TITLE

- AADS: Augmented Autonomous Driving Simulation using Data-driven Algorithms
- Augmented Autonomous Driving Simulation

Authors

W. Li,[1]*† C. W. Pan,[2]* R. Zhang,[3]* J. P. Ren,[3] Y. X. Ma,[4] J. Fang,[1] F. L. Yan,[1] Q. C. Geng,[5] X. Y. Huang,[1] H. J. Gong,[6] W. W. Xu,[3] G. P. Wang,[2] D. Manocha,[7]† R. G. Yang[1]†

Affiliations

1, Baidu Research, Beijing, China
2, Peking University, Beijing, China
3, Zhejiang University, Hangzhou, China
4, The University of Hong Kong, Hongkong, China
5, Beihang University, Beijing, China
6, Nanjing University of Aeronautics and Astronautics, Nanjing, China
7, University of Maryland, College Park, USA

* These authors contributed equally to this work.
† Corresponding authors. Emails: {liwei87, yangruigang}@baidu.com, dm@cs.umd.edu.

Abstract

Simulation systems have become an essential component in the development and validation of autonomous driving technologies. The prevailing state-of-the-art approach for simulation is to use game engines or high-fidelity computer graphics (CG) models to create driving scenarios. However, creating CG models and vehicle movements (a.k.a. the assets for simulation) remains a manual task that can be costly and time-consuming. In addition, the fidelity of CG images still lacks the richness and authenticity of real-world images and using these CG images for training leads to degraded performance.

In this paper we present a novel approach to address these issues: Augmented Autonomous Driving Simulation (AADS). Our formulation augments real-world pictures with a simulated traffic flow to create photo-realistic simulation images and renderings. More specifically, we use LiDAR and cameras to scan street scenes. From the acquired trajectory data, we generate highly plausible traffic flows for cars and pedestrians and compose them into the background. The composite images can be re-synthesized with different viewpoints and sensor models (camera or LiDAR). The resulting images are photo-realistic, fully annotated, and ready for end-to-end training and testing of autonomous driving systems from perception to planning. We explain our system design and validate our algorithms with a number of autonomous driving tasks from detection to segmentation and predictions.

Compared to traditional approaches, our method offers unmatched scalability and realism. Scalability is particularly important for AD simulation and we believe the complexity and diversity of the real world cannot be realistically captured in a virtual environment. Our augmented approach combines the flexibility of a virtual environment (e.g., vehicle movements) with the richness of the real world to allow effective simulation of any location on earth.

Summary

By augmented images with synthesized traffic, we present a truly scalable and high-fidelity simulation system to enable end-to-end training/testing of autonomous driving anywhere on earth.

MAIN TEXT

1. Introduction

Autonomous vehicles (AV) have attracted considerable attention in recent years from researchers, venture capitalists, as well as the general public. The societal benefits in terms of safety, mobility, and environmental concerns are expected to be tremendous and have captivated the attention of people across the globe. However, in light of recent accidents involving AV, it has become clear that there is still a long way to go to meet the high standards and expectations associated with AV.

Safety is the key requirement for AV. It has been argued that an AV has to be test-driven hundreds of millions of miles in challenging conditions to demonstrate statistical reliability in terms of reductions in fatalities and injuries (*1*), which could take tens of years of road tests even under the most aggressive evaluation schemes. New methods and metrics are being developed to validate the safety of AV. One possible solution is to use simulation systems, which are common in other domains like law enforcement, defense, and medical training. Simulations of autonomous driving can serve two purposes. The first is to test and validate the capability of AV in terms of environmental perception, navigation, and control. The second is to generate a large amount of labelled training data to train machine learning methods, e.g., a deep neural network. The second purpose has recently been adopted in computer vision (*2, 3*).

The most common way to generate such a simulator is to use a combination of computer graphics, physics-based modeling, and robot motion planning techniques to create a synthetic environment in which moving vehicles can be animated and rendered. A number of simulators have recently been developed, such as Intel's CARLA (*4*), Microsoft's AirSim (*5*), NVIDIA's Drive Constellation (*6*), Google/Waymo's CarCraft (*7*), etc.

While all of these simulators achieve state-of-the-art synthetic rendering results, these approaches are difficult to deploy in the real world. A major hurdle is the need for high-fidelity environmental models. The cost of creating life-like CG models is prohibitively high. Consequently, synthetic images from these simulators have a distinct, CG-rendered look-and-feel, i.e. gaming or VR system quality. In addition, the animation of moving obstacles, such as cars and pedestrians, is usually scripted and lacks the flexibility and realism of real scenes. Moreover, these systems are unable to generate different scenarios composed of vehicles, pedestrians, or bicycles, as observed in urban environments.

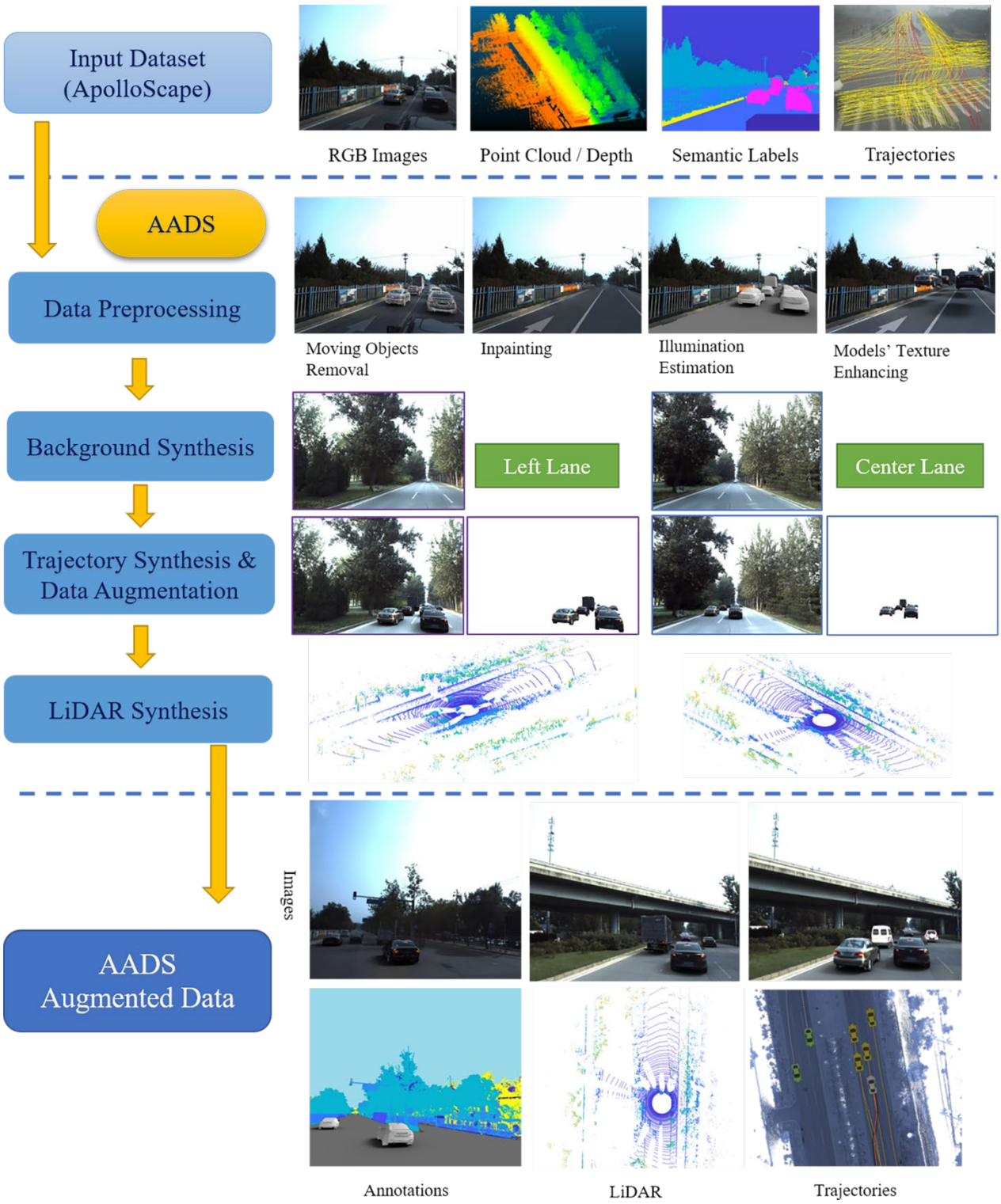

Fig. 1. The inputs, processing pipeline, and outputs of our AADS system. The top is the input dataset. The pipeline of AADS is shown between the dashed lines and contains data preprocessing, novel background synthesis, trajectory synthesis, moving objects augmenting, and LiDAR simulation. The bottom shows the outputs from the AADS system, which include synthesized RGB images, a LiDAR point cloud, and trajectories with ground truth annotations.

In this paper we present a new data-driven approach for end-to-end simulation for autonomous driving: Augmented Autonomous Driving Simulation (AADS). Our method augments real-world pictures with a simulated traffic flow to create photo-realistic simulation scenarios that resemble real-world renderings. Fig. 1 shows the pipeline of our AADS system as well as its major inputs and outputs. Specifically, we propose using LiDAR and cameras to scan street scenes. We decompose the input data into background, scene illumination, and foreground objects. We present a new view synthesis technique to enable changing viewpoints on the static background. The foreground vehicles are fitted with 3D CG models. With accurately estimated outdoor illumination, the 3D vehicle models, computer generated pedestrians, and other movable subjects can be repositioned and rendered back into the background images to create photo-realistic street-view images that look like they were captured from a dashboard camera on a vehicle. Furthermore, the simulated traffic flows, e.g., the placement and movement of synthetic objects, are based on captured real-world vehicle trajectories that look natural and capture the complexity and diversity of real-world scenarios.

