2018 IEEE Intelligent Vehicles Symposium (IV)
Changshu, Suzhou, China, June 26-30, 2018

Rendering Physically Correct Raindrops on Windshields for Robustness Verification of Camera-based Object Recognition

Alexander von Bernuth, Georg Volk and Oliver Bringmann
University of Tübingen, {bernuth, volkg, bringmann}@informatik.uni-tuebingen.de

Abstract— **Recent developments in the field of autonomous cars indicate the appearance of those vehicles on the streets of every city in the near future. This urban driving requires zero error tolerance. In order to guarantee safety requirements self-driving cars and the used software have to pass exhaustive tests under as many different conditions as possible. The more versatile the considered influences and the more thorough the tests made under those influences, the safer the car will drive under real conditions. Unfortunately, it is very time and resource intensive to record the same test set of images over and over again, every time producing, or hoping for, specific conditions; especially when using real test vehicles. This is where environment simulation comes into play.**

This research investigates the simulation of environmental influences which may affect the sensors used in autonomous vehicles, in particular how raindrops resting on a windshield affect cameras as they may occlude large parts of the field of view. We propose a novel method to render these raindrops using Continuous Nearest Neighbor search leveraging the benefits of R-trees. The 3D scene in front of the camera, which is generated from stereo images, reflects physically correct in these drops. This leads to near photo-realistic simulated results. The derived images may be used to extend the training data sets used for machine learning without being forced to capture new real pictures.

I. INTRODUCTION

Self-driving cars are going to keep researchers and manufacturers all over the world occupied for a long time. This is because autonomous cars might dominate the public transport in the future, and to do so they have to be thoroughly verified. Unimaginable what could happen if a critical system like vehicle or pedestrian detection did not undergo the most comprehensive tests, covering every environmental influence imaginable and every possible combination of those – say, light rain on an overcast day at noon with back light and a soiled camera lens.

To hope to randomly encounter all of these condition combinations on test drives or even to plan for them to happen would be foolish. Additionally, according to analysts a car which carries all the tech to be tested would have to drive in the order of 10^9 km without an error to be qualified for ISO 26262 [1]. Furthermore, every time any part of the vehicle receives an update, all the tests are required to rerun.

So one of the main challenges faced by many manufacturers is the time intensive testing, especially the dispatching of a swarm of test drivers in prototype cars. One possible solution is to capture one relatively short drive and use this recording to generate a multitude of versions by simulating many different combinations of environmental influences.

While some research has been carried out on physically correct simulations of those environmental influences, the pool of unexplored weather conditions and effects is still large. Our own investigation lead to the conclusion that falling rain, when not extremely violent, does not affect algorithms like object or traffic sign detection. Therefore this paper investigates the rendering of raindrops resting on a windshield in front of the camera or the camera lens and the effects of these on object recognition algorithms, as those drops may obstruct huge parts of the visual field.

The first section of this paper will examine other research that focused on simulating influences like rain and dust in front of and on cameras. The second section is concerned with the drop generation, beginning at the scene reconstruction and ending with our method of ray tracing. After that the third section presents results and applications. In section four we discuss the findings and propose future work that would improve performance as well as realism.

This work has been partially funded by the Deutsche Forschungsgemeinschaft (DFG) in the priority program 1835 under grant BR2321/5-1.

II. RELATED WORK

Most research related to weather and environment simulation is focused on usage in games or rendering of synthetic scenes. Only few contributions take physical correctness into account. Until now, none of the works examined the effects of raindrops resting in front of the camera but described water drops very accurately in other contexts.

The work of Starik and Werman uses the assumption that the 3D-scene has the same depth everywhere [2]. They extract characteristics of rain in videos and apply the learned mask onto images without rain. Therefore they ignore the physical properties of a single raindrop.

Better results are possible when considering the exact depth of the scene, extracted from disparity maps [3], [4]. This is possible because the authors use stereo cameras when capturing their videos. They divide the space between the camera and the objects into small parts, calculate the correct amount of rain in these parts and render the raindrops in OpenGL. Hospach and Müller propose to treat the falling raindrops as very thin triangles, color them white and apply alpha blending according to the rain intensity. This trick allows for fast graphics card supported rendering. Because not needed – the differences would be minimal – they ignore effects like refraction.

