove discontinuities around the truncation of the function and get strong gradients around the surface, giving more signal to the network around the surface, enabling it to better learn the geometry. When multiple range images are available, such as in Matterport3D [Chang et al. 2017b], most methods [Dai et al. 2018; Dai et al. 2020] use the volumetric fusion VRIP [Curless et al. 1996] to obtain the discretized TSDF representation of the scene. They also use different network

architectures to infer the missing geometry. For instance, ScanComplete [Dai et al. 2018] uses a U-Net architecture on dense volumes, and SG-NN [Dai et al. 2020] uses a U-Net with sparse convolutions (see Figure 3.5).

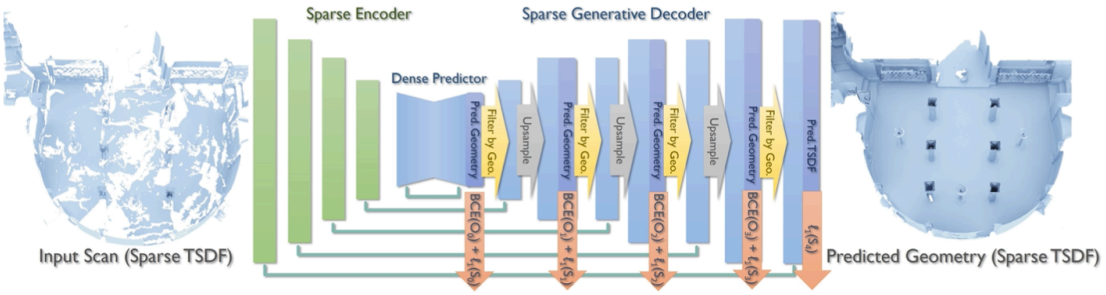


Figure 3.5: Sparse Generative Neural Network SG-NN architecture. Source: [Dai et al. 2020]

SG-NN is a self-supervised SC approach that enables training from real-world datasets. Real-world 3D scans are incomplete (see section 3.1), therefore to learn the task of completion without having ground truth complete data, more scans are removed from the original data to create the input to the network and all of the scans are used as target. In order to complete missing areas, the loss is computed only on known regions, where scans are available in the target data. Completion in SG-NN is done in a coarse to fine strategy by generating multi-resolution volumetric grids containing the DF values, which enables better completion by using multi-scale contextual information. VRIP [Curless et al. 1996] is used to compute the volumetric data from fused RGB-D images, in which only values within truncation (e.g., T=3, where 3 voxels on both sides of the surface contain the DF values) distance are left to generate sparse tensors on which sparse convolution operations can be performed to reduce the computational complexity and memory footprint. The network is trained using a combination of BCE (see equation 3.5) and $\mathcal{L}_1$ losses to ensure a good occupancy and accurate regression of the DF respectively. Sparse convolutions are used [Graham et al. 2017] to generate the feature vectors on the encoder side, while a custom sparse generative decoder is implemented to enable the generation of new coordinates. In our work, we leverage this network architecture, SG-NN, to evaluate the performance of different DF representations.

$$\mathcal{BCE} = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(p(y_i)) + (1 - y_i)\log(1 - p(y_i)) \tag{3.5}$$

Some works perform SSC (survey in [Roldao et al. 2021]), in which they complete the scene, while predicting per-voxel semantic class as well, such as with ScanComplete [Dai et al. 2018].

Other approaches achieve the SSC task using GANs for their effectiveness on generative tasks [Chen et al. 2019c]. In ASSCNet [Wang et al. 2018] a depth image is used as input to an encoder with 2D convolutional operators to extract the feature vector that is fed to a generator with 3D deconvolutions to predict voxels occupancies and semantic classes. For the target, they use a voxelized representation of the scene that is encoded into a latent vector. To compare the latent vector computed from input depth with the latent vector computed from the voxelized scene, they introduce a discriminator for latent features and another discriminator to compare the reconstruction. Others use color information to improve the occupancy completion [Li et al. 2019], while another approach predicts color information as in SPSG [Dai et al. 2021] (which is an extension of SG-NN [Dai et al. 2020]).

Our work, focuses on improving the geometry representation, but could be easily integrated with these works by adding semantic or color information.

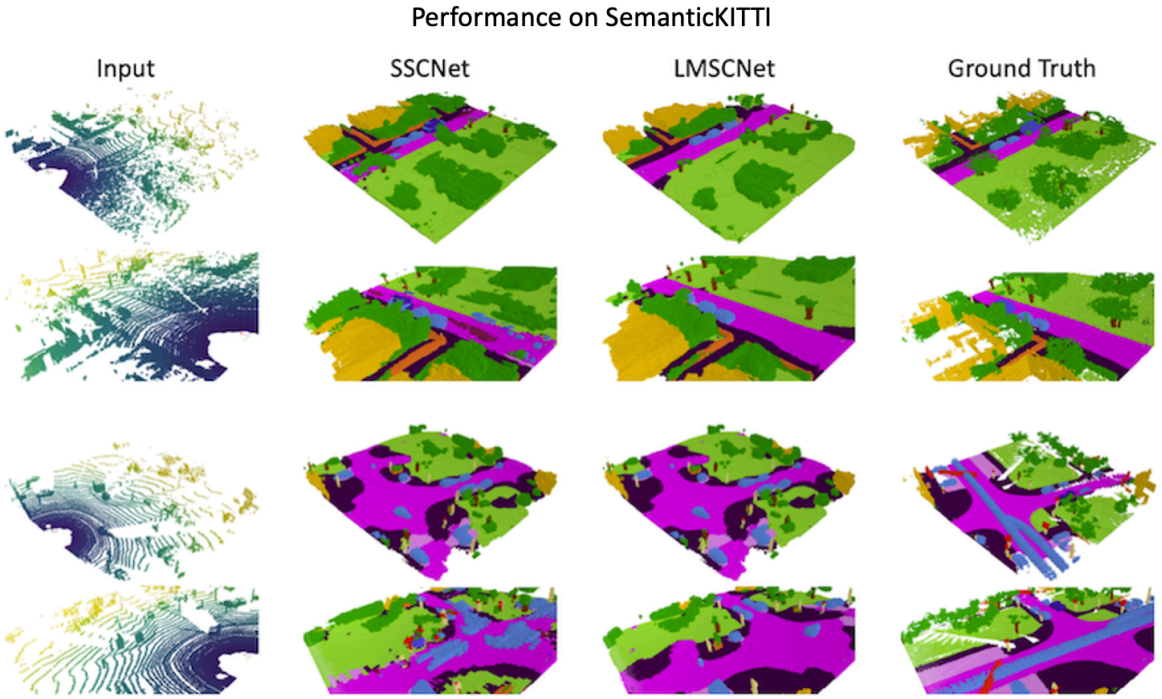


Figure 3.6: Comparing the inferences of LMSCNet [Roldão et al. 2020] and SSCNet [Song et al. 2017] networks on the SemanticKITTI dataset. Source: LMSCNet [Roldão et al. 2020]

3.2.4.2 Scene Completion from LiDAR Sensors

Fewer works were done on SC from LiDAR point clouds. LMSCNet [Roldão et al. 2020] proposes a lightweight 2D U-Net backbone that is used on the x, y dimensions to reduce the computation complexity and complete a scene from a single LiDAR frame. This method provides fast inference, at the expense of a lower geometric prediction accuracy since the input and output are occupancy grids (see Figure 3.6). Another work [Cheng et al. 2020] performs Bird's Eye View (BEV) projection and extracts sparse 2D features along the x-y plane to perform depth completion, they subsequently extract 3D normal information, which is later used to compute the TSDF values on a 3D regular grid (see figure 3.7). JS3C-Net [Yan et al. 2021] learns SSC end-to-end by using a U-Net architecture with sparse convolution [Graham et al. 2017] to first perform SS and then the segmented point cloud is fed to the SSC network that predicts a voxelized coarse completion. Features from the incomplete semantically segmented point cloud and the coarse completed voxel grid are then fed into a Point-Voxel Interaction, proposed by the authors, that uses the coarse predictions and find their nearest neighbors in the incomplete point cloud to predict a finer semantic completion of the scene. The inference of these networks is also a 3D occupancy grid, which does not allow sub-voxel extraction of the surface and hence limits the resolution of the extracted cloud. Figure 3.7 shows an example of the inference of S3CNet [Cheng et al. 2020]. However, previous works [Dai et al. 2017b; Dai et al. 2018] have shown that the use of implicit functions (TSDF) allows better learning of geometry than the use of occupancy grids, while allowing a higher resolution extraction of the completed underlying surface.

We will show that our UWED representation does not depend on a specific network architecture, through implementing a 3D U-Net architecture using Minkowski engine [Choy et al. 2019], and obtain a fast inference and better occupancy completion when compared to occupancy and TSDF representation input.

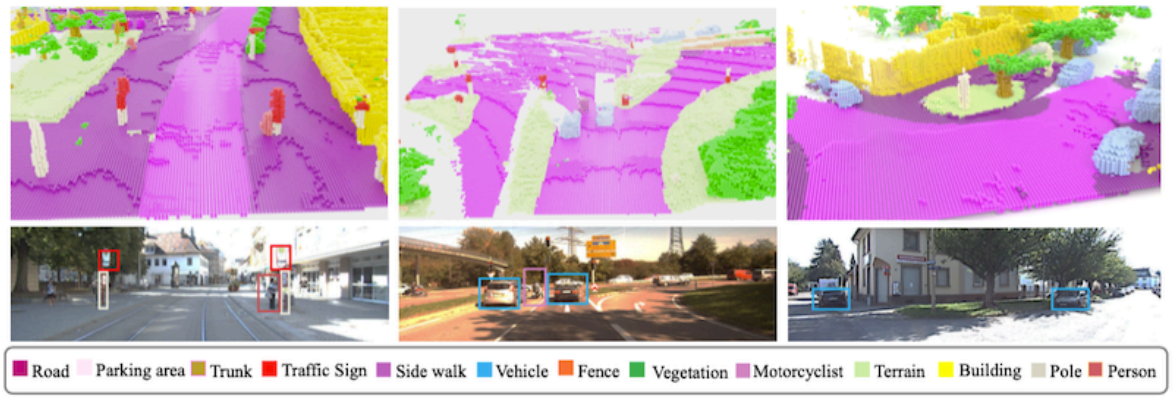


Figure 3.7: Example of the inference of S3CNet. Source: S3CNet [Cheng et al. 2020]

3.3 Scene Representation

3.3.1 Unsigned Weighted Euclidean Distance (UWED)

We propose a UDF that can be directly computed on any point cloud, without the need for normals estimation. Since using the distance to the nearest neighbor results in a noisy distance measurement to the surface, especially when computed on noisy data such as point clouds from LiDAR, we compute a weighted average of distances. Our input is a point cloud $\mathcal{P} = \{\mathbf{p}_i \in \mathbb{R}^3 \mid 0 \leq i \leq \mathcal{N}\}$ that we encapsulate using a 3D regular grid of a predefined resolution and voxel size. Then for each voxel in the regular grid, we compute our $UDF : \mathbb{R}^3 \rightarrow \mathbb{R}^+$ using the k-nearest neighbor (knn) in the point cloud. To weight the contribution of each point in the knn as a function of its distance, we use a Gaussian kernel and call our UDF representation Unsigned Weighted Euclidean Distance (UWED):

$$\text{UWED}(x) = \frac{\sum_{\mathbf{p}_k \in \mathcal{N}_x} ||x - \mathbf{p}_k||_2 \, \theta_k(x)}{\sum_{\mathbf{p}_k \in \mathcal{N}_x} \theta_k(x)} \tag{3.6}$$

where x is the voxel coordinates, $\mathcal{N}_x$ is the set of $\mathbf{p}_k$ neighboring points from x in the point cloud $\mathcal{P}$, and θ_k is the Gaussian weight defined as:

$$\theta_k(x) = e^{-||x - \mathbf{p}_k||_2^2 / \sigma^2} \tag{3.7}$$

where σ is a parameter of the influence of points in the neighborhood. $\mathcal{N}_x$ can then be approximated as a sphere of center x and radius 3σ.

We can see that our UWED function does not require normal information, which greatly reduces the computational cost compared to SDFs. UWED is very close to the classical IMLS function [Kolluri 2008] $SDF : \mathbb{R}^3 \rightarrow \mathbb{R}$:

$$\text{IMLS}(x) = \frac{\sum_{\mathbf{p}_k \in \mathcal{N}_x} \mathbf{n}_k \cdot (x - \mathbf{p}_k) \, \theta_k(x)}{\sum_{\mathbf{p}_k \in \mathcal{N}_x} \theta_k(x)} \tag{3.8}$$

where IMLS is a weighted average of the point-to-plane distances (as in Hoppe [Hoppe et al. 1992]), and UWED is a weighted average of the euclidean distances.

Our formulation results in a smooth function that is robust in the presence of noisy data, which is the case in real-world datasets from RGB-D sequences or LiDAR acquisitions.

After obtaining UWED computed on a 3D regular grid, we convert the values into voxel units and truncate the function at 3 voxels (like in [Dai et al. 2018; Dai et al. 2020]), obtaining the Truncated (T)UWED on a sparse 3D grid.

Our experiments show that we obtain better geometry-learning ability by flipping our UWED function as in [Song et al. 2017]: f-(T)UWED$(x) = 3 -$ (T)UWED(x). We get

smoother gradients between the TUDF values and empty space in the sparse grid by overcoming the discontinuity of the function at truncation distance (visible in Figure 3.3).

3.3.2 Point Cloud Extraction

At training time, a sparse SDF or UDF representation of the scene is fed to a neural network to try and predict a more complete DF representation. For effective and fast learning, the loss is directly computed on the grid containing the DF values. At inference time, we want to obtain an explicit representation of the surface, so we can compare it to the original point cloud and compare the inferences between all of the function as well as use it for surface reconstruction, modeling, rendering, etc... We first show how a point cloud can be extracted from an SDF, similar to extracting meshes from Marching Cubes [Lorensen et al. 1987]. Then we explain our method for extracting a point cloud from a UDF.

Extracting an explicit geometry in the form of a point cloud makes it possible to compare the results directly with the original data, which is also a point cloud (unlike a mesh-to-mesh comparison where the ground truth mesh is constructed by meshing the original point cloud). Point clouds are also a good geometry proxy: it is an easy representation to exploit, since no topology has to be considered. Furthermore, they can also be used directly for rendering with methods such as Splatting [Zwicker et al. 2001] or more recent works such as Neural Point-Based Graphics [Aliev et al. 2020].

3.3.2.1 Extraction from SDF

Marching Cubes [Lorensen et al. 1987] is a method widely used for extracting a mesh from a signed distance field. However, it is also possible to use the same technique to extract a point cloud instead of a mesh. A point cloud can be obtained from an SDF computed on a regular grid, by linearly interpolating a 3D point using the SDF values on each edge between two consecutive voxels having opposite SDF signs. The point cloud created corresponds exactly to the vertices of the mesh extracted using Marching Cubes [Lorensen et al. 1987]. In our experiments, we use this method for point cloud extraction from the different SDFs.

3.3.2.2 Extraction from UDF

Extracting a point cloud from a UDF is not straightforward and the marching cubes can't be directly used on an unsigned distance field, because of the absence of the signs. We propose here a method for extracting a high-quality point cloud from an unsigned distance field. We devise a candidate selection criteria to check if an occupied voxel should be used to extract a point. For each occupied voxel, If all the six neighboring voxels have UDF values (some voxels may not have UDF values because we are using sparse grids) and $1 < \text{udf}_{i,j,k} < 3$ in voxel units, then it is selected as a candidate for extraction. The condition $\text{udf}_{i,j,k} > 1$ guarantees that all edges between the current voxel and its six neighboring voxels do not cross the surface.

We then compute an approximation of the gradient of the unsigned distance field with finite differences and get the unit direction vector as follows:

$$\mathbf{g}_{i,j,k} = \begin{bmatrix} \text{udf}_{i+1,j,k} - \text{udf}_{i-1,j,k} \\ \text{udf}_{i,j+1,k} - \text{udf}_{i,j-1,k} \\ \text{udf}_{i,j,k+1} - \text{udf}_{i,j,k-1} \end{bmatrix}, \mathbf{d}_{i,j,k} = \frac{\mathbf{g}_{i,j,k}}{||\mathbf{g}_{i,j,k}||} \tag{3.9}$$

We obtain the coordinates of the new extracted point on the surface by projection, through the use of the UDF value at the current voxel and the estimated direction vector:

$$\mathbf{p}_{new} = \mathbf{v}_{i,j,k} - \text{v\_size udf}_{i,j,k}\ \mathbf{d}_{i,j,k} \tag{3.10}$$

where $\mathbf{v}_{i,j,k}$ is the 3D coordinates of the current voxel, v_size is the voxel size and $\text{udf}_{i,j,k}$ is the scalar representing the unsigned distance in voxel units.

This method does not need the sign to find the surface and extracts better point clouds than those extracted from SDFs, as shown in the experiments section.

Finally, if a mesh is required as the explicit surface representation, we can apply meshing techniques such as Screened Poisson [Kazhdan et al. 2013] on the new, more complete, point cloud of the scene.

3.4 Experiments and Results

We show that using our UWED function as an input representation improves the SC task and helps neural networks to better understand the geometry of the scene. We train two different networks on the task of SC. For the first network, we use SG-NN U-Net architecture (taken from [Dai et al. 2020]) with SparseConvNet convolutions (from [Graham et al. 2017]) that regresses distance fields on a 3D grid. The second network, MinCompNet, is also a U-Net architecture, using Minkowski generalized convolutions [Choy et al. 2019] with generative transposed convolutions [Gwak et al. 2020]. MinCompNet predicts only a binary occupancy grid. As input to the network, we use the different DF representations (signed and unsigned) and compare the predicted point clouds to target point clouds. For SG-NN, we extract the point cloud from the predicted DF representations with the methods explained in section 3.3.2. For MinCompNet, we use the predicted occupied voxel coordinates as the predicted point cloud.

We provide quantitative and qualitative results on the SC task, for which we experiment on a wide variety of scene point clouds, including real indoor scenes from RGB-D sensors using Matterport3D [Chang et al. 2017b], synthetic outdoor scenes from a simulated LiDAR using CARLA simulator [Dosovitskiy et al. 2017b] that we call PC3D-CARLA, and outdoor scenes from a real mobile LiDAR system that we call PC3D-Paris. For the synthetic and real outdoor datasets, we use the Paris-CARLA-3D (PC3D) dataset [Deschaud et al. 2021]. The tested datasets have a wide variety of noise, missing data and sparsity.

3.4.1 Distance Functions for Scene Completion

We compare six DFs which consist of signed and unsigned versions of the point-to-plane distance to the nearest neighbor (Hoppe [Hoppe et al. 1992]), the weighted average of the point-to-plane distances in a given neighborhood radius (IMLS [Kolluri 2008]), and our proposed function.

For SDFs, we compare Hoppe [Hoppe et al. 1992] (the same function used to set the baseline in chapter 2), IMLS [Kolluri 2008] and Signed Weighted Euclidean Distance (SWED), which is the signed version of UWED, using IMLS for the sign and computed as follows:

$$SWED(x) = sign\left(IMLS(x)\right)\, UWED(x). \tag{3.11}$$

For the UDFs, we test several functions in order to have a complete comparison with our proposed UDF. Specifically, we use the unsigned versions of the SDFs, obtaining UHoppe, UIMLS, and UWED, where UWED is the UDF introduced in section 3.3.1.

After obtaining the different UDFs and SDFs, we convert them into voxel units and truncate the values at 3 voxels.

We do not use robust variants of SDFs such as FSS [Fuhrmann et al. 2014] or RIMLS [Öztireli et al. 2009] because they require more parameters, unlike all the functions tested, which have at most one parameter σ, as the objective is to show the advantage of using unsigned distance fields as a weighted average of Euclidean distances over the most used SDFs like Hoppe or IMLS.

First proposed in [Song et al. 2017], a flipped version of TSDF for completing partial scans, flips the function to remove strong gradients at truncation distance from the surface and give more signal to the network near the surface. Thus, we include the non-flipped and flipped versions of all the DFs in the comparisons.

For notation simplification, we remove the T from the SDF and UDF names, but all tested DFs are truncated at 3 voxels and are defined on sparse 3D grids.

3.4.2 Dataset Preparation

We tested the different DF representations and SC on three different datasets, namely, Matterport3D [Chang et al. 2017b] (RGB-D frames in indoor environments), PC3D-CARLA (mobile LiDAR simulated in the virtual simulator CARLA [Dosovitskiy et al. 2017b]) and PC3D-Paris [Deschaud et al. 2021] (mobile LiDAR in real outdoor environments). These three datasets provide a wide variety of point clouds (indoor and outdoor environments), noise types (RGB-D and LiDAR sensor) with synthetic and real data.

To effectively evaluate the different DFs on real data where we do not have ground truth, we follow the work done in SG-NN [Dai et al. 2020] where a percentage of data is removed in order to create incomplete point clouds for the network input and all of the points are used for the target point clouds (see Figure 3.2).

To split the datasets, we use the original Matterport3D and PC3D splits train/val/test.

We remove 50 % of RGB-D frames and 90 % of LiDAR scans to build an incomplete input point cloud, compute the SDF or UDF values from it, and try to predict a new SDF or UDF using the DFs values computed on the original point cloud as the target.

For Matterport3D, we first convert the RGB-D frames into point clouds and keep the sensor positions to re-orient the computed normals needed for the SDFs. We extract 30 point cloud blocks from each room, which results in 12,000 point clouds (64x64x128 grid size with a voxel size of 2 cm) for training. The test set is composed of 6,000 point clouds from the test set rooms.

For PC3D-CARLA, we extract 3,000 small blocks (128x128x128 grid size with a voxel size of 5 cm) from each of the four training towns (T_2, T_3, T_4, T_5), resulting also in 12,000 point cloud blocks for training. The test set is composed of 6,000 point clouds from towns T_1 and T_7.

We follow the same procedure for PC3D-Paris with 2,000 training point cloud blocks (128x128x128 grid size with a voxel size of 5 cm) from S_1 and S_2. For PC3D-Paris, we use only the S_3 test set, as S_0 contains vegetation that does not make sense for surface reconstruction. As a result, the test set is composed of 1,000 point clouds.

We perform data augmentation strategies, such as random rotations around z, random scaling between 0.8 and 1.2, and local noise additions.

For all the datasets, we compute the different DFs on a regular grid, convert the values into voxel units and truncate the function at 3 voxels. As in SG-NN [Dai et al. 2020], we decrease the resolution by a factor of two each time to generate 4 hierarchical levels to train on SG-NN and use the highest resolution only to train MinCompNet.

In the blocks generation step, the seed points are chosen at random from the full point clouds. In PC3D, we observe that most of the points are contained in the road, which is expected, since it is the closest surface to the LiDAR and this gives a high probability for the seed points to be chosen on the road. To prevent this from happening, we subsample the full point cloud, where we keep 1 point from the original point cloud every 2 voxels, with a voxel size of 5 cm and use it only to choose the seed points.

3.4.3 Evaluation Metrics for Scene Completion

Previous methods such as ScanComplete [Dai et al. 2018] or SG-NN [Dai et al. 2020] using SDFs for the scene completion (SC) task, compute the $\mathcal{L}_1$ distance between the predicted SDF and the target SDF. However, $\mathcal{L}_1$ results cannot be compared between different distance functions (DFs).

As we are extracting point clouds from the predicted signed distance functions SDFs and unsigned distance functions (UDFs) (sections 3.3.2.1 and 3.3.2.2) from the predictions of SG-NN and have the occupancy from MinCompNet, we can directly compare the extracted point cloud with the target cloud (which is the original raw point cloud). The most often used metric for comparing point clouds is the Chamfer Distance (CD) between the original $\mathcal{P}_1$ and predicted $\mathcal{P}_2$ point clouds.

$$mCD = 0.5\Big(\frac{1}{\mathcal{N}}\sum_{i=1}^{\mathcal{N}} CD(\mathcal{P}_i^o, \mathcal{P}_i^p)\Big) \tag{3.12}$$

where $\mathcal{P}_i^o$ is the original i^{th} block and $\mathcal{P}_i^p$ is the predicted i^{th} block, and CD is equation 3.2. We note here that we compute the Chamfer Distance on the whole cloud without using the occupancy mask as we did in chapter 2. Although the mask was used on originally occupied voxels and measures the accuracy of predictions, we do not use it here, instead we compute the CD on the whole clouds to penalize more predictions that are far from the original surface. For this reason, we can see that the function hoppe does not give the same results as in chapter 2.

3.4.4 Implementation Details

We train MinCompNet and SG-NN on an NVIDIA GeForce RTX 2070 SUPER. For SG-NN, the loss is a combination of Binary Cross Entropy (BCE) on occupancy and $\mathcal{L}_1$ loss on SDF or UDF predictions. For MinCompNet, the loss is the BCE on the occupancy grid. The trainings were done using ADAM optimizer, a learning rate of 0.001 and a batch size of 8 for both neural networks. We include the number of epochs and training time details in Table 3.1. The training time is less for MinCompNet with a higher number of epochs than SG-NN due to the fact that the convolution operation is faster with MinkowskiEngine. The completion inference on GPU of a 128x128x128 grid (with a 5 cm voxel size) and extraction of a point cloud on CPU is fast and takes around 0.6 seconds for SG-NN (resulting in a point cloud with an average of 150,000 points) and 0.07 seconds for MinCompNet (resulting in a point cloud with an average of 23,000 points).

	SG-NN		**MinCompNet**	
Datasets	**#epochs**	**Time**	**#epochs**	**Time**
Matterport3D	5	24h	10	9h
PC3D-CARLA	5	24h	10	14h
PC3D-Paris	10	10h	50	8h

Table 3.1: Number of epochs and time taken to train each network on the different datasets. Time is reported in hours.

3.4.5 Validity of our UDF as Scene Representation

Before comparing the influence of DFs on the SC network, we evaluate the capacity of the different implicit representations by computing the error introduced from passing through the

different representations: point cloud → DF → point cloud. We then compute the CD between the original and the corresponding extracted point clouds from the different distance fields. We used the test sets of the three datasets with 6,000 point cloud blocks for Matterport3D, 6,000 point cloud blocks for PC3D-CARLA, and 1,000 point cloud blocks for PC3D-Paris. We compute the mean CD over all point cloud blocks, see equation 3.12

Figure 3.8 shows the qualitative results of these comparisons. We can see that Hoppe is very noisy, only exploiting the closest point, and that Hoppe and IMLS expand the geometry at the borders. This is clearly visible on the stair bar for Matterport, the street lamp in the PC3D-CARLA synthetic data, and the window jamb for the PC3D-Paris data. On the contrary, with our UWED, we extract a geometry in the form of a point cloud that is very close to the original cloud.

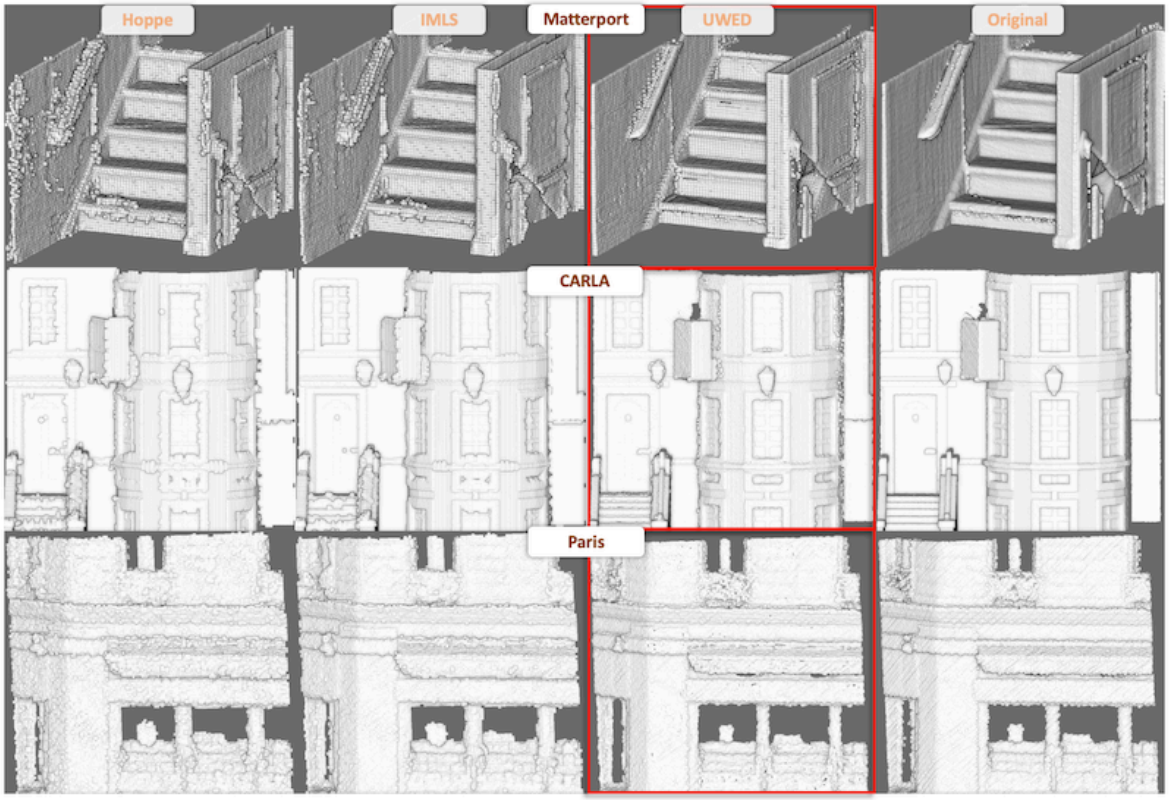


Figure 3.8: Comparison of different implicit functions representations. From point clouds, we compute Signed Distance Functions (Hoppe and IMLS) or our Unsigned Distance Function (UWED), then we extract back a point cloud (without completion) to compare the accuracy of the extracted point clouds to the original point cloud. With UWED, we are able to extract smooth point clouds: the advantages can be seen all over the extracted point clouds, such as on the stair bar in Matterport and the window in the Paris dataset (Hoppe and IMLS dilate the geometry because of the sign and point-to-plane distance functions).

For quantitative evaluation, Table 3.2 shows the mean CD results on the three datasets. We can see that our point clouds extraction from UWED is always better for all datasets. This shows that our UDF and our point cloud extraction method from our UDF (through approximate gradients by finite differences) works and allows the retrieval of an accurate point cloud.

The parameter σ used for the DFs IMLS, SWED, UIMLS, and UWED allows the functions to be robust to noise. The choice of sigma stems from the analysis on the smallest features in outdoor and indoor settings, and follows the choice of voxel size. To preserve sharp features in the scene, such as the step between the road and the sidewalk in outdoor point clouds,

	SDFs			UDFs		
Datasets	**Hoppe**	**IMLS**	**SWED**	**UHoppe**	**UIMLS**	**UWED**
Matterport3D	0.90	0.83	0.81	0.80	0.74	**0.60**
PC3D-CARLA	1.80	1.78	1.97	1.67	1.68	**1.62**
PC3D-Paris	2.41	2.25	2.44	2.90	2.72	**1.94**

Table 3.2: Mean Chamfer Distance (in cm) between the extracted point clouds from the different DFs and the original point clouds, used to compare the accuracy of each function without passing through SC networks.

and the angle on the stairs in indoor datasets, sigma is fixed to 2 times the voxel size for all datasets.

To validate the choice of sigma, we provide in Table 3.3 quantitative results on scene representation, ranging sigma between one and four times voxel size using 2 different DFs, namely, a IMLS (classical SDF) and our UWED proposition. We see the same behavior between IMLS and UWED functions based on σ. Consequently, we keep σ as two times the voxel size, the best trade-off for a fair comparison between the different distance functions based on σ.

For DFs that require normal computation on the point clouds (all except UWED), we estimate the normal at each point using PCA with $n = 30$ neighbors and obtain a consistent orientation using the RGB-D or LiDAR sensor position provided with the points.