Compared with traditional VR-based or game-engine-based AV simulation systems, AADS provides more accurate end-to-end simulation capability without requiring costly CG models or tedious programming to define the traffic flow. Therefore, it can be deployed for large-scale use, including training and evaluation of new navigation strategies for the ego-vehicle.

The key to AADS's success is the wide availability of 3D scene scans and vehicle trajectory data, both of which are needed for the automatic generation of new traffic scenarios. We will also release part of the real-world data that we have collected for the development and evaluation of AADS. These data are either the first of their kind or the largest publicly available in terms of urban scenarios for autonomous driving. The data are fully annotated by a professional labeling service. In addition to AADS, they can also be used for many perception and planning related tasks to drive further research in this area.

The technical innovations in this paper include:

- A new data-driven approach for autonomous driving simulation. By using scanned street-view images and real trajectories, both photo-realistic images and plausible movement patterns can be synthesized automatically. This direct scan-to-simulation pipeline, with little manual intervention, enables large-scale testing of autonomous cars virtually anywhere and anytime within a closed-loop simulation.
- We present a novel view synthesis method to enable view interpolation and extrapolation with only a few images. Compared to prior approaches, it generates better quality images with fewer artifacts.
- A new set of datasets, including the largest set of traffic trajectories and the largest 3D street-view dataset with pixel/point level annotation. All of these are captured in metropolitan areas, with dense and complex traffic patterns. This kind of dense urban traffic poses significant challenges for autonomous driving.

1. 1 Previous Methods

Simulation for autonomous driving (AD) is a very huge topic. Traditionally, simulation capabilities have been primarily used in the planning and control phase of AD, e.g., (*8*–*14*). More recently, simulation has been used in the entire AD pipeline, from perception and planning to control (see the survey by P. et al. (*15*)).

While Waymo has claimed that its autonomous vehicle has been tested for billions of miles in their proprietary simulation system, CarCraft (*7*), little technical detail has been released to the public in terms of its fidelity for training machine learning methods. Researchers have tried to use images from video games to train deep-learning-based perception systems (*16*, *17*).

Recently, a number of high-fidelity simulators dedicated to AD simulation have been developed, such as Intel's CARLA (*4*), Microsoft's AirSim (*5*), and NVIDIA's Drive Constellation (*6*). They allow end-to-end, closed-loop training and testing of the entire AD pipeline beyond the generation of annotated training data. All of these simulators are based on current gaming techniques or engines, which generate high-quality synthetic images in real-time. A limitation of these systems is the fidelity of the resulting environmental model. Even with the state-of-the-art rendering capabilities, the images produced by these simulators are obviously synthetic. Current state-of-the-art computer graphics rendering may not provide enough accuracy and details for machine learning methods.

With the availability of LiDAR devices and advances in structure-from-motion, it is now possible to capture large urban scenes in 3D. However, turning the large-scale point cloud into a CG-quality rendered image is still an on-going research problem. Models reconstructed from these point clouds often lack details or complete textures (*18*). In addition, AD simulators have to address the problem of realistic traffic patterns and movements. Traditional traffic flow simulation algorithms mainly focus on generating trajectories for cars and vehicles and do not take into account the realistic movements of individual cars or pedestrians. One of the challenges is to simulate realistic traffic patterns, particularly in complex situations when traffic is dense and involves heterogenous agents (e.g., an intersection scenario with pedestrians in a cross walk).

Our work is related to the approach described by Alhaija et al. (*19*) in which 3D vehicle models are rendered onto existing captured real-world background images. However, the observation viewpoint is fixed at capture time and the 3D models are chosen from an existing 3D repository that may or may not match those in the real-world images. Their approach can be used to augment still images for training perception applications. In contrast, with the ability to freely change the observation viewpoint, our system could not only play a role in data augmentation but could also enhance a closed-loop simulator like CARLA (*4*) or AirSim (*5*). Further enhanced by realistic traffics simulation ability, our system can also be used for path planning and driving decision applications. In those dynamic applications, our system can generation data in a loop for reinforcement learning and learning-by-demonstration algorithms. Overall, the proposed approach is the first to enable closed-loop, end-to-end simulation without the need for environmental modelling and human intervention.

2. Results

Since AADS is data-driven, we will first explain the unique datasets that have collected. Some of the datasets have already been released, while others will be released along with this paper. We will then show results for the synthesis of virtual views and generation of traffic flows, two key components of AADS. Finally, we will evaluate AADS's effectiveness for autonomous driving simulation. Specifically, we will show that the simulated RGB and LiDAR images are useful for improving the performance of the

perception system, while the simulated trajectories are useful for improving predictions of obstacle movements–a critical component for the planning and control phases for autonomous cars.

2. 1 The Dataset

When collecting a dataset, we use a hardware system consisting of two Riegl laser scanners, one real-time line-scanning LiDAR (Velodyne 64-Line), a VMX-CS6 stereo camera system, and a high-precision IMU/GNSS. With the Riegl scanners, our system can obtain higher-density point clouds with a better accuracy than widely used LiDAR scanners, while the VMX-CS6 system provides a wide baseline stereo camera with a high resolution (3384 * 2710). With the Velodyne LiDAR, we can obtain the shapes and positions of moving objects. To scan a scene, the hardware is calibrated, synchronized, and then mounted on the top of a mid-size SUV that cruises around the target scene at an average speed of 30km per hour. Note that the RGB images are taken about once every meter.

In our labeling process, instead of fully annotating all 2D/RGB and 3D/point cloud data manually, we developed a novel labeling pipeline to make our labeling process accurate and efficient. Because 2D labeling is expensive in terms of time and labor, we combine the two stages, i.e. 3D labeling and 2D labeling. By using easy-to-label 3D annotations, we can automatically generate high-quality 2D annotations of static backgrounds/objects in all the image frames by 3D-2D projections. Details of the labeling process can be found in (*20*).

For each image frame, we annotate 25 different classes covered by five groups in both 3D point clouds. In addition to standard annotation classes such as cars, motorcycles, traffic cones, and so on, we added a new “tricycle” class, a popular mode of transportation in East Asian countries. We also annotate 35 different lane markings in both 2D and 3D that are not currently available in open datasets. These lane markings are defined based on color (e.g., white and yellow), type (e.g., solid and broken), and usage (e.g., dividing, guiding, stopping, and parking).

	KITTI	CityScapes	Mapillary	BDD100K	ApolloScape			
Total images	14,999	25,000	25,000	120,000,000	143,906			
Annotated images (bounding-box level)	14,999	no	no	100,000	no			
Annotated images (pixel level)	400	5,000 Fine 20,000 Coarse	25,000	10,000	143,906			
Scene Complexity (average per image)	bounding-box level	pixel level	pixel level	bounding-box level	pixel level			
	person : 0.8 vehicle : 4.1	person : 7.0 vehicle : 11.8	-	person : ~1.3 vehicle : ~11.0	difficulty	easy	moderate	hard
					person	1.1	6.2	16.9
					vehicle	12.7	24.0	38.1
Diversity	day time	day time 50 cities	various weather day & night 6 continents	various weather day & night 4 regions in US	various weather day time 4 regions in China			
3D Annotation	box-level	no	no	no	point-level			
Video Annotation	box-level	no	no	no	pixel-level			
Lane Annotation	no	no	2D / 2 classes	2D / 8 classes	3D / 2D video 28 classes			
Location Accuracy	cm	-	meter	meter	cm			

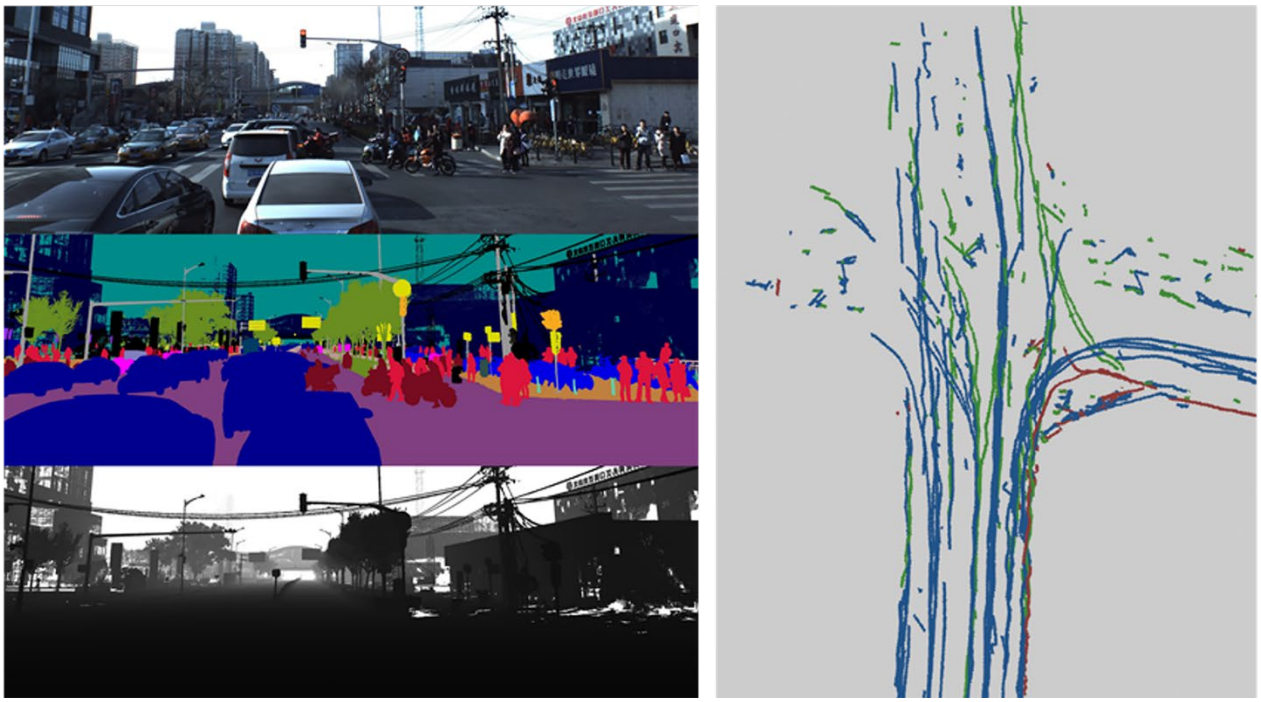

Fig. 2. The ApolloScape dataset and its extension. The top table compares ApolloScape with other popular datasets. The bottom left shows RGB images, annotations, and a point cloud, from top to bottom. The bottom right shows some labeled traffic trajectories from the dataset.

