Optimizing Image Acquisition Systems for Autonomous Driving

***Henryk Blasinski*[1], *Joyce Farrell*[1], *Trisha Lian*[1], *Zhenyi Liu*[1,2], *Brian Wandell*[1,3]**
[1] ***Department of Electrical Engineering, Stanford University, CA***
[2] ***Department of Automotive Engineering, Jilin University, China***
[3] ***Department of Psychology, Stanford University, CA***

Abstract

Task requirements for image acquisition systems vary substantially between applications: requirements for consumer photography may be irrelevant - or may even interfere - with requirements for automotive, medical and other applications. The remarkable capabilities of the imaging industry to create lens and sensor designs for specific applications has been demonstrated in the mobile computing market. We might expect that the industry can further innovate if we specify the requirements for other markets. This paper explains an approach to developing image system designs that meet the task requirements for autonomous vehicle applications. It is impractical to build a large number of image acquisition systems and evaluate each of them with real driving data; therefore, we assembled a simulation environment to provide guidance at an early stage. The open-source and freely available software (isetcam, iset3d, and isetauto) uses ray tracing to compute quantitatively how scene radiance propagates through a multi-element lens to form the sensor irradiance. The software then transforms the irradiance into the sensor pixel responses, accounting for a large number of sensor parameters. This enables the user to apply different types of image processing pipelines to generate images that are used to train and test convolutional networks used in autonomous driving. We use the simulation environment to assess performance for different cameras and networks.

Introduction

The market for image sensors in autonomous vehicles can be divided into two segments. Some image sensor data is used as images to the passengers, such as rendering views from behind the car as the driver backs up. Other image sensor data is used by computational algorithms that guide the vehicle; the output from these sensors is never rendered for the human eye. It is reasonable to expect that the optical design, sensor parameters, and image processing pipeline for these two systems will differ.

Mobile imaging applications for consumer photography dominate the market, driving the industry towards sensors with very small pixels (1 micron), a large number of pixels, a Bayer color filter array, and an infrared cutoff filter. There is a nascent market for image sensors for autonomous vehicle decision-system applications, and the most desirable features for such applications are not yet settled. The current offerings include sensors with larger pixels, a color filter array that comprises one quarter red filters and three quarters clear filters, and no infrared cutoff filter (e.g. ON Semiconductor; Omnivision). The requirements for optical properties, such as depth of field effects, may also differ between consumer photography and autonomous vehicles. Consumer photography values narrow depth of field images (bokeh), while autonomous driving value large depth of field to support detection and classification accuracy.

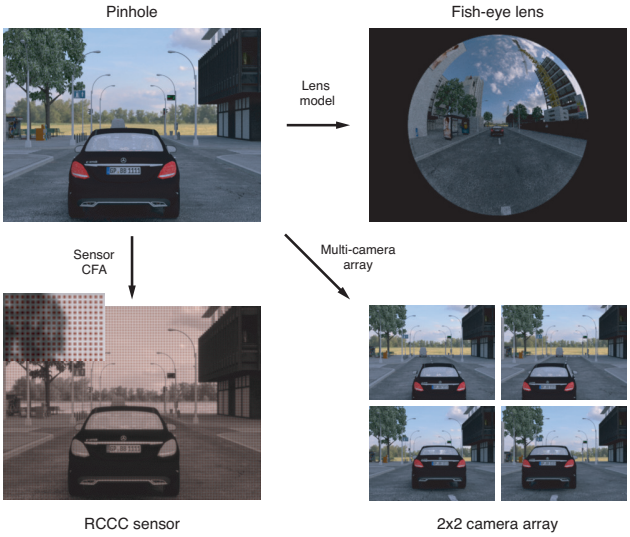


Figure 1: Variations in the camera architecture provide different images to object detection networks. The same scene is shown as measured by a pinhole camera (top left), a fish-eye lens (top right), a red-clear sensor (bottom left), and a 2×2 camera array (bottom right).

This is an early phase in the development of imaging systems for autonomous vehicles, and the space of potential camera designs for autonomous vehicles is quite large (Figure 1). The parameter choices range from color filter array, pixel size, optics and camera array format and camera control algorithms (e.g., auto-focus, auto-exposure). The possibilities in the space of convolutional neural networks (CNNs) for analysis is at least as large, and the network accuracy and resilience may depend on the type of input data. Exploring this space, spanning the joint dependency of hardware and network options using purely empirical methods, is impractical and a number of groups are exploring simulation tools that might be helpful. We describe an open-source and free implementation of software tools that can help assess the relationship between image capture devices and CNN performance.