For the data generation (completion blocks), the bottleneck is the tree search to find neighboring points. In order to decrease the search time, we inspire from sparse convolutions and perform a sparse knn search only for neighbors that are 3 unit voxels away from the surface of the point cloud, which returns a sparse volumetric grid that can be used as input to the network. See Figure 3.9 for a simplified 2D example.

	Variation of σ for IMLS / UWED			
Datasets	**1 VS**	**2 VS**	**3 VS**	**4 VS**
Matterport3D	0.85/**0.60**	**0.83**/**0.60**	0.87/0.61	0.87/0.61
PC3D-CARLA	**1.78**/1.96	**1.78**/**1.62**	1.82/1.75	1.88/1.83
PC3D-Paris	2.32/2.42	**2.25**/**1.94**	2.68/2.08	2.60/2.23

Table 3.3: Mean Chamfer Distance (in cm) between raw and extracted point clouds from IMLS and UWED representations while varying the parameter σ between one and four times the Voxel Size (VS).

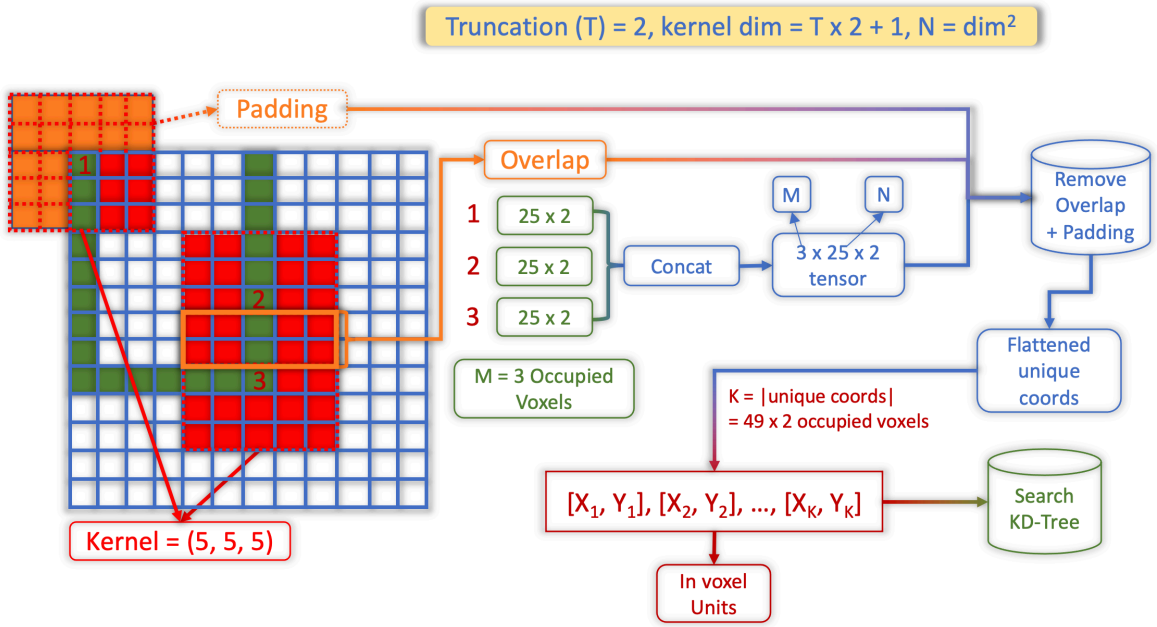


Figure 3.9: Sparse search 2D example. Starting from a grid encapsulating the point cloud (green quadrants), where points are converted into voxel coordinates. We use a kernel (red squares, here 5x5) of shape (Truncation $(T) * 2 + 1)^2$ here T = 2. We extract two voxels from all sides of the occupied voxels (green quadrants), which includes the current voxel and the 4 neighboring voxels $[-2, +2]$ on x and y. These coordinates contain overlaps, so we filter out the repeated ones and padded coordinates, then perform the search on the rest, where the final tensor containing the coordinates is flattened and queried from a KD-Tree.

3.4.6 Results of Distance Functions for Scene Completion

For SG-NN, we compare the six DFs (with non-flipped and flipped variants) on the three datasets, which results in a total of 36 trainings. For MinCompNet, we compare as input representation an occupancy grid, the flipped variants of IMLS (most robust signed function), and our UWED representation, resulting in 9 trainings. All quantitative results following training SG-NN are visible in Table 3.4, while the results on MinCompNet are available in Table 3.5. We present the mean CD (equation 3.2) across all test sets of the three datasets (6,000 for Matterport3D, 6,000 for PC3D-CARLA, and 1,000 point clouds for PC3D-Paris). We can see that our UWED representation (flipped version) results in a lower CD for all datasets and for the two SC networks.

We can see in Table 3.4 that the unsigned version of IMLS (UIMLS) produces worse results than the signed version, meaning that in general, UDFs do not outperform SDFs. This comes from the fact that in the unsigned version, UIMLS, the boundary expansion effect is still present (due to the zero-level of point-to-plane distances). This confirms the interest in using Euclidean distance in our UWED formulation.

We can note from Table 3.4 that systematically flipping the DFs improves results for UWED but not necessarily for SDFs. Indeed, for SDFs, the flipped function removes the discontinuity at the truncation distance and gets stronger gradients around the surface. However, it introduces a discontinuity at the surface level (iso-zero).

Figures 3.10, 3.11, and 3.12 show qualitative results of SG-NN inferences for Hoppe, IMLS, and UWED. Figures 3.13, 3.14, and 3.15, show qualitative results of MinCompNet inferences for occupancy, IMLS, and UWED. All DFs used for qualitative evaluations are flipped versions, and all inferences were done on the test sets of the three datasets. We can see each time that higher quality point clouds are obtained after SC using our UWED repre-

	Matterport3D		**PC3D-CARLA**		**PC3D-Paris**	
	SDFs					
	non-flipped	flipped	non-flipped	flipped	non-flipped	flipped
Hoppe	1.54	1.53	6.32	6.14	7.62	7.53
IMLS	1.52	1.51	6.10	6.09	7.48	7.56
SWED	1.52	1.54	6.06	6.19	7.57	7.54
	UDFs					
	non-flipped	flipped	non-flipped	flipped	non-flipped	flipped
UHoppe	1.25	1.26	15.24	6.00	9.95	7.96
UIMLS	1.27	1.32	7.76	6.68	9.47	7.73
UWED	1.23	**1.22**	6.28	**5.73**	7.45	**7.27**

Table 3.4: SG-NN scene completion results for different signed and unsigned distance functions on three datasets: Matterport3D (RGB-D frames in indoor), PC3D-CARLA (mobile LiDAR in virtual CARLA simulator), and PC3D-Paris (mobile LiDAR in outdoor urban scene). Results are the mean Chamfer Distance computed over the test sets of the three datasets (6,000, 6,000, and 1,000 points clouds, respectively).

Input / **Datasets**	**Occupancy**	**SDF IMLS**	**UDF UWED**
Matterport3D	1.70	1.75	**1.68**
PC3D-CARLA	5.67	4.75	**3.98**
PC3D-Paris	4.67	4.22	**4.0**

Table 3.5: MinCompNet scene completion results of using IMLS, UWED and Occupancy grid as input to predict occupancy grids that are later extracted as point clouds and compared to the original point clouds on three datasets: Matterport3D (RGB-D frames in indoor), PC3D-CARLA (mobile LiDAR in virtual CARLA simulator), and PC3D-Paris (mobile LiDAR in outdoor urban scene). Results are the mean Chamfer Distance computed over the test sets of the three datasets (6,000, 6,000, and 1,000 points clouds, respectively).

sentation (visible on stairs, trash can, and bollards). UWED is also able to complete larger missing regions (middle example, Figure 3.10). Comparing the results in Tables 3.4 and 3.5, we notice that better completion is achieved with SG-NN on Matteport3D (the network has been designed on that dataset, made of RGB-D frames in indoor), while MinCompNet has a lower CD on the two other datasets (made of LiDAR scans with sparser data). MinCompNet seems to be able to complete larger missing areas thanks to the generative transposed convolutions Gwak et al. 2020, but SG-NN still has the advantage of more accurate geometry prediction (see UWED in third row of Figure 3.12 and third row of Figure 3.15), due to the fact that it predicts a continuous DF representation. In any case, our representation gives better results than a binary occupancy grid or a classic SDF.

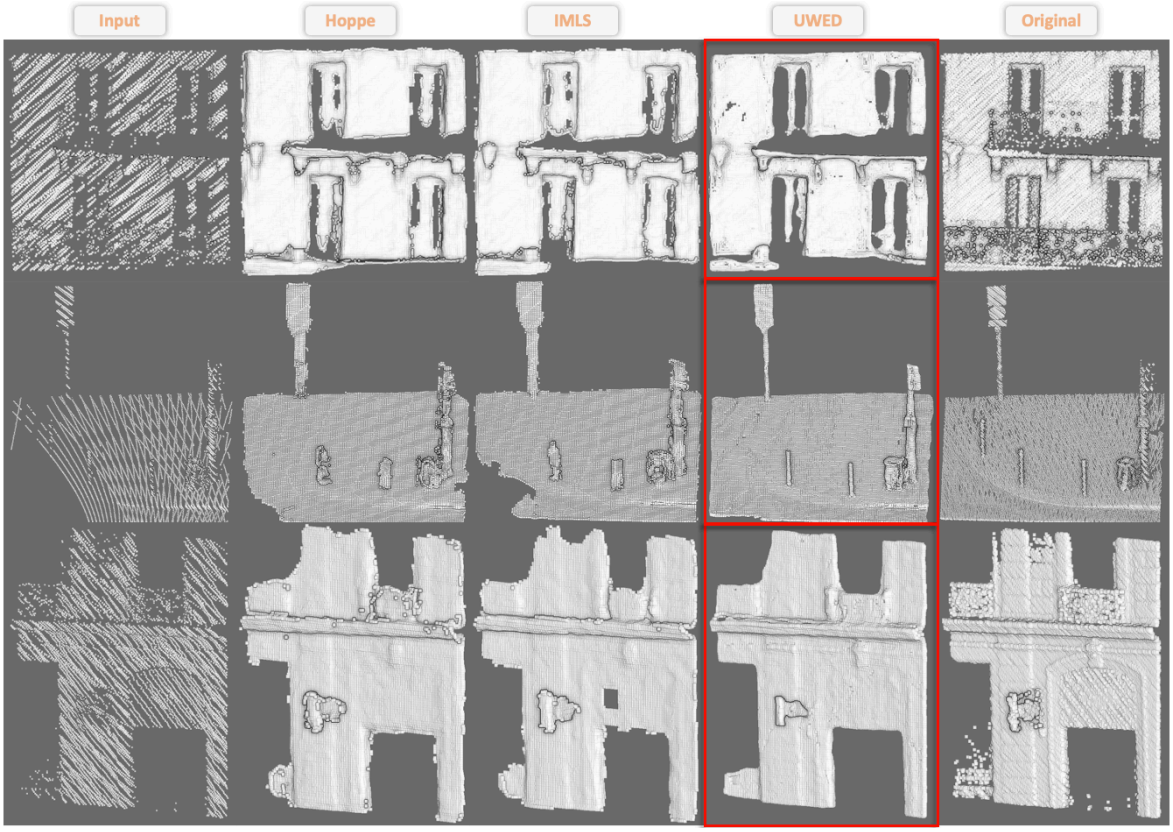


Figure 3.10: Scene completion inference of SG-NN using Hoppe, IMLS, and our UWED function on the PC3D-Paris test set (input has 10% of the original points).

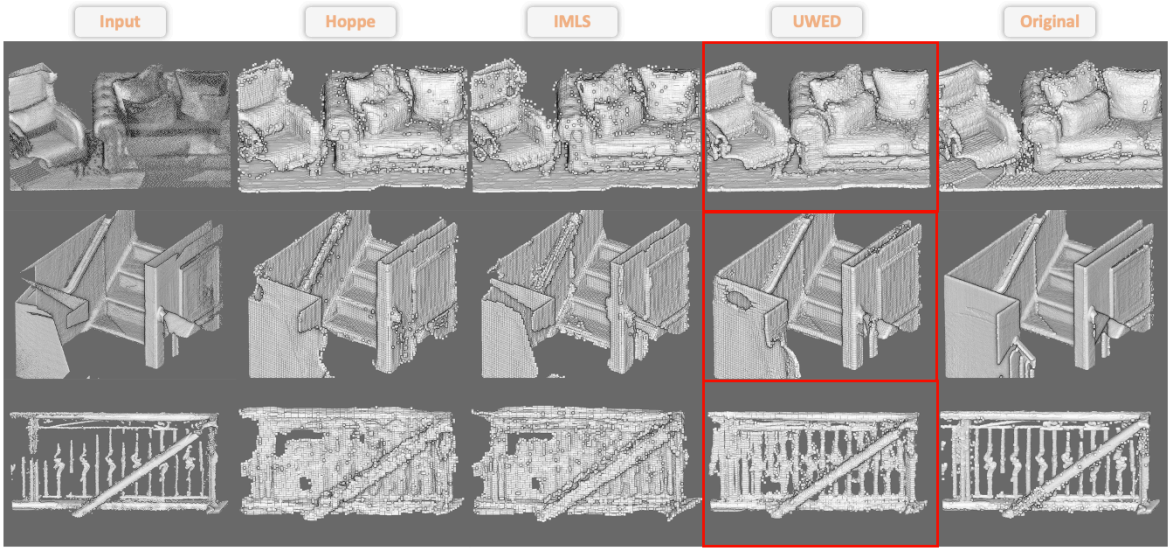


Figure 3.11: Scene completion inference of SG-NN using Hoppe, IMLS, and our UWED function on the Matterport3D test set (input has 50% of the original points).

Figure 3.12: Scene Completion inference of SG-NN using Hoppe, IMLS, and our UWED function on the PC3D-CARLA test set (input has 10% of the original points).

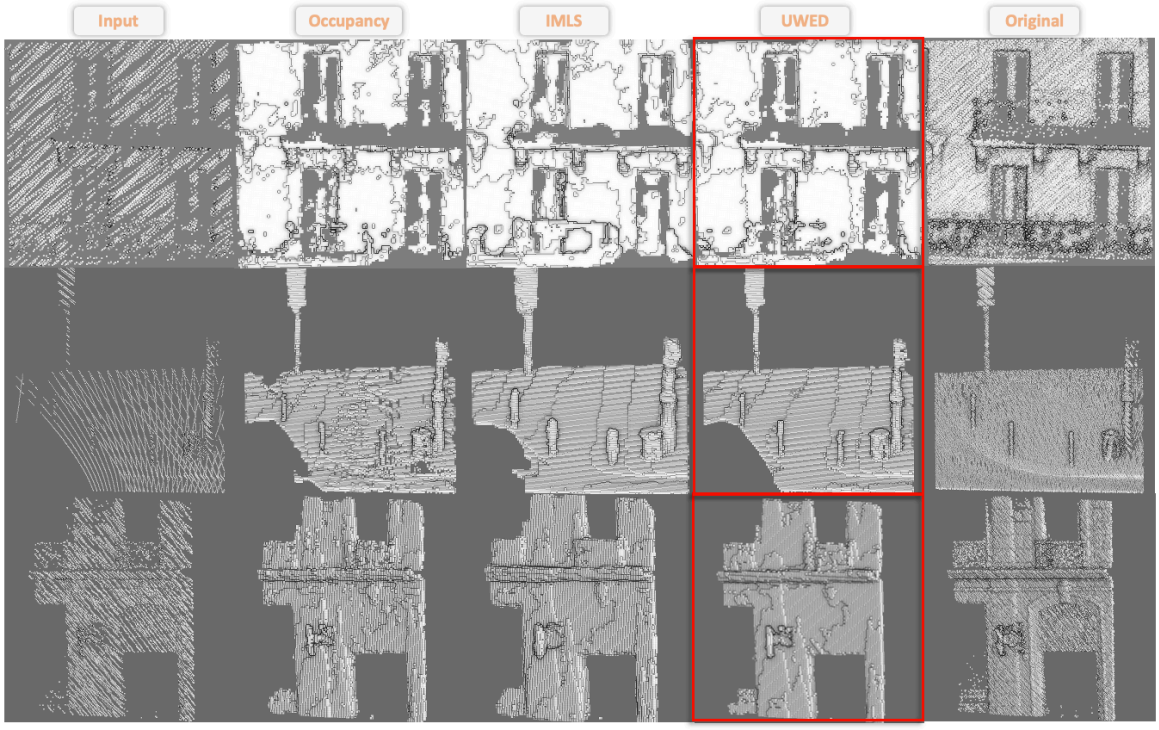


Figure 3.13: Scene completion inference of MinCompNet using Occupancy, IMLS, and our UWED function as input on the PC3D-Paris test set (input has 10% of the original points) and predicting occupancy.

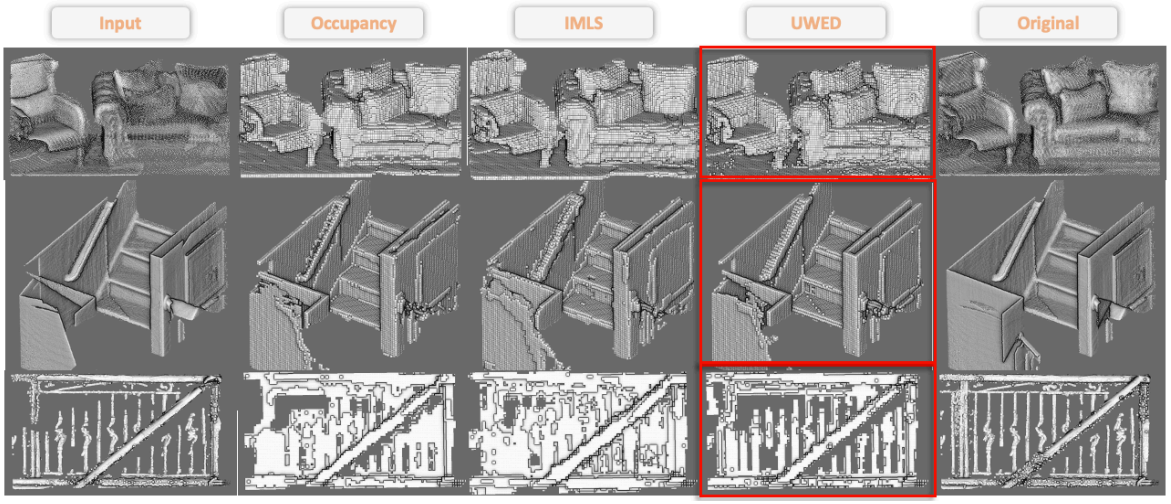


Figure 3.14: Scene completion inference of MinCompNet using Occupancy, IMLS, and our UWED function as input on the Matterport3D test set (input has 50% of the original points) and predicting occupancy.

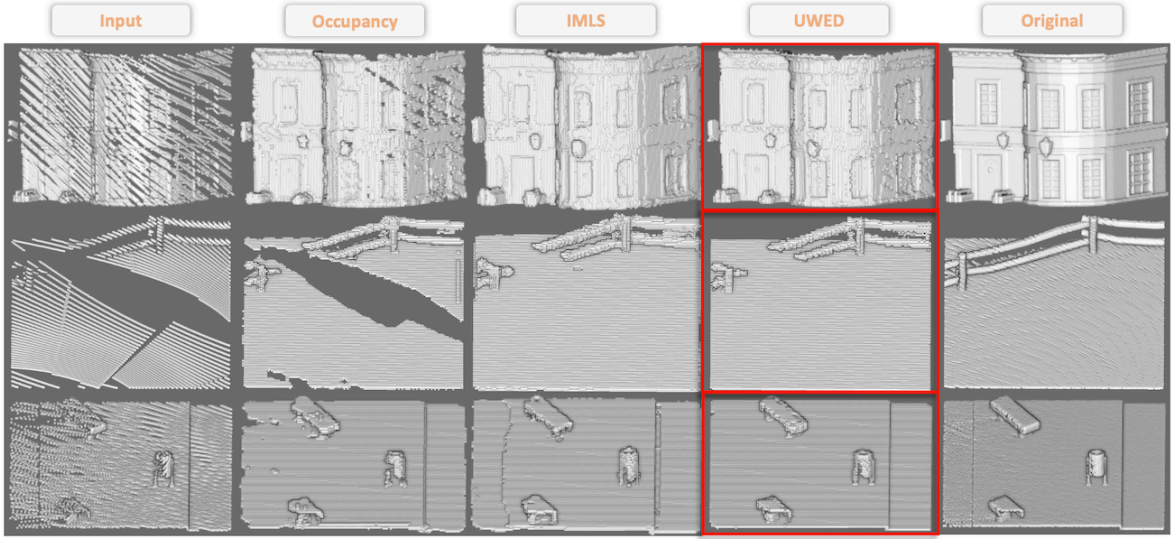


Figure 3.15: Scene completion inference of MinCompNet using Occupancy, IMLS, and our UWED function as input on the PC3D-CARLA test set (input has 10% of the original points) and predicting occupancy.

3.4.7 Scene Completion on Target

The next step in our automatic reality replication pipeline requires an accurate geometry representation of the scene, which can be provided as the original, or the completed point cloud. Our UWED function is able to accurately predict missing regions caused by the removal of dynamic objects. To complete these regions, we perform a full training of the SG-NN network, consisting of 20 epochs on PC3D-CARLA, then finetune it on PC3D-Paris for 50 epochs, and perform the completion on the original point cloud.

Since our aim is to complete the missing regions on the accumulated point cloud, we do not remove scans, instead we use all of the available points, so we can complete the original point cloud even more and use the more complete version as input to the scene modeling.

After completion, we extract the point cloud $\mathcal{P}_{pred}$ from the predicted UWED. $\mathcal{P}_{pred}$ is not used for modeling in its entirety, instead, we leverage the original point cloud $\mathcal{P}_{ori}$ data and map points from $\mathcal{P}_{pred}$ to $\mathcal{P}_{ori}$ only from regions where we did not originally have points. To identify which points should be mapped, we perform a radius search for the nearest neighbors of $\mathcal{P}_{ori}$ in $\mathcal{P}_{pred}$ and remove all of the neighboring points in $\mathcal{P}_{pred}$ that are within a 10 cm radius. Finally we merge $\mathcal{P}_{ori}$ and $\mathcal{P}_{pred}$. For a high-quality modeling and rendering, we need the semantic labels and RGB values for the new points in the final point cloud, to obtain these information, we assign the RGB color and semantic class for the new points from the nearest neighbor in $\mathcal{P}_{ori}$. Figures 3.16, 3.17 and 3.18, 3.19 show the completed PC3D-Paris without and with point-wise semantic labels, respectively.

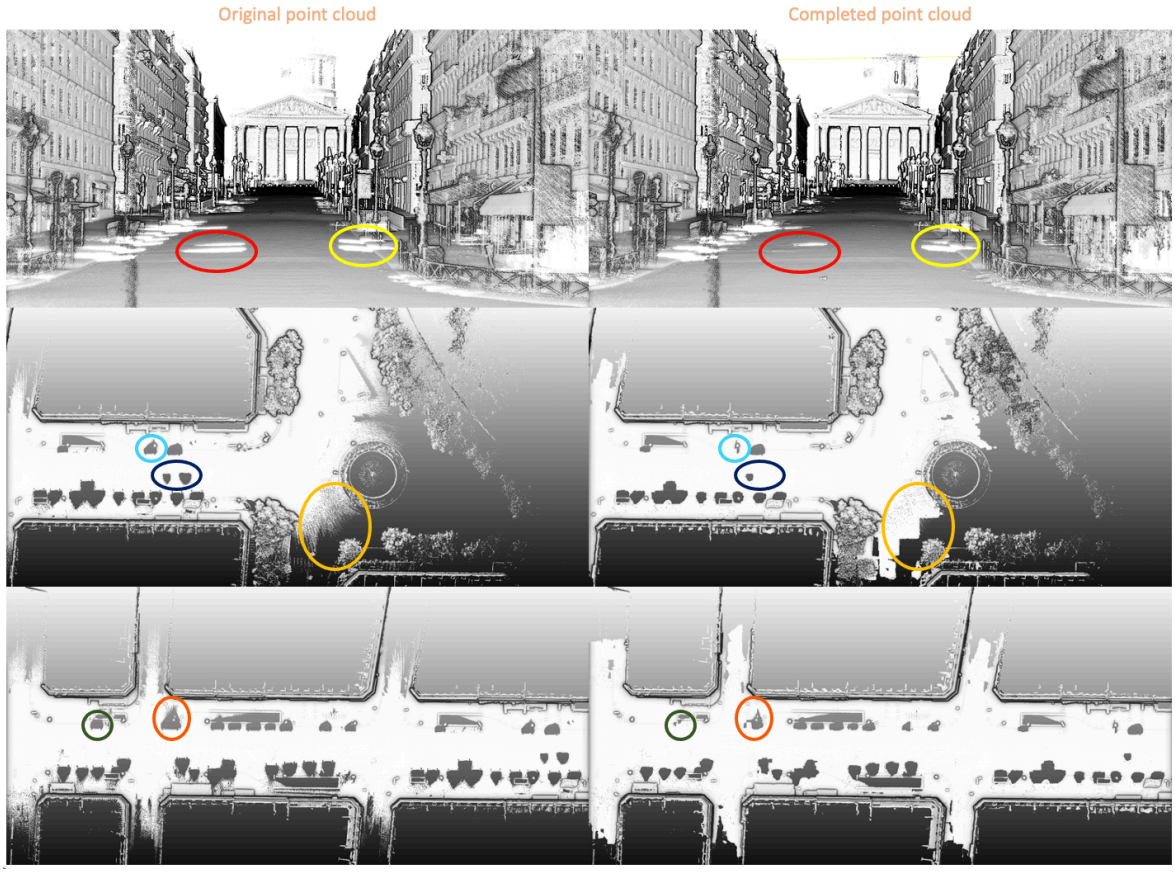


Figure 3.16: Scene completion performed on the target point cloud.

We can see from the qualitative results that we obtain a more complete point cloud representation of the original surface. We observe that the network is able to predict some of the large missing regions in their entirety (see Figure 3.16, second row in blue), while reducing the size of others, which can be seen throughout all of the completion figures where we have

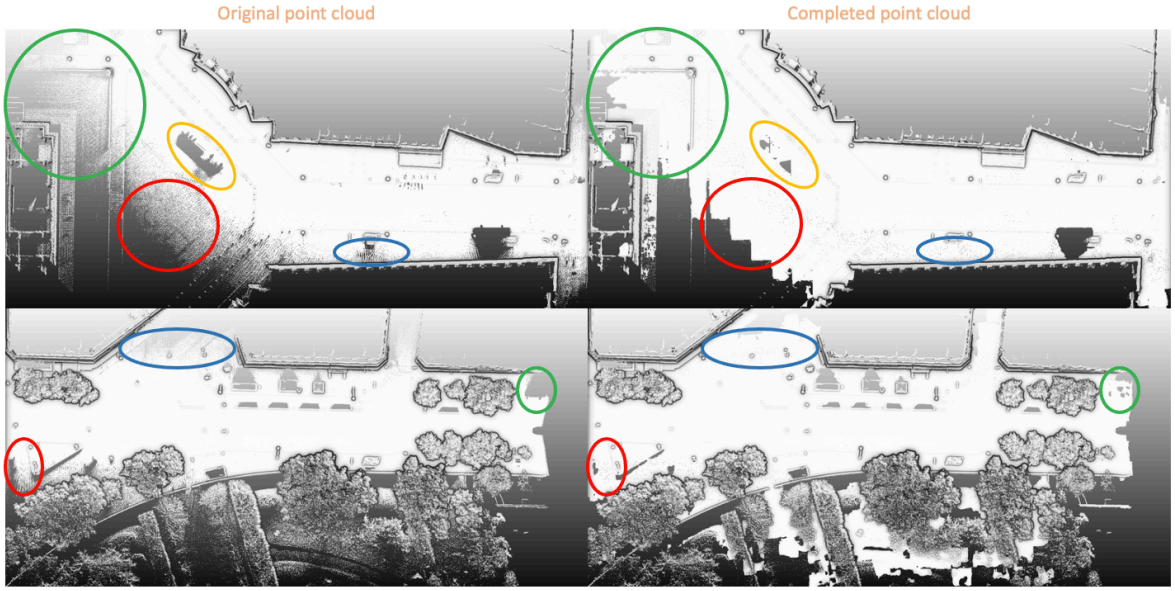


Figure 3.17: More results of scene completion performed on the target point cloud.

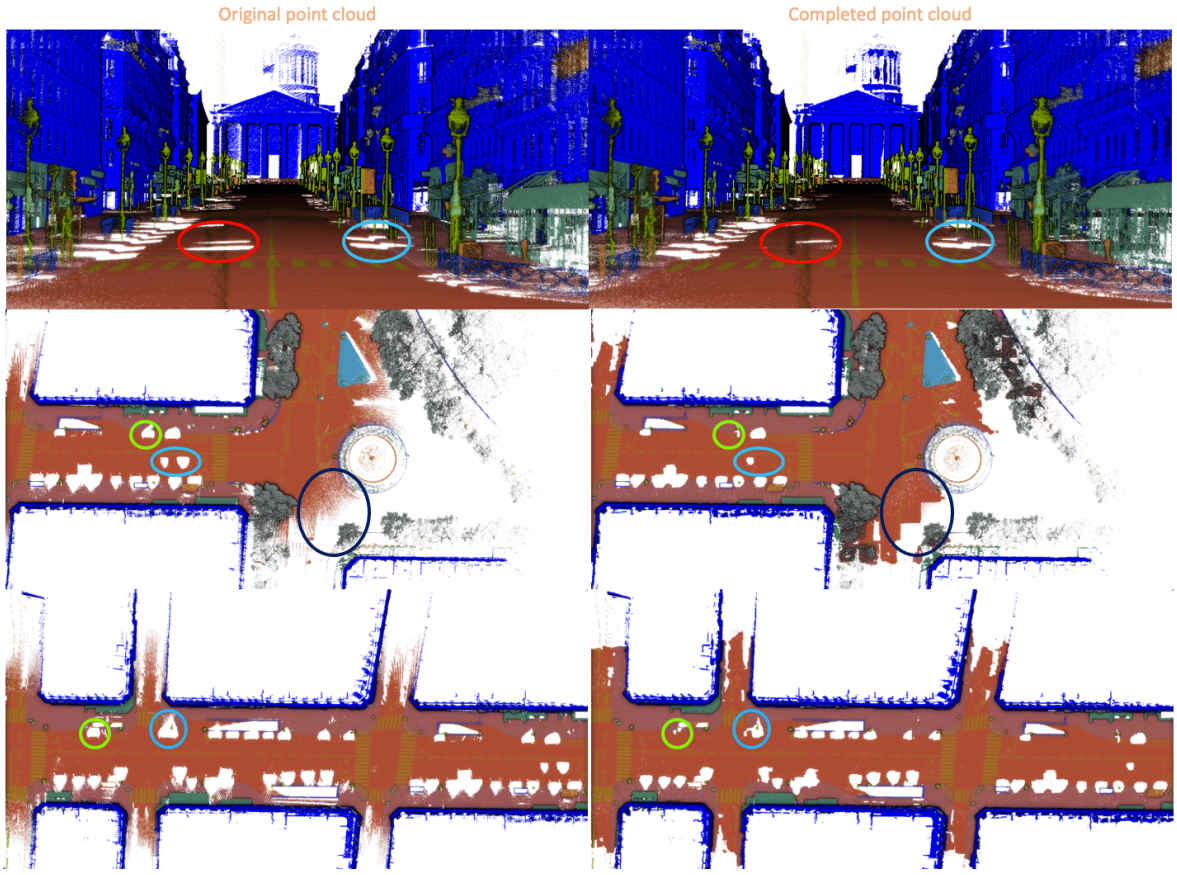


Figure 3.18: Scene completion performed on the target point cloud with point-wise semantic information taken from the nearest neighbor.

cars occlusions. More specifically, see Figure 3.17 first row in orange. In our data preparation step, we approach the scene completion problem by removing a big percentage of the scans and attempting to predict them, which still relies on some points in the neighborhood to try and predict a more complete region (see Figure 3.17 first row in green and red).