The table in Fig. 2 compares our dataset and other street-view datasets. Our dataset outperforms other datasets in many aspects such as scene complexity, number of pixel-level annotations, number of classes, and so on. We have released 143,906 video frames and corresponding pixel-level annotations. Images are assigned to three degrees of difficulty (e.g., easy, moderate, and hard) based on scene complexity, a measure of the number of movable objects in an image. Our dataset also contains challenging lighting conditions, such as high-contrast regions due to sunlight, as well as shadows from overpasses. We name the dataset of RGB images **ApolloScape-RGB**. More importantly,

we also provide 3D point-level annotations on corresponding point could which is not available in **ApolloScape-PC**, another street-view dataset.

In addition to this article, we also announce **ApolloScape-TRAJ**, a new dataset of trajectories. This dataset is a large-scale dataset for urban streets that includes RGB image sequences and trajectory files. It focuses on trajectories of heterogeneous traffic-agents for planning, prediction, and simulation tasks. The dataset includes RGB videos with around 100k 1920 × 1080 images and 1000km of trajectories for all kinds of moving traffic agents. We use the Apollo acquisition car to collect traffic data and generate trajectories. In Beijing, we collected a dataset of trajectories under a variety of lighting conditions and traffic densities. The dataset includes many challenging scenarios including many vehicles, bicycles, and pedestrians moving around one another.

2.2 Evaluations of Augmented Background Synthesis

An important part of our AADS system is synthesizing background images in specific views using images captured in fixed views when running closed-loop simulations. This ability stems from the utilization of the image-based rendering technique and avoids pre-requisite modeling of the full environment.

There is a large literature on image-based rendering techniques, though relatively little has been written on capturing scenes with sparse images. We focus on wide baseline stereo image-based rendering for street-view scenes: the overlap between left images and right images may be less than half the size of full images. Technically, obtaining reliable depth is an important challenge for image-based rendering techniques. Thus, methods such as (*21*) use the multi-view stereo (MVS) method to estimate depth maps. However, most street-view datasets provide laser scanned point clouds, which can be used to generate initial depth maps by rendering point clouds. As point clouds tend to be sparse and noisy, initial estimates of depth maps are full of outliers and holes, and need to be refined before they are passed on to downstream processing. Thus, we propose an effective depth refinement method that includes depth filtering and completion procedures. To evaluate our depth refinement method, we use initial and refined depth maps ((b) and (e) in Fig. 3) to synthesize the same novel view. Results are shown in Fig. 3 (f) and (g), respectively. Using depth maps without refinement to run image-based rendering, the results suffer from artifacts near errors and holes in depth maps. Specifically, in Fig. 3 (f) and (h), fluctuations appear in the green rectangle as the view changes, while window frames keep straight when using refined depth in the yellow rectangle.

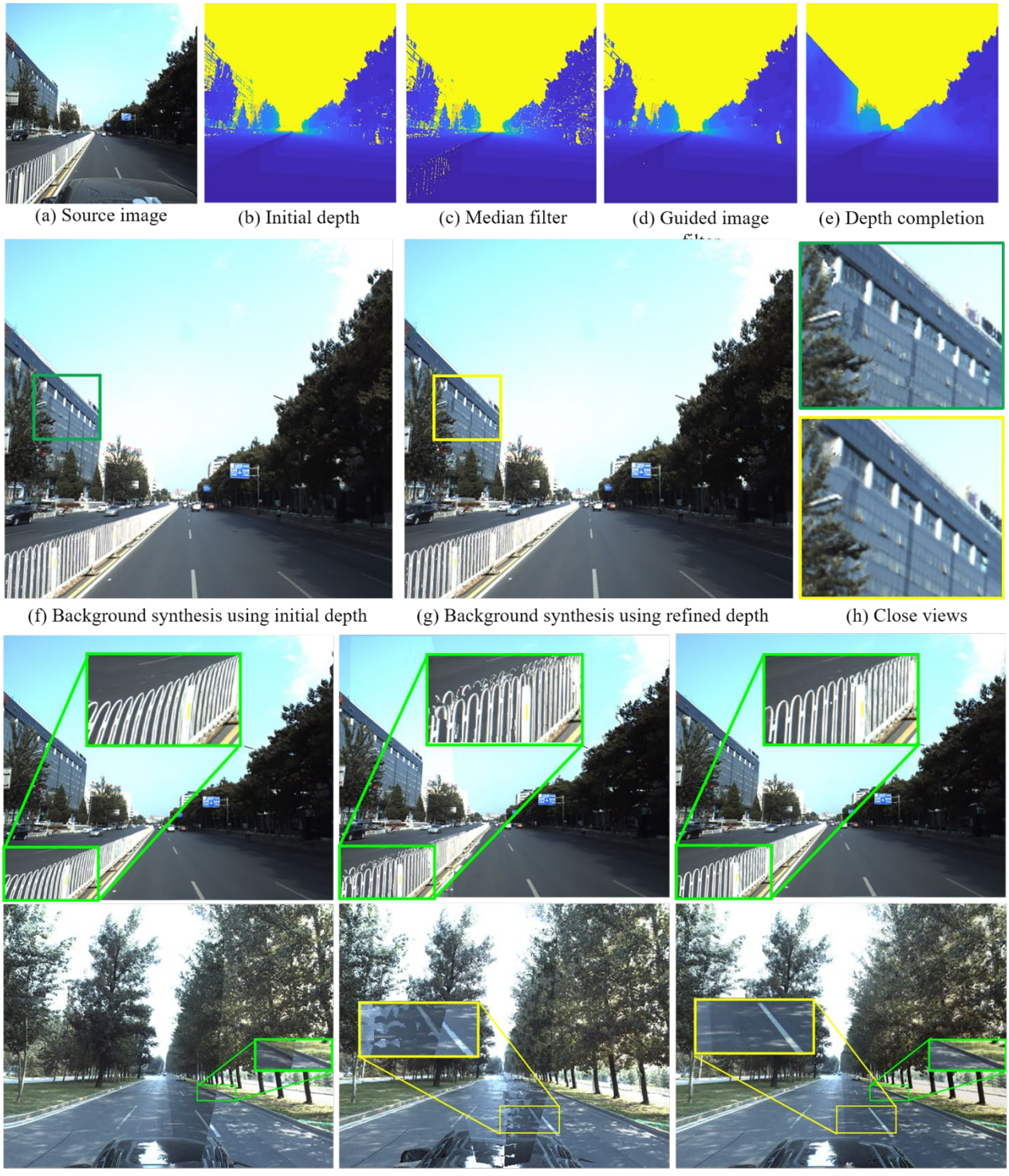

Fig. 3. View Synthesis Result and Effectiveness of Depth Refinement. (a) and (b) The raw RGB and depth images in our dataset, respectively. (c)~(e) The result of depth refinement after filtering and completion. (f) and (g) The result of view synthesis using initial and refined depths with close views in (h). (i)~(k) The final results of view synthesis using Liu et al.'s method (*22*), Chaurasia et al.'s method (*23*), and our method, respectively.

To evaluate our image-based rendering algorithm (specifically the novel view synthesis algorithm) with refined depth maps, we compare our method with two representative

approaches: the content preserving warping method (Liu et al. (*22*)) and Chaurasia et al.'s method (*23*). Note that, in the implementation of Chaurasia et al.'s method (*23*), we use the similarity of super pixels (*24*) to complete the depth map and perform a local shape-preserving warp on each super pixel.

The synthesized images in Fig. 3 are generated using four reference images. As images are captured by a stereo camera, the four reference images can be considered as two pairs of stereo images with close to parallel views in which the angle between two optical axes of the stereo images is small, but the baseline is relatively wide (about 1m). We compare our view interpolation and extrapolation results with classical methods. As shown in the third row of Fig. 3, Liu et al.'s method performs well for small changes in the novel view compared to the input views. When the view translation becomes larger, view distortion artifacts become apparent (like the fence in the green rectangle, the shape of which is deformed inappropriately). For Chaurasia et al.'s method, ghost artifacts appear when neighboring super pixels are assigned to inappropriate or incorrect depths. Our method obtains correct depths and preserve invariant shapes of objects when the view changes, handeling both interpolation and extrapolation. The fourth row of Fig 3 evaluates another scene with both a wide baseline and a large rotation angle. Because of large changes in the novel view, neither Liu et al.'s method nor Chaurasia et al.'s method aligns well with neighboring reference views. As shown in the figure, curbstones in the green rectangle and the white lane marker in the yellow rectangle suffer from misalignment artifacts. In addition, due to tone inconsistencies in the input images, seams are obvious in the results of Liu et al.'s and Chaurasia et al.'s methods. In contrast, our method can effectively eliminate misalignment as well as seam artifacts.