Experimental methods

We implemented an open-source camera simulation environment to design and evaluate different image acquisition systems (Figure 2). The system is divided into two main github repositories in the iset[1] project on github. One repository includes methods for creating physically realistic spectral radiance distributions

[1] https://github.com/iset

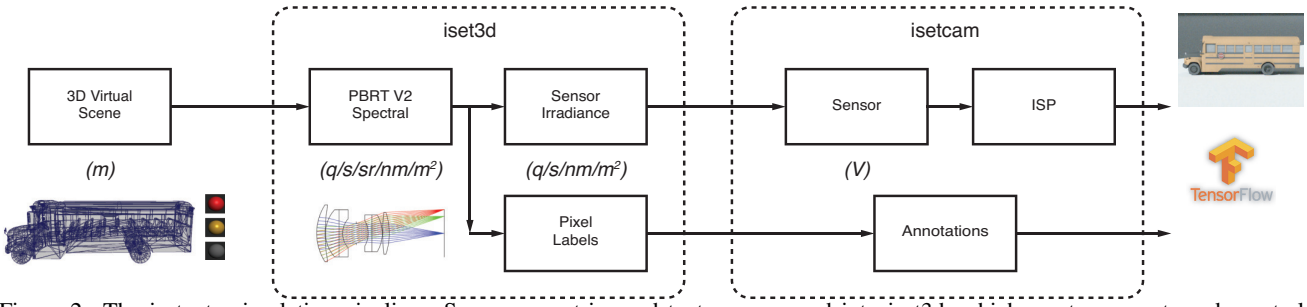


Figure 2: The isetauto simulation pipeline. Scene geometries and textures are read into iset3d, which creates a ray-traced spectral irradiance at the sensor. The iset3d methods use a platform-independent Docker container with PBRT to calculate the ray-traced irradiance and calculate pixel-level labels for objects and depths. The isetcam methods read the sensor irradiance and simulate the sensor responses using a model that accounts for pixel geometry, color filters, electrical noise, and an image processing pipeline. The simulated sensor values and pixel-level labels are used to train TensorFlow networks for object classification.

of complex driving scenes and uses ray tracing to compute quantitatively how scene radiance propagates through a multi-element lens to form the sensor irradiance (iset3d). The other repository includes methods for modeling imaging sensors, accounting for a large number of sensor parameters, in order to transform the irradiance into the sensor pixel responses (isetcam, formerly iset) [1].

The geometry and material properties of the scene are managed using methods in iset3d that organize the inputs to the Physically Based Ray Tracing (PBRT) software packaged in a platform-independent Docker container. PBRT uses geometric optics to compute how light rays propagate from light sources, interact with objects in the scene and arrive at the camera aperture [2]. PBRT includes a camera model that traces rays from the aperture through multi-element spherical lenses with each surface being defined by its curvature and wavelength-dependent indices of refraction [3]. Finally, we added a diffraction module that randomizes the ray angular direction by an amount that depends on the rays distance from the aperture [4]. The iset3d methods return an estimate of the spectral irradiance distribution at the image sensor plane.

The spectral irradiance is converted to a sensor response using isetcam[2]. The isetcam software contains a phenomenological model of the image sensor array. Given the spectral irradiance, isetcam calculates the predicted array of pixel responses, accounting for the sensor geometry and electrical characteristics. The isetcam repository includes image processing methods that can form part of an image processing pipeline that converts pixel responses to an sRGB image. Taken together, the iset3d and isetcam methods produce simulated camera images of a scene.

Finally, the simulated camera images are used for training and testing convolutional neural nets (CNNs) used in autonomous vehicle applications. The ground-truth labels for training and testing is provided by the iset3d methods. These methods label each image pixel for its depth, object label, and material property. The scripts controlling the CNN calculations for the automotive simulations in this paper are in a third repository, isetauto [3], that relies on iset3d and isetcam.

[2]https://github.com/ISETCAM/isetcam/wiki
[3]https://github.com/isetcam/isetauo

Table 1: General camera parameters.