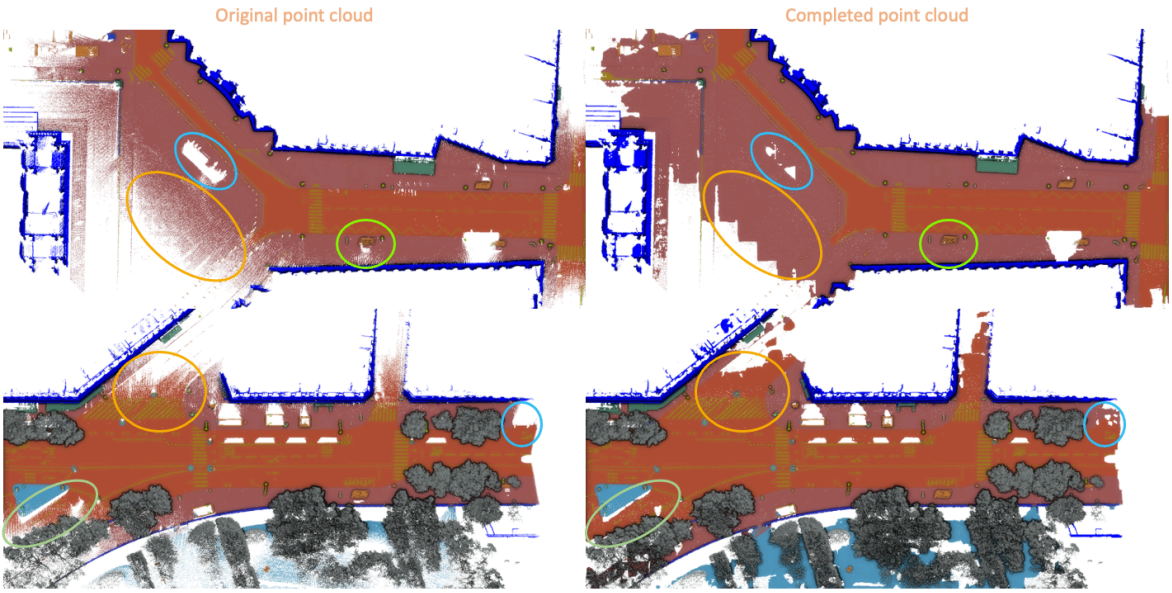


Figure 3.19: More results of scene completion performed on the target point cloud with point-wise semantic labels taken from the nearest neighbor.

We can still observe that the network is not able to completely fill the largest holes. For future works, we think that if we remove big blocks, taking into account the shape of the occlusions caused by the parked vehicles, for the network input and leave all of the points for the target, we will be able to predict larger missing regions, such as the occlusions caused by parked vehicles. Training the network on data from PC3D-CARLA, where we have more complete roads and finetuning on PC3D-Paris, we should obtain a better completion on the road and predict larger missing areas. Another approach would be training the network to complete large missing regions only on the road, where the geometry is mostly planar and train another model to complete more complex geometries separately, which would reduce the effect of planar geometry (roads) on complex geometry completion. However, we leave these proposals to future work.

3.4.8 Limitations and Perspectives

The main limitation of our UWED representation is that it does not allow the direct extraction of a mesh, unlike SDFs. The purpose of UWED is not to perform surface reconstruction, but to get an intermediate representation that gives better signal to SC networks. In perspective, our UWED function showed excellent performance as a scene representation and on the SC task: this representation of 3D geometry seems to allow neural networks to better learn geometry. Thus, it should be possible to use it for other tasks in 3D, such as registration, SS and object detection, but the state-of-the-art (SOTA) networks working on these tasks are designed to work on points clouds and not on SDFs, like SC networks. Using UWED for these tasks might require some architecture changes. We leave it for future work.

3.5 Conclusion

We presented an unsigned distance function, UWED, for 3D surface representation that is able to encode a better continuous representation of the underlying surface in a point cloud. UWED improves the quality of the completion task with respect to previously proposed SDF functions, while being able to better represent fine structures in the scene, which is verified quantitatively and qualitatively. Moreover, we introduced a point cloud extraction algorithm

from a UDF computed on a sparse 3D regular grid that recovers the finest details. Finally, we compared six different SDFs and UDFs on three different datasets, namely, Matterport3D, PC3D-CARLA, and PC3D-Paris. Our experiments showed quantitatively and qualitatively that our UWED implicit surface representation achieves the best completion results among all other SDFs and UDFs.

The next step in our reality replication pipeline is to generate a hole-free approximation of the underlying surface in a point cloud using a 3D modeling method. This requires a point cloud as input, which can be either the original cloud, or the completed cloud representing the static background only. After introducing our 3D modeling method in chapter 4, we use the completed point cloud (on target) to improve the final surface representation.

This work will result in a publication that is under submission in a conference, while the pre-print version is already on arxiv: https://arxiv.org/abs/2203.09167

- Richa, J.P., Deschaud, J.E., Goulette, F. and Dalmasso, N., 2022. UWED: Unsigned Distance Field for Accurate 3D Scene Representation and Completion. arXiv preprint arXiv:2203.09167.

Chapter 4

Scene Modeling

Contents

4.1 Introduction **64**
4.2 Related Work **67**
4.2.1 Surface Reconstruction 67
4.2.2 Point-based Surface Modeling 68
4.2.3 Neural Radiance Fields 71
4.2.4 Resampling 71
4.3 Adaptive Splatting **72**
4.3.1 Basic-Splatting 72
4.3.2 Adaptive Splatting 73
4.3.3 Splat-based Resampling and Denoising 73
4.4 Splat Ray Tracing **77**
4.4.1 Ray-splat Intersection 77
4.4.2 OptiX 77
4.4.3 Blending and Shading 78
4.5 Experiments and Results **79**
4.5.1 Experiments 79
4.5.1.1 Surface Representation 79
4.5.1.2 Datasets 79
4.5.2 Results 80
4.5.2.1 Paris-Carla-3D 81
4.5.2.2 SemanticKITTI 83
4.5.2.3 M-City 84
4.5.2.4 Blending 85
4.5.2.5 Modeling the Completed Scene 87
4.6 Conclusion **90**

Abstract

The manual construction of a virtual environment in which autonomous vehicles (AVs) can be simulated is a costly process and the results are usually far from realistic, due to the simplified geometric models used by 3D artists to reduce the creation complexity. Moreover, simulating the AV sensors in handcrafted environments and using the data to test AVs does not generalize well when deployed in the real world, due to the domain gap resulting from the over-simplified assumptions made in the handcrafted environments and sensor models. We propose to reduce this domain gap by automatically reconstructing the virtual environment from real-world data in the form of point clouds, which preserves the complexity of the real environment. There exist many methods on 3D modeling and surface reconstruction from point sets, however, most of these methods are not well suited for noisy point clouds and open surfaces acquired in outdoor environments. In this chapter, we introduce adaptive splatting to accurately model the static scene from real-world point clouds. Furthermore, we include a resampling and denoising method, that complements the scene completion method by reducing the sparsity and local anisotropy caused by the physical LiDAR sensor model and call our method AdaSplats. Finally, we detail our real-time ray tracing approach in the splatted model and achieve real-time rendering of the modeled scene as a first step toward real-time camera simulation.

Résumé

La construction manuelle d'un environnement virtuel dans lequel des véhicules autonomes (VA) peuvent être simulés est un processus coûteux et les résultats sont généralement loin d'être réalistes, en raison des modèles géométriques simplifiés utilisés par les artistes 3D pour réduire la complexité de la création. De plus, la simulation des capteurs dans des environnements artisanaux et l'utilisation des données pour tester les VA ne se généralisent pas bien lorsqu'ils sont déployés dans le monde réel, en raison de l'écart de domaine résultant des hypothèses trop simplifiées faites dans les environnements artisanaux et les modèles de capteurs. Nous proposons de réduire cet écart de domaine en reconstruisant automatiquement l'environnement virtuel à partir de données du monde réel sous forme de nuages de points, ce qui préserve la complexité de l'environnement réel. Il existe de nombreuses méthodes de modélisation 3D et de reconstruction de surface à partir d'ensembles de points, cependant, la plupart de ces méthodes ne sont pas bien adaptées aux nuages de points bruités et aux surfaces ouvertes acquises dans des environnements extérieurs. Dans ce chapitre, nous introduisons le "splatting" adaptatif pour modéliser avec précision la scène statique à partir de nuages de points réels. De plus, nous incluons une méthode de rééchantillonnage et de débruitage, qui complète la méthode de complétion de scène en réduisant l'éparsité et l'anisotropie locale causées par le modèle de capteur LiDAR physique et appelons notre méthode AdaSplats. Enfin, nous détaillons notre approche de tracé de rayons en temps réel dans le modèle splatté et réalisons un rendu en temps réel de la scène modélisée comme première étape vers la simulation de caméra en temps réel.

4.1 Introduction

In this chapter, we tackle the third step in our reality replication pipeline, scene modeling (see Figure 4.1), which aims to generate a hole-free approximation of the point cloud surface. The input to this module can be either the completed static point cloud, or the point cloud after dynamic objects removal. Moreover, we introduce a GPU-based ray-splat intersection method and achieve real-time rendering. Furthermore, we use the Scene Completion (SC)

output from the previous chapter and show how the SC task improves the rendering results in missing regions.

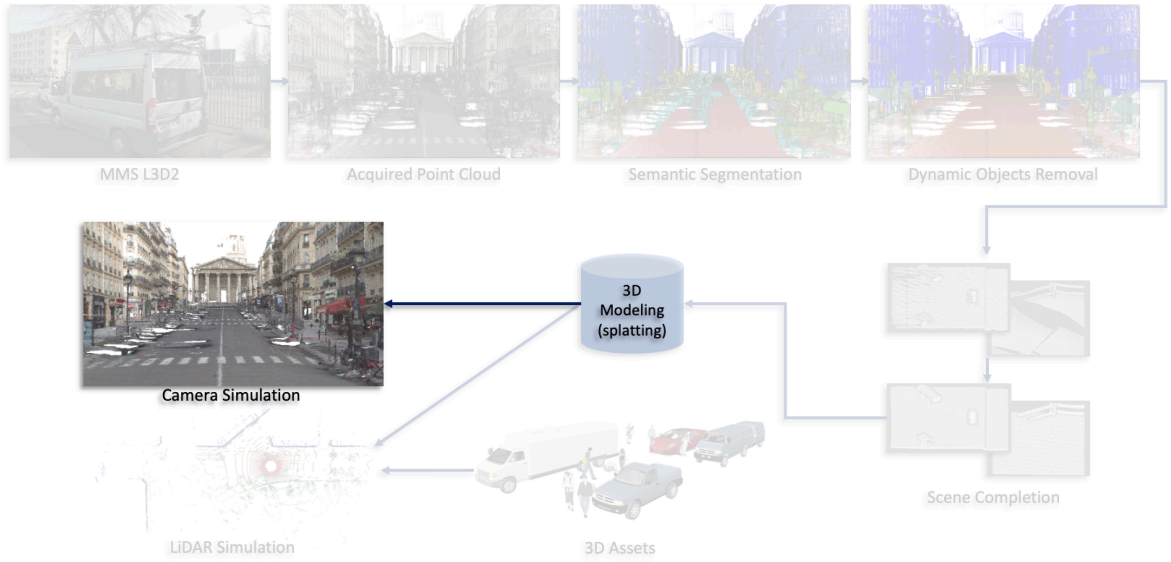


Figure 4.1: The third step in our automatic reality replication pipeline consists of generating a hole-free approximation of the surface from a point cloud and rendering the generated primitives as a first step toward camera simulation.

The most used sensors for AV tasks are 2D monocular RGB cameras and 3D Light Detection and Ranging (LiDAR). Cameras provide dense 2D information about the scene, while LiDARs output a sparse representation, but add the depth information, which gives an accurate sense of the distances to surrounding objects in the scene.

A strong requirement of deep learning techniques for vision and perception tasks is the need for very large datasets that are used to train neural networks. Although the sensors can be mounted on moving vehicles to collect datasets that can be used for the different tasks, they have a high cost in time and money to acquire the datasets. Moreover, collecting datasets from real-world scenes limits the training and testing domains to scenarios that happened at the time of acquisition. Scenarios generation can be done in closed and controlled environments, but it costs even more to prepare and maintain them. An alternative is to create realistic virtual environments and open the possibility to generate real-life scenarios, which is the aim of this thesis.

Constructing realistic virtual environments inside which AVs and the sensors used by AVs can be simulated is not an easy task. This can be achieved by 3D artists who carefully craft the scenes manually, such as in CARLA simulator [Dosovitskiy et al. 2017a]. However, manual construction takes months of labor time and other resources (e.g., costs 10K USD per kilometer) [Fang et al. 2020].

An alternative to manual scene construction is automatic surface reconstruction methods [Hoppe et al. 1992; Kolluri 2008; Kazhdan et al. 2006] that can be performed on point clouds acquired using LiDAR sensors. Surface reconstruction is a well studied area of research and many algorithms were proposed to reconstruct an explicit surface representation from unorganized point sets, in the form of triangular meshes, and primarily focused on closed surfaces of small shapes. Some of these methods discretize the point clouds, compute the distance from each voxel in the grid to the nearest neighbor, or a weighted average of the K-nearest neighbors in the point cloud and perform inside/outside classification of the surface using the normal information. These surface reconstruction methods need to perform inside/outside classification of closed surfaces to determine the sign of the distance function. Although this can be achieved using normal information, the surfaces acquired using LiDARs

are open surfaces. Normals orientation can be achieved using the sensor position, however, it stays inconsistent, due to the fact that neighboring points can be acquired from different sides of the same surface. Following these limitations, surface reconstruction methods fail to accurately represent the open surfaces, especially for thin structures. In addition, when using surface reconstruction methods on open surfaces, the border of the surface will be dilated because the function will attempt to close the surface, since it is performing inside/outside classification.

Point-based modeling and rendering [Levoy et al. 2000], more specifically splatting [Zwicker et al. 2001], achieves high-quality 3D modeling while reducing the number of generated geometric primitives. Some approaches use planar geometries or hybrid mesh splat surface representation [Devaux et al. 2016; Pagés et al. 2015], but they either do not focus on accurate geometry representation, or, in the case of the latter, use very expensive mesh generation techniques.

Accurate geometry representation of large outdoor point clouds is essential for high-quality rendering. Previous methods were designed to work on dense, small and carefully scanned data, which is a strong assumption to make on outdoor datasets acquired by LiDAR sensors, which are sparse and contain a large number of points, noise and outliers. Moreover, although previous splatting methods reduce the amount of primitives used to model the scene, they still produce a large number of primitives on such datasets. In this work, we compensate for the data sparsity and local geometry perturbations, by using local geometric and point-wise semantic labels, which we obtain from the annotated datasets, or using deep learning methods for Semantic Segmentation (SS).

SS on 3D point clouds is the task of assigning a point-wise semantic label. There exist a plethora of methods for the SS task [Thomas et al. 2019; Landrieu et al. 2018; Boulch et al. 2020; Choy et al. 2019]. We make use of Kernel Point Convolution (KPConv) [Thomas et al. 2019], which achieves an overall accuracy of 95% and an mIoU of 82% on the Paris-Lille-3D dataset [Roynard et al. 2018b] (outdoor LiDAR dataset from an MMS) on the SS task. We leverage the accuracy of KPConv, perform SS on the datasets and adapt the size of our generated splats using the semantic labels.

LiDAR point clouds from MMS are highly anisotropic [Roynard et al. 2018b]. Resampling the point cloud reduces the anisotropy and increases uniformity by redistributing the points. Previous upsampling methods [Alexa et al. 2003; Chen et al. 2018] address this problem, but they require passing through computationally expensive representations. Deep learning methods for point cloud upsampling [Yu et al. 2018; Zhou et al. 2021] are used to increase the uniformity of the points' distribution, and deep learning for depth completion [Chen et al. 2019d; Xu et al. 2019] can also help in the case of sparse point clouds. However, these methods are data hungry and usually limited to small objects or scenes.

Physics-based sensor simulation, such as the simulation of the camera and LiDAR sensors requires ray tracing the generated primitives. Previous splats ray-tracing algorithms [Linsen et al. 2008] are limited to ray-splat intersection using a CPU parallelized implementation. However, accelerating sensor simulation requires achieving real-time ray-tracing, which is not possible to achieve with CPU parallelization. We introduce a ray-splat intersection method leveraging the GPU architecture and an acceleration data structure, achieving real-time rendering in the splats model.

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.
- Fast GPU-based splats ray casting for real-time rendering.

4.2 Related Work

We introduce surface reconstruction methods, which were the first natural choice to automate the process of meshing point clouds obtained from laser scanners, state their limitations, and then we talk about point-based modeling and resampling techniques that allow us to achieve a more realistic surface representation with a lower number of geometric primitives.

4.2.1 Surface Reconstruction

The earliest methods on surface representation from point clouds mainly focused on surface reconstruction [Hoppe et al. 1992] producing triangular meshes. These methods were designed to work on closed surfaces and required a good point-wise normals estimation with no errors in orientation, which is a difficult problem and an active area of research [Mitra et al. 2003; Dey et al. 2005; Boulch et al. 2012; Zhao et al. 2019]. Using distance functions (DFs) on 3D volumetric grids for surface reconstruction traces back to the seminal work of Hoppe *et al.* [Hoppe et al. 1992], in which a Signed Distance Field $\phi : \mathbb{R}^3 \rightarrow \mathbb{R}$ was used to represent the underlying surface from a point cloud:

$$\phi(x) = \mathbf{n}_i \cdot (x - \mathbf{p}_i) \tag{4.1}$$

where x is the voxel coordinates in the 3D grid, $\mathbf{p}_i$ is the nearest neighbor of $\mathbf{x}$ in the point cloud $\mathcal{P}$, and $\mathbf{n}_i$ is the normal unit vector associated with the point $\mathbf{p}_i$.

Having computed the signed distance function (SDF) on a regular 3D grid, the Marching Cubes algorithm [Lorensen et al. 1987] extracts the final iso-surface as a mesh.

Following this pioneering work, several methods were proposed to deal with noisy data, such as Implicit Moving Least Squares (IMLS) [Kolluri 2008], which approximates the local neighborhood of a given voxel in the grid as a weighted average of the local point functions:

$$\text{IMLS}(x) = \frac{\sum_{\mathbf{p}_k \in N_x} \mathbf{n}_k \cdot (x - \mathbf{p}_k)\, \theta_k(x)}{\sum_{\mathbf{p}_k \in N_x} \theta_k(x)} \tag{4.2}$$

where x is the voxel coordinates, N_x is the set of $\mathbf{p}_k$ neighboring points from x in the point cloud $\mathcal{P}$, $\mathbf{n}_k$ is the normal unit vector associated with the point $\mathbf{p}_k$, and θ_k is the Gaussian weight defined as:

$$\theta_k(x) = e^{-||x - \mathbf{p}_k||_2^2 / \sigma^2} \tag{4.3}$$

σ is a parameter of the influence of points in N_x.

In a different approach, the volumetric fusion method of VRIP [Curless et al. 1996] takes advantage of range images by meshing the depth image to cast a ray from the sensor origin to the voxel of the volumetric grid, obtaining a signed distance to the mesh, and then merging the scans distances in a least-squares sense. However, this DF can only be computed from range images and cannot be used directly on point clouds.

Other surface reconstruction methods [Kazhdan et al. 2006; Kazhdan et al. 2013] use global implicit functions such as indicator functions where the reconstruction problem is solved using a Poisson system equation. Some use 3D Delaunay tetrahedralization [Labatut et al. 2009; Zhou et al. 2019]. Volumetric segmentation is a sub-category of indicator function that classifies whether a voxel is occupied or empty with a confidence level using octrees [Caraffa et al. 2015], or delaunay triangulation [Caraffa et al. 2017], which can also be scaled to arbitrarily large point clouds through distributing the surface reconstruction problem [Caraffa et al. 2021]. Some works on surface reconstruction focus on identifying drive-able zones, for robot navigation, through the creation of simplistic 3D models of roads and building [Soheilian et al. 2013], others propose improvements for a detailed facade reconstruction [Demantke et al. 2013]. Moreover, some applications have real-time

constraints [Boussaha et al. 2018], however, they don't create watertight meshes and introduce many disconnected parts and holes. More recently, 3DConvNets [Peng et al. 2020; Ummenhofer et al. 2021] are applied for surface reconstruction, see also section 3.2.2 introducing Implicit Function Learning (IFL) methods. These methods have the same limitation as classical surface reconstruction methods, which is surface dilation, while some of them generate lower quality geometries.

4.2.2 Point-based Surface Modeling

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.

Accurate surface modeling from point clouds was and still is an active area of research. Most of the previous approaches, which will be detailed in the following sub sections, either start from meshes and sample them into point clouds, or directly acquire the point clouds from 3D laser scanners. Using the sampled points, they represent the geometry with point primitives giving them basic geometrical shapes, mainly circles or ellipses, in a final step. Although these approaches work well on point clouds sampled from synthetic shapes, or collected with care from different viewpoints around a physical object using fixed 3D scanners, they perform very poorly when used on noisy point clouds collected from a mobile mapping system (MMS).

Point-based rendering gained interest after the report published by Levoy and Whitted on using points as display primitives [Levoy et al. 2000]. Using points without any processing does not provide useful information about the underlying surface, hence not enough information are available to model the surface and generate a realistic scene or object rendering, so they are not very useful without any processing. A better representation of the surface can be obtained by giving the points a side length, turning them into small squares in the 3D space, which results in a representation that can be used in more complex post processing such as ray tracing. The first methods on rendering such primitives without any connectivity information [Rusinkiewicz et al. 2000; Pfister et al. 2000] focused on achieving a low rendering time and interactively displaying large amounts of points. They used accelerating hierarchical data structures to accelerate the rendering of the generated splats, which are oriented 2D disks expanding to generate a hole-free approximation of the surface. They used splats as their modeling and rendering primitives.

Using splats as rendering primitives became the best choice for surface modeling after proving their efficiency and effectiveness in many methods. Surface Splatting [Zwicker et al. 2001] introduce a point rendering and texture filtering technique and achieve high quality anisotropic anti-aliasing. They combine oriented 2D reconstruction kernels, circular or elliptical splats, with a band-limiting image-space Elliptical Weighted Average (EWA) texture filter, which basically limits the bandwidth of the texture function frequency in a pre-filtering step to prevent high frequencies to be treated as low frequencies, then samples the continuous function to be processed and reconstructed in the final step (see figure 4.2). Other methods in this area [Botsch et al. 2003; Botsch et al. 2005] exploit the programmability of modern GPUs and detail the best practices for rendering point-based methods using elliptical splats, achieving a high frame rate even in the presence of a high number of primitives (see Figures 4.3 a and b, respectively), which makes it possible for real-time applications containing more details than similar scenes based on polygonal meshes. Surfels [Pfister et al. 2000] is another approach that focuses on accurate mapping of textures to splats to increase the visual details of the rendered objects, while rendering high numbers of primitives at interactive rates.

Achieving a high frame rate in the presence of a high number of primitives is essential, but it is also important to reduce the pre-processing computational complexity as much as

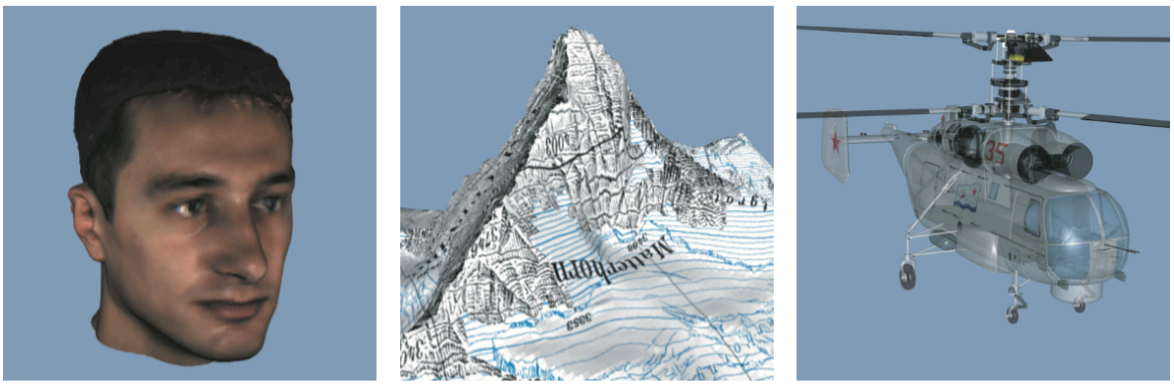
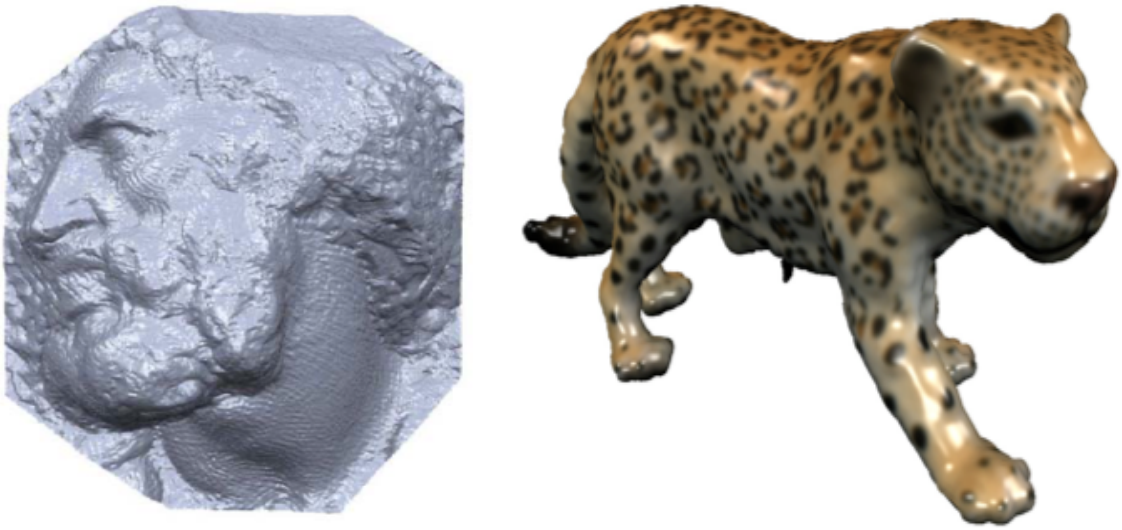


Figure 4.2: Surface splatting of a scan of a human face, textured terrain, and a complex point-sampled object with semi-transparent surfaces. Source [Zwicker et al. 2001]



a b

Figure 4.3: Left and right: Textured Splats rendering of point cloud. Source [Botsch et al. 2003] and [Botsch et al. 2005].

possible. One approach using circular or elliptical splats [Wu et al. 2004] creates a hole-free approximation of the surface, then performs a relaxation procedure that results in a minimal set of splats that best estimates the surface. Being able to obtain a hole-free approximation of a point cloud surface while achieving real-time rendering is essential for interactive applications, but the aforementioned approaches don't focus on generating photo-realistic rendering, because they use one normal vector per point, which results in rendering that is comparable to Gouraud or flat shading. In order to generate photo-realistic rendering, focus was shifted to solve the problem of disconnection between splats, which makes it hard to interpolate normals between primitives. In order to overcome this challenge, splats are associated with a normal field that is created using the normals of points in the neighborhood of the splat center [Botsch et al. 2004]. Although this approach provides a better local approximation of the surface normals and results in rendering comparable to Phong shading for regular meshes, hence the name Phong splatting, it is computationally expensive to generate a per-splat normal field and increases the host (CPU) to device (GPU) communication time and memory footprint resulting from transferring more information to the device.

A recent approach on point clouds rasterization provides new data structure implementation to achieve real-time rendering on large point clouds [Kang et al. 2019]. The method

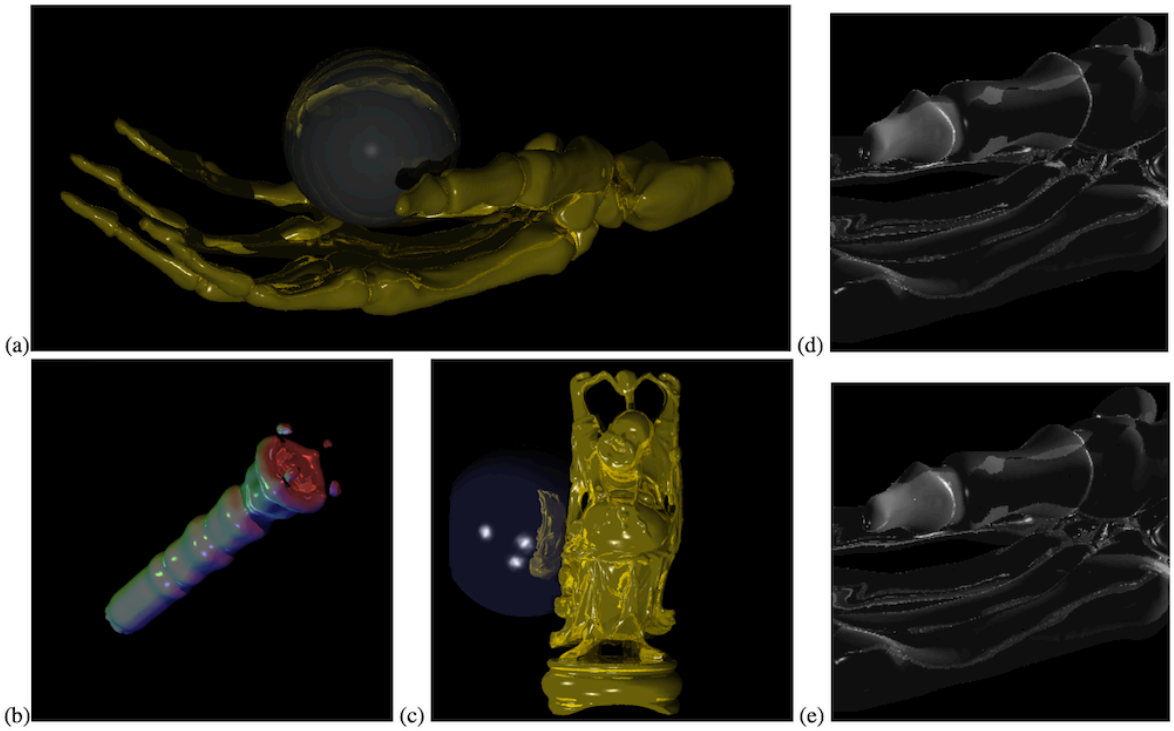


Figure 4.4: Splat-based ray tracing from method [Linsen et al. 2008]. Showing the results on the Skeleton Hand, Fuel, and Happy Buddha datasets. Source [Linsen et al. 2008]

first divides the point cloud into nested blocks, then the data in each block are organized in an octree. Finally all of the Octrees are merged into one big hierarchy that can be efficiently used for accelerating the rendering with a level-of-detail (LOD) structure. To overcome the limitations on the hardware imposed by graphics APIs, some methods propose GPGPU approaches to increase the GPU rendering ability [Günther et al. 2013] by using compute shaders to render point primitives, which can be used to reduce the computation complexity imposed by the standard rendering pipeline that is usually optimized to render triangular meshes. A more recent approach [Schütz et al. 2021] achieves a real-time rendering of 60 FPS on around 800 million points on an RTX 3090 without a LOD structure using a compute shader implementation of point rendering, which draws 1 point per pixel, instead of standard GL_POINTS which are defined as quads instances, opening the possibility for massive point clouds visualization in real time. Extending this last work, the authors add support for LOD and frustum culling to achieve real-time rendering on 2 billion points [Schütz et al. 2022]

The previous approaches focused on several aspects of using the points as rendering primitives, but one important aspect that is still missing is ray tracing the generated splats. In [Linsen et al. 2008], the authors leverage the previous methods, while modifying the pipeline to best suit their ray tracing algorithm. They generate a hole-free approximation of the surface using a globally defined percentage (*perc*) operator. The operator can be used to check whether the current splat lies within a percentage of the radii of neighboring splats, based on which the decision of whether a new splat should be generated is taken. Moreover, they associate each splat with a normal field and then ray trace the generated splats, performing per-pixel Phong shading achieving photo-realistic rendering on dense and uniformly distributed point clouds (see Figure 4.4), while 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.