Wang et al. use ray tracing to render the raindrops, but rely on the knowledge of the light source's exact position [5].

The authors shoot rays from the light source to all of the raindrops and calculate the pixel color according to the Phong illumination model. Objects that do not emit light are left out and will not refract in the raindrops.

One of the most similar works compared to this paper assumed a lot of simplifications in order to satisfy real time constraints [6]. The authors map the video image onto a single hemisphere in front of the camera, regardless of the real distance of the object to the camera. They then cast rays through each of the pixels occupied by raindrops and check, where in the hemisphere these rays hit the image. The corresponding pixel values are then used as texels when rendering the drops.

Many researchers investigating raindrops on a windshield have focused on the positioning, merging and moving of those. Different works concentrate on simulating fluid dynamics [7]–[11]. Extrand et al. examine the shape of water drops resting on inclined surfaces like glass plates [12]. More studies regarding the shape of falling water drops were made by Garg and Nayar [13]. Additionally they go out of their way to perfectly describe physical and optical properties of water drops [14]. A completely different approach was made by different researchers as they detect and remove raindrops in images instead of creating them [15], [16]. Much can be learned from the methods to remove the drops, as the described methods take advantage of some interesting physical properties.

Previously published studies are all limited to single aspects of raindrops, resting and falling, but neglect to consider the environment of the camera as 3D-scene. Either they ignore depth altogether or make broad assumptions, as well as overlook correct refraction of the scene objects in the water drops. Our approach takes all this into account. The results are near photo-realistic which is desired for training data sets that will be fed to neural networks.

Regarding rendering of point clouds there are many different existing methods. One is to create a mesh representing the surfaces and mapping images as textures onto them [17]. Others use heavy preprocessing to create multiple depth maps and refine the render step by step [18]. But even when meshing would be simple and cheap we would have to fill gaps that appear behind objects like trees or cars. If not fixed, those gaps could result in black areas in the drops as they might refract rays into those normally invisible areas.

Schaufler and Jensen shoot rays into the point cloud and search for point densities above a certain threshold [19]. There all points within a range are used for interpolation of surface normal and position. That way they avoid costly 3D reconstruction. Unfortunately holes in the surface may appear if not enough points are available or are not sampled densely enough. These holes would manifest as black areas and occur especially near edges of objects with very different depths, for example when looking at a traffic sign in front of the sky.

Another method for finding the nearest neighbor for a line segment was proposed by Adnoni et al. [20]. They use approximations when searching in high dimensions. This does not necessarily apply to our use case and involves complicated data structures that go too far for our three dimensions.

None of the works mentioned above combine both realistic rendering of raindrops, finding the corresponding objects in the scene without costly reconstructing the scene and checking the results with state of the art object detection algorithms. This paper will do all of the above and show that additionally training neural nets with modified pictures might give the edge to modern object recognition.

III. DROP GENERATION

To render the drops while accounting for refraction of objects in the scene it has to be reproduced in 3D as a point cloud. This process requires knowledge of the depth of the scene. A depth map, assigning each pixel its distance from the camera, may be calculated using epipolar geometry (resulting in a disparity map which corresponds to depth) or utilizing LiDAR or other sensors.

After the scene is calculated the raindrops on the windshield have to be modeled. We settled for a simple but powerful abstraction: every drop is represented by a sphere which is cut off by the windshield (a plane). This results in a sphere cap, a geometrically easy to handle object.

Then rays are cast through every image pixel covered by a drop and are refracted in those. The resulting rays are sent into the scene and the nearest neighbor in the point cloud is searched. This point's color contributes to the pixel color of the drop. Searching the nearest neighbor of a single point in a group of up to two million points might be trivial and fast, the same search with respect to a line in the search space makes it very exhausting. Only the use of Continuous Nearest Neighbour Search made this approach feasible.

A. Scene Reconstruction

The first step is the 3D-reconstruction of the scene in front of the camera. When working with images from (calibrated) stereo cameras it is possible to calculate the distance from the camera for each pixel using stereo reconstruction [21]. Other methods may include an extra sensor like LiDAR to capture the depth.