To further illustrate the effectiveness of our view synthesis approach for closed-loop simulation, we have included a video (Movie S3 in supplementary materials titled "Synthesizing lane changes"). The video shows the synthesized front camera view from a driving car that is changing lanes several times. Our view synthesis approach is sufficient for handling such lane changes because it interpolates or extrapolates the viewpoint.

2.3 Evaluations of Trajectories Synthesis

Another pillar for AADS is its ability to generate plausible traffic flow, particularly when there are interactions between vehicles and pedestrians, e.g., heterogeneous agents who move at different speeds and with different dynamics. This topic is a full research area in its own right, and we have developed new techniques for heterogeneous agent simulations. For the sake of completeness, we briefly show the main result here in Fig. 4. Readers are referred to (*25*) for more technical details. Specifically, Fig. 4 shows the comparison with the ground truth from the input dataset, results of our simulation method, and results of Chao et al.'s method (*26*), a state-of-the-art multi-agent simulation approach. In the evaluation, the traffic is simulated on a straight 4-lane road. In our method, the number, positions, and velocities of agents are randomly initialized according to the dataset. We evaluate the comparison using the metric of velocity and minimal distance probability distributions. The metrics are divided into 30 intervals and probabilities are calculated over all the intervals. As shown in Fig. 4, our simulation results are closer to the input data in both the velocity distribution and the minimal distance distribution.

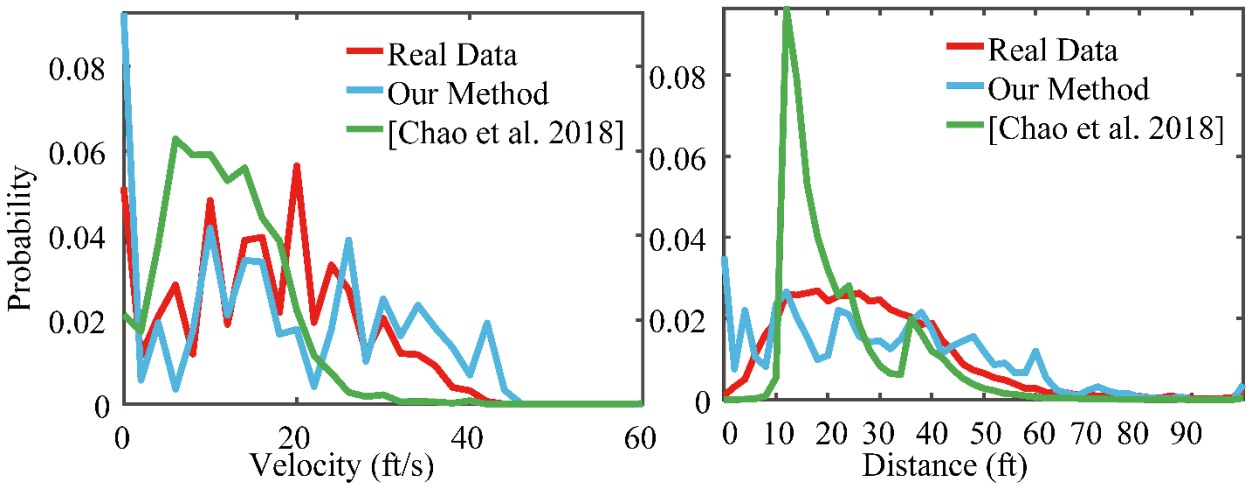

Fig. 4. Comparison of Traffic Synthesis. Velocity and minimal distance distribution of traffic simulation using our method, Chao et al.'s method, and the ground truth.

2.4 AADS Evaluations by Autonomous Driving Applications

As shown in Fig. 1, simulation and our AADS can simultaneously produce the following augmented data: 1) photo-realistic RGB images with annotation information such as semantic labels, 3D bounding boxes, etc.; 2) an augmented LiDAR point cloud; 3) typical traffic flows. In following evaluations, those data augmentations are synthesized based on our ApolloScape dataset. We summarize the AADS synthetic data and evaluations in terms of RGB images, point clouds, and trajectories:

- **AADS-RGB**: For the baseline training set of ApolloScape-RGB, we augment RGB images with AADS and generate corresponding annotations for augmented moving agents. This dataset is named AADS-RGB and is used to evaluate our image synthesis method.
- **AADS-PC**: With our AADS system, we synthesize up to 100k new point cloud frames by simulating the Velodyne HDL-64E S3 LiDAR sensor based on the ApolloScape-PC dataset. The simulation dataset has the same object categories as and numbers of objects in each category similar to ApolloScape-PC.
- **AADS-TRAJ**: Our AADS system can also produce new trajectories based on the ApolloScape-TRAJ dataset. We further evaluate such augmented data using a trajectory prediction method.

2.4.1 Object Detection with AADS-RGB

With respect to the evaluations of AADS' capability to simulate camera images, we use two real and three virtual datasets: ApolloScape RGB annotated images (ApolloScape-RGB), CityScapes, virtual KITTI (VKITTI), synthesized data from the popular simulator CARLA, and our synthesized data (AADS-RGB).

VKITTI: We use the VKITTI (*2*) dataset to compare our system with a fully synthetic method. The full dataset contains 21260 images with different weather and lighting conditions. 1600 images were randonwly selected as a training set.

CityScapes: CityScapes (*27*) is a dataset of urban street scenes. There are 5000 annotated images with fine instance level semantic labels. We use the validation set of 492 images as the testing data set.

CARLA: CARLA(*4*) is the most recent and popular virtual-reality simulator for autonomous driving. Up to now, it provides two manually built scenes with car models. Because the size of the scene is limited, we generated 1600 images distributed as evenly as possible in the simulated scene.

In this section, we will show the effectiveness of the AADS simulated RGB data. We use the state-of-the-art objection detection algorithm Mask-RCNN (*28*) to perform experiments. The results are compared with the standard average precision metric of an intersection-over-union (IoU) threshold of 50% (AP50) and 70% (AP70), and a mean bounding box AP (mAP) with an across threshold at IoU ranging from 5% to 95% in steps of 5%. Because we mainly augment textured vehicles onto images in our object detection evaluation, the evaluation results come from vehicles.

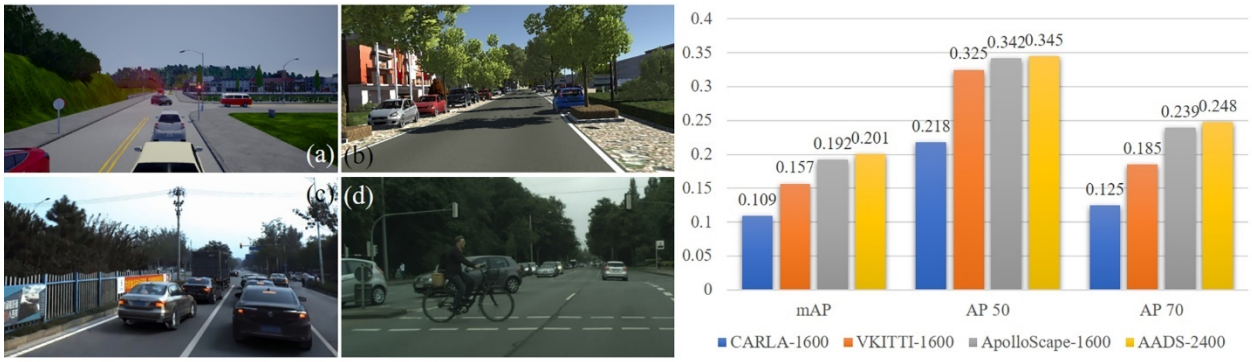

Fig. 5. RGB Image Augmentation Evaluations. The left four images are selected from CARLA (a), VKITTI dataset (b), our AADS-RGB dataset (c), and the testing dataset CityScapes (d), respectively. The right diagram is the evaluation results.

Synthetic data generation is an easy way to obtain large-scale datasets and has been proven to be effective in autonomous driving. However, the data statistics and distribution limit the capabilities of virtual data. When applying a model trained with synthetic data to real images, there is a domain gap. As our simulation method is built on realistic background, placement, and moving object synthesis, it will effectively reduce the domain problem. From images in Fig. 5, our method produces an image (c) that is more visually similar to a real image from CityScapes (d) than it is to the virtual-reality simulator CARLA (a) or the fully synthetic dataset VKITTI (b), i.e. images from our system may have small domain gap.

To quantitatively verify the effectiveness of our simulated data, we chose to train object detectors with our data and data from CARLA and VKITTI. The trained detectors are tested on the CityScape dataset, which has no overlap with any of the training sets.

We train models on CARLA-1600, VKITTI-1600, ApolloScape-RGB-1600, and AADS-RGB-2400 separately, where the suffix shows the number of images used for training. Then the object detection performance of the trained model is evaluated on the CityScapes validation set. Results are shown in Fig. 5 (right). It can be seen that, due to the domain gap, the metrics of ApolloScape-1600 are higher than those of VKITTI-1600 or CARLA-1600. Note that images in VKITTI are smaller than images in other datasets. We therefore apply the VKITTI-1600 model on downsampled CityScapes to make the comparisons fair. Otherwise, the VKITTI-1600 model tends to miss large cars, leading to a degradation in

detection performance. Adding 800 additional simulated images to ApolloScape-1600 (AADS-RGB-2400), our method improves the results by roughly one percentage point. This demonstrates that our simulation data may be closer to real-world data than data from virtual-reality.

2.4.2 Instance Segmentation with AADS-PC

To evaluate AADS point cloud simulations, we use the KITTI point cloud dataset (KITTI-PC), the ApolloScape point cloud (ApolloScape-PC), and our simulated point cloud (AADS-PC).

KITTI-PC: The KITTI point cloud dataset (*29*) consists of 7481 training and 7518 testing frames. These real point cloud frames are labeled corresponding to captured RGB images in the front view. This dataset provides evaluation benchmarks for 1) 2D object detection and orientation estimation, 2) 3D object detection, and 3) bird's eye view evaluations.