Parameter	Value
Resolution	640×480
Pixel size	3 µm
Sensor diagonal	2.42 mm
Dark voltage	1 mV s^{-1}
Read noise	1 mV
Voltage swing	1 V
Field of view	44 deg
Lens model	pinhole
Bit depth	8 bits

Scene generation

We explored the influence of camera design for city driving using a background environment of a city model spanning four blocks, each approximately 100×100 meters. The blocks were visually distinct, comprising a variety of different buildings, plants, and other typical city objects (lamps, benches, mailboxes). We added test objects (cars, people and buses) from three different categories at semantically meaningful locations within the city blocks: Cars were placed on the streets, pedestrians on or near the sidewalk. For this project we had 12 car models, 24 pedestrian models and 2 bus models. Each model is scaled to approximately correct dimensions, and we generated five arrangements for each of the four blocks, producing 20 distinct city scenes. By varying camera placement, we generated several thousand images.

Camera

We simulated a camera using the specifications of typical cameras for autonomous driving applications (Table 1; [5]). Automotive sensors have significantly larger pixels and lower sampling resolution compared to those used in mobile imaging. The reduced resolution is motivated in part to reduce the computational complexity of real-time automotive calculations, and also because majority of neural networks are designed to process low resolution images at their inputs. Using larger pixel size also increases sensitivity, well-capacity, and improves low light performance.

Neural networks

We trained and tested two object detection network architectures, SSD-Mobilenet [6] and RFCN-Resnet101 [7]. We modified the architectures to directly operate on 640×480 input im-

Figure 3: Simulated scenes. The city backdrop and objects were either purchased (https://www.cgtrader.com) or downloaded from free sites. The original images were adjusted so that cars were roughly 2 m wide and pedestrians about 1.8 m tall. Objects were placed using simple scene generation principles (cars mainly in roads, people mainly on sidewalks). The cameras were placed randomly in the scene near a plane parallel and 1.5 m above the road. The camera 'look at' direction was randomly arranged within a cone near the plane. The realism of the simulated scenes will continue to improve as we develop better material models and develop procedural models for adding more objects.

ages. We initialized networks with weights optimized for object detection on MS COCO dataset [8, 9] and tuned them using 2200 synthetic images and as many as 1000 epochs. Performance plateaued after training for 100 epochs, and thus we used this number of training epochs in our experiments. The detection performance of three classes - cars, pedestrians and buses - was measured using mean average precision (mAP) at 50% intersection over union (IoU) between estimated and ground truth bounding boxes [10].

Rendering at scale

Quantitative computer graphics using ray tracing is compute-limited, not memory-limited. Moreover, physically based ray tracing is not well-suited to GPU-based rendering. Consequently, the preferred method for speeding up the ray-tracing calculation is to leverage multi-core architectures and cloud-scale job scheduling. Specifically, it takes about 10 minutes to render a single 640×480 image, using 1024 rays per pixel, on a machine equipped with two 8-core Intel Xeon processors; rendering a collection of images sufficiently large for machine learning algorithm training and evaluation is prohibitively slow with only one machine. Using cloud resources, in our case the Google Cloud Platform, we created many images in parallel to create a significant data set in a practical amount of time.

Experiments

We describe two experiments that illustrate how to use simulation to understand the impact of different camera architectures. The first experiment analyzes the impact of the cameras image processing pipeline on network performance. The second experiment analyzes network resilience to errors that arise from auto exposure algorithms.

In both experiments, we trained and tested using simulated images. These were generated by placing a virtual camera within the city scenes at random locations but near the ground. Specifically, the camera viewing direction was initialized within a plane 1.5 m above the ground, and then the 'look at' direction was modified by varying the pan over a range of -10 deg to 10 deg, and randomizing the tilt and roll over a range of -5 deg to 5 deg. The networks were altered so that the input size matched the size of the simulated images (640×480).

Image processing pipeline

Consumer photography applications use a sophisticated processing pipeline that is optimized and standardized to produce a good visual reproduction of the original scene when the images are rendered on a display. Many aspects of the image processing pipeline are designed to match the properties of the human visual system. Here we explored whether these operations benefit convolutional neural network performance, as well.