We are working with images from the Cityscapes Dataset [22]. It consists of several thousand pictures containing all kinds of street scenes, was captured using stereo cameras and ships with precomputed disparity maps. These can easily be converted into depth maps. Invalid depth values – results from pixels that could not be matched in the process above or were excluded on purpose – are assigned the maximum occurring valid depth.

Another source of images with corresponding depth maps used by us is the VIRES Virtual Test Drive software [23]. The depth maps are very exact because they are based on ground truth data. Additionally custom scenarios may be built very quickly using cars (user or computer controlled), pedestrians and other obstacles.

Our source of images used for object detection is the KITTI data set [24]. It contains stereo images, ground truth

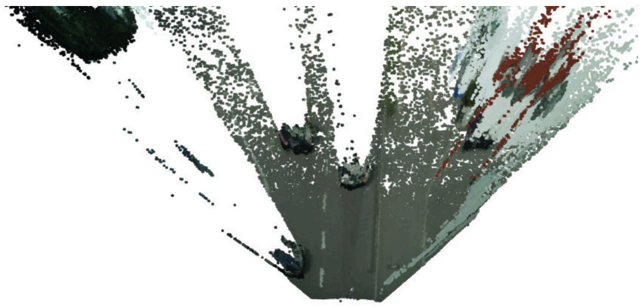

Fig. 1. Point cloud resulting from our 3D reconstruction. The source image may be seen in Fig. 5 as background. Zoomed in to the area close in front of the ego vehicle. The cars left and in front of the car as well as the road and its marks are clearly visible.

for objects and intrinsic and extrinsic camera parameters. Depth maps had to be calculated using the OpenCV implementation of Hirschmüller's matching algorithm [25].

Combining depth and color information we position the camera on 3D-position $c = (0,0,0)^T$ looking at $(0,0,1)^T$ and create vectors pointing from c to every pixel in the image plane. The x-axis points right, the y-axis down and the z-axis into the scene. The image plane gets centered on the z-axis. Its z-value may be calculated from the image pixel height h and the cameras vertical field of view fov_v as

$$z_{\text{image plane}} = 0.5 \cdot h \cdot \tan(\mathrm{fov}_v/2)\,. \tag{1}$$

We then convert all values from pixel to meter using the size of one physical pixel on the image sensor of the camera. Then all pixel vectors get normalized and scaled until they have a length matching the values of the depth map. The points may then be rotated about the x-axis to account for pitched cameras. The result – a 3D point cloud consisting of all pixels – is shown in Fig. 1. These points will represent the objects in the scene when rendering the raindrops.

B. Drop Model

Once the 3D representation is done, it is necessary to create and distribute the raindrops on the virtual windshield. A single drop is modeled as sphere cap. This cap is characterized by its center $\vec{c}$, radius r, height and normal $\vec{n}$ (pointing from the base to the dome). Generating drops means to set a fixed z-value (the distance from camera to windshield), randomly or deliberately set x- and y-coordinates, height and radius for each drop. As height and radius correlate one value might be derived from the other [26]. We chose $h = \tan(\theta/2) \cdot d$, where h denotes the drop height, θ the contact angle of the drop and d the drop diameter. Contact angles θ were determined to be around 87° when looking at drops with radii in the order of 2 mm by Park et al. [27]. Respective normals simply point in the same direction as the camera view vector does. When tilting the windshield forward, all drop centers and normals are rotated about the x-axis, too.

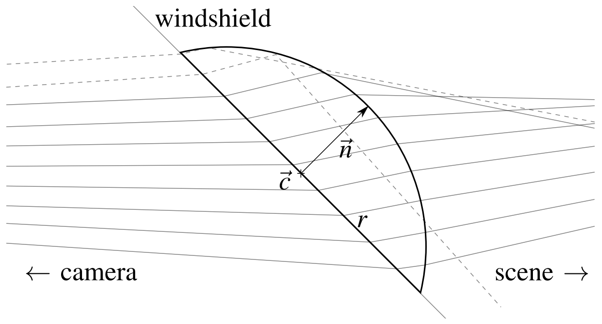

Fig. 2. Model of all refraction steps happening in every raindrop for every ray. Shows refraction (solid rays) and total internal reflection (dashed rays). The reflected rays account for the dark portion on top of the drops, hitting the road. Our model ignores secondary reflections, exchanging minor color changes in the top region of the drop in favor for vast speed impromevents.