Based on those datasets, we evaluate our AADS system using 3D instance segmentation. It is a typical point cloud-based autonomous driving application, which simultaneously runs 3D object detection and point cloud segmentation. We utilize the state-of-the-art algorithm PointNet++ (*30*) to perform quantitative evaluation. The results are evaluated using a mean bounding box AP named mAP(Bbox) and a mean mask AP named mAP(mask).

dataset	mAP(Bbox)	mAP(mask)
16k sim	91.02	72.63
4k real	93.27	75.8
100k sim	94.1	75.32
16k real	94.41	79.21
100k real	94.39	80.55

(a)

methods	mAP(Bbox)	mAP(mask)
Random	28.68	65.96
Rule-based	39.43	74.32
Ours	44.14	82.47
Real data	46.57	86.33

(b)

dataset	car	pedestrian	cyclist	all
100k sim	91.02	72.63	57.21	83.03
16k real	93.27	75.80	62.62	86.33
100k sim +1.6k real	94.10	75.32	61.35	86.40
100k sim+16k real	94.41	79.21	67.01	88.31
100k real	94.39	80.55	67.30	88.50
100k sim+32k real	94.91	80.61	67.51	88.91
100k sim+100k real	95.27	81.20	68.25	89.30

(c)

Fig. 6. LiDAR Simulation Evaluations. (a) Evaluation on dataset's size and type (real or simulation) for real-time instance segmentation. (b) Evaluation results of different object placement methods. (c) Real data boosting evaluations (mean mask AP) using instance segmentation.

We evaluate the accuracy and effectiveness of the model trained by our simulation data and compare it with the models trained with manually labeled real data. These simulation and real data are randomly selected from the AADS-PC and ApolloScape-PC datasets, respectively. The mean AP evaluation results of the instance segmentation models are presented in Fig.6 (a). As shown in Fig. 6 (a), when trained with only our simulation data,

the instance segmentation models produce results competitive with the precisely labeled real data. When using 100k datapoints generated by simulation, segmentation performance is better than a model trained on 4k real datapoints, and comes close to models trained on 16k and 100k real datapoints. In short, by using simulation to increase the size of the training set, performance can approach that of models trained on real-world data.

Next, we use simulation data to boost the real data (i.e. pre-train the model), as shown in Fig.6 (c). Boosting with simulation data significantly improves (by 2% ~ 4%) the validation accuracy of the original model trained with only the real data. On the ApolloScape-PC dataset, we find that using 100k simulated datapoints a to pre-train the model and 1.6k real data for fine-tuning outperforms a model trained with 16k real datapoints. When fine-tuned with 32k real datapoints, the model surpasses a model trained on 100k real data. It means that our simulation approach could reduce up to 80% ~ 90% of manually labeled data, greatly reducing the need to label images, saving time and money. More details can be found in (*31*).

Finally, based on instance segmentation, we compare our object placement (traffic simulation) method with alternative placement strategies, e.g. placing object randomly or under specific rules (*32*). As shown in Fig.6 (c), the accuracy of models trained with simulated data outperform (by 4 ~ 7%) those trained with the other object placement strategies. The accuracy of models trained with our simulated data is close to that of a model trained on real data (gap of just 1~4%, depending on the application).

2.4.3 Traffic Predict with AADS-TRAJ

To evaluate the effectiveness of synthesized traffic, i.e. trajectories of cars, cyclist and pedestrians, we adopt Ma's TrafficPredict method (*33*) for quantitative evaluation. This method takes motion patterns of traffic-agents in the first T_{obs} frames as input and predicts their positions in the following T_{pred} frames. In our evaluation, T_{obs} and T_{pred} are set to 5 and 7, respectively. We extend 20k real frames from ApolloScape-TRAJ dataset with an additional 20k simulated frames from our AADS-TRAJ dataset to train the DNN proposed in Ma's method. Performance of the trained model is measured using mean Euclidean distance between predicted positions and ground truth. In our case, average displacement error (mean Euclidean distance of all predicted frames) and final displacement error (mean Euclidean distance of the T_{pred}-th predicted frame) are evaluated. The results in Fig.7 show that prediction error reduces sharply when training with an additional 20k simulated datapoints. The error rate for cars is reduced the most because cars are well represented in the simulated trajectories.

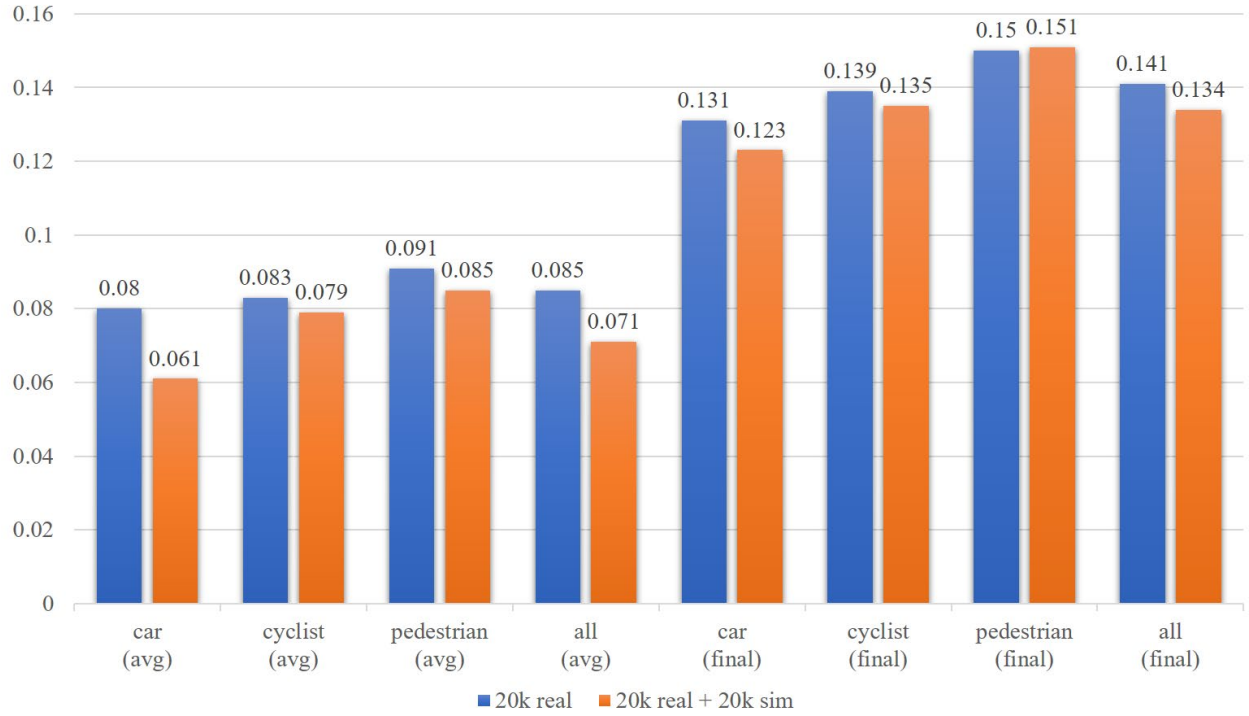

Fig. 7. Traffic Predict Evaluations. Comparison on trajectory prediction with 20k real trajectory frames and an additional 20k simulation trajectory frames.

3. Conclusions

In the previous section, we showed the effectiveness of AADS for various tasks in autonomous driving. All of these tasks are achieved by using captured scene data (location specific) and traffic trajectory data (general). The entire AADS system requires very little human intervention. The system can be used to generate a virtually limitless amount of realistic training data with fine annotation or it can be used in-line to simulate the entire AD system from perception to planning. The realism and scalability of AADS make it possible to be widely used in any real-world scenarios, as long as the background can be captured.

Compared to VR-based simulations, AADS's viewpoint change for RGB data is limited. Deviating too much from the original captured viewpoints leads to degraded image quality. However, we believe the limited viewing range is actually acceptable for AD simulation. For the most part, a vehicle drives on flat roads and the possible viewpoint changes are limited to rotation and 2D translation on the road plane. There is no need to support a bird's-eye view or a third person perspective for RGB-based perception. Another major limitation of AADS is the lack of lighting/environmental changes (snow/rain) in the scene. For now, these must be captured, but there have been significant advances in image synthesis using generative adversary networks (GANs) (*34*, *35*). Preliminary results synthesizing seasonal changes have been demonstrated. We believe that enabling full lighting/environmental effect synthesis within AADS is a promising direction and we are actively pursuing it now.

4. Materials and Methods

4.1 Data Preprocessing

The mind behind our AADS utilizes the scanned real images for simulation. Our goal is to simulate new vehicles and pedestrians in the scanned scene with new trajectories. To achieve this goal, before simulating data, our AADS should remove moving objects, e.g., vehicles and pedestrians, from scanned RGB images and point clouds. However, automatic detection and removal of moving objects constitutes a full research topic in its own right. Fortunately, most recent datasets provide semantic labels of RGB images, including point cloud. In our case, by utilizing semantic information in the ApolloScape dataset, we remove objects of a specific type, e.g., cars, bicycles, trucks, and pedestrians. After removing moving objects, numerous holes in both RGB images and point clouds appear, which should be carefully filled to generate a complete and clean background for AADS. We utilize the most recent RGB image inpainting method (*36*) to close the holes in the images. This method uses the semantic label to guide a learning-based inpainting technique, which achieves acceptable levels of quality in our case. The point cloud completion will be later introduced in the depth processing for novel background synthesis (Section 4.2).