Figure 4 shows three stages in a conventional image processing pipeline that would be used for a sensor with a color filter

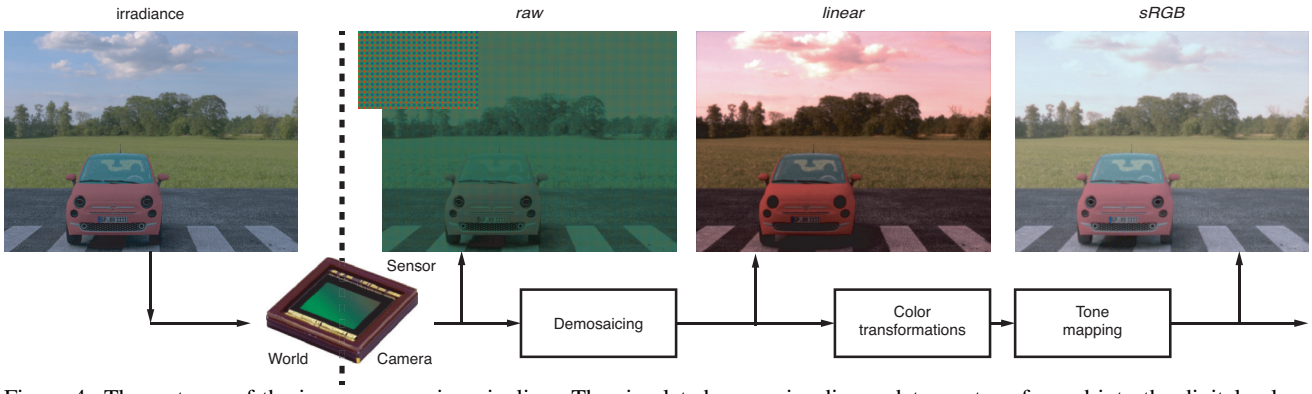


Figure 4: Three stages of the image processing pipeline. The simulated sensor irradiance data are transformed into the digital values sampled from the pixels (*raw*). These are then converted an RGB image using bilinear demosaicing (*linear*). A color transform and a nonlinear power function transform these values to a final representation (*sRGB*) .

array (CFA) arranged in a Bayer pattern (RG;GB). The pixel values read from the sensor is the *raw* input. Each pixel measures a pixel value from one of the three color filters. These data are typically demosaiced, in this case by linearly interpolating the image to represent RGB values at every position. The pixel values in this representation are a weighted sum of the number of photons incident at the pixel, hence we call this representation *linear*. The third stage shows the output of the image processing pipeline, a color-corrected image in sRGB space. This representation is calculated by applying (a) color transformations that account for the color filters and the illuminant, and (b) a nonlinear calculation that produces RGB values suitable for display on a typical monitor. In the pipeline used here, we chose a linear color transformation that mapped the pixel data of a Macbeth chart into a calibrated color space (CIE XYZ), and a gray-world illuminant correction algorithm. The data are gamma encoded to approximately match the industry-standard *sRGB* representation.

Figure 5 summarizes detection results for the SSD and RFCN networks. The height of each bar represents the mean average precision for testing and training on one of the three types of data. For the SSD network, object detection performance was lowest for the *raw* data and similar for the *linear* and *sRGB* data. We suspect this small difference arises because the spatial averaging for the demosaicking (*linear*, *sRGB*) have slightly higher pointwise signal-to-noise than the *raw* data. For the RFCN network the three cases are not meaningfully different, and this network has uniformly higher mean average precision than the SSD network.

We performed an additional experiment to assess whether training with the different inputs produce very different network parameters. Specifically, we assessed performance for a network trained with *raw* and *linear* on *sRGB* images. Training with *raw* data does not transfer well to the *sRGB* task, with a performance decline of about 20%. Training with *linear* data reduces performance with *sRGB* data by about 10%.

Neural network resilience to exposure errors

Auto-exposure (AE) is a fundamental camera algorithm whose purpose is to estimate camera settings (aperture, pixel conversion gain, and exposure duration) so that the acquired pixel responses span the dynamic range. In practice, exposure algorithms often acquire several frames to stabilize, particularly for a dynamic scene such as a vehicle leaving a tunnel and emerging into daylight. Under these conditions frames with sub-optimal exposures will be acquired and processed; we tested the resilience of a detection network with respect to bias errors in the exposure settings.