C. Refraction

Prior to refracting the scene in the drops we shoot rays from the camera to each pixel in the image plane. To reduce the computational work only rays hitting the back of a drop – a circle with center, radius and normal – are tracked. All other rays are discarded and the pixel values from the original image get used for the result image.

After hitting the back of the drop, we refract the ray according to Snell's law. For more information on refraction see established literature [28]. The refracted ray is now inside the drop and about to exit it somewhere on the cap's surface.

We then calculate the sphere's center – the sphere which the cap originates from – by generating random vectors that are perpendicular to the normal and as long as the drop radius. These lay in the windshield plane. Together with the normal scaled to radius length we now have four points which lay on the sphere. Those are sufficient to calculate the sphere's center [29].

Now it is possible to find the point where our ray will leave the sphere [30]. Connecting this point with the sphere center results in the surface normal needed for the second refraction. In case of a total internal reflection the resulting ray does not get refracted any further. A visual representation of every refraction may be found in Fig. 2.

D. Nearest Neighbor Search

Next we are going to calculate what objects our rays come closest to in order to defer the pixel colors that rays correspond to. For that we continue with the search for the nearest neighbor (NN) for each ray in the point cloud generated in section III-A. That point's color will be assigned to the respective pixel and be visible in the drop.

A very naïve attempt for finding the NN is to look at every point for every ray, calculate the distance from point to ray and take the closest one. Even after organizing the points in R-trees, which allow for fast NN queries, sampling the ray and searching the NN for each sample, the search takes very long for even small images, not to mention the 2000×1000 pixel images from Cityscapes. For more details on how we sampled the ray see the appendix.

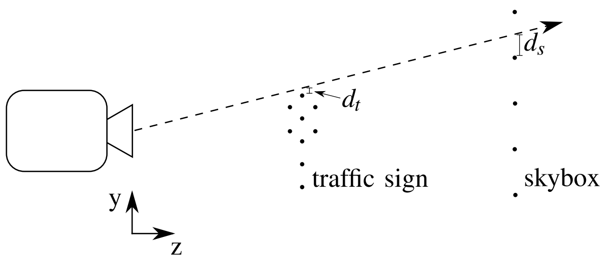

Fig. 3. Illustration of the z-value problem. When a ray should not be closest to a foreground object (here: traffic sign, distance d_t to the ray) but closest to a background object (here: sky 3D point, part of the background, distance d_s). This happens because further away objects have their points spread out more than closer objects, are shot "through" and therefore are ignored in the NN search.

Finally we chose Continuous Nearest Neighbor search [31] as it not only provides an exact nearest neighbor but does so in very short time (mere milliseconds) in large point clouds containing more than two million points. In preparation for the search all points are stored in a R-tree. Then the ray gets cut off at the planes with $z = 0$ and $z = z_{\max}$ (maximum z-coordinate of all points). The resulting line segments serve as input for the Continuous NN algorithm. It starts with initializing a "split list" (SL), which in the beginning contains the start and end point of a ray. Then the R-tree gets traversed depth first. Whenever a bounding volume lies closer to a split list element than its previous nearest neighbor (for more algebraic details on how to do this see [32]) it is put on a stack for further investigation. As the traversal comes across a point, the same check is made. If it lies closer to any point in the split list than that point's previous NN, that point and relevant SL neighbors are getting updated, new SL points are inserted or obsolete ones removed. As soon as the stack is empty, the SL entry which has the smallest NN distance gets chosen – it contains the NN of the whole ray.

Utilizing additional smart heuristics (which may be looked-up in the original paper [31]) a huge portion of the points may be ignored and NN search is blazing fast.

Because of a phenomenon displayed in Fig. 3, the z-value problem, we had to adapt the Continuous NN algorithm to our use case. When a ray barely misses an object in the 3D space, for example a traffic sign, it may be closer to that object than to the background, say, a pixel in the sky. This leads to frayed edges when not countered. To counteract on this problem we divide all distances in the SL by its corresponding z-value. Close NN are penalized by this and have to be really close to the ray in order to count as a hit.