Given synthesized background images, we could place any 3D CG model on the ground and then render it into the image space to create a new, composite simulation image. However, to make the composite image photo-realistic (look close to the real image), we should first estimate the illumination in background images. This enables our AADS to render 3D CG models with consistent shadows on the ground and on vehicle bodies. We solve such outdoor lighting estimation problems according to the method in (*37*). In addition, to further improve the reality of composite images, our AADS also provides an optional feature to enhance the appearance of 3D CG models by grabbing textures from real images. Specifically, given an RGB image with unremoved vehicles, we retrieve the corresponding 3D vehicle models and align those models to the input image using the method in (*38*). Similar to (*39*), we then use a symmetric priors to transfer and complete the appearance of 3D CG models from aligned real images.

4.2 Augmented Background Synthesis

Given a dense point cloud and image sequence produced from automatic scanning, the most straightforward way to build virtual assets for an autonomous driving simulator is to reconstruct the full environment. This line of work focuses on utilizing geometry and texture reconstruction methods to produce complete large-scale 3D models from the captured real scene. However, these methods cannot avoid hand editing while modeling, which is expensive in terms of time, computation and storage.

In this article, we propose a method to directly synthesize an augmented background in a specific view as needed during simulation. Our method avoids modeling the full scene ahead of running the simulation. Technically, our method creates such a scan-and-simulate system by utilizing the novel view synthesis technique.

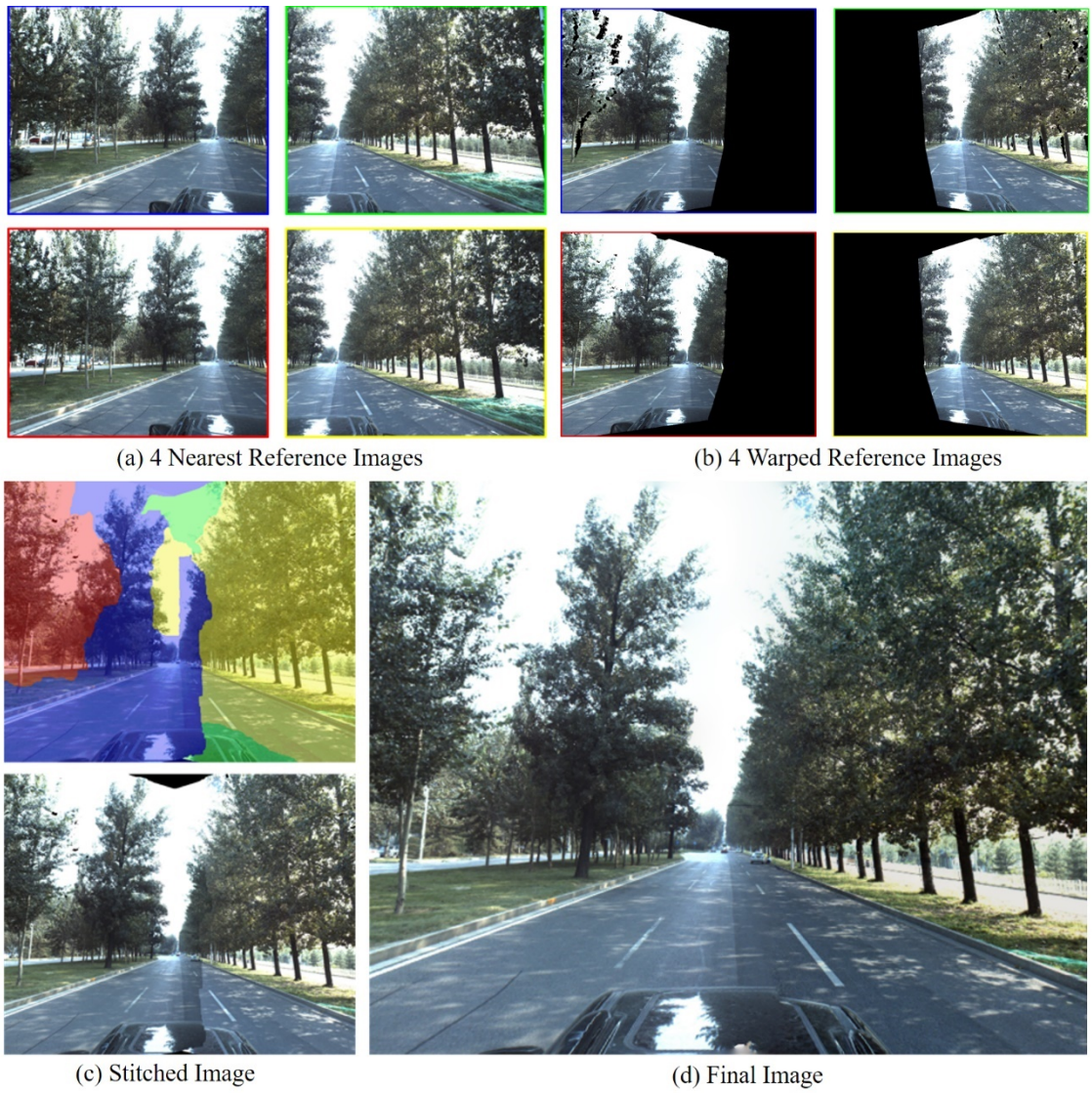

Fig. 8. Novel View Synthesis Pipeline. (a) The four nearest reference images are used to synthesize the target view in (d). (b) The four reference images are warped into the target view via depth proxy. (c) A stitching method is used to yield a complete image. (d) Final results in the novel view are synthesized after postprocessing, e.g., hole filling and color blending.

To synthesize a target view, we need to first obtain dense depth maps for input reference images. Ideally, these depth maps should be extracted from a scanned point cloud. Unfortunately, such depth maps will be incomplete and unreliable. In our case, the problems come with scanning: (a) The baseline of our stereo camera is too small compared to the size of street-view scenes, and consequently, there will be too few data points for objects that are too far from the camera. (b) The scenes are full of numerous moving vehicles and pedestrians that need to be removed. Unfortunately, their removal will produce holes in scanned point clouds; (c) The scenes are always complicated (e.g. many buildings are fully covered with glasses), which leads scanning sensor failed and thus scanned point clouds incomplete. We introduce a two step procedure to address the lack of reliability and incompleteness in depth maps: depth filtering and depth completion.

With respect to depth filtering, we carry out a judiciously selected combination of pruning filters. The first pruning filter is a median filter: a pixel is pruned if its depth value is sufficiently different from the median filtered value. To prevent removing thin structures, the kernel size of the median filter is set to small (e.g. 5 * 5 in our implementation). Then, a guided-filter (*40*) is applied to keep thin structures and to enhance edge alignment between the depth map and the color image. After getting a much more reliable depth, we complete the depth map by propagating the existing depth value to the pixels in the holes by solving a first-order Poisson equation similar to the one used in colorization algorithms (*41*).

After depth filtering and completion, reliable dense depth maps are produced that can provide enough geometry information to render an image into virtual views. Similar to (*23*), given a target virtual view, we select the four nearest reference views to synthesize the virtual view. For each reference view, we first use the forward mapping method to produce a depth map using camera parameters of the virtual view and then perform a depth inpainting to close small holes. Then a backward mapping method and occlusion test are used to warp the reference color image into the target view.

A naïve way to synthesize the target image is to blend all the warped images together. However, when we blend the warped images using the view angle penalty following article (*23*), there always exist obvious artifacts. Thus, we solve this problem as an image stitching problem rather than direct blending. Technically, for each pixel x_i of the synthesized image in the target virtual view, it is optimized to choose a color from one of those warped images. This can be formulated as a discrete pixel labeling energy function:

$$\arg\min_{\{x_i\}} \sum_i \lambda_1 E_1(x_i) + \lambda_2 E_2(x_i) + \sum_{(i,j)\in N} \lambda_3 E_3(x_i, x_j) + \lambda_4 E_4(x_i, x_j) + \lambda_5 E_5(x_i, x_j) \quad (1)$$

Here, x_i is the $i\text{-}th$ pixel of the target image and N is the pixel set of x_i's one ring neighbor. $E_1(x_i)$ is the pixel-wise data term, which is defined by extending the view angle penalty in (*42*). In contrast to the scenarios in (*42*), depth maps of the street-view scene always contain pixels with large depth values, which leads the angle view penalty to be too small. To address this problem, when the penalty is close to zero, we add another term to help choose the appropriate image by taking advantage of camera position information. Specifically, $E_1(x_i)$ is defined as $E_1(x_i) = E_{angle}(x_i)W_{label}(x_i)$. Here, $E_{angle}(x_i)$ is the view angle penalty in (*42*). When $E_{angle}(x_i)$ is too small, it will be hooked and set to 0.01 in our implementation. This is done to balance two energy terms and make $W_{label}(x_i)$ effective. $W_{label}(x_i)$ is defined as $W_{label}(x_i) = D_{pos}(C_{x_i}, C_{syn})D_{dir}(C_{x_i}, C_{syn})$, which evaluates the difference between the reference view and the target view. Here, C_{x_i} and C_{syn} denote the view choice for the camera for pixel x_i and for the target view's camera, respectively. Further, D_{pos} represents the distance from the camera center and D_{dir} is the angle between the optical axes of the two cameras.

$E_2(x_i)$ is the occlusion term used to exclude the occlusion areas while minimizing the pixel labeling energy. Most occlusions appear near depth edges. Thus, when using the backward mapping method to render the warped images, we detect occlusions by performing depth testing. All pixels in the reference view with larger depth values than

those of the source depth can yield an occlusion mask, which is then used to define $E_2(x_i)$. Specifically, when an occlusion mask is invalid, i.e. the pixel is non-occlusion, we set $E_2(x_i) = 0$ to add no penalty into the energy function. When a pixel is occluded, we set $E_2(x_i) = \infty$ to exclude this pixel completely.