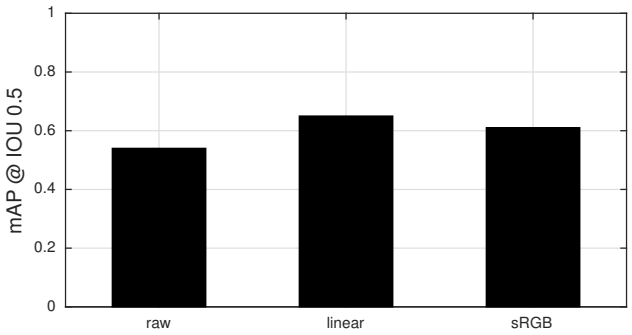
To simplify the auto exposure robustness, we fixed the the mean sensor illuminance (1000 lx), pixel gain (1.0) and aperture and swept out the exposure duration. We considered an exposure setting with a mean pixel response of 0.4 of the voltage swing to be ideal, and we simulated we exposure errors that deviated from ideal by a factor of 2^i where $i \in [-4,4]$. The exponent is sometimes called the bias of the exposure value (EV), and EV $= 0$ is a correctly exposed frame (Figure 6).

To establish an upper bound on performance, we first trained and tested each network to each of the separate EV bias levels, determining network parameters optimized for each exposure bias. We used these measurements to establish an upper bound on performance. This bound defines an 'unreachable' performance level, the shaded region in each of the panels in Figure 7; the accuracy estimates should fall below this shaded region. To explore robustness, we first trained the baseline network using only unbiased images (EV $= 0$, solid curve), and we then tested performance using images with varying amounts of EV bias (EV mix, dashed curve), a common data augmentation strategy. In the EV mix case each bias was represented by equal number of examples, and the total number of training images was the same as the EV $= 0$ and optimal conditions.

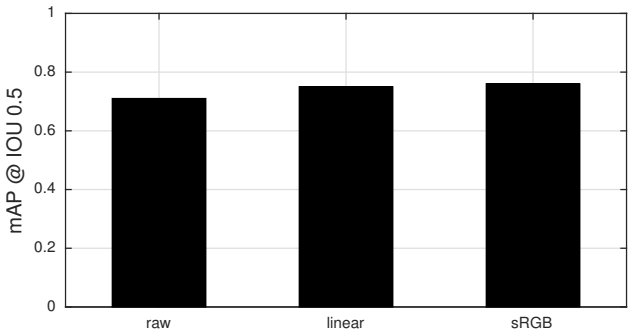
The analysis reveals that the RFCN network is more resilient than the SSD network to exposure value bias. Training with a mixture of exposure values (EV mix) helps network resilience at the cost of lowering the accuracy, for an equal number of training images. Finally, the asymmetry of the curves around EV $= 0$ value shows that both networks handle underexposed images better than overexposed ones.

Discussion

There is substantial interest in using synthetic images for for autonomous vehicle applications where data in dangerous driving conditions are difficult to obtain. This article provides an additional reason for using synthetic images: to explore the range of image acquisition architectures that might improve performance and reduce system cost. In the following, we describe how our



(a) SSD-Mobilenet



(b) RFCN-Resnet101

Figure 5: Network performance when trained and tested using raw, linear and sRGB data. Accuracy is measured as a fraction, the intersection of union (IoU) between the true bounding box and the estimated bounding box. The mean average precision (mAP) across many test images and classes is reported. The SSD network (a) performs best when trained with *linear* data; accuracy decreases by 15 percentage points when trained with *raw* data. The RFCN network (b) has higher accuracy than SSD for all of the data types, and the performance when trained using *raw* data is only slightly lower than training with the other types.

work relates to other projects developing image simulation tools.