IV. RESULTS

In order to evaluate the quality of our ray tracing algorithm we rendered scenes completely, ignoring raindrops and shooting rays through every pixel. This allows for detailed error detection. A difference image, resulting from subtraction the images and taking the absolute value, scaled by factor four for visibility, may be found in Fig. 4. The brighter

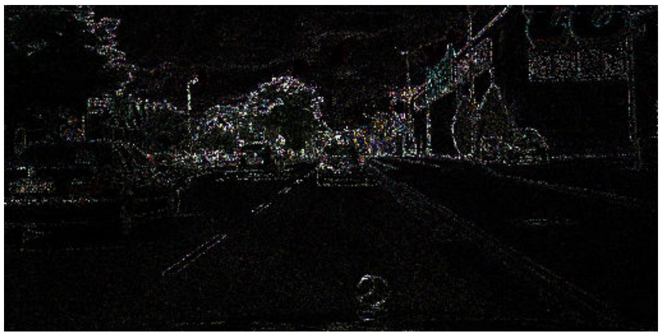

Fig. 4. Difference between the original image (see background in Fig. 5) and a complete render using our ray tracing algorithm in combination with the 3D reconstructed scene. The image shows the absolute difference of the two images, scaled by factor four to improve visibility. The only significant differences may be observed at the edges of two objects with clear depth disparity. Other minor changes may be traced back to depth map generation errors. The PSNR is 30.5 dB.

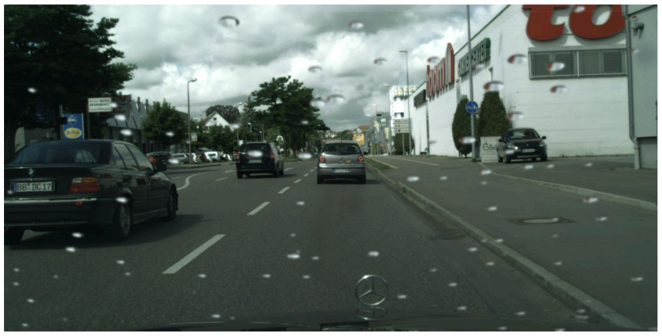

Fig. 5. Example image from the Cityscapes data set with rendered raindrops. The windshield was placed 30 cm in front of the camera and tilted to 27° from the horizontal. The drop radii have their mean at $\bar{r} = 1.5$ mm and a standard deviation of $\sigma = 0.4$ mm. The raindrops were convoluted with a disk kernel to simulate out of focus effects [33].

a pixel the higher the difference between the two images. Clearly the edges separating object of different depth show the largest discrepancy. This is due to the aforementioned z-value problem but by far better than without our correction for it.

As example for a render with raindrops we took an image from the Cityscapes data set, generated random drops and rendered those. The result may be found in Fig. 5.

To test the behavior of object recognition algorithms we applied our drop generation to one of the top ranked neural networks on the KITTI data set. One could expect that large raindrops might occlude small objects like pedestrians or mask parts of vehicles in ways that could confuse neural nets that were trained on perfect images.

The network we used was the Recurrent Rolling Convolution (RRC) network, which is a single stage object detection network [34]. It is currently ranked fourth place on the KITTI data set. We reconstructed the 3D scenes for the KITTI data set from 450 given stereo images with the corresponding camera parameters to then simulate the raindrops. Those images served as input for RRC network. As control group we use the same images without raindrops. An overview of this process may be found in Fig 6.

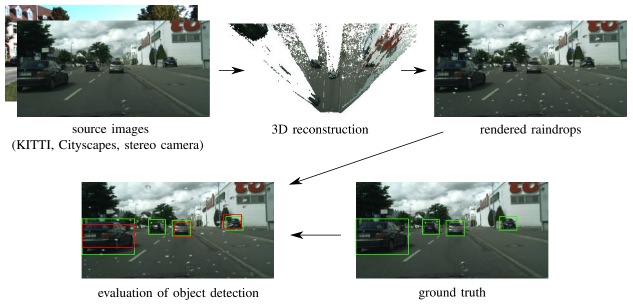
source images
(KITTI, Cityscapes, stereo camera)
3D reconstruction
rendered raindrops
evaluation of object detection
ground truth

Fig. 6. Pipeline for our object recognition evaluation. We take stereo images as input (from the KITTI data set), calculate the depth for each pixel, then reconstruct the 3D scene and render the raindrops. After that we run object detection on those images with raindrops and compare the results with the ground truth.