4.2.3 Neural Radiance Fields

A more recent method, such as Neural Radiance Fields (NeRF) [Mildenhall et al. 2020] gained a lot of interest in the rendering and novel view synthesis communities as neural networks are trained to generate photo-realistic novel views of a scene from a set of calibrated input images. Several subsequent methods [Garbin et al. 2021; Hedman et al. 2021; Liu et al. 2020; Rebain et al. 2021; Reiser et al. 2021] introduced different approaches to reduce the computational complexity and providing real-time inference ability. An extension to NeRF predicts Spherical Harmonics (SH) coefficients NeRF-SH [Yu et al. 2021a] instead of RGB colors. At inference time, the SH coefficients are stored in sparse Octree voxels and used to compute the colors by querying the SH function depending on the viewing angle, which overcomes passing the viewing direction input to the network and allows real-time inference. A more recent method [Fridovich-Keil et al. 2022] extends the latter by using a sparse grid representation and optimizing SH of order 2, resulting in 27 coefficients for the 3 color channels. Moreover, they perform trilinear interpolation instead of nearest neighbor only, achieving an optimization time that is two orders of magnitude faster, without passing through a neural network. Extending the work on NeRF, realistic scene re-illumination can also be achieved with novel views [Srinivasan et al. 2021]. These methods achieve impressive rendering quality of complex scenes. However, we focus our work on high-fidelity LiDAR simulation and leave photo-realistic camera simulation for future works.

4.2.4 Resampling

Most of previous approaches didn't tackle the problem of data sparsity and nonuniformity, which is highly needed in the case where raw point clouds collected using a MMS are being used as shown in [Roynard et al. 2018b]. When the density of a point cloud is very high, the set of points in a given neighborhood is better handled when down-sampled, because having excess of points would not give any additional information about the local surface, so down-sampling techniques are used to minimize the number of points used in the surface reconstruction or modeling process, which consequently minimizes the rendering cost. Moreover, point clouds collected with a MMS is highly anisotropic, due to the physical model of the LiDAR sensor, resulting in dense regions along the sweep lines and sparser regions elsewhere. Isotropic resampling redistributes the points to increase the uniformity of the points' distribution across the point cloud, which facilitates the splats generation and improves the normals estimation.

A previous upsampling approach [Alexa et al. 2003] 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 neighoborhood 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.

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 [Maglo et al. 2015], 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.

Other methods use deep learning techniques directly on point clouds to achieve higher density on the sparse point clouds through upsampling [Yu et al. 2018; Zhou et al. 2021]. 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 [Chen et al. 2019d; Xu

et al. 2019]. However, current deep learning methods are limited to small scenes and suffer from a performance drop with unseen real-world data.

Inspired by [Alexa et al. 2003], we introduce a novel splat-based point cloud resampling approach, which increases the uniformity of points distribution. Moreover, we embed a denoising and an outliers rejection step in the sampling algorithm that helps achieving a minimal set of points that would be later used to generate the splats. With the resampled point cloud, we achieve high-quality, hole-free surface modeling using our adaptive splats approach.

4.3 Adaptive Splatting

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 rendering, which can be used for camera simulation based on GPU ray casting. In our approach, we leverage the accuracy of the state-of-the-art deep learning method KP-Conv [Thomas et al. 2019] in SS 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.

4.3.1 Basic-Splatting

We describe here a variant of the splatting method of Linsen *et al.* [Linsen et al. 2008], the basic algorithm on which we will develop our adaptive method. Let a vector $\mathbf{x}$ with hat $\hat{\mathbf{x}}$ 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 $K = 40$. 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 $N_{\mathbf{p}_i}$ to obtain the normal $\hat{\mathbf{n}}_i$ at each point $\mathbf{p}_i$ and reorient it with respect to the LiDAR sensor position.

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

$$\epsilon_i^k = \hat{\mathbf{n}}_i \cdot (\mathbf{p}_i^k - \mathbf{p}_i) \tag{4.4}$$

We stop the growing in $N_{\mathbf{p}_i}$ when $|\epsilon_i^k|$ 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:

$$\mathbf{c}_{S_i} = \mathbf{p}_i + \bar{\epsilon_i}\, \hat{\mathbf{n}}_\mathbf{i} \tag{4.5}$$

with $\bar{\epsilon_i}$ 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 $\mathbf{p}_i^{k_{last}}$ from $\mathbf{c}_{S_i}$:

$$r_{S_i} = ||(\mathbf{p}_i^{k_{last}} - \mathbf{c}_{S_i}) - \hat{\mathbf{n}}_i \cdot (\mathbf{p}_i^{k_{last}} - \mathbf{c}_{S_i})\, \hat{\mathbf{n}}_i||_2 \tag{4.6}$$

Then, all points in the neighborhood $N_{\mathbf{p}_i}$ inside the sphere of radius $\alpha\, r_{S_i}$ are discarded from the splat generation. α 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 $\bar{E}$ as the average unsigned point-to-plane distance of points in all $N_{\mathbf{p}_i}$. 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.

4.3.2 Adaptive Splatting

As shown in Figure 4.1, we first perform a SS of the raw point cloud using deep learning [Thomas et al. 2019] 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 $\mathbf{p}_i$, we change the neighborhood $N_{\mathbf{p}_i}$ with parameters K and $\bar{R}$ (as a reminder, $N_{\mathbf{p}_i}$ 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: $3K = 120$, $3\bar{R}$, $3\bar{E}$
- Surface: $K = 40$, $\bar{R}$, $\bar{E}$ (no change compared to basic splat)
- Linear: $0.33K = 13$, $0.33\bar{R}$, $0.33\bar{E}$
- Non-surface: $0.25K = 10$, $0.25\bar{R}$, $0.25\bar{E}$

We also stop growing splat S_i when a new point $\mathbf{p}_i^k \in N_{\mathbf{p}_i}$ has a semantic class different from the class of $\mathbf{p}_i$.

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 $\hat{\mathbf{n}}_i$ will include a neighboring point $\mathbf{p}_i^k$ with a normal $\hat{\mathbf{n}}_i^k$ only if it passes the smoothness check $\hat{\mathbf{n}}_i \cdot \hat{\mathbf{n}}_i^k > \beta$, (we took $\beta = 0.6$). Once a point fails to pass this check, we stop growing the splat.

4.3.3 Splat-based Resampling and Denoising

As demonstrated in the scene completion (SC) chapter, the SC task is able to accurately complete regions that are missing, or having high sparsity. However, we do not use the trained model to complete the geometry on regions with low sparsity, so we do not lose useful local geometric features, instead, we introduce a splat-based resampling approach that relies on AdaSplats to increase the uniformity of points distribution throughout the whole point clouds.

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 $\bar{\delta}$ 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 $\delta_i < \bar{\delta}$. We select the farthest splat S_j in the neighborhood of radius $\bar{R}$. We verify whether both splats belong to the same semantic class and if they pass the smoothness check $\hat{\mathbf{n}}_{S_i} \cdot \hat{\mathbf{n}}_{S_j} > \beta$. 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 to reorient the normals 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, see Figures 4.5 and 4.6. The smoothness and semantic class equality checks ensure that we do not smooth out sharp features.

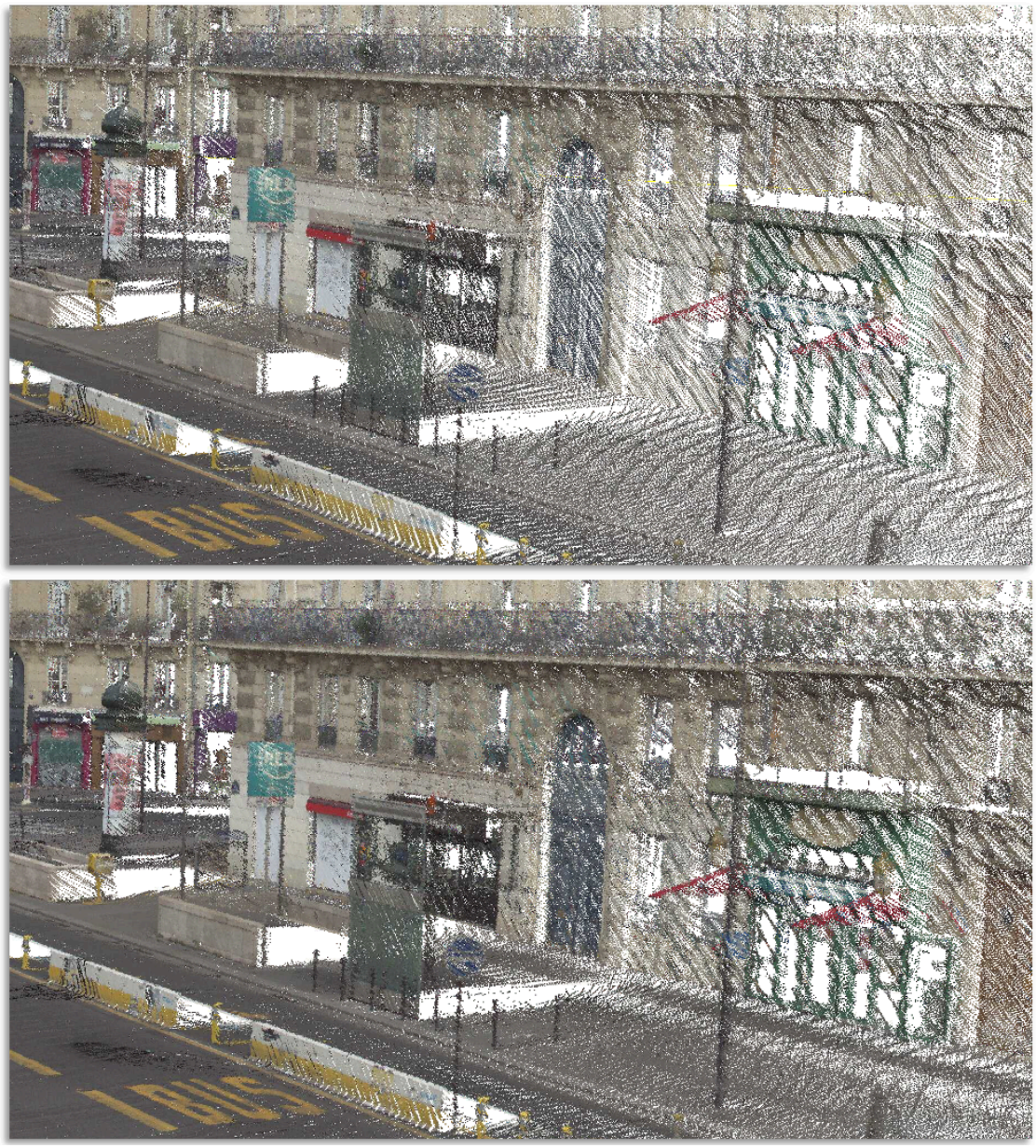


Figure 4.5: Qualitative evaluation of the resampling algorithm. We show the original and resampled point clouds at the top and bottom, respectively. Our resampling algorithm is able to re-distribute the point density by reducing the number of points in dense regions and increasing the number of points in sparse regions, while preserving the underlying surface structure.

Figure 4.6: Another qualitative evaluation of the resampling algorithm. We show the original and resampled point clouds at the top and bottom, respectively.

4.4 Splat Ray Tracing

Simulating the sensors used by AVs, such as cameras and LiDARs, requires the implementation of an efficient ray-splat intersection algorithm. Very few approaches worked on ray tracing splats and achieving a high rendering quality [Linsen et al. 2008], however, their method was implemented to work on CPUs only, which makes it impossible to achieve real-time rendering. In this section, we introduce a ray-splats intersection method that leverages OptiX [Parker et al. 2010] to accelerate the process, achieving faster than real-time sensor simulation. OptiX is a ray-tracing engine introduced by NVIDIA that makes use of the GPU architecture to parallelize the ray-casting process and implements a Bounding Volume Hierarchy (BVH) accelerating structure, to accelerate the ray-primitive intersection. Using our ray-splat intersection method, we can easily adapt our pipeline to include the simulation of other sensors, such as the RADAR.

4.4.1 Ray-splat Intersection

A splat is defined by its center $\mathbf{c}_{S_i}$, a normal vector $\hat{\mathbf{n}}_{S_i}$ and a radius r_{S_i}, lies in a plane defined by a point $\mathbf{p}_i$ and a normal vector $\hat{\mathbf{n}}_i$. To intersect the splat, we first need to intersect the plane in which the splat lies. For this, we define the plane using $\mathbf{c}_{S_i}$ and $\hat{\mathbf{n}}_{S_i}$. A ray is defined by its origin r_o and a unit direction vector $\hat{\mathbf{r}}_{dir}$. The intersection point $\mathbf{p}_{int}$ along a ray can be found at position $t \in \mathbb{R}^+$ along $\hat{\mathbf{r}}_{dir}$:

$$\mathbf{p}_{int} = \mathbf{r}_0 + \hat{\mathbf{r}}_{dir} * t \tag{4.7}$$

A vector $\mathbf{v}_{i_k}$ can be computed from any point $\mathbf{p}_{i_k}$ lying on the plane with $\mathbf{v}_{i_k} = \mathbf{p}_{i_k} - \mathbf{c}_{S_i}$. This vector lies in the plane, so it is orthogonal to the normal vector and this can be checked by taking the dot product $\mathbf{v}_{i_k} \cdot \hat{\mathbf{n}}_{S_i} = 0$. We cast a ray from the camera origin and compute $\mathbf{p}_{int}$ at a given t then report an intersection using the following check:

$$intersected = \begin{cases} 1 & if \quad (\mathbf{p}_{int} - \mathbf{c}_{S_i}) \cdot \hat{\mathbf{n}}_{S_i} = 0 \\ 0 & otherwise \end{cases} \tag{4.8}$$

solving for t

$$t * \hat{\mathbf{r}}_{dir} \cdot \hat{\mathbf{n}}_{S_i} + (\mathbf{r}_o - \mathbf{c}_{S_i}) \cdot \hat{\mathbf{n}}_{S_i} = 0 \tag{4.9}$$

with

$$t = \frac{(\mathbf{c}_{S_i} - \mathbf{r}_0) \cdot \hat{\mathbf{n}}_{S_i}}{\mathbf{r}_{dir} \cdot \hat{\mathbf{n}}_{S_i}} \tag{4.10}$$

If a ray-plane intersection was reported, we check if the point of intersection lies inside the radius of the splat

$$\mathbf{v}_{int} \cdot \mathbf{v}_{int} < r_{S_i}^2 \quad \text{with} : \mathbf{v}_{int} = \mathbf{p}_{int} - \mathbf{c}_{S_i} \tag{4.11}$$

4.4.2 OptiX

The intersection algorithm explained above is implemented in CUDA to cast rays in parallel. On top of the parallel ray casting, OptiX provides the BVH acceleration structure that is used to accelerate the ray-primitive intersection. More specifically, the BVH is created, on the host side (CPU) using the splats centers, where each splat is encapsulated in an Axis Aligned Bounding Box (AABB) used to build the acceleration structure, with a side length equal to the splat's diameter. The AABB serves as a first ray-primitive encounter, since when a ray traverses the BVH, it keeps traversing it until an AABB is intersected, which invokes

the custom intersection program (e.g., ray-splat intersection). Once the BVH is created, the ray generation program can be invoked on the device (GPU) side, which generates the rays and calls the BVH traversal program. When the ray-primitive intersection program reports the intersection, the closest hit program is invoked to perform the custom shading and return the final pixel color, otherwise, the miss program is invoked to return the background color.

4.4.3 Blending and Shading

In the splats generation step (see section 4.3), we carefully move the splat's center along the normal vector to minimize its maximal distance to the neighboring points (see equation 4.5). This results in a smooth approximate surface representation. However, due to local density changes, registration errors and other factors, we don't have a perfect normal estimation, which results in small perturbations in the normal vectors between neighboring splats that leads to a reduction in the shading quality. With the first ray-splat intersection, we obtain a good rendering of the modeled surface, but we obtain a low rendering quality resembling Gouraud shading. We extend our intersection implementation to blend the normals, by taking the weighted average of the normals of overlapping splats, resulting in a high quality Phong shading and a smoother surface rendering.

First, we define the depth ($\mathcal{D}$) of intersection, which is how many overlapping splats we want to intersect along the same ray. The higher the depth, the more computations are required, which ultimately affects the simulation frequency, so we are left with a trade-off between precision and time complexity. Having defined the depth, we cast the rays from the sensor origin, traverse the BVH and return the data from the intersected splat, such as the center, normal, radius and color. We offset the intersection point by an ϵ (we use 10^{-4} in our implementation), to prevent self intersections, and cast a new ray, then we save the intersection data. We repeat these steps until the maximum depth is reached, or all splats are intersected. If the number of overlapping splats is less than $\mathcal{D}$, we reset $\mathcal{D}$ to the maximum number of intersections, then we compute a weighted average of the normal taking into consideration the depth of the intersection and the distance from the intersected point on the splat to the center of each splat using a Gaussian kernel.

$$\bar{\hat{\mathbf{n}}} = \frac{\sum_{i=1}^{\mathcal{D}} (\theta_i + \beta_i)\hat{\mathbf{n}}_{S_i}}{\sum_{i=1}^{\mathcal{D}} \theta_i + \beta_i} \tag{4.12}$$

Where θ_i is a Gaussian kernel used to weight the contribution of the normal of each intersected splat as a function of the distance of the intersected point from the center of the intersected splat.

$$\theta_i = e^{-||\mathbf{c}_{S_i} - \mathbf{p}_{int}||_2^2/\sigma^2} \tag{4.13}$$

σ is chosen to be equal to the intersected splat radius.

β_i is also a Gaussian kernel used to weight the contribution of each splat to the final normal along the ray direction using the depth information.

$$\beta_i = e^{-|d_i - \frac{\mathcal{D}}{2}|/\frac{\mathcal{D}}{2}} \tag{4.14}$$

where d_i is the current ray-splat intersection depth.

The accumulation process is stopped whenever the maximum depth is reached, or all of the overlapping splats are intersected. However, this stopping condition does not account for splats lying on other surfaces, so the depth should be limited to locally overlapping splats, otherwise, we might include information about splats from other surfaces. Thus, the accumulation process is also stopped if the distance between two overlapping splats overcomes a pre-defined threshold. We choose the threshold based on analysis on the sensor error and registration error, which is +/- 2 cm, so we choose a threshold of 5 cm.

After this step, we use the corrected normal vector to perform the shading computation and return the final pixel color. We place multiple lights in the scene, implement a Phong shading algorithm and compute the contribution of each point light source to the final RGB pixel values.

4.5 Experiments and Results

In this section, we detail the experiments done on the different datasets and show qualitative results to compare the modeling capabilities of the used methods. Extensive evaluation was done on PC3D-Paris, since it is the only dataset that we use which was collected in mapping configuration and contains full building facades and a large set of objects with complex geometries. The SemanticKITTI and M-City datasets were used to validate that our method obtains better results than others, even when using other datasets collected from different sensors and in different configurations.

4.5.1 Experiments

To demonstrate the accuracy of our approach, we choose to compare it to 3 other methods, namely, Basic Splats (see section 4.3.1), IMLS [Kolluri 2008], and screened poisson [Kazhdan et al. 2013]. Moreover, to test the ability of our method to correctly model scenes collected using different sensors, we use 3 different datasets collected using different sensors, in different configurations and different cities.

Finding a good metric to quantitatively evaluate the accuracy of the geometric modeling of the proposed method is not straightforward on rendered images, therefore, we show only qualitative results in this chapter and provide quantitative results on the LiDAR simulation task in the next chapter.

4.5.1.1 Surface Representation

We use the latest available code (version 13.72) for Screened Poisson surface reconstruction to mesh the point cloud, obtaining the finest mesh with : octree depth of 13, Neumann boundary constraints, samples per node as 2 and other parameters as default values. We used also SurfaceTrimmer to remove parts of the reconstructed surface that are generated in low-sampling-density regions with trim value as 10.0 and other parameters as default. For IMLS, we use a voxel size of 7 cm, we fix sigma to two times the voxel size and perform a sparse grid search with a truncation of 3 voxels, where we fill the IMLS SDF values only in voxels near the surface and use the marching cubes algorithm [Lorensen et al. 1987] to extract the iso-surface.

4.5.1.2 Datasets

To test the performance and robustness of the different representations, we choose three different datasets: two acquired by mobile LiDARs (PC3D and SemanticKITTI) and one built from a Terrestrial Laser Scanner (M-City).

Paris-Carla-3D We use the PC3D-Paris part from the PC3D dataset introduced in chapter 2. The high diversity of complex objects (barriers, street lamps, traffic lights, vegetation, facades with balconies) present in this dataset, allows us to compare the modeling capacities of the different techniques in real situations.

First, we remove dynamic objects using the semantic information and apply the different surface representations on the static background.

For the comparison between the different surface representation techniques, we use only two parts of PC3D-Paris, namely, Soufflot-1 and Soufflot-2 containing 12.3 million points.

Comparisons were done to determine the ability of the different methods to represent the geometry. After the comparison, we demonstrate that our method can be used with point-wise color information, resulting in a high visual quality. Moreover, we provide qualitative evaluation of the different k-nn to validate the choice of the parameters used for the different semantic groups. Finally, we use the full PC3D-Paris dataset, containing 60 million points, to show the full modeled scene and include point lights in the modeled scene at the same positions of the street lamps in the real world, limit the light distance to 25 meters and attenuate the light with respect to the distance between the light source and the intersected splat.

SemanticKITTI Validating the accuracy of our modeling method requires testing it on other types of data. More specifically, we use SemanticKITTI [Behley et al. 2019] dataset, which was acquired using a different sensor mounted in a different configuration, namely the Velodyne HDL-64E in AV mode. The different sensor and mount result in a different scan pattern, which helps in validating if our method generalizes to other types of sensors, or data. Moreover, the geometry in SemanticKITTI is different from PC3D-Paris, since it was acquired in a different country and not in the heart of a large city. The difference in the configuration between both datasets (mapping and AV), results in a different scanning pattern, which gives us the possibility to test if our method is able to generalize to this kind of data.

Same as in PC3D-Paris, we first remove dynamic objects using the semantic information and apply the different surface representations on the static background.

For the comparison between the different surface representation techniques, we use the first 150 scans from sequence-00 accumulated with a LiDAR SLAM [Deschaud 2018], which results in a point cloud with 15.5 million points.

M-City We take the testing even further and model a point cloud that was acquired using a TLS (M-City), which has a completely different scanning pattern, since the point cloud is acquired at fixed points in a controlled environment with no dynamic objects.

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). Having said that, for the comparison between the different surface representations, we take the part containing the 17.5 million points and compare the Basic Splats and AdaSplats approaches to the manually reconstructed scene mesh.

4.5.2 Results

We show qualitative results on the three datasets and show the effect of blending on increasing the visual quality of the rendering. Moreover, we show how we increase the level of realism, especially on PC3D, when realistically illuminating the scene. Tables 4.1, 4.2, and 4.3 show the generation time (Gen T), number of generated primitives (Gen Prim) and the rendering frequency (Render Freq) with a resolution of 2560 x 1440. Moreover, we show that using deep learning methods for SS, such as KPConv [Thomas et al. 2019] (AdaSplats-KPConv) is able to obtain results very close to using the ground truth semantic information (AdaSplats-GT) on PC3D-Paris and SemanticKITTI.

4.5.2.1 Paris-Carla-3D

We start by validating the choice of parameters used for each semantic group. To do so, we generate the basic splats model on PC3D-Paris using different K-nn, with K = 10, 40 and 120 and compare the results to our AdaSplats method (see Figure 4.7). For a fair comparison with Basic Splats on the rendering and LiDAR simulation sides, we found that the best trade-off between geometric accuracy and hole-free approximation is to use K = 40 (which results in holes on the surface, especially the ground), since a larger K would result in very large splats and a smaller K results in many holes, which does not express well the local geometry. Figure 4.8 shows renderings of the different surface representation methods on PC3D-Paris dataset. The rendered splats are shown without blending. In these qualitative results we show the modeling capabilities of the different surface representations. We can see that we obtain the best results with AdaSplats, especially on fine structures, shown in colored squares, thanks to the adaptiveness of our method.

Figure 4.7: Rendering results on different choices of K-nn on PC3D-Paris dataset. A small K (e.g., 10 or 20) results in holes on the surface and ground groups, while resulting in a better approximation on the non-surface and linear groups. A large K (40 to 120) results in a hole-free approximation of the surface and ground semantic groups, while creating artefacts on small structures, belonging to the linear and surface groups.

We report the generation and rendering details of PC3D-Paris in Table 4.1. We obtain the lowest number of primitives and highest frequency with our AdaSplats method. We observe that Basic Splats has the lowest generation time and this is because there is no resampling, which results in generating the splatted environment only once. Moreover, the generation time of AdaSplats-KPConv includes the inference time of KPConv, which is 600 seconds for 10 million points. The generation time of IMLS is very high and we attribute this to the fact that our implementation can be improved. However, it would still result in a generation time higher than Screened Poisson.

Figure 4.8: Rendering the different surface representations on PC3D-Paris. The first row shows the meshed scene using Poisson (left) and IMLS (right). The second shows the splatted scene using basic splats (left) and AdaSplats using KPConv semantics (right). The splatted scene using AdaSplats-GT, which contains the ground truth point-wise semantic information (left) and the original point cloud (right).

Model	**Gen T (in s)**	**Gen Prim (#)**	**Render Freq (in Hz)**
Mesh - Poisson	797	5.20M	1000Hz
Mesh - IMLS	3216	6.32M	920Hz
Basic Splats	200	5.40M	100Hz
AdaSplats-KPConv	1064	1.75M	240Hz
AdaSplats-GT	451	1.72M	250Hz

Table 4.1: Results 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) and rendering frequency (Render Freq) in Hz.

4.5.2.2 SemanticKITTI

Figure 4.9 shows renderings of the different surface representation methods on SemanticKITTI. Looking at the results, we can see that using a point cloud acquired using a different sensor (Velodyne HDL-64E), mounted in a different configuration (AV mode) does not change, or reduce the quality of our AdaSplats method. We can observe that we are able to obtain the best quality with AdaSplats, most noticeable on the traffic signs. We also observe that we obtain the smoothest planar surface (the road), even when there are registration errors, such as in the sequence that we demonstrate in the figure. Having said that, we can compensate to small registration errors, thanks to the resampling algorithm and our adaptive method, that takes a larger neighborhood into consideration for the generation of the splats on the ground.

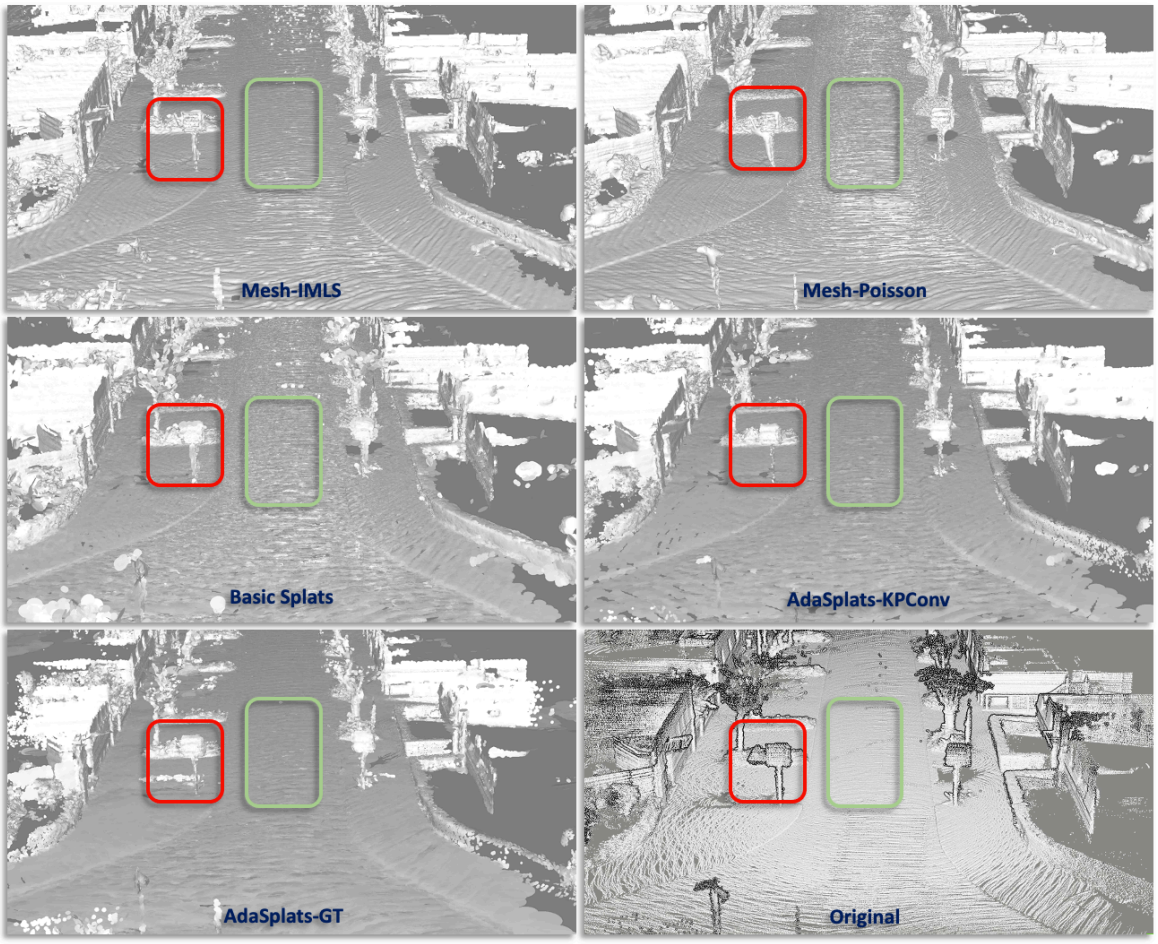


Figure 4.9: Rendering the different surface representations on SemanticKITTI. The first row shows the meshed scene using Poisson (left) and IMLS (right). The second shows the splatted scene using basic splats (left) and AdaSplats using KPConv semantics (right). The splatted scene using AdaSplats-GT, which contains the ground truth point-wise semantic information (left) and the original point cloud (right).

We report the generation and rendering details of SemanticKITTI in Table 4.2. We obtain the lowest number of primitives with our AdaSplats method. The generation time of Basic Splats is the lowest and IMLS is the highest as we explain above. Moreover, the generation time of AdaSplats-KPConv includes the inference time of KPConv, which is 10 minutes for the first 150 frames of sequence 00.

Model	Gen T (in s)	Gen Prim (#)	Render Freq (in Hz)
Mesh - Poisson	796	5.97M	1050Hz
Mesh - IMLS	1380	7.05M	1020Hz
Basic Splats	185	7.77M	170Hz
AdaSplats-KPConv	1166	6.11M	220Hz
AdaSplats-GT	544	4.56M	240Hz

Table 4.2: Results 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) and rendering frequency (Render Freq) in Hz.

4.5.2.3 M-City

Figure 4.10 shows renderings of the different surface representation methods on M-City. For M-City, we did not perform Poisson and IMLS surface reconstruction, since we do not have the position of the scanners to orient the normals. However, we have the manually reconstructed mesh, which we use to compare a manual reconstruction of the scene to Basic Splats and our AdaSplats method. Moreover, we cannot train KPConv on this small dataset, so we do not include AdaSplats-KPConv in the comparisons. Looking at the results, we observe that our method obtains a better surface representation, which can be clearly seen on the grass and vegetation that are hard to manually reconstruct due to the complexity of the geometry. When manually reconstructing complex geometry, 3D artists need to simplify the local geometry.