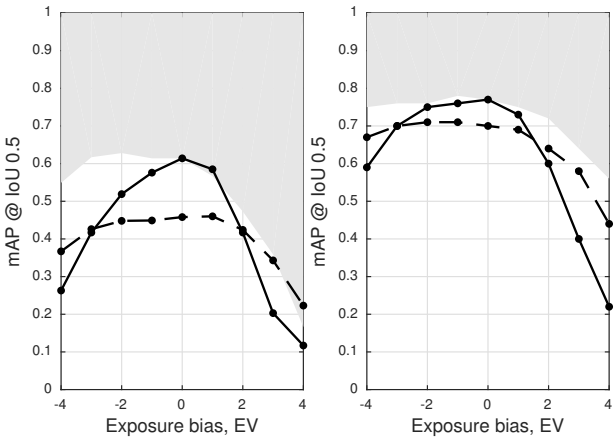
Related work

Tsirikoglou et al. describe procedural methods for creating a large variety of plausible driving scenarios [11]. They advocate using ray tracing methods to generate images rather than rasterization methods from game engines. In support of this view, they show that training networks on synthetic ray-traced images perform better on camera images compared to networks trained on rasterized images. Specifically, they report that training a network on the SYNTHIA [12] data set yields an accuracy of 21%, the rasterized images from Grand Theft Auto yields an accuracy of 31%, and their ray-traced images yields an accuracy of 37%. In the case of analyzing indoor scenes, Zhang et al. report similar findings - pre-training with a large synthetic ray-traced data set improves network performance tested on camera data [13].

We support the principle of using ray-tracing to obtain the most realistic synthetic images. There is a significant computational cost, but the quality of the ray-traced images is very important for the process of camera design.

Buckler et al. analyze how the camera image processing pipeline influences computer vision algorithm performance [14].

Figure 6: A simulated scene captured with the correct exposure value (top), under-exposed (left), or over-exposed (right). Such images, with different amounts of exposure value bias, were used for training and testing to explore network resilience to imperfections in the auto-exposure algorithm.



(a) SSD-Mobilenet (b) RFCN-Resnet101

Figure 7: Network resilience to exposure value bias. The boundary of the shaded region marks the upper bound on accuracy; it is estimated by training and testing the network using sRGB images at each EV bias. The two curves show accuracy when trained at EV = 0 and tested at multiple EV values (solid) or trained at multiple EV values (dashed). The two panels are for the SSD (a) and RFCN (b) networks. The RFCN network has higher performance and better resilience, at a higher computational cost.

They set out to simplify processing to reduce power consumption, while maintaining accuracy across a range of computer vision algorithms. They do not use an end-to-end camera simulation to investigate this issue; instead they approximate sensor data from existing RGB images [15]. They report that demosaicing, the major differences between *raw* and *linear*, and tone mapping, the major difference between *linear* and *sRGB*, both influence algorithm performance.

The SSD network performance differs between raw and demosaiced, as described by Buckler et al., but the difference is much smaller for the RFCN network. Tone-mapping had almost no effect on performance for both networks. These experiments suggest that camera architecture should be evaluated for specific networks. The end-to-end simulation enables such a co-design methodology (e.g., Diamond et al. [16])

Conclusions

The contributions of this paper are:

1. A collection of open-source, free software tools that implement end-to-end simulation from a spectral scene, ray-traced through the optics to the sensor irradiance, sampled by a sensor model, transformed by an image processing pipeline and delivered to a CNN.
2. An assessment of the impact of the image processing pipeline for consumer photography on the performance of two CNNs.
3. An assessment of network resilience as measured by how accuracy varies with imperfect auto-exposure settings.

Some networks can be effectively trained to use raw sensor data, bypassing the image processing pipeline, and other networks can be effectively trained with linear data without color transforms or tone-mapping. We evaluated parameter generalization by training on *raw* or *linear* sensor data and testing on sRGB images; in both cases we found a significant (25%) reduction in accuracy (poor generalization). Hence, achieving a high level of performance requires training with the specific data type.

The limits on network generalization between image data types is supported by Tsirikoglou, et al. [11] who trained networks trained using SYNTHIA, Grand Theft Auto and Ray-Tracing and assessed performance on Cityscapes images. The generalization between data types is low, with accuracy dropping by even more than 50%. The poor generalization is also supported by Buckler, et al. [14]. These results and ours suggest that training a network with images from one type of camera may produce solutions that do not generalize to other cameras. We conclude that co-design of the camera and network is a preferred approach.

Author Biographies

Henryk Blasinski received a MS degree in Telecommunications and Computer Science from the Lodz University of Technology, Lodz, Poland (2008), and the Diplome d'Ingeneiur degree from the Institut Superieur d'Electronique de Paris, France (2009). He graduated with a PhD in Electrical Engineering from Stanford University (2018). Henryk's research interests lie at the intersection of imaging, computer vision and machine learning.