TABLE I
RESULTS FROM OUR OBJECT DETECTION TEST

	Overall mAP	Overall AA
Original KITTI images	81.79 %	89.23 %
KITTI images with drops	80.61 %	88.86 %
Decrease due to artificial drops	1.18 %	0.37 %

	mAP Cars	mAP Ped.
Original KITTI images	85.78 %	55.20 %
KITTI images with drops	84.92 %	51.20 %
Decrease due to artifficial drops	0.86 %	4.00 %

We evaluated the results by calculating the measured average precision (mAP) as used in [35]. For this the intersection over union metric for predicted and ground truth bounding boxes gets calculated. If the overlap of those boxes is greater than the specified threshold of 70 % the detection counts as true positive (TP).

Additionally we evaluate the average accuracy (AA), also known from [35]. Here the accuracy of the detection is evaluated by $\mathrm{AA} = \mathrm{TP}/(\mathrm{FP} + \mathrm{TP})$ where FP denotes the number of false positives. Our change to the evaluation of [35] is that we only count object detections as TP if the overlap of predicted and ground truth bounding box is greater than 70 % instead of the proposed 50 %. We also do not distinguish between different classes and don't use a lower pedestrian threshold of 50 % like suggested in [36]. Duplicate detections are counted as FP.

The results in table I show that the overall mAP decreases by 1.18 % as well as the overall average accuracy decreases by 0.37 % when raindrops are present in the images. Furthermore, we see that smaller objects like pedestrians are more vulnerable to occlusion by raindrops (decrease of 4 % in mAP) than bigger objects like cars. In a time where the top neural nets lie fractions of a percent apart regarding recognition rate, even smallest improvements as additionally training with our rainy images might give one the edge over the others.

V. CONCLUSIONS

This study set out to artificially create raindrops based on only a pair of stereo images. Physically correct refraction and reflection were included in our model as well as some simplifications that allowed for faster development. Nevertheless, the results are notable. We achieved photo realistic drops on a pitched windshield that reveal the whole range of objects behind them. Although being represented by sphere caps, our drops look lifelike.

Our ray tracing method produces near perfect renders of a 3D scene represented by a point cloud containing millions of points in acceptable time without costly precomputation of meshes and textures. Even thin and small objects like traffic signs do not get lost in the entirety of the scene; letters in license plates are readable. This all is possible while the 3D scene reconstruction relies on feature matching and disparity maps with discrete values.

Neural networks dealing with object detection that are additionally trained with images that are prepared with raindrops could achieve better results and surpass other competitors. This work could lead to better overall scores and robustness regarding sensor flaws for those machine learning algorithms.

To this date, our Continuous NN-search runs on up to 8 CPU cores (this is the highest number of threads our CPU, an Intel i7-7700K, provides). There, run times of about 3 min per KITTI image were achieved while rendering relatively large and many drops. The smaller and fewer the drops the faster the computation as we only trace rays that hit drops. Because the search is highly parallelizable one could implement the R-tree for GPU memory usage and leverage the power of graphics cards. This would be rewarded by vastly shorter execution times in the order of milliseconds per image.

An interesting application of this algorithm might be to use an image generated with it as input for the work of Iseringhausen et al. [37]. They use water drops on a glass plate in front of a scene as light field camera, because the drops act as multiple lenses in different positions. After reverse engineering the drop shapes they infer the scene behind the class plate and are able to create 3D renders from different perspectives. Our work represents the reverse – we infer the raindrops from our knowledge of the scene.

APPENDIX

To find the NN of a ray in a point cloud we first tried to sample the ray and find the NN for every sample point, in the end collecting the one with the smallest distance. This is how we sampled the ray.

Our point cloud is generated by looking at the disparity of every pixel in the images from a stereo camera. The disparity, standing for the pixel distance a feature has from one image to the other, is an integer ranging from 0 (feature did not move between the images, it is very distant) to "width of image" (feature did move from the left to the right side of the image, therefore is very close to the camera). This leads to discrete depth values and layers in which each set of pixels with equal disparity (and therefore equal depth) lies. We sampled our rays on each intersection with one of those depth layers. That way we assured that each sample point finds its NN on its layer, if possible. To accelerate rendering we even tested sub-sampling the ray, but that resulted in large patches of e.g. sky to have the same color.