The rest of the terms in Eq.1 are smoothness terms: color term $E_3(x_i, x_j)$, depth term $E_4(x_i, x_j)$, and color gradient term $E_5(x_i, x_j)$. Similar to (*43*), the color term $E_3(x_i, x_j)$ is defined by a truncated seam-hiding pairwise cost first introduced in (*44*): $E_3(x_i, x_j) = min(\| c_i^{x_i} - c_i^{x_j} \|^2, \tau_c) + min(\| c_j^{x_i} - c_j^{x_j} \|^2, \tau_c)$, where $c_i^{x_i}$ is the RGB value of pixel x_i. Similarly, the depth term $E_4(x_i, x_j)$ is defined as: $E_4(x_i, x_j) = min(| d_i^{x_i} - d_i^{x_j} |, \tau_d) + min(| d_j^{x_i} - d_j^{x_j} |, \tau_d)$, where $d_i^{x_i}$ is the depth of pixel x_i and the truncation thresholds are set to τ_c= 0.5, τ_d = 5m in our implementation. Because the illumination difference may occur between different reference images, the color difference is not sufficient to ensure a good stitch. An additional gradient difference $E_5(x_i, x_j)$ is used. By assuming that the gradient vector should be similar on both sides of the seam, we define $E_5(x_i, x_j) = | g_i^x - g_j^x | + | g_i^y - g_j^y |$, where g_i^x is a color space gradient of the i_{th} pixel in the image and includes x_i.

The term weights in Eq. 1 are set to $\lambda_1 = 200, \lambda_2 = 1, \lambda_3 = 200, \lambda_4 = 100, and\ \lambda_5 = 50$. The labeling problems are solved using the TRW-S method (*45*). Fig. 8 shows the pipeline and results of augmented background synthesis. Note that in Fig. 8 (c), a color difference may exist near the stitching seams after image stitching. To obtain consistent results, a modified Poisson image blending method (*46*) is performed. Specifically, we select the nearest reference image as the source domain and then fix its edges to propagate color brightness to the other side of the stitch seams. After solving the Poisson equation, we obtain the fusion result shown in Fig. 8 (d). Note that when the novel view is far from the input views, e.g., large view extrapolation, there will be artifacts due to dis-occluded regions that cannot be filled in by stitching together pieces of the input images. We mark those regions as holes and set their gradient value to zero. Thus, these holes will be filled with plausible blurred color when solving the Poisson equation.

4.3 Moving Objects Synthesis and Data Augmentation

With a synthesized background image in the target view, a complete simulator should have the ability to synthesize realistic traffics with diverse moving objects (e.g., vehicles, bicycles, pedestrians) and produce corresponding semantic labels and bounding boxes in simulated images and LiDAR point clouds.

We use the data-driven method described in (*26*) to address challenges involving traffic generation and placement of moving objects . Specifically, given localization information, we first extract lane information from an associated HD map. Then we randomly initialize the moving objects' positions within lanes and ensure that the directions of moving objects are consistent with the lanes. We use agents to simulate the moving objects' movements under constraints such as avoiding collisions and yielding to pedestrians. The multi-agent system is iteratively deduced and optimized using previously captured traffics following a data-driven method. Specifically, we estimate motion states from our real-world trajectory dataset ApolloScape-TRAJ; these motion states include position, velocity, and control

direction information of cars, cyclists, and pedestrians. Note that such real dataset processing is performed in advance of simulation and need be processed just once. During simulation runtime, we use an interactive optimization algorithm to make decisions for each agent at each frame of the simulation. In particular, we solve this optimization problem by choosing a velocity from the datasets that tends to minimize our energy function. The energy function is defined based on the locomotion or dynamics rules of heterogeneous agents, including continuity of velocity, collision avoidance, attraction, direction control, and other user-defined constraints.

With generated traffic, i.e. the object placement in each simulation frame, we render 3D models into the RGB image space and generate annotated data using the physical rendering engine PBRT (*47*). Meanwhile, we also generate a corresponding LiDAR point cloud with annotations using the method introduced in the next section.

4.4 LiDAR Synthesis

Given 3D models and corresponding placement, it is relatively straightforward to synthesize LiDAR point clouds with popular simulators such as CARLA(*4*). Nevertheless, there are opportunities to take advantage of specific LiDAR sensors (e.g., Velodyne HDL-64E S3). We propose a realistic point cloud synthesis method by effective modeling the specific LiDAR sensor following a data-driven fashion. Technically, a real LiDAR sensor captures the surrounding scene by measuring the time of flight for pulses of each laser beam(*48*). One laser beam is emitted from the LiDAR and then reflected from target surfaces. A 3D point is then generated if the returned pulse energy of a laser beam is big enough. We model the behavior of laser beams to simulate this physical process. Specifically, the emitted laser beam could be modeled using parameters including the vertical and azimuth angles and their angular noises, as well as the distance measurement noise. For example, the Velodyne HDL-64E S3 LiDAR sensor emits 64 laser beams in different vertical angles ranging from -24.33° to 2°. During data acquisition, HDL-64E S3 rotates around its own upright direction and shoots laser beams at a predefined rate to accomplish 360° coverage of the scenes. Ideally, such model can adaptive to other types of LiDAR sensors. Model parameters should depend on the specific type of sensor. However, we found experimentally that parameters vary considerably, even among devices of the same type. To be as close as possible to reality, we fit the model from real point clouds to statistically derive those parameters.

Specifically, we collect real point clouds from these HDL-64E S3 sensors on top of parked vehicles, guaranteeing smoothness of point curves from different laser beams. The points of each laser beam are then marked manually and fit by a cone with the apex located in the LiDAR center. The half-angle of the cone minus $\pi/2$ forms the real vertical angle, while the noise variance is determined from the deviation of lines constructed by the cone apex and points from the cone surface. The real vertical angles usually differ from ideal angles by 1°-3°. In our implementation, we model the noise with Standard Gaussian Distribution, setting the distance noise variance to 0.5cm and the azimuth angular noise variance to 0.05°.

To generate a point cloud, we have to compute intersections between the laser beams and the scene. Specifically, we propose a cubed, map-based method to mix the background of the scenes in the form of points and meshes of CAD models. Instead of computing intersections between beams and the mixed data, we compute the intersection with the projected maps (e.g., depth map) of scenes, which offer the equivalent information but in a

much simpler form. Note that our LiDAR simulation method can be easily extended for arbitrary LiDAR sensors and to any sensor solution for different numbers and poses of sensors. Fig. S1 shows the visual results of our LiDAR simulation.

References and Notes

1. N. Kalra, S. M. Paddock, Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability? *Transportation Research Part A: Policy and Practice*. **94**, 182–193 (2016).

2. A. Gaidon, Q. Wang, Y. Cabon, E. Vig, Virtual Worlds as Proxy for Multi-Object Tracking Analysis. *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition* (2016).

3. M. Müller, V. Casser, J. Lahoud, N. Smith, B. Ghanem, Sim4CV: A photo-realistic simulator for computer vision applications. *Int. J. Comput. Vis*. **126**, 902–919 (2018).

4. A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, V. Koltun, CARLA: An Open Urban Driving Simulator. *Proceedings of the 1st Annual Conference on Robot Learning*, 1–16 (2017).

5. S. Shah, D. Dey, C. Lovett, A. Kapoor, AirSim: High-fidelity visual and physical simulation for autonomous vehicles. *Field and Service Robotics*. **5**, 621–635 (2018).

6. NVIDIA, *NVIDIA drive constellation: virtual reality autonomous vehicle simulator* (2017).

7. A. C. Madrigal, Inside Waymo's Secret World for Training Self-Driving Cars. *the Altantis* (2017).

8. M. Likhachev, D. Ferguson, Planning long dynamically feasible maneuvers for autonomous vehicles. *The International Journal of Robotics Research*. **28**, 933–945 (2009).

9. S. J. Anderson, S. C. Peters, T. E. Pilutti, K. Iagnemma, Design and development of an optimal-control-based framework for trajectory planning, threat assessment, and semi-autonomous control of passenger vehicles in hazard avoidance scenarios. *Robotics Research*. **70**, 39–54 (2011).

10. C. Katrakazas, M. Quddus, W.-H. Chen, L. Deka, Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions. *Transportation Research Part C: Emerging Technologies*. **60**, 416–442 (2015).

11. J. Ziegler *et al.*, Making Bertha drive - An autonomous journey on a historic route. *IEEE Intelligent Transportation Systems Magazine*. **6**, 8–20 (2014).

12. A. Geiger *et al.*, Team AnnieWAY's entry to the 2011 grand cooperative driving challenge. *IEEE Transactions on Intelligent Transportation Systems*. **13**, 1008–1017 (2012).

13. M. Buehler, K. Iagnemma, S. Singh, The DARPA urban challenge: autonomous vehicles in city traffic. **56** (2009).

14. A. Best, S. Narang, D. Barber, D. Manocha, Autonovi: Autonomous vehicle planning with dynamic maneuvers and traffic constraints. *Proceedings of the 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2629–2636 (2017).

15. S. D. Pendleton *et al.*, Perception, planning, control, and coordination for autonomous vehicles. *Machines*. **5**, 6 (2017).

16. M. Johnson-Roberson *et al.*, Driving in the matrix: Can virtual worlds replace human-generated annotations for real world tasks? *arXiv preprint arXiv:1610.01983* (2016).

17. S. R. Richter, V. Vineet, S. Roth, V. Koltun, Playing for data: Ground truth from computer games. *Proceedings of the 2016 European Conference on Computer Vision*, 102–118 (2016).

18. H. Lin *et al.*, Semantic Decomposition and Reconstruction of Residential Scenes from LiDAR Data. *ACM Trans. Graph.* **32** (2013).