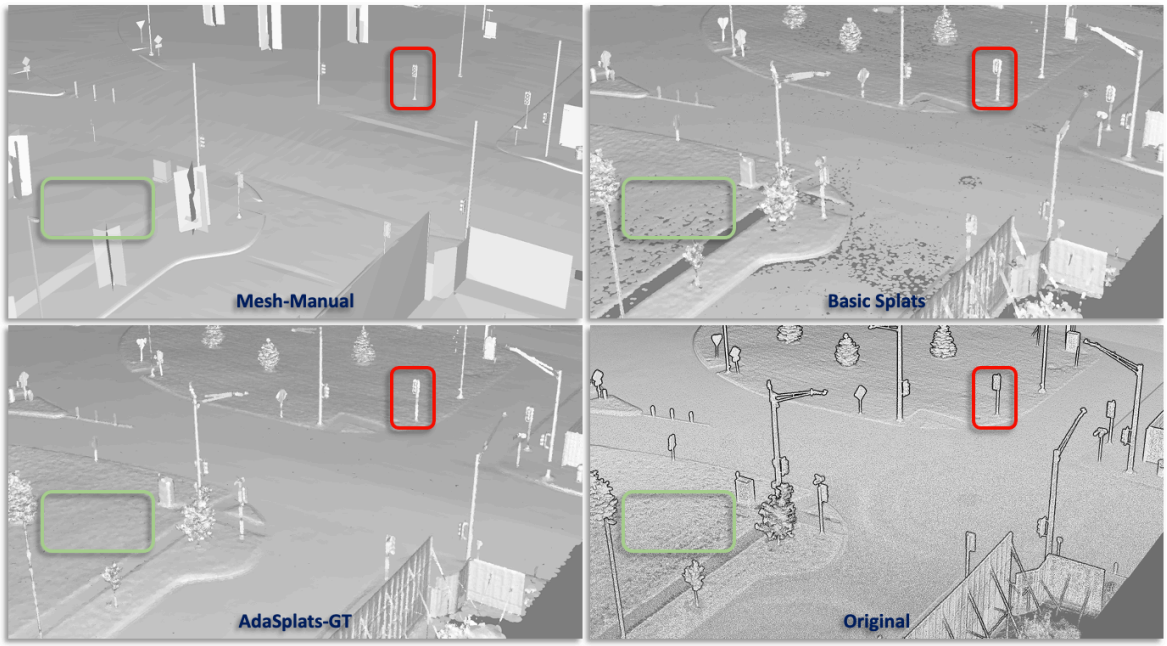


Figure 4.10: Rendering the different surface representations on M-City. The first row shows the manually meshed scene (left) and basic splats (right). The second shows the results of rendering AdaSplats using GT semantics (left) and the original point cloud (right).

We report the generation and rendering details of M-City in Table 4.3. Same as PC3D-Paris and SemanticKITTI, we are able to obtain the lowest number of primitives with our

AdaSplats method. Moreover, our automatic pipeline drastically reduces the generation time (more than 1 month for manual reconstruction, against 8.5 minutes for AdaSplats), while obtaining a higher rendering quality.

Model	**Gen T (in s)**	**Gen Prim (#)**	**Render Freq (in Hz)**
Mesh - Manual	1 month	71.5K	1930Hz
Basic Splats	199	5.82M	140Hz
AdaSplats-GT	513	3.01M	440Hz

Table 4.3: Results 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) and rendering frequency (Render Freq) in Hz.

4.5.2.4 Blending

To show the full capacity of our method, we include our colors and normals blending implementation and use the point-wise RGB data to render the colored PC3D-Paris splat model. Figure 4.11 compares rendering with and without blending colors and normals. Even with only point-wise color information, we are able to increase the realism of the rendered scene with correct blending. Moreover, texture mapping can be implemented to increase the rendering quality even further, but we leave it to future works.

Figure 4.11: Results on rendering the splatted environment with splat-wise color information. We show a rendering of the splatted Soufflot-0 and Soufflot-1 from PC3D-Paris dataset. To the left, we show the rendering without blending. To the right we show the rendering results with blending.

To replicate real-world environments, more complexity can be added to the scene, such as adding street lights during night time. Figure 4.12 shows a rendering of the full PC3D-Paris dataset. We add 43 light sources in the scene at the positions of the street lamp lights in the original point cloud. We achieve a high quality Phong shading, using the weighted

average of normals of overlapping splats, while dynamically illuminating the scene. For the blending, we use an intersection depth of 5 and even with the high intersection depth, we achieve a rendering frequency of 20 Hz in street view, where we have the highest number of ray primitive intersection.

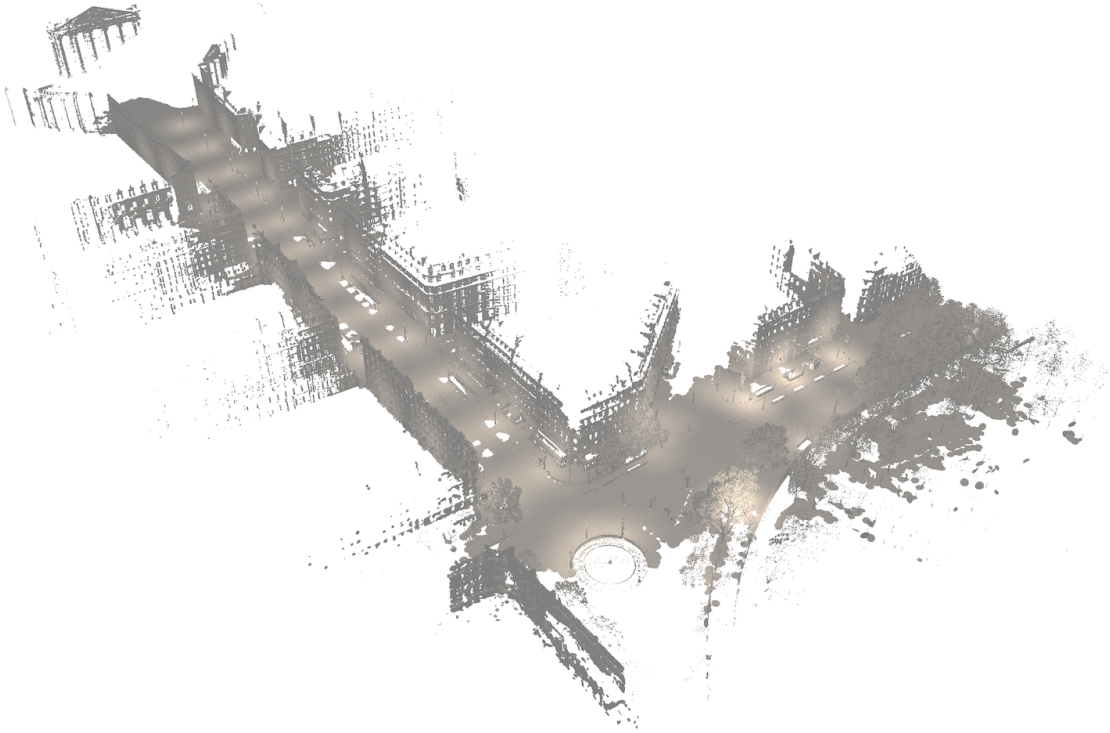


Figure 4.12: Showing the splat ray tracing results on the full PC3D-Paris dataset. The scene contains 8.7M million splats, 43 light sources and rendered at a resolution of 2560 x 1440, obtaining a real-time performance.

4.5.2.5 Modeling the Completed Scene

Now that we have the scene completion and scene modeling methods, we activate the scene completion module (see the pipeline in Figure 4.13) and complete the whole PC3D-Paris dataset. The output of the scene completion algorithm is then used to model the full scene with AdaSplats.

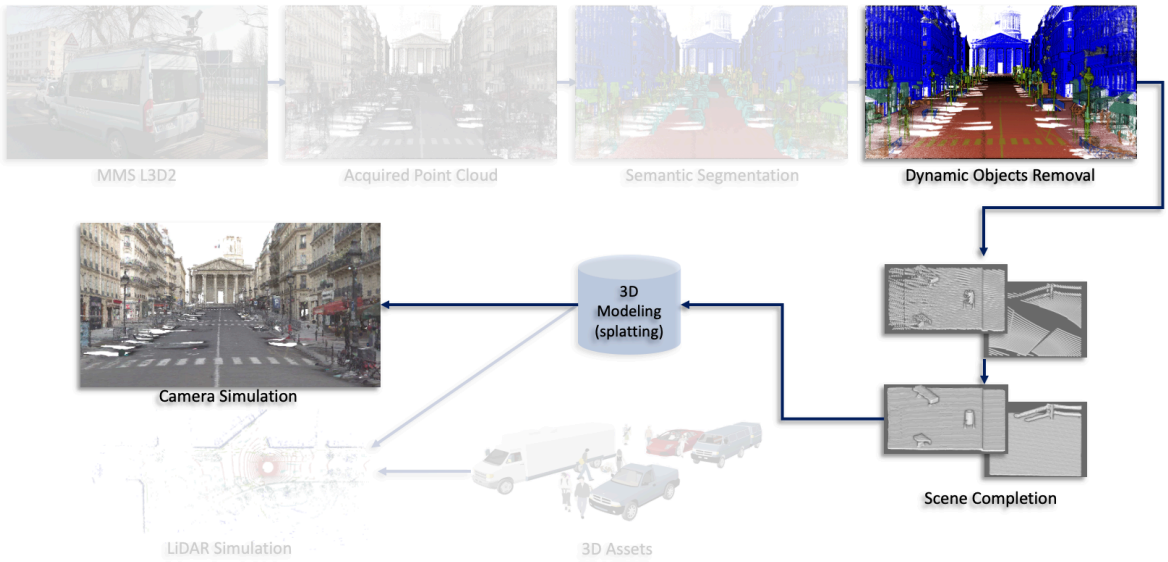


Figure 4.13: Combining the scene completion and scene modeling steps to obtain a more complete geometry.

Figures 4.14 and 4.16 show the original point cloud in RGB from two different view points, without dynamic objects. Figures 4.15 and 4.17 show the rendered AdaSplats model of the same views, respectively.

Figure 4.14: Original PC3D-Paris without dynamic objects (first view).

Figure 4.15: Results of ray tracing the AdaSplats model generated on the completed PC3D-Paris scene (first view).

Figure 4.16: Original PC3D-Paris without dynamic objects (second view).

Figure 4.17: Results of ray tracing the AdaSplats model generated on the completed PC3D-Paris scene (second view).

We see from the rendering of the splatted and completed scene that we are able to reduce the sparsity of the point cloud, while reducing the number and size of the missing regions, thanks to the resampling algorithm and the scene completion task, respectively. Morever, we observe that we are able to obtain a high-quality rendering, while maintaining color consistency, even though we only use color information from the nearest neighbor for new points. These results show a qualitative evaluation of the first steps in our simulation pipeline as shown in Figure 4.13, which will be used as input to our LiDAR simulation method.

4.6 Conclusion

Starting from a raw, or a completed point cloud with point-wise semantic labels, we introduced a new splatting method, which we call AdaSplats, following its ability to adapt the splat size based on the semantic class of its center point. Moreover, we presented a resampling algorithm that is able to re-distribute the points in the point cloud by removing some points in very dense regions and introducing new points in sparse regions, resulting in a more uniformly distributed point cloud, with a lower number of primitives. Furthermore, we implemented a real-time ray tracing algorithm, using OptiX to parallelize and accelerate our custom ray-splat intersection, using the ability of GPUs and the BVH structure, respectively. Finally, we introduced a blending method to refine our rendering, by including the contribution of overlapping splats, and taking into consideration the distance of the intersection from the splat center, to the final normal and color computation. Our simulation pipeline can be used to simulate a variety of sensors used by AVs, such as camera, LiDAR, RADAR, etc.. We demonstrated in this chapter the first steps toward realistic camera simulation, however, there are still improvements to explore, such as texture mapping, or using more recent point-based neural rendering methods [Bui et al. 2018; Xu et al. 2022] to increase the level of realism. In chapter 5, we focus on LiDAR simulation, which uses the modeled scene with AdaSplats, from the raw, or the completed point cloud (using the scene completion task), as input and leave other sensors to future works.

This work and a part of chapter 5, will result in a publication that is under submission in a journal, while the pre-print version is already on arxiv: https://arxiv.org/pdf/2203.09155.pdf

- Richa, J.P., Deschaud, J.E., Goulette, F. and Dalmasso, N., 2022. AdaSplats: Adaptive Splatting of Point Clouds for Accurate 3D Modeling and Real-Time High-Fidelity LiDAR Simulation. Remote Sensing, 14(24), p.6262.

Chapter 5

LiDAR Simulation

Contents

5.1 Introduction **92**
5.2 Related Work **94**
5.3 LiDAR Simulation **96**
5.3.1 New Trajectory Simulation 96
5.3.2 Firing Sequence Simulation 97
5.3.2.1 Velodyne HDL-64E 97
5.3.2.2 Firing Sequence Rays Generation 97
5.4 Experiments and Results **97**
5.4.1 LiDAR Simulation with offset trajectories in PC3D, SemanticKITTI and M-City 98
5.4.1.1 PC3D-Paris 99
5.4.1.2 SemanticKITTI 103
5.4.1.3 M-City 105
5.4.2 Simulation on the Completed PC3D-Paris 106
5.4.3 LiDAR Simulation integration in a full simulation software (ANSYS sensor SDK with SCANeR Simulator) 109
5.4.3.1 Dynamic Objects Addition and Scenarios Generation . . 109
5.5 Conclusion **110**

Abstract

A common approach to test and validate autonomous vehicles is to deploy the vehicle on the road for millions of miles with a human in the loop that needs to be always ready to take full control of the vehicle. This approach involves high risk to the human in the driver's seat, the pedestrians on the road, and other vehicles in the surrounding. Moreover, the cost in time and money is very high, which places limitations on the acceleration of the validation process. Simulation on the other hand, is a powerful alternative that can be used to accelerate the validation process. However, existing simulators either do not provide accurate geometry representation, which we propose to solve in the previous chapter, or they simply do not scale to large scenes and do not achieve real-time. In this chapter, we introduce a real-time GPU LiDAR simulator that is performed inside the scene modeled using AdaSplats, which benefits from a highly accurate geometry representation of real-world scenes and achieves a LiDAR simulation that is faster than real time, opening the possibility for massive simulation.

Résumé

Une approche commune pour tester et valider les véhicules autonomes consiste à déployer le véhicule sur la route sur des millions de kilomètres avec la présence de l'Homme dans la boucle qui doit toujours être prêt à prendre le contrôle total du véhicule. Cette approche implique un risque élevé pour l'être humain dans le siège du conducteur, les piétons sur la route et les autres véhicules dans les environs. De plus, le coût en temps et en argent est très élevé, ce qui limite l'accélération du processus de validation. La simulation, en revanche, est une alternative puissante qui peut être utilisée pour accélérer le processus de validation. Cependant, les simulateurs existants ne fournissent pas de représentation géométrique précise, ce que nous proposons de résoudre dans le chapitre précédent, ou ils ne s'adaptent tout simplement pas aux grandes scènes et n'atteignent pas le temps réel. Dans ce chapitre, nous introduisons un simulateur LiDAR GPU en temps réel qui est fait à l'intérieur de la scène modélisée à l'aide d'AdaSplats, qui bénéficie d'une représentation géométrique très précise des scènes du monde réel et réalise une simulation LiDAR plus rapide qu'en temps réel, ouvrant la possibilité de simulation massive.

5.1 Introduction

In this chapter, we tackle the fourth and final step in our pipeline, LiDAR simulation (see Figure 5.1), which aims to achieve real-time LiDAR simulation in the splatted scene. The input to this module is the splatted scene.

Algorithms used by AVs should be tested in different scenarios, especially events that rarely appear in the real world, different illumination and weather conditions, and other identified corner cases. For this, a shift towards simulated environments [Dosovitskiy et al. 2017a] was necessary, mainly to reduce cost and increase safety. Although handcrafted simulators provide the leverage of testing AV algorithms, they introduce a large domain gap between real-world and simulated environments. This gap arises from the difference between the almost perfect geometry present in such simulators, as they contain carefully designed simple 3D objects, which results in simplistic environments. Manually constructed environments also contain a lower objects diversity, due to the time needed to create 3D objects that resemble their real-life counterparts, while a higher diversity of more complex object are present in the real world.

The limitations of manually constructed simulation environments gave rise to more realistic simulators [Manivasagam et al. 2020; Fang et al. 2020] using real-world point clouds collected using LiDAR scanners mounted on a MMS. These methods model the real-world

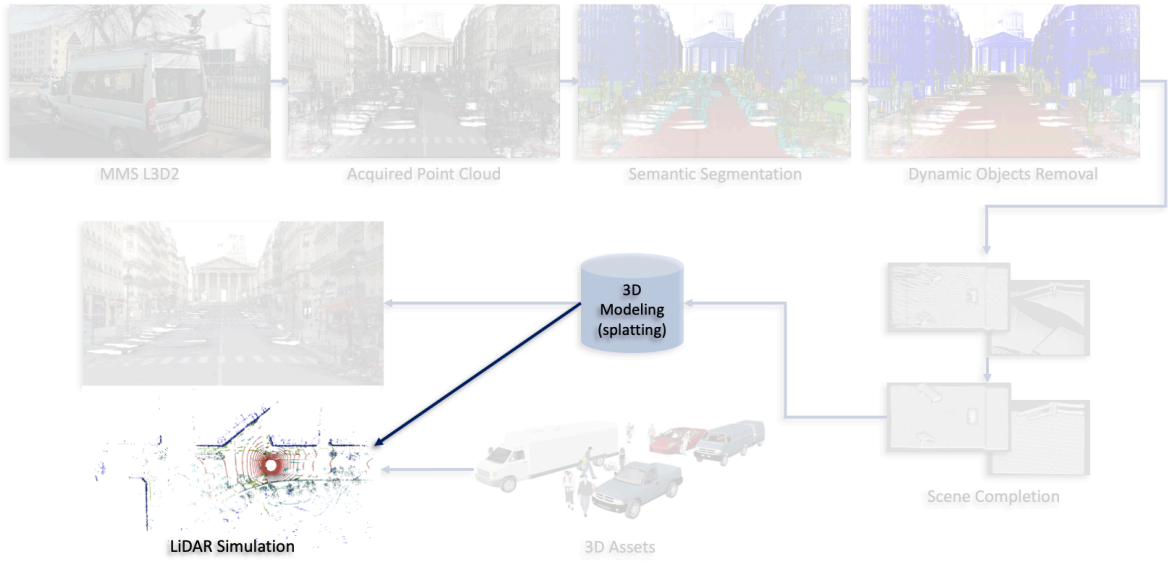


Figure 5.1: The final step in our pipeline consists of performing real-time LiDAR simulation in the modeled scene. We first start by simulating the LiDAR in the splatted scene without completion and without dynamic objects addition.

from outdoor point clouds using well-known splatting techniques [Zwicker et al. 2001; Pfister et al. 2000] to reduce the domain gap. Automatic surface reconstruction methods could be used, however, as we prove in chapter 4 and will prove in this chapter as well, they are not well suited to be used on data collected from a MMS. The previously proposed methods [Manivasagam et al. 2020; Fang et al. 2020] simulate the LiDAR sensor in a splatted environment, resulting in higher accuracy on the tasks learned from data collected from the splatted environment when compared to data collected from handcrafted simulators. However, these methods 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.

In chapter 4, we have shown that our splatting approach provides the most accurate surface modeling compared to the Basic Splatting approaches used by previous LiDAR simulation algorithms leveraging real-world data. Moreover, we demonstrated how we make use of the OptiX library and GPU architecture to accelerate the ray-primitive intersection and parallelize the ray-casting process to achieve real-time rendering. LiDAR sensors cast a lower number of rays per frame, which reduces the computation requirements resulting from the reduction in the number of rays cast in parallel and traversing the BVH structure. This enables us to achieve faster-than-real-time LiDAR simulation as we will see in the experiments section 5.4. After obtaining the simulation results on the original point cloud data, we embed the SC step, use the full simulation pipeline that we introduce throughout this work including the Scene Completion (SC), and simulate the LiDAR sensor on the completed scene.

Testing the surface representation accuracy is not an easy task. To do so, we perform the LiDAR simulation using different surface representations on datasets acquired in different LiDAR configurations, namely, AV and mapping, then we finally compute the cloud-to-cloud (C2C) distance between the simulated point cloud and the original one. We call AV configuration a LiDAR mounted on a vehicle with a 0° roll, pitch and yaw angles, in which the LiDAR maximizes the number of points in the surrounding of the vehicle and does not care about objects on higher elevation angles. On the other hand, in mapping configuration, the LiDAR is pitched at 45°, to maximize the acquired geometry information on higher elevation angles, which results in the full scene geometry, including building facades, tree leaves, etc...

We propose a fully automatic pipeline for accurate 3D modeling of the environment

and physics-based simulation of sensors used by AVs from real-world data, which speeds up AVs testing and validation, and reduces the risks associated to this task. In our pipeline, we focus on accurate geometric modeling of the static environment, which excludes moving obstacles. However, the proposed pipeline is highly dynamic and can be modified to include intermediate steps, or increase its accuracy, such as including texture mapping. In a final step, we introduce some of the extentions that can be added to the pipeline, which was integrated at ANSYS on the product side. In the extended pipeline, we include CAD models of dynamic objects, namely, moving vehicles and pedestrians, with predefined trajectories and simulate the LiDAR sensor to augment the scenarios domain. This can be used to identify corner cases and test the behavior of AVs.

Our contributions can be summarized as follows:

- Faster-than-real-time GPU ray casting in the splat model for LiDAR sensor simulation.
- The use of CAD models inside the splatted model for scenarios generation.

5.2 Related Work

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. In previous methods, 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 [Dosovitskiy et al. 2017a] or BlenSor [Gschwandtner et al. 2011], or game engines (GTA-V), offers the ability to simulate the LiDAR sensor and collect scans [Wu et al. 2018; Hurl et al. 2019b; Yue et al. 2018], 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 [Wu et al. 2019; Zhao et al. 2020], 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 [Deschaud et al. 2012] extended in [Tallavajhula et al. 2018] 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.

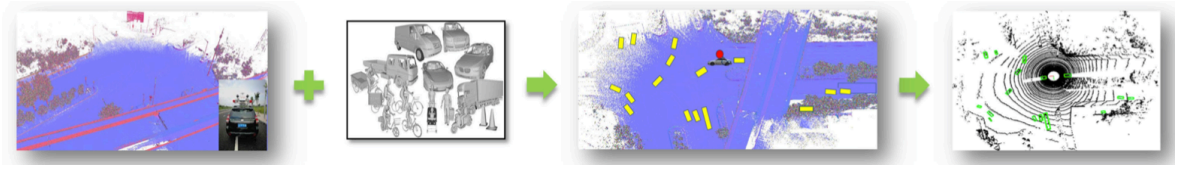


Figure 5.2: In [Fang et al. 2020], first the scene is modeled, then CAD models of dynamic objects, such as vehicles, pedestrians and cyclists are added to generate a new scenario in which the LiDAR is simulated. Source [Fang et al. 2020]

More recent approaches [Fang et al. 2020; Manivasagam et al. 2020] acquire data using a LiDAR mounted on a MMS and model the 3D geometry using splatting techniques after removing the dynamic objects in the foreground (e.g., pedestrians, cars, etc.). These approaches add dynamic objects on top of the reconstructed 3D environment in the form of

CAD models like in [Fang et al. 2020], see Figure 5.2, or in the form of point clouds collected from the real world like in [Manivasagam et al. 2020], see Figure 5.3. In the first approach [Fang et al. 2020] 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 approach [Manivasagam et al. 2020], they use Embree [Wald et al. 2014] to accelerate the ray-primitive intersection, however, Embree runs on CPU and is still far from real-time sensor simulation, since the parallelization of the ray casting is limited to the number of CPU cores. Furthermore, although they achieve higher performance from their simulated data compared with simulated data in CARLA on object detection and SS 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. A recent method [Guillard et al. 2022], follows the works on raydrop done in [Manivasagam et al. 2020] and [Fang et al. 2020] and suggests to reduce the domain gap between simulated and real-world LiDAR data by jointly learning to drop rays and predict intensities on the simulated LiDAR frames. They project the LiDAR frames from public datasets [Behley et al. 2019; Sun et al. 2020] onto their corresponding RGB images, then densify the LiDAR points in the image space, by connecting neighboring triplets of points creating a "triangular mesh" in image space to be used as the network input. Then the network is trained in a supervised way, with the target being the original range image (containing the missing returns), and used to predict intensities and ray drop on LiDAR frames simulated in CARLA simulator [Dosovitskiy et al. 2017a], then the final data from CARLA is used to augment semantic segmentation datasets [Behley et al. 2019; Sun et al. 2020]. They report improved results on the task of semantic segmentation using the augmented dataset. Although intensities and ray drop learned jointly may improve the results of downstream AVs tasks, the simulated data in CARLA is still far from realistic. A more recent work [Marchand et al. 2021] simulates an aerial laser scanner to scan a reconstructed scene as a base model. After scanning they use distance metrics to evaluate the different reconstruction algorithms. However, they use a meshed model as ground truth, which is subject to reconstruction error. Instead, we use the ground truth point cloud and compare the simulated to original point clouds.

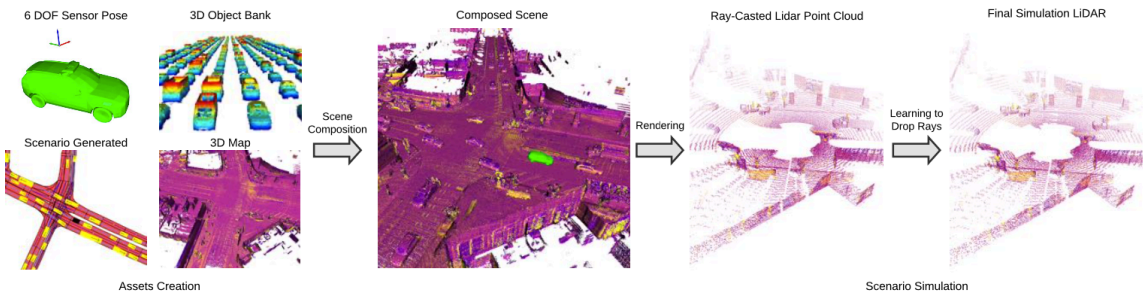


Figure 5.3: In [Manivasagam et al. 2020], first the scene is modeled, then dynamic objects models collected from the real world, such as vehicles are added to generate a new scenario in which the LiDAR is simulated. Source [Manivasagam et al. 2020]

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 [Wald et al. 2014] 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 [Parker et al. 2010], on the other hand, builds a 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. To this end, we choose to use OptiX and implement the ray-splat intersection using CUDA to accelerate the intersection process and achieve real-time LiDAR simulation.

5.3 LiDAR Simulation

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 [Manivasagam et al. 2020; Fang et al. 2020] 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 the splat ray tracing method we introduced in section 4.4. With the implementation of ray-casting we can simulate any LiDAR type. We implement the firing sequence of multiple LiDAR sensor models, launch the rays in the splatted environment and return the ray-splat intersection points.

5.3.1 New Trajectory Simulation

To simulate the LiDAR sensor, we generate new trajectories to prevent the intersection of the ray with the centers of the splats, which are the points in the original point clouds. If the same trajectory is used to simulate the same sensor in the same configuration, we would obtain the same points of intersection as we had in the original point cloud. This raises the question of whether our modeling is accurate, since we would not need an accurate modeling, or a hole-free approximation if our rays are not intersecting random points on the splats.

To change the scan pattern and prevent ray-center intersection when simulating using the firing sequence implementation, we shift the original trajectory provided by the sensor positions in PC3D and SemanticKITTI. As for M-City, since we do not have the sensor positions and since the dataset was acquired using a TLS, we perform the simulation on interpolated 3D points on a linear line segment.

In PC3D, we have the absolute sensor translation and rotation angles on the x, y, z axes, defined with respect to the initial frame's position and orientation, for every frame. In our setup, we use the right hand convention with z up. The transformation (rotation and translation) of a rigid body can be described by the chain of the individual transformations of each axis in homogeneous coordinates. More specifically, rotations on the x, y and z axes can be respectively represented by their corresponding rotation matrices $R_x(\theta)$, $R_y(\beta)$ and $R_z(\gamma)$.

We obtain the new position of the LiDAR, where we neglect the last entry from the final vector, since the weight in homogeneous coordinates is 1, obtaining a 1x3 positional vector. The extrinsic parameters of the LiDAR are already known and integrated in the calculation of its position, so the translations and rotations that we have during the acquisition are already in the reference frame of the LiDAR. Having said that, we do not need to transform the position from the vehicle to the LiDAR reference frame. If the reported position corresponds to the vehicle reference frame, then an extra transformation should initially be done.

5.3.2 Firing Sequence Simulation

Using our pipeline, we can simulate any LiDAR model. However, in this thesis, we focus on simulating the Velodyne HDL-64E LiDAR sensor. This LiDAR has an emiter and a receiver, it emits vertical equally spaced laser beams and compute the Time of Flight (ToF) of each beam to deduce the distance to the object, in the reference frame of the vehicle, that reflected the light pulse. An azimuth revolution is a full 360° azimuth turn. We place the simulated LiDAR in AV configuration and approximate the origin of the sensor beams by making them equal to the sensor position.

5.3.2.1 Velodyne HDL-64E

The Velodyne HDL-64E emits 64 vertical laser beams, it has an azimuth angular resolution of 0.08° at 10 HZ with a 360° horizontal FOV, an elevation angular resolution of 0.419°, and ranges between -24.8° and +2.0°, summing to a 26.8° vertical FOV. Moreover, each laser emits 4500 laser pulses per azimuth revolution, which sums to 288,000 laser pulses over the 64 lasers, working between 5 Hz and 15 Hz, and has a range of 120 meters.

5.3.2.2 Firing Sequence Rays Generation

When simulating the LiDAR in AV configuration, there is no initial roll, pitch, or yaw angles, which results in a 360 rotation around the vertical axis (z in our virtual environment). If an initial transformation is required (e.g., initial pitch on the y axis), we pitch the initial ray $\mathbf{r\hat{a}y}^1$ by an angle η^1:

$$\bar{\mathbf{r\hat{a}y}}^1 = R_y(\eta^1)\mathbf{r\hat{a}y}^1 \tag{5.1}$$

If the LiDAR has no initial adjustments (roll, pitch and yaw angles are equal to 0°), the previous step is ignored and $\bar{\mathbf{r\hat{a}y}}^1 = \mathbf{r\hat{a}y}^1$. We perform the elevation and azimuth rotations to generate the rays for the LiDAR frame at timestamp ts by accumulating the angles using the angular resolution of the simulated LiDAR and transforming $\bar{\mathbf{r\hat{a}y}}^1$ into the ray corresponding to the new angle (azimuth on z and elevation on y). We first apply the azimuth rotation followed by n elevation angles for the n laser beams

$$\mathbf{r\hat{a}y}_{ts}^{i,j} = R_z(\psi^i)R_y(\eta^j)\bar{\mathbf{r\hat{a}y}}^1 \tag{5.2}$$

where ψ^i and η^j are the azimuth angle i and elevation angle j, respectively. For the rays, we only need the direction of the vector, its position is provided separately, consequently, when launching the rays, we pass its origin (LiDAR position) and direction.

5.4 Experiments and Results

We use the LiDAR simulation described above and compare the accuracy of the surface representation methods introduced in chapter 4. Validating the correct modeling of the environment to simulate a LiDAR is a complex task. The only experiments done by LiDARsim [Manivasagam et al. 2020] and AugmentedLiDAR [Fang et al. 2020] were comparisons of learning results of deep networks between their simulated 3D data and data simulated under a simplified synthetic environment from CARLA [Dosovitskiy et al. 2017a]. 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.

In our comparisons, 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. We demonstrate this qualitatively in a separate step in section 5.4.3.

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 SS [Thomas et al. 2019] achieves similar results to AdaSplats using ground truth semantic information.

5.4.1 LiDAR Simulation with offset trajectories in PC3D, SemanticKITTI and M-City

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 TLS (see Figure 5.4). More specifically, we choose (1) the Paris dataset from PC3D [Deschaud et al. 2021] (PC3D-Paris), acquired using a Velodyne HDL-32E in mapping configuration. The high objects diversity allows us to compare the modeling capacities of the different techniques in real situations. (2) SemanticKITTI [Behley et al. 2019] acquired in a different environment, using a different LiDAR, namely Velodyne HDL-64E and mounted in a different configuration (AV configuration). (3) M-City, which is acquired using a TLS at fixed points in a controlled environment with no dynamic objects. The diversity of the datasets and the used sensors allows us to test the generalization capability of our pipeline, especially our surface representation and LiDAR simulation.