References

[1] A. Weitzel, H. Winner, C. Peng, S. Geyer, F. Lotz, and M. Sefati, *Absicherungsstrategien fuer Fahrerassistenzsysteme mit Umfeldwahrnehmung*. Fachverlag NW in der Carl Schuenemann Verlag GmbH, 2014. [Online]. Available: https://trid.trb.org/view/1339882

[2] S. Starik and M. Werman, “Simulation of rain in videos,” *Texture Workshop, International Conference on Computer Vision, 2003 (ICCV)*, vol. 2, pp. 406–409, 2003. [Online]. Available: http://www.cs.huji.ac.il/~werman/Papers/Rain.pdf

[3] D. Hospach, S. Mueller, O. Bringmann, J. Gerlach, and W. Rosenstiel, “Simulation and evaluation of sensor characteristics in vision based advanced driver assistance systems,” *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pp. 2610–2615, 2014. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6958108

[4] D. Hospach, S. Mueller, W. Rosenstiel, and O. Bringmann, “Simulation of Falling Rain for Robustness Testing of Video-Based Surround Sensing Systems,” *Proceedings of the 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 233–236, 2016.

[5] C. Wang, M. Yang, X. Liu, and G. Yang, “Realistic Simulation for Rainy Scene,” *Journal of Software*, vol. 10, no. 1, pp. 106–115, 2015.

[6] T. Sato, Y. Dobashi, and T. Yamamoto, “A Method for Real-Time Rendering of Water Droplets Taking Into Account Interactive Depth of Field Effects,” in *Iwec 2002*, 2002, pp. 110–117.

[7] H. Wang, P. J. Mucha, and G. Turk, “Water drops on surfaces,” *ACM Transactions on Graphics*, vol. 24, no. 3, p. 921, 2005. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1073204.1073284

[8] K. Kaneda, Y. Zuyama, H. Yamashita, and T. Nishita, “Animation of Water Droplet Flow on Curved Surfaces,” *PACIFIC GRAPHICS*, pp. 50–65, 1996.

[9] K. Kaneda, S. Ikeda, and H. Yamashita, “Animation of water droplets moving down a surface,” *The Journal of Visualization and Computer Animation*, vol. 10, no. 1, pp. 15–26, 1999.

[10] S. Takenaka, Y. Mizukami, and K. Tadamura, “A Fast Rendering Method for Water Droplets on Glass Surfaces,” *The 23rd International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC)*, vol. 23, pp. 13–16, 2008.

[11] M. Jonsson and A. Hast, “Animation of Water Droplet Flow on Structured Surfaces,” *Special Effects and Rendering. Proceedings from SIGRAD 2002*, pp. 17–22, 2002.

[12] C. W. Extrand and Y. Kumagai, “Liquid Drops on an Inclined Plane: The Relation between Contact Angles, Drop Shape, and Retentive Force,” *Journal of Colloid and Interface Science*, vol. 170, no. 2, pp. 515–521, 1995. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0021979785711307

[13] K. Garg and S. K. Nayar, “Vision and rain,” *International Journal of Computer Vision*, vol. 75, no. 1, pp. 3–27, 2007.

[14] ——, “Photometric Model for Raindrops,” *Columbia University Technical Report*, 2003.

[15] S. You, R. T. Tan, R. Kawakami, Y. Mukaigawa, and K. Ikeuchi, “Raindrop detection and removal from long range trajectories,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9004, pp. 569–585, 2015.

[16] J. C. Halimeh and M. Roser, “Raindrop detection on car windshields using geometric-photometric environment construction and intensity-based correlation,” *IEEE Intelligent Vehicles Symposium, Proceedings*, pp. 610–615, 2009.

[17] M. Bolitho, M. Kazhdan, R. Burns, and H. Hoppe, “Multilevel streaming for out-of-core surface reconstruction,” *Proceedings of the fifth Eurographics symposium on Geometry processing*, pp. 69–78, 2007. [Online]. Available: http://portal.acm.org/citation.cfm?id=1282001

[18] M. Arikan, R. Preiner, and M. Wimmer, “Multi-depth-map raytracing for efficient large-scene reconstruction,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 22, no. 2, pp. 1127–1137, 2016.