Joyce Farrell is the Executive Director of the Stanford Center for Image Systems Engineering and a Senior Research Associate in the Department of Electrical Engineering at Stanford University. She received a doctorate from Stanford in 1981 and has worked at several companies and research institutions, including the NASA Ames Research Center, New York University, the Xerox Palo Alto Research Center and Shutterfly.

Trisha Lian received her BS in Biomedical Engineering from Duke University (2014) and is currently a PhD student in Electrical Engineering at Stanford University. Her work has focused on the development of simulation tools for novel camera systems, as well as simulation of the human visual system.

Zhenyi Liu received his MS in Electrical Engineering at Uslan National Institute of Science and Technology, UNIST in Korea and is currently a PhD candidate in Automotive Engineering at Jilin University, China. His research interests focus on different machine perception systems for autonomous vehicles such as cameras and lidar.

Brian A. Wandell is the first Isaac and Madeline Stein Family Professor. He joined the Stanford Psychology faculty in 1979 and is a member, by courtesy, of Electrical Engineering, Ophthalmology, and the Graduate School of Education. He is Director of Stanfords Center for Cognitive and Neurobiological Imaging and Deputy Director of Stanfords Neuroscience Institute. Wandells research centers on vision science, spanning topics from visual disorders, reading development in children, to digital imaging devices and algorithms for both magnetic resonance imaging and digital imaging.

References

[1] J. Farrell, P. Catrysse, and B. Wandell, "Digital camera simulation," *Applied optics*, vol. 51, no. 4, pp. A80–A90, 2012.

[2] M. Pharr, W. Jakob, and G. Humphreys, *Physically based rendering: From theory to implementation*. Morgan Kaufmann, 2016.

[3] A. Lin, "The computational image systems evaluation toolbox (CISET)," Ph.D. dissertation, Stanford University, 2015.

[4] E. Freniere, G. Gregory, and R. Hassler, "Edge diffraction in Monte Carlo ray tracing," in *Optical Design and Analysis Software*, vol. 3780. International Society for Optics and Photonics, 1999, pp. 151–158.

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

[6] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Fu, and A. Berg, "Ssd: Single shot multibox detector," in *Proceedings of the European Conference on Computer Vision, ECCV*. Springer, 2016, pp. 21–37.

[7] J. Dai, Y. Li, K. He, and J. Sun, "R-FCN: Object detection via region-based fully convolutional networks," in *Proceedings of Advances in Neural Information Processing Systems, NIPS*, 2016, pp. 379–387.

[8] T. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. Zitnick, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014. [Online]. Available: http://arxiv.org/abs/1405.0312

[9] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy, "Speed/accuracy trade-offs for modern convolutional object detectors," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, Jul. 2017, pp. 3296–3297.

[10] M. Everingham, L. Van Gool, C. 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, 2010.

[11] A. Tsirikoglolu, J. Kronander, M. Wrenninge, and J. Unger, "Procedural modeling and physically based rendering for synthetic data generation in automotive applications," *arXiv preprint arXiv:1710.06270*, 2017.

[12] G. Ros, L. Sellart, J. Materzynska, D. Vazquez, and A. Lopez, "The SYNTHIA dataset: A large collection of synthetic images for semantic segmentation of urban scenes," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2016, pp. 3234–3243.

[13] Y. Zhang, S. Song, E. Yumer, M. Savva, J. Lee, H. Jin, and T. Funkhouser, "Physically-based rendering for indoor scene understanding using convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR*. IEEE, 2017, pp. 5057–5065.

[14] M. Buckler, S. Jayasuriya, and A. Sampson, "Reconfiguring the imaging pipeline for computer vision," in *Proceedings of the IEEE International Conference on Computer Vision, ICCV*, Oct. 2017, pp. 975–984.

[15] S. Kim, H. Lin, Z. Lu, S. Süsstrunk, S. Lin, and M. Brown, "A new in-camera imaging model for color computer vision and its application," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 12, pp. 2289–2302, 2012.

[16] S. Diamond, V. Sitzmann, S. Boyd, G. Wetzstein, and F. Heide, "Dirty pixels: Optimizing image classification architectures for raw sensor data," *arXiv preprint arXiv:1701.06487*, 2017.