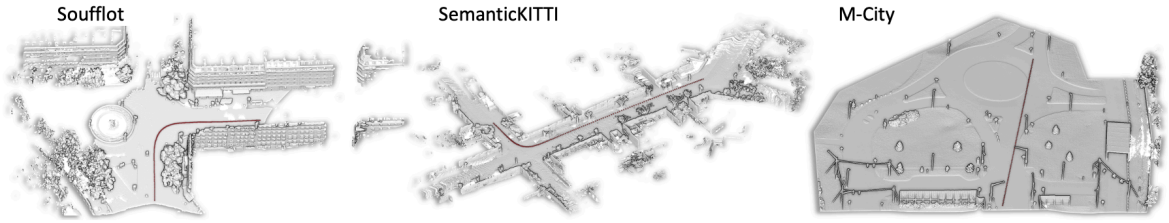


Figure 5.4: Point clouds used in the experiments. PC3D-Paris, SemanticKITTI, and M-City, from left to right, respectively. In red, the trajectory used for simulation.

For PC3D-Paris, we use parts from Soufflot-1 and Soufflot-2, creating a point cloud containing 12.3 million points. For SemanticKITTI, we use the first 150 scans from sequence-00 accumulated with a LiDAR SLAM [Deschaud 2018], which results in a point cloud with 15.5 million points. Finally, for M-City, we take the part containing 17.5 million points, which represents 25% of the full dataset. We remove dynamic objects from SemanticKITTI and PC3D-Paris using the semantic information and model the scene to be used for LiDAR simulation on a static background only.

We compare four different surface representations: two surface reconstruction methods (Screened Poisson [Kazhdan et al. 2013] and IMLS [Kolluri 2008]), and the two splatting methods we introduced in section 4.3 (Basic Splats and our proposed AdaSplats methods). We use the latest available code (version 13.72) for Screened Poisson surface reconstruction to mesh the point cloud, obtaining the finest mesh with : octree depth of 13, Neumann boundary constraints, samples per node as 2 and other parameters as default values. We used also SurfaceTrimmer to remove parts of the reconstructed surface that are generated in low-sampling-density regions with trim value as 10.0 and other parameters as default. For IMLS, we use a voxel size of 7 cm, we fix sigma to two times the voxel size and perform a sparse grid search with a truncation of 3 voxels, where we fill the IMLS signed distance function values only in voxels near the surface and use the marching cubes algorithm [Lorensen et al. 1987] to extract the iso-surface.

Compared with concurrent LiDAR simulation methods, such as LiDARsim [Manivasagam

et al. 2020] and AugmentedLiDAR [Fang et al. 2020], 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 the first validation, we simulate the Velodyne HDL-64E LiDAR in AV configuration (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. We use our ray-splat intersection method implemented with 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 5.4). More precisely, the offset for PC3D-Paris is $[1.0, 1.0, -1.0]^T$ along the x, y, and z axes and $[1.0, 1.0, -0.5]^T$ for SemanticKITTI. For M-City, we generate a linear trajectory across the dataset. For all of our LiDAR simulation experiments, we use an 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):

$$C2C(\mathcal{P}_{sim}, \mathcal{P}_{ori}) = \frac{1}{|\mathcal{P}_{sim}|} \sum_{x \in \mathcal{P}_{sim}} \min_{y \in \mathcal{P}_{ori}} ||x - y||_2 \tag{5.3}$$

where $\mathcal{P}_{sim}$ and $\mathcal{P}_{ori}$ are the simulated and original point clouds, respectively.

5.4.1.1 PC3D-Paris

Figures 5.5 and 5.6 illustrate qualitative results comparing the accumulated point clouds from the LiDAR simulation with the different surface representations. In Figure 5.5, the last image is the original point cloud used to model the environment. The other images are an accumulation of simulated scans (in blue we show one simulated LiDAR scan). We can see in both figures that AdaSplats results in higher-quality LiDAR data when compared to Basic Splats and other meshing techniques.

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 [Kazhdan et al. 2013] and IMLS [Kolluri 2008] do not work well on outdoor noisy LiDAR data, especially on thin objects. These surface reconstruction methods result in artefacts on open shapes: borders are dilated because these functions attempt to close the surface, as they are performing inside/outside classification. To limit this effect, we truncate the IMLS function at 3 voxels and perform surface trimmering with Poisson. However, we can still see artifacts on Figure 5.5 in the red and orange areas.

Our method is also verified quantitatively (see Table 5.1). We compare the meshing time of Poisson and IMLS, and the splatting time of Basic Splats and AdaSplats. AdaSplats-KPConv includes KPConv for automatic SS (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. Using KPConv, our adaptive splatting and resampling technique 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

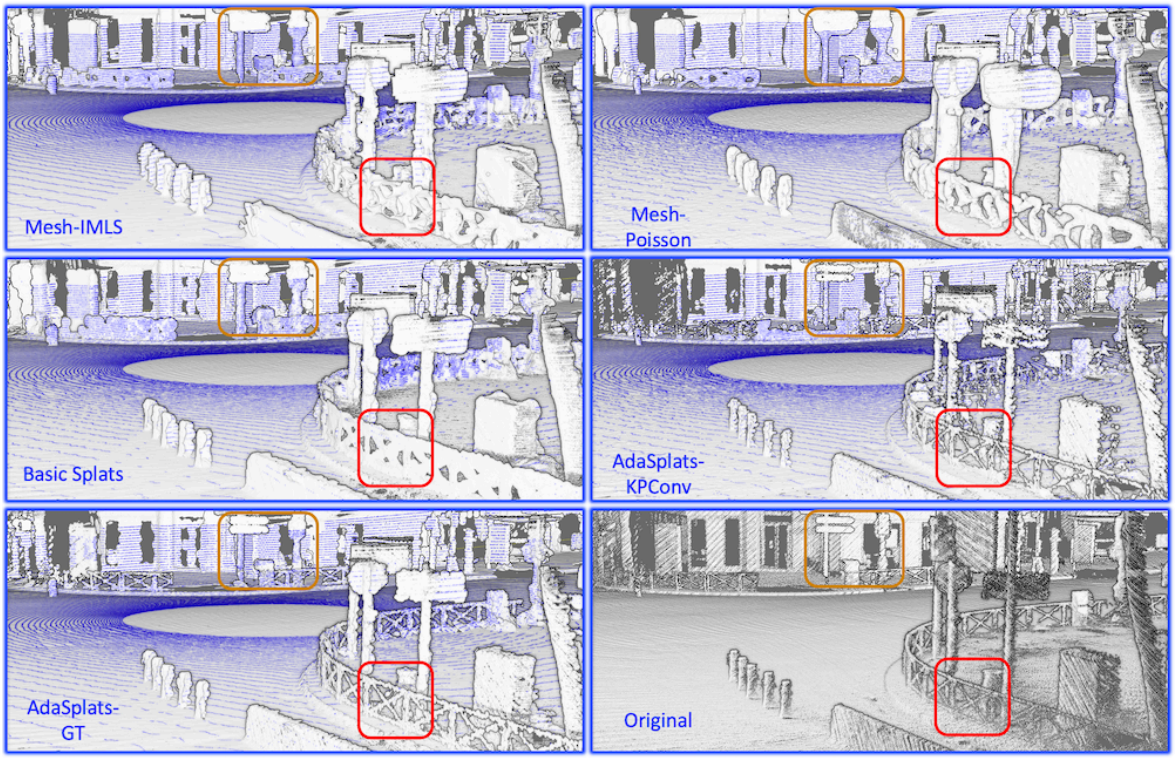


Figure 5.5: Comparison of simulated LiDAR data using different reconstruction and modeling methods on PC3D-Paris. Top: simulation in meshed IMLS (left) and Poisson (right). Middle: the simulation with Basic Splats (left) and AdaSplats-KPConv (right). Bottom: the simulation with AdaSplats-GT (left) and original point cloud (right).

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 5.3). AdaSplats-KPConv improves the accuracy over Basic Splats, while AdaSplats-GT shows that with improved semantics, the simulated data can be the closest to the original.

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	451	1.72M	205 Hz	**1.97 cm**

Table 5.1: 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 clouds.

To see the effect of resampling on the final simulation, we remove the resampling step from the AdaSplats generation and report the results in Table 5.2. We observe that without

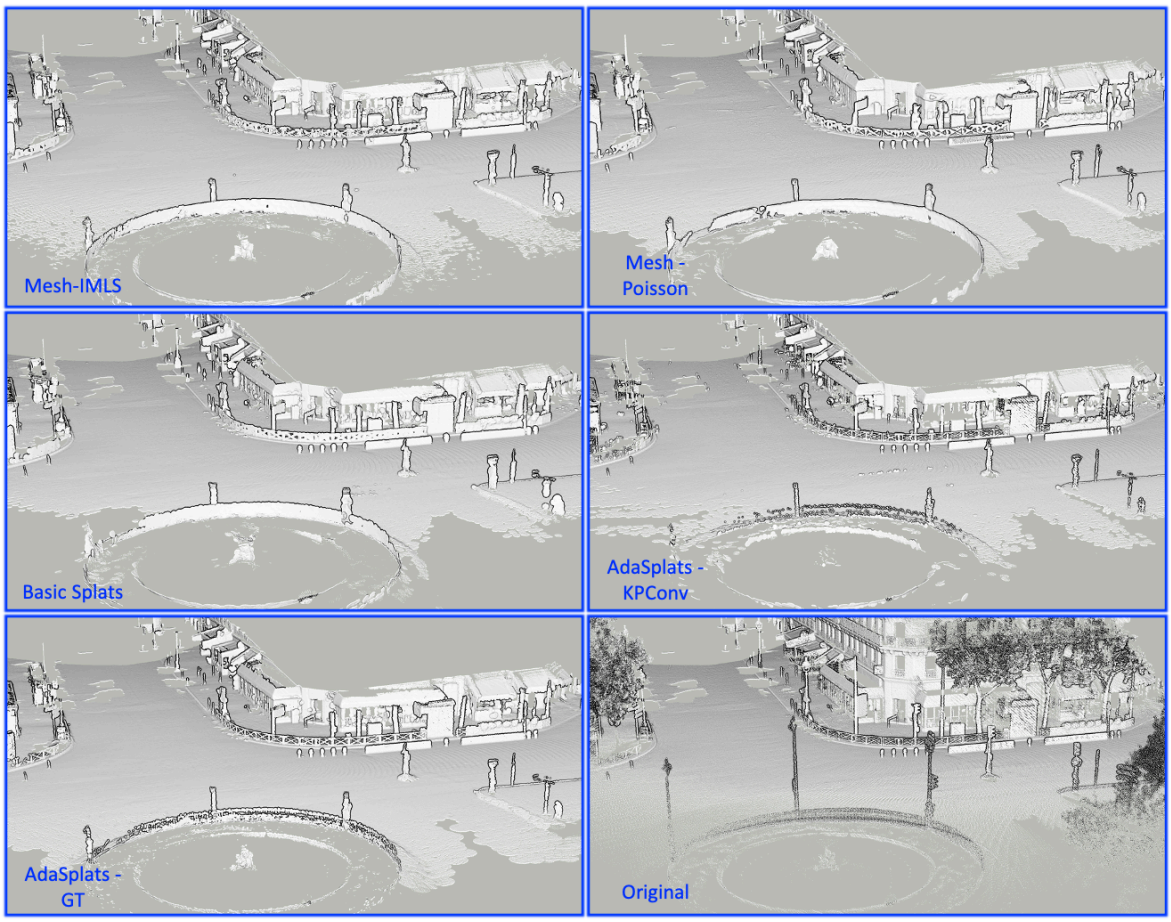


Figure 5.6: Another view of the comparison of simulated LiDAR data using different reconstruction and modeling methods on PC3D-Paris. Top: simulation in meshed IMLS (left) and Poisson (right). Middle: the simulation with Basic Splats (left) and AdaSplats-KPConv (right). Bottom: the simulation with AdaSplats-GT (left) and original point cloud (right).

resampling, we obtain a higher number of geometric primitives, which affects the rendering frequency as well and the result in a higher C2C distance. This demonstrates that with our resampling technique, we are able to increase the accuracy of splats generation, while resulting in a lower number of generated primitives thanks to the re-distribution of points.

Model	**Gen T (in s)**	**Gen Prim (#)**	**Sim Freq (in Hz)**	**C2C (in cm)**
AdaSplats-GT no resampling	169	2.84M	180 Hz	1.99 cm
AdaSplats-GT	451	1.72M	205 Hz	**1.97 cm**

Table 5.2: Results of the LiDAR simulation on the PC3D-Paris using AdaSplats with ground truth semantics without resampling (top row), compared to the the simulation on the resampled model (bottom row). We report the time taken (Gen T) in seconds to generate the primitives, 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 clouds.

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. However,

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 5.3.

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.

We make an important observation, which is, we always obtain better quantitative and qualitative results independent from the source of the semantic classes. This proves that our method achieves a better scene modeling, especially on fine structures, even if the semantic information is not perfect (see Tables 5.3 and 5.4).

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	**2.4**	2.3	1.8	**2.2**

Table 5.3: 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. The AdaSplats methods include resampling

We measure the contribution of resampling on thin structures, we perform once more the semantic cloud-to-cloud distance on the AdaSplats model without resampling and report the results in Table 5.4. We observe a drop in performance, which is seen from the higher distance we obtain between the simulated and the original point clouds on thin structures.

Model	**Fences**	**Poles**	**Traffic Signs**	**Average**
AdaSplats-GT no resampling	2.5	2.4	1.8	2.3
AdaSplats-GT	**2.4**	2.3	1.8	**2.2**

Table 5.4: Cloud-to-Cloud distance (in cm) computed on PC3D-Paris for points that belong to classes of thin structures, between the simulated using AdaSplats without resampling and original point cloud

In Tables 5.2 and 5.4, we compare our AdaSplats method with and without resampling. 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 from the simulation results inside the scene modeled 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.

5.4.1.2 SemanticKITTI

Figures 5.7, 5.8 and Table 5.5 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. The generation time of AdaSplats-KPConv, includes the inference time of KPConv, which is 10 minutes for the first 150 frames of sequence 00.

Model	Gen T (in s)	Gen Prim (#)	Sim Freq (in Hz)	C2C (in cm)
Mesh - Poisson	796	5.97M	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-KPConv	1166	6.11M	157 Hz	2.2 cm
AdaSplats-GT	544	4.56M	180 Hz	**2.0 cm**

Table 5.5: 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.

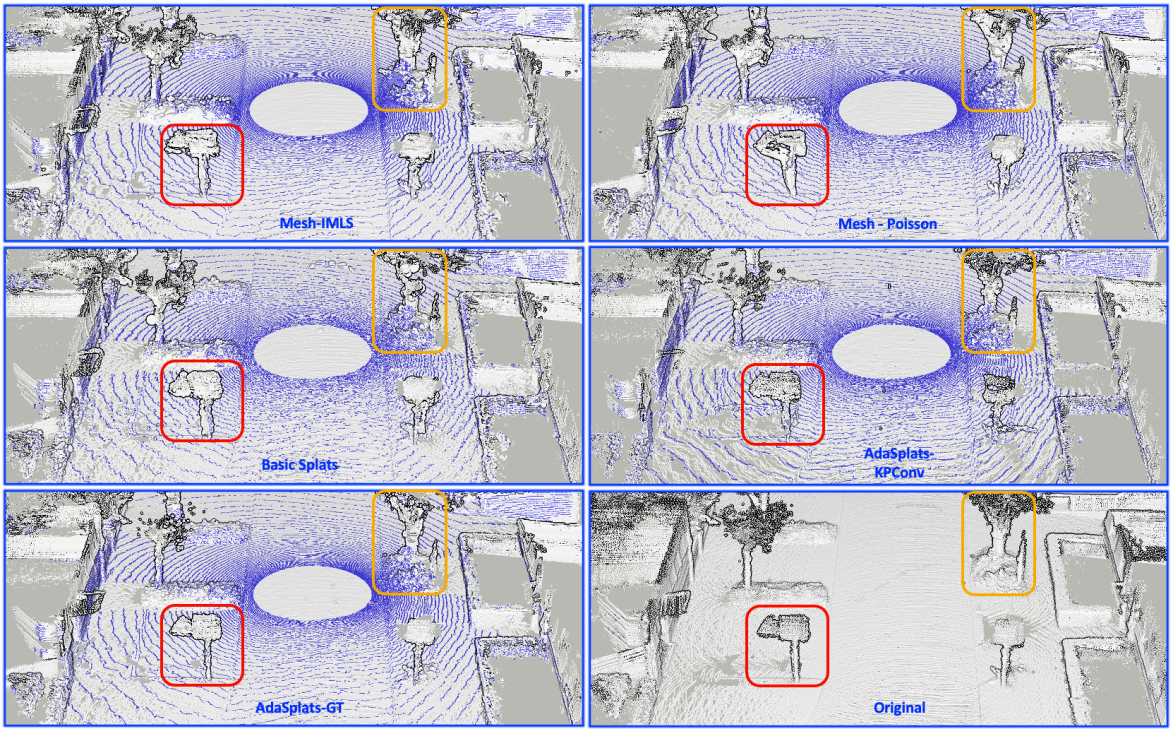


Figure 5.7: Comparison of simulated LiDAR data using different reconstruction and modeling methods on SemanticKITTI. The top row: the simulation in meshed IMLS (left) and Poisson (right). The middle row: the simulation with Basic Splats (left) and AdaSplats-KPConv (right). The bottom row: the simulation with AdaSplats-GT (left) and the original point cloud (right)

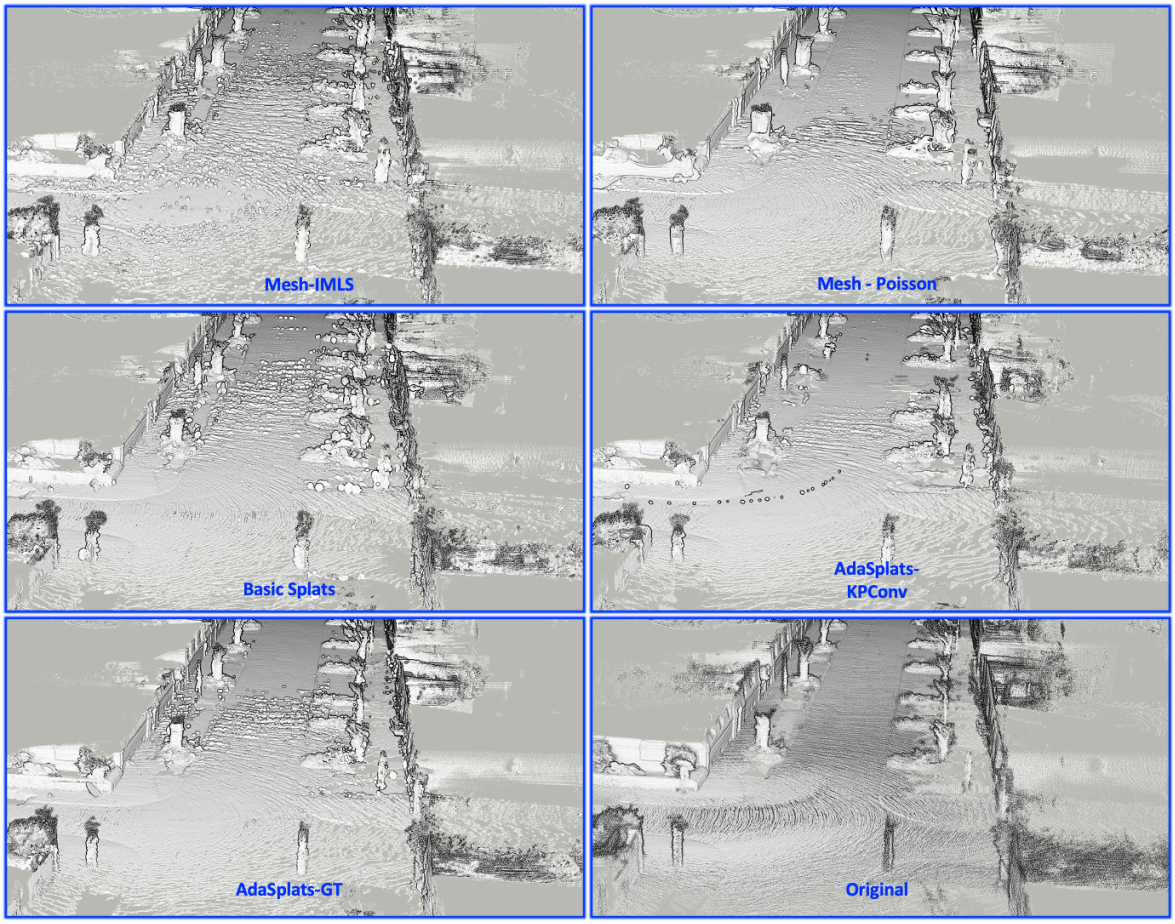


Figure 5.8: Another comparison of simulated LiDAR data using different reconstruction and modeling methods on SemanticKITTI. The top row: the simulation in meshed IMLS (left) and Poisson (right). The middle row: the simulation with Basic Splats (left) and AdaSplats-KPConv (right). The bottom row: the simulation with AdaSplats-GT (left) and the original point cloud (right)

5.4.1.3 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, which took one month of manual work, for the part that we use (25% of the full dataset). Hhence we use this manually meshed model, performing a comparison against a carefully reconstructed scene. Moreover, we cannot train KPConv on this small dataset, so we do not include AdaSplats-KPConv in the comparisons. We can see from the results in Figure 5.9 and Table 5.6 that AdaSplats still provides accurate surface modeling capabilities even without a correct normals orientation. Figure 5.9 and Table 5.6 show qualitative and quantitative results comparing the simulated point clouds, using M-City, inside the different scene models.

Model	**Gen T (in s)**	**Gen Prim (#)**	**Sim Freq (in Hz)**	**C2C (in cm)**
Mesh - Manual	1 month	71.5K	259 Hz	7.0 cm
Basic Splats	199	5.82M	110 Hz	1.7 cm
AdaSplats-GT	513	3.01M	204 Hz	**1.5 cm**

Table 5.6: 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.

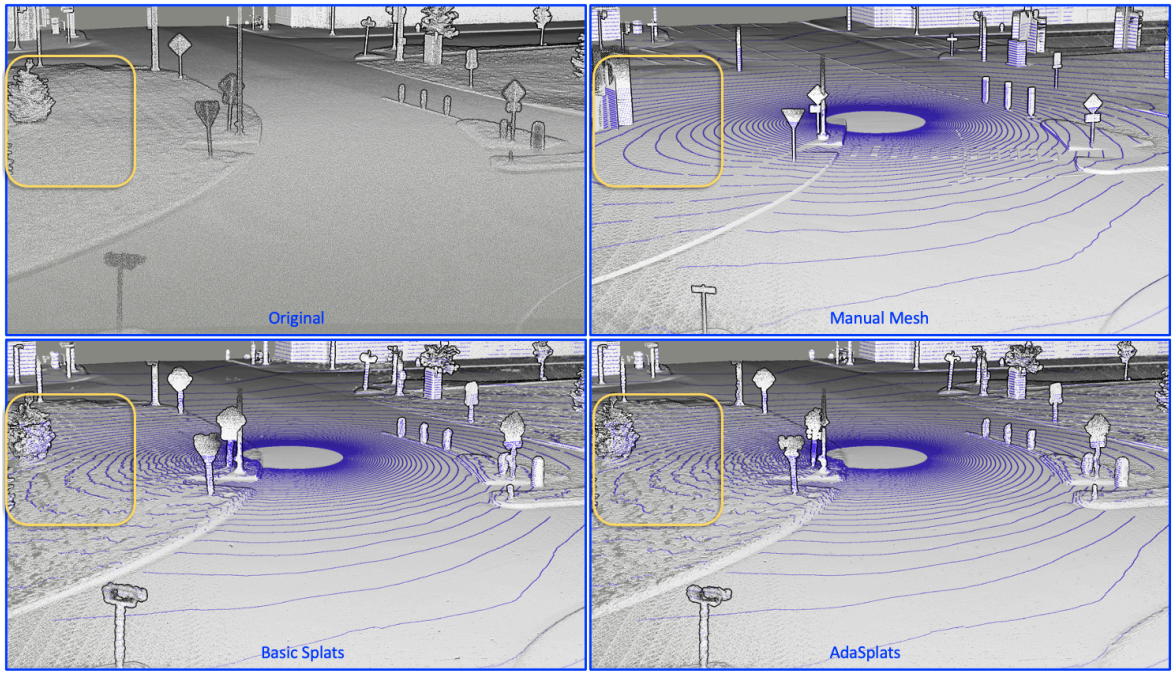


Figure 5.9: Comparing the manually meshed (bottom left) and automatically splatted (bottom right). Original M-City point cloud (top image). Modeling vegetation is not an easy task and usually requires different ray-primitive intersection methods for a more accurate rendering.

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).

5.4.2 Simulation on the Completed PC3D-Paris

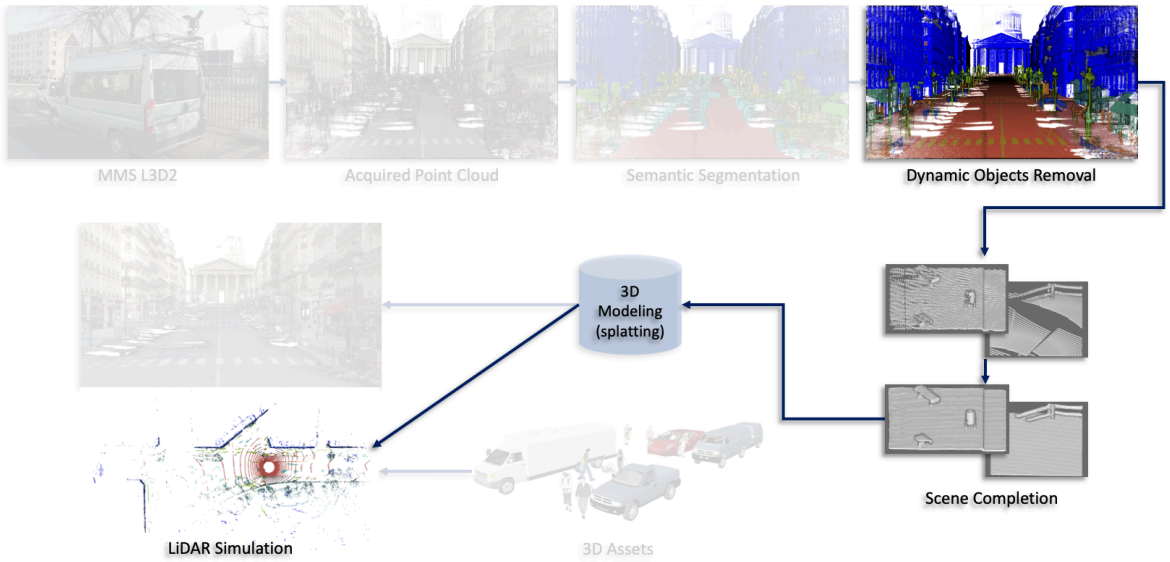


Figure 5.10: The simulation pipeline including the scene completion, scene modeling and LiDAR simulation modules.

We integrate the completion module as demonstrated in the simulation pipeline Figure 5.10 and perform scene completion on the dataset PC3D-Paris, followed by AdaSplats to generate the modeled environment and simulate the HDL-64E Velodyne LiDAR on the same trajectory used in section 5.4.1.1. Figure 5.11 provides qualitative results, showing the effect of completion on the final simulation. We can see that we are able to obtain high-fidelity simulation even with low density in the acquired point cloud. Figures 5.12 and 5.13 shows the full simulated point cloud and the same cloud with point-wise semantic labels, respectively. During the splats generation step, we keep splat-wise semantic labels, which can be retrieved at the time of intersection when simulating the LiDAR sensor.

Figure 5.11: Close-up comparison. Original point cloud (top). Simulated point cloud (bottom).

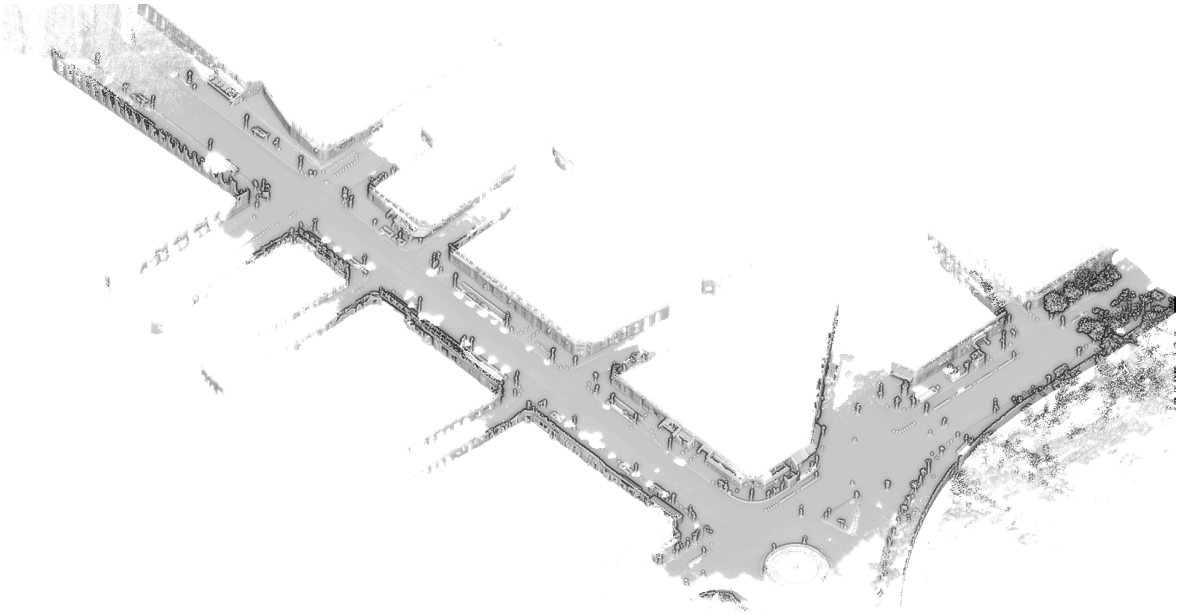


Figure 5.12: Simulated Velodyne HDL-64E LiDAR in PC3D-Paris with scene completion. Output: accumulated point cloud.

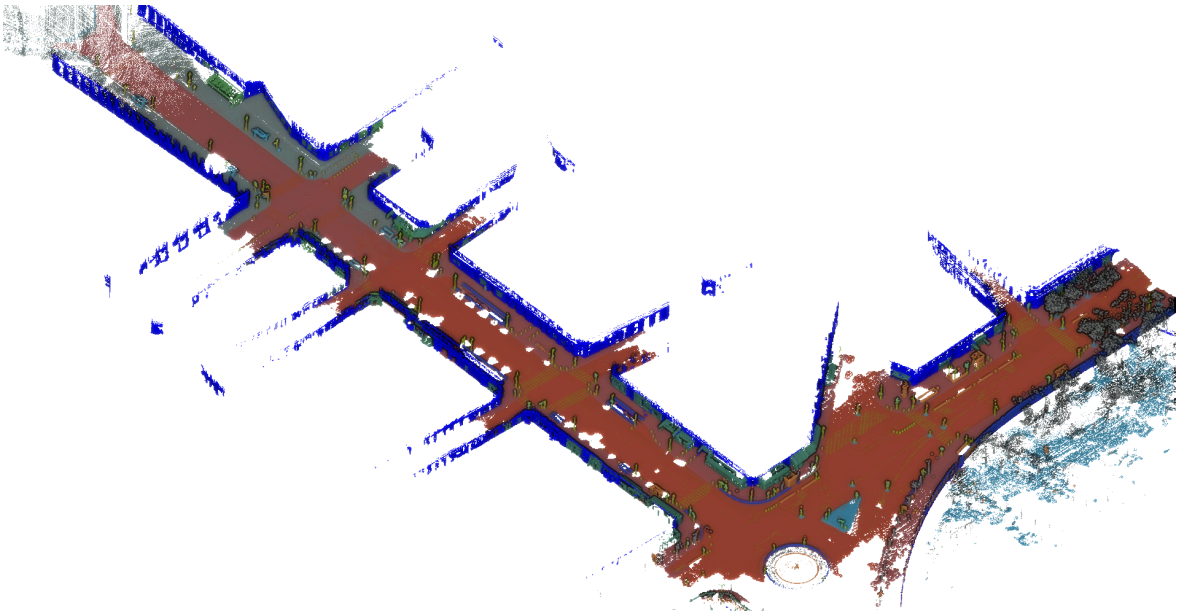


Figure 5.13: Simulated Velodyne HDL-64E LiDAR in PC3D-Paris with scene completion. Output: accumulated point cloud with point-wise semantic labels.