[19] G. Schaufler and H. Jensen, “Ray tracing point sampled geometry,” *In Rendering Techniques 2000: 11th Eurographics Workshop on Rendering*, pp. 319—-328, 2000. [Online]. Available: https://www.cs.princeton.edu/courses/archive/spring01/cs598b/papers/schaufler00a.pdf

[20] A. Andoni, P. Indyk, R. Krauthgamer, and H. L. Nguyen, “Approximate line nearest neighbor in high dimensions,” *Symposium on Discrete Algorithms*, pp. 293–301, 2009. [Online]. Available: http://portal.acm.org/citation.cfm?id=1496770.1496803

[21] M. Sonka, V. Hlavac, and R. Boyle, *Image Processing, Analysis, and Machine Vision*, 2nd ed. Pacific Grove: Cengage Learning, 1998.

[22] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, “The Cityscapes Dataset for Semantic Urban Scene Understanding,” *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. [Online]. Available: http://arxiv.org/abs/1604.01685

[23] M. Dupuis and W. Karl, “VTD - VIRES Virtual Test Drive,” 2017. [Online]. Available: https://vires.com/vtd-vires-virtual-test-drive/

[24] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, “Vision meets robotics: The KITTI dataset,” *International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[25] H. Hirschmuller, “Stereo Processing by Semiglobal Matching and Mutual Information,” *Ieee Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 2, pp. 328–341, 2008.

[26] G. Bracco and B. Holst, *Surface science techniques*. Springer-Verlag Berlin Heidelberg, 2013, vol. 51, no. 1.

[27] J. Park, H. S. Han, Y. C. Kim, J. P. Ahn, M. R. Ok, K. E. Lee, J. W. Lee, P. R. Cha, H. K. Seok, and H. Jeon, “Direct and accurate measurement of size dependent wetting behaviors for sessile water droplets,” *Scientific Reports*, vol. 5, no. June, pp. 1–13, 2015. [Online]. Available: http://dx.doi.org/10.1038/srep18150

[28] P. Shirley, *Fundamentals of computer graphics*. Natick (Mass.): Peters, 2002.

[29] W. H. Beyer, *CRC handbook of mathematical sciences*. CRC press, 1987.

[30] D. H. Eberly, *3D game engine design: a practical approach to real-time computer graphics*. CRC Press, 2006.

[31] Y. Tao, D. Papadias, and Q. Shen, “Continuous Nearest Neighbor Search,” *VLDB ’02 Proceedings of the 28th international conference on Very Large Data Bases*, pp. 287–298, 2002.

[32] N. Roussopoulos, S. Kelley, and F. Vincent, “Nearest neighbor queries,” *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data - SIGMOD ’95*, pp. 71–79, 1995. [Online]. Available: http://portal.acm.org/citation.cfm?doid=223784.223794

[33] H. G. Dietz, “Out-of-focus point spread functions,” *Proceedings of SPIE*, vol. 9023, pp. 1–11, 2014. [Online]. Available: http://dx.doi.org/10.1117/12.2040490

[34] J. S. J. Ren, X. Chen, J. Liu, W. Sun, J. Pang, Q. Yan, Y.-W. Tai, and L. Xu, “Accurate Single Stage Detector Using Recurrent Rolling Convolution,” *CoRR*, vol. abs/1704.0, 2017. [Online]. Available: http://arxiv.org/abs/1704.05776

[35] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, “The Pascal Visual Object Classes (VOC) Challenge,” *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, jun 2010. [Online]. Available: https://doi.org/10.1007/s11263-009-0275-4

[36] A. Geiger, P. Lenz, and R. Urtasun, “Are we ready for autonomous driving? the KITTI vision benchmark suite,” *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 3354–3361, 2012.

[37] J. Iseringhausen, M. B. Hullin, B. Goldlücke, N. Pesheva, S. Iliev, A.-d. Wender, and M. Fuchs, “4D Imaging through Spray-On Optics,” *ACM Trans. Graph. Article*, vol. 36, no. 35, 2017. [Online]. Available: http://dx.doi.org/10.1145/3072959.3073589