19. H. A. Alhaija, S. K. Mustikovela, L. Mescheder, A. Geiger, C. Rother, Augmented reality meets computer vision: Efficient data generation for urban driving scenes. *Int. J. Comput. Vis*. **126**, 961–972 (2018).

20. X. Huang *et al.*, The ApolloScape Dataset for Autonomous Driving. *arXiv preprint arXiv:1803.06184* (2018).

21. E. Penner, L. Zhang, Soft 3D reconstruction for view synthesis. *ACM Trans. Graph.* **36**, 235 (2017).

22. F. Liu, M. Gleicher, H. Jin, A. Agarwala, Content-preserving warps for 3D video stabilization. *ACM Trans. Graph.* **28**, 44 (2009).

23. G. Chaurasia, S. Duchene, O. Sorkine-Hornung, G. Drettakis, Depth synthesis and local warps for plausible image-based navigation. *ACM Trans. Graph.* **32**, 1–12 (2013).

24. R. Achanta *et al.*, SLIC superpixels compared to state-of-the-art superpixel methods. *IEEE Trans. Pattern Anal. Mach. Intell*. **34**, 2274–2282 (2012).

25. J. Ren *et al.*, Heter-Sim: Interactive data-driven optimization for simulating heterogeneous multi-agent systems. *arXiv preprint arXiv:1812.00307* (2018).

26. Q. Chao, Z. Deng, J. Ren, Q. Ye, X. Jin, Realistic Data-Driven Traffic Flow Animation Using Texture Synthesis. *IEEE Trans. Vis. Comput. Graph*, 1167–1178 (2018).

27. M. Cordts *et al.*, The Cityscapes Dataset for Semantic Urban Scene Understanding. *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition* (2016).

28. K. He, G. Gkioxari, P. Dollár, R. Girshick, Mask r-cnn. *Proceedings of the 2017 IEEE International Conference on Computer Vision*, 2980–2988 (2017).

29. A. Geiger, P. Lenz, R. Urtasun, Are we ready for autonomous driving? the kitti vision benchmark suite. *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition*, 3354–3361 (2012).

30. C. R. Qi, L. Yi, H. Su, L. J. Guibas, Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Proceedings of the 2017 Advances in Neural Information Processing Systems*, 5099–5108 (2017).

31. J. Fang *et al.*, Simulating LIDAR Point Cloud for Autonomous Driving using Real-world Scenes and Traffic Flows. *arXiv preprint arXiv:1811.07112* (2018).

32. X. Yue, B. Wu, S. A. Seshia, K. Keutzer, A. L. Sangiovanni-Vincentelli, A lidar point cloud generator: from a virtual world to autonomous driving. *Proceedings of the ACM International Conference on Multimedia Retrieval*, 458–464 (2018).

33. Y. Ma *et al.*, TrafficPredict: Trajectory Prediction for Heterogeneous Traffic-Agents. *arXiv preprint arXiv:1811.02146* (2018).

34. P. Isola, J.-Y. Zhu, T. Zhou, A. A. Efros, Image-to-image translation with conditional adversarial networks. *arXiv preprint* (2017).

35. T.-C. Wang *et al.*, High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs. *Proceedings of the 2018 IEEE Conference on Computer Vision and Pattern Recognition* (2018).

36. Y. Song *et al.*, SPG-Net: Segmentation Prediction and Guidance Network for Image Inpainting. *arXiv preprint arXiv:1805.03356* (2018).

37. Y. Liu, X. Qin, S. Xu, E. Nakamae, Q. Peng, Light source estimation of outdoor scenes for mixed reality. *The Visual Computer*. **25**, 637–646 (2009).

38. M. Corsini, M. Dellepiane, F. Ponchio, R. Scopigno, in *Computer Graphics Forum* (Wiley Online Library, 2009), vol. 28, pp. 1755–1764.

39. N. Kholgade, T. Simon, A. Efros, Y. Sheikh, 3D object manipulation in a single photograph using stock 3D models. *ACM Trans. Graph.* **33**, 1–12 (2014).

40. K. He, J. Sun, X. Tang, Guided image filtering. *IEEE Trans. Pattern Anal. Mach. Intell*. **35**, 1397–1409 (2013).

41. A. Levin, D. Lischinski, Y. Weiss, Colorization using optimization. *ACM Trans. Graph.* **23**, 689–694 (2004).

42. C. Buehler, M. Bosse, L. McMillan, S. Gortler, M. Cohen, Unstructured lumigraph rendering. *Proceedings of the 28th annual conference on computer graphics and interactive techniques*, 425–432 (2001).

43. P. Hedman, S. Alsisan, R. Szeliski, J. Kopf, Casual 3D photography. *ACM Trans. Graph.* **36**, 1–15 (2017).

44. V. Kwatra, A. Schödl, I. Essa, G. Turk, A. Bobick, Graphcut textures: image and video synthesis using graph cuts. *ACM Trans. Graph.* **22**, 277–286 (2003).

45. V. Kolmogorov, Convergent tree-reweighted message passing for energy minimization. *IEEE Trans. Pattern Anal. Mach. Intell*. **28**, 1568–1583 (2006).

46. P. Pérez, M. Gangnet, A. Blake, Poisson image editing. *ACM Trans. Graph.* **22**, 313–318 (2003).

47. M. Pharr, W. Jakob, G. Humphreys, *Physically based rendering: From theory to implementation* (Morgan Kaufmann, 2016).

48. S. Kim, I. Lee, Y. J. Kwon, Simulation of a Geiger-mode imaging ladar system for performance assessment. *sensors*. **13**, 8461–8489 (2013).

Acknowledgments

Author contributions: The project was conceived by Rg. Y.. W. L. and Cw. P. developed the concept and systems. Jp. R. developed the trajectory synthesis framework. R. Z. and Qc. G. performed synthesized RGB image evaluations. Xy. H. helped collect RGB and point cloud datasets. J. F. and Fl. Y. performed synthesized LiDAR point cloud evaluations. Yx. M. helped collect the trajectories dataset and performed simulated trajectory evaluations. Gp. W., Ww. X., and Hj. G. discussed the results and contributed to the final manuscript. The paper was written by W. L., D. M., and Rg.Y..

Competing interests: The authors declare that they have no competing interests.

Data and materials availability: The RGB and point cloud datasets (ApolloScape-RGB and ApolloScape-PC) are hosted with the web link http://apolloscape.auto/scene.html. The trajectory dataset (ApolloScape-TRAJ) announced along with this paper can be freely downloaded through the link http://apolloscape.auto/trajectory.html. The high-resolution video can also be watched through https://youtu.be/NEt0d_hhb6k.

SUPPLEMENTARY MATERIALS

Figures

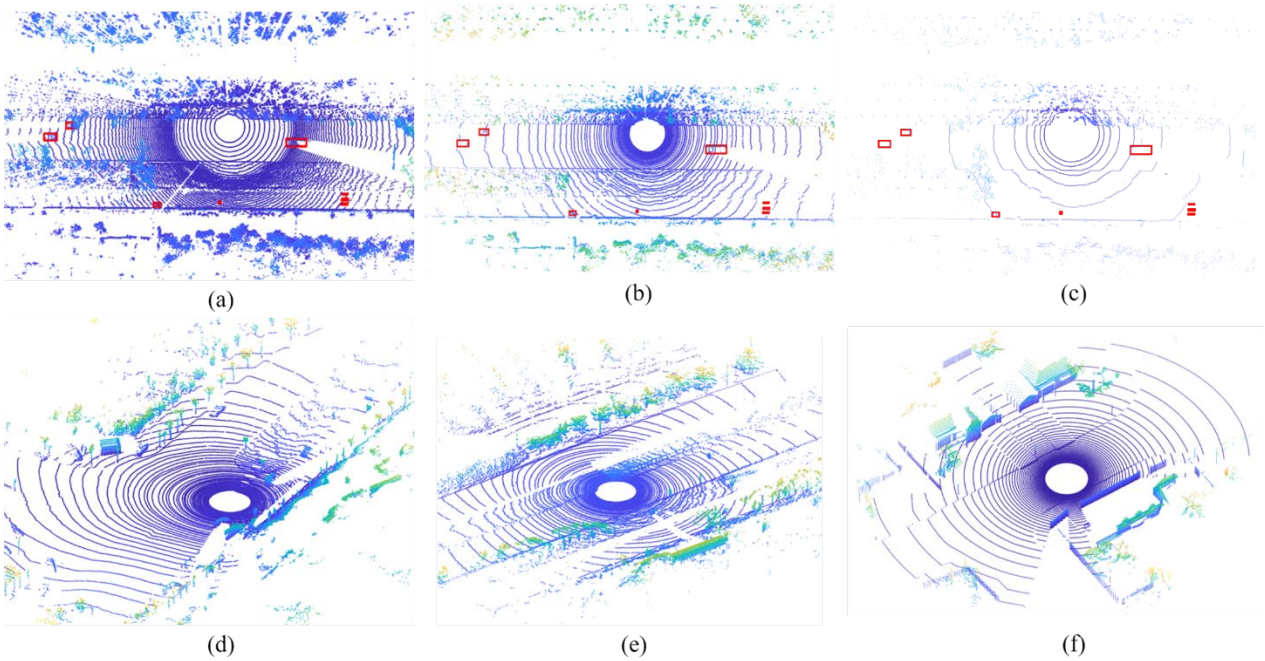

Fig. S1. Visual Evaluations of Point Cloud Simulation. Top row shows the ability of our system to simulate different types of LiDAR sensors: 128 channel VLS-128 (a), 64 channel HDL-64E (b), and 16 channel VLP-16 (c) sensor from Velodyne LIDAR, Inc. Note the red rectangles, which are projected 3D bounding boxes of cars and pedestrians. The bottom row shows the comparison of our method (e) for simulating 64 channel LiDAR with captured real data (d) and data simulated by CARLA (f).