5.4.3 LiDAR Simulation integration in a full simulation software (ANSYS sensor SDK with SCANeR Simulator)

AdaSplats surface modeling and the ray-splat intersection algorithm were integrated in the sensor SDK at ANSYS. After the integration, we performed the LiDAR simulation using the more complex LiDAR model, which simulates the complete physical model of the LiDAR including the emitter, receiver and light beam simulation.

5.4.3.1 Dynamic Objects Addition and Scenarios Generation

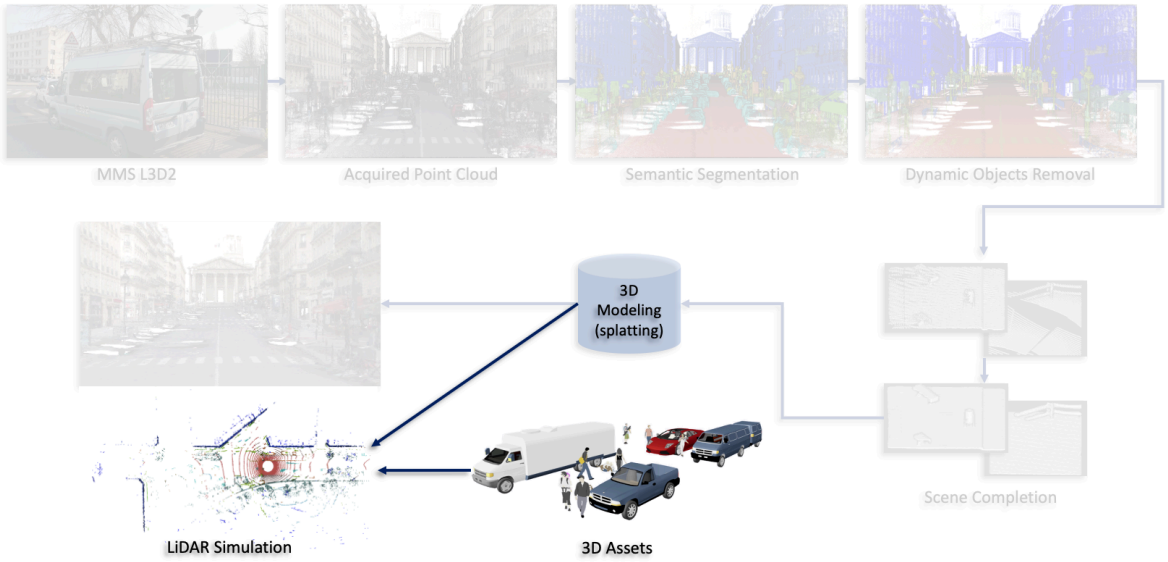


Figure 5.14: Updated simulation pipeline to include meshed dynamic objects, such as pedestrians and vehicles to generate novel scenarios.

As a final step, we include pedestrians and cars from the bank of assets available at ANSYS, generate new scenarios and simulate the LiDAR sensor (see Figure 5.14).

Multiple geometric primitives can be defined in OptiX, which are assigned unique IDs. A BVH can be built from each group of primitives, which can later be used to accelerate the ray primitive intersection. When intersecting a primitive through traversing the BVH, we retrieve its unique ID, which can be used to invoke the corresponding intersection program (ray-splat intersection program for splats and ray-triangle intersection program for triangles) on the device (GPU). By doing this, we can achieve hybrid splat-mesh intersection.

The hybrid intersection makes it possible to add CAD objects, such as cars and pedestrians in the scene. The ability to intersect multiple primitives and the availability of in-house constructed CAD models of cars and pedestrians at ANSYS offers us the possibility to generate new scenarios in the splatted static background. We add cars and pedestrians, and use SCANeR to generate the scenarios and simulate the LiDAR provided by Sensor SDK, see the results of scenarios generation in Figure 5.15. The REPLICA project bases the validation of AV tasks on SCANeR, because of its ability to generate very complex scenarios with detailed environments. However, SCANeR is not able to reconstruct a realistic environment from real-world data, which is why the software is used to generate the scenarios in the environment generated using our pipeline. Some improvements should be made on the integration with ANSYS. For example, we can see that there are no points on the road in Figure 5.17, which is caused by the low intensity resulting from a wrong materials mapping on the road class.

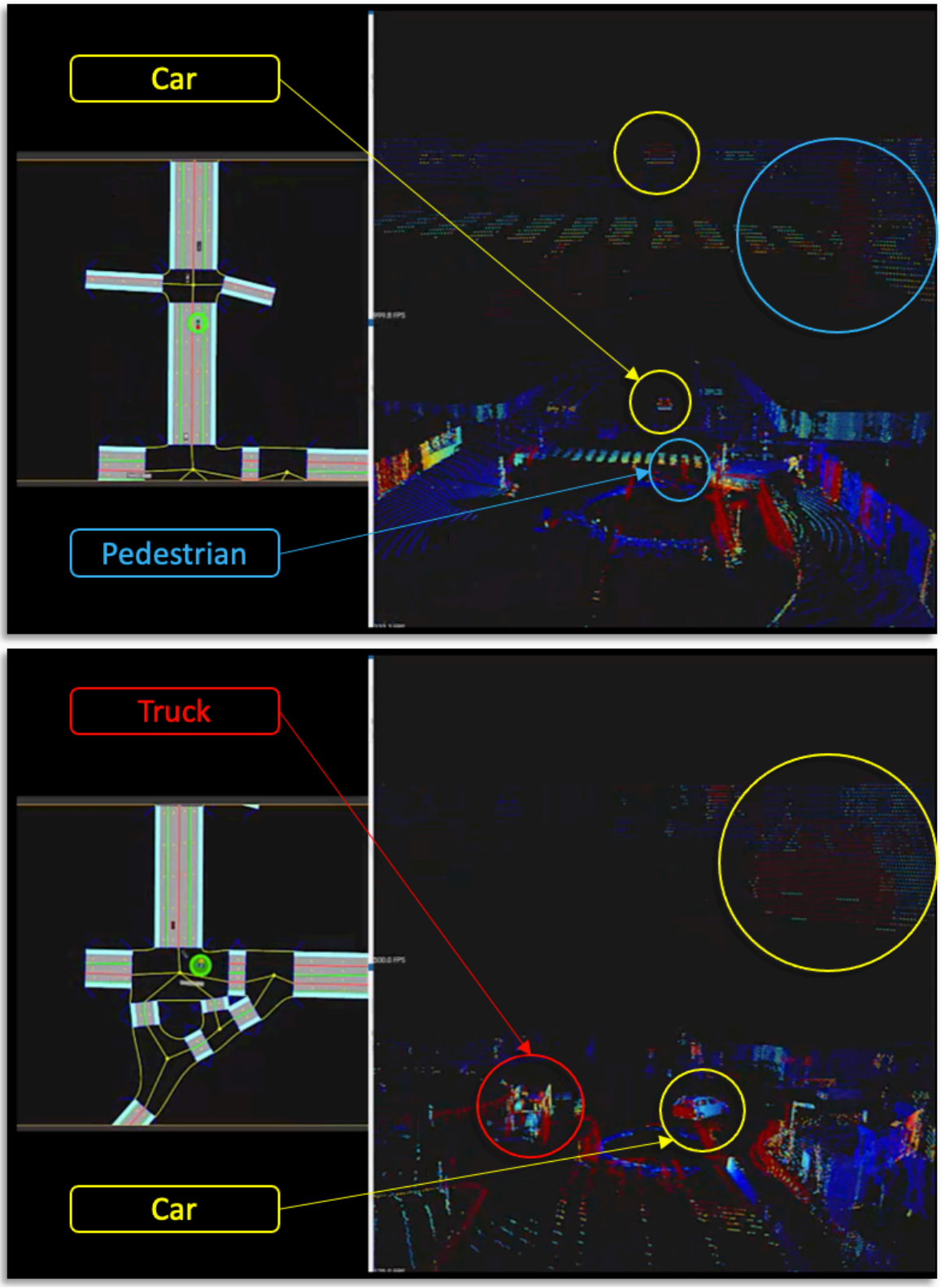


Figure 5.15: Scenarios generation in PC3D-Paris using the integration at ANSYS.

5.5 Conclusion

We have demonstrated how to achieve a faster-than-real-time LiDAR simulation using GPU accelerated ray casting on a variety of datasets collected in diverse environments, using

a variety of sensors mounted in different configurations. Moreover, we validated the accuracy of our LiDAR simulation done in the environment automatically generated using our AdaSplats method by providing qualitative as well as quantitative results, showing that our method achieves the best simulation results. Furthermore, we demonstrated the simulation results on the completed point cloud, using the full simulation pipeline that we propose in this thesis. Finally, we tested the integration of meshed dynamic objects that are added to the final modeled scene and how our work is integrated in ANSYS' SDK. 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.

Chapter 6

Conclusions and Future Works

6.1 Conclusions

The goal of this thesis was to develop a fully automatic reality replication for sensor simulation pipeline that is able to leverage real-world data. As a part of the REPLICA project, our aim is to reduce the cost and time involved in AVs development and testing, while enabling massive sensor simulation, which can be achieved through faster-than-real-time sensor simulation.

As a starting point, a dataset on which the rest of the pipeline relies, was collected using a MMS equipped with a LiDAR sensor and RGB cameras. This dataset contains a real part and a synthetic part, which can be leveraged to augment the dataset. Moreover, the real part was collected in a dense urban area containing a complex geometry and a large number of dynamic objects that were removed using point-wise semantic information.

The acquired point cloud contains many missing regions caused by occlusions from dynamic objects in the scene. To solve this, we introduced unsigned weighted euclidean distance (UWED), compared it to 5 other distance functions and showed that we could obtain the best completion results with our UWED function, while solving the geometry dilation problem resulting from signed distance functions on open surfaces. Moreover, we proposed a point cloud extraction algorithm from a UDF computed on a regular 3D grid.

After Scene Completion (SC), we introduced our 3D scene modeling method, namely AdaSplats, which uses the semantic information obtained from the ground truth annotation, or from deep learning methods, such as KPConv [Thomas et al. 2019]. We could see throughout the experiments that using the ground truth semantic labels or the labels predicted by KPConv, we could obtain similar 3D modeling results, thanks to the ability of our AdaSplats approach to adaptively generate the splats using the semantic information and following the local geometric cues, and to the high accuracy of semantic prediction.

Moreover, we introduced a novel splats-based resampling algorithm that is able to increase the points density in sparse regions and reduce the density in dense regions to increase the level of distribution uniformity throughout the point cloud. We could also see that it results in a lower number of generated splats on the same scene.

Modeling the 3D environment and simulating the different sensors used by AVs is a complex task and is not straightforward to evaluate. In the scene modeling step, we provided qualitative results through rendered images of the different modeled scenes. Even though the high quality rendering that we obtain provide a good basis for camera simulation, we cannot base our evaluation solely on qualitative results. Having said that, we developed a real-time LiDAR simulation method to simulate the LiDAR in the modeled environment and compared

the results of the simulations done in the modeled scenes using our AdaSplats method and other surface reconstruction methods, achieving the best results and validating qualitatively and quantitatively that our pipeline is able to produce the best simulation results.

After comparing our modeling method to other state of the art surface modeling and reconstruction methods, we integrated our method and LiDAR simulation in ANSYS sensor SDK and used SCANeR to generate new scenarios using CAD models of dynamic objects, then we simulated the LiDAR sensor in an environment containing both meshed and splatted geometries, removing the limitation to the scenario of the acquired dataset and opening the possibility of massive simulation, since we obtain faster-than-real-time LiDAR simulation.

Following the above conclusions, we have demonstrated that we can achieve massive LiDAR simulation, which aligns with the aim of the REPLICA project and the goal of the proposed thesis.

6.2 Future Works

Our pipeline is able to generate a high quality surface modeling and is highly flexible, where every step can be improved separately, while positively affecting the quality of the results. We have shown that we could obtain good completion results, however, future experiments can focus on removing large areas resembling cars occlusions and not focus only on removing scans for completion, which might result in better completion on areas occluded by large dynamic objects.

Another improvement would be to increase the rendering quality, which can be achieved by performing proper texture mapping instead of using only point-wise color information. Another method is to combine semantic splats with point-based neural rendering [Xu et al. 2022; Bui et al. 2018] methods.

Generating new scenarios for test cases can be achieved through adding dynamic objects on top of the splatted scene. An improvement would be to use real-world dynamic object models in the form of point clouds, which can be accurately modeled and added to the modeled scene instead of CAD models.

Ray tracing for rendering and LiDAR simulation can be accelerated even further with the use of a level-of-detail (LOD) structure, which can be used to reduce the detailed representation of a scene beyond a given distance threshold, reducing the ray-primitive intersection time.

Bibliography

Alexa, M., Behr, J., Cohen-Or, D., Fleishman, S., Levin, D., and Silva, C.T. (2003). "Computing and rendering point set surfaces". In: *IEEE Transactions on Visualization and Computer Graphics* 9.1, pp. 3–15. DOI: 10.1109/TVCG.2003.1175093.

Aliev, Kara-Ali, Sevastopolsky, Artem, Kolos, Maria, Ulyanov, Dmitry, and Lempitsky, Victor (2020). "Neural Point-Based Graphics". In: *Computer Vision – ECCV 2020*. Ed. by Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm. Cham: Springer International Publishing, pp. 696–712. ISBN: 978-3-030-58542-6.

Armeni, Iro, Sener, Ozan, Zamir, Amir R., Jiang, Helen, Brilakis, Ioannis, Fischer, Martin, and Savarese, Silvio (2016). "3D Semantic Parsing of Large-Scale Indoor Spaces". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1534–1543. DOI: 10.1109/CVPR.2016.170.

Atzmon, Matan and Lipman, Yaron (2020). "SAL: Sign Agnostic Learning of Shapes From Raw Data". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2562–2571. DOI: 10.1109/CVPR42600.2020.00264.

Behley, Jens, Garbade, Martin, Milioto, Andres, Quenzel, Jan, Behnke, Sven, Stachniss, Cyrill, and Gall, Jürgen (2019). "SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 9296–9306. DOI: 10.1109/ICCV.2019.00939.

Botsch, M., Hornung, A., Zwicker, M., and Kobbelt, L. (2005). "High-quality surface splatting on today's GPUs". In: *Proceedings Eurographics/IEEE VGTC Symposium Point-Based Graphics, 2005.* Pp. 17–141. DOI: 10.1109/PBG.2005.194059.

Botsch, Mario and Kobbelt, Leif (2003). "High-Quality Point-Based Rendering on Modern GPUs". In: *Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*. PG '03. USA: IEEE Computer Society, p. 335. ISBN: 0769520286.

Botsch, Mario, Spernat, Michael, and Kobbelt, Leif (2004). "Phong Splatting". In: *Proceedings of the First Eurographics Conference on Point-Based Graphics*. SPBG'04. Switzerland: Eurographics Association, pp. 25–32. ISBN: 3905673096.

Boulch, Alexandre and Marlet, Renaud (2012). "Fast and Robust Normal Estimation for Point Clouds with Sharp Features". In: *Computer Graphics Forum* 31.5, pp. 1765–1774. DOI: https://doi.org/10.1111/j.1467-8659.2012.03181.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2012.03181.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2012.03181.x.

Boulch, Alexandre, Puy, Gilles, and Marlet, Renaud (2020). "FKAConv: Feature-Kernel Alignment for Point Cloud Convolutions". In: *15th Asian Conference on Computer Vision (ACCV 2020)*.

Boussaha, Mohamed, Fernandez-Moral, Eduardo, Vallet, Bruno, and Rives, Patrick (2018). "On the production of semantic and textured 3d meshes of large scale urban environments from mobile mapping images and lidar scans". In: *RFIAP 2018, Reconnaissance des Formes, Image, Apprentissage et Perception*.

Bui, Giang, Le, Truc, Morago, Brittany, and Duan, Ye (June 2018). "Point-Based Rendering Enhancement via Deep Learning". In: *Vis. Comput.* 34.6–8, pp. 829–841. ISSN: 0178-2789.

DOI: 10.1007/s00371-018-1550-6. URL: https://doi.org/10.1007/s00371-018-1550-6.

Caesar, Holger, Bankiti, Varun, Lang, Alex H., Vora, Sourabh, Liong, Venice Erin, Xu, Qiang, Krishnan, Anush, Pan, Yu, Baldan, Giancarlo, and Beijbom, Oscar (2020). "nuScenes: A Multimodal Dataset for Autonomous Driving". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11618–11628. DOI: 10.1109/CVPR42600.2020.01164.

Campos, Carlos, Elvira, Richard, Rodríguez, Juan J Gómez, Montiel, José MM, and Tardós, Juan D (2021). "Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam". In: *IEEE Transactions on Robotics* 37.6, pp. 1874–1890.

Caraffa, Laurent, Brédif, Mathieu, and Vallet, Bruno (2015). "3D OCTREE BASED WATERTIGHT MESH GENERATION FROM UBIQUITOUS DATA". In: *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, pp. 613–617.

Caraffa, Laurent, Brédif, Mathieu, and Vallet, Bruno (2017). "3D Watertight Mesh Generation with Uncertainties from Ubiquitous Data". In: *Computer Vision – ACCV 2016*. Ed. by Shang-Hong Lai, Vincent Lepetit, Ko Nishino, and Yoichi Sato. Cham: Springer International Publishing, pp. 377–391. ISBN: 978-3-319-54190-7.

Caraffa, Laurent, Marchand, Yanis, Brédif, Mathieu, and Vallet, Bruno (2021). "Efficiently Distributed Watertight Surface Reconstruction". In: *2021 International Conference on 3D Vision (3DV)*, pp. 1432–1441. DOI: 10.1109/3DV53792.2021.00150.

Chang, Angel, Dai, Angela, Funkhouser, Thomas, Halber, Maciej, Niebner, Matthias, Savva, Manolis, Song, Shuran, Zeng, Andy, and Zhang, Yinda (2017a). "Matterport3D: Learning from RGB-D Data in Indoor Environments". In: *2017 International Conference on 3D Vision (3DV)*, pp. 667–676. DOI: 10.1109/3DV.2017.00081.

Chang, Angel, Dai, Angela, Funkhouser, Thomas, Halber, Maciej, Niebner, Matthias, Savva, Manolis, Song, Shuran, Zeng, Andy, and Zhang, Yinda (2017b). "Matterport3D: Learning from RGB-D Data in Indoor Environments". In: *2017 International Conference on 3D Vision (3DV)*, pp. 667–676. DOI: 10.1109/3DV.2017.00081.

Chen, Rong, Huang, Zhiyong, and Yu, Yuanlong (2019a). "AM2FNet: Attention-Based Multiscale and Multi-Modality Fused Network". In: *2019 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. Dali, China: IEEE Press, pp. 1192–1197. DOI: 10.1109/ROBIO49542.2019.8961556. URL: https://doi.org/10.1109/ROBIO49542.2019.8961556.

Chen, Yueh-Tung, Garbade, Martin, and Gall, Juergen (2019b). "3D Semantic Scene Completion from a Single Depth Image Using Adversarial Training". In: *2019 IEEE International Conference on Image Processing (ICIP)*, pp. 1835–1839. DOI: 10.1109/ICIP.2019.8803174.

Chen, Yueh-Tung, Garbade, Martin, and Gall, Juergen (2019c). "3D Semantic Scene Completion from a Single Depth Image Using Adversarial Training". In: *2019 IEEE International Conference on Image Processing (ICIP)*, pp. 1835–1839. DOI: 10.1109/ICIP.2019.8803174.

Chen, Yun, Yang, Bin, Liang, Ming, and Urtasun, Raquel (2019d). "Learning Joint 2D-3D Representations for Depth Completion". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 10022–10031. DOI: 10.1109/ICCV.2019.01012.

Chen, Zhiqin and Zhang, Hao (2019e). "Learning Implicit Fields for Generative Shape Modeling". In: *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Chen, Zhonggui, Zhang, Tieyi, Cao, Juan, Zhang, Yongjie Jessica, and Wang, Cheng (2018). "Point cloud resampling using centroidal Voronoi tessellation methods". In: *Computer-Aided Design* 102. Proceeding of SPM 2018 Symposium, pp. 12–21. ISSN: 0010-4485. DOI:

https://doi.org/10.1016/j.cad.2018.04.010. URL: https://www.sciencedirect.com/science/article/pii/S0010448518302197.

Cheng, Ran, Agia, Christopher, Ren, Yuan, Li, Xinhai, and Liu, Bingbing (2020). "S3CNet: A Sparse Semantic Scene Completion Network for LiDAR Point Clouds". In: *CoRR* abs/2012.09242. arXiv: 2012.09242. URL: https://arxiv.org/abs/2012.09242.

Chibane, Julian, Alldieck, Thiemo, and Pons-Moll, Gerard (June 2020a). "Implicit Functions in Feature Space for 3D Shape Reconstruction and Completion". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE.

Chibane, Julian, Mir, Aymen, and Pons-Moll, Gerard (Dec. 2020b). "Neural Unsigned Distance Fields for Implicit Function Learning". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Choy, Christopher, Gwak, JunYoung, and Savarese, Silvio (2019). "4D Spatio-Temporal ConvNets: Minkowski Convolutional Neural Networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3075–3084.

CloudCompare (version 2.11.1) [Anoia] (2021). http://www.cloudcompare.org/.

Congote, John, Moreno, Aitor, Barandiaran, Iñigo, Barandiarán, Javier, Posada, Jorge, and Ruiz, Oscar E. (2010). "Marching Cubes in an Unsigned Distance Field for Surface Reconstruction from Unorganized Point Sets". In: *GRAPP*.

Curless, Brian and Levoy, Marc (1996). "A Volumetric Method for Building Complex Models from Range Images". In: *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '96. New York, NY, USA: Association for Computing Machinery, pp. 303–312. ISBN: 0897917464. DOI: 10.1145/237170.237269. URL: https://doi.org/10.1145/237170.237269.

Dai, Angela, Chang, Angel X., Savva, Manolis, Halber, Maciej, Funkhouser, Thomas, and Nießner, Matthias (2017a). "ScanNet: Richly-Annotated 3D Reconstructions of Indoor Scenes". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2432–2443. DOI: 10.1109/CVPR.2017.261.

Dai, Angela, Diller, Christian, and Niessner, Matthias (2020). "SG-NN: Sparse Generative Neural Networks for Self-Supervised Scene Completion of RGB-D Scans". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 846–855. DOI: 10.1109/CVPR42600.2020.00093.

Dai, Angela, Qi, Charles Ruizhongtai, and Nießner, Matthias (2017b). "Shape Completion Using 3D-Encoder-Predictor CNNs and Shape Synthesis". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6545–6554. DOI: 10.1109/CVPR.2017.693.

Dai, Angela, Ritchie, Daniel, Bokeloh, Martin, Reed, Scott, Sturm, Jürgen, and Nießner, Matthias (2018). "ScanComplete: Large-Scale Scene Completion and Semantic Segmentation for 3D Scans". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4578–4587. DOI: 10.1109/CVPR.2018.00481.

Dai, Angela, Siddiqui, Yawar, Thies, Justus, Valentin, Julien, and Nießner, Matthias (2021). "SPSG: Self-Supervised Photometric Scene Generation from RGB-D Scans". In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1747–1756. DOI: 10.1109/CVPR46437.2021.00179.

Demantke, J., Vallet, Bruno, and Paparoditis, Nicolas (Oct. 2013). "Facade Reconstruction with Generalized 2.5d Grids". In: *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* II-5/W2, pp. 67–72. DOI: 10.5194/isprsannals-II-5-W2-67-2013.

Deschaud, Jean-Emmanuel (2018). "IMLS-SLAM: Scan-to-Model Matching Based on 3D Data". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2480–2485. DOI: 10.1109/ICRA.2018.8460653.

Deschaud, Jean-Emmanuel (2021). "KITTI-CARLA: a KITTI-like dataset generated by CARLA Simulator". In: *arXiv e-prints*, arXiv:2109.00892.

Deschaud, Jean-Emmanuel, Duque, David, Richa, Jean Pierre, Velasco-Forero, Santiago, Marcotegui, Beatriz, and Goulette, François (2021). "Paris-CARLA-3D: A Real and Synthetic Outdoor Point Cloud Dataset for Challenging Tasks in 3D Mapping". In: *Remote Sensing* 13.22. ISSN: 2072-4292. DOI: 10.3390/rs13224713. URL: https://www.mdpi.com/2072-4292/13/22/4713.

Deschaud, Jean-Emmanuel, Prasser, David, Dias, M. Freddie, Browning, Brett, and Rander, Peter (2012). "Automatic data driven vegetation modeling for lidar simulation". In: *2012 IEEE International Conference on Robotics and Automation*, pp. 5030–5036. DOI: 10.1109/ICRA.2012.6225269.

Devaux, Alexandre and Brédif, Mathieu (2016). "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. Anaheim, California: Association for Computing Machinery, pp. 105–108. ISBN: 9781450344289. DOI: 10.1145/2945292.2945311. URL: https://doi.org/10.1145/2945292.2945311.

Dey, T.K., Li, G., and Sun, J. (2005). "Normal estimation for point clouds: a comparison study for a Voronoi based method". In: *Proceedings Eurographics/IEEE VGTC Symposium Point-Based Graphics, 2005.* Pp. 39–46. DOI: 10.1109/PBG.2005.194062.

Dosovitskiy, Alexey, Ros, German, Codevilla, Felipe, Lopez, Antonio, and Koltun, Vladlen (2017a). "CARLA: An Open Urban Driving Simulator". In: *Proceedings of the 1st Annual Conference on Robot Learning*, pp. 1–16.

Dosovitskiy, Alexey, Ros, German, Codevilla, Felipe, Lopez, Antonio, and Koltun, Vladlen (Nov. 2017b). "CARLA: An Open Urban Driving Simulator". In: *Proceedings of the 1st Annual Conference on Robot Learning*. Ed. by Sergey Levine, Vincent Vanhoucke, and Ken Goldberg. Vol. 78. Proceedings of Machine Learning Research. PMLR, pp. 1–16. URL: https://proceedings.mlr.press/v78/dosovitskiy17a.html.

Duque-Arias, David (Oct. 2021). "3D urban scene understanding by analysis of LiDAR, color and hyperspectral data". Theses. Université Paris sciences et lettres. URL: https://pastel.archives-ouvertes.fr/tel-03434199.

Duque-Arias, David, Velasco-Forero, Santiago, Deschaud, Jean-Emmanuel, Goulette, François, Serna, Andrés, Decencière, Etienne, and Marcotegui, Beatriz (2021). "On Power Jaccard Losses for Semantic Segmentation". In: *VISIGRAPP*.

Fang, Jin, Zhou, Dingfu, Yan, Feilong, Zhao, Tongtong, Zhang, Feihu, Ma, Yu, Wang, Liang, and Yang, Ruigang (2020). "Augmented LiDAR Simulator for Autonomous Driving". In: *IEEE Robotics and Automation Letters* 5.2, pp. 1931–1938. DOI: 10.1109/LRA.2020.2969927.

Fridovich-Keil, Sara, Yu, Alex, Tancik, Matthew, Chen, Qinhong, Recht, Benjamin, and Kanazawa, Angjoo (2022). "Plenoxels: Radiance Fields without Neural Networks". In: *CVPR*.

Fuhrmann, Simon and Goesele, Michael (July 2014). "Floating Scale Surface Reconstruction". In: *ACM Trans. Graph.* 33.4. ISSN: 0730-0301. DOI: 10.1145/2601097.2601163. URL: https://doi.org/10.1145/2601097.2601163.

Garbin, Stephan J, Kowalski, Marek, Johnson, Matthew, Shotton, Jamie, and Valentin, Julien (2021). "Fastnerf: High-fidelity neural rendering at 200fps". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 14346–14355.

Geyer, Jakob, Kassahun, Yohannes, Mahmudi, Mentar, Ricou, Xavier, Durgesh, Rupesh, Chung, Andrew S., Hauswald, Lorenz, Pham, Viet Hoang, Mühlegg, Maximilian, Dorn, Sebastian, Fernandez, Tiffany, Jänicke, Martin, Mirashi, Sudesh, Savani, Chiragkumar, Sturm, Martin, Vorobiov, Oleksandr, Oelker, Martin, Garreis, Sebastian, and Schuberth,

Peter (2020). "A2D2: Audi Autonomous Driving Dataset". In: *arXiv preprint*. arXiv: 2004.06320 [cs.CV]. URL: https://www.a2d2.audi.

Gomes, Leonardo, Regina Pereira Bellon, Olga, and Silva, Luciano (2014). "3D reconstruction methods for digital preservation of cultural heritage: A survey". In: *Pattern Recognition Letters* 50. Depth Image Analysis, pp. 3–14. ISSN: 0167-8655. DOI: https://doi.org/10.1016/j.patrec.2014.03.023. URL: https://www.sciencedirect.com/science/article/pii/S0167865514001032.

Goulette, F, Nashashibi, F, Abuhadrous, I, Ammoun, S, and Laurgeau, C (2006). "AN INTEGRATED ON-BOARD LASER RANGE SENSING SYSTEM FOR ON-THE-WAY CITY AND ROAD MODELLING". In: *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 34.A. URL: https://hal.archives-ouvertes.fr/hal-01259660.

Graham, Benjamin and Maaten, Laurens van der (2017). *Submanifold Sparse Convolutional Networks*. arXiv: 1706.01307 [cs.NE].

Griffiths, David and Boehm, Jan (2019). "SynthCity: A large scale synthetic point cloud". In: *arXiv preprint arXiv:1907.04758*.

Gschwandtner, Michael, Kwitt, Roland, Uhl, Andreas, and Pree, Wolfgang (2011). "BlenSor: Blender Sensor Simulation Toolbox". In: *Advances in Visual Computing*. Ed. by 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. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 199–208. ISBN: 978-3-642-24031-7.

Guillard, Benoit, Vemprala, Sai, Gupta, Jayesh K, Miksik, Ondrej, Vineet, Vibhav, Fua, Pascal, and Kapoor, Ashish (2022). "Learning to Simulate Realistic LiDARs". In: *arXiv preprint arXiv:2209.10986*.

Günther, Christian, Kanzok, Thomas, Linsen, Lars, and Rosenthal, Paul (2013). "A GPGPU-based pipeline for accelerated rendering of point clouds". In.

Guo, Xiaoyuan, Xiao, Jun, and Wang, Ying (2018a). "A Survey on Algorithms of Hole Filling in 3D Surface Reconstruction". In: *Vis. Comput.* 34.1, pp. 93–103. ISSN: 0178-2789. DOI: 10.1007/s00371-016-1316-y. URL: https://doi.org/10.1007/s00371-016-1316-y.

Guo, Yuxiao and Tong, Xin (July 2018b). "View-Volume Network for Semantic Scene Completion from a Single Depth Image". In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*. International Joint Conferences on Artificial Intelligence Organization, pp. 726–732. DOI: 10.24963/ijcai.2018/101. URL: https://doi.org/10.24963/ijcai.2018/101.

Gwak, JunYoung, Choy, Christopher, and Savarese, Silvio (2020). "Generative Sparse Detection Networks for 3D Single-Shot Object Detection". In: *Computer Vision – ECCV 2020*. Ed. by Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm. Cham: Springer International Publishing, pp. 297–313. ISBN: 978-3-030-58548-8.

Hackel, Timo, Savinov, N., Ladicky, L., Wegner, Jan D., Schindler, K., and Pollefeys, M. (2017). "SEMANTIC3D.NET: A new large-scale point cloud classification benchmark". In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*. Vol. IV-1-W1, pp. 91–98.

Hedman, Peter, Srinivasan, Pratul P., Mildenhall, Ben, Barron, Jonathan T., and Debevec, Paul E. (2021). "Baking Neural Radiance Fields for Real-Time View Synthesis". In: *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 5855–5864.

Hoppe, Hugues, DeRose, Tony, Duchamp, Tom, McDonald, John, and Stuetzle, Werner (1992). "Surface Reconstruction from Unorganized Points". In: *Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '92. New York, NY, USA: Association for Computing Machinery, pp. 71–78. ISBN: 0897914791. DOI: 10.1145/133994.134011. URL: https://doi.org/10.1145/133994.134011.

Hu, Qingyong, Yang, Bo, Khalid, Sheikh, Xiao, Wen, Trigoni, Niki, and Markham, Andrew (June 2021). "Towards Semantic Segmentation of Urban-Scale 3D Point Clouds: A Dataset, Benchmarks and Challenges". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4977–4987.

Hurl, Braden, Czarnecki, Krzysztof, and Waslander, Steven (2019a). "Precise Synthetic Image and LiDAR (PreSIL) Dataset for Autonomous Vehicle Perception". In: *2019 IEEE Intelligent Vehicles Symposium (IV)*, pp. 2522–2529. DOI: 10.1109/IVS.2019.8813809.

Hurl, Braden, Czarnecki, Krzysztof, and Waslander, Steven (2019b). "Precise Synthetic Image and LiDAR (PreSIL) Dataset for Autonomous Vehicle Perception". In: *2019 IEEE Intelligent Vehicles Symposium (IV)*, pp. 2522–2529. DOI: 10.1109/IVS.2019.8813809.

Kang, Lai, Jiang, Jie, Wei, Yingmei, and Xie, Yuxiang (2019). "Efficient Randomized Hierarchy Construction for Interactive Visualization of Large Scale Point Clouds". In: *2019 IEEE Fourth International Conference on Data Science in Cyberspace (DSC)*, pp. 593–597. DOI: 10.1109/DSC.2019.00096.

Kazhdan, Michael, Bolitho, Matthew, and Hoppe, Hugues (2006). "Poisson Surface Reconstruction". In: *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*. SGP '06. Cagliari, Sardinia, Italy: Eurographics Association, pp. 61–70. ISBN: 3905673363.

Kazhdan, Michael and Hoppe, Hugues (July 2013). "Screened Poisson Surface Reconstruction". In: *ACM Trans. Graph.* 32.3. ISSN: 0730-0301. DOI: 10.1145/2487228.2487237. URL: https://doi.org/10.1145/2487228.2487237.

Kolluri, Ravikrishna (May 2008). "Provably Good Moving Least Squares". In: *ACM Trans. Algorithms* 4.2. ISSN: 1549-6325. DOI: 10.1145/1361192.1361195. URL: https://doi.org/10.1145/1361192.1361195.

Ku, Tao, Veltkamp, Remco C., Boom, Bas, Duque-Arias, David, Velasco-Forero, Santiago, Deschaud, Jean-Emmanuel, Goulette, François, Marcotegui, Beatriz, Ortega, Sebastián, Trujillo, Agustín, Suárez, José Pablo, Santana, José Miguel, Ramírez, Cristian, Akadas, Kiran, and Gangisetty, Shankar (2020). "SHREC 2020: 3D point cloud semantic segmentation for street scenes". In: *Comput. Graph.* 93, pp. 13–24.

Labatut, P., Pons, J.-P., and Keriven, R. (2009). "Robust and Efficient Surface Reconstruction From Range Data". In: *Computer Graphics Forum* 28.8, pp. 2275–2290. DOI: https://doi.org/10.1111/j.1467-8659.2009.01530.x.

Landrieu, Loic and Simonovsky, Martin (2018). "Large-Scale Point Cloud Semantic Segmentation with Superpoint Graphs". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4558–4567. DOI: 10.1109/CVPR.2018.00479.

Levoy, Marc and Whitted, Turner (2000). "The Use of Points as a Display Primitive". In.

Li, Jie, Liu, Yu, Gong, Dong, Shi, Qinfeng, Yuan, Xia, Zhao, Chunxia, and Reid, Ian D. (2019). "RGBD Based Dimensional Decomposition Residual Network for 3D Semantic Scene Completion". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7685–7694.

Li, Xinke, Li, Chongshou, Tong, Zekun, Lim, Andrew, Yuan, Junsong, Wu, Yuwei, Tang, Jing, and Huang, Raymond (2020). "Campus3D: A Photogrammetry Point Cloud Benchmark for Hierarchical Understanding of Outdoor Scene". In: *Proceedings of the 28th ACM International Conference on Multimedia*. MM '20. Seattle, WA, USA: Association for Computing Machinery, pp. 238–246. ISBN: 9781450379885. DOI: 10.1145/3394171.3413661. URL: https://doi.org/10.1145/3394171.3413661.

Liao, Yiyi, Donné, Simon, and Geiger, Andreas (2018). "Deep Marching Cubes: Learning Explicit Surface Representations". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2916–2925. DOI: 10.1109/CVPR.2018.00308.

Linsen, Lars, Müller, Karsten, and Rosenthal, Paul (2008). "Splat-based Ray Tracing of Point Clouds". In: *Journal of WSCG*, pp. 51–58.

Liu, Lingjie, Gu, Jiatao, Lin, Kyaw Zaw, Chua, Tat-Seng, and Theobalt, Christian (2020). "Neural Sparse Voxel Fields". In: *NeurIPS*.

Lorensen, William E. and Cline, Harvey E. (Aug. 1987). "Marching Cubes: A High Resolution 3D Surface Construction Algorithm". In: *SIGGRAPH Comput. Graph.* 21.4, pp. 163–169. ISSN: 0097-8930. DOI: 10.1145/37402.37422. URL: https://doi.org/10.1145/37402.37422.

Maglo, Adrien, Lavoué, Guillaume, Dupont, Florent, and Hudelot, Céline (Feb. 2015). "3D Mesh Compression: Survey, Comparisons, and Emerging Trends". In: *ACM Comput. Surv.* 47.3. ISSN: 0360-0300. DOI: 10.1145/2693443. URL: https://doi.org/10.1145/2693443.

Manivasagam, Sivabalan, Wang, Shenlong, Wong, Kelvin, Zeng, Wenyuan, Sazanovich, Mikita, Tan, Shuhan, Yang, Bin, Ma, Wei-Chiu, and Urtasun, Raquel (2020). "LiDARsim: Realistic LiDAR Simulation by Leveraging the Real World". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11164–11173. DOI: 10.1109/CVPR42600.2020.01118.

Marchand, Yanis, Vallet, Bruno, and Caraffa, Laurent (2021). "Evaluating Surface Mesh Reconstruction of Open Scenes". In: *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences* 43, pp. 369–376.

McCormac, John, Handa, Ankur, Leutenegger, Stefan, and Davison, Andrew J. (2017). "SceneNet RGB-D: Can 5M Synthetic Images Beat Generic ImageNet Pre-training on Indoor Segmentation?" In: *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 2697–2706. DOI: 10.1109/ICCV.2017.292.

Mescheder, Lars, Oechsle, Michael, Niemeyer, Michael, Nowozin, Sebastian, and Geiger, Andreas (2019a). "Occupancy Networks: Learning 3D Reconstruction in Function Space". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4455–4465. DOI: 10.1109/CVPR.2019.00459.

Mescheder, Lars, Oechsle, Michael, Niemeyer, Michael, Nowozin, Sebastian, and Geiger, Andreas (2019b). "Occupancy Networks: Learning 3D Reconstruction in Function Space". In: *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.

Metzer, Gal, Hanocka, Rana, Zorin, Denis, Giryes, Raja, Panozzo, Daniele, and Cohen-Or, Daniel (July 2021). "Orienting Point Clouds with Dipole Propagation". In: *ACM Trans. Graph.* 40.4. ISSN: 0730-0301. DOI: 10.1145/3450626.3459835. URL: https://doi.org/10.1145/3450626.3459835.

Mildenhall, Ben, Srinivasan, Pratul P., Tancik, Matthew, Barron, Jonathan T., Ramamoorthi, Ravi, and Ng, Ren (2020). "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis". In: *ECCV*.

Mitra, Niloy J. and Nguyen, An (2003). "Estimating Surface Normals in Noisy Point Cloud Data". In: *Proceedings of the Nineteenth Annual Symposium on Computational Geometry*. SCG '03. San Diego, California, USA: Association for Computing Machinery, pp. 322–328. ISBN: 1581136633. DOI: 10.1145/777792.777840. URL: https://doi.org/10.1145/777792.777840.

Mullen, Patrick, Goes, Fernando, Desbrun, Mathieu, Cohen-Steiner, David, and Alliez, Pierre (July 2010). "Signing the Unsigned: Robust Surface Reconstruction from Raw Pointsets". In: *Computer Graphics Forum* 29. DOI: 10.1111/j.1467-8659.2010.01782.x.

Munoz, Daniel, Bagnell, J. Andrew, Vandapel, Nicolas, and Hebert, Martial (2009). "Contextual classification with functional Max-Margin Markov Networks". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 975–982. DOI: 10.1109/CVPR.2009.5206590.

Newcombe, Richard A., Izadi, Shahram, Hilliges, Otmar, Molyneaux, David, Kim, David, Davison, Andrew J., Kohi, Pushmeet, Shotton, Jamie, Hodges, Steve, and Fitzgibbon, Andrew (2011). "KinectFusion: Real-time dense surface mapping and tracking". In: *2011*

10th IEEE International Symposium on Mixed and Augmented Reality, pp. 127–136. DOI: 10.1109/ISMAR.2011.6092378.

Oleynikova, Helen, Taylor, Zachary, Fehr, Marius, Siegwart, Roland, and Nieto, Juan (2017). "Voxblox: Incremental 3D Euclidean Signed Distance Fields for on-board MAV planning". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1366–1373. DOI: 10.1109/IROS.2017.8202315.

Öztireli, A. C., Guennebaud, G., and Gross, M. (2009). "Feature Preserving Point Set Surfaces based on Non-Linear Kernel Regression". In: *Computer Graphics Forum* 28.2, pp. 493–501. DOI: https://doi.org/10.1111/j.1467-8659.2009.01388.x.

Pagés, Rafael, García, Sergio, Berjón, Daniel, and Morán, Francisco (2015). "SPLASH: A Hybrid 3D Modeling/Rendering Approach Mixing Splats and Meshes". In: *Proceedings of the 20th International Conference on 3D Web Technology*. Web3D '15. Heraklion, Crete, Greece: Association for Computing Machinery, pp. 231–234. ISBN: 9781450336475. DOI: 10.1145/2775292.2775320. URL: https://doi.org/10.1145/2775292.2775320.

Pan, Liang, Chen, Xinyi, Cai, Zhongang, Zhang, Junzhe, Zhao, Haiyu, Yi, Shuai, and Liu, Ziwei (2021). "Variational Relational Point Completion Network". In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8520–8529. DOI: 10.1109/CVPR46437.2021.00842.

Pan, Yancheng, Gao, Biao, Mei, Jilin, Geng, Sibo, Li, Chengkun, and Zhao, Huijing (2020). "SemanticPOSS: A Point Cloud Dataset with Large Quantity of Dynamic Instances". In: *2020 IEEE Intelligent Vehicles Symposium (IV)*, pp. 687–693. DOI: 10.1109/IV47402.2020.9304596.

Park, Jeong Joon, Florence, Peter, Straub, Julian, Newcombe, Richard, and Lovegrove, Steven (2019). "DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 165–174. DOI: 10.1109/CVPR.2019.00025.

Parker, Steven G., Bigler, James, Dietrich, Andreas, Friedrich, Heiko, Hoberock, Jared, Luebke, David, McAllister, David, McGuire, Morgan, Morley, Keith, Robison, Austin, and Stich, Martin (July 2010). "OptiX: A General Purpose Ray Tracing Engine". In: *ACM Trans. Graph.* 29.4. ISSN: 0730-0301. DOI: 10.1145/1778765.1778803. URL: https://doi.org/10.1145/1778765.1778803.

Peng, Songyou, Niemeyer, Michael, Mescheder, Lars, Pollefeys, Marc, and Geiger, Andreas (2020). "Convolutional Occupancy Networks". In: *Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III*. Glasgow, United Kingdom: Springer-Verlag, pp. 523–540. ISBN: 978-3-030-58579-2.

Pfister, Hanspeter, Zwicker, Matthias, Baar, Jeroen van, and Gross, Markus (2000). "Surfels: Surface Elements as Rendering Primitives". In: *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '00. USA: ACM Press/Addison-Wesley Publishing Co., pp. 335–342. ISBN: 1581132085. DOI: 10.1145/344779.344936. URL: https://doi.org/10.1145/344779.344936.

Pomerleau, Dean A. (1988). "ALVINN: An Autonomous Land Vehicle in a Neural Network". In: *Advances in Neural Information Processing Systems*. Ed. by D. Touretzky. Vol. 1. Morgan-Kaufmann. URL: https://proceedings.neurips.cc/paper/1988/file/812b4ba287f5ee0bc9d43bbf5bbe87fb-Paper.pdf.

Rebain, Daniel, Jiang, Wei, Yazdani, Soroosh, Li, Ke, Yi, Kwang Moo, and Tagliasacchi, Andrea (2021). "DeRF: Decomposed Radiance Fields". In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 14148–14156.

Reiser, C., Peng, S., Liao, Y., and Geiger, A. (Oct. 2021). "KiloNeRF: Speeding up Neural Radiance Fields with Thousands of Tiny MLPs". In: *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. Los Alamitos, CA, USA: IEEE Computer So-

ciety, pp. 14315–14325. DOI: 10.1109/ICCV48922.2021.01407. URL: https://doi.ieeecomputersociety.org/10.1109/ICCV48922.2021.01407.

Roldao, Luis, Charette, Raoul de, and Verroust-Blondet, Anne (2021). "3D Semantic Scene Completion: a Survey". In: *CoRR* abs/2103.07466. arXiv: 2103.07466. URL: https://arxiv.org/abs/2103.07466.

Roldão, Luis, Charette, Raoul de, and Verroust-Blondet, Anne (2020). "LMSCNet: Lightweight Multiscale 3D Semantic Completion". In: *2020 International Conference on 3D Vision (3DV)*, pp. 111–119. DOI: 10.1109/3DV50981.2020.00021.

Roynard, Xavier, Deschaud, Jean-Emmanuel, and Goulette, François (2018a). "Paris-Lille-3D: A large and high-quality ground-truth urban point cloud dataset for automatic segmentation and classification". In: *The International Journal of Robotics Research* 37.6, pp. 545–557. DOI: 10.1177/0278364918767506. eprint: https://doi.org/10.1177/0278364918767506. URL: https://doi.org/10.1177/0278364918767506.

Roynard, Xavier, Deschaud, Jean-Emmanuel, and Goulette, François (2018b). "Paris-Lille-3D: A large and high-quality ground-truth urban point cloud dataset for automatic segmentation and classification". In: *The International Journal of Robotics Research* 37.6, pp. 545–557. DOI: 10.1177/0278364918767506. eprint: https://doi.org/10.1177/0278364918767506. URL: https://doi.org/10.1177/0278364918767506.

Rusinkiewicz, Szymon and Levoy, Marc (2000). "QSplat: A Multiresolution Point Rendering System for Large Meshes". In: *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '00. USA: ACM Press/Addison-Wesley Publishing Co., pp. 343–352. ISBN: 1581132085. DOI: 10.1145/344779.344940. URL: https://doi.org/10.1145/344779.344940.

Schertler, Nico, Savchynskyy, Bogdan, and Gumhold, Stefan (Jan. 2017). "Towards Globally Optimal Normal Orientations for Large Point Clouds". In: *Comput. Graph. Forum* 36.1, pp. 197–208. ISSN: 0167-7055. DOI: 10.1111/cgf.12795. URL: https://doi.org/10.1111/cgf.12795.

Schütz, Markus, Kerbl, Bernhard, and Wimmer, Michael (2021). "Rendering Point Clouds with Compute Shaders and Vertex Order Optimization". In: *Computer Graphics Forum* 40.

Schütz, Markus, Kerbl, Bernhard, and Wimmer, Michael (July 2022). "Software Rasterization of 2 Billion Points in Real Time". In: *Proc. ACM Comput. Graph. Interact. Tech.* 5.3. DOI: 10.1145/3543863. URL: https://doi.org/10.1145/3543863.

Serna, Andrés, Marcotegui, Beatriz, Goulette, François, and Deschaud, Jean-Emmanuel (2014). "Paris-rue-Madame database: a 3D mobile laser scanner dataset for benchmarking urban detection, segmentation and classification methods". In: *4th International Conference on Pattern Recognition, Applications and Methods ICPRAM 2014*. Angers, France. URL: https://hal.archives-ouvertes.fr/hal-00963812.

Silberman, Nathan, Hoiem, Derek, Kohli, Pushmeet, and Fergus, Rob (2012). "Indoor Segmentation and Support Inference from RGBD Images". In: *Computer Vision – ECCV 2012*. Ed. by Andrew Fitzgibbon, Svetlana Lazebnik, Pietro Perona, Yoichi Sato, and Cordelia Schmid. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 746–760.

Soheilian, Bahman, Tournaire, Olivier, Paparoditis, Nicolas, Vallet, Bruno, and Papelard, Jean-Pierre (2013). "Generation of an integrated 3D city model with visual landmarks for autonomous navigation in dense urban areas". In: *2013 IEEE Intelligent Vehicles Symposium (IV)*, pp. 304–309. DOI: 10.1109/IVS.2013.6629486.

Song, Shuran, Yu, Fisher, Zeng, Andy, Chang, Angel X., Savva, Manolis, and Funkhouser, Thomas (2017). "Semantic Scene Completion from a Single Depth Image". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 190–198. DOI: 10.1109/CVPR.2017.28.

Srinivasan, Pratul P., Deng, Boyang, Zhang, Xiuming, Tancik, Matthew, Mildenhall, Ben, and Barron, Jonathan T. (2021). "NeRV: Neural Reflectance and Visibility Fields for Relighting and View Synthesis". In: *CVPR*.

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

Tallavajhula, Abhijeet, Mericli, Cetin, and Kelly, Alonzo (2018). "Off-Road Lidar Simulation with Data-Driven Terrain Primitives". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7470–7477. DOI: 10.1109/ICRA.2018.8461198.

Tan, Weikai, Qin, Nannan, Ma, Lingfei, Li, Ying, Du, Jing, Cai, Guorong, Yang, Ke, and Li, Jonathan (June 2020). "Toronto-3D: A Large-Scale Mobile LiDAR Dataset for Semantic Segmentation of Urban Roadways". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*.

Tchapmi, Lyne P., Kosaraju, Vineet, Rezatofighi, Hamid, Reid, Ian, and Savarese, Silvio (2019). "TopNet: Structural Point Cloud Decoder". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 383–392. DOI: 10.1109/CVPR.2019.00047.

Thomas, Hugues, Qi, Charles R., Deschaud, Jean-Emmanuel, Marcotegui, Beatriz, Goulette, François, and Guibas, Leonidas (2019). "KPConv: Flexible and Deformable Convolution for Point Clouds". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 6410–6419. DOI: 10.1109/ICCV.2019.00651.

Ummenhofer, Benjamin and Koltun, Vladlen (2021). "Adaptive Surface Reconstruction with Multiscale Convolutional Kernels". In: *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 5631–5640. DOI: 10.1109/ICCV48922.2021.00560.

Vallet, Bruno, Brédif, Mathieu, Serna, Andres, Marcotegui, Beatriz, and Paparoditis, Nicolas (2015). "TerraMobilita/iQmulus urban point cloud analysis benchmark". In: *Computers & Graphics* 49, pp. 126–133. ISSN: 0097-8493. DOI: https://doi.org/10.1016/j.cag.2015.03.004.

Varney, Nina, Asari, Vijayan K., and Graehling, Quinn (2020). "DALES: A Large-scale Aerial LiDAR Data Set for Semantic Segmentation". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 717–726. DOI: 10.1109/CVPRW50498.2020.00101.

Venkatesh, Rahul, Sharma, Sarthak, Ghosh, Aurobrata, Jeni, László A., and Singh, Maneesh (2020). "DUDE: Deep Unsigned Distance Embeddings for Hi-Fidelity Representation of Complex 3D Surfaces". In: *CoRR* abs/2011.02570. arXiv: 2011.02570. URL: https://arxiv.org/abs/2011.02570.

Wald, Ingo, Woop, Sven, Benthin, Carsten, Johnson, Gregory S., and Ernst, Manfred (July 2014). "Embree: A Kernel Framework for Efficient CPU Ray Tracing". In: *ACM Trans. Graph.* 33.4. ISSN: 0730-0301. DOI: 10.1145/2601097.2601199. URL: https://doi.org/10.1145/2601097.2601199.

Wang, Xiaogang, Ang, Marcelo H, and Hee Lee, Gim (2020). "Point Cloud Completion by Learning Shape Priors". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Las Vegas, NV, USA: IEEE Press, pp. 10719–10726. DOI: 10.1109/IROS45743.2020.9340862. URL: https://doi.org/10.1109/IROS45743.2020.9340862.

Wang, Yida, Tan, David Joseph, Navab, Nassir, and Tombari, Federico (2018). "Adversarial Semantic Scene Completion from a Single Depth Image". In: *2018 International Conference on 3D Vision (3DV)*, pp. 426–434. DOI: 10.1109/3DV.2018.00056.

Wen, Xin, Li, Tianyang, Han, Zhizhong, and Liu, Yu-Shen (2020). "Point Cloud Completion by Skip-Attention Network With Hierarchical Folding". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1936–1945.

Weng, Xinshuo, Wang, Jianren, Held, David, and Kitani, Kris (2020). "3D Multi-Object Tracking: A Baseline and New Evaluation Metrics". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 10359–10366.

Wu, Bichen, Wan, Alvin, Yue, Xiangyu, and Keutzer, Kurt (2018). "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)*, pp. 1887–1893. DOI: 10.1109/ICRA.2018.8462926.

Wu, Bichen, Zhou, Xuanyu, Zhao, Sicheng, Yue, Xiangyu, and Keutzer, Kurt (2019). "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)*, pp. 4376–4382. DOI: 10.1109/ICRA.2019.8793495.

Wu, Jainhua and Kobbelt, Leif (2004). "Optimized Sub-Sampling of Point Sets for Surface Splatting". In: *Computer Graphics Forum* 23.3, pp. 643–652. DOI: 10.1111/j.1467-8659.2004.00796.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2004.00796.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2004.00796.x.

Xiao, Aoran, Huang, Jiaxing, Guan, Dayan, Zhan, Fangneng, and Lu, Shijian (2021). "SynLiDAR: Learning From Synthetic LiDAR Sequential Point Cloud for Semantic Segmentation". In: *arXiv preprint arXiv:2107.05399*.

Xiao, Jianxiong, Owens, Andrew, and Torralba, Antonio (2013). "SUN3D: A Database of Big Spaces Reconstructed Using SfM and Object Labels". In: *2013 IEEE International Conference on Computer Vision (ICCV)*, pp. 1625–1632. DOI: 10.1109/ICCV.2013.458.

Xu, Qiangeng, Xu, Zexiang, Philip, Julien, Bi, Sai, Shu, Zhixin, Sunkavalli, Kalyan, and Neumann, Ulrich (2022). "Point-nerf: Point-based neural radiance fields". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5438–5448.

Xu, Yan, Zhu, Xinge, Shi, Jianping, Zhang, Guofeng, Bao, Hujun, and Li, Hongsheng (2019). "Depth Completion From Sparse LiDAR Data With Depth-Normal Constraints". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 2811–2820. DOI: 10.1109/ICCV.2019.00290.

Yan, Xu, Gao, Jiantao, Li, Jie, Zhang, Ruimao, Li, Zhen, Huang, Rui, and Cui, Shuguang (2021). "Sparse Single Sweep LiDAR Point Cloud Segmentation via Learning Contextual Shape Priors from Scene Completion". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 4, pp. 3101–3109.

Yang, Guandao, Huang, Xun, Hao, Zekun, Liu, Ming-Yu, Belongie, Serge, and Hariharan, Bharath (2019). "PointFlow: 3D Point Cloud Generation With Continuous Normalizing Flows". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 4540–4549. DOI: 10.1109/ICCV.2019.00464.

Yin, Tianwei, Zhou, Xingyi, and Krahenbuhl, Philipp (2021). "Center-based 3d object detection and tracking". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11784–11793.

Yu, Alex, Li, Ruilong, Tancik, Matthew, Li, Hao, Ng, Ren, and Kanazawa, Angjoo (2021a). "PlenOctrees for Real-time Rendering of Neural Radiance Fields". In: *ICCV*.

Yu, Lequan, Li, Xianzhi, Fu, Chi-Wing, Cohen-Or, Daniel, and Heng, Pheng-Ann (2018). "PU-Net: Point Cloud Upsampling Network". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2790–2799. DOI: 10.1109/CVPR.2018.00295.

Yu, Xumin, Rao, Yongming, Wang, Ziyi, Liu, Zuyan, Lu, Jiwen, and Zhou, Jie (2021b). "PoinTr: Diverse Point Cloud Completion with Geometry-Aware Transformers". In: *ICCV*.

Yuan, Wentao, Khot, Tejas, Held, David, Mertz, Christoph, and Hebert, Martial (2018). "PCN: Point Completion Network". In: *2018 International Conference on 3D Vision (3DV)*, pp. 728–737. DOI: 10.1109/3DV.2018.00088.

Yue, Xiangyu, Wu, Bichen, Seshia, Sanjit A., Keutzer, Kurt, and Sangiovanni-Vincentelli, Alberto L. (2018). "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. Yokohama, Japan: Association for Computing Machinery, pp. 458–464. ISBN: 9781450350464. DOI: 10.1145/3206025.3206080. URL: https://doi.org/10.1145/3206025.3206080.

Zhang, Jiahui, Zhao, Hao, Yao, Anbang, Chen, Yurong, Zhang, Li, and Liao, Hongen (2018). "Efficient Semantic Scene Completion Network with Spatial Group Convolution". In: *Computer Vision – ECCV 2018*. Ed. by Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss. Cham: Springer International Publishing, pp. 749–765. ISBN: 978-3-030-01258-8.

Zhang, Pingping, Liu, Wei, Lei, Yinjie, Lu, Huchuan, and Yang, Xiaoyun (2019). "Cascaded Context Pyramid for Full-Resolution 3D Semantic Scene Completion". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 7800–7809. DOI: 10.1109/ICCV.2019.00789.

Zhao, Fang, Wang, Wenhao, Liao, Shengcai, and Shao, Ling (2021). "Learning Anchored Unsigned Distance Functions with Gradient Direction Alignment for Single-view Garment Reconstruction". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 12674–12683.

Zhao, Ruibin, Pang, Mingyong, Liu, Caixia, and Zhang, Yanling (2019). "Robust Normal Estimation for 3D LiDAR Point Clouds in Urban Environments". In: *Sensors* 19.5. ISSN: 1424-8220. DOI: 10.3390/s19051248. URL: https://www.mdpi.com/1424-8220/19/5/1248.

Zhao, Sicheng, Wang, Yezhen, Li, Bo, Wu, Bichen, Gao, Yang, Xu, Pengfei, Darrell, Trevor, and Keutzer, Kurt (2020). "ePointDA: An End-to-End Simulation-to-Real Domain Adaptation Framework for LiDAR Point Cloud Segmentation". In: *CoRR* abs/2009.03456. arXiv: 2009.03456. URL: https://arxiv.org/abs/2009.03456.

Zhou, Kaiyue, Dong, Ming, and Arslanturk, Suzan (2021). ""Zero Shot" Point Cloud Upsampling". In: *CoRR* abs/2106.13765. arXiv: 2106.13765. URL: https://arxiv.org/abs/2106.13765.

Zhou, Yang, Shen, Shuhan, and Hu, Zhanyi (2019). "Detail Preserved Surface Reconstruction from Point Cloud". In: *Sensors* 19.6. ISSN: 1424-8220. DOI: 10.3390/s19061278. URL: https://www.mdpi.com/1424-8220/19/6/1278.

Zhu, Jingwei, Gehrung, Joachim, Huang, Rong, Borgmann, Björn, Sun, Zhenghao, Hoegner, Ludwig, Hebel, Marcus, Xu, Yusheng, and Stilla, Uwe (2020). "TUM-MLS-2016: An Annotated Mobile LiDAR Dataset of the TUM City Campus for Semantic Point Cloud Interpretation in Urban Areas". In: *Remote Sensing* 12.11. ISSN: 2072-4292. DOI: 10.3390/rs12111875. URL: https://www.mdpi.com/2072-4292/12/11/1875.

Zwicker, Matthias, Pfister, Hanspeter, Baar, Jeroen van, and Gross, Markus (2001). "Surface Splatting". In: *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '01. New York, NY, USA: Association for Computing Machinery, pp. 371–378. ISBN: 158113374X. DOI: 10.1145/383259.383300. URL: https://doi.org/10.1145/383259.383300.

RÉSUMÉ

Le développement de véhicules autonomes a suscité un intérêt accru de la part de la communauté de la recherche et de l'industrie au cours de la dernière décennie. Développer des algorithmes de véhicules autonomes nécessite de moderniser les véhicules avec plusieurs capteurs, ce qui a un coût élevé. De plus, l'exploitation et l'entretien quotidiens des véhicules augmentent encore les coûts et présentent un risque élevé pour les autres véhicules et les personnes se trouvant dans leur environnement. Le développement et les tests peuvent également être réalisés en simulant le véhicule autonome et les différents capteurs dans un environnement virtuel. Cependant, les environnements virtuels créés manuellement ne parviennent pas à se généraliser aux scènes du monde réel, en raison de l'écart de domaine qui découle des modèles trop simplifiés utilisés dans de tels environnements. Dans cette thèse, nous proposons de réduire cet écart de domaine en exploitant des scans du monde réel de scènes urbaines sous la forme de nuages de points 3D acquis à l'aide d'un capteur LiDAR monté sur un système de cartographie mobile. Pour cette raison, nous créons un nouveau jeu de données de nuage de points 3D annoté et proposons une chaîne de traitement de simulation automatique. La chaîne de traitement introduit une nouvelle représentation intermédiaire pour compléter la géométrie de la scène grâce à des architectures 3D profondes, une méthode de modélisation de scène précise basée sur du splatting adaptatif sémantique pour créer l'environnement virtuel et enfin, une méthode de simulation LiDAR en temps réel. Notre chaîne de traitement est rapide, et automatique et peut être utilisée pour augmenter le domaine des scénarios et générer des simulations massives.

MOTS CLÉS

Nuages de points 3D, représentation de scène, complétion de scène, modélisation des scènes 3D, reconstruction de surface, lancer de rayons en temps réel, simulation de LiDAR

ABSTRACT

The development of autonomous vehicles has seen increased interest from the research community and industry over the last decade. Developing algorithms used by autonomous vehicles requires retrofitting vehicles with multiple sensors, which has a high cost. Moreover, the vehicles daily operation and maintenance raise the cost even further and introduce a high risk on other vehicles and people in their surrounding. The development and testing can be also achieved by simulating the autonomous vehicle and the different sensors in a virtual environment. However, manually handcrafted virtual environments fail to generalize to real-world scenes, due to the domain gap that arises from the over-simplified models used in such environments. In this thesis, we propose to reduce this domain gap by leveraging real-world scans of urban scenes in the form of 3D point clouds acquired using a LiDAR sensor mounted on a mobile mapping system. Toward this end, we create a new annotated 3D point cloud dataset and propose an automatic simulation pipeline. The pipeline introduces a new intermediate representation to complete the scene geometry through deep 3D architectures, an accurate scene modeling method based on semantic adaptive splatting to create the virtual environment and finally, a real-time LiDAR simulation method. Our pipeline is fast, automatic and can be used to augment the scenarios domain and generate massive simulations.

KEYWORDS

3D point clouds, scene representation, scene completion, scene modeling, surface reconstruction, real-time ray tracing, LiDAR simulation